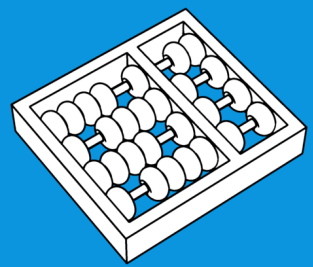
NP-complexidade

MC558 - Projeto e Análise de Algoritmos II

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

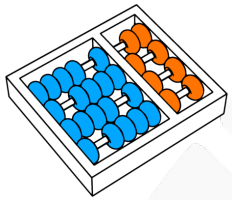
UNICAMP



"If $P = NP$, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in "creative leaps,"no fundamental gap between solving a problem and recognizing the solution once it's found. Everyone who could appreciate a symphony would be Mozart; everyone who could follow a step-by-step argument would be Gauss; everyone who could recognize a good investment strategy would be Warren Buffett."

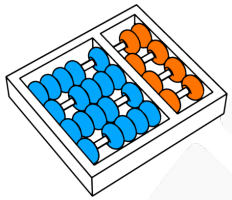
Scott Aaronson.

Introdução



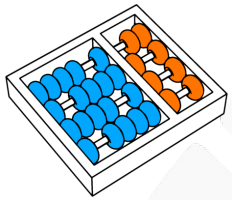
Algoritmos eficientes e ineficientes

- Para vários problemas, vimos algoritmos rápidos:



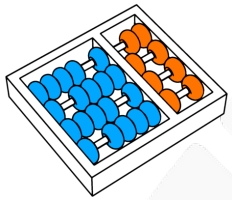
Algoritmos eficientes e ineficientes

- Para vários problemas, vimos algoritmos rápidos:
  - Ordenação: $O(n \log n)$.



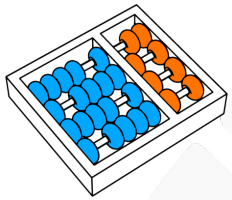
Algoritmos eficientes e ineficientes

- Para vários problemas, vimos algoritmos rápidos:
  - Ordenação: $O(n \log n)$.
  - Multiplicação de matrizes: $O(n^{2,72})$.



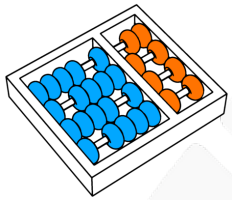
Algoritmos eficientes e ineficientes

- Para vários problemas, vimos algoritmos rápidos:
  - Ordenação: $O(n \log n)$.
  - Multiplicação de matrizes: $O(n^{2,72})$.
  - Caminho mais curto*: $O(mE)$.



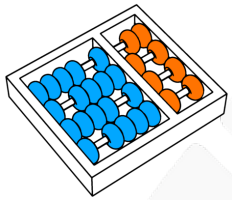
Algoritmos eficientes e ineficientes

- Para vários problemas, vimos algoritmos rápidos:
  - Ordenação: $O(n \log n)$.
  - Multiplicação de matrizes: $O(n^{2,72})$.
  - Caminho mais curto*: $O(mE)$.
  - Circuito euleriano: $O(V + E)$.



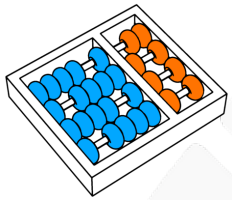
Algoritmos eficientes e ineficientes

- Para vários problemas, vimos algoritmos rápidos:
  - Ordenação: $O(n \log n)$.
  - Multiplicação de matrizes: $O(n^{2,72})$.
  - Caminho mais curto*: $O(mE)$.
  - Circuito euleriano: $O(V + E)$.
- Para outros, só são conhecidos algoritmos lentos:



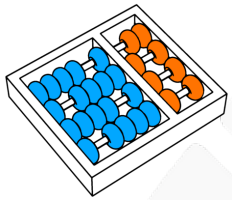
Algoritmos eficientes e ineficientes

- Para vários problemas, vimos algoritmos rápidos:
  - Ordenação: $O(n \log n)$.
  - Multiplicação de matrizes: $O(n^{2,72})$.
  - Caminho mais curto*: $O(mE)$.
  - Circuito euleriano: $O(V + E)$.
- Para outros, só são conhecidos algoritmos lentos:
  - Problema da mochila: $O(2^n)$.



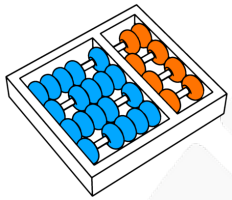
Algoritmos eficientes e ineficientes

- Para vários problemas, vimos algoritmos rápidos:
  - Ordenação: $O(n \log n)$.
  - Multiplicação de matrizes: $O(n^{2,72})$.
  - Caminho mais curto*: $O(mE)$.
  - Circuito euleriano: $O(V + E)$.
- Para outros, só são conhecidos algoritmos lentos:
  - Problema da mochila: $O(2^n)$.
  - Caminho mais longo*: $O(m!2^m E)$.



Algoritmos eficientes e ineficientes

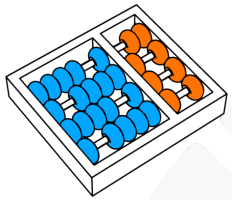
- Para vários problemas, vimos algoritmos rápidos:
  - Ordenação: $O(n \log n)$.
  - Multiplicação de matrizes: $O(n^{2,72})$.
  - Caminho mais curto*: $O(mE)$.
  - Circuito euleriano: $O(V + E)$.
- Para outros, só são conhecidos algoritmos lentos:
  - Problema da mochila: $O(2^n)$.
  - Caminho mais longo*: $O(m!2^m E)$.
  - Circuito hamiltoniano: $O(2^V)$.



Algoritmos eficientes e ineficientes

- Para vários problemas, vimos algoritmos rápidos:
  - Ordenação: $O(n \log n)$.
  - Multiplicação de matrizes: $O(n^{2,72})$.
  - Caminho mais curto*: $O(mE)$.
  - Circuito euleriano: $O(V + E)$.
- Para outros, só são conhecidos algoritmos lentos:
  - Problema da mochila: $O(2^n)$.
  - Caminho mais longo*: $O(m!2^m E)$.
  - Circuito hamiltoniano: $O(2^V)$.

Como identificar algoritmos **RÁPIDOS**?

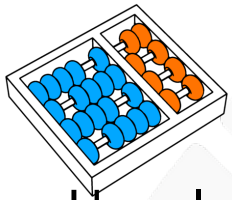


Algoritmos eficientes e ineficientes

- Para vários problemas, vimos algoritmos rápidos:
  - Ordenação: $O(n \log n)$.
  - Multiplicação de matrizes: $O(n^{2,72})$.
  - Caminho mais curto*: $O(mE)$.
  - Circuito euleriano: $O(V + E)$.
- Para outros, só são conhecidos algoritmos lentos:
  - Problema da mochila: $O(2^n)$.
  - Caminho mais longo*: $O(m!2^m E)$.
  - Circuito hamiltoniano: $O(2^V)$.

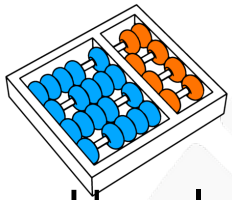
Como identificar algoritmos **RÁPIDOS**?

*m é o tamanho do caminho



Algoritmos de tempo polinomial

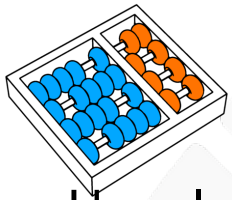
Um algoritmo é **POLINOMIAL** se o tempo de execução for limitado por $O(n^k)$ para alguma constante k.



Algoritmos de tempo polinomial

Um algoritmo é **POLINOMIAL** se o tempo de execução for limitado por $O(n^k)$ para alguma constante k.

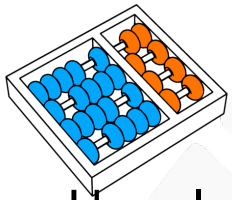
- Nesse caso, dizemos que ele é um algoritmo **EFICIENTE**.



Algoritmos de tempo polinomial

Um algoritmo é **POLINOMIAL** se o tempo de execução for limitado por $O(n^k)$ para alguma constante k.

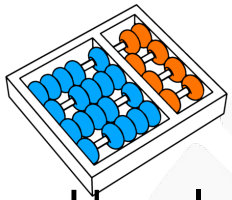
- Nesse caso, dizemos que ele é um algoritmo **EFICIENTE**.
- Não necessariamente é rápido na prática.



Algoritmos de tempo polinomial

Um algoritmo é **POLINOMIAL** se o tempo de execução for limitado por $O(n^k)$ para alguma constante k.

- Nesse caso, dizemos que ele é um algoritmo **EFICIENTE**.
- Não necessariamente é rápido na prática.
- Mas exclui muitos dos algoritmos considerados lentos.



Algoritmos de tempo polinomial

Um algoritmo é **POLINOMIAL** se o tempo de execução for limitado por $O(n^k)$ para alguma constante k.

- Nesse caso, dizemos que ele é um algoritmo **EFICIENTE**.
- Não necessariamente é rápido na prática.
- Mas exclui muitos dos algoritmos considerados lentos.

indebted to many people, at the Symposium and at the National Bureau of Standards, who have taken an interest in the matching problem. There has been much animated discussion on possible versions of an algorithm.

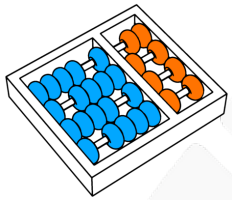
2. Digression. An explanation is due on the use of the words "efficient algorithm." First, what I present is a conceptual description of an algorithm and not a particular formalized algorithm or "code."

For practical purposes computational details are vital. However, my is an efficient quate in opera- the sense that

PATHS, TREES, AND FLOWERS

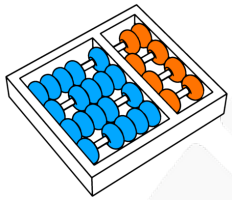
JACK EDMONDS

1. Introduction. A *graph* G for purposes here is a finite set of elem called *vertices* and a finite set of elements called *edges* such that each e *meets* exactly two vertices, called the *end-points* of the edge. An edge is s to *join* its end-points.



Organizando os problemas

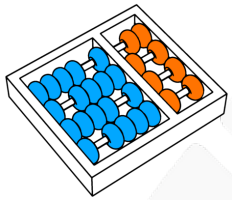
Por que se preocupar com isso?



Organizando os problemas

Por que se preocupar com isso?

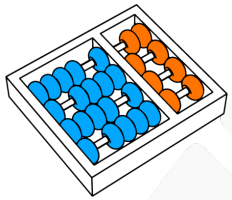
- Desconhecemos algoritmos rápidos para vários problemas.



Organizando os problemas

Por que se preocupar com isso?

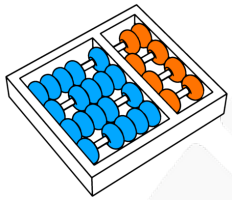
- Desconhecemos algoritmos rápidos para vários problemas.
- Acreditamos que não há algoritmos eficientes para eles.



Organizando os problemas

Por que se preocupar com isso?

- Desconhecemos algoritmos rápidos para vários problemas.
- Acreditamos que não há algoritmos eficientes para eles.
- Queremos saber quais deles têm **ALGORITMOS POLINOMIAIS**!

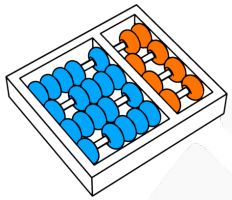


Organizando os problemas

Por que se preocupar com isso?

- Desconhecemos algoritmos rápidos para vários problemas.
- Acreditamos que não há algoritmos eficientes para eles.
- Queremos saber quais deles têm **ALGORITMOS POLINOMIAIS**!

Antes vamos discutir:



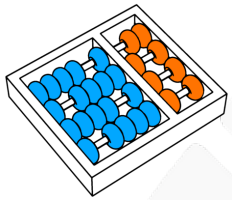
Organizando os problemas

Por que se preocupar com isso?

- Desconhecemos algoritmos rápidos para vários problemas.
- Acreditamos que não há algoritmos eficientes para eles.
- Queremos saber quais deles têm **ALGORITMOS POLINOMIAIS**!

Antes vamos discutir:

1. Como representar problemas?



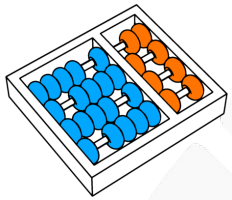
Organizando os problemas

Por que se preocupar com isso?

- Desconhecemos algoritmos rápidos para vários problemas.
- Acreditamos que não há algoritmos eficientes para eles.
- Queremos saber quais deles têm **ALGORITMOS POLINOMIAIS**!

Antes vamos discutir:

1. Como representar problemas?
2. Como comparar problemas?



Organizando os problemas

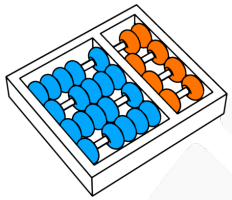
Por que se preocupar com isso?

- Desconhecemos algoritmos rápidos para vários problemas.
- Acreditamos que não há algoritmos eficientes para eles.
- Queremos saber quais deles têm **ALGORITMOS POLINOMIAIS**!

Antes vamos discutir:

1. Como representar problemas?
2. Como comparar problemas?

Respondemos essas perguntas da seguinte forma:



Organizando os problemas

Por que se preocupar com isso?

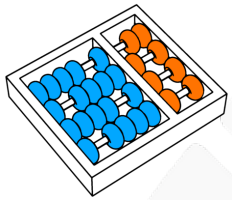
- Desconhecemos algoritmos rápidos para vários problemas.
- Acreditamos que não há algoritmos eficientes para eles.
- Queremos saber quais deles têm **ALGORITMOS POLINOMIAIS**!

Antes vamos discutir:

1. Como representar problemas?
2. Como comparar problemas?

Respondemos essas perguntas da seguinte forma:

1. Representamos problemas como **LINGUAGENS FORMAIS**.



Organizando os problemas

Por que se preocupar com isso?

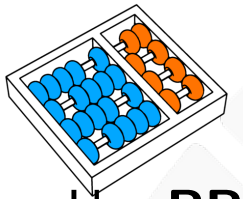
- Desconhecemos algoritmos rápidos para vários problemas.
- Acreditamos que não há algoritmos eficientes para eles.
- Queremos saber quais deles têm **ALGORITMOS POLINOMIAIS**!

Antes vamos discutir:

1. Como representar problemas?
2. Como comparar problemas?

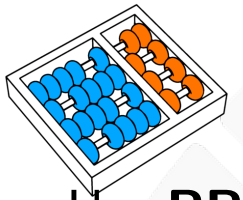
Respondemos essas perguntas da seguinte forma:

1. Representamos problemas como **LINGUAGENS FORMAIS**.
2. Comparamos problemas com um tipo de **REDUÇÃO**.



Problemas de decisão

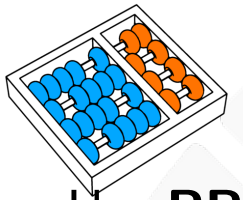
Um **PROBLEMA DE DECISÃO** é um problema em que a resposta de cada elemento da entrada é SIM ou NAO.



Problemas de decisão

Um **PROBLEMA DE DECISÃO** é um problema em que a resposta de cada elemento da entrada é SIM ou NAO.

São problemas de decisão:

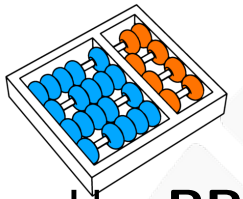


Problemas de decisão

Um **PROBLEMA DE DECISÃO** é um problema em que a resposta de cada elemento da entrada é SIM ou NAO.

São problemas de decisão:

- Dado um número m, m é primo?

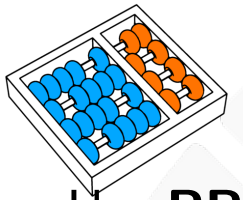


Problemas de decisão

Um **PROBLEMA DE DECISÃO** é um problema em que a resposta de cada elemento da entrada é SIM ou NAO.

São problemas de decisão:

- Dado um número m, m é primo?
- Dadas as posições das peças em um tabuleiro de xadrez, o rei está em xeque?



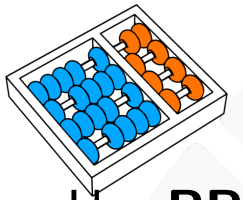
Problemas de decisão

Um **PROBLEMA DE DECISÃO** é um problema em que a resposta de cada elemento da entrada é SIM ou NAO.

São problemas de decisão:

- Dado um número m, m é primo?
- Dadas as posições das peças em um tabuleiro de xadrez, o rei está em xeque?

Não são problemas de decisão:



Problemas de decisão

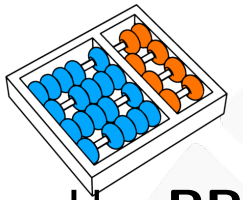
Um **PROBLEMA DE DECISÃO** é um problema em que a resposta de cada elemento da entrada é SIM ou NAO.

São problemas de decisão:

- Dado um número m, m é primo?
- Dadas as posições das peças em um tabuleiro de xadrez, o rei está em xeque?

Não são problemas de decisão:

- Soma, ordenação, caminho mínimo etc.



Problemas de decisão

Um **PROBLEMA DE DECISÃO** é um problema em que a resposta de cada elemento da entrada é SIM ou NAO.

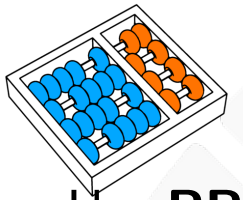
São problemas de decisão:

- Dado um número m, m é primo?
- Dadas as posições das peças em um tabuleiro de xadrez, o rei está em xeque?

Não são problemas de decisão:

- Soma, ordenação, caminho mínimo etc.

Por que estudar problemas de decisão?



Problemas de decisão

Um **PROBLEMA DE DECISÃO** é um problema em que a resposta de cada elemento da entrada é SIM ou NAO.

São problemas de decisão:

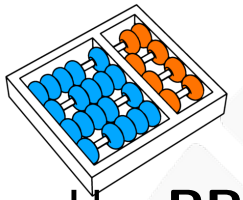
- Dado um número m, m é primo?
- Dadas as posições das peças em um tabuleiro de xadrez, o rei está em xeque?

Não são problemas de decisão:

- Soma, ordenação, caminho mínimo etc.

Por que estudar problemas de decisão?

- É mais simples estudá-los do que problemas em geral.



Problemas de decisão

Um **PROBLEMA DE DECISÃO** é um problema em que a resposta de cada elemento da entrada é SIM ou NAO.

São problemas de decisão:

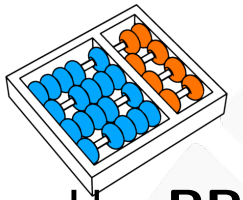
- Dado um número m, m é primo?
- Dadas as posições das peças em um tabuleiro de xadrez, o rei está em xeque?

Não são problemas de decisão:

- Soma, ordenação, caminho mínimo etc.

Por que estudar problemas de decisão?

- É mais simples estudá-los do que problemas em geral.
- Várias situações são postas como problemas de decisão.



Problemas de decisão

Um **PROBLEMA DE DECISÃO** é um problema em que a resposta de cada elemento da entrada é SIM ou NAO.

São problemas de decisão:

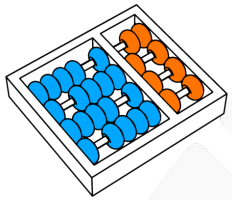
- Dado um número m, m é primo?
- Dadas as posições das peças em um tabuleiro de xadrez, o rei está em xeque?

Não são problemas de decisão:

- Soma, ordenação, caminho mínimo etc.

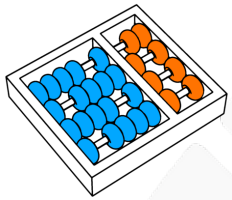
Por que estudar problemas de decisão?

- É mais simples estudá-los do que problemas em geral.
- Várias situações são postas como problemas de decisão.
- Às vezes, decidir se existe alguma solução para um problema em geral é tão difícil quanto encontrá-la.



Versão de decisão de problema de busca

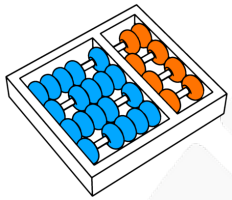
Problema (Busca do ciclo hamiltoniano)



Versão de decisão de problema de busca

Problema (Busca do ciclo hamiltoniano)

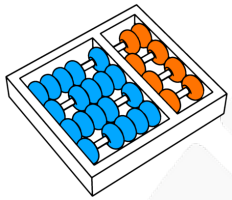
- ***Entrada:*** *Um grafo G.*



Versão de decisão de problema de busca

Problema (Busca do ciclo hamiltoniano)

- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Um ciclo em G que percorre todos vértices.*

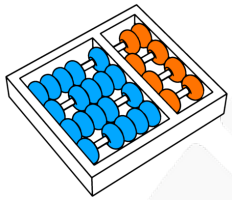


Versão de decisão de problema de busca

Problema (Busca do ciclo hamiltoniano)

- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Um ciclo em G que percorre todos vértices.*

Problema (Decisão do ciclo hamiltoniano)



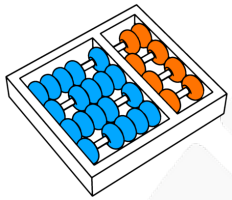
Versão de decisão de problema de busca

Problema (Busca do ciclo hamiltoniano)

- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Um ciclo em G que percorre todos vértices.*

Problema (Decisão do ciclo hamiltoniano)

- ***Entrada:*** *Um grafo G.*



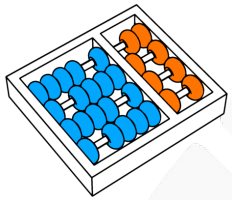
Versão de decisão de problema de busca

Problema (Busca do ciclo hamiltoniano)

- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Um ciclo em G que percorre todos vértices.*

Problema (Decisão do ciclo hamiltoniano)

- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Decidir se existe ciclo em G que percorre todos vértices.*



Versão de decisão de problema de busca

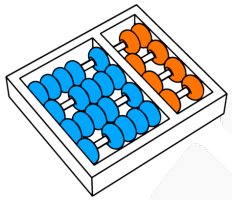
Problema (Busca do ciclo hamiltoniano)

- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Um ciclo em G que percorre todos vértices.*

Problema (Decisão do ciclo hamiltoniano)

- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Decidir se existe ciclo em G que percorre todos vértices.*

Suponha que sabemos **DECIDIR** em tempo polinomial



Versão de decisão de problema de busca

Problema (Busca do ciclo hamiltoniano)

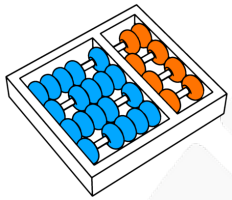
- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Um ciclo em G que percorre todos vértices.*

Problema (Decisão do ciclo hamiltoniano)

- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Decidir se existe ciclo em G que percorre todos vértices.*

Suponha que sabemos **DECIDIR** em tempo polinomial

- Como **ENCONTRAR** em tempo polinomial?



Versão de decisão de problema de busca

Problema (Busca do ciclo hamiltoniano)

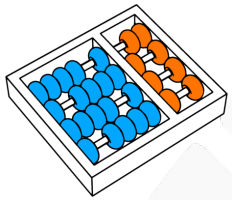
- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Um ciclo em G que percorre todos vértices.*

Problema (Decisão do ciclo hamiltoniano)

- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Decidir se existe ciclo em G que percorre todos vértices.*

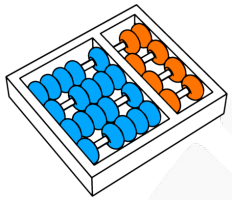
Suponha que sabemos **DECIDIR** em tempo polinomial

- Como **ENCONTRAR** em tempo polinomial?
- Descubra uma aresta por vez (exercício).



Versão de decisão de problema de otimização

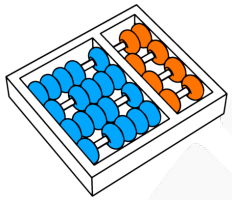
Problema (Caixeiro Viajante)



Versão de decisão de problema de otimização

Problema (Caixeiro Viajante)

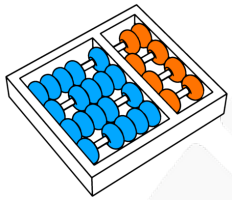
- ***Entrada:*** *Um grafo ponderado G.*



Versão de decisão de problema de otimização

Problema (Caixeiro Viajante)

- ***Entrada:*** *Um grafo ponderado G.*
- ***Saída:*** *Um ciclo hamiltoniano com peso mínimo.*

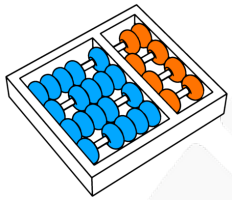


Versão de decisão de problema de otimização

Problema (Caixeiro Viajante)

- ***Entrada:*** *Um grafo ponderado G.*
- ***Saída:*** *Um ciclo hamiltoniano com peso mínimo.*

Problema (Versão de decisão)



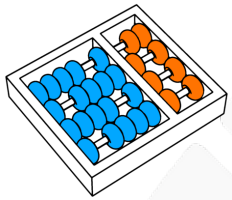
Versão de decisão de problema de otimização

Problema (Caixeiro Viajante)

- ***Entrada:*** *Um grafo ponderado G.*
- ***Saída:*** *Um ciclo hamiltoniano com peso mínimo.*

Problema (Versão de decisão)

- ***Entrada:*** *Um grafo ponderado G, um parâmetro k.*



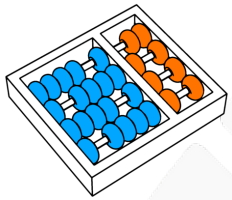
Versão de decisão de problema de otimização

Problema (Caixeiro Viajante)

- ***Entrada:*** *Um grafo ponderado G.*
- ***Saída:*** *Um ciclo hamiltoniano com peso mínimo.*

Problema (Versão de decisão)

- ***Entrada:*** *Um grafo ponderado G, um parâmetro k.*
- ***Saída:*** *Decidir se existe um ciclo hamiltoniano em G com peso no máximo k.*



Versão de decisão de problema de otimização

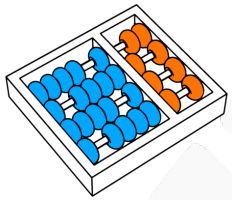
Problema (Caixeiro Viajante)

- ***Entrada:*** *Um grafo ponderado G.*
- ***Saída:*** *Um ciclo hamiltoniano com peso mínimo.*

Problema (Versão de decisão)

- ***Entrada:*** *Um grafo ponderado G, um parâmetro k.*
- ***Saída:*** *Decidir se existe um ciclo hamiltoniano em G com peso no máximo k.*

Suponha que sabemos **DECIDIR** em tempo polinomial



Versão de decisão de problema de otimização

Problema (Caixeiro Viajante)

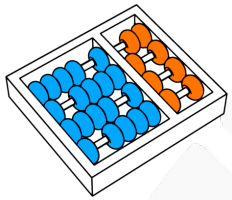
- ***Entrada:*** *Um grafo ponderado G.*
- ***Saída:*** *Um ciclo hamiltoniano com peso mínimo.*

Problema (Versão de decisão)

- ***Entrada:*** *Um grafo ponderado G, um parâmetro k.*
- ***Saída:*** *Decidir se existe um ciclo hamiltoniano em G com peso no máximo k.*

Suponha que sabemos **DECIDIR** em tempo polinomial

- Como determinar o **VALOR ÓTIMO** em tempo polinomial?



Versão de decisão de problema de otimização

Problema (Caixeiro Viajante)

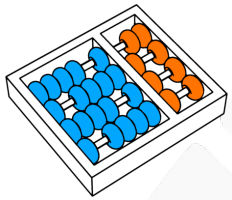
- ***Entrada:*** *Um grafo ponderado G.*
- ***Saída:*** *Um ciclo hamiltoniano com peso mínimo.*

Problema (Versão de decisão)

- ***Entrada:*** *Um grafo ponderado G, um parâmetro k.*
- ***Saída:*** *Decidir se existe um ciclo hamiltoniano em G com peso no máximo k.*

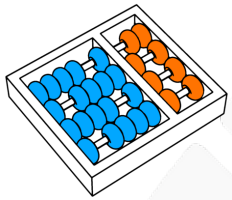
Suponha que sabemos **DECIDIR** em tempo polinomial

- Como determinar o **VALOR ÓTIMO** em tempo polinomial?
- Faça uma busca binária (exercício).



Olhando para frente

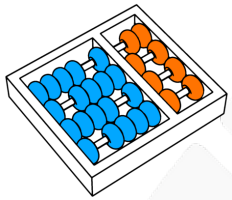
Vamos classificar **PROBLEMAS DE DECISÃO** nas classes:



Olhando para frente

Vamos classificar **PROBLEMAS DE DECISÃO** nas classes:

P: podem ser resolvidos em tempo polinomial

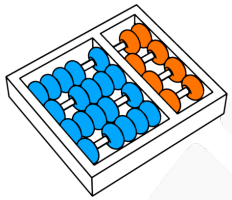


Olhando para frente

Vamos classificar **PROBLEMAS DE DECISÃO** nas classes:

P: podem ser resolvidos em tempo polinomial

NP: têm soluções curtas que podem ser verificadas em tempo polinomial



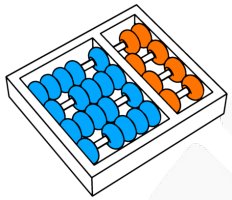
Olhando para frente

Vamos classificar **PROBLEMAS DE DECISÃO** nas classes:

P: podem ser resolvidos em tempo polinomial

NP: têm soluções curtas que podem ser verificadas em tempo polinomial

NP-difícil: pelo menos tão difíceis quanto quaisquer outros problemas em NP



Olhando para frente

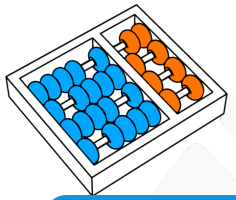
Vamos classificar **PROBLEMAS DE DECISÃO** nas classes:

P: podem ser resolvidos em tempo polinomial

NP: têm soluções curtas que podem ser verificadas em tempo polinomial

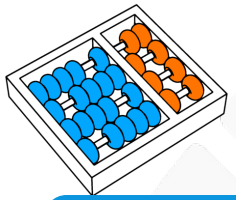
NP-difícil: pelo menos tão difíceis quanto quaisquer outros problemas em NP

NP-completo: são NP e NP-difícil



Exemplos de problemas em P

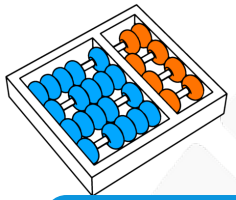
Problema (Soma de elemento)



Exemplos de problemas em P

Problema (Soma de elemento)

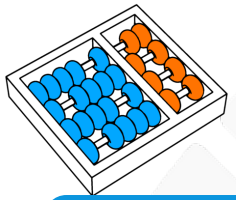
- ***Entrada:*** *Um conjunto de números S.*



Exemplos de problemas em P

Problema (Soma de elemento)

- ***Entrada:*** *Um conjunto de números S.*
- ***Saída:*** *Decidir se existe $n \in S$ tal que $n = \sum_{m \in S \setminus \{n\}} m$.*

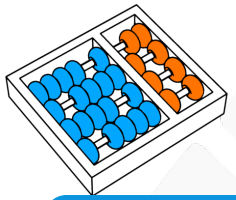


Exemplos de problemas em P

Problema (Soma de elemento)

- ***Entrada:*** *Um conjunto de números S.*
- ***Saída:*** *Decidir se existe $n \in S$ tal que $n = \sum_{m \in S \setminus \{n\}} m$.*

Problema (Conexidade)



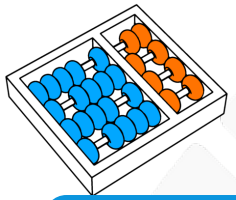
Exemplos de problemas em P

Problema (Soma de elemento)

- ***Entrada:*** *Um conjunto de números S.*
- ***Saída:*** *Decidir se existe $n \in S$ tal que $n = \sum_{m \in S \setminus \{n\}} m$.*

Problema (Conexidade)

- ***Entrada:*** *Um grafo G.*



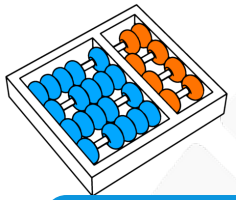
Exemplos de problemas em P

Problema (Soma de elemento)

- ***Entrada:*** *Um conjunto de números S.*
- ***Saída:*** *Decidir se existe $n \in S$ tal que $n = \sum_{m \in S \setminus \{n\}} m$.*

Problema (Conexidade)

- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Decidir se G é conexo.*



Exemplos de problemas em P

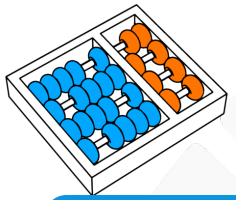
Problema (Soma de elemento)

- ***Entrada:*** *Um conjunto de números S.*
- ***Saída:*** *Decidir se existe $n \in S$ tal que $n = \sum_{m \in S \setminus \{n\}} m$.*

Problema (Conexidade)

- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Decidir se G é conexo.*

Problema (Caminho mínimo)



Exemplos de problemas em P

Problema (Soma de elemento)

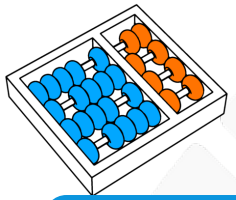
- ***Entrada:*** *Um conjunto de números S.*
- ***Saída:*** *Decidir se existe $n \in S$ tal que $n = \sum_{m \in S \setminus \{n\}} m$.*

Problema (Conexidade)

- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Decidir se G é conexo.*

Problema (Caminho mínimo)

- ***Entrada:*** *Um grafo ponderado G, dois vértices s e t e um inteiro k.*



Exemplos de problemas em P

Problema (Soma de elemento)

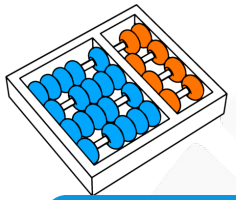
- ***Entrada:*** *Um conjunto de números S.*
- ***Saída:*** *Decidir se existe $n \in S$ tal que $n = \sum_{m \in S \setminus \{n\}} m$.*

