Reduções

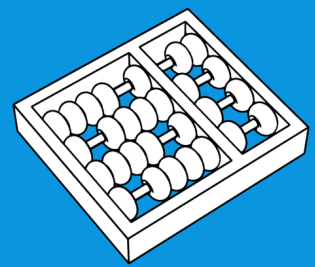
MC558 - Projeto e Análise de Algoritmos II

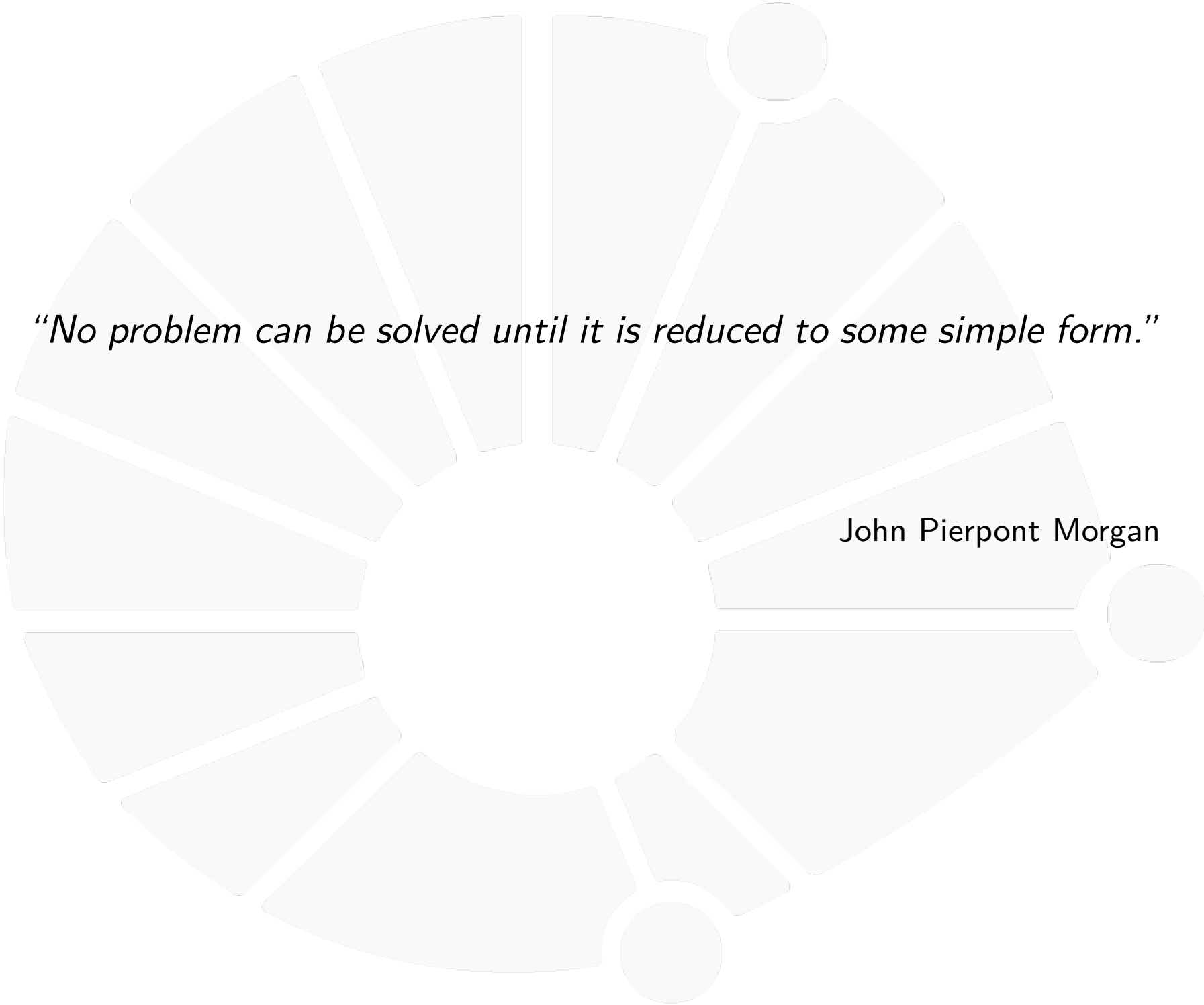
01/24

11

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

UNICAMP

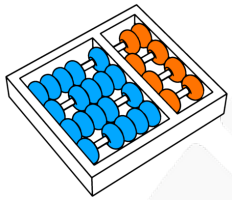




"No problem can be solved until it is reduced to some simple form."

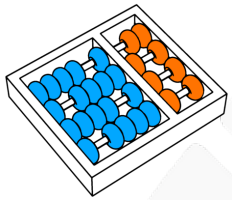
John Pierpont Morgan

Cotas para um problema



Formalizando problema

Como formalizar um problema **GENERICAMENTE**?

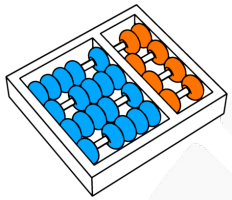


Formalizando problema

Como formalizar um problema **GENERICAMENTE**?

Definição (Problema computacional)

Um problema computacional é uma ***RELAÇÃO*** $P \subseteq \mathcal{I} \times \mathcal{S}$, *onde:*



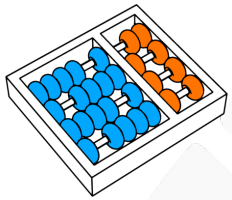
Formalizando problema

Como formalizar um problema **GENERICAMENTE**?

Definição (Problema computacional)

*Um problema computacional é uma **RELAÇÃO** $P \subseteq \mathcal{I} \times \mathcal{S}$, onde:*

- $\mathcal{I}$ *é o conjunto de entradas e*



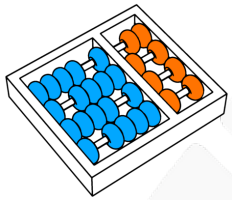
Formalizando problema

Como formalizar um problema **GENERICAMENTE**?

Definição (Problema computacional)

*Um problema computacional é uma **RELAÇÃO** $P \subseteq \mathcal{I} \times \mathcal{S}$, onde:*

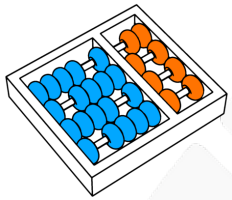
- $\mathcal{I}$ *é o conjunto de entradas e*
- $\mathcal{S}$ *é o conjunto de saídas.*



Algoritmo para um problema

Definição

Dizemos que um algoritmo ALG ***RESOLVE*** *um problema* $P = (\mathcal{I}, \mathcal{P})$ *se para toda entrada* $I \in \mathcal{I}$*, ele devolve uma saída* $S \in \mathcal{S}$ *tal que* $(I, S) \in P$*.*

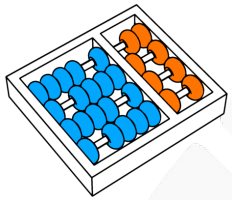


Algoritmo para um problema

Definição

Dizemos que um algoritmo ALG ***RESOLVE*** *um problema* $P = (\mathcal{I}, \mathcal{P})$ *se para toda entrada* $I \in \mathcal{I}$*, ele devolve uma saída* $S \in \mathcal{S}$ *tal que* $(I, S) \in P$*.*

- Escrevemos $I \in P$ para representar uma entrada.

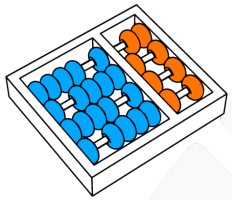


Algoritmo para um problema

Definição

Dizemos que um algoritmo ALG ***RESOLVE*** *um problema* $P = (\mathcal{I}, \mathcal{P})$ *se para toda entrada* $I \in \mathcal{I}$*, ele devolve uma saída* $S \in \mathcal{S}$ *tal que* $(I, S) \in P$*.*

- Escrevemos $I \in P$ para representar uma entrada.
- Escrevemos $A(I)$ para representar a saída do algoritmo.

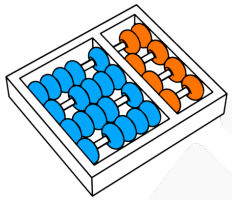


Algoritmo para um problema

Definição

Dizemos que um algoritmo ALG ***RESOLVE*** *um problema* $P = (\mathcal{I}, \mathcal{P})$ *se para toda entrada* $I \in \mathcal{I}$*, ele devolve uma saída* $S \in \mathcal{S}$ *tal que* $(I, S) \in P$*.*

- Escrevemos $I \in P$ para representar uma entrada.
- Escrevemos $A(I)$ para representar a saída do algoritmo.
- Denotamos por n o "tamanho" de I.

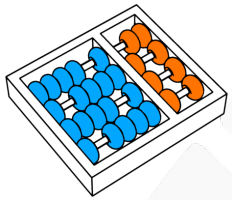


Algoritmo para um problema

Definição

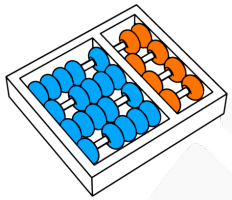
Dizemos que um algoritmo ALG ***RESOLVE*** *um problema* $P = (\mathcal{I}, \mathcal{P})$ *se para toda entrada* $I \in \mathcal{I}$*, ele devolve uma saída* $S \in \mathcal{S}$ *tal que* $(I, S) \in P$*.*

- Escrevemos $I \in P$ para representar uma entrada.
- Escrevemos $A(I)$ para representar a saída do algoritmo.
- Denotamos por n o "tamanho" de I.
- Normalmente n é o números de bits de I.



Revisitando a complexidade de um algoritmo

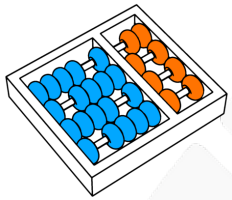
Seja $\textsc{Alg}$ um algoritmo para um problema P e n um parâmetro.



Revisitando a complexidade de um algoritmo

Seja ALG um algoritmo para um problema P e n um parâmetro.

Notação O:

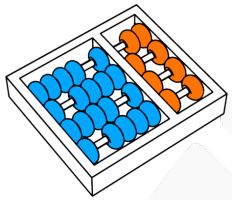


Revisitando a complexidade de um algoritmo

Seja ALG um algoritmo para um problema P e n um parâmetro.

Notação O:

- Se o algoritmo leva tempo **NO MÁXIMO** $f(n)$ para toda entrada de tamanho n, então dizemos que ALG executa em tempo $O(f(n))$.



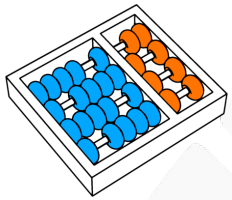
Revisitando a complexidade de um algoritmo

Seja $\textsc{Alg}$ um algoritmo para um problema P e n um parâmetro.

Notação O:

- Se o algoritmo leva tempo **NO MÁXIMO** $f(n)$ para toda entrada de tamanho n, então dizemos que $\textsc{Alg}$ executa em tempo $O(f(n))$.

Notação Ω:



Revisitando a complexidade de um algoritmo

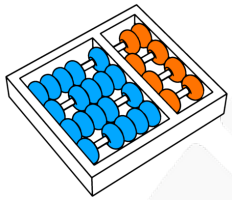
Seja $\textsc{Alg}$ um algoritmo para um problema P e n um parâmetro.

Notação O:

- Se o algoritmo leva tempo **NO MÁXIMO** $f(n)$ para toda entrada de tamanho n, então dizemos que $\textsc{Alg}$ executa em tempo $O(f(n))$.

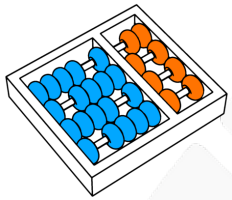
Notação Ω:

- Se o algoritmo leva tempo **PELO MENOS** $g(n)$ para alguma entrada de tamanho n, então dizemos que $\textsc{Alg}$ executa em tempo $\Omega(g(n))$.