Problema (Conexidade)

- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Decidir se G é conexo.*

Problema (Caminho mínimo)

- ***Entrada:*** *Um grafo ponderado G, dois vértices s e t e um inteiro k.*
- ***Saída:*** *Decidir se existe um caminho de s até t de peso no máximo k.*



Exemplos de problemas em P

Problema (Soma de elemento)

- ***Entrada:*** *Um conjunto de números S.*
- ***Saída:*** *Decidir se existe $n \in S$ tal que $n = \sum_{m \in S \setminus \{n\}} m$.*

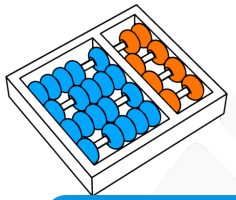
Problema (Conexidade)

- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Decidir se G é conexo.*

Problema (Caminho mínimo)

- ***Entrada:*** *Um grafo ponderado G, dois vértices s e t e um inteiro k.*
- ***Saída:*** *Decidir se existe um caminho de s até t de peso no máximo k.*

Problema (4-Coloração)



Exemplos de problemas em P

Problema (Soma de elemento)

- ***Entrada:*** *Um conjunto de números S.*
- ***Saída:*** *Decidir se existe $n \in S$ tal que $n = \sum_{m \in S \setminus \{n\}} m$.*

Problema (Conexidade)

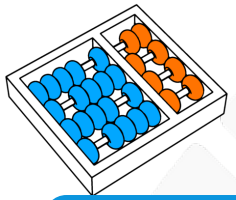
- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Decidir se G é conexo.*

Problema (Caminho mínimo)

- ***Entrada:*** *Um grafo ponderado G, dois vértices s e t e um inteiro k.*
- ***Saída:*** *Decidir se existe um caminho de s até t de peso no máximo k.*

Problema (4-Coloração)

- ***Entrada:*** *Um mapa de regiões.*



Exemplos de problemas em P

Problema (Soma de elemento)

- ***Entrada:*** *Um conjunto de números S.*
- ***Saída:*** *Decidir se existe $n \in S$ tal que $n = \sum_{m \in S \setminus \{n\}} m$.*

Problema (Conexidade)

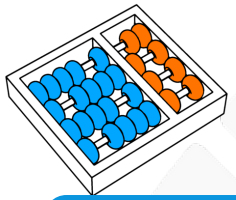
- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Decidir se G é conexo.*

Problema (Caminho mínimo)

- ***Entrada:*** *Um grafo ponderado G, dois vértices s e t e um inteiro k.*
- ***Saída:*** *Decidir se existe um caminho de s até t de peso no máximo k.*

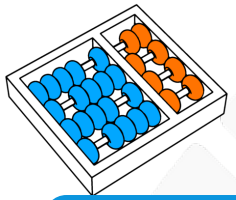
Problema (4-Coloração)

- ***Entrada:*** *Um mapa de regiões.*
- ***Saída:*** *Decidir se é possível colorir as regiões com até 4 cores de forma que regiões adjacentes tenham cores distintas.*



Exemplos de problemas em NP-completo

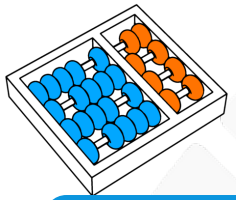
Problema (Bipartição)



Exemplos de problemas em NP-completo

Problema (Bipartição)

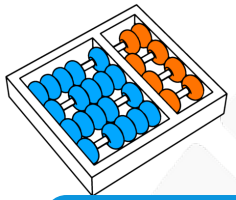
- ***Entrada:*** *Um conjunto de números S.*



Exemplos de problemas em NP-completo

Problema (Bipartição)

- ***Entrada:*** *Um conjunto de números S.*
- ***Saída:*** *Decidir se existe $N \subseteq S$ tal que $\sum_{m \in N} m = \sum_{m \in S \setminus N} m$.*

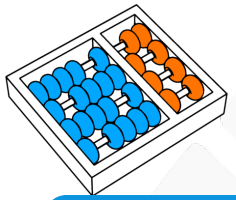


Exemplos de problemas em NP-completo

Problema (Bipartição)

- ***Entrada:*** *Um conjunto de números S.*
- ***Saída:*** *Decidir se existe $N \subseteq S$ tal que $\sum_{m \in N} m = \sum_{m \in S \setminus N} m$.*

Problema (Ciclo hamiltoniano)



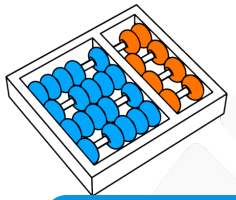
Exemplos de problemas em NP-completo

Problema (Bipartição)

- ***Entrada:*** *Um conjunto de números S.*
- ***Saída:*** *Decidir se existe $N \subseteq S$ tal que $\sum_{m \in N} m = \sum_{m \in S \setminus N} m$.*

Problema (Ciclo hamiltoniano)

- ***Entrada:*** *Um grafo G.*



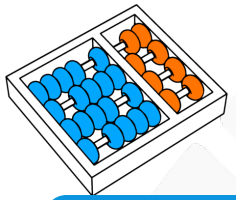
Exemplos de problemas em NP-completo

Problema (Bipartição)

- ***Entrada:*** *Um conjunto de números S.*
- ***Saída:*** *Decidir se existe $N \subseteq S$ tal que $\sum_{m \in N} m = \sum_{m \in S \setminus N} m$.*

Problema (Ciclo hamiltoniano)

- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Decidir se existe um ciclo hamiltoniano em G.*



Exemplos de problemas em NP-completo

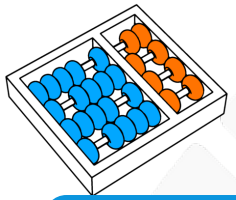
Problema (Bipartição)

- ***Entrada:*** *Um conjunto de números S.*
- ***Saída:*** *Decidir se existe $N \subseteq S$ tal que $\sum_{m \in N} m = \sum_{m \in S \setminus N} m$.*

Problema (Ciclo hamiltoniano)

- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Decidir se existe um ciclo hamiltoniano em G.*

Problema (Caminho mais longo)



Exemplos de problemas em NP-completo

Problema (Bipartição)

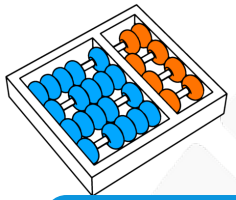
- ***Entrada:*** *Um conjunto de números S.*
- ***Saída:*** *Decidir se existe $N \subseteq S$ tal que $\sum_{m \in N} m = \sum_{m \in S \setminus N} m$.*

Problema (Ciclo hamiltoniano)

- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Decidir se existe um ciclo hamiltoniano em G.*

Problema (Caminho mais longo)

- ***Entrada:*** *Um grafo ponderado G, dois vértices s e t e um inteiro k.*



Exemplos de problemas em NP-completo

Problema (Bipartição)

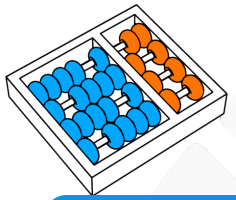
- ***Entrada:*** *Um conjunto de números S.*
- ***Saída:*** *Decidir se existe $N \subseteq S$ tal que $\sum_{m \in N} m = \sum_{m \in S \setminus N} m$.*

Problema (Ciclo hamiltoniano)

- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Decidir se existe um ciclo hamiltoniano em G.*

Problema (Caminho mais longo)

- ***Entrada:*** *Um grafo ponderado G, dois vértices s e t e um inteiro k.*
- ***Saída:*** *Decidir se existe um caminho de s até t de peso no mínimo k.*



Exemplos de problemas em NP-completo

Problema (Bipartição)

- ***Entrada:*** *Um conjunto de números S.*
- ***Saída:*** *Decidir se existe $N \subseteq S$ tal que $\sum_{m \in N} m = \sum_{m \in S \setminus N} m$.*

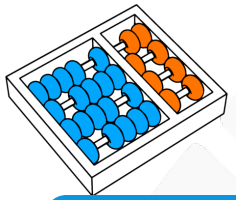
Problema (Ciclo hamiltoniano)

- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Decidir se existe um ciclo hamiltoniano em G.*

Problema (Caminho mais longo)

- ***Entrada:*** *Um grafo ponderado G, dois vértices s e t e um inteiro k.*
- ***Saída:*** *Decidir se existe um caminho de s até t de peso no mínimo k.*

Problema (3-Coloração)



Exemplos de problemas em NP-completo

Problema (Bipartição)

- ***Entrada:*** *Um conjunto de números S.*
- ***Saída:*** *Decidir se existe $N \subseteq S$ tal que $\sum_{m \in N} m = \sum_{m \in S \setminus N} m$.*

Problema (Ciclo hamiltoniano)

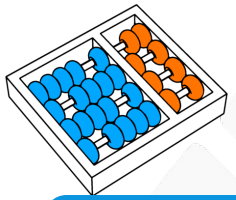
- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Decidir se existe um ciclo hamiltoniano em G.*

Problema (Caminho mais longo)

- ***Entrada:*** *Um grafo ponderado G, dois vértices s e t e um inteiro k.*
- ***Saída:*** *Decidir se existe um caminho de s até t de peso no mínimo k.*

Problema (3-Coloração)

- ***Entrada:*** *Um mapa de regiões.*



Exemplos de problemas em NP-completo

Problema (Bipartição)

- ***Entrada:*** *Um conjunto de números S.*
- ***Saída:*** *Decidir se existe $N \subseteq S$ tal que $\sum_{m \in N} m = \sum_{m \in S \setminus N} m$.*

Problema (Ciclo hamiltoniano)

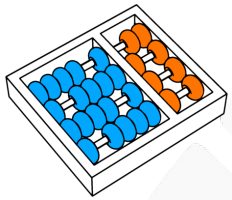
- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Decidir se existe um ciclo hamiltoniano em G.*

Problema (Caminho mais longo)

- ***Entrada:*** *Um grafo ponderado G, dois vértices s e t e um inteiro k.*
- ***Saída:*** *Decidir se existe um caminho de s até t de peso no mínimo k.*

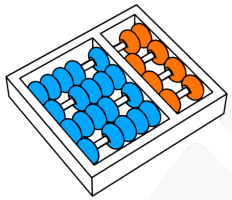
Problema (3-Coloração)

- ***Entrada:*** *Um mapa de regiões.*
- ***Saída:*** *Decidir se é possível colorir as regiões com até 3 cores de forma que regiões adjacentes tenham cores distintas.*



Uma motivação prática

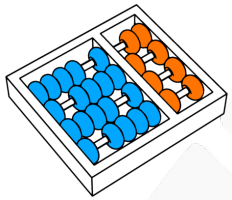
Vários problemas importantes são NP-difíceis:



Uma motivação prática

Vários problemas importantes são NP-difíceis:

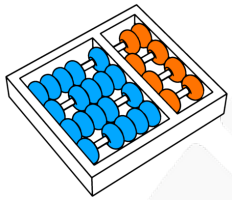
- Roteamento de veículos de cadeias de distribuição.



Uma motivação prática

Vários problemas importantes são NP-difíceis:

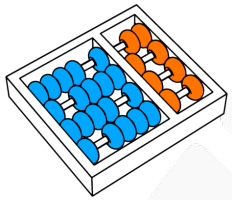
- Roteamento de veículos de cadeias de distribuição.
- Atribuição de frequências em telefonia celular.



Uma motivação prática

Vários problemas importantes são NP-difíceis:

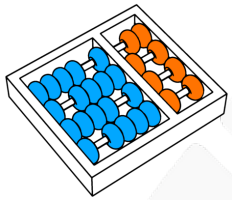
- Roteamento de veículos de cadeias de distribuição.
- Atribuição de frequências em telefonia celular.
- Empacotamento de objetos em contêineres.



Uma motivação prática

Vários problemas importantes são NP-difíceis:

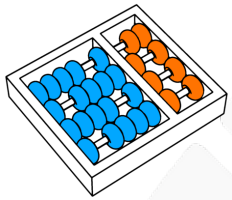
- Roteamento de veículos de cadeias de distribuição.
- Atribuição de frequências em telefonia celular.
- Empacotamento de objetos em contêineres.
- Escalonamento de funcionários em turnos de trabalho.



Uma motivação prática

Vários problemas importantes são NP-difíceis:

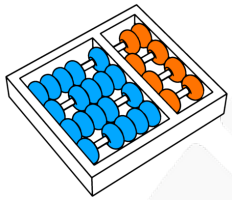
- Roteamento de veículos de cadeias de distribuição.
- Atribuição de frequências em telefonia celular.
- Empacotamento de objetos em contêineres.
- Escalonamento de funcionários em turnos de trabalho.
- Escalonamento de tarefas em computadores.



Uma motivação prática

Vários problemas importantes são NP-difíceis:

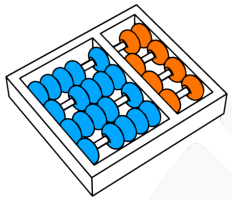
- Roteamento de veículos de cadeias de distribuição.
- Atribuição de frequências em telefonia celular.
- Empacotamento de objetos em contêineres.
- Escalonamento de funcionários em turnos de trabalho.
- Escalonamento de tarefas em computadores.
- Classificação de objetos.



Uma motivação prática

Vários problemas importantes são NP-difíceis:

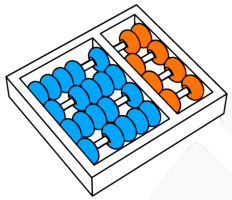
- Roteamento de veículos de cadeias de distribuição.
- Atribuição de frequências em telefonia celular.
- Empacotamento de objetos em contêineres.
- Escalonamento de funcionários em turnos de trabalho.
- Escalonamento de tarefas em computadores.
- Classificação de objetos.
- Coloração de mapas.



Uma motivação prática

Vários problemas importantes são NP-difíceis:

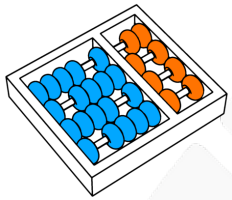
- Roteamento de veículos de cadeias de distribuição.
- Atribuição de frequências em telefonia celular.
- Empacotamento de objetos em contêineres.
- Escalonamento de funcionários em turnos de trabalho.
- Escalonamento de tarefas em computadores.
- Classificação de objetos.
- Coloração de mapas.
- Projetos de redes de computadores.



Uma motivação prática

Vários problemas importantes são NP-difíceis:

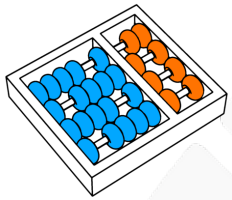
- Roteamento de veículos de cadeias de distribuição.
- Atribuição de frequências em telefonia celular.
- Empacotamento de objetos em contêineres.
- Escalonamento de funcionários em turnos de trabalho.
- Escalonamento de tarefas em computadores.
- Classificação de objetos.
- Coloração de mapas.
- Projetos de redes de computadores.
- Otimização de código.



Uma motivação prática

Vários problemas importantes são NP-difíceis:

- Roteamento de veículos de cadeias de distribuição.
- Atribuição de frequências em telefonia celular.
- Empacotamento de objetos em contêineres.
- Escalonamento de funcionários em turnos de trabalho.
- Escalonamento de tarefas em computadores.
- Classificação de objetos.
- Coloração de mapas.
- Projetos de redes de computadores.
- Otimização de código.
- Muitos outros...

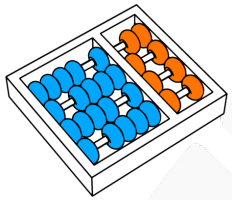


Uma motivação prática

Vários problemas importantes são NP-difíceis:

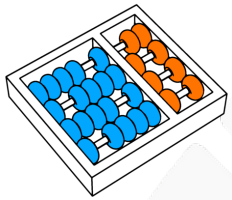
- Roteamento de veículos de cadeias de distribuição.
- Atribuição de frequências em telefonia celular.
- Empacotamento de objetos em contêineres.
- Escalonamento de funcionários em turnos de trabalho.
- Escalonamento de tarefas em computadores.
- Classificação de objetos.
- Coloração de mapas.
- Projetos de redes de computadores.
- Otimização de código.
- Muitos outros...

É **IMPRESCINDÍVEL** saber se nosso problema é NP-difícil!



Algumas sutilezas

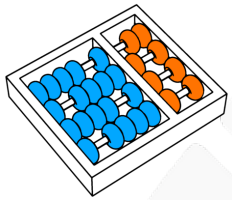
Se sabe:



Algumas sutilezas

Se sabe:

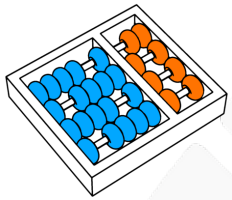
- Que o problema do caminho mais curto está em P.



Algumas sutilezas

Se sabe:

- Que o problema do caminho mais curto está em P.
- Que o problema da 4-coloração está em P.

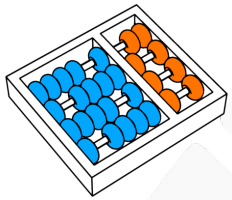


Algumas sutilezas

Se sabe:

- Que o problema do caminho mais curto está em P.
- Que o problema da 4-coloração está em P.

Ainda não se sabe:



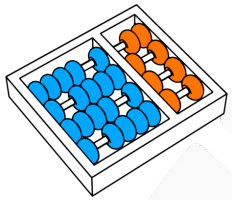
Algumas sutilezas

Se sabe:

- Que o problema do caminho mais curto está em P.
- Que o problema da 4-coloração está em P.

Ainda não se sabe:

- O problema do caminho mais **LONGO** está em P?



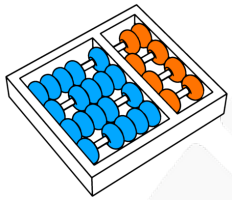
Algumas sutilezas

Se sabe:

- Que o problema do caminho mais curto está em P.
- Que o problema da 4-coloração está em P.

Ainda não se sabe:

- O problema do caminho mais **LONGO** está em P?
- O problema da **3**-coloração está em P?



Algumas sutilezas

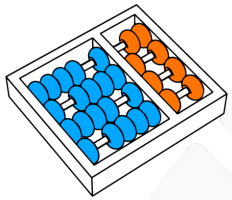
Se sabe:

- Que o problema do caminho mais curto está em P.
- Que o problema da 4-coloração está em P.

Ainda não se sabe:

- O problema do caminho mais **LONGO** está em P?
- O problema da **3**-coloração está em P?

Parecem perguntas bem diferentes:



Algumas sutilezas

Se sabe:

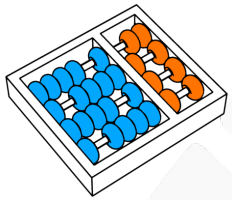
- Que o problema do caminho mais curto está em P.
- Que o problema da 4-coloração está em P.

Ainda não se sabe:

- O problema do caminho mais **LONGO** está em P?
- O problema da **3**-coloração está em P?

Parecem perguntas bem diferentes:

- Mas as duas têm respostas idênticas.



Algumas sutilezas

Se sabe:

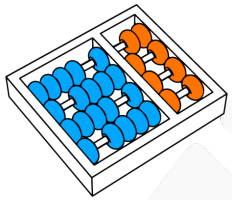
- Que o problema do caminho mais curto está em P.
- Que o problema da 4-coloração está em P.

Ainda não se sabe:

- O problema do caminho mais **LONGO** está em P?
- O problema da **3**-coloração está em P?

Parecem perguntas bem diferentes:

- Mas as duas têm respostas idênticas.
- Responder SIM é o mesmo que dizer P $=$ NP.



Algumas sutilezas

Se sabe:

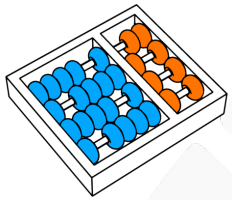
- Que o problema do caminho mais curto está em P.
- Que o problema da 4-coloração está em P.

Ainda não se sabe:

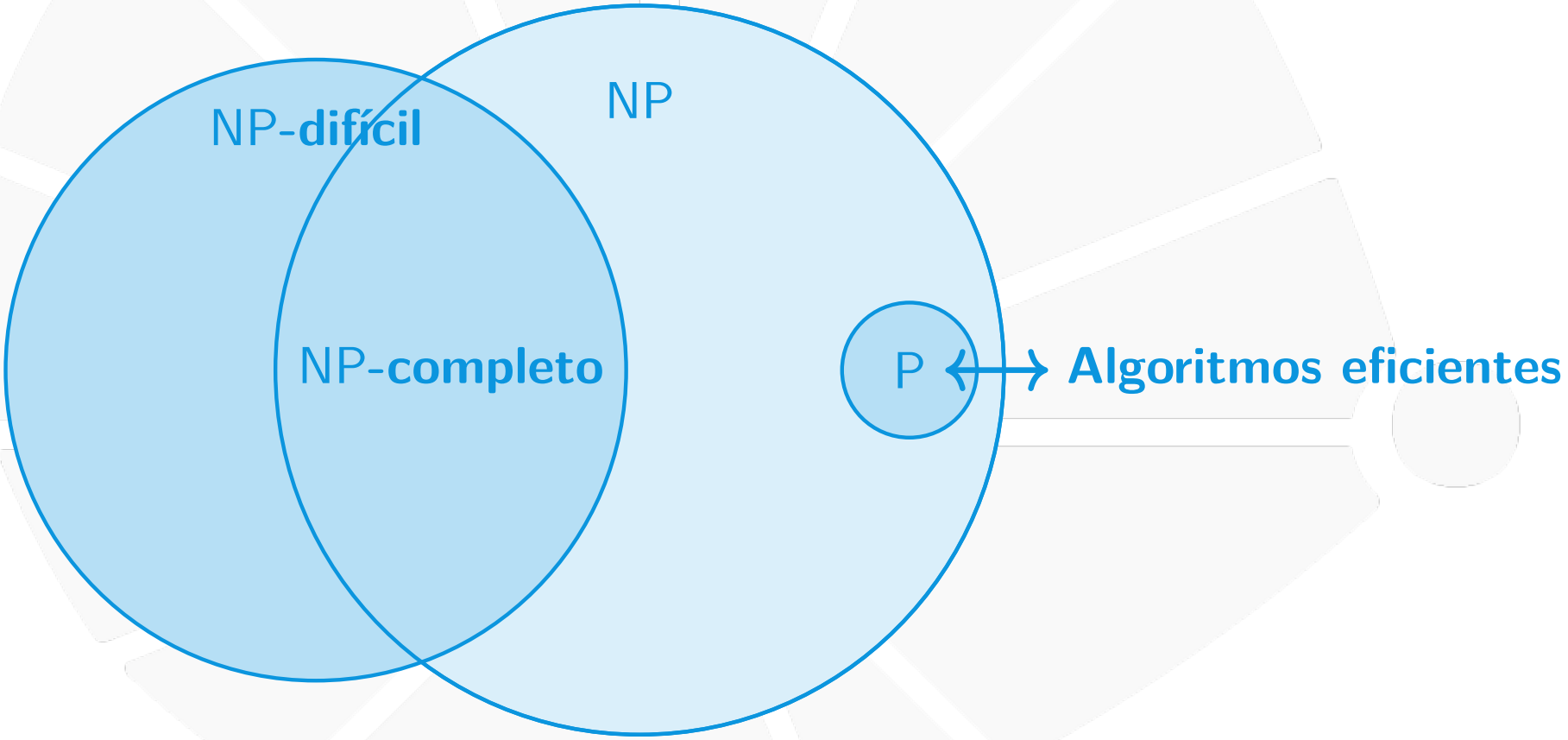
- O problema do caminho mais **LONGO** está em P?
- O problema da **3**-coloração está em P?

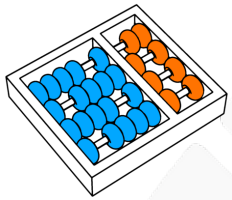
Parecem perguntas bem diferentes:

- Mas as duas têm respostas idênticas.
- Responder SIM é o mesmo que dizer P $=$ NP.
- Contudo, conjecturamos que a resposta seja NAO!

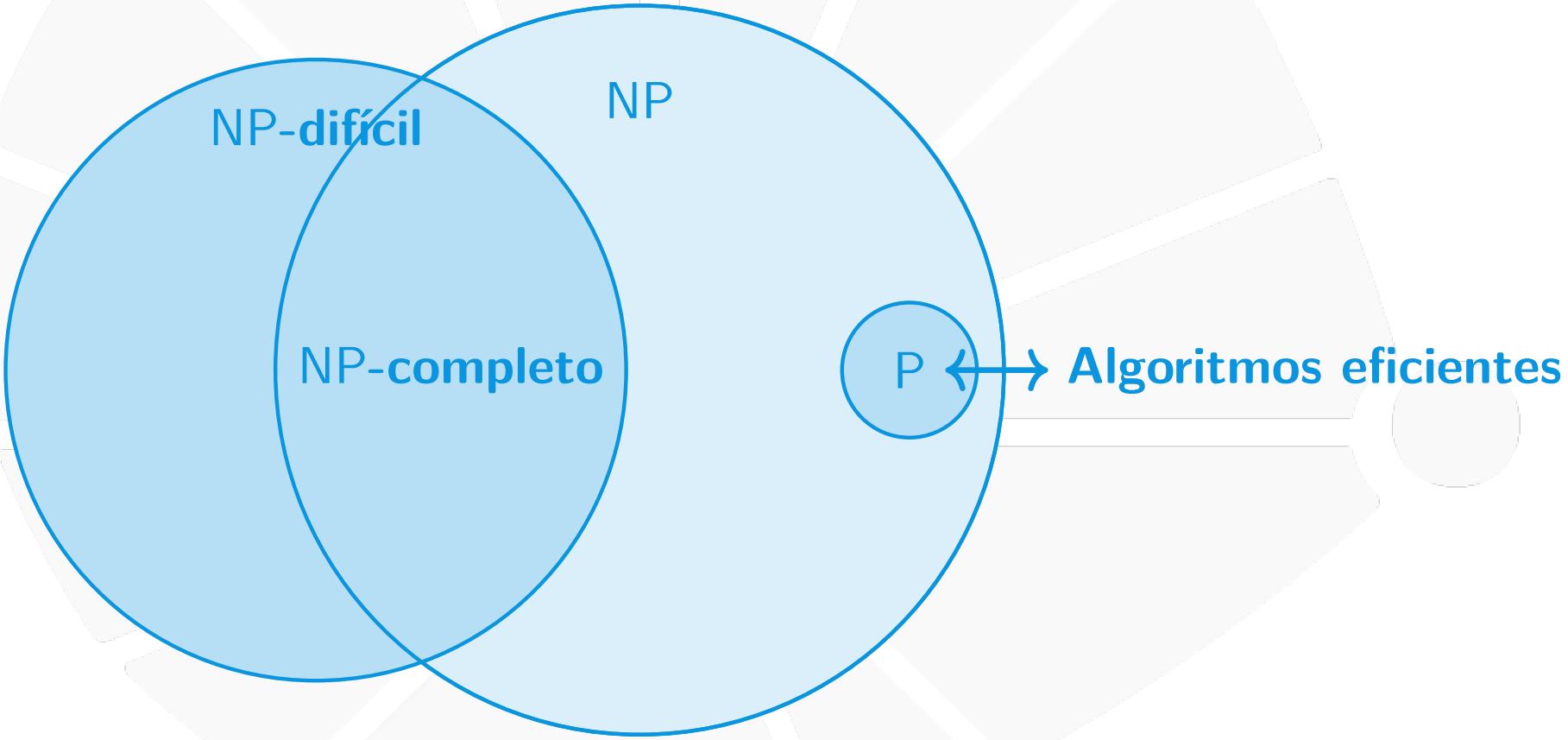


Possível configuração of classes

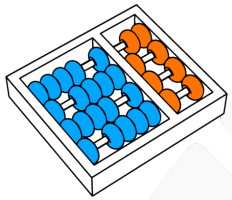




Possível configuração of classes

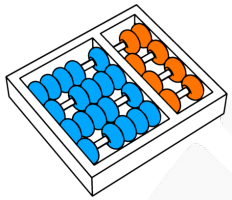


No restante do curso, procuraremos entender essa figura!



Evidências para essa configuração

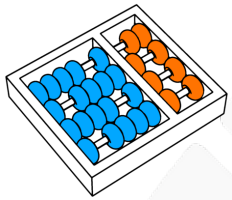
Por que acreditamos que **NÃO HÁ** algoritmo rápido para problemas NP-difíceis?



Evidências para essa configuração

Por que acreditamos que **NÃO HÁ** algoritmo rápido para problemas NP-difíceis?

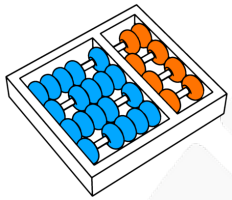
- Demonstrou-se que um problema é NP-completo pela primeira vez na década de 1970 (Cook-Levin).



Evidências para essa configuração

Por que acreditamos que **NÃO HÁ** algoritmo rápido para problemas NP-difíceis?

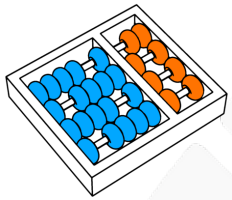
- Demonstrou-se que um problema é NP-completo pela primeira vez na década de 1970 (Cook-Levin).
- Anteriormente, vários desses problemas já eram estudados.



Evidências para essa configuração

Por que acreditamos que **NÃO HÁ** algoritmo rápido para problemas NP-difíceis?

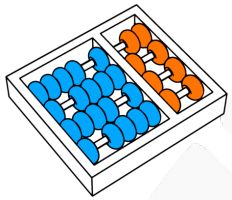
- Demonstrou-se que um problema é NP-completo pela primeira vez na década de 1970 (Cook-Levin).
- Anteriormente, vários desses problemas já eram estudados.
- Desde então, mostrou-se que inúmeros problemas importantes também são NP-completos ou NP-difíceis.



Evidências para essa configuração

Por que acreditamos que **NÃO HÁ** algoritmo rápido para problemas NP-difíceis?

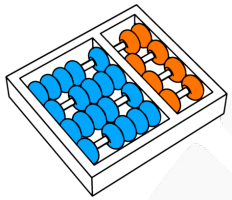
- Demonstrou-se que um problema é NP-completo pela primeira vez na década de 1970 (Cook-Levin).
- Anteriormente, vários desses problemas já eram estudados.
- Desde então, mostrou-se que inúmeros problemas importantes também são NP-completos ou NP-difíceis.
- Vários pesquisadores estudaram seus problemas NP-completos preferidos, mas ninguém descobriu qualquer algoritmo polinomial.



Evidências para essa configuração

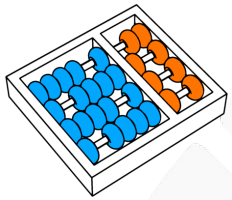
Por que acreditamos que **NÃO HÁ** algoritmo rápido para problemas NP-difíceis?

- Demonstrou-se que um problema é NP-completo pela primeira vez na década de 1970 (Cook-Levin).
- Anteriormente, vários desses problemas já eram estudados.
- Desde então, mostrou-se que inúmeros problemas importantes também são NP-completos ou NP-difíceis.
- Vários pesquisadores estudaram seus problemas NP-completos preferidos, mas ninguém descobriu qualquer algoritmo polinomial.
- Basta que um problema NP-difícil tenha algoritmo de tempo polinomial para que **TODOS** problemas em NP tenham algoritmos de tempo polinomial.



O que fazer se um problema for NP-difícil?

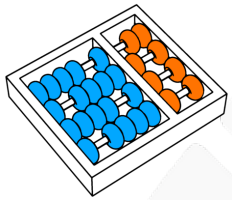
Se sabemos que um problema é NP-difícil, podemos concentrar os recursos em busca de:



O que fazer se um problema for NP-difícil?

Se sabemos que um problema é NP-difícil, podemos concentrar os recursos em busca de:

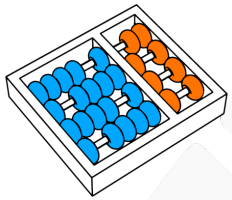
- Algoritmos para **INSTÂNCIAS PEQUENAS**.



O que fazer se um problema for NP-difícil?

Se sabemos que um problema é NP-difícil, podemos concentrar os recursos em busca de:

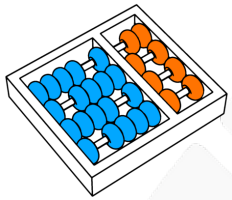
- Algoritmos para **INSTÂNCIAS PEQUENAS**.
- Algoritmos que obtêm **SOLUÇÕES APROXIMADAS**.



O que fazer se um problema for NP-difícil?

Se sabemos que um problema é NP-difícil, podemos concentrar os recursos em busca de:

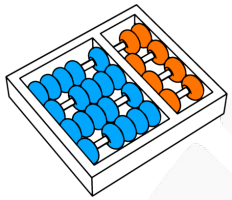
- Algoritmos para **INSTÂNCIAS PEQUENAS**.
- Algoritmos que obtêm **SOLUÇÕES APROXIMADAS**.
- Algoritmos eficientes exatos, mas para **CASOS PARTICULARES**.



O que fazer se um problema for NP-difícil?

Se sabemos que um problema é NP-difícil, podemos concentrar os recursos em busca de:

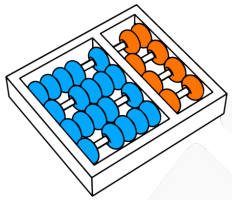
- Algoritmos para **INSTÂNCIAS PEQUENAS**.
- Algoritmos que obtêm **SOLUÇÕES APROXIMADAS**.
- Algoritmos eficientes exatos, mas para **CASOS PARTICULARES**.
- Algoritmos e métodos **HEURÍSTICOS**.



O que fazer se um problema for NP-difícil?

Se sabemos que um problema é NP-difícil, podemos concentrar os recursos em busca de:

- Algoritmos para **INSTÂNCIAS PEQUENAS**.
- Algoritmos que obtêm **SOLUÇÕES APROXIMADAS**.
- Algoritmos eficientes exatos, mas para **CASOS PARTICULARES**.
- Algoritmos e métodos **HEURÍSTICOS**.
- etc.



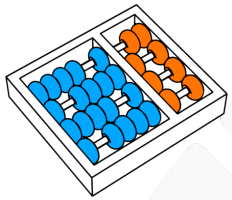
O que fazer se um problema for NP-difícil?

Se sabemos que um problema é NP-difícil, podemos concentrar os recursos em busca de:

- Algoritmos para **INSTÂNCIAS PEQUENAS**.
- Algoritmos que obtêm **SOLUÇÕES APROXIMADAS**.
- Algoritmos eficientes exatos, mas para **CASOS PARTICULARES**.
- Algoritmos e métodos **HEURÍSTICOS**.
- etc.

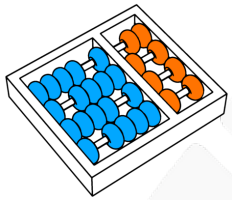
Antes, queremos identificar que problemas são NP-difíceis.

Classes de problemas



Linguagens formais

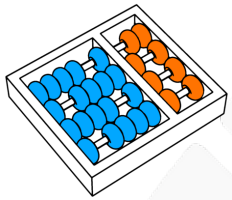
Alguns termos:



Linguagens formais

Alguns termos:

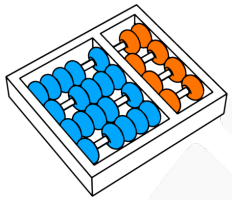
- Um alfabeto Σ é um conjunto finito de símbolos.



Linguagens formais

Alguns termos:

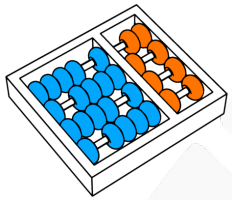
- Um alfabeto Σ é um conjunto finito de símbolos.
- Uma palavra é uma sequência finita de símbolos de Σ.



Linguagens formais

Alguns termos:

- Um alfabeto Σ é um conjunto finito de símbolos.
- Uma palavra é uma sequência finita de símbolos de Σ.
- O conjunto de todas as palavras é denotado por Σ^*, que inclui a sequência vazia, ε.

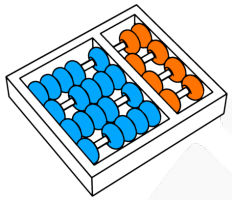


Linguagens formais

Alguns termos:

- Um alfabeto Σ é um conjunto finito de símbolos.
- Uma palavra é uma sequência finita de símbolos de Σ.
- O conjunto de todas as palavras é denotado por Σ^*, que inclui a sequência vazia, ε.

Uma **LINGUAGEM FORMAL** sobre Σ é um subconjunto $L \subseteq \Sigma^*$.



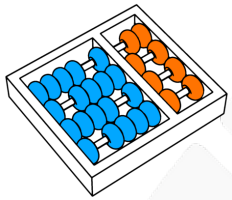
Linguagens formais

Alguns termos:

- Um alfabeto Σ é um conjunto finito de símbolos.
- Uma palavra é uma sequência finita de símbolos de Σ.
- O conjunto de todas as palavras é denotado por Σ^*, que inclui a sequência vazia, ε.

Uma **LINGUAGEM FORMAL** sobre Σ é um subconjunto $L \subseteq \Sigma^*$.

- A linguagem vazia é $L = \emptyset$.



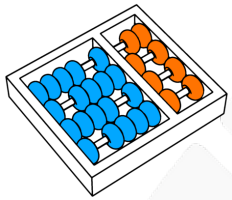
Linguagens formais

Alguns termos:

- Um alfabeto Σ é um conjunto finito de símbolos.
- Uma palavra é uma sequência finita de símbolos de Σ.
- O conjunto de todas as palavras é denotado por Σ^*, que inclui a sequência vazia, ε.

Uma **LINGUAGEM FORMAL** sobre Σ é um subconjunto $L \subseteq \Sigma^*$.

- A linguagem vazia é $L = \emptyset$.
- Uma finita é $F = \{banana, uva, pera\}$ sobre $\Sigma = \{a, b, \ldots, z\}$.



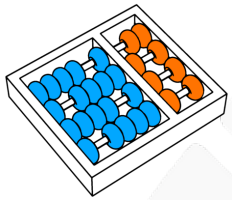
Linguagens formais

Alguns termos:

- Um alfabeto Σ é um conjunto finito de símbolos.
- Uma palavra é uma sequência finita de símbolos de Σ.
- O conjunto de todas as palavras é denotado por Σ^*, que inclui a sequência vazia, ε.

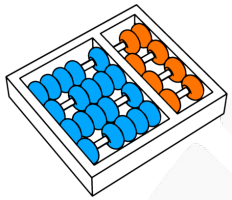
Uma **LINGUAGEM FORMAL** sobre Σ é um subconjunto $L \subseteq \Sigma^*$.

- A linguagem vazia é $L = \emptyset$.
- Uma finita é $F = \{banana, uva, pera\}$ sobre $\Sigma = \{a, b, \ldots, z\}$.
- Uma infinita é $P = \{2, 3, 7, 11, \ldots\}$ sobre $\Sigma = \{0, 1, \ldots, 9\}$.



Representando problemas de decisão

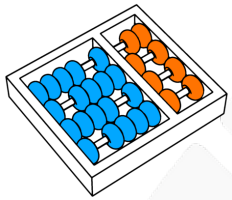
Um **PROBLEMA DE DECISÃO** é uma função $Q : \{0,1\}^* \to \{0,1\}$.



Representando problemas de decisão

Um **PROBLEMA DE DECISÃO** é uma função $Q : \{0,1\}^* \to \{0,1\}$.

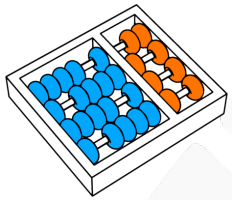
- O conjunto $\{0,1\}$ representa o alfabeto do computador.



Representando problemas de decisão

Um **PROBLEMA DE DECISÃO** é uma função $Q : \{0,1\}^* \to \{0,1\}$.

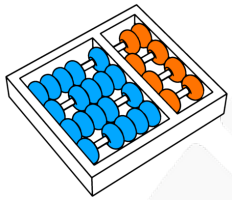
- O conjunto $\{0,1\}$ representa o alfabeto do computador.
- Uma sequência de bits $x \in \{0,1\}^*$ representa uma entrada.



Representando problemas de decisão

Um **PROBLEMA DE DECISÃO** é uma função $Q : \{0,1\}^* \to \{0,1\}$.

- O conjunto $\{0,1\}$ representa o alfabeto do computador.
- Uma sequência de bits $x \in \{0,1\}^*$ representa uma entrada.
- Um bit 0 ou 1 representa a resposta da pergunta.



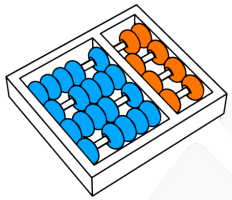
Representando problemas de decisão

Um **PROBLEMA DE DECISÃO** é uma função $Q : \{0,1\}^* \to \{0,1\}$.

- O conjunto $\{0,1\}$ representa o alfabeto do computador.
- Uma sequência de bits $x \in \{0,1\}^*$ representa uma entrada.
- Um bit 0 ou 1 representa a resposta da pergunta.

A **LINGUAGEM** correspondente a um problema de decisão Q é o conjunto de instâncias L com resposta SIM, isso é:

$$L = \{x \in \{0,1\}^* \ : \ Q(x) = 1\}.$$



Representando problemas de decisão

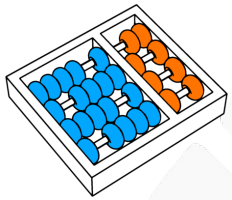
Um **PROBLEMA DE DECISÃO** é uma função $Q : \{0,1\}^* \to \{0,1\}$.

- O conjunto $\{0,1\}$ representa o alfabeto do computador.
- Uma sequência de bits $x \in \{0,1\}^*$ representa uma entrada.
- Um bit 0 ou 1 representa a resposta da pergunta.

A **LINGUAGEM** correspondente a um problema de decisão Q é o conjunto de instâncias L com resposta SIM, isso é:

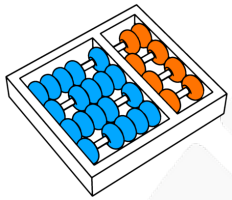
$$L = \{x \in \{0,1\}^* \ : \ Q(x) = 1\}.$$

- Identificamos um problema Q com sua linguagem L.



Codificação

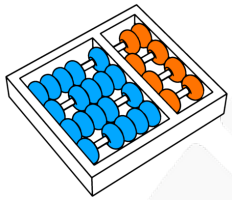
Uma **CODIFICAÇÃO** é o mapeamento utilizado para representar um objeto como uma palavra de Σ^*.



Codificação

Uma **CODIFICAÇÃO** é o mapeamento utilizado para representar um objeto como uma palavra de Σ^*.

- Dado um objeto A, sua codificação é denotada por $\langle A \rangle$.



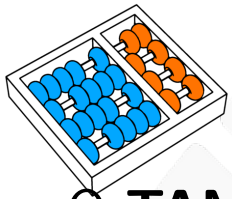
Codificação

Uma **CODIFICAÇÃO** é o mapeamento utilizado para representar um objeto como uma palavra de Σ^*.

- Dado um objeto A, sua codificação é denotada por $\langle A \rangle$.

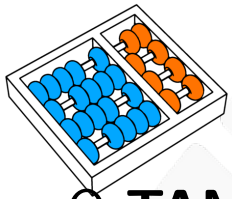
Problema (Caminho mínimo)

$$PATH = \{\langle G, u, v, k\rangle \quad : \quad \begin{array}{l} G \text{ é um grafo,} \\ u, v \text{ são vértices de } G \text{ e} \\ k \text{ é um inteiro tais que} \\ \text{existe caminho de } u \text{ até } v \text{ em } G \\ \text{de tamanho no máximo } k\} \end{array}$$



Tamanho da instância

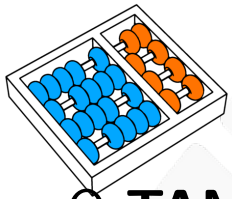
O **TAMANHO** de uma instância $x \in \{0,1\}^*$ é o número de bits de x.



Tamanho da instância

O **TAMANHO** de uma instância $x \in \{0,1\}^*$ é o número de bits de x.

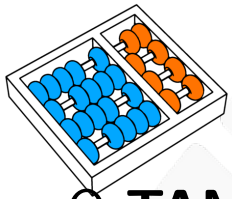
- Denotamos o tamanho de x por $n = |x|$.



Tamanho da instância

O **TAMANHO** de uma instância $x \in \{0,1\}^*$ é o número de bits de x.

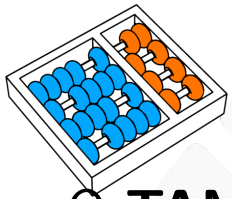
- Denotamos o tamanho de x por $n = |x|$.
- Para um objeto de alto nível A, temos $n = |\langle A \rangle|$.



Tamanho da instância

O **TAMANHO** de uma instância $x \in \{0,1\}^*$ é o número de bits de x.

- Denotamos o tamanho de x por $n = |x|$.
- Para um objeto de alto nível A, temos $n = |\langle A \rangle|$.
- Precisamos tomar cuidado com a codificação.

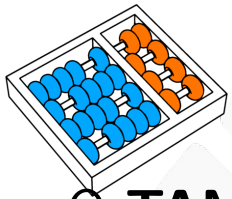


Tamanho da instância

O **TAMANHO** de uma instância $x \in \{0,1\}^*$ é o número de bits de x.

- Denotamos o tamanho de x por $n = |x|$.
- Para um objeto de alto nível A, temos $n = |\langle A \rangle|$.
- Precisamos tomar cuidado com a codificação.

Considere a linguagem PARES $= \{\langle k \rangle : k \text{ é par}\}$.



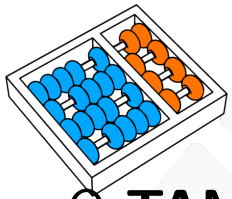
Tamanho da instância

O **TAMANHO** de uma instância $x \in \{0,1\}^*$ é o número de bits de x.

- Denotamos o tamanho de x por $n = |x|$.
- Para um objeto de alto nível A, temos $n = |\langle A \rangle|$.
- Precisamos tomar cuidado com a codificação.

Considere a linguagem PARES $= \{\langle k \rangle : k \text{ é par}\}$.

- Em unário:



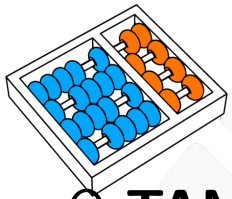
Tamanho da instância

O **TAMANHO** de uma instância $x \in \{0,1\}^*$ é o número de bits de x.

- Denotamos o tamanho de x por $n = |x|$.
- Para um objeto de alto nível A, temos $n = |\langle A \rangle|$.
- Precisamos tomar cuidado com a codificação.

Considere a linguagem PARES $= \{\langle k \rangle : k \text{ é par}\}$.

- Em unário:
  - Representamos 100 como $1111111 \ldots 1111111$.



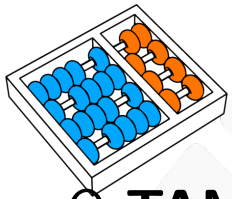
Tamanho da instância

O **TAMANHO** de uma instância $x \in \{0,1\}^*$ é o número de bits de x.

- Denotamos o tamanho de x por $n = |x|$.
- Para um objeto de alto nível A, temos $n = |\langle A \rangle|$.
- Precisamos tomar cuidado com a codificação.

Considere a linguagem PARES $= \{\langle k \rangle : k \text{ é par}\}$.

- Em unário:
  - Representamos 100 como $1111111 \ldots 1111111$.
  - Nesse caso, o tamanho é $n = |\langle k \rangle| = \Theta(k)$.



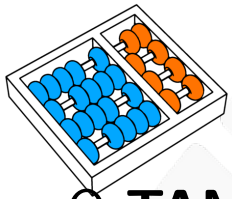
Tamanho da instância

O **TAMANHO** de uma instância $x \in \{0,1\}^*$ é o número de bits de x.

- Denotamos o tamanho de x por $n = |x|$.
- Para um objeto de alto nível A, temos $n = |\langle A \rangle|$.
- Precisamos tomar cuidado com a codificação.

Considere a linguagem PARES $= \{\langle k \rangle : k \text{ é par}\}$.

- Em unário:
  - Representamos 100 como $1111111 \ldots 1111111$.
  - Nesse caso, o tamanho é $n = |\langle k \rangle| = \Theta(k)$.
- Em binário:



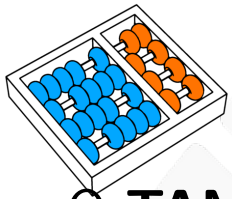
Tamanho da instância

O **TAMANHO** de uma instância $x \in \{0,1\}^*$ é o número de bits de x.

- Denotamos o tamanho de x por $n = |x|$.
- Para um objeto de alto nível A, temos $n = |\langle A \rangle|$.
- Precisamos tomar cuidado com a codificação.

Considere a linguagem PARES $= \{\langle k \rangle : k \text{ é par}\}$.

- Em unário:
  - Representamos 100 como $1111111\ldots1111111$.
  - Nesse caso, o tamanho é $n = |\langle k \rangle| = \Theta(k)$.
- Em binário:
  - Representamos 100 como 1100100.



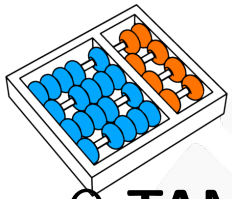
Tamanho da instância

O **TAMANHO** de uma instância $x \in \{0,1\}^*$ é o número de bits de x.

- Denotamos o tamanho de x por $n = |x|$.
- Para um objeto de alto nível A, temos $n = |\langle A \rangle|$.
- Precisamos tomar cuidado com a codificação.

Considere a linguagem PARES $= \{\langle k \rangle : k \text{ é par}\}$.

- Em unário:
  - Representamos 100 como 1111111 . . . 1111111.
  - Nesse caso, o tamanho é $n = |\langle k \rangle| = \Theta(k)$.
- Em binário:
  - Representamos 100 como 1100100.
  - Nesse caso, o tamanho é $n = |\langle k \rangle| = \lceil \log_2 k \rceil = \Theta(\log k)$.



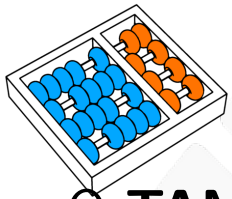
Tamanho da instância

O **TAMANHO** de uma instância $x \in \{0,1\}^*$ é o número de bits de x.

- Denotamos o tamanho de x por $n = |x|$.
- Para um objeto de alto nível A, temos $n = |\langle A \rangle|$.
- Precisamos tomar cuidado com a codificação.

Considere a linguagem PARES $= \{\langle k \rangle : k \text{ é par}\}$.

- Em unário:
  - Representamos 100 como 1111111 . . . 1111111.
  - Nesse caso, o tamanho é $n = |\langle k \rangle| = \Theta(k)$.
- Em binário:
  - Representamos 100 como 1100100.
  - Nesse caso, o tamanho é $n = |\langle k \rangle| = \lceil \log_2 k \rceil = \Theta(\log k)$.
- Em ternário, quaternário etc:



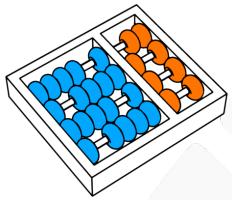
Tamanho da instância

O **TAMANHO** de uma instância $x \in \{0,1\}^*$ é o número de bits de x.

- Denotamos o tamanho de x por $n = |x|$.
- Para um objeto de alto nível A, temos $n = |\langle A \rangle|$.
- Precisamos tomar cuidado com a codificação.

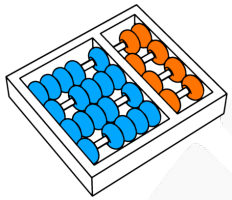
Considere a linguagem PARES $= \{\langle k \rangle : k \text{ é par}\}$.

- Em unário:
  - Representamos 100 como 1111111 . . . 1111111.
  - Nesse caso, o tamanho é $n = |\langle k \rangle| = \Theta(k)$.
- Em binário:
  - Representamos 100 como 1100100.
  - Nesse caso, o tamanho é $n = |\langle k \rangle| = \lceil \log_2 k \rceil = \Theta(\log k)$.
- Em ternário, quaternário etc:
  - Também, o tamanho é $n = |\langle k \rangle| = \lceil \log_b k \rceil = \Theta(\log k)$.



Linguagem aceita

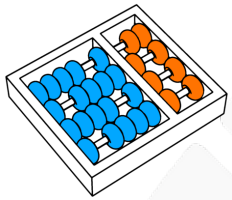
Considere um algoritmo A e uma entrada $x \in \{0,1\}^*$.



Linguagem aceita

Considere um algoritmo A e uma entrada $x \in \{0,1\}^*$.

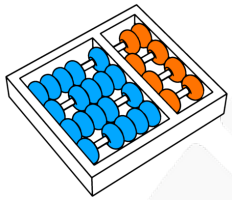
- Pode ser que A termina ao receber x e devolve $A(x) = 1$.



Linguagem aceita

Considere um algoritmo A e uma entrada $x \in \{0,1\}^*$.

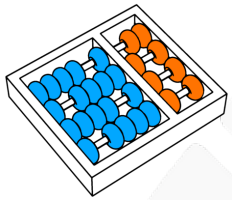
- Pode ser que A termina ao receber x e devolve $A(x) = 1$.
- Pode ser que A termina ao receber x e devolve $A(x) = 0$.



Linguagem aceita

Considere um algoritmo A e uma entrada $x \in \{0,1\}^*$.

- Pode ser que A termina ao receber x e devolve $A(x) = 1$.
- Pode ser que A termina ao receber x e devolve $A(x) = 0$.
- Pode ser que A não termina ao receber x.



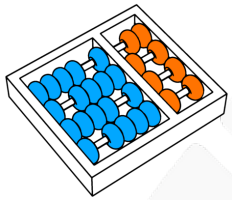
Linguagem aceita

Considere um algoritmo A e uma entrada $x \in \{0,1\}^*$.

- Pode ser que A termina ao receber x e devolve $A(x) = 1$.
- Pode ser que A termina ao receber x e devolve $A(x) = 0$.
- Pode ser que A não termina ao receber x.

Um algoritmo A **ACEITA** uma linguagem L se:

$$L = \{x \in \{0,1\}^* \ : \ A(x) = 1\}.$$



Linguagem aceita

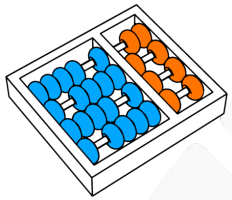
Considere um algoritmo A e uma entrada $x \in \{0,1\}^*$.

- Pode ser que A termina ao receber x e devolve $A(x) = 1$.
- Pode ser que A termina ao receber x e devolve $A(x) = 0$.
- Pode ser que A não termina ao receber x.

Um algoritmo A **ACEITA** uma linguagem L se:

$$L = \{x \in \{0,1\}^* \ : \ A(x) = 1\}.$$

Um algoritmo A **DECIDE** L se para todo $x \in \{0,1\}^*$:



Linguagem aceita

Considere um algoritmo A e uma entrada $x \in \{0,1\}^*$.

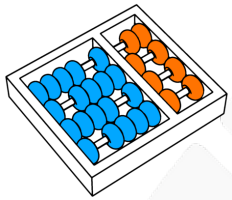
- Pode ser que A termina ao receber x e devolve $A(x) = 1$.
- Pode ser que A termina ao receber x e devolve $A(x) = 0$.
- Pode ser que A não termina ao receber x.

Um algoritmo A **ACEITA** uma linguagem L se:

$$L = \{x \in \{0,1\}^* \ : \ A(x) = 1\}.$$

Um algoritmo A **DECIDE** L se para todo $x \in \{0,1\}^*$:

- se $x \in L$, então $A(x) = 1$ (A aceita x).



Linguagem aceita

Considere um algoritmo A e uma entrada $x \in \{0,1\}^*$.

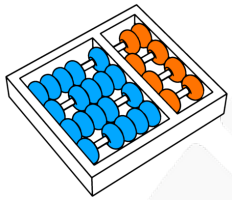
- Pode ser que A termina ao receber x e devolve $A(x) = 1$.
- Pode ser que A termina ao receber x e devolve $A(x) = 0$.
- Pode ser que A não termina ao receber x.

Um algoritmo A **ACEITA** uma linguagem L se:

$$L = \{x \in \{0,1\}^* \ : \ A(x) = 1\}.$$

Um algoritmo A **DECIDE** L se para todo $x \in \{0,1\}^*$:

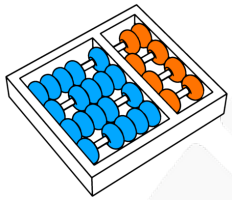
- se $x \in L$, então $A(x) = 1$ (A aceita x).
- se $x \notin L$, então $A(x) = 0$ (A rejeita x).



Classe P

Definição

*A **CLASSE** P é o conjunto de linguagens $L \subseteq \{0,1\}^*$ para as quais existe algoritmo A que decide L em tempo polinomial.*

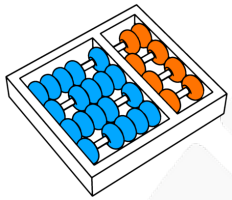


Classe P

Definição

*A **CLASSE** P é o conjunto de linguagens $L \subseteq \{0,1\}^*$ para as quais existe algoritmo A que decide L em tempo polinomial.*

Em outras palavras, se $L \in$ P, então:



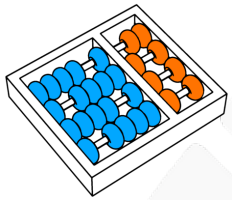
Classe P

Definição

*A **CLASSE** P é o conjunto de linguagens $L \subseteq \{0,1\}^*$ para as quais existe algoritmo A que decide L em tempo polinomial.*

Em outras palavras, se $L \in$ P, então:

1. Existe algoritmo $A(x)$ que decide L.



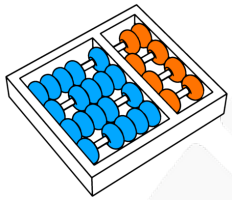
Classe P

Definição

*A **CLASSE** P é o conjunto de linguagens $L \subseteq \{0,1\}^*$ para as quais existe algoritmo A que decide L em tempo polinomial.*

Em outras palavras, se $L \in \mathrm{P}$, então:

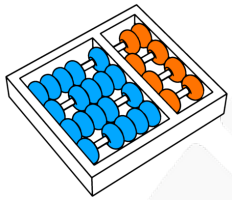
1. Existe algoritmo $A(x)$ que decide L.
2. Esse algoritmo executa em tempo polinomial em $|x|$.



Aceita e decide em tempo polinomial

Teorema

$\mathsf{P} = \{L \subseteq \{0,1\}^* : L \text{ é aceita por um algoritmo polinomial}\}$

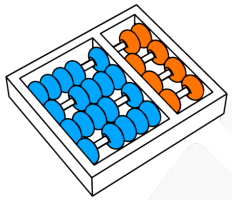


Aceita e decide em tempo polinomial

Teorema

$\mathrm{P} = \{L \subseteq \{0,1\}^* : L \text{ é aceita por um algoritmo polinomial}\}$

Demonstração:

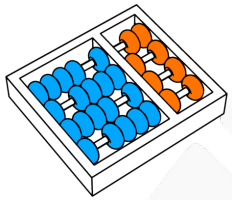


Aceita e decide em tempo polinomial

Teorema

$\mathrm{P} = \{L \subseteq \{0,1\}^* : L \text{ é aceita por um algoritmo polinomial}\}$

Demonstração:



Aceita e decide em tempo polinomial

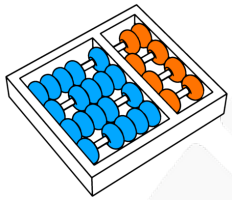
Teorema

$\mathrm{P} = \{L \subseteq \{0,1\}^* : L \textit{ é aceita por um algoritmo polinomial}\}$

Demonstração:

$\subseteq$

- Considere $L \in \mathrm{P}$.



Aceita e decide em tempo polinomial

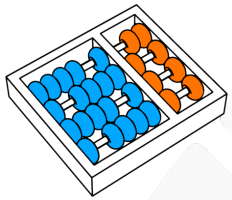
Teorema

$\mathrm{P} = \{L \subseteq \{0,1\}^* : L \textit{ é aceita por um algoritmo polinomial}\}$

Demonstração:

$\subseteq$

- Considere $L \in \mathrm{P}$.
- Por definição de P, existe um algoritmo que decide L.



Aceita e decide em tempo polinomial

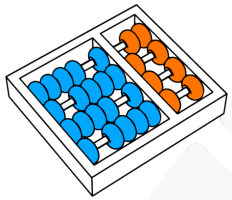
Teorema

$\mathrm{P} = \{L \subseteq \{0,1\}^* : L \textit{ é aceita por um algoritmo polinomial}\}$

Demonstração:

$\subseteq$

- Considere $L \in \mathrm{P}$.
- Por definição de P, existe um algoritmo que decide L.
- Logo, também existe um algoritmo que aceita L.



Aceita e decide em tempo polinomial

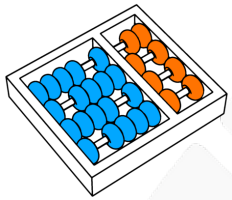
Teorema

$\mathrm{P} = \{L \subseteq \{0,1\}^* : L \textit{ é aceita por um algoritmo polinomial}\}$

Demonstração:

$\subseteq$

- Considere $L \in \mathrm{P}$.
- Por definição de P, existe um algoritmo que decide L.
- Logo, também existe um algoritmo que aceita L.



Aceita e decide em tempo polinomial

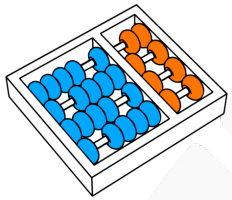
Teorema

$\mathrm{P} = \{L \subseteq \{0,1\}^* : L \text{ é aceita por um algoritmo polinomial}\}$

Demonstração:

$\subseteq$
- Considere $L \in \mathrm{P}$.
- Por definição de P, existe um algoritmo que decide L.
- Logo, também existe um algoritmo que aceita L.

$\supseteq$
- Suponha que L é aceita por um algoritmo polinomial A.



Aceita e decide em tempo polinomial

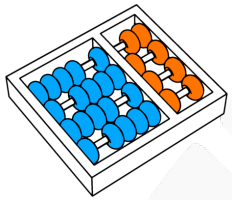
Teorema

$\mathrm{P} = \{L \subseteq \{0,1\}^* : L \text{ é aceita por um algoritmo polinomial}\}$

Demonstração:

$\subseteq$
- Considere $L \in \mathrm{P}$.
- Por definição de P, existe um algoritmo que decide L.
- Logo, também existe um algoritmo que aceita L.

$\supseteq$
- Suponha que L é aceita por um algoritmo polinomial A.
- O tempo do algoritmo é n^k, para alguma constante k.



Aceita e decide em tempo polinomial

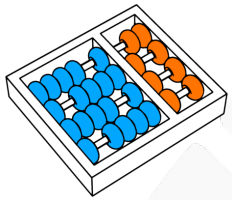
Teorema

$\mathrm{P} = \{L \subseteq \{0,1\}^* : L \text{ é aceita por um algoritmo polinomial}\}$

Demonstração:

$\subseteq$
- Considere $L \in \mathrm{P}$.
- Por definição de P, existe um algoritmo que decide L.
- Logo, também existe um algoritmo que aceita L.

$\supseteq$
- Suponha que L é aceita por um algoritmo polinomial A.
- O tempo do algoritmo é n^k, para alguma constante k.
- Construa um algoritmo A' para uma entrada x:



Aceita e decide em tempo polinomial

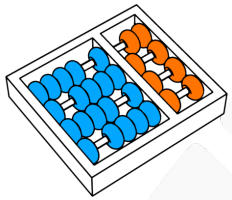
Teorema

$\mathrm{P} = \{L \subseteq \{0,1\}^* : L \text{ é aceita por um algoritmo polinomial}\}$

Demonstração:

$\subseteq$
- Considere $L \in \mathrm{P}$.
- Por definição de P, existe um algoritmo que decide L.
- Logo, também existe um algoritmo que aceita L.

$\supseteq$
- Suponha que L é aceita por um algoritmo polinomial A.
- O tempo do algoritmo é n^k, para alguma constante k.
- Construa um algoritmo A' para uma entrada x:
  1. Simule o algoritmo A executando **NO MÁXIMO** n^k passos.



Aceita e decide em tempo polinomial

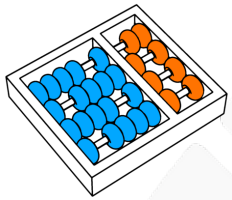
Teorema

$\mathrm{P} = \{L \subseteq \{0,1\}^* : L \text{ é aceita por um algoritmo polinomial}\}$

Demonstração:

$\subseteq$
- Considere $L \in \mathrm{P}$.
- Por definição de P, existe um algoritmo que decide L.
- Logo, também existe um algoritmo que aceita L.

$\supseteq$
- Suponha que L é aceita por um algoritmo polinomial A.
- O tempo do algoritmo é n^k, para alguma constante k.
- Construa um algoritmo A' para uma entrada x:
  1. Simule o algoritmo A executando **NO MÁXIMO** n^k passos.
  2. Se A aceitou x, então responda SIM.



Aceita e decide em tempo polinomial

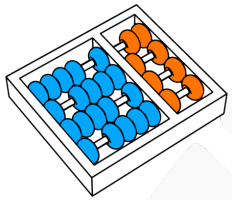
Teorema

$\mathrm{P} = \{L \subseteq \{0,1\}^* : L \text{ é aceita por um algoritmo polinomial}\}$

Demonstração:

$\subseteq$
- Considere $L \in \mathrm{P}$.
- Por definição de P, existe um algoritmo que decide L.
- Logo, também existe um algoritmo que aceita L.

$\supseteq$
- Suponha que L é aceita por um algoritmo polinomial A.
- O tempo do algoritmo é n^k, para alguma constante k.
- Construa um algoritmo A' para uma entrada x:
  1. Simule o algoritmo A executando **NO MÁXIMO** n^k passos.
  2. Se A aceitou x, então responda SIM.
  3. Se A rejeitou x ou não terminou, então responda NAO.



Aceita e decide em tempo polinomial

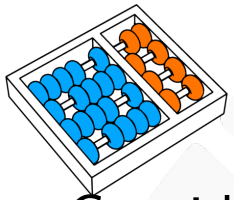
Teorema

$\mathrm{P} = \{L \subseteq \{0,1\}^* : L \text{ é aceita por um algoritmo polinomial}\}$

Demonstração:

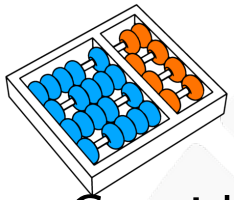
$\subseteq$
- Considere $L \in \mathrm{P}$.
- Por definição de P, existe um algoritmo que decide L.
- Logo, também existe um algoritmo que aceita L.

$\supseteq$
- Suponha que L é aceita por um algoritmo polinomial A.
- O tempo do algoritmo é n^k, para alguma constante k.
- Construa um algoritmo A' para uma entrada x:
  1. Simule o algoritmo A executando **NO MÁXIMO** n^k passos.
  2. Se A aceitou x, então responda SIM.
  3. Se A rejeitou x ou não terminou, então responda NAO.
- Observe que o algoritmo A' decide L em tempo polinomial.



Certificado

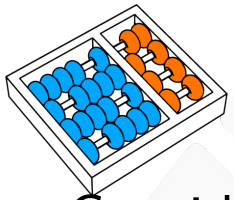
Considere uma linguagem L:



Certificado

Considere uma linguagem L:

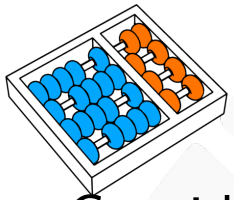
- Tome uma instância x do problema correspondente.