Cotas superior e inferior de um problema

Seja P um problema e seja n um parâmetro:

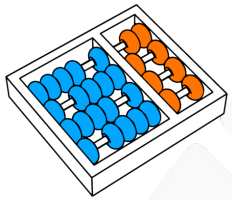


Cotas superior e inferior de um problema

Seja P um problema e seja n um parâmetro:

Definição (Cota superior)

Uma função $f(n)$ é chamada de cota superior para P se ***existe algum algoritmo*** *que resolve P em tempo $O(f(n))$.*



Cotas superior e inferior de um problema

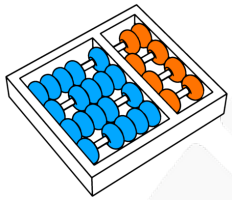
Seja P um problema e seja n um parâmetro:

Definição (Cota superior)

Uma função $f(n)$ é chamada de cota superior para P se ***existe algum algoritmo*** *que resolve P em tempo $O(f(n))$.*

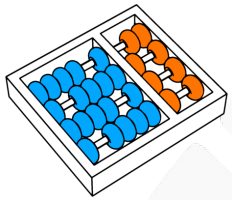
Definição (Cota inferior)

Uma função $g(n)$ é chamada de cota inferior para P se ***todo algoritmo*** *que resolve P leva tempo $\Omega(f(n))$.*



Algoritmo ótimo

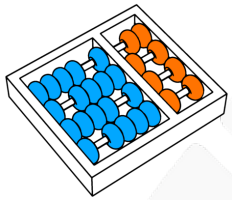
Um algoritmo ALG é **ÓTIMO** para um problema P se:



Algoritmo ótimo

Um algoritmo ALG é **ÓTIMO** para um problema P se:

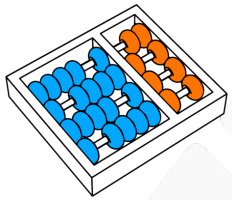
1. ALG resolve P em tempo $O(f(n))$ e



Algoritmo ótimo

Um algoritmo ALG é **ÓTIMO** para um problema P se:

1. ALG resolve P em tempo $O(f(n))$ e
2. $f(n)$ é uma cota inferior de P.

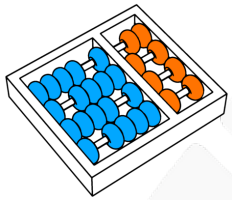


Algoritmo ótimo

Um algoritmo ALG é **ÓTIMO** para um problema P se:

1. ALG resolve P em tempo $O(f(n))$ e
2. $f(n)$ é uma cota inferior de P.

- HEAPSORT e MERGESORT são ótimos para ordenação:

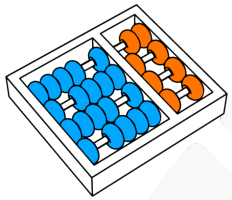


Algoritmo ótimo

Um algoritmo ALG é **ÓTIMO** para um problema P se:

1. ALG resolve P em tempo $O(f(n))$ e
2. $f(n)$ é uma cota inferior de P.

- HEAPSORT e MERGESORT são ótimos para ordenação:
  - Eles têm complexidade $O(n \log n)$.

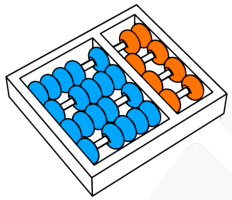


Algoritmo ótimo

Um algoritmo ALG é **ÓTIMO** para um problema P se:

1. ALG resolve P em tempo $O(f(n))$ e
2. $f(n)$ é uma cota inferior de P.

- HEAPSORT e MERGESORT são ótimos para ordenação:
  - Eles têm complexidade $O(n \log n)$.
  - Ordenação tem cota inferior $\Omega(n \log n)$.

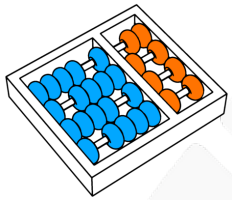


Algoritmo ótimo

Um algoritmo ALG é **ÓTIMO** para um problema P se:

1. ALG resolve P em tempo $O(f(n))$ e
2. $f(n)$ é uma cota inferior de P.

- HEAPSORT e MERGESORT são ótimos para ordenação:
  - Eles têm complexidade $O(n \log n)$.
  - Ordenação tem cota inferior $\Omega(n \log n)$.
- BUSCA-BINARIA é ótimo para busca em vetor ordenado:



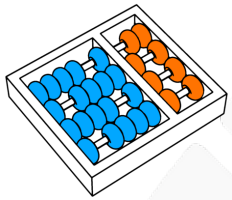
Algoritmo ótimo

Um algoritmo ALG é **ÓTIMO** para um problema P se:

1. ALG resolve P em tempo $O(f(n))$ e
2. $f(n)$ é uma cota inferior de P.

- HEAPSORT e MERGESORT são ótimos para ordenação:
  - Eles têm complexidade $O(n \log n)$.
  - Ordenação tem cota inferior $\Omega(n \log n)$.

- BUSCA-BINARIA é ótimo para busca em vetor ordenado:
  - Tem complexidade $O(\log n)$.



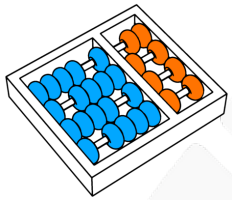
Algoritmo ótimo

Um algoritmo ALG é **ÓTIMO** para um problema P se:

1. ALG resolve P em tempo $O(f(n))$ e
2. $f(n)$ é uma cota inferior de P.

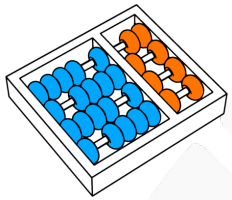
- HEAPSORT e MERGESORT são ótimos para ordenação:
  - Eles têm complexidade $O(n \log n)$.
  - Ordenação tem cota inferior $\Omega(n \log n)$.

- BUSCA-BINARIA é ótimo para busca em vetor ordenado:
  - Tem complexidade $O(\log n)$.
  - Qualquer algoritmo leva tempo $\Omega(\log n)$.



Comparando problemas

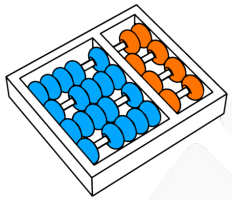
Como comparamos dois algoritmos para **UM ÚNICO PROBLEMA**?



Comparando problemas

Como comparamos dois algoritmos para **UM ÚNICO PROBLEMA**?

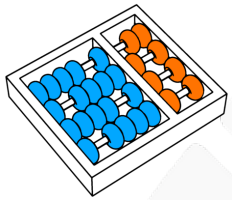
- Comparamos a complexidade de cada algoritmo.



Comparando problemas

Como comparamos dois algoritmos para **UM ÚNICO PROBLEMA**?

- Comparamos a complexidade de cada algoritmo.
- Descobrimos se um algoritmo é mais "rápido" que outro.

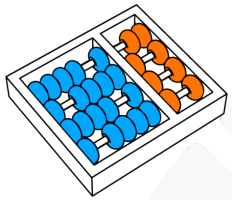


Comparando problemas

Como comparamos dois algoritmos para **UM ÚNICO PROBLEMA**?

- Comparamos a complexidade de cada algoritmo.
- Descobrimos se um algoritmo é mais "rápido" que outro.

E se quisermos comparar **DOIS PROBLEMAS** A e B?



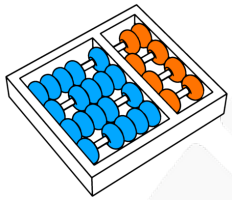
Comparando problemas

Como comparamos dois algoritmos para **UM ÚNICO PROBLEMA**?

- Comparamos a complexidade de cada algoritmo.
- Descobrimos se um algoritmo é mais "rápido" que outro.

E se quisermos comparar **DOIS PROBLEMAS** A e B?

- Queremos descobrir se então A é mais "fácil" do que B.



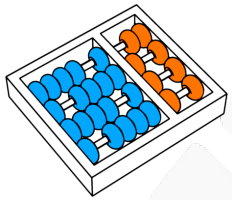
Comparando problemas

Como comparamos dois algoritmos para **UM ÚNICO PROBLEMA**?

- Comparamos a complexidade de cada algoritmo.
- Descobrimos se um algoritmo é mais "rápido" que outro.

E se quisermos comparar **DOIS PROBLEMAS** A e B?

- Queremos descobrir se então A é mais "fácil" do que B.
- Podemos comparar as cotas de cada algoritmo.



Comparando problemas

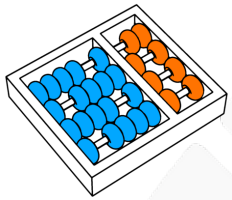
Como comparamos dois algoritmos para **UM ÚNICO PROBLEMA**?

- Comparamos a complexidade de cada algoritmo.
- Descobrimos se um algoritmo é mais "rápido" que outro.

E se quisermos comparar **DOIS PROBLEMAS** A e B?

- Queremos descobrir se então A é mais "fácil" do que B.
- Podemos comparar as cotas de cada algoritmo.

Exemplo: Achar o máximo é mais **FÁCIL** que ordenar um vetor!



Comparando problemas

Como comparamos dois algoritmos para **UM ÚNICO PROBLEMA**?

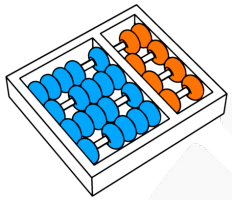
- Comparamos a complexidade de cada algoritmo.
- Descobrimos se um algoritmo é mais "rápido" que outro.

E se quisermos comparar **DOIS PROBLEMAS** A e B?

- Queremos descobrir se então A é mais "fácil" do que B.
- Podemos comparar as cotas de cada algoritmo.

Exemplo: Achar o máximo é mais **FÁCIL** que ordenar um vetor!

- Máximo tem cota superior $O(n)$.



Comparando problemas

Como comparamos dois algoritmos para **UM ÚNICO PROBLEMA**?

- Comparamos a complexidade de cada algoritmo.
- Descobrimos se um algoritmo é mais "rápido" que outro.

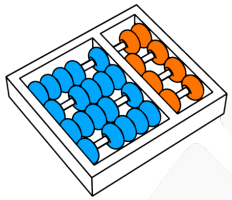
E se quisermos comparar **DOIS PROBLEMAS** A e B?

- Queremos descobrir se então A é mais "fácil" do que B.
- Podemos comparar as cotas de cada algoritmo.

Exemplo: Achar o máximo é mais **FÁCIL** que ordenar um vetor!

- Máximo tem cota superior $O(n)$.
- Ordenação tem cota inferior $\Omega(n \log n)$.

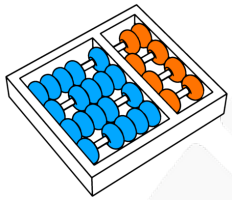
Reduções



Combinando problemas

Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

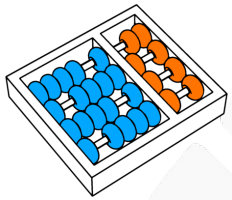


Combinando problemas

Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- Cook, responsável por traduzir de **inglês para português**.

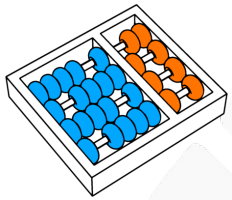


Combinando problemas

Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- Cook, responsável por traduzir de **inglês para português**.
- Levin, responsável por traduzir de **russo para inglês**.



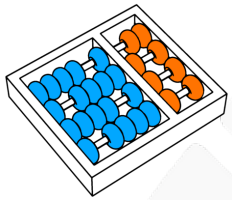
Combinando problemas

Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- Cook, responsável por traduzir de **inglês para português**.
- Levin, responsável por traduzir de **russo para inglês**.

Em uma edição especial, ele irá publicar Crime e Castigo.



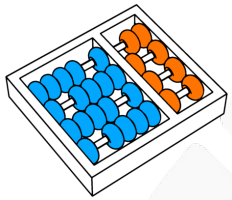
Combinando problemas

Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- Cook, responsável por traduzir de **inglês para português**.
- Levin, responsável por traduzir de **russo para inglês**.