Certificado

Considere uma linguagem L:

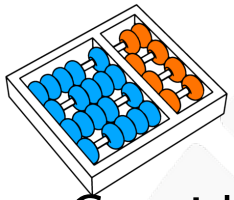
- Tome uma instância x do problema correspondente.
- Queremos encontrar uma sequência de bits $y \in \{0,1\}^*$.



Certificado

Considere uma linguagem L:

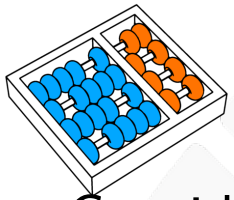
- Tome uma instância x do problema correspondente.
- Queremos encontrar uma sequência de bits $y \in \{0,1\}^*$.
- De forma que verificar y permite concluir que $x \in L$.



Certificado

Considere uma linguagem L:

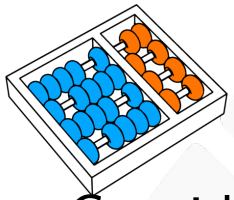
- Tome uma instância x do problema correspondente.
- Queremos encontrar uma sequência de bits $y \in \{0,1\}^*$.
- De forma que verificar y permite concluir que $x \in L$.
- Normalmente y é a codificação de uma solução para x.



Certificado

Considere uma linguagem L:

- Tome uma instância x do problema correspondente.
- Queremos encontrar uma sequência de bits $y \in \{0,1\}^*$.
- De forma que verificar y permite concluir que $x \in L$.
- Normalmente y é a codificação de uma solução para x.
- Chamamos y de **CERTIFICADO** para x.

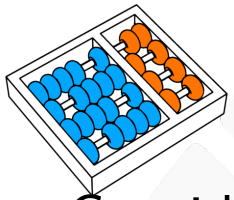


Certificado

Considere uma linguagem L:

- Tome uma instância x do problema correspondente.
- Queremos encontrar uma sequência de bits $y \in \{0,1\}^*$.
- De forma que verificar y permite concluir que $x \in L$.
- Normalmente y é a codificação de uma solução para x.
- Chamamos y de **CERTIFICADO** para x.

Problema (Certificado para PATH)



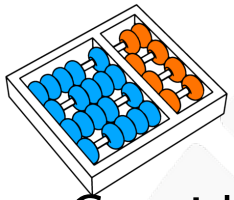
Certificado

Considere uma linguagem L:

- Tome uma instância x do problema correspondente.
- Queremos encontrar uma sequência de bits $y \in \{0,1\}^*$.
- De forma que verificar y permite concluir que $x \in L$.
- Normalmente y é a codificação de uma solução para x.
- Chamamos y de **CERTIFICADO** para x.

Problema (Certificado para PATH)

- *Tome uma instância* $x = \langle G, u, v, k \rangle$ *de PATH*



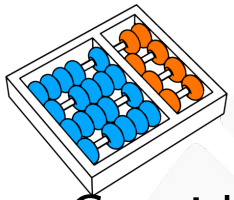
Certificado

Considere uma linguagem L:

- Tome uma instância x do problema correspondente.
- Queremos encontrar uma sequência de bits $y \in \{0,1\}^*$.
- De forma que verificar y permite concluir que $x \in L$.
- Normalmente y é a codificação de uma solução para x.
- Chamamos y de **CERTIFICADO** para x.

Problema (Certificado para PATH)

- *Tome uma instância* $x = \langle G, u, v, k \rangle$ *de PATH*
  1. *Se* x *é* SIM, ***EXISTE*** *caminho* P *de* u *a* v *com até* k *arestas.*



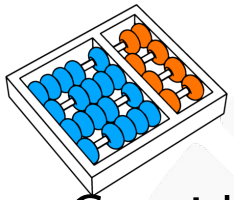
Certificado

Considere uma linguagem L:

- Tome uma instância x do problema correspondente.
- Queremos encontrar uma sequência de bits $y \in \{0,1\}^*$.
- De forma que verificar y permite concluir que $x \in L$.
- Normalmente y é a codificação de uma solução para x.
- Chamamos y de **CERTIFICADO** para x.

Problema (Certificado para PATH)

- *Tome uma instância $x = \langle G, u, v, k \rangle$ de PATH*
  1. *Se x é* SIM, ***EXISTE*** *caminho P de u a v com até k arestas.*
  2. *Se x é* NAO, ***NÃO EXISTE*** *caminho de u a v com até k arestas.*



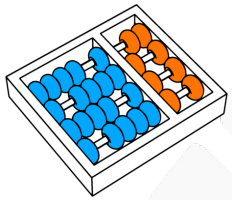
Certificado

Considere uma linguagem L:

- Tome uma instância x do problema correspondente.
- Queremos encontrar uma sequência de bits $y \in \{0,1\}^*$.
- De forma que verificar y permite concluir que $x \in L$.
- Normalmente y é a codificação de uma solução para x.
- Chamamos y de **CERTIFICADO** para x.

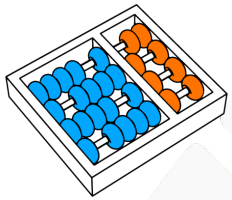
Problema (Certificado para PATH)

- *Tome uma instância $x = \langle G, u, v, k \rangle$ de PATH*
  1. *Se x é* SIM, ***EXISTE*** *caminho P de u a v com até k arestas.*
  2. *Se x é* NAO, ***NÃO EXISTE*** *caminho de u a v com até k arestas.*
- *No primeiro caso, $y = \langle P \rangle$ é um certificado de que $x \in PATH$.*



Verificador

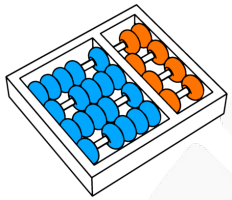
Um **VERIFICADOR** para uma linguagem L é um algoritmo que recebe uma instância x e um sequência de bits y tal que:



Verificador

Um **VERIFICADOR** para uma linguagem L é um algoritmo que recebe uma instância x e um sequência de bits y tal que:

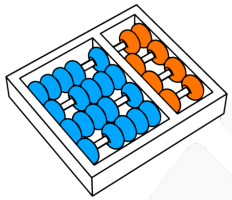
- Se $x \in L$, ele devolve SIM para algum certificado y.



Verificador

Um **VERIFICADOR** para uma linguagem L é um algoritmo que recebe uma instância x e um sequência de bits y tal que:

- Se $x \in L$, ele devolve SIM para algum certificado y.
- Se $x \notin L$, ele devolve NAO independentemente de y.



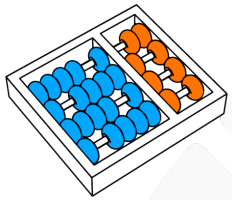
Verificador

Um **VERIFICADOR** para uma linguagem L é um algoritmo que recebe uma instância x e um sequência de bits y tal que:

- Se $x \in L$, ele devolve SIM para algum certificado y.
- Se $x \notin L$, ele devolve NAO independentemente de y.

Algoritmo 4: VERIFICA-PATH$(\langle G, u, v, k\rangle, \langle P\rangle)$

```
1 se ⟨P⟩ não é codificação de um caminho de G
2     devolva NAO
3 se P tem mais que k arestas
4     devolva NAO
5 se P não sai de u e chega em v
6     devolva NAO
7 devolva SIM
```



Verificador

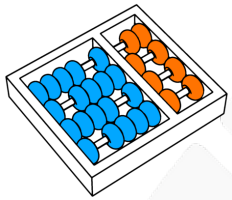
Um **VERIFICADOR** para uma linguagem L é um algoritmo que recebe uma instância x e um sequência de bits y tal que:

- Se $x \in L$, ele devolve SIM para algum certificado y.
- Se $x \notin L$, ele devolve NAO independentemente de y.

Algoritmo 5: VERIFICA-PATH$(\langle G, u, v, k\rangle, \langle P\rangle)$

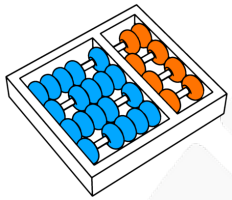
```
1 se ⟨P⟩ não é codificação de um caminho de G
2     devolva NAO
3 se P tem mais que k arestas
4     devolva NAO
5 se P não sai de u e chega em v
6     devolva NAO
7 devolva SIM
```

- Normalmente, omitimos o passo que valida a codificação.



Tempo de verificação

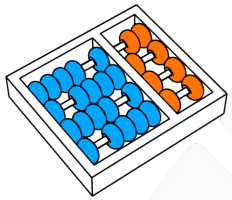
Queremos executar o verificador em tempo polinomial:



Tempo de verificação

Queremos executar o verificador em tempo polinomial:

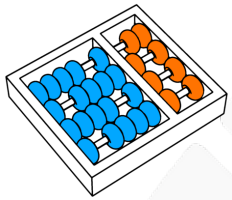
1. O **TEMPO DO VERIFICADOR** deve ser polinomial em $|x|$ e $|y|$.



Tempo de verificação

Queremos executar o verificador em tempo polinomial:

1. O **TEMPO DO VERIFICADOR** deve ser polinomial em $|x|$ e $|y|$.
2. o **TAMANHO DO CERTIFICADO** $|y|$ deve ser polinomial em $|x|$.

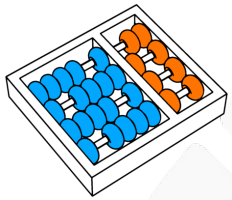


Tempo de verificação

Queremos executar o verificador em tempo polinomial:

1. O **TEMPO DO VERIFICADOR** deve ser polinomial em $|x|$ e $|y|$.
2. o **TAMANHO DO CERTIFICADO** $|y|$ deve ser polinomial em $|x|$.

Por quê?



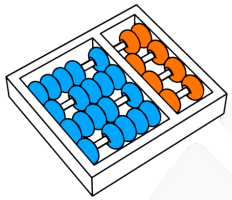
Tempo de verificação

Queremos executar o verificador em tempo polinomial:

1. O **TEMPO DO VERIFICADOR** deve ser polinomial em $|x|$ e $|y|$.
2. o **TAMANHO DO CERTIFICADO** $|y|$ deve ser polinomial em $|x|$.

Por quê?

- Queremos diferenciar as tarefas de decidir e verificar.



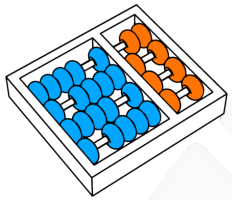
Tempo de verificação

Queremos executar o verificador em tempo polinomial:

1. O **TEMPO DO VERIFICADOR** deve ser polinomial em $|x|$ e $|y|$.
2. o **TAMANHO DO CERTIFICADO** $|y|$ deve ser polinomial em $|x|$.

Por quê?

- Queremos diferenciar as tarefas de decidir e verificar.
- Para certos problemas, **DECIDIR** uma instância é difícil.



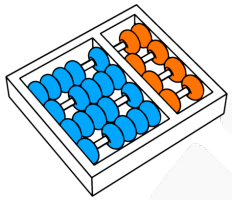
Tempo de verificação

Queremos executar o verificador em tempo polinomial:

1. O **TEMPO DO VERIFICADOR** deve ser polinomial em $|x|$ e $|y|$.
2. o **TAMANHO DO CERTIFICADO** $|y|$ deve ser polinomial em $|x|$.

Por quê?

- Queremos diferenciar as tarefas de decidir e verificar.
- Para certos problemas, **DECIDIR** uma instância é difícil.
- Mas pode ser que **VERIFICAR** uma solução seja fácil.



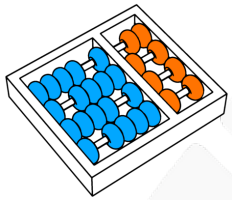
Tempo de verificação

Queremos executar o verificador em tempo polinomial:

1. O **TEMPO DO VERIFICADOR** deve ser polinomial em $|x|$ e $|y|$.
2. o **TAMANHO DO CERTIFICADO** $|y|$ deve ser polinomial em $|x|$.

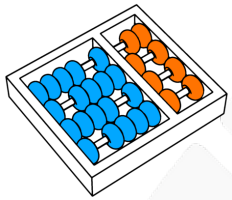
Por quê?

- Queremos diferenciar as tarefas de decidir e verificar.
- Para certos problemas, **DECIDIR** uma instância é difícil.
- Mas pode ser que **VERIFICAR** uma solução seja fácil.
- Veremos um exemplo em seguida.



Ciclo hamiltoniano

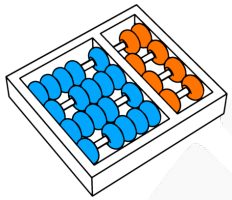
Um **CICLO HAMILTONIANO** em um grafo G é um ciclo que passa por todos os vértices.



Ciclo hamiltoniano

Um **CICLO HAMILTONIANO** em um grafo G é um ciclo que passa por todos os vértices.

- Se houver esse ciclo, dizemos que G é hamiltoniano.



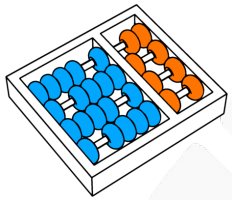
Ciclo hamiltoniano

Um **CICLO HAMILTONIANO** em um grafo G é um ciclo que passa por todos os vértices.

- Se houver esse ciclo, dizemos que G é hamiltoniano.

Problema (Ciclo hamiltoniano)

$$\mathit{HAM\text{-}CYCLE} = \{\langle G \rangle : G \textit{ é um grafo hamiltoniano } \}$$



Ciclo hamiltoniano

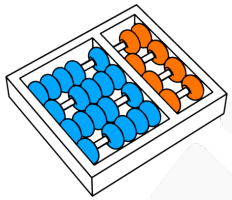
Um **CICLO HAMILTONIANO** em um grafo G é um ciclo que passa por todos os vértices.

- Se houver esse ciclo, dizemos que G é hamiltoniano.

Problema (Ciclo hamiltoniano)

$$\mathit{HAM\text{-}CYCLE} = \{\langle G \rangle : G \text{ é um grafo hamiltoniano } \}$$

Escolhas para certificado:



Ciclo hamiltoniano

Um **CICLO HAMILTONIANO** em um grafo G é um ciclo que passa por todos os vértices.

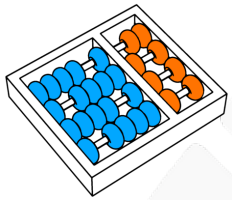
- Se houver esse ciclo, dizemos que G é hamiltoniano.

Problema (Ciclo hamiltoniano)

$$\mathit{HAM\text{-}CYCLE} = \{\langle G \rangle : G \textit{ é um grafo hamiltoniano } \}$$

Escolhas para certificado:

- Um ciclo hamiltoniano de G.



Ciclo hamiltoniano

Um **CICLO HAMILTONIANO** em um grafo G é um ciclo que passa por todos os vértices.

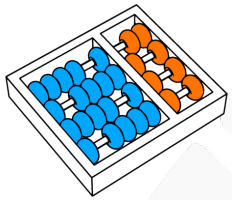
- Se houver esse ciclo, dizemos que G é hamiltoniano.

Problema (Ciclo hamiltoniano)

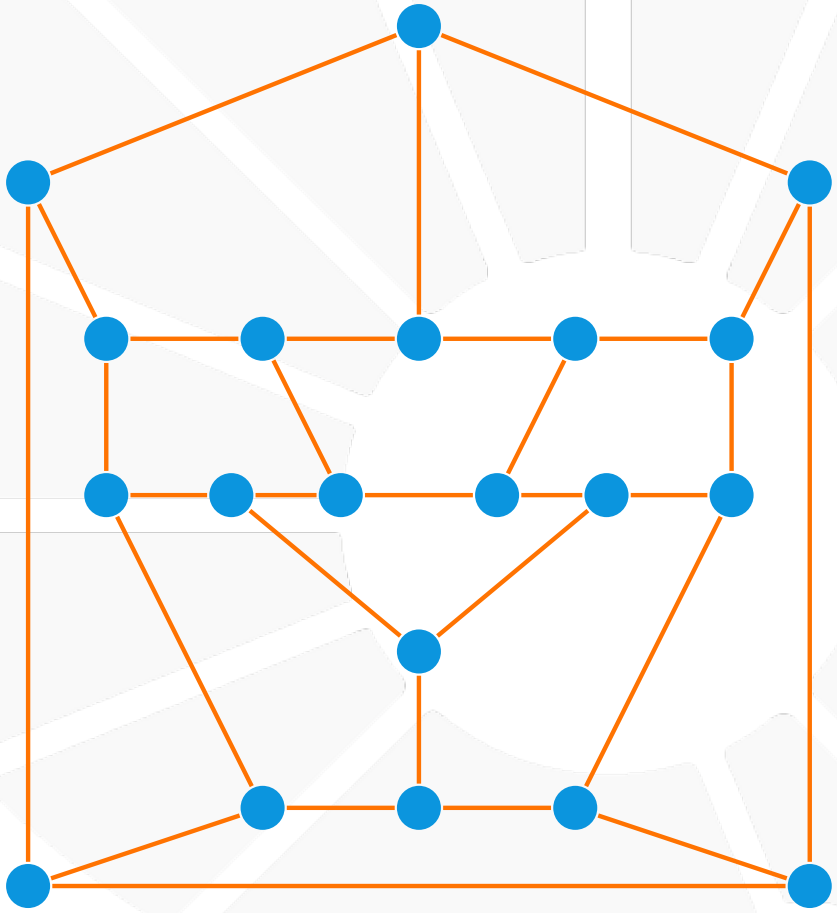
$$\mathit{HAM\text{-}CYCLE} = \{\langle G \rangle : G \textit{ é um grafo hamiltoniano } \}$$

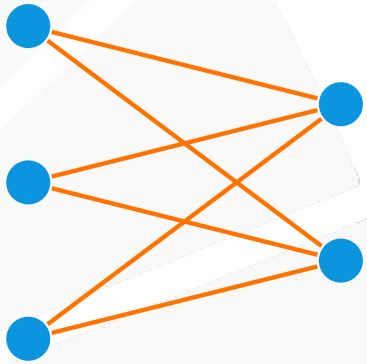
Escolhas para certificado:

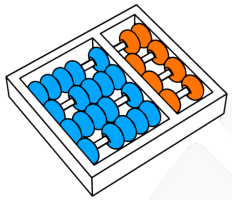
- Um ciclo hamiltoniano de G.
- Uma sequência de vértices de G.



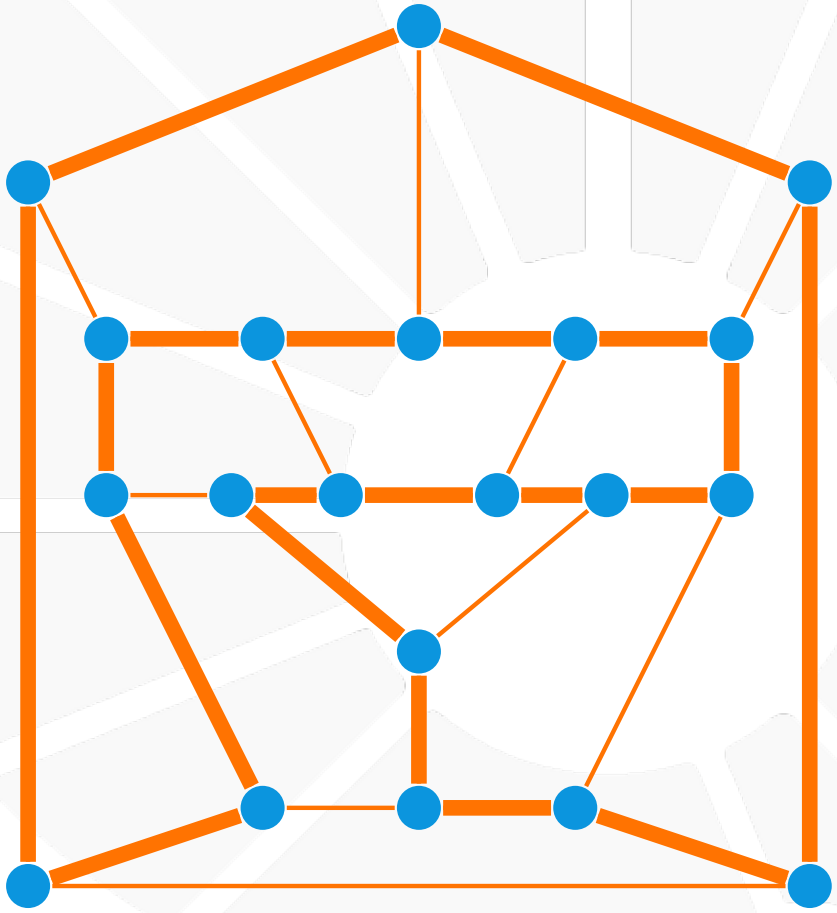
Exemplo de ciclo hamiltoniano

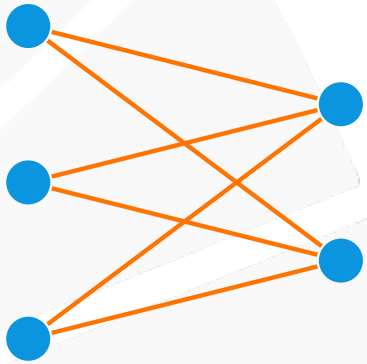




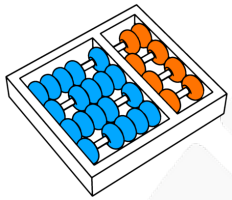


Exemplo de ciclo hamiltoniano



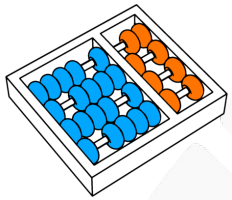


O grafo da direita não tem ciclo hamiltoniano!



Procurando um ciclo hamiltoniano

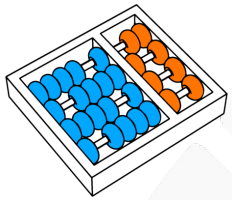
Podemos **DECIDIR** se há um ciclo hamiltoniano:



Procurando um ciclo hamiltoniano

Podemos **DECIDIR** se há um ciclo hamiltoniano:

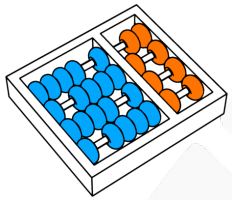
- O algoritmo trivial gasta tempo $O(V!)$.



Procurando um ciclo hamiltoniano

Podemos **DECIDIR** se há um ciclo hamiltoniano:

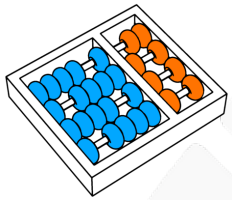
- O algoritmo trivial gasta tempo $O(V!)$.
- Os melhores algoritmos têm tempo $O(n^2 2^n)$.



Procurando um ciclo hamiltoniano

Podemos **DECIDIR** se há um ciclo hamiltoniano:

- O algoritmo trivial gasta tempo $O(V!)$.
- Os melhores algoritmos têm tempo $O(n^2 2^n)$.
- Esses algoritmos são determinísticos.

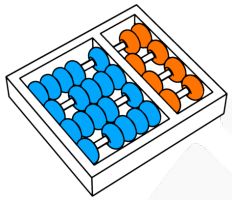


Procurando um ciclo hamiltoniano

Podemos **DECIDIR** se há um ciclo hamiltoniano:

- O algoritmo trivial gasta tempo $O(V!)$.
- Os melhores algoritmos têm tempo $O(n^2 2^n)$.
- Esses algoritmos são determinísticos.

Mas se **SORTEARMOS** um ciclo C:



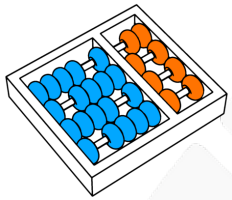
Procurando um ciclo hamiltoniano

Podemos **DECIDIR** se há um ciclo hamiltoniano:

- O algoritmo trivial gasta tempo $O(V!)$.
- Os melhores algoritmos têm tempo $O(n^2 2^n)$.
- Esses algoritmos são determinísticos.

Mas se **SORTEARMOS** um ciclo C:

- É fácil verificar se ele é hamiltoniano em tempo linear.



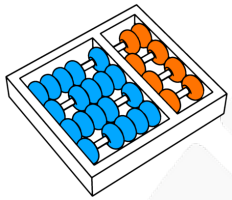
Procurando um ciclo hamiltoniano

Podemos **DECIDIR** se há um ciclo hamiltoniano:

- O algoritmo trivial gasta tempo $O(V!)$.
- Os melhores algoritmos têm tempo $O(n^2 2^n)$.
- Esses algoritmos são determinísticos.

Mas se **SORTEARMOS** um ciclo C:

- É fácil verificar se ele é hamiltoniano em tempo linear.
- Basta sortear uma sequência S de $|V|$ vértices.



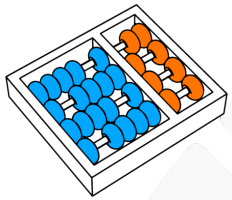
Procurando um ciclo hamiltoniano

Podemos **DECIDIR** se há um ciclo hamiltoniano:

- O algoritmo trivial gasta tempo $O(V!)$.
- Os melhores algoritmos têm tempo $O(n^2 2^n)$.
- Esses algoritmos são determinísticos.

Mas se **SORTEARMOS** um ciclo C:

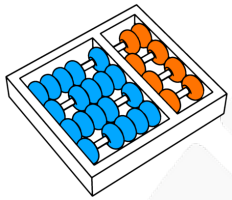
- É fácil verificar se ele é hamiltoniano em tempo linear.
- Basta sortear uma sequência S de $|V|$ vértices.
- O sorteio do ciclo é um processo não determinístico.



Verificando um ciclo

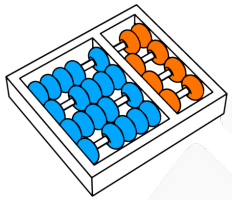
Algoritmo 6: VERIFICA-HAMCICLO($\langle G \rangle, \langle S \rangle$)

```
1  se S não contém todos os vértices de G
2      devolva NAO
3  n ← |S|
4  S[n + 1] ← S[1]
5  para cada i = 1, 2, ..., |S|
6      u ← S[i]
7      v ← S[i + 1]
8      se (u, v) não é aresta de G
9          devolva NAO
10 devolva SIM
```



Linguagem verificada

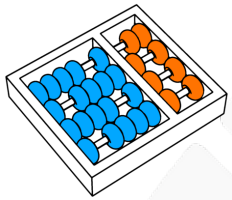
Um algoritmo V **VERIFICA** uma linguagem L se:



Linguagem verificada

Um algoritmo V **VERIFICA** uma linguagem L se:

$$L = \{x \in \{0,1\}^* \ : \ \text{existe } y \in \{0,1\}^* \text{ tal que } V(x,y) = 1\}$$

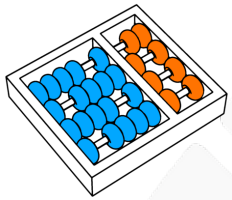


Linguagem verificada

Um algoritmo V **VERIFICA** uma linguagem L se:

$$L = \{x \in \{0,1\}^* \ : \ \text{existe } y \in \{0,1\}^* \text{ tal que } V(x,y) = 1\}$$

Em outras palavras:



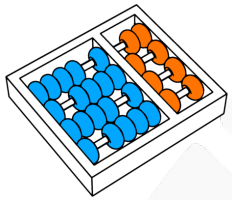
Linguagem verificada

Um algoritmo V **VERIFICA** uma linguagem L se:

$$L = \{x \in \{0,1\}^* \ : \ \text{existe } y \in \{0,1\}^* \text{ tal que } V(x,y) = 1\}$$

Em outras palavras:

1. Se $x \in L$, então **EXISTE** certificado y tal que $V(x,y) = 1$.



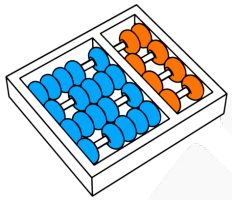
Linguagem verificada

Um algoritmo V **VERIFICA** uma linguagem L se:

$$L = \{x \in \{0,1\}^* \ : \ \text{existe } y \in \{0,1\}^* \text{ tal que } V(x,y) = 1\}$$

Em outras palavras:

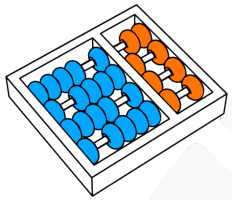
1. Se $x \in L$, então **EXISTE** certificado y tal que $V(x,y) = 1$.
2. Se $x \notin L$, então **NÃO EXISTE** certificado y tal que $V(x,y) = 1$.



A classe NP

Definição

*A **CLASSE** NP é o conjunto de linguagens $L \subseteq \{0,1\}^*$ para as quais existe algoritmo V que verifica L em tempo polinomial.*

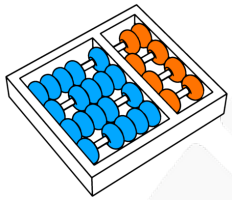


A classe NP

Definição

*A **CLASSE** NP é o conjunto de linguagens $L \subseteq \{0,1\}^*$ para as quais existe algoritmo V que verifica L em tempo polinomial.*

Em outras palavras, se $L \in$ NP, então:



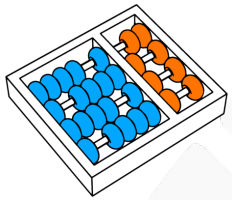
A classe NP

Definição

*A **CLASSE** NP é o conjunto de linguagens $L \subseteq \{0,1\}^*$ para as quais existe algoritmo V que verifica L em tempo polinomial.*

Em outras palavras, se $L \in$ NP, então:

1. Existe algoritmo $V(x, y)$ que verifica L.



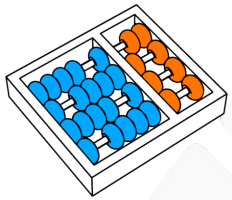
A classe NP

Definição

*A **CLASSE** NP é o conjunto de linguagens $L \subseteq \{0,1\}^*$ para as quais existe algoritmo V que verifica L em tempo polinomial.*

Em outras palavras, se $L \in$ NP, então:

1. Existe algoritmo $V(x, y)$ que verifica L.
2. Esse algoritmo executa em tempo polinomial em $|x|$ e $|y|$.



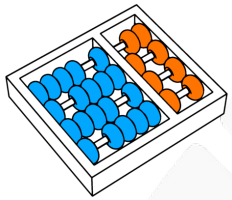
A classe NP

Definição

*A **CLASSE** NP é o conjunto de linguagens $L \subseteq \{0,1\}^*$ para as quais existe algoritmo V que verifica L em tempo polinomial.*

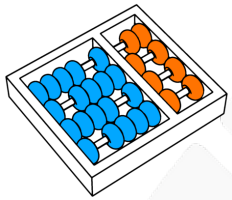
Em outras palavras, se $L \in$ NP, então:

1. Existe algoritmo $V(x, y)$ que verifica L.
2. Esse algoritmo executa em tempo polinomial em $|x|$ e $|y|$.
3. Para cada $x \in L$, existe certificado y polinomial em $|x|$.



Determinando se problema é NP

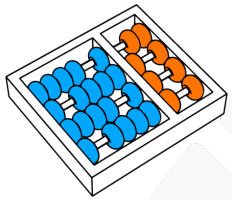
Dado problema L, devemos seguir esses passos:



Determinando se problema é NP

Dado problema L, devemos seguir esses passos:

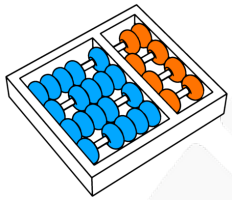
1. Identifique um **CERTIFICADO** de tamanho polinomial para L.



Determinando se problema é NP

Dado problema L, devemos seguir esses passos:

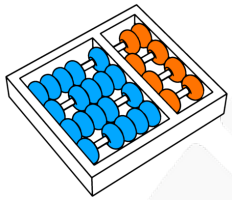
1. Identifique um **CERTIFICADO** de tamanho polinomial para L.
2. Construa um **ALGORITMO VERIFICADOR** $V(x, y)$ polinomial.



Determinando se problema é NP

Dado problema L, devemos seguir esses passos:

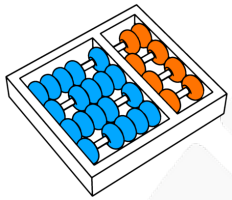
1. Identifique um **CERTIFICADO** de tamanho polinomial para L.
2. Construa um **ALGORITMO VERIFICADOR** $V(x, y)$ polinomial.
3. Demonstre que:



Determinando se problema é NP

Dado problema L, devemos seguir esses passos:

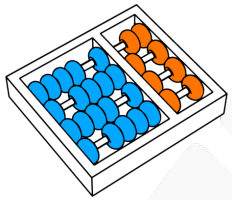
1. Identifique um **CERTIFICADO** de tamanho polinomial para L.
2. Construa um **ALGORITMO VERIFICADOR** $V(x, y)$ polinomial.
3. Demonstre que:
   - Se $x \in L$, então **existe** y tal que $V(x, y) = 1$.



Determinando se problema é NP

Dado problema L, devemos seguir esses passos:

1. Identifique um **CERTIFICADO** de tamanho polinomial para L.
2. Construa um **ALGORITMO VERIFICADOR** $V(x, y)$ polinomial.
3. Demonstre que:
   - Se $x \in L$, então **existe** y tal que $V(x, y) = 1$.
   - Se $x \notin L$, então para **qualquer** y, vale $V(x, y) = 0$.

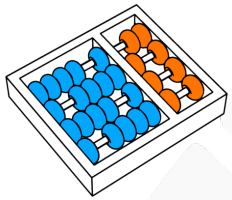


Determinando se problema é NP

Dado problema L, devemos seguir esses passos:

1. Identifique um **CERTIFICADO** de tamanho polinomial para L.
2. Construa um **ALGORITMO VERIFICADOR** $V(x, y)$ polinomial.
3. Demonstre que:
   - Se $x \in L$, então **existe** y tal que $V(x, y) = 1$.
   - Se $x \notin L$, então para **qualquer** y, vale $V(x, y) = 0$.

Exemplos:



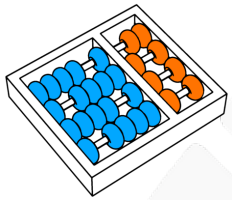
Determinando se problema é NP

Dado problema L, devemos seguir esses passos:

1. Identifique um **CERTIFICADO** de tamanho polinomial para L.
2. Construa um **ALGORITMO VERIFICADOR** $V(x, y)$ polinomial.
3. Demonstre que:
   - Se $x \in L$, então **existe** y tal que $V(x, y) = 1$.
   - Se $x \notin L$, então para **qualquer** y, vale $V(x, y) = 0$.

Exemplos:

- VERIFICA-PATH é um verificador para PATH.



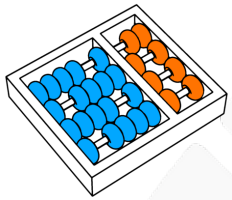
Determinando se problema é NP

Dado problema L, devemos seguir esses passos:

1. Identifique um **CERTIFICADO** de tamanho polinomial para L.
2. Construa um **ALGORITMO VERIFICADOR** $V(x, y)$ polinomial.
3. Demonstre que:
   - Se $x \in L$, então **existe** y tal que $V(x, y) = 1$.
   - Se $x \notin L$, então para **qualquer** y, vale $V(x, y) = 0$.

Exemplos:

- VERIFICA-PATH é um verificador para PATH.
- VERIFICA-HAMCICLO é um verificador para HAM-CYCLE.



Determinando se problema é NP

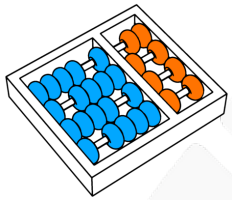
Dado problema L, devemos seguir esses passos:

1. Identifique um **CERTIFICADO** de tamanho polinomial para L.
2. Construa um **ALGORITMO VERIFICADOR** $V(x, y)$ polinomial.
3. Demonstre que:
   - Se $x \in L$, então **existe** y tal que $V(x, y) = 1$.
   - Se $x \notin L$, então para **qualquer** y, vale $V(x, y) = 0$.

Exemplos:

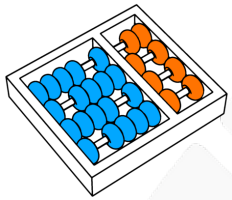
- VERIFICA-PATH é um verificador para PATH.
- VERIFICA-HAMCICLO é um verificador para HAM-CYCLE.

Portanto, PATH e HAM-CYCLE estão em NP.



NP contém P

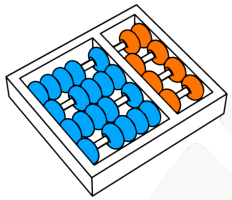
Seja $L \in \mathrm{P}$:



NP contém P

Seja $L \in$ P:

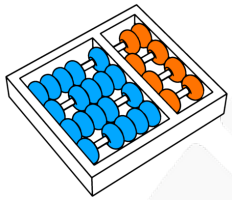
- Exite algoritmo A que decide L.



NP contém P

Seja $L \in \mathrm{P}$:

- Exite algoritmo A que decide L.
- Vamos construir um verificador para L.



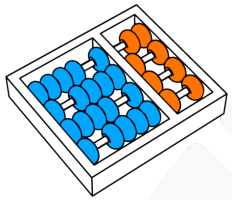
NP contém P

Seja $L \in$ P:

- Exite algoritmo A que decide L.
- Vamos construir um verificador para L.

Algoritmo 10: VERIFICA-L(x, y):

1 **devolva** $A(x)$



NP contém P

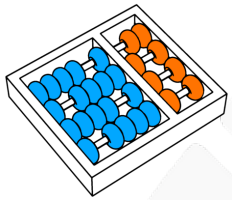
Seja $L \in \mathrm{P}$:

- Exite algoritmo A que decide L.
- Vamos construir um verificador para L.

Algoritmo 11: VERIFICA-L(x, y):

```
1 devolva A(x)
```

- O verificador só precisa ignorar o argumento y.



NP contém P

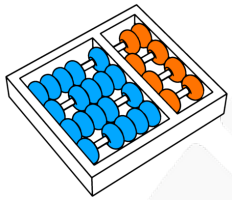
Seja $L \in \mathrm{P}$:

- Exite algoritmo A que decide L.
- Vamos construir um verificador para L.

Algoritmo 12: VERIFICA-L(x, y):

```
1 devolva A(x)
```

- O verificador só precisa ignorar o argumento y.
- Podemos escolher qualquer certificado (e.g. $y = \varepsilon$).



NP contém P

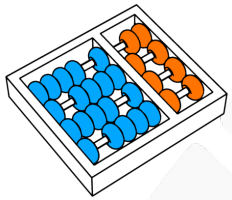
Seja $L \in \mathrm{P}$:

- Exite algoritmo A que decide L.
- Vamos construir um verificador para L.

Algoritmo 13: VERIFICA-L(x, y):

```
1 devolva A(x)
```

- O verificador só precisa ignorar o argumento y.
- Podemos escolher qualquer certificado (e.g. $y = \varepsilon$).
- Analisamos:



NP contém P

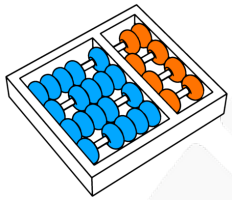
Seja $L \in \mathrm{P}$:

- Exite algoritmo A que decide L.
- Vamos construir um verificador para L.

Algoritmo 14: $\textsc{Verifica-L}(x, y)$:

```
1 devolva A(x)
```

- O verificador só precisa ignorar o argumento y.
- Podemos escolher qualquer certificado (e.g. $y = \varepsilon$).
- Analisamos:
  - Se $x \in L$, então $A(x) = 1$, daí $\textsc{Verifica-L}(x, \varepsilon) = 1$.



NP contém P

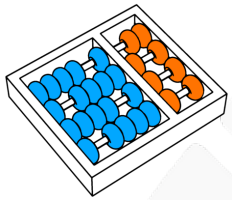
Seja $L \in \mathrm{P}$:

- Exite algoritmo A que decide L.
- Vamos construir um verificador para L.

Algoritmo 15: $\textsc{Verifica-L}(x, y)$:

1 **devolva** $A(x)$

- O verificador só precisa ignorar o argumento y.
- Podemos escolher qualquer certificado (e.g. $y = \varepsilon$).
- Analisamos:
  - Se $x \in L$, então $A(x) = 1$, daí $\textsc{Verifica-L}(x, \varepsilon) = 1$.
  - Se $x \notin L$, então $A(x) = 0$, daí $\textsc{Verifica-L}(x, y) = 0$ para todo y.



NP contém P

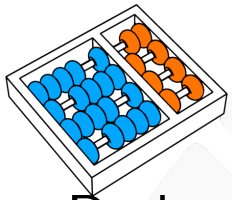
Seja $L \in \mathrm{P}$:

- Exite algoritmo A que decide L.
- Vamos construir um verificador para L.

Algoritmo 16: $\textsc{Verifica-L}(x, y)$:

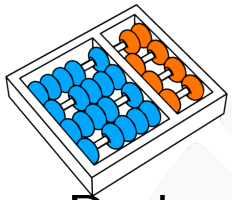
1 **devolva** $A(x)$

- O verificador só precisa ignorar o argumento y.
- Podemos escolher qualquer certificado (e.g. $y = \varepsilon$).
- Analisamos:
  - Se $x \in L$, então $A(x) = 1$, daí $\textsc{Verifica-L}(x, \varepsilon) = 1$.
  - Se $x \notin L$, então $A(x) = 0$, daí $\textsc{Verifica-L}(x, y) = 0$ para todo y.
- Assim $L \in \mathrm{NP}$ e concluímos que $\mathrm{P} \subseteq \mathrm{NP}$.



O problema complementar

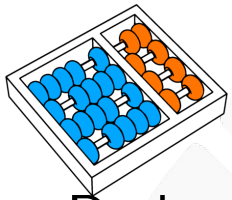
Dada uma linguagem L, o seu complementar é $\overline{L} = \{0,1\}^* \setminus L$.



O problema complementar

Dada uma linguagem L, o seu complementar é $\overline{L} = \{0,1\}^* \setminus L$.

- $\overline{L}$ é o conjunto de instâncias de L com resposta NAO.



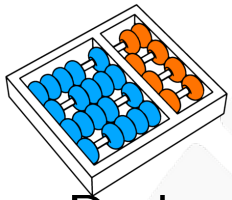
O problema complementar

Dada uma linguagem L, o seu complementar é $\overline{L} = \{0,1\}^* \setminus L$.

- $\overline{L}$ é o conjunto de instâncias de L com resposta NAO.
- Exemplo:

$$\text{HAM-CYCLE} = \{\langle G \rangle : G \text{ possui ciclo hamiltoniano}\}$$

$$\overline{\text{HAM-CYCLE}} = \{\langle G \rangle : G \textbf{ NÃO} \text{ possui ciclo hamiltoniano}\}$$



O problema complementar

Dada uma linguagem L, o seu complementar é $\overline{L} = \{0,1\}^* \setminus L$.

- $\overline{L}$ é o conjunto de instâncias de L com resposta NAO.
- Exemplo:

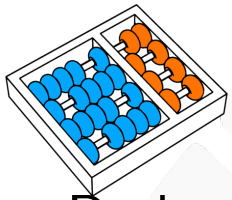
$$\text{HAM-CYCLE} = \{\langle G \rangle : G \text{ possui ciclo hamiltoniano}\}$$

$$\overline{\text{HAM-CYCLE}} = \{\langle G \rangle : G \textbf{ NÃO} \text{ possui ciclo hamiltoniano}\}$$

Definição

A classe complementar de NP *é o conjunto de linguagens*

$$\text{co-NP} = \{L \subseteq \Sigma^* : \overline{L} \in \text{NP}\}$$



O problema complementar

Dada uma linguagem L, o seu complementar é $\overline{L} = \{0,1\}^* \setminus L$.

- $\overline{L}$ é o conjunto de instâncias de L com resposta NAO.
- Exemplo:

$$\text{HAM-CYCLE} = \{\langle G \rangle : G \text{ possui ciclo hamiltoniano}\}$$

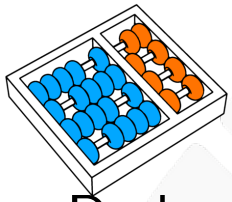
$$\overline{\text{HAM-CYCLE}} = \{\langle G \rangle : G \textbf{ NÃO} \text{ possui ciclo hamiltoniano}\}$$

Definição

A classe complementar de NP *é o conjunto de linguagens*

$$\text{co-NP} = \{L \subseteq \Sigma^* : \overline{L} \in \text{NP}\}$$

Exemplos:



O problema complementar

Dada uma linguagem L, o seu complementar é $\overline{L} = \{0,1\}^* \setminus L$.

- $\overline{L}$ é o conjunto de instâncias de L com resposta NAO.
- Exemplo:

$$\text{HAM-CYCLE} = \{\langle G \rangle : G \text{ possui ciclo hamiltoniano}\}$$

$$\overline{\text{HAM-CYCLE}} = \{\langle G \rangle : G \textbf{ NÃO} \text{ possui ciclo hamiltoniano}\}$$

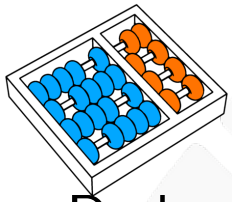
Definição

A classe complementar de NP *é o conjunto de linguagens*

$$\text{co-NP} = \{L \subseteq \Sigma^* : \overline{L} \in \text{NP}\}$$

Exemplos:

- $\overline{\text{HAM-CYCLE}} \in$ co-NP



O problema complementar

Dada uma linguagem L, o seu complementar é $\overline{L} = \{0,1\}^* \setminus L$.

- $\overline{L}$ é o conjunto de instâncias de L com resposta NAO.
- Exemplo:

$$\text{HAM-CYCLE} = \{\langle G \rangle : G \text{ possui ciclo hamiltoniano}\}$$

$$\overline{\text{HAM-CYCLE}} = \{\langle G \rangle : G \textbf{ NÃO} \text{ possui ciclo hamiltoniano}\}$$

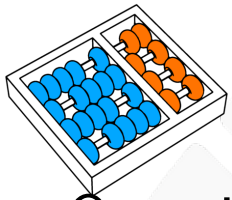
Definição

A classe complementar de NP *é o conjunto de linguagens*

$$\text{co-NP} = \{L \subseteq \Sigma^* : \overline{L} \in \text{NP}\}$$

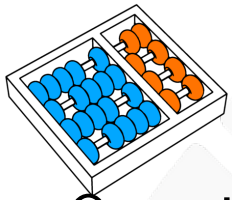
Exemplos:

- $\overline{\text{HAM-CYCLE}} \in$ co-NP
- P $\subseteq$ co-NP



Outro exemplo: tautologia

O conjunto co-NP é contém as linguagens que possuem um certificado curto para a resposta NAO.

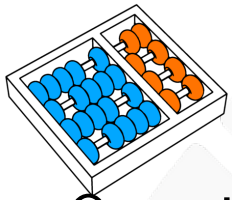


Outro exemplo: tautologia

O conjunto co-NP é contém as linguagens que possuem um certificado curto para a resposta NAO.

Problema (Tautologia)

Dada uma fórmula booleana com um conjunto de n variáveis e operadores $\wedge$, $\vee$, $\neg$ etc, ela é verdadeira para toda atribuição de variáveis?



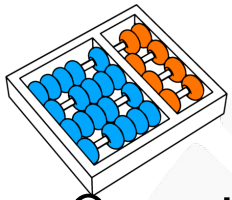
Outro exemplo: tautologia

O conjunto co-NP é contém as linguagens que possuem um certificado curto para a resposta NAO.

Problema (Tautologia)

Dada uma fórmula booleana com um conjunto de n variáveis e operadores $\wedge$, $\vee$, $\neg$ etc, ela é verdadeira para toda atribuição de variáveis?

- $p \vee \neg p$, $((z \wedge y) \vee \neg x \vee (x \wedge \neg y)) \vee (\neg z \wedge y)$ são tautologias.



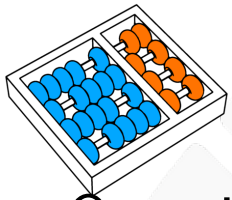
Outro exemplo: tautologia

O conjunto co-NP é contém as linguagens que possuem um certificado curto para a resposta NAO.

Problema (Tautologia)

Dada uma fórmula booleana com um conjunto de n variáveis e operadores $\wedge$, $\vee$, $\neg$ etc, ela é verdadeira para toda atribuição de variáveis?

- $p \vee \neg p$, $((z \wedge y) \vee \neg x \vee (x \wedge \neg y)) \vee (\neg z \wedge y)$ são tautologias.
- p, $(\neg y \vee x) \wedge (y \vee \neg x)$ **NÃO** são tautologias.



Outro exemplo: tautologia

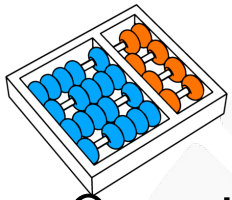
O conjunto co-NP é contém as linguagens que possuem um certificado curto para a resposta NAO.

Problema (Tautologia)

Dada uma fórmula booleana com um conjunto de n variáveis e operadores $\wedge$, $\vee$, $\neg$ etc, ela é verdadeira para toda atribuição de variáveis?

- $p \vee \neg p$, $((z \wedge y) \vee \neg x \vee (x \wedge \neg y)) \vee (\neg z \wedge y)$ são tautologias.
- p, $(\neg y \vee x) \wedge (y \vee \neg x)$ **NÃO** são tautologias.

- Certificado curto para instâncias NAO: $x = 0, y = 1$.



Outro exemplo: tautologia

O conjunto co-NP é contém as linguagens que possuem um certificado curto para a resposta NAO.

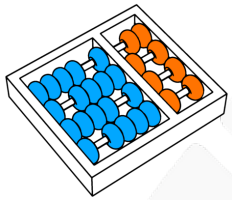
Problema (Tautologia)

Dada uma fórmula booleana com um conjunto de n variáveis e operadores $\wedge$, $\vee$, $\neg$ etc, ela é verdadeira para toda atribuição de variáveis?

- $p \vee \neg p$, $((z \wedge y) \vee \neg x \vee (x \wedge \neg y)) \vee (\neg z \wedge y)$ são tautologias.
- p, $(\neg y \vee x) \wedge (y \vee \neg x)$ **NÃO** são tautologias.

- Certificado curto para instâncias NAO: $x = 0, y = 1$.
- Não conhecemos certificado curto para instâncias SIM.

NP-completude

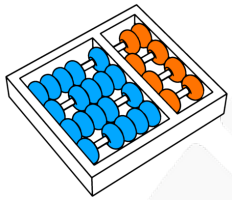


Resumo das classes até agora

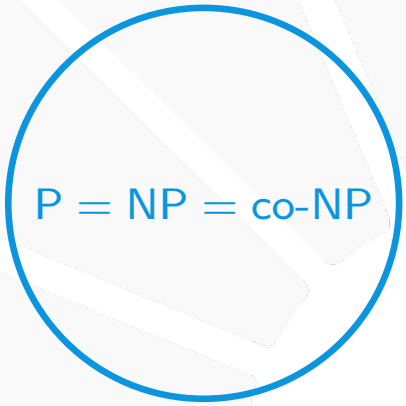
$$\mathrm{P} = \{L \subseteq \Sigma^* : \text{há algoritmo que } \textbf{DECIDE } L \text{ em tempo polinomial}\}$$

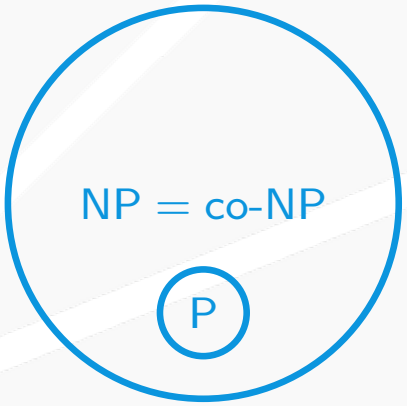
$$\mathrm{NP} = \{L \subseteq \Sigma^* : \text{há algoritmo que } \textbf{VERIFICA } L \text{ em tempo polinomial}$$

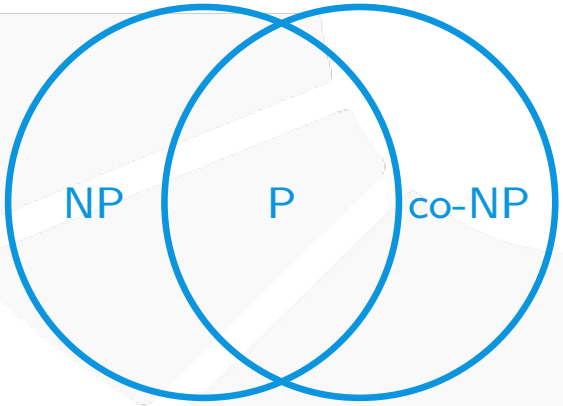
$$\text{co-NP} = \{L \subseteq \Sigma^* : \text{há algoritmo que } \textbf{VERIFICA } \overline{L} \text{ em tempo polinomial}$$

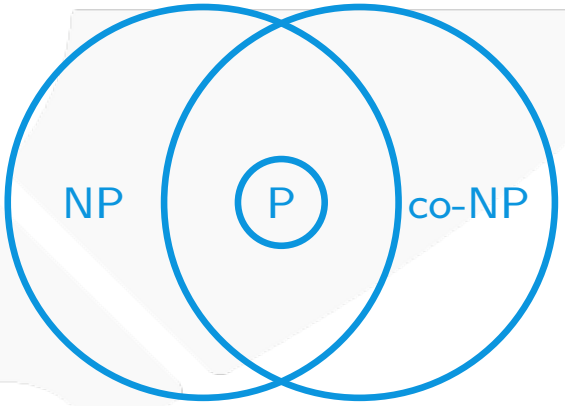


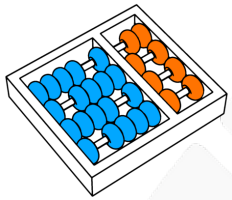
Possíveis configurações dessas classes





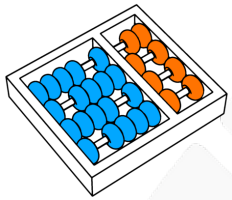






Refletindo sobre o que vimos

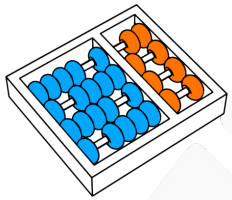
Vimos alguns exemplos de problemas que:



Refletindo sobre o que vimos

Vimos alguns exemplos de problemas que:

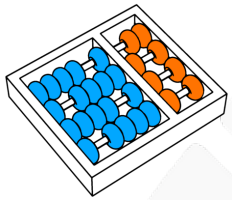
- Podemos **DECIDIR** em tempo polinomial: PATH.



Refletindo sobre o que vimos

Vimos alguns exemplos de problemas que:

- Podemos **DECIDIR** em tempo polinomial: PATH.
- Só sabemos **VERIFICAR** em tempo polinomial: HAM-CYCLE.

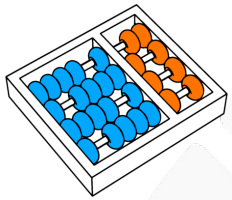


Refletindo sobre o que vimos

Vimos alguns exemplos de problemas que:

- Podemos **DECIDIR** em tempo polinomial: PATH.
- Só sabemos **VERIFICAR** em tempo polinomial: HAM-CYCLE.

Por que não conhecemos algoritmo rápido para HAM-CYCLE?



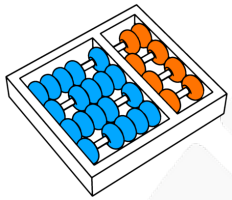
Refletindo sobre o que vimos

Vimos alguns exemplos de problemas que:

- Podemos **DECIDIR** em tempo polinomial: PATH.
- Só sabemos **VERIFICAR** em tempo polinomial: HAM-CYCLE.

Por que não conhecemos algoritmo rápido para HAM-CYCLE?

- Será que HAM-CYCLE é mais difícil do que PATH?



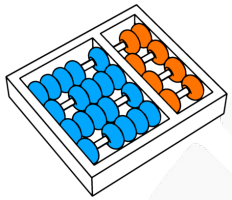
Refletindo sobre o que vimos

Vimos alguns exemplos de problemas que:

- Podemos **DECIDIR** em tempo polinomial: PATH.
- Só sabemos **VERIFICAR** em tempo polinomial: HAM-CYCLE.

Por que não conhecemos algoritmo rápido para HAM-CYCLE?

- Será que HAM-CYCLE é mais difícil do que PATH?
- Será que HAM-CYCLE é mais difícil que qualquer um em P?



Refletindo sobre o que vimos

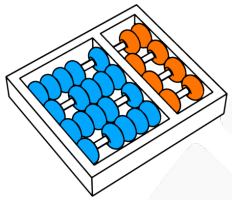
Vimos alguns exemplos de problemas que:

- Podemos **DECIDIR** em tempo polinomial: PATH.
- Só sabemos **VERIFICAR** em tempo polinomial: HAM-CYCLE.

Por que não conhecemos algoritmo rápido para HAM-CYCLE?

- Será que HAM-CYCLE é mais difícil do que PATH?
- Será que HAM-CYCLE é mais difícil que qualquer um em P?

Vamos mostrar que HAM-CYCLE é NP-**DIFÍCIL**:



Refletindo sobre o que vimos

Vimos alguns exemplos de problemas que:

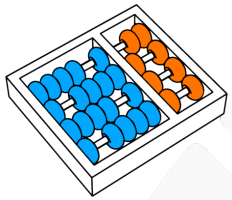
- Podemos **DECIDIR** em tempo polinomial: PATH.
- Só sabemos **VERIFICAR** em tempo polinomial: HAM-CYCLE.

Por que não conhecemos algoritmo rápido para HAM-CYCLE?

- Será que HAM-CYCLE é mais difícil do que PATH?
- Será que HAM-CYCLE é mais difícil que qualquer um em P?

Vamos mostrar que HAM-CYCLE é NP-**DIFÍCIL**:

- Não sabemos se é mais difícil do que algum problema P.



Refletindo sobre o que vimos

Vimos alguns exemplos de problemas que:

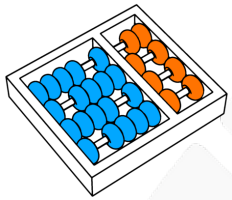
- Podemos **DECIDIR** em tempo polinomial: PATH.
- Só sabemos **VERIFICAR** em tempo polinomial: HAM-CYCLE.

Por que não conhecemos algoritmo rápido para HAM-CYCLE?

- Será que HAM-CYCLE é mais difícil do que PATH?
- Será que HAM-CYCLE é mais difícil que qualquer um em P?

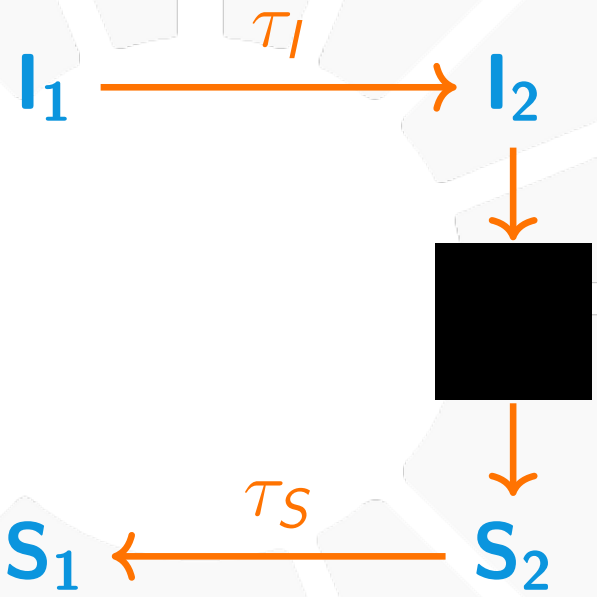
Vamos mostrar que HAM-CYCLE é NP-**DIFÍCIL**:

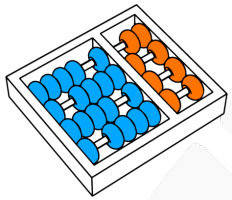
- Não sabemos se é mais difícil do que algum problema P.
- Mas sabemos que é **TÃO DIFÍCIL QUANTO** qualquer problema NP.



Como comparar problemas?

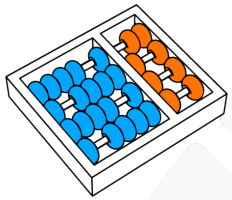
A ferramenta adequada para comparar problemas são as reduções





Reduções de Karp

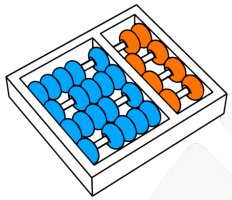
Estamos interessados em reduções em que:



Reduções de Karp

Estamos interessados em reduções em que:

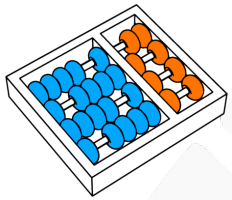
1. A transformação de entrada τ_I leva tempo polinomial.



Reduções de Karp

Estamos interessados em reduções em que:

1. A transformação de entrada τ_I leva tempo polinomial.
2. **NÃO** há transformação de saída τ_S.



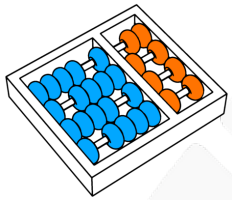
Reduções de Karp

Estamos interessados em reduções em que:

1. A transformação de entrada τ_I leva tempo polinomial.
2. **NÃO** há transformação de saída τ_S.

Definição (Redução de Karp)

*A linguagem L_1 é **REDUTÍVEL EM TEMPO POLINOMIAL** para L_2 se:*



Reduções de Karp

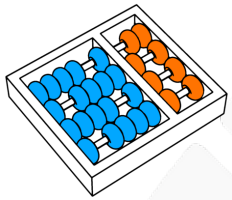
Estamos interessados em reduções em que:

1. A transformação de entrada τ_I leva tempo polinomial.
2. **NÃO** há transformação de saída τ_S.

Definição (Redução de Karp)

*A linguagem L_1 é **REDUTÍVEL EM TEMPO POLINOMIAL** para L_2 se:*

- *Existe algoritmo $f : \{0,1\}^* \to \{0,1\}^*$,*



Reduções de Karp

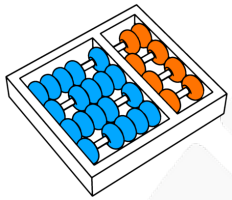
Estamos interessados em reduções em que:

1. A transformação de entrada τ_I leva tempo polinomial.
2. **NÃO** há transformação de saída τ_S.

Definição (Redução de Karp)

*A linguagem L_1 é **REDUTÍVEL EM TEMPO POLINOMIAL** para L_2 se:*

- *Existe algoritmo $f : \{0,1\}^* \to \{0,1\}^*$,*
- *$f(x)$ leva tempo polinomial em $|x|$,*



Reduções de Karp

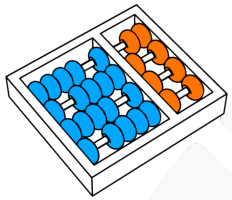
Estamos interessados em reduções em que:

1. A transformação de entrada τ_I leva tempo polinomial.
2. **NÃO** há transformação de saída τ_S.

Definição (Redução de Karp)

*A linguagem L_1 é **REDUTÍVEL EM TEMPO POLINOMIAL** para L_2 se:*

- *Existe algoritmo $f : \{0,1\}^* \to \{0,1\}^*$,*
- *$f(x)$ leva tempo polinomial em $|x|$,*
- *$x \in L_1$ se e somente se $f(x) \in L_2$.*



Reduções de Karp

Estamos interessados em reduções em que:

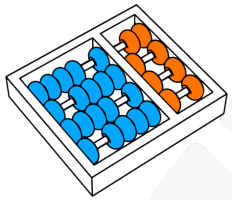
1. A transformação de entrada τ_I leva tempo polinomial.
2. **NÃO** há transformação de saída τ_S.

Definição (Redução de Karp)

*A linguagem L_1 é **REDUTÍVEL EM TEMPO POLINOMIAL** para L_2 se:*

- *Existe algoritmo $f : \{0,1\}^* \to \{0,1\}^*$,*
- *$f(x)$ leva tempo polinomial em $|x|$,*
- *$x \in L_1$ se e somente se $f(x) \in L_2$.*

- Escrevemos $L_1 \preccurlyeq_p L_2$.



Reduções de Karp

Estamos interessados em reduções em que:

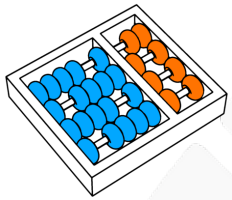
1. A transformação de entrada τ_I leva tempo polinomial.
2. **NÃO** há transformação de saída τ_S.

Definição (Redução de Karp)

*A linguagem L_1 é **REDUTÍVEL EM TEMPO POLINOMIAL** para L_2 se:*

- *Existe algoritmo $f : \{0,1\}^* \to \{0,1\}^*$,*
- *$f(x)$ leva tempo polinomial em $|x|$,*
- *$x \in L_1$ se e somente se $f(x) \in L_2$.*

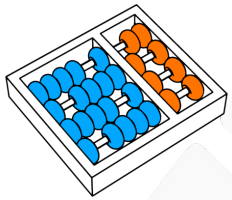
- Escrevemos $L_1 \preccurlyeq_p L_2$.
- Dizemos que f reduz L_1 para L_2.



Reduções de tempo polinomial

Teorema

Considere linguagens $L_1, L_2 \subseteq \{0,1\}^*$ *tais que* $L_1 \preccurlyeq_p L_2$.
Se $L_2 \in \mathrm{P}$, *então* $L_1 \in \mathrm{P}$.

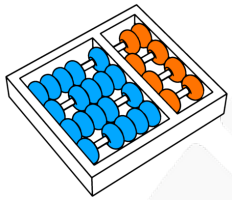


Reduções de tempo polinomial

Teorema

Considere linguagens $L_1, L_2 \subseteq \{0,1\}^*$ *tais que* $L_1 \preccurlyeq_p L_2$.
Se $L_2 \in \mathrm{P}$, *então* $L_1 \in \mathrm{P}$.

Demonstração:



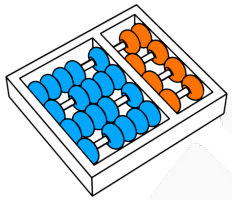
Reduções de tempo polinomial

Teorema

Considere linguagens $L_1, L_2 \subseteq \{0,1\}^*$ *tais que* $L_1 \preccurlyeq_p L_2$.
Se $L_2 \in \mathrm{P}$, *então* $L_1 \in \mathrm{P}$.

Demonstração:

- Suponha que $L_2 \in \mathrm{P}$.



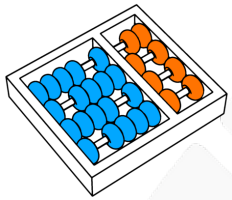
Reduções de tempo polinomial

Teorema

Considere linguagens $L_1, L_2 \subseteq \{0,1\}^*$ *tais que* $L_1 \preccurlyeq_p L_2$.
Se $L_2 \in \mathrm{P}$, *então* $L_1 \in \mathrm{P}$.

Demonstração:

- Suponha que $L_2 \in \mathrm{P}$.
- Seja A_2 um algoritmo que decide L_2 em tempo polinomial.



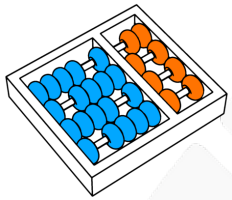
Reduções de tempo polinomial

Teorema

Considere linguagens $L_1, L_2 \subseteq \{0,1\}^*$ *tais que* $L_1 \preccurlyeq_p L_2$.
Se $L_2 \in \mathrm{P}$, *então* $L_1 \in \mathrm{P}$.

Demonstração:

- Suponha que $L_2 \in \mathrm{P}$.
- Seja A_2 um algoritmo que decide L_2 em tempo polinomial.
- Seja f um algoritmo polinomial que reduz L_1 a L_2.