Em uma edição especial, ele irá publicar Crime e Castigo.
Como traduzir de **RUSSO PARA PORTUGUÊS**?



Combinando problemas

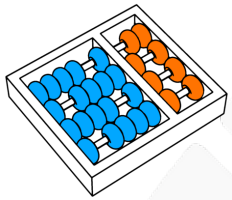
Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- Cook, responsável por traduzir de **inglês para português**.
- Levin, responsável por traduzir de **russo para inglês**.

Em uma edição especial, ele irá publicar Crime e Castigo.
Como traduzir de **RUSSO PARA PORTUGUÊS**?

- Em geral, lidamos com problemas bem conhecidos.



Combinando problemas

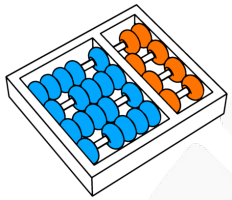
Uma analogia

Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- Cook, responsável por traduzir de **inglês para português**.
- Levin, responsável por traduzir de **russo para inglês**.

Em uma edição especial, ele irá publicar Crime e Castigo.
Como traduzir de **RUSSO PARA PORTUGUÊS**?

- Em geral, lidamos com problemas bem conhecidos.
- Mas eventualmente, topamos com problemas novos.



Combinando problemas

Uma analogia

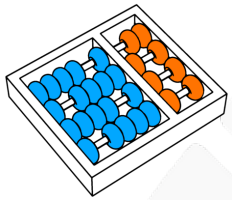
Um certo editor de literatura internacional é especialista em publicar livros em português. Ele conta com um time de tradutores, entre os quais:

- Cook, responsável por traduzir de **inglês para português**.
- Levin, responsável por traduzir de **russo para inglês**.

Em uma edição especial, ele irá publicar Crime e Castigo.
Como traduzir de **RUSSO PARA PORTUGUÊS**?

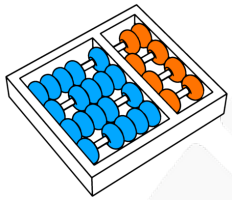
- Em geral, lidamos com problemas bem conhecidos.
- Mas eventualmente, topamos com problemas novos.

  Pergunta: como relacionar esses problemas?



Redução

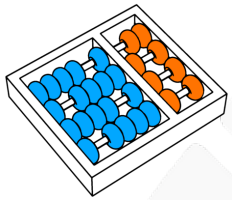
Problema A:



Redução

Problema A:

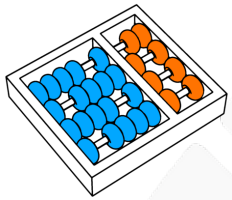
- Instância: I_A



Redução

Problema A:

- Instância: I_A
- Solução: S_A

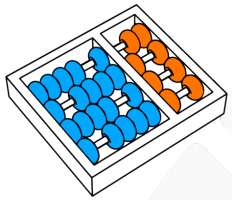


Redução

Problema A:

- Instância: I_A
- Solução: S_A

Problema B:



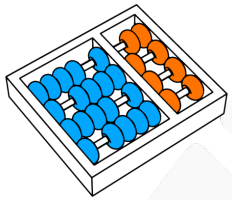
Redução

Problema A:

- Instância: I_A
- Solução: S_A

Problema B:

- Instância: I_B



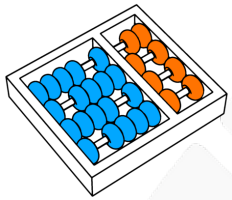
Redução

Problema A:

- Instância: I_A
- Solução: S_A

Problema B:

- Instância: I_B
- Solução: S_B



Redução

Problema A:

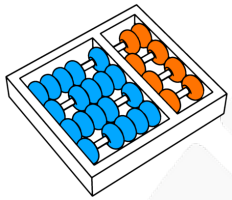
- Instância: I_A
- Solução: S_A

Problema B:

- Instância: I_B
- Solução: S_B

Definição

*Uma **REDUÇÃO** do problema A ao problema B é um par de sub-rotinas τ_I e τ_S tais que:*



Redução

Problema A:

- Instância: I_A
- Solução: S_A

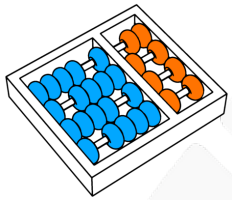
Problema B:

- Instância: I_B
- Solução: S_B

Definição

*Uma **REDUÇÃO** do problema A ao problema B é um par de sub-rotinas τ_I e τ_S tais que:*

- τ_I *transforma uma instância I_A de A em uma instância I_B de B.*



Redução

Problema A:

- Instância: I_A
- Solução: S_A

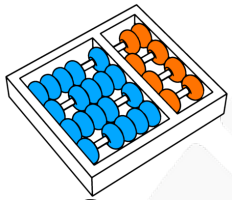
Problema B:

- Instância: I_B
- Solução: S_B

Definição

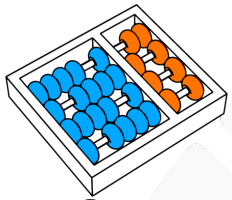
*Uma **REDUÇÃO** do problema A ao problema B é um par de sub-rotinas τ_I e τ_S tais que:*

- τ_I *transforma uma instância I_A de A em uma instância I_B de B.*
- τ_S *transforma uma solução S_B de I_B em uma solução S_A de I_A.*



Redução como um algoritmo

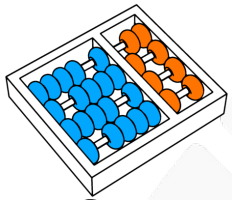
Como podemos resolver o problema A?



Redução como um algoritmo

Como podemos resolver o problema A?

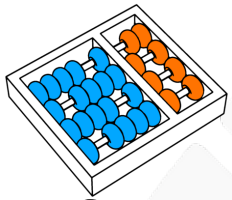
1. Suponha que existe um algoritmo $\textsc{Alg}_{\mathbf{B}}$ para o problema B.



Redução como um algoritmo

Como podemos resolver o problema A?

1. Suponha que existe um algoritmo $\textsc{Alg}_{\mathbf{B}}$ para o problema B.
2. Podemos usar $\textsc{Alg}_{\mathbf{B}}$ como uma **CAIXA-PRETA**.



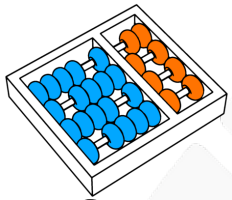
Redução como um algoritmo

Como podemos resolver o problema A?

1. Suponha que existe um algoritmo $\textsc{Alg}_{\mathbf{B}}$ para o problema B.
2. Podemos usar $\textsc{Alg}_{\mathbf{B}}$ como uma **CAIXA-PRETA**.

Algoritmo 4: $\textsc{Redução}(I, A)$

1 $I_B \leftarrow \tau_I(I_A)$
2 $S_B \leftarrow \textsc{Alg}_{\mathbf{B}}(\mathbf{I_B})$
3 $S_A \leftarrow \tau_S(I_A, S_B)$
4 **devolva** S_A



Redução como um algoritmo

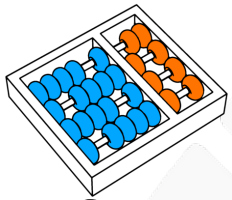
Como podemos resolver o problema A?

1. Suponha que existe um algoritmo $\textsc{Alg}_{\mathbf{B}}$ para o problema B.
2. Podemos usar $\textsc{Alg}_{\mathbf{B}}$ como uma **CAIXA-PRETA**.

Algoritmo 5: $\textsc{Redução}(I, A)$

1 $I_B \leftarrow \tau_I(I_A)$
2 $S_B \leftarrow \textsc{Alg}_{\mathbf{B}}(\mathbf{I_B})$
3 $S_A \leftarrow \tau_S(I_A, S_B)$
4 **devolva** S_A

Em outras palavras:



Redução como um algoritmo

Como podemos resolver o problema A?

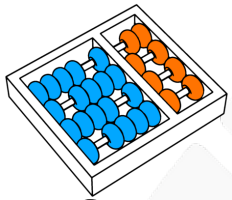
1. Suponha que existe um algoritmo $\textsc{Alg}_{\mathbf{B}}$ para o problema B.
2. Podemos usar $\textsc{Alg}_{\mathbf{B}}$ como uma **CAIXA-PRETA**.

Algoritmo 6: $\textsc{Redução}(I, A)$

1 $I_B \leftarrow \tau_I(I_A)$
2 $S_B \leftarrow \textsc{Alg}_{\mathbf{B}}(\mathbf{I_B})$
3 $S_A \leftarrow \tau_S(I_A, S_B)$
4 **devolva** S_A

Em outras palavras:

- Se sabemos resolver B, então também sei resolver A!



Redução como um algoritmo

Como podemos resolver o problema A?

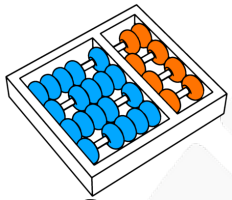
1. Suponha que existe um algoritmo $\textsc{Alg}_{\mathbf{B}}$ para o problema B.
2. Podemos usar $\textsc{Alg}_{\mathbf{B}}$ como uma **CAIXA-PRETA**.

Algoritmo 7: $\textsc{Redução}(I, A)$

1 $I_B \leftarrow \tau_I(I_A)$
2 $S_B \leftarrow \textsc{Alg}_{\mathbf{B}}(\mathbf{I_B})$
3 $S_A \leftarrow \tau_S(I_A, S_B)$
4 **devolva** S_A

Em outras palavras:

- Se sabemos resolver B, então também sei resolver A!
- A é mais “fácil"que B.



Redução como um algoritmo

Como podemos resolver o problema A?

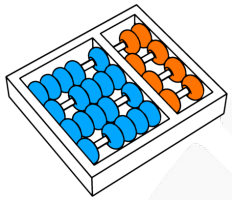
1. Suponha que existe um algoritmo $\textsc{Alg}_{\mathbf{B}}$ para o problema B.
2. Podemos usar $\textsc{Alg}_{\mathbf{B}}$ como uma **CAIXA-PRETA**.

Algoritmo 8: $\textsc{Redução}(I, A)$

1 $I_B \leftarrow \tau_I(I_A)$
2 $S_B \leftarrow \textsc{Alg}_{\mathbf{B}}(\mathbf{I_B})$
3 $S_A \leftarrow \tau_S(I_A, S_B)$
4 **devolva** S_A

Em outras palavras:

- Se sabemos resolver B, então também sei resolver A!
- A é mais “fácil"que B.
- Denotamos $\mathbf{A \preccurlyeq B}$.

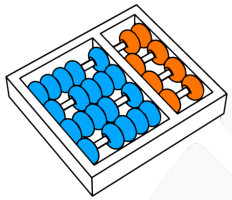


Um problema de origem

Problema (Alocação de Centros (AC))

Entrada: *Um grafo bipartido conexo* $G = ((X \cup Y), E)$ *e uma função de pesos nas arestas* $w : E \to \mathbb{R}_+$.

Saída: *Uma função* $\phi : X \to Y$ *que aloque cada vértice* v *em* X *a um vértice* $\phi[v]$ *em* Y *tal que o peso* $w(v, \phi[v])$ *seja* ***MÍNIMO***.