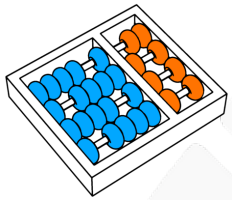
Reduções de tempo polinomial

Teorema

Considere linguagens $L_1, L_2 \subseteq \{0,1\}^*$ *tais que* $L_1 \preccurlyeq_p L_2$.
Se $L_2 \in \mathrm{P}$, *então* $L_1 \in \mathrm{P}$.

Demonstração:

- Suponha que $L_2 \in \mathrm{P}$.
- Seja A_2 um algoritmo que decide L_2 em tempo polinomial.
- Seja f um algoritmo polinomial que reduz L_1 a L_2.
- Crie um algoritmo A_1 fazendo $A_1(x) = A_2(f(x))$.



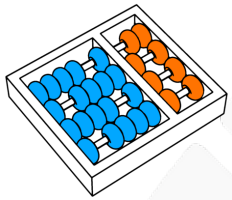
Reduções de tempo polinomial

Teorema

Considere linguagens $L_1, L_2 \subseteq \{0,1\}^*$ *tais que* $L_1 \preccurlyeq_p L_2$.
Se $L_2 \in \mathrm{P}$, *então* $L_1 \in \mathrm{P}$.

Demonstração:

- Suponha que $L_2 \in \mathrm{P}$.
- Seja A_2 um algoritmo que decide L_2 em tempo polinomial.
- Seja f um algoritmo polinomial que reduz L_1 a L_2.
- Crie um algoritmo A_1 fazendo $A_1(x) = A_2(f(x))$.
- Já que f é uma redução, obtemos:



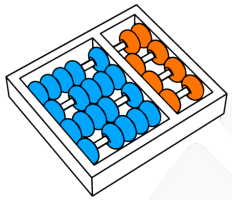
Reduções de tempo polinomial

Teorema

Considere linguagens $L_1, L_2 \subseteq \{0,1\}^*$ *tais que* $L_1 \preccurlyeq_p L_2$.
Se $L_2 \in \mathrm{P}$, *então* $L_1 \in \mathrm{P}$.

Demonstração:

- Suponha que $L_2 \in \mathrm{P}$.
- Seja A_2 um algoritmo que decide L_2 em tempo polinomial.
- Seja f um algoritmo polinomial que reduz L_1 a L_2.
- Crie um algoritmo A_1 fazendo $A_1(x) = A_2(f(x))$.
- Já que f é uma redução, obtemos:
  - Se $x \in L_1$, então $f(x) \in L_2$ e $A_2(f(x)) = 1$, daí $A_1(x) = 1$.



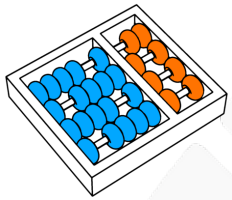
Reduções de tempo polinomial

Teorema

Considere linguagens $L_1, L_2 \subseteq \{0,1\}^*$ *tais que* $L_1 \preccurlyeq_p L_2$.
Se $L_2 \in \mathrm{P}$, *então* $L_1 \in \mathrm{P}$.

Demonstração:

- Suponha que $L_2 \in \mathrm{P}$.
- Seja A_2 um algoritmo que decide L_2 em tempo polinomial.
- Seja f um algoritmo polinomial que reduz L_1 a L_2.
- Crie um algoritmo A_1 fazendo $A_1(x) = A_2(f(x))$.
- Já que f é uma redução, obtemos:
  - Se $x \in L_1$, então $f(x) \in L_2$ e $A_2(f(x)) = 1$, daí $A_1(x) = 1$.
  - Se $x \notin L_1$, então $f(x) \notin L_2$ e $A_2(f(x)) = 0$, daí $A_1(x) = 0$.



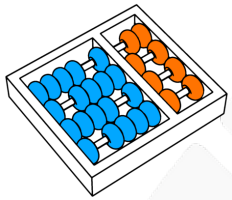
Reduções de tempo polinomial

Teorema

Considere linguagens $L_1, L_2 \subseteq \{0,1\}^*$ *tais que* $L_1 \preccurlyeq_p L_2$.
Se $L_2 \in \mathrm{P}$, *então* $L_1 \in \mathrm{P}$.

Demonstração:

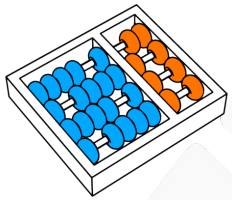
- Suponha que $L_2 \in \mathrm{P}$.
- Seja A_2 um algoritmo que decide L_2 em tempo polinomial.
- Seja f um algoritmo polinomial que reduz L_1 a L_2.
- Crie um algoritmo A_1 fazendo $A_1(x) = A_2(f(x))$.
- Já que f é uma redução, obtemos:
  - Se $x \in L_1$, então $f(x) \in L_2$ e $A_2(f(x)) = 1$, daí $A_1(x) = 1$.
  - Se $x \notin L_1$, então $f(x) \notin L_2$ e $A_2(f(x)) = 0$, daí $A_1(x) = 0$.
- Assim, A_1 decide L_1 em tempo polinomial.



Classe NP-completo

Definição

*A **CLASSE** NP-**COMPLETO** é o conjunto de linguagens $L \subseteq \{0,1\}^*$ tais que:*

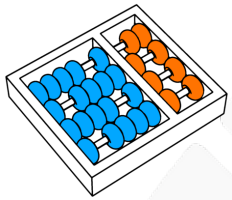


Classe NP-completo

Definição

*A **CLASSE** NP-**COMPLETO** é o conjunto de linguagens $L \subseteq \{0,1\}^*$ tais que:*

1. $L \in \text{NP}$.

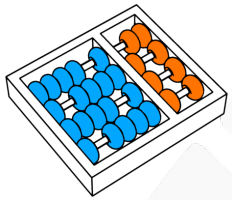


Classe NP-completo

Definição

*A **CLASSE** NP-**COMPLETO** é o conjunto de linguagens $L \subseteq \{0,1\}^*$ tais que:*

1. $L \in \text{NP}$.
2. $L' \preccurlyeq_p L$ *para todo* $L' \in \text{NP}$.



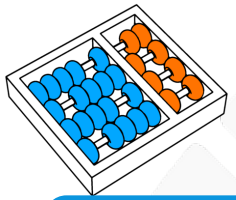
Classe NP-completo

Definição

*A **CLASSE** NP-**COMPLETO** é o conjunto de linguagens $L \subseteq \{0,1\}^*$ tais que:*

1. $L \in \text{NP}$.
2. $L' \preccurlyeq_p L$ *para todo* $L' \in \text{NP}$.

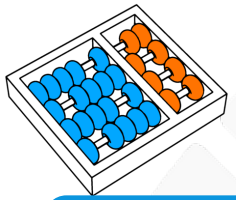
- Se apenas 2 for satisfeita, dizemos que L é NP-**DIFÍCIL**.



Condição para NP = NP

Teorema

Se existe algoritmo que decide $L \in$ NP*-completo em tempo polinomial, então* P = NP*.*

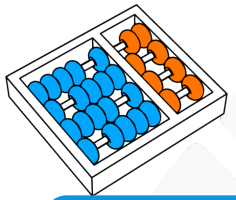


Condição para NP = NP

Teorema

Se existe algoritmo que decide $L \in$ NP*-completo em tempo polinomial, então* P = NP*.*

Demonstração:



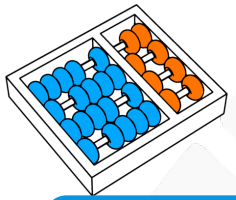
Condição para NP = NP

Teorema

Se existe algoritmo que decide $L \in$ NP*-completo em tempo polinomial, então* P = NP*.*

Demonstração:

- Suponha que existe $L \in \mathrm{P} \cap$ NP-completo.



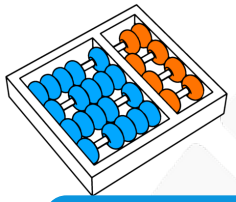
Condição para NP = NP

Teorema

Se existe algoritmo que decide $L \in$ NP*-completo em tempo polinomial, então* P = NP*.*

Demonstração:

- Suponha que existe $L \in \mathrm{P} \cap$ NP-completo.
- Como $L \in$ NP-completo, para toda $L' \in$ NP, temos $L' \preccurlyeq_p L$.



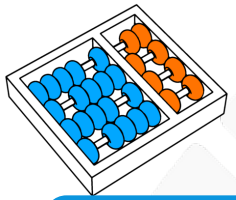
Condição para NP = NP

Teorema

Se existe algoritmo que decide $L \in$ NP*-completo em tempo polinomial, então* P = NP*.*

Demonstração:

- Suponha que existe $L \in \mathrm{P} \cap$ NP-completo.
- Como $L \in$ NP-completo, para toda $L' \in$ NP, temos $L' \preccurlyeq_p L$.
- Mas como $L \in$ P, isso implica $L' \in$ P pelo teorema anterior.



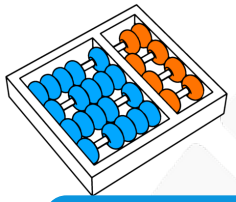
Condição para NP = NP

Teorema

Se existe algoritmo que decide $L \in$ NP-completo em tempo polinomial, então $\mathrm{P} = \mathrm{NP}$.

Demonstração:

- Suponha que existe $L \in \mathrm{P} \cap$ NP-completo.
- Como $L \in$ NP-completo, para toda $L' \in \mathrm{NP}$, temos $L' \preccurlyeq_p L$.
- Mas como $L \in \mathrm{P}$, isso implica $L' \in \mathrm{P}$ pelo teorema anterior.
- Então, $\mathrm{NP} \subseteq \mathrm{P}$ e, portanto, $\mathrm{NP} = \mathrm{P}$.



Condição para NP = NP

Teorema

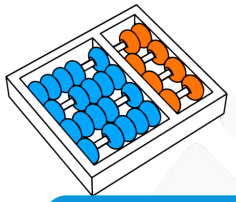
Se existe algoritmo que decide $L \in$ NP*-completo em tempo polinomial, então* P = NP*.*

Demonstração:

- Suponha que existe $L \in \mathrm{P} \cap$ NP-completo.
- Como $L \in$ NP-completo, para toda $L' \in$ NP, temos $L' \preccurlyeq_p L$.
- Mas como $L \in$ P, isso implica $L' \in$ P pelo teorema anterior.
- Então, NP $\subseteq$ P e, portanto, NP = P.

Teorema

Se existe uma linguagem $L \in$ NP *tal que* $L \notin$ P*, então* NP*-completo* $\cap$ P $= \emptyset$*.*



Condição para NP = NP

Teorema

Se existe algoritmo que decide $L \in$ NP-completo em tempo polinomial, então $\mathrm{P} = \mathrm{NP}$.

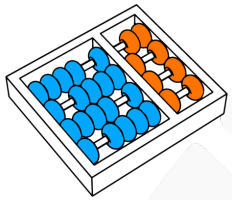
Demonstração:

- Suponha que existe $L \in \mathrm{P} \cap$ NP-completo.
- Como $L \in$ NP-completo, para toda $L' \in \mathrm{NP}$, temos $L' \preccurlyeq_p L$.
- Mas como $L \in \mathrm{P}$, isso implica $L' \in \mathrm{P}$ pelo teorema anterior.
- Então, $\mathrm{NP} \subseteq \mathrm{P}$ e, portanto, $\mathrm{NP} = \mathrm{P}$.

Teorema

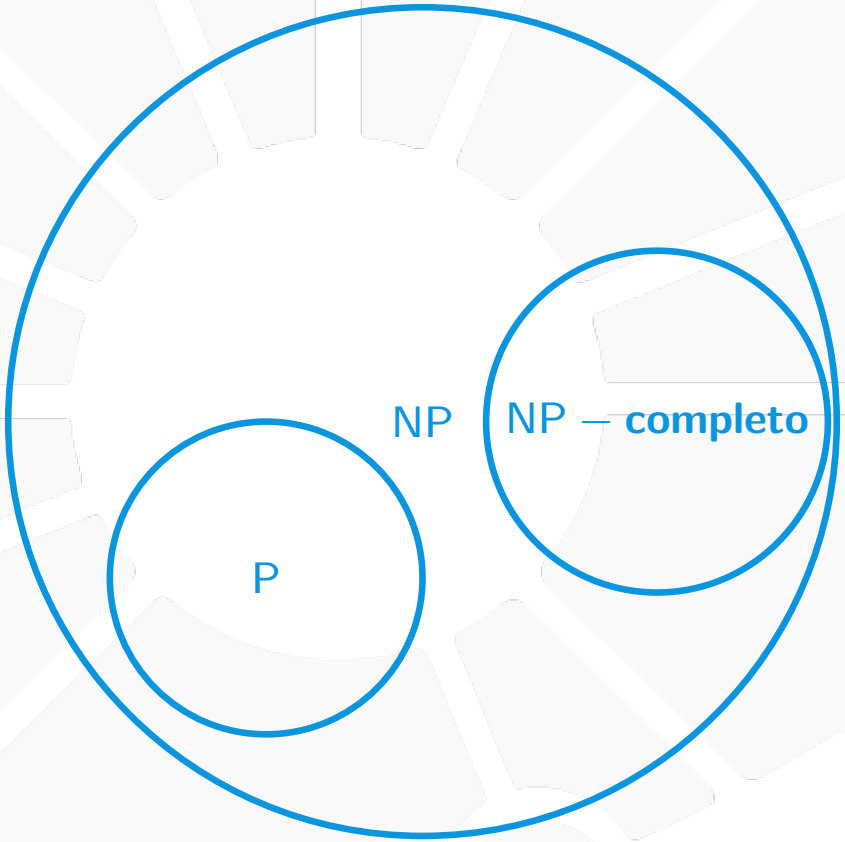
Se existe uma linguagem $L \in \mathrm{NP}$ tal que $L \notin \mathrm{P}$, então NP-*completo* $\cap \mathrm{P} = \emptyset$.

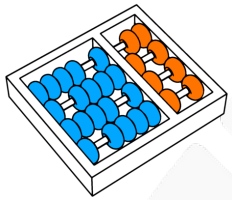
- Exercício.



Possível configuração de NP-completo

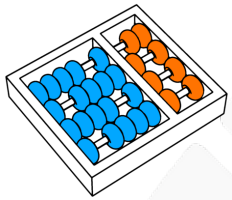
Como acreditamos que é a relação das classes:





A classe NP-completo não é vazia

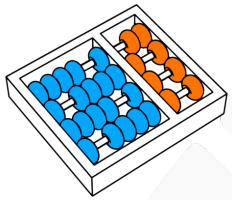
Mas será que a classe NP-completo é vazia?



A classe NP-completo não é vazia

Mas será que a classe NP-completo é vazia?

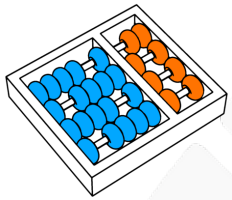
- Cook e Levin responderam que **NÃO** (independentemente).



A classe NP-completo não é vazia

Mas será que a classe NP-completo é vazia?

- Cook e Levin responderam que **NÃO** (independentemente).
- Mostraram que SAT $\in$ NP-completo.



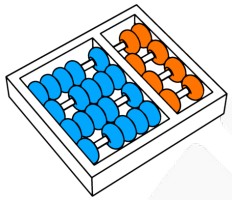
A classe NP-completo não é vazia

Mas será que a classe NP-completo é vazia?

- Cook e Levin responderam que **NÃO** (independentemente).
- Mostraram que SAT $\in$ NP-completo.

Teorema (Cook-Levin)

SAT é NP*-completo.*



A classe NP-completo não é vazia

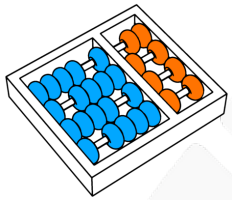
Mas será que a classe NP-completo é vazia?

- Cook e Levin responderam que **NÃO** (independentemente).
- Mostraram que SAT $\in$ NP-completo.

Teorema (Cook-Levin)

SAT é NP*-completo.*

- SAT é o problema da satisfatibilidade.



A classe NP-completo não é vazia

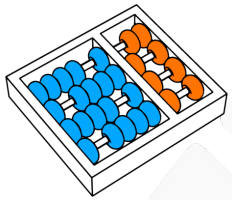
Mas será que a classe NP-completo é vazia?

- Cook e Levin responderam que **NÃO** (independentemente).
- Mostraram que SAT $\in$ NP-completo.

Teorema (Cook-Levin)

SAT é NP*-completo.*

- SAT é o problema da satisfatibilidade.
- Não vamos demonstrar esse teorema.



A classe NP-completo não é vazia

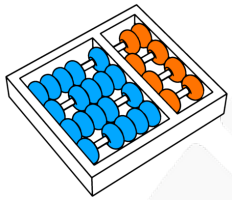
Mas será que a classe NP-completo é vazia?

- Cook e Levin responderam que **NÃO** (independentemente).
- Mostraram que SAT $\in$ NP-completo.

Teorema (Cook-Levin)

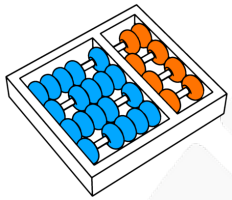
SAT é NP*-completo.*

- SAT é o problema da satisfatibilidade.
- Não vamos demonstrar esse teorema.
- Mas veremos um rascunho para um problema parecido.



Satisfatibilidade

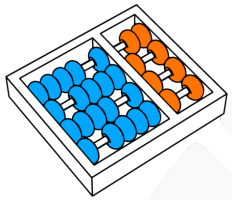
Considere uma **FÓRMULA BOOLEANA**:



Satisfatibilidade

Considere uma **FÓRMULA BOOLEANA**:

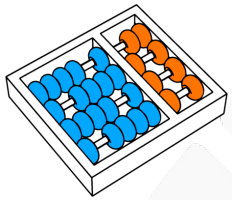
- Contém um conjuntos de variáveis booleanas.



Satisfatibilidade

Considere uma **FÓRMULA BOOLEANA**:

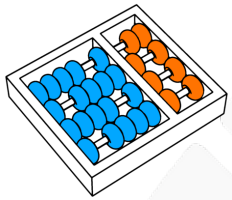
- Contém um conjuntos de variáveis booleanas.
- É escrita usando os seguintes operadores:



Satisfatibilidade

Considere uma **FÓRMULA BOOLEANA**:

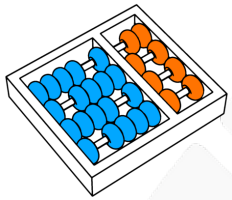
- Contém um conjuntos de variáveis booleanas.
- É escrita usando os seguintes operadores:
  1. Negação ($\neg$).



Satisfatibilidade

Considere uma **FÓRMULA BOOLEANA**:

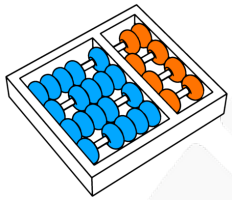
- Contém um conjuntos de variáveis booleanas.
- É escrita usando os seguintes operadores:
  1. Negação ($\neg$).
  2. Conjunção ($\wedge$).



Satisfatibilidade

Considere uma **FÓRMULA BOOLEANA**:

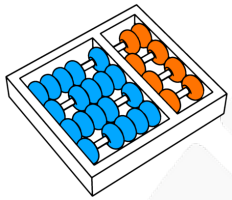
- Contém um conjuntos de variáveis booleanas.
- É escrita usando os seguintes operadores:
  1. Negação ($\neg$).
  2. Conjunção ($\wedge$).
  3. Disjunção ($\vee$).



Satisfatibilidade

Considere uma **FÓRMULA BOOLEANA**:

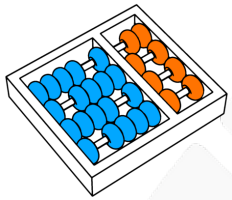
- Contém um conjuntos de variáveis booleanas.
- É escrita usando os seguintes operadores:
  1. Negação ($\neg$).
  2. Conjunção ($\wedge$).
  3. Disjunção ($\vee$).
  4. Implicação ($\rightarrow$).



Satisfatibilidade

Considere uma **FÓRMULA BOOLEANA**:

- Contém um conjuntos de variáveis booleanas.
- É escrita usando os seguintes operadores:
  1. Negação ($\neg$).
  2. Conjunção ($\wedge$).
  3. Disjunção ($\vee$).
  4. Implicação ($\rightarrow$).
  5. Equivalência ($\leftrightarrow$).

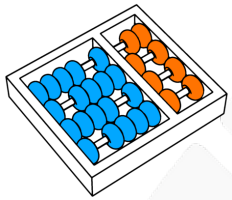


Satisfatibilidade

Considere uma **FÓRMULA BOOLEANA**:

- Contém um conjuntos de variáveis booleanas.
- É escrita usando os seguintes operadores:
  1. Negação ($\neg$).
  2. Conjunção ($\wedge$).
  3. Disjunção ($\vee$).
  4. Implicação ($\rightarrow$).
  5. Equivalência ($\leftrightarrow$).

Problema (Satisfatibilidade (SAT))



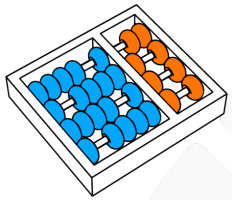
Satisfatibilidade

Considere uma **FÓRMULA BOOLEANA**:

- Contém um conjuntos de variáveis booleanas.
- É escrita usando os seguintes operadores:
  1. Negação ($\neg$).
  2. Conjunção ($\wedge$).
  3. Disjunção ($\vee$).
  4. Implicação ($\rightarrow$).
  5. Equivalência ($\leftrightarrow$).

Problema (Satisfatibilidade (SAT))

- ***Entrada:*** *Uma fórmula booleana.*



Satisfatibilidade

Considere uma **FÓRMULA BOOLEANA**:

- Contém um conjuntos de variáveis booleanas.
- É escrita usando os seguintes operadores:
  1. Negação ($\neg$).
  2. Conjunção ($\wedge$).
  3. Disjunção ($\vee$).
  4. Implicação ($\rightarrow$).
  5. Equivalência ($\leftrightarrow$).

Problema (Satisfatibilidade (SAT))

- ***Entrada:*** *Uma fórmula booleana.*
- ***Saída:*** *Decidir se existe atribuição de variáveis booleana para a qual a avaliação da fórmula é verdadeira.*

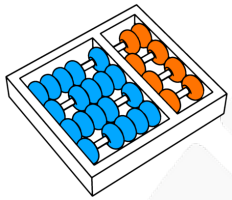
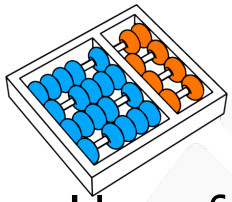


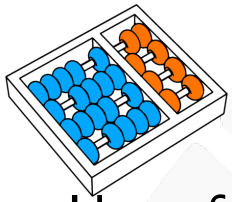
Tabela de operadores

a	b	$\neg a$	$a \wedge b$	$a \vee b$	$a \rightarrow b$	$a \leftrightarrow b$
F	F	V	F	F	V	V
F	V	V	F	V	V	F
V	F	F	F	V	F	F
V	V	F	V	V	V	V



Fórmula satisfazível

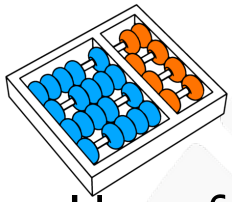
Uma formula é **SATISFAZÍVEL** se houver atribuição das variáveis para a qual a avaliação é verdadeira.



Fórmula satisfazível

Uma formula é **SATISFAZÍVEL** se houver atribuição das variáveis para a qual a avaliação é verdadeira.

Exemplo:

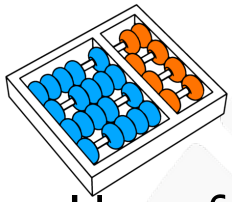


Fórmula satisfazível

Uma formula é **SATISFAZÍVEL** se houver atribuição das variáveis para a qual a avaliação é verdadeira.

Exemplo:

- Fórmula: $f = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$.

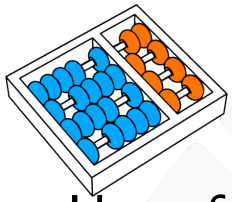


Fórmula satisfazível

Uma formula é **SATISFAZÍVEL** se houver atribuição das variáveis para a qual a avaliação é verdadeira.

Exemplo:

- Fórmula: $f = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$.
- Atribuição: $x_1 = 0,\ x_2 = 0,\ x_3 = 1,\ x_4 = 1$

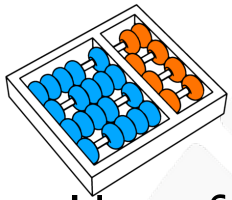


Fórmula satisfazível

Uma formula é **SATISFAZÍVEL** se houver atribuição das variáveis para a qual a avaliação é verdadeira.

Exemplo:

- Fórmula: $f = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$.
- Atribuição: $x_1 = 0,\ x_2 = 0,\ x_3 = 1,\ x_4 = 1$
- Avaliação:



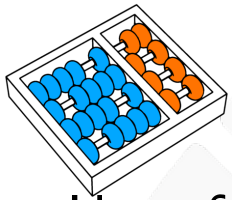
Fórmula satisfazível

Uma formula é **SATISFAZÍVEL** se houver atribuição das variáveis para a qual a avaliação é verdadeira.

Exemplo:

- Fórmula: $f = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$.
- Atribuição: $x_1 = 0,\ x_2 = 0,\ x_3 = 1,\ x_4 = 1$
- Avaliação:

$$f = ((0 \rightarrow 0) \vee \neg((\neg 0 \leftrightarrow 1) \vee 1)) \wedge \neg 0$$



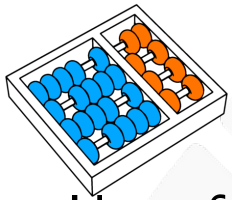
Fórmula satisfazível

Uma formula é **SATISFAZÍVEL** se houver atribuição das variáveis para a qual a avaliação é verdadeira.

Exemplo:

- Fórmula: $f = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$.
- Atribuição: $x_1 = 0,\ x_2 = 0,\ x_3 = 1,\ x_4 = 1$
- Avaliação:

$$\begin{aligned} f &= ((0 \rightarrow 0) \vee \neg((\neg 0 \leftrightarrow 1) \vee 1)) \wedge \neg 0 \\ &= (1 \vee \neg((1 \leftrightarrow 1) \vee 1)) \wedge 1 \end{aligned}$$



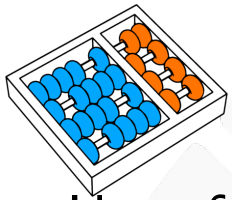
Fórmula satisfazível

Uma formula é **SATISFAZÍVEL** se houver atribuição das variáveis para a qual a avaliação é verdadeira.

Exemplo:

- Fórmula: $f = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$.
- Atribuição: $x_1 = 0,\ x_2 = 0,\ x_3 = 1,\ x_4 = 1$
- Avaliação:

$$\begin{aligned} f &= ((0 \rightarrow 0) \vee \neg((\neg 0 \leftrightarrow 1) \vee 1)) \wedge \neg 0 \\ &= (1 \vee \neg((1 \leftrightarrow 1) \vee 1)) \wedge 1 \\ &= (1 \vee \neg(1 \vee 1)) \wedge 1 \end{aligned}$$



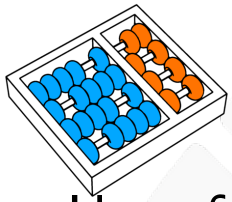
Fórmula satisfazível

Uma formula é **SATISFAZÍVEL** se houver atribuição das variáveis para a qual a avaliação é verdadeira.

Exemplo:

- Fórmula: $f = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$.
- Atribuição: $x_1 = 0,\ x_2 = 0,\ x_3 = 1,\ x_4 = 1$
- Avaliação:

$$\begin{aligned} f &= ((0 \rightarrow 0) \vee \neg((\neg 0 \leftrightarrow 1) \vee 1)) \wedge \neg 0 \\ &= (1 \vee \neg((1 \leftrightarrow 1) \vee 1)) \wedge 1 \\ &= (1 \vee \neg(1 \vee 1)) \wedge 1 \\ &= (1 \vee 0) \wedge 1 \end{aligned}$$



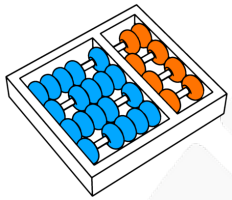
Fórmula satisfazível

Uma formula é **SATISFAZÍVEL** se houver atribuição das variáveis para a qual a avaliação é verdadeira.

Exemplo:

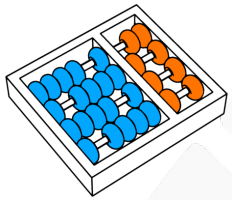
- Fórmula: $f = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$.
- Atribuição: $x_1 = 0,\ x_2 = 0,\ x_3 = 1,\ x_4 = 1$
- Avaliação:

$$
\begin{aligned}
f &= ((0 \rightarrow 0) \vee \neg((\neg 0 \leftrightarrow 1) \vee 1)) \wedge \neg 0 \\
&= (1 \vee \neg((1 \leftrightarrow 1) \vee 1)) \wedge 1 \\
&= (1 \vee \neg(1 \vee 1)) \wedge 1 \\
&= (1 \vee 0) \wedge 1 \\
&= 1
\end{aligned}
$$



Linguagem correspondente

A linguagem SAT é aquela que contém fórmulas booleanas com uma atribuição verdadeira.

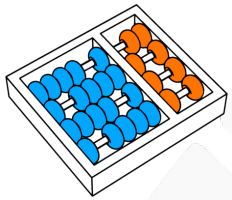


Linguagem correspondente

A linguagem SAT é aquela que contém fórmulas booleanas com uma atribuição verdadeira.

Problema (Satisfatibilidade)

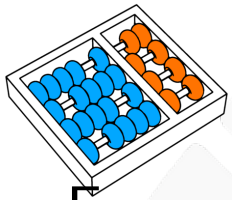
$$SAT = \{\langle f \rangle : f \text{ é uma fórmula booleana satisfazível}\}$$



É fácil verificar

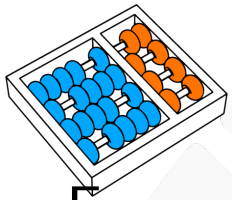
Lema

SAT está em NP*.*



Prova

Escrevemos um algoritmo verificador:

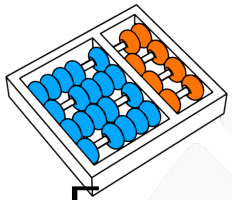


Prova

Escrevemos um algoritmo verificador:

Algoritmo 18: VERIFICA-SAT($\langle f \rangle, \langle y \rangle$)

```
1 se f(y) = 1
2     devolva SIM
3 senão
4     devolva NAO
```



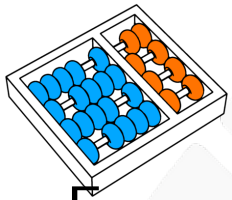
Prova

Escrevemos um algoritmo verificador:

Algoritmo 19: VERIFICA-SAT($\langle f \rangle, \langle y \rangle$)

```
1 se f(y) = 1
2     devolva SIM
3 senão
4     devolva NAO
```

Observe que o algoritmo verifica SAT em tempo polinomial:



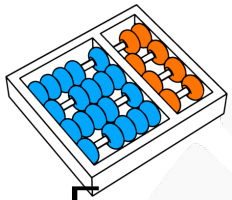
Prova

Escrevemos um algoritmo verificador:

Algoritmo 20: VERIFICA-SAT($\langle f \rangle, \langle y \rangle$)

```
1 se f(y) = 1
2     devolva SIM
3 senão
4     devolva NAO
```

Observe que o algoritmo verifica SAT em tempo polinomial:



Prova

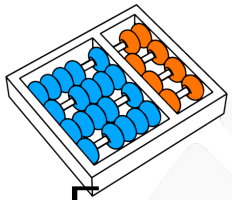
Escrevemos um algoritmo verificador:

Algoritmo 21: VERIFICA-SAT($\langle f \rangle, \langle y \rangle$)

```
1 se f(y) = 1
2     devolva SIM
3 senão
4     devolva NAO
```

Observe que o algoritmo verifica SAT em tempo polinomial:

1. ▶ Se $\langle f \rangle \in$ SAT, então f é satisfazível.



Prova

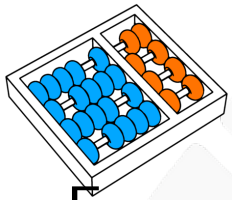
Escrevemos um algoritmo verificador:

Algoritmo 22: VERIFICA-SAT($\langle f \rangle, \langle y \rangle$)

```
1 se f(y) = 1
2     devolva SIM
3 senão
4     devolva NAO
```

Observe que o algoritmo verifica SAT em tempo polinomial:

1.
   - Se $\langle f \rangle \in$ SAT, então f é satisfazível.
   - Portanto, existe uma atribuição y das variáveis que faz f verdadeira.



Prova

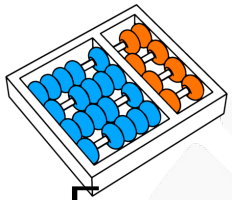
Escrevemos um algoritmo verificador:

Algoritmo 23: VERIFICA-SAT($\langle f \rangle, \langle y \rangle$)

```
1 se f(y) = 1
2     devolva SIM
3 senão
4     devolva NAO
```

Observe que o algoritmo verifica SAT em tempo polinomial:

1.
   - Se $\langle f \rangle \in$ SAT, então f é satisfazível.
   - Portanto, existe uma atribuição y das variáveis que faz f verdadeira.
   - Logo, VERIFICA-SAT($\langle f \rangle, \langle y \rangle$) devolve SIM.



Prova

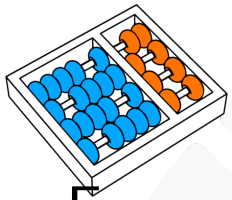
Escrevemos um algoritmo verificador:

Algoritmo 24: VERIFICA-SAT($\langle f \rangle, \langle y \rangle$)

```
1 se f(y) = 1
2     devolva SIM
3 senão
4     devolva NAO
```

Observe que o algoritmo verifica SAT em tempo polinomial:

1.
   - Se $\langle f \rangle \in$ SAT, então f é satisfazível.
   - Portanto, existe uma atribuição y das variáveis que faz f verdadeira.
   - Logo, VERIFICA-SAT($\langle f \rangle, \langle y \rangle$) devolve SIM.



Prova

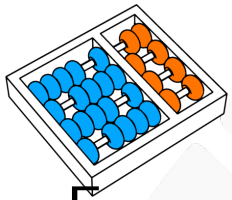
Escrevemos um algoritmo verificador:

Algoritmo 25: VERIFICA-SAT($\langle f \rangle, \langle y \rangle$)

```
1 se f(y) = 1
2     devolva SIM
3 senão
4     devolva NAO
```

Observe que o algoritmo verifica SAT em tempo polinomial:

1.
   - Se $\langle f \rangle \in$ SAT, então f é satisfazível.
   - Portanto, existe uma atribuição y das variáveis que faz f verdadeira.
   - Logo, VERIFICA-SAT($\langle f \rangle, \langle y \rangle$) devolve SIM.
2.
   - Suponha VERIFICA-SAT($\langle f \rangle, \langle y \rangle$) devolve SIM.



Prova

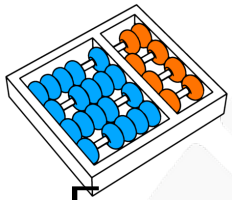
Escrevemos um algoritmo verificador:

Algoritmo 26: VERIFICA-SAT($\langle f \rangle, \langle y \rangle$)

```
1 se f(y) = 1
2     devolva SIM
3 senão
4     devolva NAO
```

Observe que o algoritmo verifica SAT em tempo polinomial:

1.
   - Se $\langle f \rangle \in$ SAT, então f é satisfazível.
   - Portanto, existe uma atribuição y das variáveis que faz f verdadeira.
   - Logo, VERIFICA-SAT($\langle f \rangle, \langle y \rangle$) devolve SIM.
2.
   - Suponha VERIFICA-SAT($\langle f \rangle, \langle y \rangle$) devolve SIM.
   - Então, y é uma atribuição verdadeira para a fórmula.



Prova

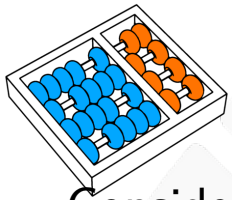
Escrevemos um algoritmo verificador:

Algoritmo 27: VERIFICA-SAT($\langle f \rangle, \langle y \rangle$)

```
1 se f(y) = 1
2     devolva SIM
3 senão
4     devolva NAO
```

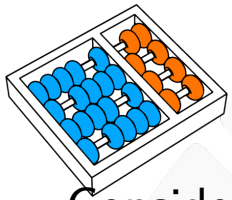
Observe que o algoritmo verifica SAT em tempo polinomial:

1.
   - Se $\langle f \rangle \in$ SAT, então f é satisfazível.
   - Portanto, existe uma atribuição y das variáveis que faz f verdadeira.
   - Logo, VERIFICA-SAT($\langle f \rangle, \langle y \rangle$) devolve SIM.
2.
   - Suponha VERIFICA-SAT($\langle f \rangle, \langle y \rangle$) devolve SIM.
   - Então, y é uma atribuição verdadeira para a fórmula.
   - Portanto, f é satisfazível e $\langle f \rangle \in$ SAT.



Circuito lógico

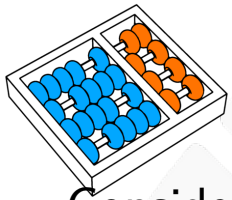
Considere um **CIRCUITO LÓGICO**:



Circuito lógico

Considere um **CIRCUITO LÓGICO**:

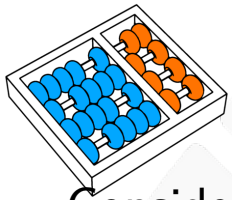
- Contém n fios de entrada.



Circuito lógico

Considere um **CIRCUITO LÓGICO**:

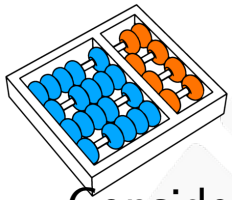
- Contém n fios de entrada.
- É formado combinando portas lógicas:



Circuito lógico

Considere um **CIRCUITO LÓGICO**:

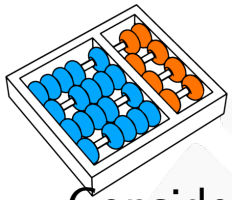
- Contém n fios de entrada.
- É formado combinando portas lógicas:
  - NOT ($\neg$).



Circuito lógico

Considere um **CIRCUITO LÓGICO**:

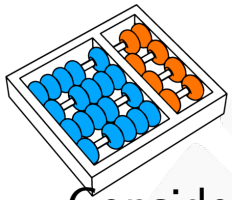
- Contém n fios de entrada.
- É formado combinando portas lógicas:
  - NOT ($\neg$).
  - AND ($\wedge$).



Circuito lógico

Considere um **CIRCUITO LÓGICO**:

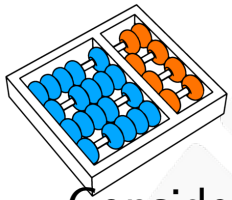
- Contém n fios de entrada.
- É formado combinando portas lógicas:
  - NOT ($\neg$).
  - AND ($\wedge$).
  - OR ($\vee$).



Circuito lógico

Considere um **CIRCUITO LÓGICO**:

- Contém n fios de entrada.
- É formado combinando portas lógicas:
  - NOT ($\neg$).
  - AND ($\wedge$).
  - OR ($\vee$).
- A saída do circuito é dada por um fio.

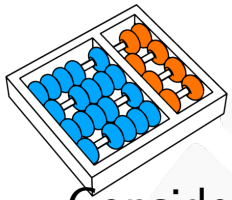


Circuito lógico

Considere um **CIRCUITO LÓGICO**:

- Contém n fios de entrada.
- É formado combinando portas lógicas:
  - NOT ($\neg$).
  - AND ($\wedge$).
  - OR ($\vee$).
- A saída do circuito é dada por um fio.

Uma **ATRIBUIÇÃO** de um circuito é uma atribuição de um bit 0 ou 1 para cada fio de entrada.



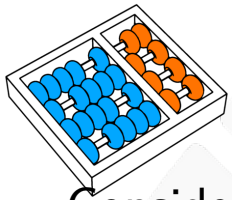
Circuito lógico

Considere um **CIRCUITO LÓGICO**:

- Contém n fios de entrada.
- É formado combinando portas lógicas:
  - NOT ($\neg$).
  - AND ($\wedge$).
  - OR ($\vee$).
- A saída do circuito é dada por um fio.

Uma **ATRIBUIÇÃO** de um circuito é uma atribuição de um bit 0 ou 1 para cada fio de entrada.

- Um circuito é **SATISFAZÍVEL** se ele possuir uma atribuição que resulta em bit 1 no fio de saída.



Circuito lógico

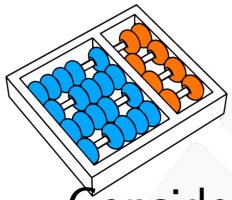
Considere um **CIRCUITO LÓGICO**:

- Contém n fios de entrada.
- É formado combinando portas lógicas:
  - NOT ($\neg$).
  - AND ($\wedge$).
  - OR ($\vee$).
- A saída do circuito é dada por um fio.

Uma **ATRIBUIÇÃO** de um circuito é uma atribuição de um bit 0 ou 1 para cada fio de entrada.

- Um circuito é **SATISFAZÍVEL** se ele possuir uma atribuição que resulta em bit 1 no fio de saída.

Problema (Satisfatibilidade de circuito (C-SAT))



Circuito lógico

Considere um **CIRCUITO LÓGICO**:

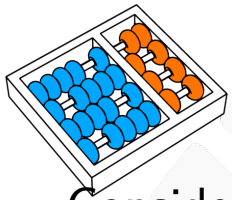
- Contém n fios de entrada.
- É formado combinando portas lógicas:
  - NOT ($\neg$).
  - AND ($\wedge$).
  - OR ($\vee$).
- A saída do circuito é dada por um fio.

Uma **ATRIBUIÇÃO** de um circuito é uma atribuição de um bit 0 ou 1 para cada fio de entrada.

- Um circuito é **SATISFAZÍVEL** se ele possuir uma atribuição que resulta em bit 1 no fio de saída.

Problema (Satisfatibilidade de circuito (C-SAT))

- ***Entrada:*** *Um circuito lógico.*



Circuito lógico

Considere um **CIRCUITO LÓGICO**:

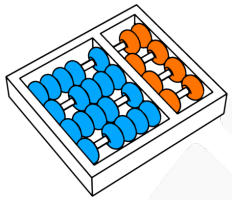
- Contém n fios de entrada.
- É formado combinando portas lógicas:
  - NOT ($\neg$).
  - AND ($\wedge$).
  - OR ($\vee$).
- A saída do circuito é dada por um fio.

Uma **ATRIBUIÇÃO** de um circuito é uma atribuição de um bit 0 ou 1 para cada fio de entrada.

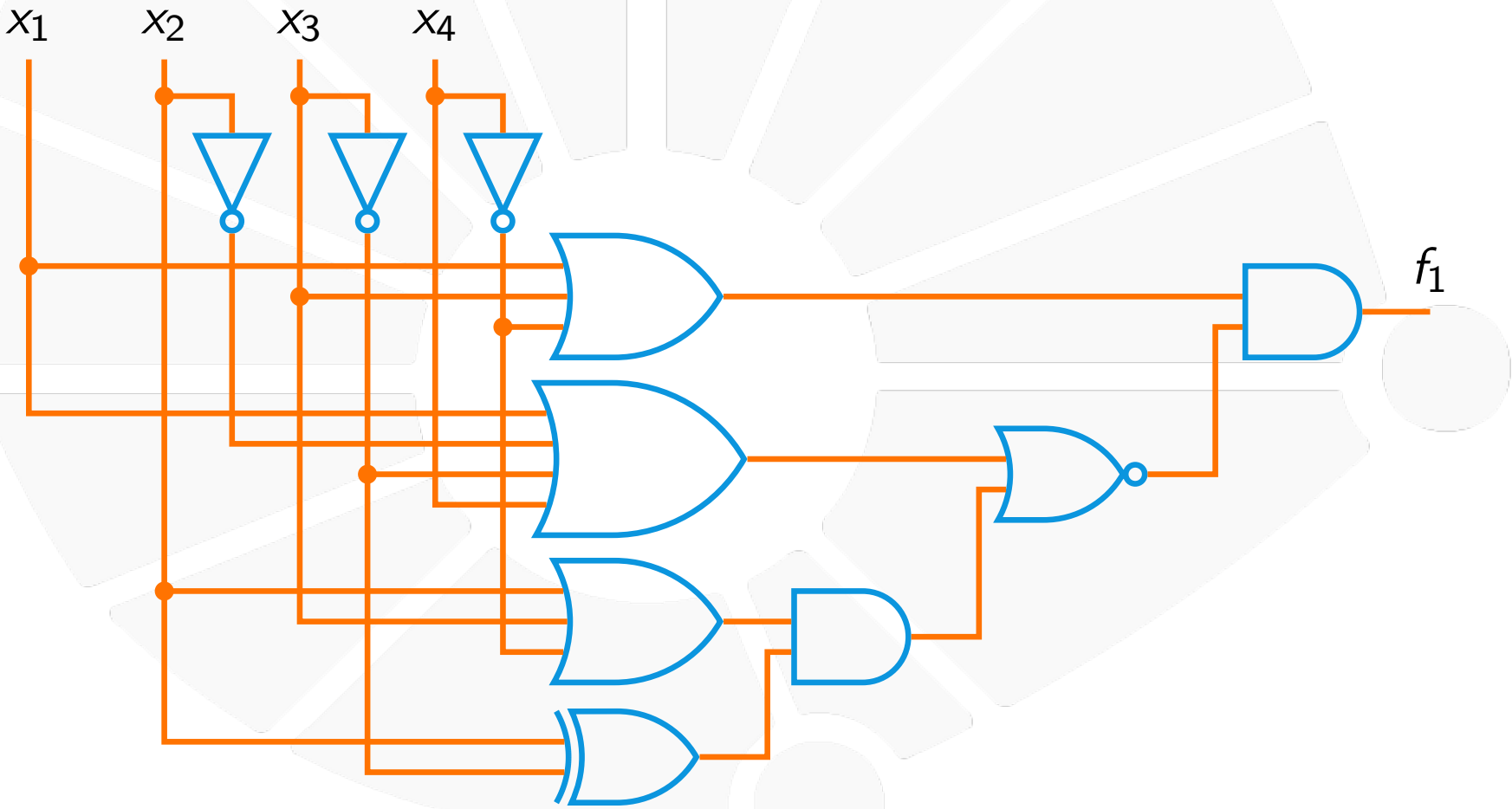
- Um circuito é **SATISFAZÍVEL** se ele possuir uma atribuição que resulta em bit 1 no fio de saída.

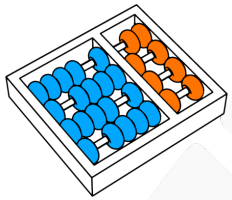
Problema (Satisfatibilidade de circuito (C-SAT))

- ***Entrada:*** *Um circuito lógico.*
- ***Saída:*** *Decidir se o circuito é satisfazível.*



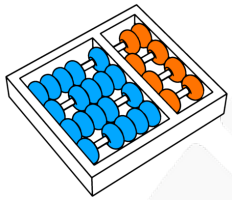
Exemplo de circuito





Linguagem correspondente

A linguagem C-SAT contém circuitos lógicos satisfatíveis.

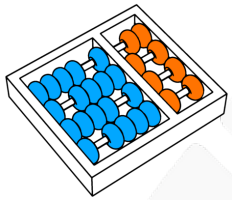


Linguagem correspondente

A linguagem C-SAT contém circuitos lógicos satisfatíveis.

Problema (Satisfatibilidade de circuito)

$$\textit{C-SAT} = \{\langle C \rangle : C \textit{ é um circuito lógico satisfazível}\}.$$



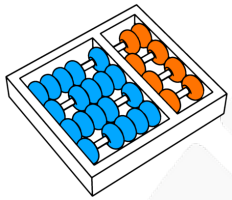
Linguagem correspondente

A linguagem C-SAT contém circuitos lógicos satisfatíveis.

Problema (Satisfatibilidade de circuito)

$$\textit{C-SAT} = \{\langle C \rangle : C \textit{ é um circuito lógico satisfazível}\}.$$

Há um algoritmo simples de tempo $O(2^n(n+m))$:



Linguagem correspondente

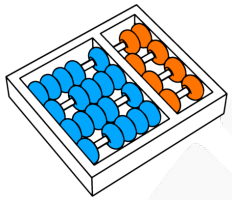
A linguagem C-SAT contém circuitos lógicos satisfatíveis.

Problema (Satisfatibilidade de circuito)

$$\textit{C-SAT} = \{\langle C \rangle : C \textit{ é um circuito lógico satisfazível}\}.$$

Há um algoritmo simples de tempo $O(2^n(n+m))$:

- n é o número de fios de entrada.



Linguagem correspondente

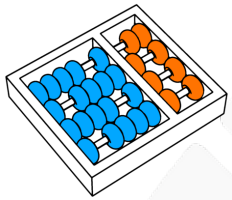
A linguagem C-SAT contém circuitos lógicos satisfatíveis.

Problema (Satisfatibilidade de circuito)

$$\textit{C-SAT} = \{\langle C \rangle : C \textit{ é um circuito lógico satisfazível}\}.$$

Há um algoritmo simples de tempo $O(2^n(n+m))$:

- n é o número de fios de entrada.
- m é o número de ligações entre portas.



Linguagem correspondente

A linguagem C-SAT contém circuitos lógicos satisfatíveis.

Problema (Satisfatibilidade de circuito)

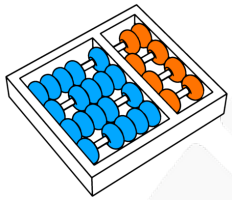
$$\textit{C-SAT} = \{\langle C \rangle : C \textit{ é um circuito lógico satisfazível}\}.$$

Há um algoritmo simples de tempo $O(2^n(n+m))$:

- n é o número de fios de entrada.
- m é o número de ligações entre portas.

Lema

C-SAT é NP.



Linguagem correspondente

A linguagem C-SAT contém circuitos lógicos satisfatíveis.

Problema (Satisfatibilidade de circuito)

$$\textit{C-SAT} = \{\langle C \rangle : C \textit{ é um circuito lógico satisfazível}\}.$$

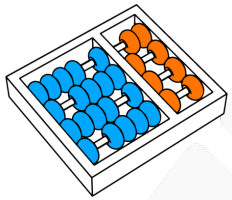
Há um algoritmo simples de tempo $O(2^n(n+m))$:

- n é o número de fios de entrada.
- m é o número de ligações entre portas.

Lema

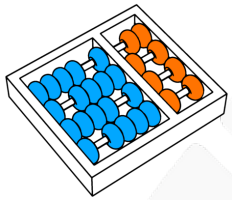
C-SAT é NP.

- Análogo à demonstração de que SAT $\in$ NP. (exercício)



Redução de NP para circuitos

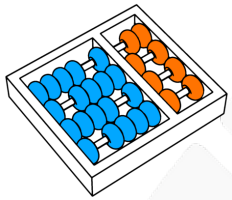
Queremos mostrar que C-SAT é NP-completo:



Redução de NP para circuitos

Queremos mostrar que C-SAT é NP-completo:

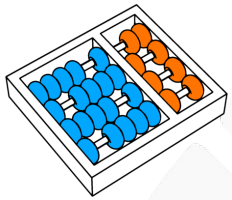
- Falta mostrar então que C-SAT é NP-difícil.



Redução de NP para circuitos

Queremos mostrar que C-SAT é NP-completo:

- Falta mostrar então que C-SAT é NP-difícil.
- Queremos que para todo $Q \in \text{NP}$, tenhamos $Q \preccurlyeq_p \text{C-SAT}$.

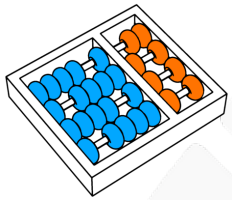


Redução de NP para circuitos

Queremos mostrar que C-SAT é NP-completo:

- Falta mostrar então que C-SAT é NP-difícil.
- Queremos que para todo $Q \in \text{NP}$, tenhamos $Q \preccurlyeq_p \text{C-SAT}$.

Nosso objetivo é projetar uma redução polinomial F:



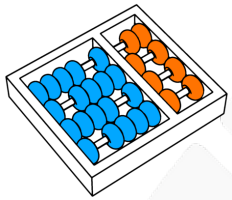
Redução de NP para circuitos

Queremos mostrar que C-SAT é NP-completo:

- Falta mostrar então que C-SAT é NP-difícil.
- Queremos que para todo $Q \in \text{NP}$, tenhamos $Q \preccurlyeq_p \text{C-SAT}$.

Nosso objetivo é projetar uma redução polinomial F:

Problema



Redução de NP para circuitos

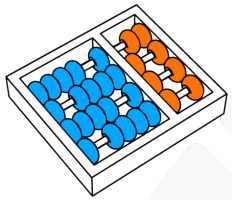
Queremos mostrar que C-SAT é NP-completo:

- Falta mostrar então que C-SAT é NP-difícil.
- Queremos que para todo $Q \in \text{NP}$, tenhamos $Q \preccurlyeq_p \text{C-SAT}$.

Nosso objetivo é projetar uma redução polinomial F:

Problema

- ***Entrada:*** *Instância* $x \in \{0,1\}^*$ *de* Q.



Redução de NP para circuitos

Queremos mostrar que C-SAT é NP-completo:

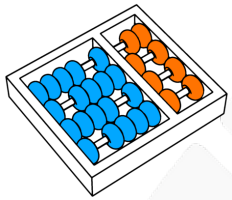
- Falta mostrar então que C-SAT é NP-difícil.
- Queremos que para todo $Q \in \text{NP}$, tenhamos $Q \preccurlyeq_p \text{C-SAT}$.

Nosso objetivo é projetar uma redução polinomial F:

Problema

- ***Entrada:*** *Instância* $x \in \{0,1\}^*$ *de* Q.
- ***Saída:*** *Circuito* $F(x)$ *tal que:*

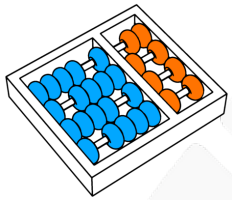
$$x \in Q \quad \textbf{\textit{se e somente se}} \quad F(x) \in \textit{C-SAT}.$$



NP-dificuldade

Lema

C-SAT é NP*-difícil.*

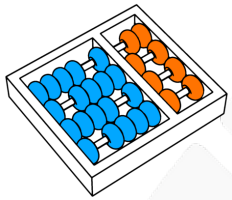


NP-dificuldade

Lema

C-SAT é NP-*difícil.*

Demonstração:



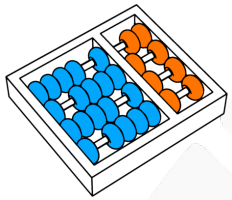
NP-dificuldade

Lema

C-SAT é NP-*difícil.*

Demonstração:

- Considere um problema arbitrário $Q \in \text{NP}$.



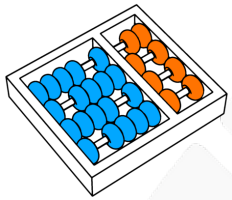
NP-dificuldade

Lema

C-SAT é NP-*difícil.*

Demonstração:

- Considere um problema arbitrário $Q \in \text{NP}$.
- Existe um algoritmo verificador polinomial A para Q.



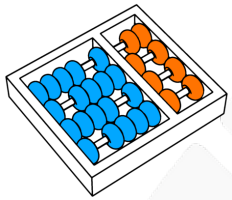
NP-dificuldade

Lema

C-SAT é NP*-difícil.*

Demonstração:

- Considere um problema arbitrário $Q \in$ NP.
- Existe um algoritmo verificador polinomial A para Q.
- Dada uma instância x de Q, criaremos a instância $F(x)$ de C-SAT.



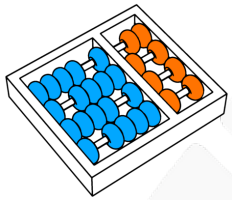
NP-dificuldade

Lema

C-SAT é NP*-difícil.*

Demonstração:

- Considere um problema arbitrário $Q \in$ NP.
- Existe um algoritmo verificador polinomial A para Q.
- Dada uma instância x de Q, criaremos a instância $F(x)$ de C-SAT.
- O algoritmo verificador de Q recebe duas strings, x e y:



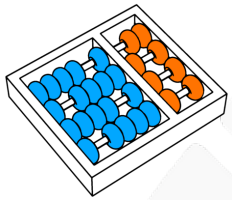
NP-dificuldade

Lema

C-SAT é NP*-difícil.*

Demonstração:

- Considere um problema arbitrário $Q \in$ NP.
- Existe um algoritmo verificador polinomial A para Q.
- Dada uma instância x de Q, criaremos a instância $F(x)$ de C-SAT.
- O algoritmo verificador de Q recebe duas strings, x e y:
  - A entrada x tem tamanho $|x| = n$.



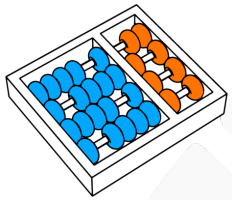
NP-dificuldade

Lema

C-SAT é NP*-difícil.*

Demonstração:

- Considere um problema arbitrário $Q \in \text{NP}$.
- Existe um algoritmo verificador polinomial A para Q.
- Dada uma instância x de Q, criaremos a instância $F(x)$ de C-SAT.
- O algoritmo verificador de Q recebe duas strings, x e y:
  - A entrada x tem tamanho $|x| = n$.
  - O certificado y tem tamanho $|y| = n^{k'}$, para k' constante.



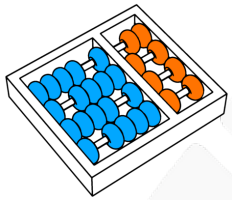
NP-dificuldade

Lema

C-SAT é NP*-difícil.*

Demonstração:

- Considere um problema arbitrário $Q \in$ NP.
- Existe um algoritmo verificador polinomial A para Q.
- Dada uma instância x de Q, criaremos a instância $F(x)$ de C-SAT.
- O algoritmo verificador de Q recebe duas strings, x e y:
  - A entrada x tem tamanho $|x| = n$.
  - O certificado y tem tamanho $|y| = n^{k'}$, para k' constante.
- Relembre que A leva tempo polinomial em $|x|$ e $|y|$.



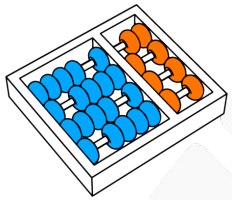
NP-dificuldade

Lema

C-SAT é NP*-difícil.*

Demonstração:

- Considere um problema arbitrário $Q \in$ NP.
- Existe um algoritmo verificador polinomial A para Q.
- Dada uma instância x de Q, criaremos a instância $F(x)$ de C-SAT.
- O algoritmo verificador de Q recebe duas strings, x e y:
  - A entrada x tem tamanho $|x| = n$.
  - O certificado y tem tamanho $|y| = n^{k'}$, para k' constante.
- Relembre que A leva tempo polinomial em $|x|$ e $|y|$.
- Assim, ele executa até n^k passos, para k constante.



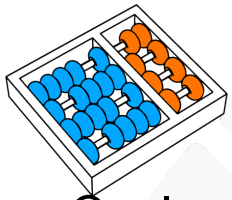
NP-dificuldade

Lema

C-SAT é NP*-difícil.*

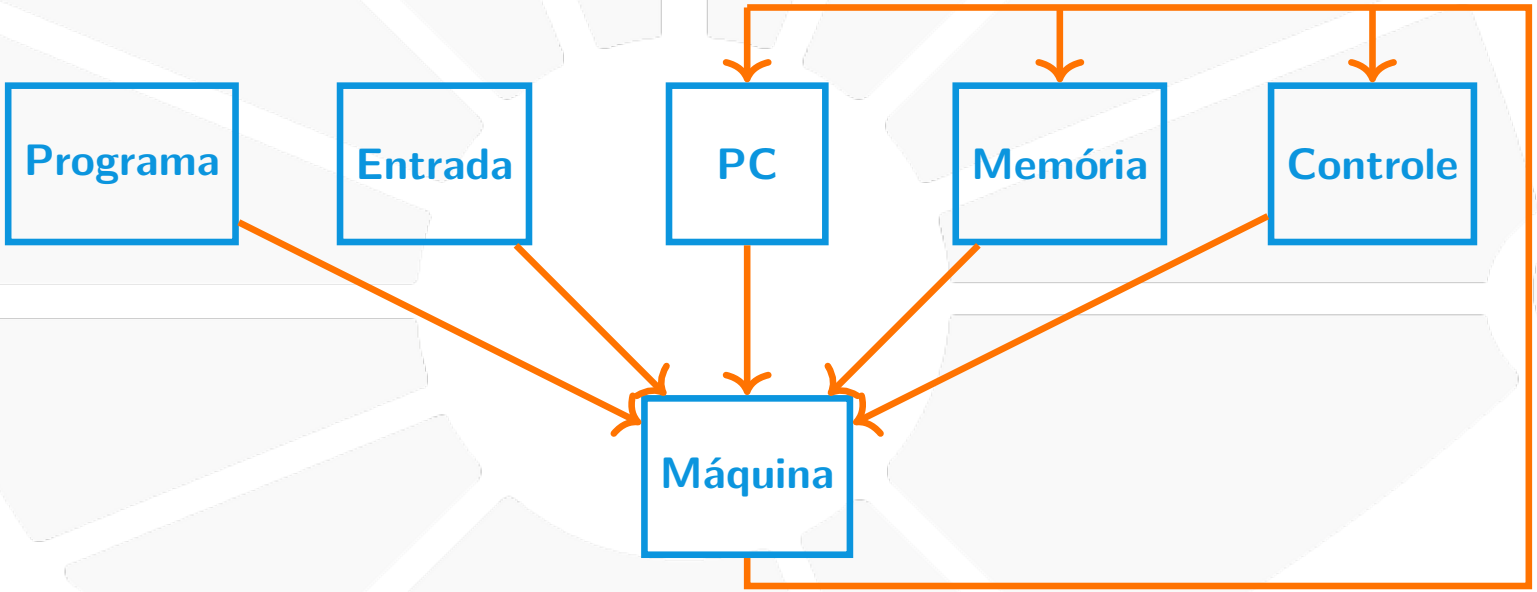
Demonstração:

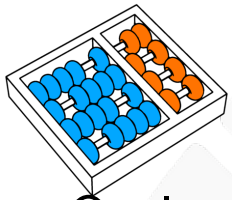
- Considere um problema arbitrário $Q \in$ NP.
- Existe um algoritmo verificador polinomial A para Q.
- Dada uma instância x de Q, criaremos a instância $F(x)$ de C-SAT.
- O algoritmo verificador de Q recebe duas strings, x e y:
  - A entrada x tem tamanho $|x| = n$.
  - O certificado y tem tamanho $|y| = n^{k'}$, para k' constante.
- Relembre que A leva tempo polinomial em $|x|$ e $|y|$.
- Assim, ele executa até n^k passos, para k constante.
- A ideia é montar um circuito que simula n^k passos de A.



Continuação da demonstração

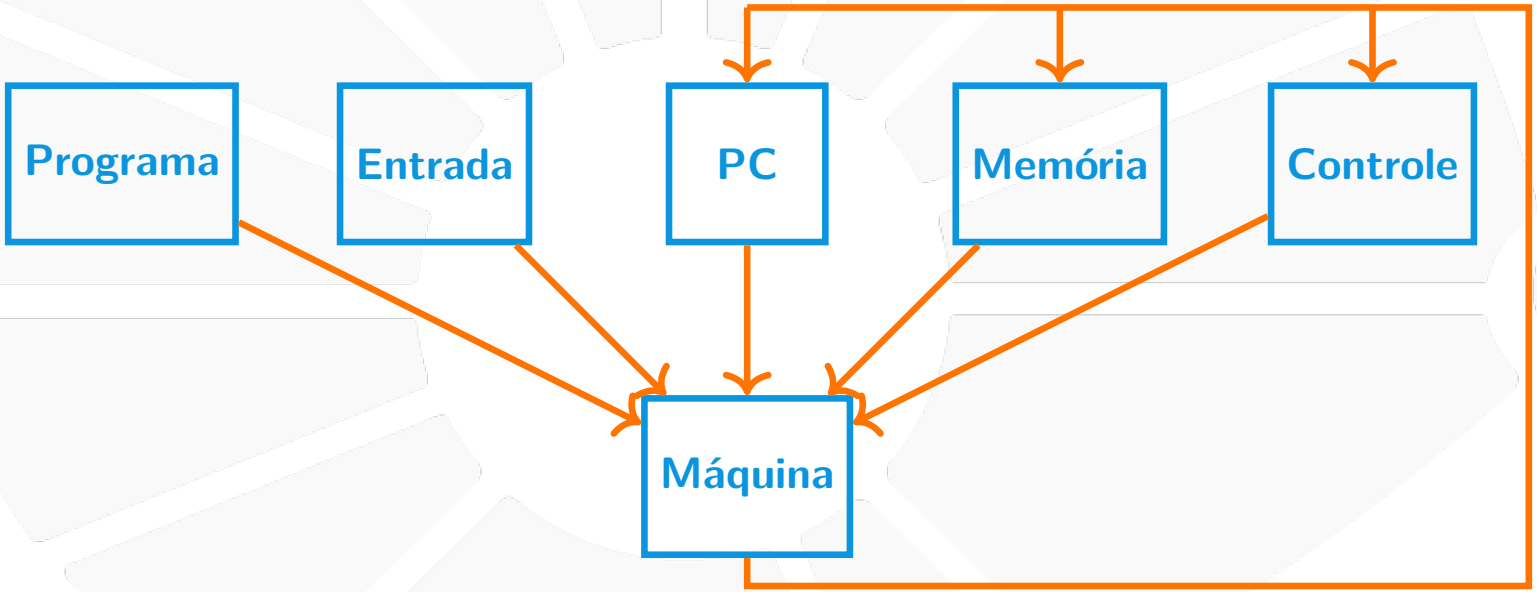
O algoritmo A pode ser implementado por um computador de circuitos lógicos.



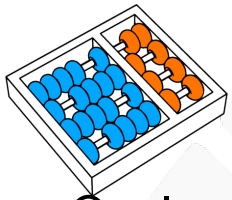


Continuação da demonstração

O algoritmo A pode ser implementado por um computador de circuitos lógicos.