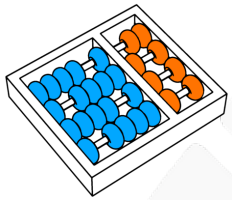


Um problema de destino

Problema (Caminho Mínimo (CM))

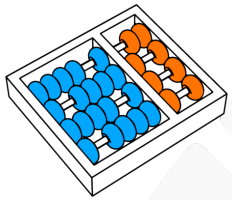
Entrada: *Um grafo direcionado acíclico* $G = (V, E)$*, uma função de peso* $w : E \to \mathbb{R}_+$ *nas arestas e um vértice origem origem* s*.*

Saída: *Um vetor* d *com* $d[v] = \text{dist}(s, v)$ *para* $v \in V$ *e um vetor* π *definindo uma* ***ÁRVORE DE CAMINHOS MÍNIMOS****.*



Reduzindo. Transformação da entrada

Recebemos uma **ENTRADA** do problema de origem AC:

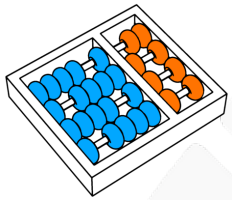


Reduzindo. Transformação da entrada

Recebemos uma **ENTRADA** do problema de origem AC:

Algoritmo 10: $\tau_I(G, w)$

1 $G' \leftarrow G$
2 $w' \leftarrow w$
3 Adicione um novo vértice s a G'
4 **para cada** $v \in Y$
5 Adicione a aresta (s, v) a G'
6 $w'(s, v) \leftarrow 0$
7 **devolva** (G', w', s)

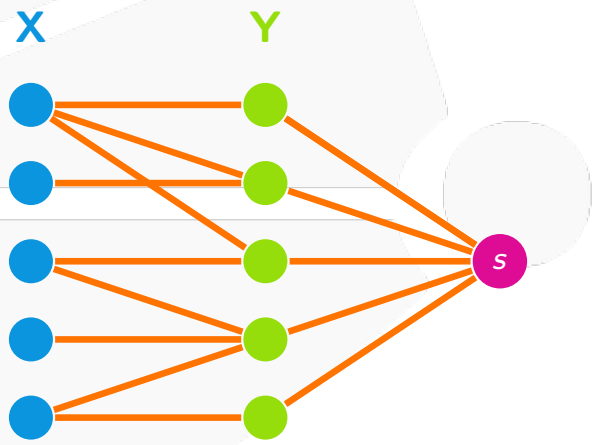


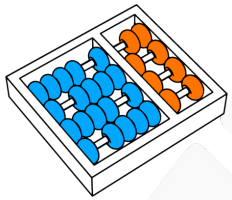
Reduzindo. Transformação da entrada

Recebemos uma **ENTRADA** do problema de origem AC:

Algoritmo 11: $\tau_I(G, w)$

1 $G' \leftarrow G$
2 $w' \leftarrow w$
3 Adicione um novo vértice s a G'
4 **para cada** $v \in Y$
5 Adicione a aresta (s, v) a G'
6 $w'(s, v) \leftarrow 0$
7 **devolva** (G', w', s)



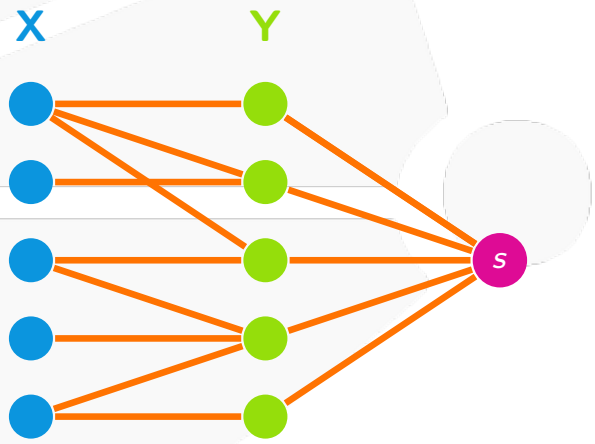


Reduzindo. Transformação da entrada

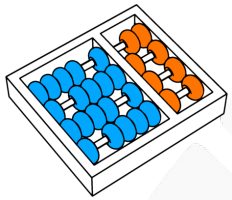
Recebemos uma **ENTRADA** do problema de origem AC:

Algoritmo 12: $\tau_I(G, w)$

1 $G' \leftarrow G$
2 $w' \leftarrow w$
3 Adicione um novo vértice s a G'
4 **para cada** $v \in Y$
5 Adicione a aresta (s, v) a G'
6 $w'(s, v) \leftarrow 0$
7 **devolva** (G', w', s)

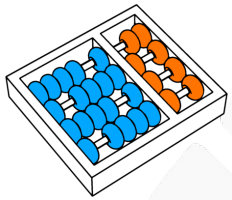


Tempo: $O(Y)$.



Reduzindo. Transformação da saída

Também recebemos uma **SOLUÇÃO** do problema de destino CM:

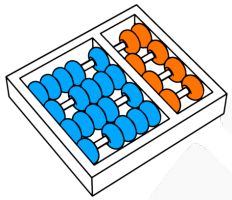


Reduzindo. Transformação da saída

Também recebemos uma **SOLUÇÃO** do problema de destino CM:

Algoritmo 14: $\tau_S(G, w, \mathbf{d}, \pi)$

1 **para cada** $v \in X$
2 $\quad \phi[v] \leftarrow \pi[v]$
3 **devolva** ϕ

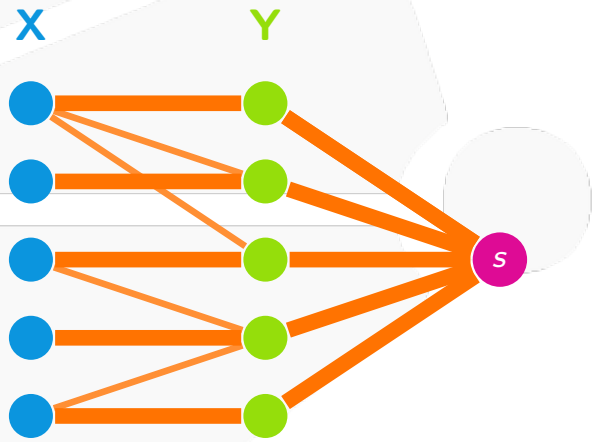


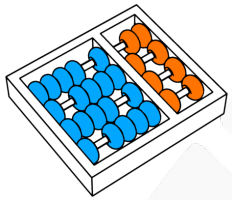
Reduzindo. Transformação da saída

Também recebemos uma **SOLUÇÃO** do problema de destino CM:

Algoritmo 15: $\tau_S(G, w, \mathbf{d}, \pi)$

1 **para cada** $v \in X$
2 $\quad \phi[v] \leftarrow \pi[v]$
3 **devolva** ϕ



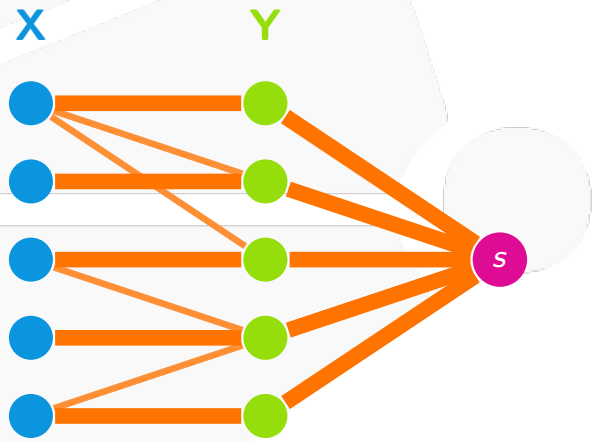


Reduzindo. Transformação da saída

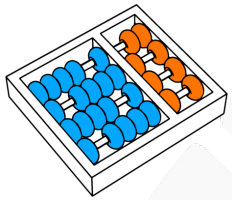
Também recebemos uma **SOLUÇÃO** do problema de destino CM:

Algoritmo 16: $\tau_S(G, w, \mathbf{d}, \pi)$

1 **para cada** $v \in X$
2 $\quad \phi[v] \leftarrow \pi[v]$
3 **devolva** ϕ

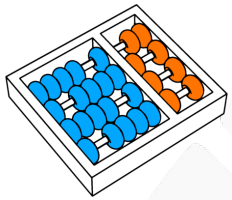


Tempo: $O(X)$.



Reduzindo. AC $\preceq$ CM

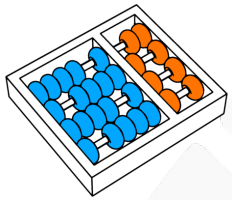
Seja ALG_{CM} um algoritmo para Caminho Mínimo.



Reduzindo. AC $\prec$ CM

Seja $\textsc{Alg}_{\mathsf{CM}}$ um algoritmo para Caminho Mínimo.

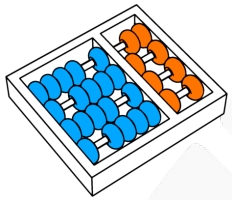
- $\textsc{Alg}_{\mathsf{CM}}$ poderia ser Dijkstra, Bellman-Ford...



Reduzindo. AC $\asymp$ CM

Seja ALG_{CM} um algoritmo para Caminho Mínimo.

- ALG_{CM} poderia ser Dijkstra, Bellman-Ford...
- Pode ser que **NÃO CONHEÇAMOS** um algoritmo para o problema de destino!



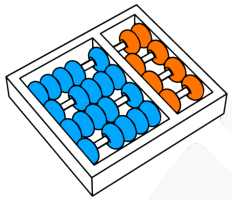
Reduzindo. AC $\preceq$ CM

Seja $\textsc{Alg}_{\text{CM}}$ um algoritmo para Caminho Mínimo.

- $\textsc{Alg}_{\text{CM}}$ poderia ser $\textsc{Dijkstra}$, $\textsc{Bellman-Ford}$...
- Pode ser que **NÃO CONHEÇAMOS** um algoritmo para o problema de destino!

Algoritmo 20: $\textsc{Redução-AC-CM}(G, w)$

1 $(G', w', s) \leftarrow \tau_I(G, w)$
2 $(d, \pi) \leftarrow \textsc{Alg}_{\text{CM}}(G', w', s)$
3 $\phi \leftarrow \tau_S(G, w, d, \pi)$
4 **devolva** ϕ



Reduzindo. AC $\preccurlyeq$ CM

Seja $\textsc{Alg}_{\text{CM}}$ um algoritmo para Caminho Mínimo.

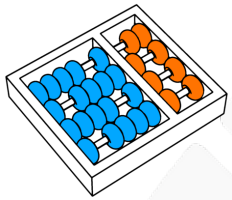
- $\textsc{Alg}_{\text{CM}}$ poderia ser $\textsc{Dijkstra}$, $\textsc{Bellman-Ford}$...
- Pode ser que **NÃO CONHEÇAMOS** um algoritmo para o problema de destino!

Algoritmo 21: $\textsc{Redução-AC-CM}(G, w)$

---

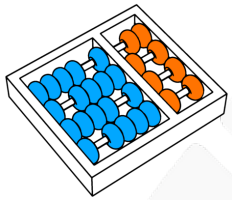
1 $(G', w', s) \leftarrow \tau_I(G, w)$
2 $(d, \pi) \leftarrow \textsc{Alg}_{\text{CM}}(G', w', s)$
3 $\phi \leftarrow \tau_S(G, w, d, \pi)$
4 **devolva** ϕ

Tempo total: [tempo da redução] + [tempo de$\textsc{Alg}_{\text{CM}}$]



Tempo da redução

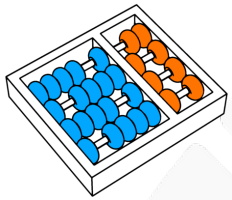
Quanto tempo gastamos só com a redução?



Tempo da redução

Quanto tempo gastamos só com a redução?

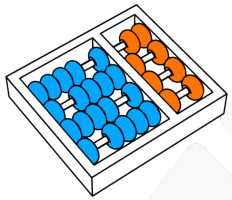
- Não contamos o tempo do algoritmo para o problema B.



Tempo da redução

Quanto tempo gastamos só com a redução?

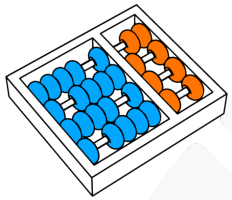
- Não contamos o tempo do algoritmo para o problema B.
- A **COMPLEXIDADE DE UMA REDUÇÃO** $f(n)$ é a soma dos tempos das transformações τ_I e τ_S.



Tempo da redução

Quanto tempo gastamos só com a redução?

- Não contamos o tempo do algoritmo para o problema B.
- A **COMPLEXIDADE DE UMA REDUÇÃO** $f(n)$ é a soma dos tempos das transformações τ_I e τ_S.
- Escrevemos $\mathbf{A \preccurlyeq_{f(n)} B}$.

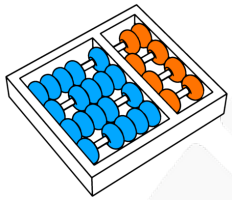


Tempo da redução

Quanto tempo gastamos só com a redução?

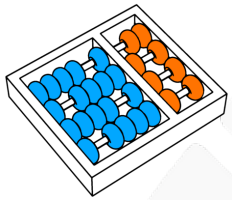
- Não contamos o tempo do algoritmo para o problema B.
- A **COMPLEXIDADE DE UMA REDUÇÃO** $f(n)$ é a soma dos tempos das transformações τ_I e τ_S.
- Escrevemos $\mathbf{A} \preccurlyeq_{\mathbf{f(n)}} \mathbf{B}$.

No caso de Redução-AC-CM: $\mathbf{AC} \preccurlyeq_{|\mathbf{X}|+|\mathbf{Y}|} \mathbf{CM}$.



Reduções polinomiais

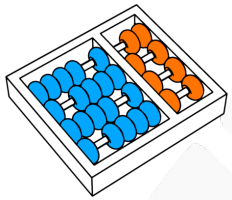
Queremos construir algoritmos rápidos. Mas, o que é “rápido”?



Reduções polinomiais

Queremos construir algoritmos rápidos. Mas, o que é “rápido”?

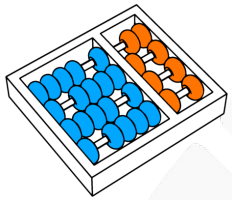
- Normalmente, dizemos que um algoritmo é rápido se ele executa em **TEMPO POLINOMIAL**.



Reduções polinomiais

Queremos construir algoritmos rápidos. Mas, o que é "rápido"?

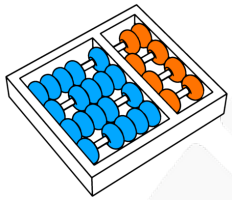
- Normalmente, dizemos que um algoritmo é rápido se ele executa em **TEMPO POLINOMIAL**.
- Daí, queremos reduções de tempo polinomial.



Reduções polinomiais

Queremos construir algoritmos rápidos. Mas, o que é "rápido"?

- Normalmente, dizemos que um algoritmo é rápido se ele executa em **TEMPO POLINOMIAL**.
- Daí, queremos reduções de tempo polinomial.
- Nesse caso, escrevemos $\mathbf{A} \preccurlyeq_{\text{poli}} \mathbf{B}$.

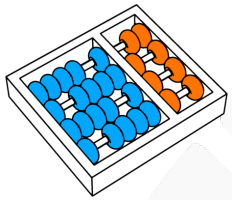


Reduções polinomiais

Queremos construir algoritmos rápidos. Mas, o que é "rápido"?

- Normalmente, dizemos que um algoritmo é rápido se ele executa em **TEMPO POLINOMIAL**.
- Daí, queremos reduções de tempo polinomial.
- Nesse caso, escrevemos $\mathbf{A} \preccurlyeq_{\text{poli}} \mathbf{B}$.

Qual a consequência de $A \preccurlyeq_{\text{poli}} B$?



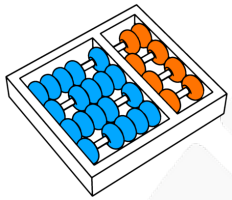
Reduções polinomiais

Queremos construir algoritmos rápidos. Mas, o que é "rápido"?

- Normalmente, dizemos que um algoritmo é rápido se ele executa em **TEMPO POLINOMIAL**.
- Daí, queremos reduções de tempo polinomial.
- Nesse caso, escrevemos $\mathbf{A} \preccurlyeq_{\text{poli}} \mathbf{B}$.

Qual a consequência de $A \preccurlyeq_{\text{poli}} B$?

1. Se B tem um algoritmo de tempo polinomial, então A também.



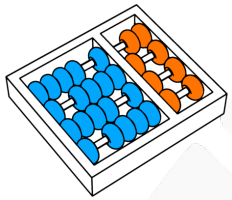
Reduções polinomiais

Queremos construir algoritmos rápidos. Mas, o que é "rápido"?

- Normalmente, dizemos que um algoritmo é rápido se ele executa em **TEMPO POLINOMIAL**.
- Daí, queremos reduções de tempo polinomial.
- Nesse caso, escrevemos $\mathbf{A} \preccurlyeq_{\text{poli}} \mathbf{B}$.

Qual a consequência de $A \preccurlyeq_{\text{poli}} B$?

1. Se B tem um algoritmo de tempo polinomial, então A também.
2. Se A **NÃO** tem algoritmos de tempo polinomial, tampouco B.



Reduções polinomiais

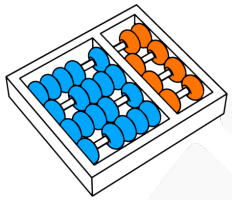
Queremos construir algoritmos rápidos. Mas, o que é "rápido"?

- Normalmente, dizemos que um algoritmo é rápido se ele executa em **TEMPO POLINOMIAL**.
- Daí, queremos reduções de tempo polinomial.
- Nesse caso, escrevemos $\mathbf{A} \preccurlyeq_{\text{poli}} \mathbf{B}$.

Qual a consequência de $A \preccurlyeq_{\text{poli}} B$?

1. Se B tem um algoritmo de tempo polinomial, então A também.
2. Se A **NÃO** tem algoritmos de tempo polinomial, tampouco B.

- Isso é útil para distinguir problemas fáceis de difíceis!



Reduções polinomiais

Queremos construir algoritmos rápidos. Mas, o que é "rápido"?

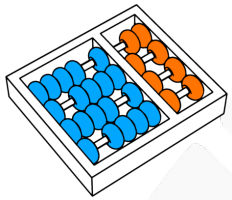
- Normalmente, dizemos que um algoritmo é rápido se ele executa em **TEMPO POLINOMIAL**.
- Daí, queremos reduções de tempo polinomial.
- Nesse caso, escrevemos $\mathbf{A} \preccurlyeq_{\text{poli}} \mathbf{B}$.

Qual a consequência de $A \preccurlyeq_{\text{poli}} B$?

1. Se B tem um algoritmo de tempo polinomial, então A também.
2. Se A **NÃO** tem algoritmos de tempo polinomial, tampouco B.

- Isso é útil para distinguir problemas fáceis de difíceis!
- Mas, é assunto para depois...

Exemplos de reduções



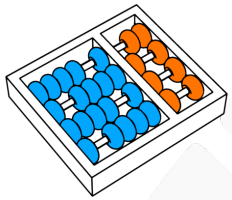
Problema de origem

Problema (Sistema Linear (LS))

Entrada: *Uma matriz M de dimensões $n \times n$ com determinante* ***NÃO*** *nulo e um vetor b de dimensão n.*

Saída: *Um vetor x de dimensão n que satisfaz o seguinte sistema linear:*

$$Mx = b.$$



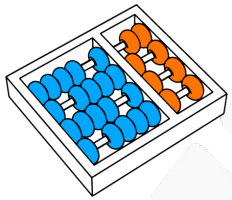
Problema de destino

Problema (Sistema Linear Simétrico (SLS))

Entrada: *Uma matriz M* ***SIMÉTRICA*** *de dimensões $n \times n$ com determinante* ***NÃO*** *nulo e um vetor b de dimensão n.*

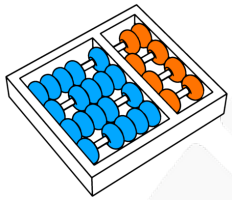
Saída: *Um vetor x de dimensão n que satisfaz o seguinte sistema linear:*

$$Mx = b.$$



Perguntas

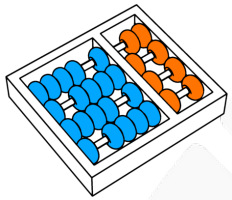
SLS é um caso particular de LS.



Perguntas

SLS é um caso particular de LS.

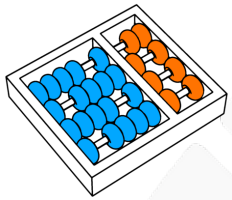
- Logo, trivialmente SLS $\propto$ LS.



Perguntas

SLS é um caso particular de LS.

- Logo, trivialmente SLS $\preceq$ LS.
- Será que LS é estritamente mais difícil?

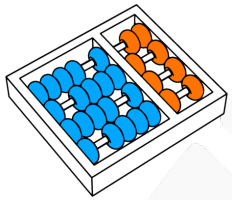


Perguntas

SLS é um caso particular de LS.

- Logo, trivialmente SLS $\preceq$ LS.
- Será que LS é estritamente mais difícil?

A resposta é **NÃO!**



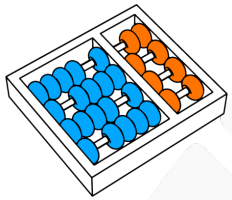
Perguntas

SLS é um caso particular de LS.

- Logo, trivialmente SLS $\preceq$ LS.
- Será que LS é estritamente mais difícil?

A resposta é **NÃO!**

- Iremos reduzir LS para SLS.



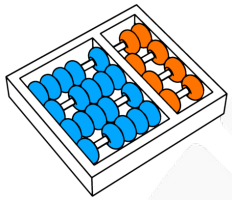
Perguntas

SLS é um caso particular de LS.

- Logo, trivialmente SLS $\preceq$ LS.
- Será que LS é estritamente mais difícil?

A resposta é **NÃO!**

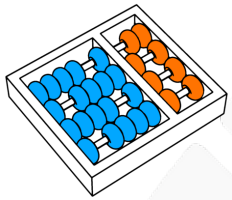
- Iremos reduzir LS para SLS.
- Isso é, LS $\preceq$ SLS.



Um fato simples

Lema

Um vetor x é solução de $Mx = b$ se, e só se, x é solução de $M^T Mx = M^T b$.

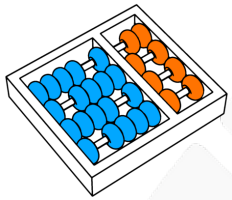


Um fato simples

Lema

Um vetor x é solução de $Mx = b$ se, e só se, x é solução de $M^T Mx = M^T b$.

Demonstração:

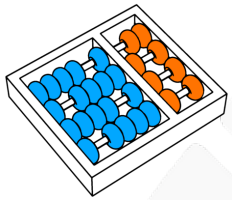


Um fato simples

Lema

Um vetor x é solução de $Mx = b$ se, e só se, x é solução de $M^T Mx = M^T b$.

Demonstração:



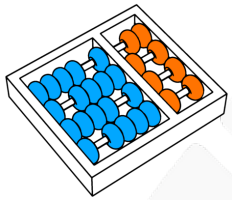
Um fato simples

Lema

Um vetor x é solução de $Mx = b$ se, e só se, x é solução de $M^T Mx = M^T b$.

Demonstração:

($\Rightarrow$)
- Multiplicamos $Mx = b$ por M^T.



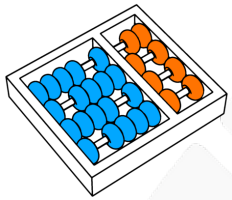
Um fato simples

Lema

Um vetor x é solução de $Mx = b$ se, e só se, x é solução de $M^T Mx = M^T b$.

Demonstração:

($\Rightarrow$)
- Multiplicamos $Mx = b$ por M^T.
- Obtemos $M^T Mx = M^T b$.



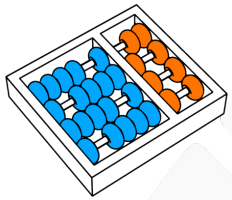
Um fato simples

Lema

Um vetor x é solução de $Mx = b$ se, e só se, x é solução de $M^T Mx = M^T b$.

Demonstração:

($\Rightarrow$)
- Multiplicamos $Mx = b$ por M^T.
- Obtemos $M^T Mx = M^T b$.



Um fato simples

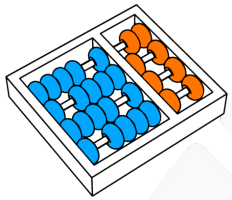
Lema

Um vetor x é solução de $Mx = b$ se, e só se, x é solução de $M^T Mx = M^T b$.

Demonstração:

($\Rightarrow$)
- Multiplicamos $Mx = b$ por M^T.
- Obtemos $M^T Mx = M^T b$.

($\Leftarrow$)
- M^T tem determinante não nulo.



Um fato simples

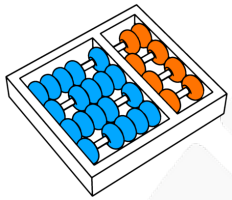
Lema

Um vetor x é solução de $Mx = b$ se, e só se, x é solução de $M^T Mx = M^T b$.

Demonstração:

($\Rightarrow$)
- Multiplicamos $Mx = b$ por M^T.
- Obtemos $M^T Mx = M^T b$.

($\Leftarrow$)
- M^T tem determinante não nulo.
- Logo, M^T tem inversa Z.



Um fato simples

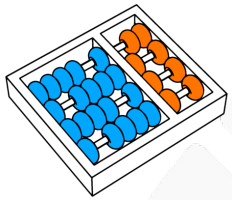
Lema

Um vetor x é solução de $Mx = b$ se, e só se, x é solução de $M^T Mx = M^T b$.

Demonstração:

($\Rightarrow$)
- Multiplicamos $Mx = b$ por M^T.
- Obtemos $M^T Mx = M^T b$.

($\Leftarrow$)
- M^T tem determinante não nulo.
- Logo, M^T tem inversa Z.
- Multiplicamos $M^T Mx = M^T b$ por Z.



Um fato simples

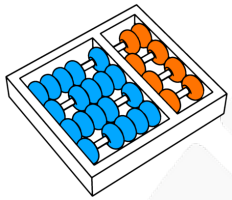
Lema

Um vetor x é solução de $Mx = b$ se, e só se, x é solução de $M^T Mx = M^T b$.

Demonstração:

($\Rightarrow$)
- Multiplicamos $Mx = b$ por M^T.
- Obtemos $M^T Mx = M^T b$.

($\Leftarrow$)
- M^T tem determinante não nulo.
- Logo, M^T tem inversa Z.
- Multiplicamos $M^T Mx = M^T b$ por Z.
- Obtendo $Mx = b$



Um fato simples

Lema

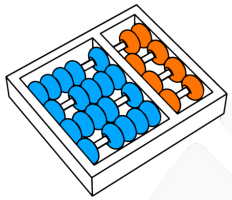
Um vetor x é solução de $Mx = b$ se, e só se, x é solução de $M^T Mx = M^T b$.

Demonstração:

($\Rightarrow$)
- Multiplicamos $Mx = b$ por M^T.
- Obtemos $M^T Mx = M^T b$.

($\Leftarrow$)
- M^T tem determinante não nulo.
- Logo, M^T tem inversa Z.
- Multiplicamos $M^T Mx = M^T b$ por Z.
- Obtendo $Mx = b$

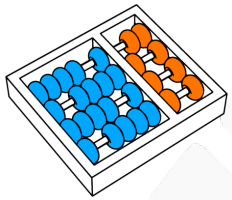
Observe que $M' = M^T M$ é uma matriz simétrica!



LS $\propto$ SLS

Algoritmo 22: REDUÇÃO-LS-SLS(M, b)

1 $M' \leftarrow M^T M$
2 $b' \leftarrow M^T b$
3 $x \leftarrow \text{ALG}_{\text{SLS}}(M', b')$
4 **devolva** x

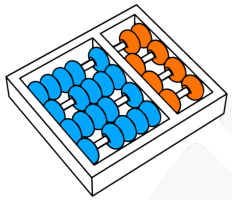


LS $\preccurlyeq$ SLS

Algoritmo 23: REDUÇÃO-LS-SLS(M, b)

1 $M' \leftarrow M^T M$
2 $b' \leftarrow M^T b$
3 $x \leftarrow \text{ALG}_{\text{SLS}}(M', b')$
4 **devolva** x

Concluímos que de fato LS $\preccurlyeq$ SLS.

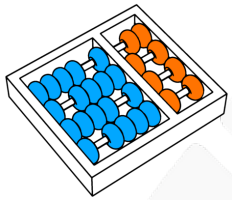


Problema de origem

Problema (Casamento cíclico de strings (CSM))

Entrada: *Um alfabeto* Σ*, uma cadeia* $A = a_0 a_1 \ldots a_{n-1}$ *com* n *símbolos e uma cadeia* $B = b_0 b_1 \ldots b_{n-1}$ *com* n *símbolos.*

Saída: SIM, *se* B *for um* ***DESLOCAMENTO CÍCLICO*** *de* A*, ou* NAO, *caso contrário. Se* SIM, *então o número* k *de letras deslocadas faz parte da saída.*



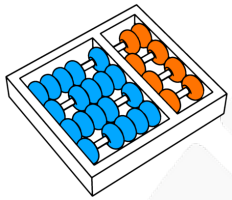
Problema de origem

Problema (Casamento cíclico de strings (CSM))

Entrada: *Um alfabeto* Σ*, uma cadeia* $A = a_0 a_1 \ldots a_{n-1}$ *com* n *símbolos e uma cadeia* $B = b_0 b_1 \ldots b_{n-1}$ *com* n *símbolos.*

Saída: SIM, *se* B *for um* ***DESLOCAMENTO CÍCLICO*** *de* A*, ou* NAO, *caso contrário. Se* SIM, *então o número* k *de letras deslocadas faz parte da saída.*

Exemplo:



Problema de origem

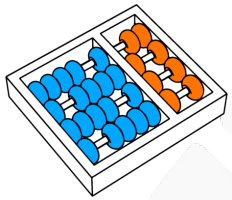
Problema (Casamento cíclico de strings (CSM))

Entrada: *Um alfabeto* Σ*, uma cadeia* $A = a_0 a_1 \ldots a_{n-1}$ *com* n *símbolos e uma cadeia* $B = b_0 b_1 \ldots b_{n-1}$ *com* n *símbolos.*

Saída: SIM, *se* B *for um* ***DESLOCAMENTO CÍCLICO*** *de* A*, ou* NAO, *caso contrário. Se* SIM, *então o número* k *de letras deslocadas faz parte da saída.*

Exemplo:

- Entrada: $A = acgtact$ e $B = gtactac$



Problema de origem

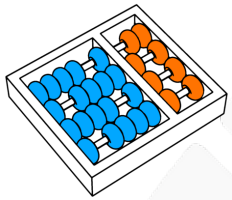
Problema (Casamento cíclico de strings (CSM))

Entrada: *Um alfabeto* Σ*, uma cadeia* $A = a_0 a_1 \ldots a_{n-1}$ *com* n *símbolos e uma cadeia* $B = b_0 b_1 \ldots b_{n-1}$ *com* n *símbolos.*

Saída: SIM, *se* B *for um* ***DESLOCAMENTO CÍCLICO*** *de* A*, ou* NAO, *caso contrário. Se* SIM, *então o número* k *de letras deslocadas faz parte da saída.*

Exemplo:

- Entrada: $A = acgtact$ e $B = gtactac$
- Saída: SIM, $k = 2$

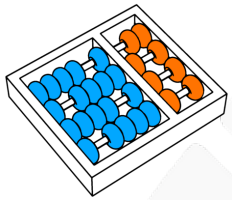


Problema de destino

Problema (Casamento de strings (SM))

Entrada: *UM alfabeto* Σ*, uma cadeia* $A = a_0 a_1 \ldots a_{n-1}$ *com* n *símbolos e uma cadeia* $B = b_0 b_1 \ldots b_{m-1}$ *com* m *símbolos.*

Saída: SIM, *se* B *for* ***SUBCADEIA*** *de* A*, ou* NAO, *caso contrário. Se* SIM, *o índice* k *da primeira ocorrência de* B *em* A*, faz parte da saída.*



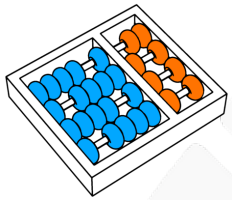
Problema de destino

Problema (Casamento de strings (SM))

Entrada: *UM alfabeto* Σ*, uma cadeia* $A = a_0 a_1 \ldots a_{n-1}$ *com* n *símbolos e uma cadeia* $B = b_0 b_1 \ldots b_{m-1}$ *com* m *símbolos.*

Saída: SIM, *se* B *for* ***SUBCADEIA*** *de* A*, ou* NAO, *caso contrário. Se* SIM, *o índice* k *da primeira ocorrência de* B *em* A*, faz parte da saída.*

Exemplo:



Problema de destino

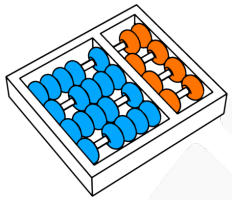
Problema (Casamento de strings (SM))

Entrada: *UM alfabeto* Σ*, uma cadeia* $A = a_0 a_1 \ldots a_{n-1}$ *com* n *símbolos e uma cadeia* $B = b_0 b_1 \ldots b_{m-1}$ *com* m *símbolos.*

Saída: SIM, *se* B *for* ***SUBCADEIA*** *de* A*, ou* NAO, *caso contrário. Se* SIM, *o índice* k *da primeira ocorrência de* B *em* A*, faz parte da saída.*

Exemplo:

- Entrada: $A = acgttaccgtacccg$ e $B = tac$



Problema de destino

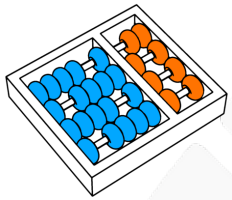
Problema (Casamento de strings (SM))

Entrada: *UM alfabeto* Σ*, uma cadeia* $A = a_0 a_1 \ldots a_{n-1}$ *com* n *símbolos e uma cadeia* $B = b_0 b_1 \ldots b_{m-1}$ *com* m *símbolos.*

Saída: SIM, *se* B *for* ***SUBCADEIA*** *de* A*, ou* NAO, *caso contrário. Se* SIM, *o índice* k *da primeira ocorrência de* B *em* A*, faz parte da saída.*

Exemplo:

- Entrada: $A = acgttaccgtacccg$ e $B = tac$
- Saída: SIM, $k = 4$



Problema de destino

Problema (Casamento de strings (SM))

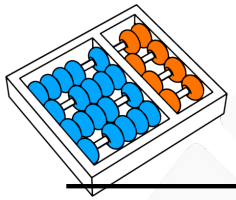
Entrada: *UM alfabeto* Σ*, uma cadeia* $A = a_0 a_1 \ldots a_{n-1}$ *com* n *símbolos e uma cadeia* $B = b_0 b_1 \ldots b_{m-1}$ *com* m *símbolos.*

Saída: SIM, *se* B *for* ***SUBCADEIA*** *de* A*, ou* NAO, *caso contrário. Se* SIM, *o índice* k *da primeira ocorrência de* B *em* A*, faz parte da saída.*

Exemplo:

- Entrada: $A = acgttaccgtacccg$ e $B = tac$
- Saída: SIM, $k = 4$

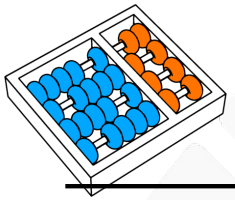
Observação: o problema SM pode ser resolvido em tempo $O(n + m)$ pelo algoritmo KMP de Knuth, Morris and Pratt (1977).



CSM $\asymp$ SM

Algoritmo 24: $\textsc{Redução-CSM-SM}(A, B, n)$

1 $A' \leftarrow AA$ ▷ concatena duas cópias de A
2 $B' \leftarrow B$
3 $n' \leftarrow 2n$
4 $m' \leftarrow n$
5 **devolva** $\textsc{Alg}_{\mathsf{SM}}(A', n', B', m')$

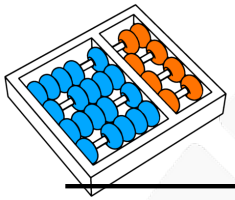


CSM $\asymp$ SM

Algoritmo 25: $\textsc{Redução-CSM-SM}(A, B, n)$

1 $A' \leftarrow AA$ ▷ concatena duas cópias de A
2 $B' \leftarrow B$
3 $n' \leftarrow 2n$
4 $m' \leftarrow n$
5 **devolva** $\textsc{Alg}_{\mathsf{SM}}(A', n', B', m')$

- **Tempo da redução:** $O(n)$

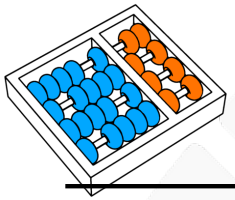


CSM $\preceq$ SM

Algoritmo 26: $\textsc{Redução-CSM-SM}(A, B, n)$

1 $A' \leftarrow AA$ ▷ concatena duas cópias de A
2 $B' \leftarrow B$
3 $n' \leftarrow 2n$
4 $m' \leftarrow n$
5 **devolva** $\textsc{Alg}_{\text{SM}}(A', n', B', m')$

- **Tempo da redução:** $O(n)$
- **Correção:** basta mostrar que se k é a solução de SM, então k também é solução de CSM.



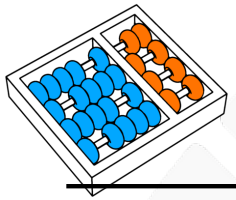
CSM $\asymp$ SM

Algoritmo 27: $\textsc{Redução-CSM-SM}(A, B, n)$

1 $A' \leftarrow AA$ ▷ concatena duas cópias de A
2 $B' \leftarrow B$
3 $n' \leftarrow 2n$
4 $m' \leftarrow n$
5 **devolva** $\textsc{Alg}_{\text{SM}}(A', n', B', m')$

- **Tempo da redução:** $O(n)$
- **Correção:** basta mostrar que se k é a solução de SM, então k também é solução de CSM.

Exemplo:



CSM $\asymp$ SM

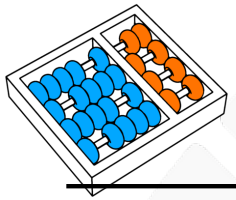
Algoritmo 28: $\textsc{Redução-CSM-SM}(A, B, n)$

```
1 A' ← AA        ▷ concatena duas cópias de A
2 B' ← B
3 n' ← 2n
4 m' ← n
5 devolva ALG_SM(A', n', B', m')
```

- **Tempo da redução:** $O(n)$
- **Correção:** basta mostrar que se k é a solução de SM, então k também é solução de CSM.

Exemplo:

- $I_{CSM} = (acgtact, gtactac, 7)$.



CSM $\preceq$ SM

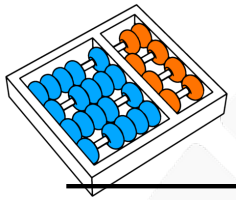
Algoritmo 29: $\textsc{Redução-CSM-SM}(A, B, n)$

```
1 A' ← AA        ▷ concatena duas cópias de A
2 B' ← B
3 n' ← 2n
4 m' ← n
5 devolva ALG_SM(A', n', B', m')
```

- **Tempo da redução:** $O(n)$
- **Correção:** basta mostrar que se k é a solução de SM, então k também é solução de CSM.

Exemplo:

- $I_{CSM} = (acgtact, gtactac, 7)$.
- $I_{SM} = (acgtactacgtact, 14, gtactac, 7)$.



CSM $\preceq$ SM

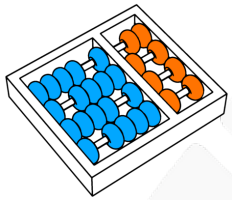
Algoritmo 30: $\textsc{Redução-CSM-SM}(A, B, n)$

```
1 A' ← AA        ▷ concatena duas cópias de A
2 B' ← B
3 n' ← 2n
4 m' ← n
5 devolva ALG_SM(A', n', B', m')
```

- **Tempo da redução:** $O(n)$
- **Correção:** basta mostrar que se k é a solução de SM, então k também é solução de CSM.

Exemplo:

- $I_{CSM} = (acgtact, gtactac, 7)$.
- $I_{SM} = (acgtactacgtact, 14, gtactac, 7)$.
- $S_{SM} = S_{CSM} = (\text{SIM}, 2)$.

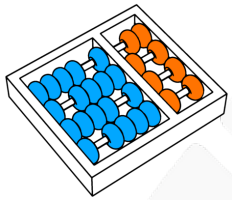


Problema de origem

Problema (Existência de triângulo (PET))

Entrada: *Um grafo conexo* $G = (V, E)$ *sem laços com* $n = |V|$ *e* $m = |E|$.

Saída: *Decidir se* G *contém um triângulo.*



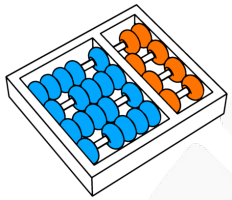
Problema de origem

Problema (Existência de triângulo (PET))

Entrada: *Um grafo conexo* $G = (V, E)$ *sem laços com* $n = |V|$ *e* $m = |E|$.

Saída: *Decidir se* G *contém um triângulo.*

Exemplo:



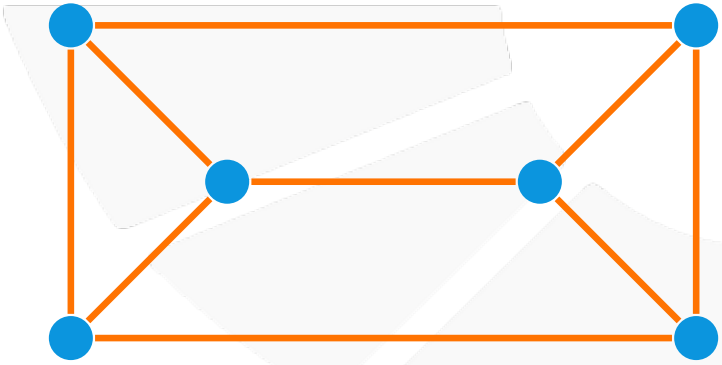
Problema de origem

Problema (Existência de triângulo (PET))

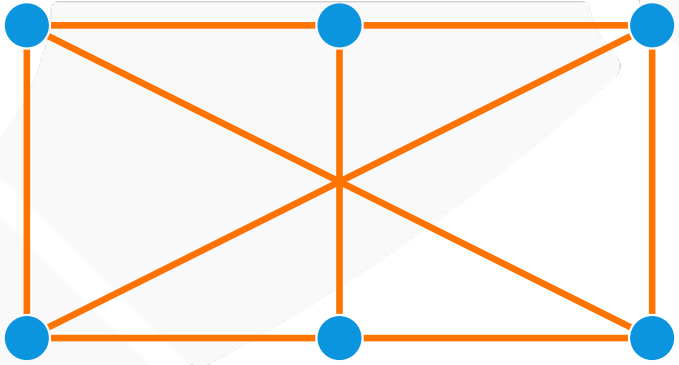
Entrada: *Um grafo conexo* $G = (V, E)$ *sem laços com* $n = |V|$ *e* $m = |E|$.

Saída: *Decidir se* G *contém um triângulo.*

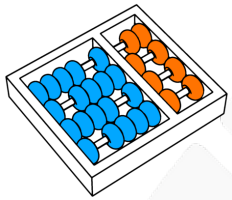
Exemplo:



SIM

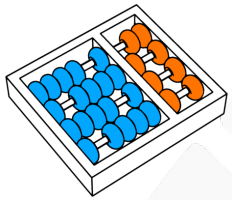


NAO



Observações sobre o PET

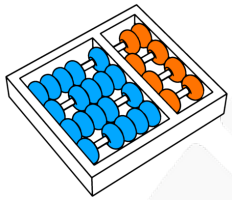
Alguns algoritmos conhecidos:



Observações sobre o PET

Alguns algoritmos conhecidos:

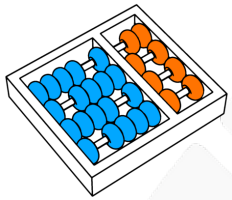
- Um algoritmo trivial de tempo $O(n^3)$:



Observações sobre o PET

Alguns algoritmos conhecidos:

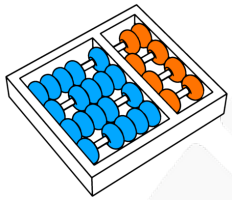
- Um algoritmo trivial de tempo $O(n^3)$:
  - Verifica todas as triplas de vértices.



Observações sobre o PET

Alguns algoritmos conhecidos:

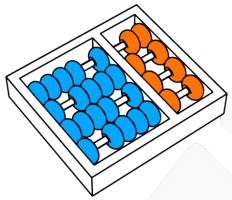
- Um algoritmo trivial de tempo $O(n^3)$:
  - Verifica todas as triplas de vértices.
- Um algoritmo $O(mn)$:



Observações sobre o PET

Alguns algoritmos conhecidos:

- Um algoritmo trivial de tempo $O(n^3)$:
  - Verifica todas as triplas de vértices.
- Um algoritmo $O(mn)$:
  - É muito bom se o grafo é **ESPARSO**.

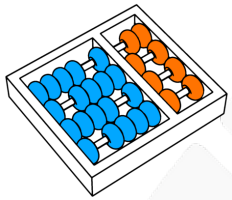


Observações sobre o PET

Alguns algoritmos conhecidos:

- Um algoritmo trivial de tempo $O(n^3)$:
  - Verifica todas as triplas de vértices.
- Um algoritmo $O(mn)$:
  - É muito bom se o grafo é **ESPARSO**.

Vamos supor que o grafo é denso:



Observações sobre o PET

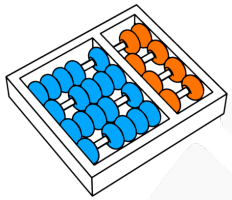
Alguns algoritmos conhecidos:

- Um algoritmo trivial de tempo $O(n^3)$:
  - Verifica todas as triplas de vértices.
- Um algoritmo $O(mn)$:
  - É muito bom se o grafo é **ESPARSO**.

Vamos supor que o grafo é denso:

- G será representado por uma matriz de adjacência A:

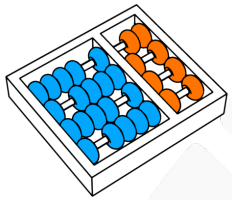
$$a_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E \\ 0 & \text{se } (i,j) \notin E \end{cases}$$



Um lema útil

Lema

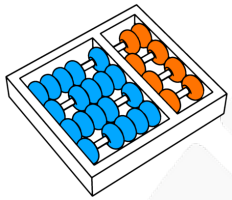
Seja $A^2 = A \times A$, *ou seja,* $a_{ij}^2 = \sum_{k=1}^{n} a_{ik} a_{kj}$.



Um lema útil

Lema

Seja $A^2 = A \times A$, *ou seja,* $a^2_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$.
Então, $a^2_{ij} > 0$ *se e somente se existe caminho de tamanho dois saindo de* i *e chegando em* j.

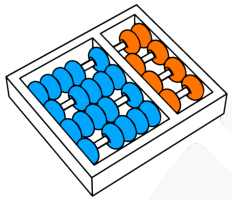


Um lema útil

Lema

Seja $A^2 = A \times A$, *ou seja,* $a^2_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$.
Então, $a^2_{ij} > 0$ *se e somente se existe caminho de tamanho dois saindo de* i *e chegando em* j.

Demonstração:

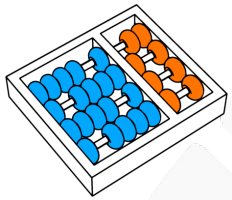


Um lema útil

Lema

Seja $A^2 = A \times A$, *ou seja,* $a^2_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$.
Então, $a^2_{ij} > 0$ *se e somente se existe caminho de tamanho dois saindo de* i *e chegando em* j.

Demonstração:



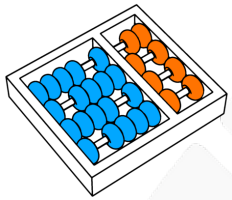
Um lema útil

Lema

Seja $A^2 = A \times A$, *ou seja,* $a^2_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$.
Então, $a^2_{ij} > 0$ *se e somente se existe caminho de tamanho dois saindo de* i *e chegando em* j.

Demonstração:

($\Rightarrow$)
- Se $a^2_{ij} > 0$, então algum termo $a_{ik} a_{kj}$ é positivo.



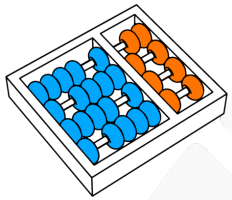
Um lema útil

Lema

Seja $A^2 = A \times A$, *ou seja,* $a^2_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$.
Então, $a^2_{ij} > 0$ *se e somente se existe caminho de tamanho dois saindo de* i *e chegando em* j.

Demonstração:

($\Rightarrow$)
- Se $a^2_{ij} > 0$, então algum termo $a_{ik} a_{kj}$ é positivo.
- Segue que $a_{ik} = 1$ e $a_{kj} = 1$.



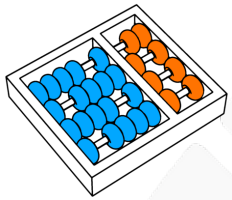
Um lema útil

Lema

Seja $A^2 = A \times A$, *ou seja,* $a^2_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$.
Então, $a^2_{ij} > 0$ *se e somente se existe caminho de tamanho dois saindo de* i *e chegando em* j.

Demonstração:

($\Rightarrow$)
- Se $a^2_{ij} > 0$, então algum termo $a_{ik} a_{kj}$ é positivo.
- Segue que $a_{ik} = 1$ e $a_{kj} = 1$.
- Ou seja, há arestas (i, k) e (k, j).



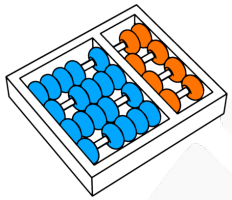
Um lema útil

Lema

Seja $A^2 = A \times A$, *ou seja,* $a^2_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$.
Então, $a^2_{ij} > 0$ *se e somente se existe caminho de tamanho dois saindo de* i *e chegando em* j.

Demonstração:

($\Rightarrow$)
- Se $a^2_{ij} > 0$, então algum termo $a_{ik} a_{kj}$ é positivo.
- Segue que $a_{ik} = 1$ e $a_{kj} = 1$.
- Ou seja, há arestas (i, k) e (k, j).



Um lema útil

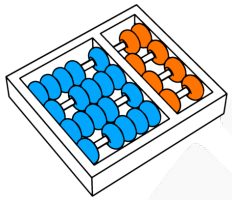
Lema

Seja $A^2 = A \times A$, *ou seja,* $a^2_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$.
Então, $a^2_{ij} > 0$ *se e somente se existe caminho de tamanho dois saindo de* i *e chegando em* j.

Demonstração:

($\Rightarrow$)
- Se $a^2_{ij} > 0$, então algum termo $a_{ik} a_{kj}$ é positivo.
- Segue que $a_{ik} = 1$ e $a_{kj} = 1$.
- Ou seja, há arestas (i, k) e (k, j).

($\Leftarrow$)
- Seja um caminho (i, k, j) de i até j.



Um lema útil

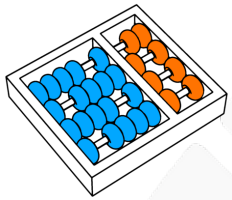
Lema

Seja $A^2 = A \times A$, *ou seja,* $a^2_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$.
Então, $a^2_{ij} > 0$ *se e somente se existe caminho de tamanho dois saindo de* i *e chegando em* j.

Demonstração:

($\Rightarrow$)
- Se $a^2_{ij} > 0$, então algum termo $a_{ik} a_{kj}$ é positivo.
- Segue que $a_{ik} = 1$ e $a_{kj} = 1$.
- Ou seja, há arestas (i, k) e (k, j).

($\Leftarrow$)
- Seja um caminho (i, k, j) de i até j.
- Então, $a_{ik} = 1$ e $a_{kj} = 1$.



Um lema útil

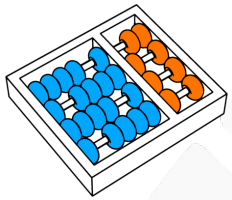
Lema

Seja $A^2 = A \times A$, *ou seja,* $a^2_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$.
Então, $a^2_{ij} > 0$ *se e somente se existe caminho de tamanho dois saindo de* i *e chegando em* j.

Demonstração:

($\Rightarrow$)
- Se $a^2_{ij} > 0$, então algum termo $a_{ik} a_{kj}$ é positivo.
- Segue que $a_{ik} = 1$ e $a_{kj} = 1$.
- Ou seja, há arestas (i, k) e (k, j).

($\Leftarrow$)
- Seja um caminho (i, k, j) de i até j.
- Então, $a_{ik} = 1$ e $a_{kj} = 1$.
- Daí, $a_{ik} a_{kj} > 0$ e, portanto, $a^2_{ij} > 0$.

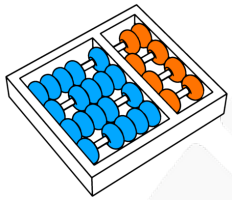


Problema destino

Problema (Multiplicação de Matrizes Quadradas (MMQ))

Entrada: *Uma matriz quadrada A de ordem n e uma matriz quadrada B de ordem n.*

Saída: *O produto $P = A \times B$.*



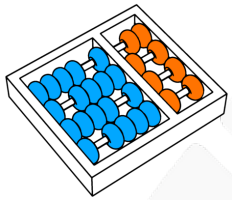
Problema destino

Problema (Multiplicação de Matrizes Quadradas (MMQ))

Entrada: *Uma matriz quadrada A de ordem n e uma matriz quadrada B de ordem n.*

Saída: *O produto $P = A \times B$.*

Observações:



Problema destino

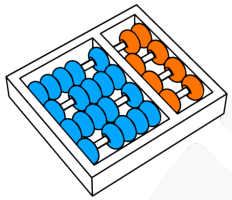
Problema (Multiplicação de Matrizes Quadradas (MMQ))

Entrada: *Uma matriz quadrada A de ordem n e uma matriz quadrada B de ordem n.*

Saída: *O produto $P = A \times B$.*

Observações:

- Há um algoritmo óbvio de complexidade $O(n^3)$.



Problema destino

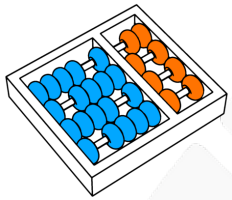
Problema (Multiplicação de Matrizes Quadradas (MMQ))

Entrada: *Uma matriz quadrada A de ordem n e uma matriz quadrada B de ordem n.*

Saída: *O produto $P = A \times B$.*

Observações:

- Há um algoritmo óbvio de complexidade $O(n^3)$.
- MMQ pode ser resolvida mais rapidamente:



Problema destino

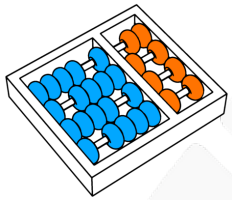
Problema (Multiplicação de Matrizes Quadradas (MMQ))

Entrada: *Uma matriz quadrada A de ordem n e uma matriz quadrada B de ordem n.*

Saída: *O produto $P = A \times B$.*

Observações:

- Há um algoritmo óbvio de complexidade $O(n^3)$.
- MMQ pode ser resolvida mais rapidamente:
  - Em tempo $O(n^{2,807})$ pelo algoritmo de Strassen (1969).



Problema destino

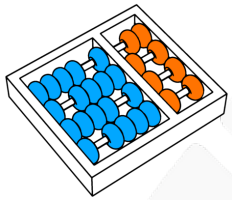
Problema (Multiplicação de Matrizes Quadradas (MMQ))

Entrada: *Uma matriz quadrada A de ordem n e uma matriz quadrada B de ordem n.*

Saída: *O produto $P = A \times B$.*

Observações:

- Há um algoritmo óbvio de complexidade $O(n^3)$.
- MMQ pode ser resolvida mais rapidamente:
  - Em tempo $O(n^{2,807})$ pelo algoritmo de Strassen (1969).
  - Em tempo $O(n^{2,376})$ pelo algoritmo de Coppersmith e Winograd (1990).



Problema destino

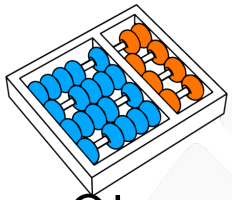
Problema (Multiplicação de Matrizes Quadradas (MMQ))

Entrada: *Uma matriz quadrada A de ordem n e uma matriz quadrada B de ordem n.*

Saída: *O produto $P = A \times B$.*

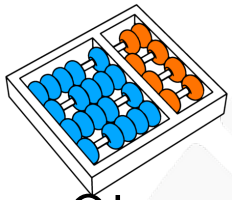
Observações:

- Há um algoritmo óbvio de complexidade $O(n^3)$.
- MMQ pode ser resolvida mais rapidamente:
  - Em tempo $O(n^{2,807})$ pelo algoritmo de Strassen (1969).
  - Em tempo $O(n^{2,376})$ pelo algoritmo de Coppersmith e Winograd (1990).
  - Em tempo $O(n^{2,3728639})$ pelo de François Le Gall (2014).



PET $\asymp$ MMQ

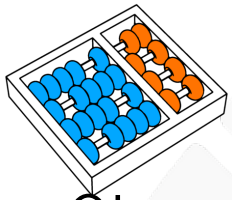
Observe que só existe triângulo com aresta (i, j) se:



PET $\propto$ MMQ

Observe que só existe triângulo com aresta (i, j) se:

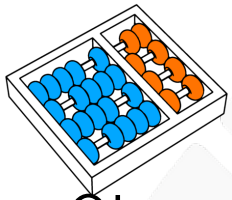
1. Existir um caminho de tamanho 2 de i a j.



PET $\asymp$ MMQ

Observe que só existe triângulo com aresta (i, j) se:

1. Existir um caminho de tamanho 2 de i a j.
2. Existir a aresta (i, j).



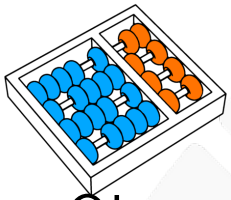
PET $\preceq$ MMQ

Observe que só existe triângulo com aresta (i, j) se:

1. Existir um caminho de tamanho 2 de i a j.
2. Existir a aresta (i, j).

Algoritmo 34: $\textsc{Redução-PET-MMQ}(A, n)$

```
1 A² ← Alg_MMQ(A, n)
2 para i = 1 até n
3     para j = 1 até n
4         se a²_ij > 0 e a_ij = 1
5             devolva SIM
6 devolva NAO
```



PET $\preceq$ MMQ

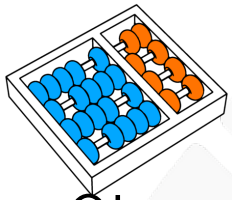
Observe que só existe triângulo com aresta (i, j) se:

1. Existir um caminho de tamanho 2 de i a j.
2. Existir a aresta (i, j).

Algoritmo 35: $\textsc{Redução-PET-MMQ}(A, n)$

```
1 A² ← Alg_MMQ(A, n)
2 para i = 1 até n
3     para j = 1 até n
4         se a²_ij > 0 e a_ij = 1
5             devolva SIM
6 devolva NAO
```

- **Tempo da redução:** $O(n^2)$.



PET $\preceq$ MMQ

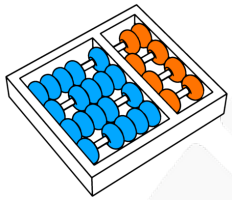
Observe que só existe triângulo