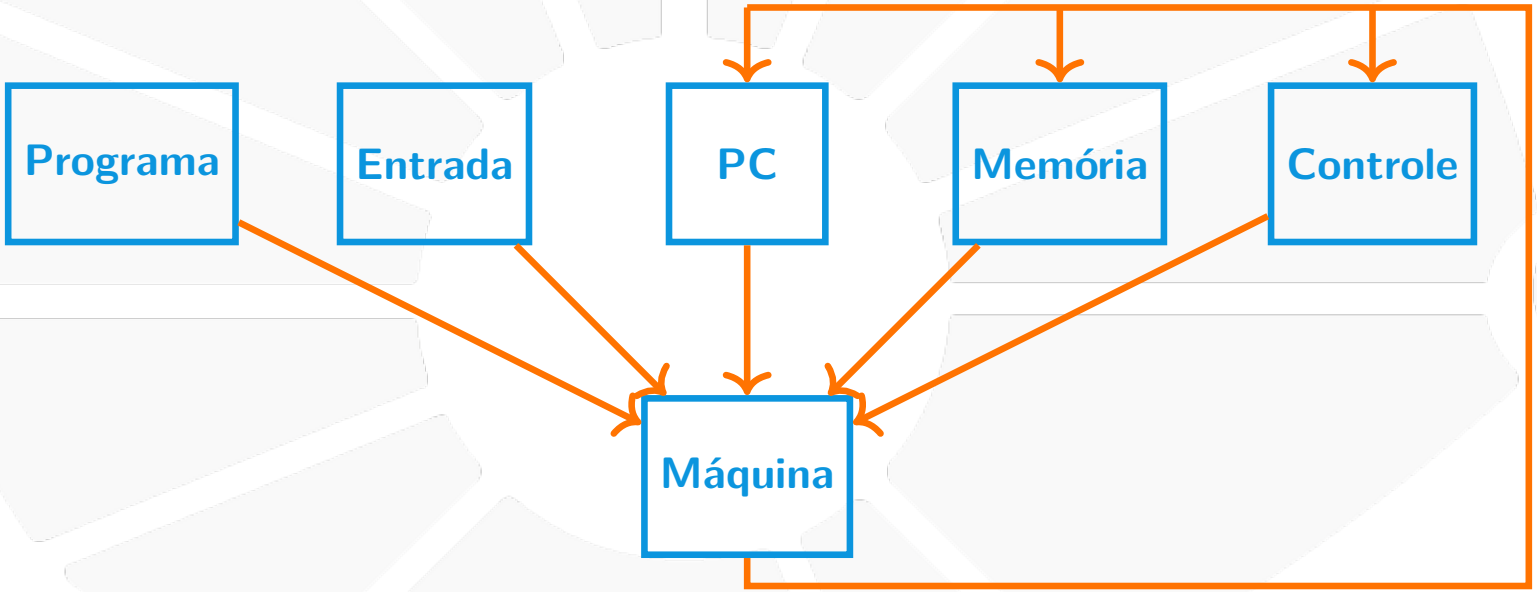


- Os circuitos têm **RETROALIMENTAÇÃO**.

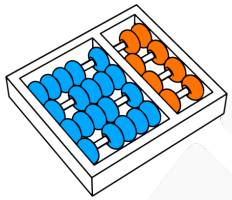


Continuação da demonstração

O algoritmo A pode ser implementado por um computador de circuitos lógicos.

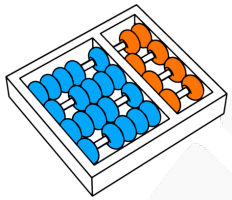


- Os circuitos têm **RETROALIMENTAÇÃO**.
- Após executar uma instrução, a memória é modificada



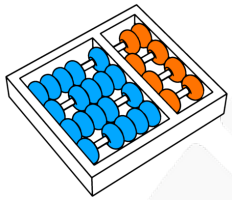
Continuação da demonstração

- Ajustamos o algoritmo A para que a saída seja escrita em um bit específico da memória denotado por C.



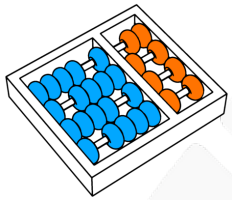
Continuação da demonstração

- Ajustamos o algoritmo A para que a saída seja escrita em um bit específico da memória denotado por C.
- Cada instrução executada pelo algoritmo A:



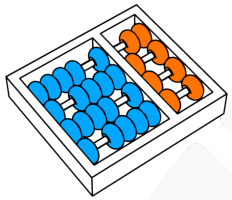
Continuação da demonstração

- Ajustamos o algoritmo A para que a saída seja escrita em um bit específico da memória denotado por C.
- Cada instrução executada pelo algoritmo A:
  - Começa em um estado de memória, PC e controle.



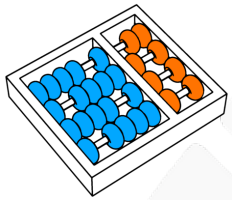
Continuação da demonstração

- Ajustamos o algoritmo A para que a saída seja escrita em um bit específico da memória denotado por C.
- Cada instrução executada pelo algoritmo A:
  - Começa em um estado de memória, PC e controle.
  - Modifica esse estado de memória, PC e controle.



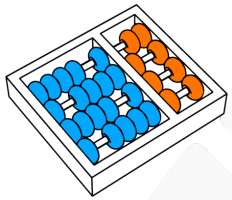
Continuação da demonstração

- Ajustamos o algoritmo A para que a saída seja escrita em um bit específico da memória denotado por C.
- Cada instrução executada pelo algoritmo A:
  - Começa em um estado de memória, PC e controle.
  - Modifica esse estado de memória, PC e controle.
- Criamos n^k cópias da máquina, uma para cada instrução:



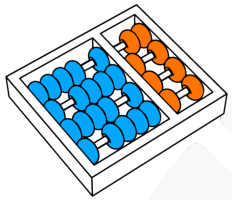
Continuação da demonstração

- Ajustamos o algoritmo A para que a saída seja escrita em um bit específico da memória denotado por C.
- Cada instrução executada pelo algoritmo A:
  - Começa em um estado de memória, PC e controle.
  - Modifica esse estado de memória, PC e controle.
- Criamos n^k cópias da máquina, uma para cada instrução:
  1. O estado de entrada da primeira máquina corresponde ao estado inicial da execução do algoritmo.



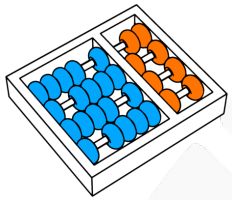
Continuação da demonstração

- Ajustamos o algoritmo A para que a saída seja escrita em um bit específico da memória denotado por C.
- Cada instrução executada pelo algoritmo A:
  - Começa em um estado de memória, PC e controle.
  - Modifica esse estado de memória, PC e controle.
- Criamos n^k cópias da máquina, uma para cada instrução:
  1. O estado de entrada da primeira máquina corresponde ao estado inicial da execução do algoritmo.
  2. A saída de uma instrução correspondente a uma cópia modifica o estado de entrada da cópia seguinte.

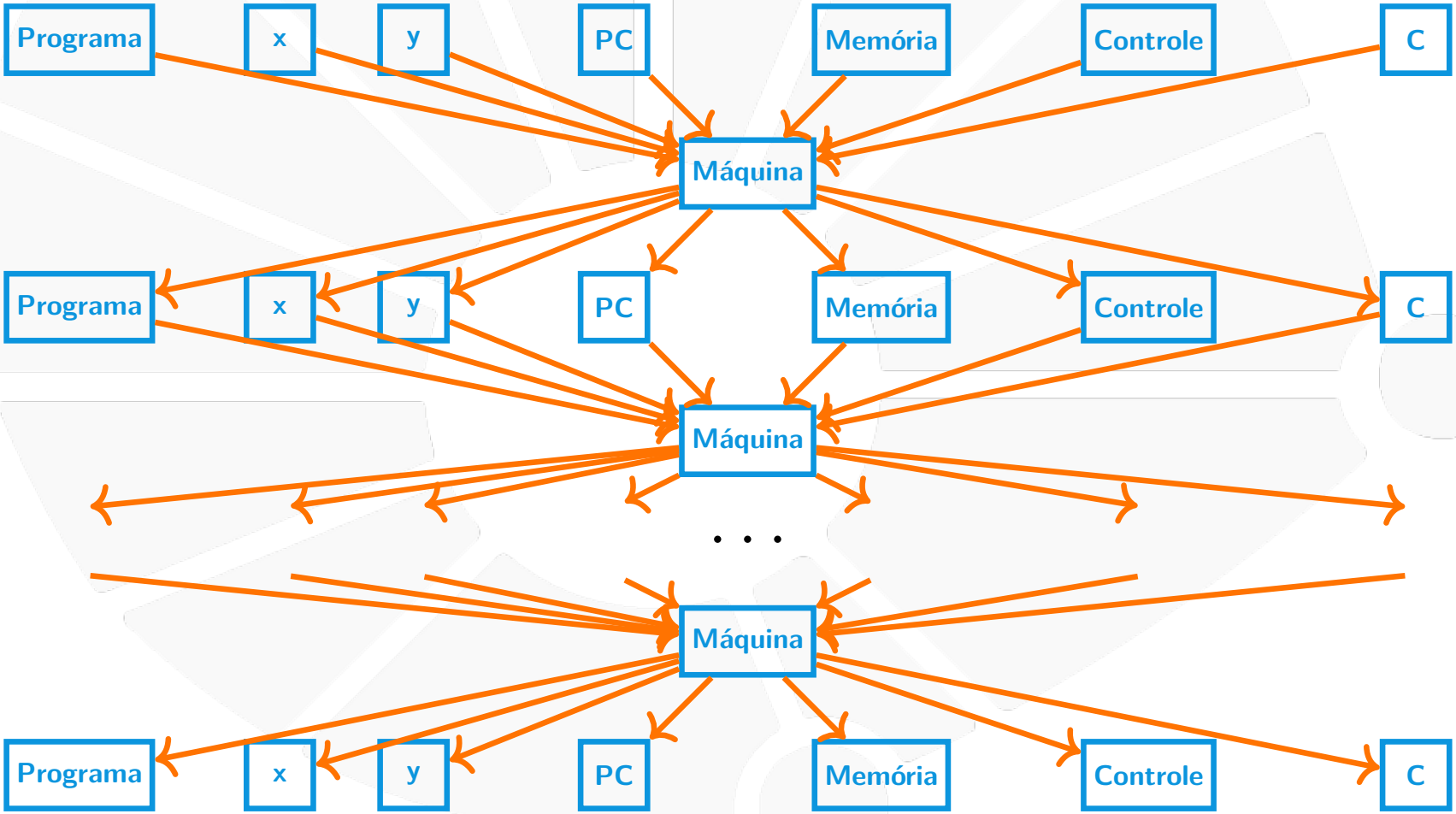


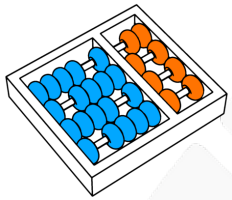
Continuação da demonstração

- Ajustamos o algoritmo A para que a saída seja escrita em um bit específico da memória denotado por C.
- Cada instrução executada pelo algoritmo A:
  - Começa em um estado de memória, PC e controle.
  - Modifica esse estado de memória, PC e controle.
- Criamos n^k cópias da máquina, uma para cada instrução:
  1. O estado de entrada da primeira máquina corresponde ao estado inicial da execução do algoritmo.
  2. A saída de uma instrução correspondente a uma cópia modifica o estado de entrada da cópia seguinte.
  3. A saída do circuito é a saída de uma conjunção (porta $\vee$) ligando os bits correspondentes a C.



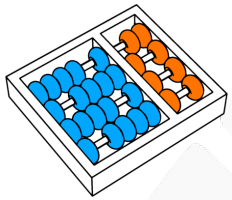
Continuação da demonstração





Continuação da demonstração

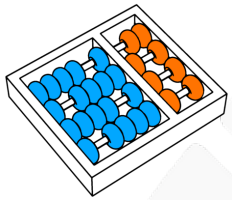
O circuito construído tem tamanho polinomial.



Continuação da demonstração

O circuito construído tem tamanho polinomial.

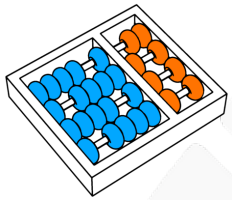
- O tamanho da máquina, controle, PC e A independem de x.



Continuação da demonstração

O circuito construído tem tamanho polinomial.

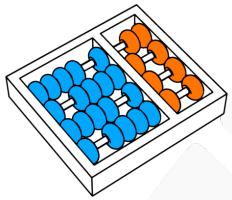
- O tamanho da máquina, controle, PC e A independem de x.
- A memória, y e x têm tamanho polinomial em $|x|$.



Continuação da demonstração

O circuito construído tem tamanho polinomial.

- O tamanho da máquina, controle, PC e A independem de x.
- A memória, y e x têm tamanho polinomial em $|x|$.
- Fazemos apenas n^k cópias da máquina.

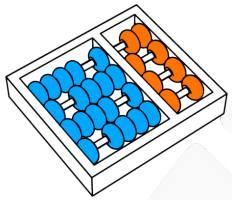


Continuação da demonstração

O circuito construído tem tamanho polinomial.

- O tamanho da máquina, controle, PC e A independem de x.
- A memória, y e x têm tamanho polinomial em $|x|$.
- Fazemos apenas n^k cópias da máquina.

Fios de entrada e de saída:



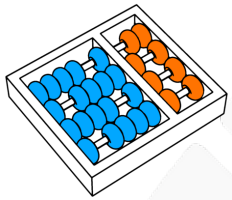
Continuação da demonstração

O circuito construído tem tamanho polinomial.

- O tamanho da máquina, controle, PC e A independem de x.
- A memória, y e x têm tamanho polinomial em $|x|$.
- Fazemos apenas n^k cópias da máquina.

Fios de entrada e de saída:

- Todo circuito é fixo e pode ser construído a partir de x.



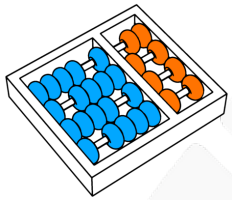
Continuação da demonstração

O circuito construído tem tamanho polinomial.

- O tamanho da máquina, controle, PC e A independem de x.
- A memória, y e x têm tamanho polinomial em $|x|$.
- Fazemos apenas n^k cópias da máquina.

Fios de entrada e de saída:

- Todo circuito é fixo e pode ser construído a partir de x.
- Há um **FIO DE ENTRADA** para cada bit do certificado y.



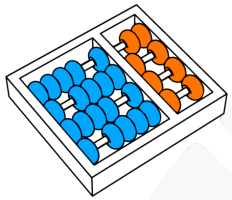
Continuação da demonstração

O circuito construído tem tamanho polinomial.

- O tamanho da máquina, controle, PC e A independem de x.
- A memória, y e x têm tamanho polinomial em $|x|$.
- Fazemos apenas n^k cópias da máquina.

Fios de entrada e de saída:

- Todo circuito é fixo e pode ser construído a partir de x.
- Há um **FIO DE ENTRADA** para cada bit do certificado y.
- O **FIO DE SAÍDA** vale 1 se algum campo C foi mudado para 1.



Continuação da demonstração

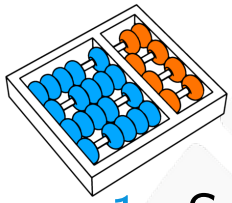
O circuito construído tem tamanho polinomial.

- O tamanho da máquina, controle, PC e A independem de x.
- A memória, y e x têm tamanho polinomial em $|x|$.
- Fazemos apenas n^k cópias da máquina.

Fios de entrada e de saída:

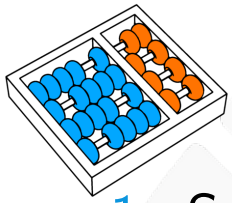
- Todo circuito é fixo e pode ser construído a partir de x.
- Há um **FIO DE ENTRADA** para cada bit do certificado y.
- O **FIO DE SAÍDA** vale 1 se algum campo C foi mudado para 1.

Como a redução levou tempo polinomial, falta mostrar apenas que $x \in Q$ se e somente se $F(x) \in$ C-SAT.



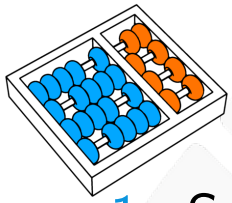
Continuação da demonstração

1. Suponha que $x \in Q$:



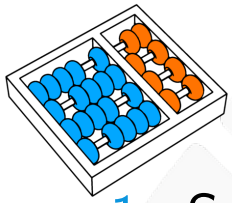
Continuação da demonstração

1. Suponha que $x \in Q$:
   - Então há certificado y tal que $A(x, y) = 1$.



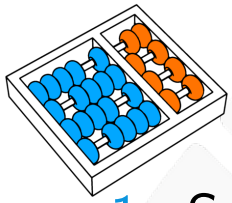
Continuação da demonstração

1. Suponha que $x \in Q$:
   - Então há certificado y tal que $A(x, y) = 1$.
   - Forneça y como entrada para o circuito $F(x)$.



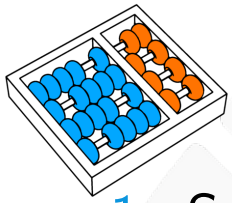
Continuação da demonstração

1. Suponha que $x \in Q$:
   - Então há certificado y tal que $A(x, y) = 1$.
   - Forneça y como entrada para o circuito $F(x)$.
   - Então, alguma cópia da máquina muda C para 1.



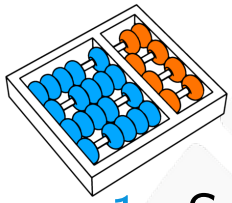
Continuação da demonstração

1. Suponha que $x \in Q$:
   - Então há certificado y tal que $A(x, y) = 1$.
   - Forneça y como entrada para o circuito $F(x)$.
   - Então, alguma cópia da máquina muda C para 1.
   - A saída do circuito será 1.



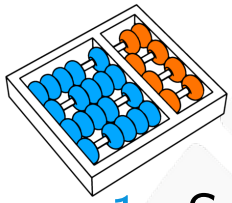
Continuação da demonstração

1. Suponha que $x \in Q$:
   - Então há certificado y tal que $A(x, y) = 1$.
   - Forneça y como entrada para o circuito $F(x)$.
   - Então, alguma cópia da máquina muda C para 1.
   - A saída do circuito será 1.
2. Suponha que o circuito $F(x)$ é satisfazível:



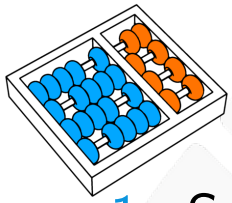
Continuação da demonstração

1. Suponha que $x \in Q$:
   - Então há certificado y tal que $A(x, y) = 1$.
   - Forneça y como entrada para o circuito $F(x)$.
   - Então, alguma cópia da máquina muda C para 1.
   - A saída do circuito será 1.
2. Suponha que o circuito $F(x)$ é satisfazível:
   - Então, para algum valor y dos fios de entrada a saída do circuito é 1.



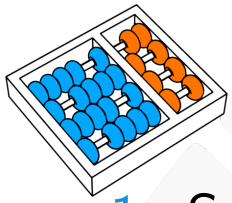
Continuação da demonstração

1. Suponha que $x \in Q$:
   - Então há certificado y tal que $A(x, y) = 1$.
   - Forneça y como entrada para o circuito $F(x)$.
   - Então, alguma cópia da máquina muda C para 1.
   - A saída do circuito será 1.
2. Suponha que o circuito $F(x)$ é satisfazível:
   - Então, para algum valor y dos fios de entrada a saída do circuito é 1.
   - Logo, para esse y, alguma cópia da máquina muda o campo C para 1.



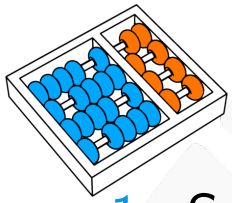
Continuação da demonstração

1. Suponha que $x \in Q$:
   - Então há certificado y tal que $A(x, y) = 1$.
   - Forneça y como entrada para o circuito $F(x)$.
   - Então, alguma cópia da máquina muda C para 1.
   - A saída do circuito será 1.
2. Suponha que o circuito $F(x)$ é satisfazível:
   - Então, para algum valor y dos fios de entrada a saída do circuito é 1.
   - Logo, para esse y, alguma cópia da máquina muda o campo C para 1.
   - Isso só acontece quando $A(x, y) = 1$.



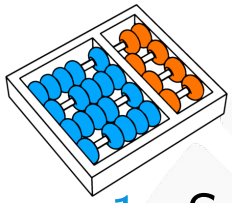
Continuação da demonstração

1. Suponha que $x \in Q$:
   - Então há certificado y tal que $A(x, y) = 1$.
   - Forneça y como entrada para o circuito $F(x)$.
   - Então, alguma cópia da máquina muda C para 1.
   - A saída do circuito será 1.
2. Suponha que o circuito $F(x)$ é satisfazível:
   - Então, para algum valor y dos fios de entrada a saída do circuito é 1.
   - Logo, para esse y, alguma cópia da máquina muda o campo C para 1.
   - Isso só acontece quando $A(x, y) = 1$.
   - Portanto, y é um certificado tal que $A(x, y) = 1$.



Continuação da demonstração

1. Suponha que $x \in Q$:
   - Então há certificado y tal que $A(x, y) = 1$.
   - Forneça y como entrada para o circuito $F(x)$.
   - Então, alguma cópia da máquina muda C para 1.
   - A saída do circuito será 1.
2. Suponha que o circuito $F(x)$ é satisfazível:
   - Então, para algum valor y dos fios de entrada a saída do circuito é 1.
   - Logo, para esse y, alguma cópia da máquina muda o campo C para 1.
   - Isso só acontece quando $A(x, y) = 1$.
   - Portanto, y é um certificado tal que $A(x, y) = 1$.
   - Assim, $x \in Q$.



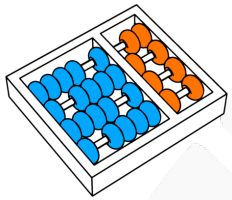
Continuação da demonstração

1. Suponha que $x \in Q$:
   - Então há certificado y tal que $A(x, y) = 1$.
   - Forneça y como entrada para o circuito $F(x)$.
   - Então, alguma cópia da máquina muda C para 1.
   - A saída do circuito será 1.
2. Suponha que o circuito $F(x)$ é satisfazível:
   - Então, para algum valor y dos fios de entrada a saída do circuito é 1.
   - Logo, para esse y, alguma cópia da máquina muda o campo C para 1.
   - Isso só acontece quando $A(x, y) = 1$.
   - Portanto, y é um certificado tal que $A(x, y) = 1$.
   - Assim, $x \in Q$.

Teorema

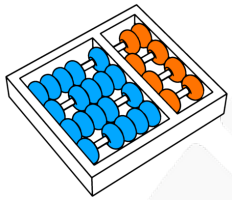
C-SAT é NP*-completo.*

Demonstrando NP-completude



Que outros problemas são NP-difíceis?

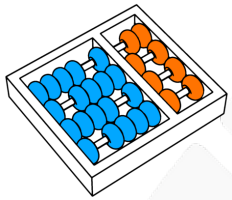
Já sabemos que C-SAT é NP-completo.



Que outros problemas são NP-difíceis?

Já sabemos que C-SAT é NP-completo.

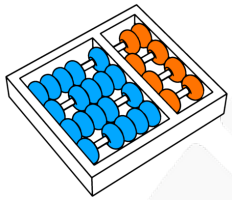
- Como descobrir se outro problema Q é NP-difícil?



Que outros problemas são NP-difíceis?

Já sabemos que C-SAT é NP-completo.

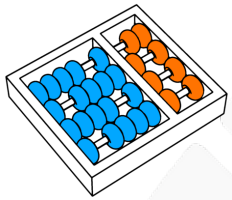
- Como descobrir se outro problema Q é NP-difícil?
- Temos duas possibilidades:



Que outros problemas são NP-difíceis?

Já sabemos que C-SAT é NP-completo.

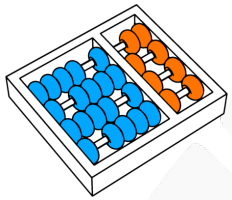
- Como descobrir se outro problema Q é NP-difícil?
- Temos duas possibilidades:
  1. Mostrar que $L \preccurlyeq_p Q$ para todo problema $L \in \text{NP}$.



Que outros problemas são NP-difíceis?

Já sabemos que C-SAT é NP-completo.

- Como descobrir se outro problema Q é NP-difícil?
- Temos duas possibilidades:
  1. Mostrar que $L \preccurlyeq_p Q$ para todo problema $L \in \text{NP}$.
  2. Mostrar que $L \preccurlyeq_p Q$ para algum problema $L \in \text{NP-difícil}$.

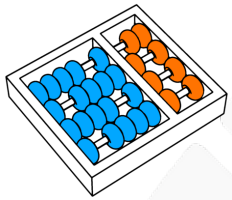


Que outros problemas são NP-difíceis?

Já sabemos que C-SAT é NP-completo.

- Como descobrir se outro problema Q é NP-difícil?
- Temos duas possibilidades:
  1. Mostrar que $L \preccurlyeq_p Q$ para todo problema $L \in \text{NP}$.
  2. Mostrar que $L \preccurlyeq_p Q$ para algum problema $L \in \text{NP-difícil}$.

Normalmente usamos a segunda opção:



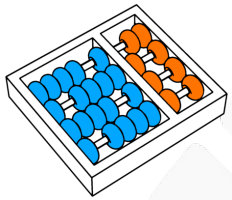
Que outros problemas são NP-difíceis?

Já sabemos que C-SAT é NP-completo.

- Como descobrir se outro problema Q é NP-difícil?
- Temos duas possibilidades:
  1. Mostrar que $L \preccurlyeq_p Q$ para todo problema $L \in \text{NP}$.
  2. Mostrar que $L \preccurlyeq_p Q$ para algum problema $L \in \text{NP-difícil}$.

Normalmente usamos a segunda opção:

- Basta reduzir um problema NP-difícil para o problema Q.



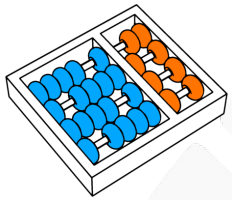
Que outros problemas são NP-difíceis?

Já sabemos que C-SAT é NP-completo.

- Como descobrir se outro problema Q é NP-difícil?
- Temos duas possibilidades:
  1. Mostrar que $L \preccurlyeq_p Q$ para todo problema $L \in \text{NP}$.
  2. Mostrar que $L \preccurlyeq_p Q$ para algum problema $L \in \text{NP-difícil}$.

Normalmente usamos a segunda opção:

- Basta reduzir um problema NP-difícil para o problema Q.
- Ou seja, mostramos que Q é **TÃO DIFÍCIL** quanto um NP-difícil.



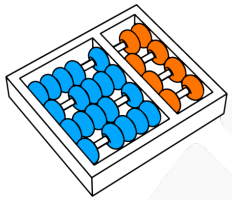
Que outros problemas são NP-difíceis?

Já sabemos que C-SAT é NP-completo.

- Como descobrir se outro problema Q é NP-difícil?
- Temos duas possibilidades:
  1. Mostrar que $L \preccurlyeq_p Q$ para todo problema $L \in \text{NP}$.
  2. Mostrar que $L \preccurlyeq_p Q$ para algum problema $L \in \text{NP-difícil}$.

Normalmente usamos a segunda opção:

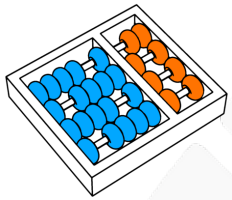
- Basta reduzir um problema NP-difícil para o problema Q.
- Ou seja, mostramos que Q é **TÃO DIFÍCIL** quanto um NP-difícil.
- Esse problema NP-difícil pode ser C-SAT, por exemplo.



NP-completude via redução

Teorema

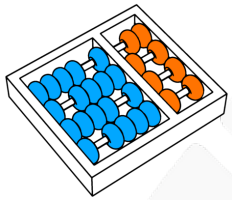
Considere uma linguagem Q e seja $L \in$ NP-difícil.



NP-completude via redução

Teorema

Considere uma linguagem Q e seja $L \in$ NP-difícil.
Se $L \preccurlyeq_p Q$, então Q é NP-*difícil.*

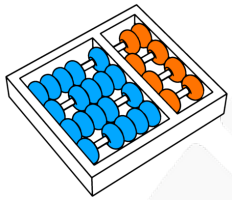


NP-completude via redução

Teorema

Considere uma linguagem Q e seja $L \in$ NP-difícil.
Se $L \preccurlyeq_p Q$, então Q é NP-*difícil.*

Demonstração:



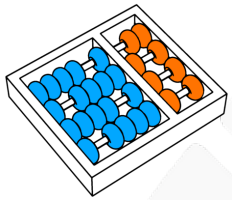
NP-completude via redução

Teorema

Considere uma linguagem Q e seja $L \in$ NP-difícil.
Se $L \preccurlyeq_p Q$, então Q é NP-*difícil.*

Demonstração:

- Como L é NP-difícil, para todo $L' \in$ NP temos $L' \preccurlyeq_p L$.



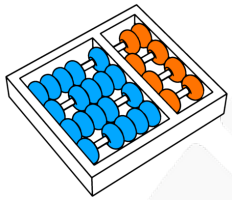
NP-completude via redução

Teorema

Considere uma linguagem Q e seja $L \in$ NP-difícil.
Se $L \preccurlyeq_p Q$, então Q é NP-*difícil.*

Demonstração:

- Como L é NP-difícil, para todo $L' \in$ NP temos $L' \preccurlyeq_p L$.
- Assim, $L' \preccurlyeq_p L$ e $L \preccurlyeq_p Q$, o que implica $L' \preccurlyeq_p Q$.



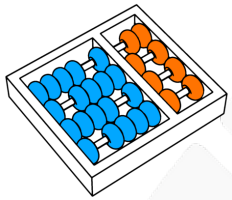
NP-completude via redução

Teorema

Considere uma linguagem Q e seja $L \in$ NP-difícil.
Se $L \preceq_p Q$, então Q é NP-difícil.

Demonstração:

- Como L é NP-difícil, para todo $L' \in$ NP temos $L' \preceq_p L$.
- Assim, $L' \preceq_p L$ e $L \preceq_p Q$, o que implica $L' \preceq_p Q$.
- Portanto Q é NP-difícil.



NP-completude via redução

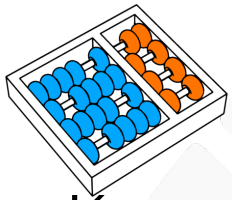
Teorema

Considere uma linguagem Q e seja $L \in$ NP-difícil.
Se $L \preccurlyeq_p Q$, então Q é NP-difícil.

Demonstração:

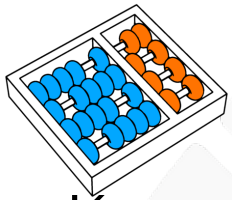
- Como L é NP-difícil, para todo $L' \in$ NP temos $L' \preccurlyeq_p L$.
- Assim, $L' \preccurlyeq_p L$ e $L \preccurlyeq_p Q$, o que implica $L' \preccurlyeq_p Q$.
- Portanto Q é NP-difícil.

Se além disso $Q \in$ NP, então Q é NP-completo.



Literatura

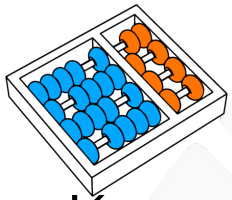
Karp mostrou em 1972 que 21 problemas são NP-completos:



Literatura

Karp mostrou em 1972 que 21 problemas são NP-completos:

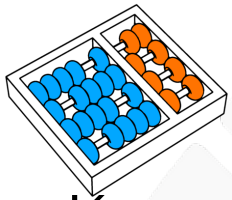
- Listou várias reduções de problemas NP-completos.



Literatura

Karp mostrou em 1972 que 21 problemas são NP-completos:

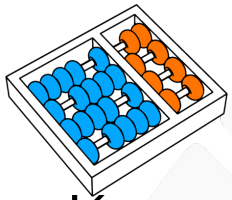
- Listou várias reduções de problemas NP-completos.
- As reduções induzem uma árvore com raiz em SAT.



Literatura

Karp mostrou em 1972 que 21 problemas são NP-completos:

- Listou várias reduções de problemas NP-completos.
- As reduções induzem uma árvore com raiz em SAT.
- veja a Wikipédia!

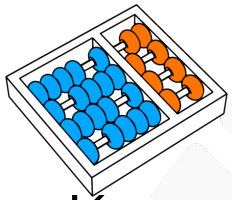


Literatura

Karp mostrou em 1972 que 21 problemas são NP-completos:

- Listou várias reduções de problemas NP-completos.
- As reduções induzem uma árvore com raiz em SAT.
- veja a Wikipédia!

Com sorte, seu problema foi estudado por Garey e Johnson:



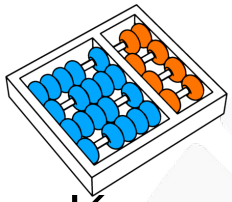
Literatura

Karp mostrou em 1972 que 21 problemas são NP-completos:

- Listou várias reduções de problemas NP-completos.
- As reduções induzem uma árvore com raiz em SAT.
- veja a Wikipédia!

Com sorte, seu problema foi estudado por Garey e Johnson:

- Livro publicado em 1979.



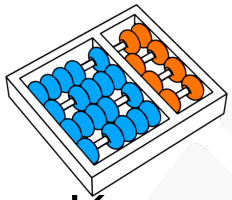
Literatura

Karp mostrou em 1972 que 21 problemas são NP-completos:

- Listou várias reduções de problemas NP-completos.
- As reduções induzem uma árvore com raiz em SAT.
- veja a Wikipédia!

Com sorte, seu problema foi estudado por Garey e Johnson:

- Livro publicado em 1979.
- Estuda e classifica dezenas de problemas.



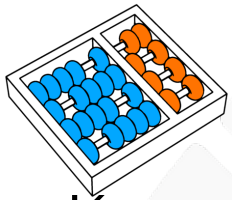
Literatura

Karp mostrou em 1972 que 21 problemas são NP-completos:

- Listou várias reduções de problemas NP-completos.
- As reduções induzem uma árvore com raiz em SAT.
- veja a Wikipédia!

Com sorte, seu problema foi estudado por Garey e Johnson:

- Livro publicado em 1979.
- Estuda e classifica dezenas de problemas.
- Entre os mais citados em Ciência da Computação.



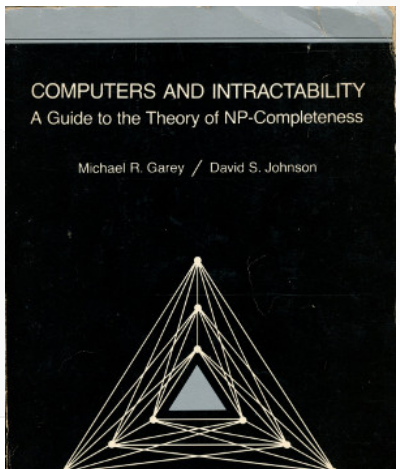
Literatura

Karp mostrou em 1972 que 21 problemas são NP-completos:

- Listou várias reduções de problemas NP-completos.
- As reduções induzem uma árvore com raiz em SAT.
- veja a Wikipédia!

Com sorte, seu problema foi estudado por Garey e Johnson:

- Livro publicado em 1979.
- Estuda e classifica dezenas de problemas.
- Entre os mais citados em Ciência da Computação.



NP-complexidade

MC558 - Projeto e Análise de Algoritmos II

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP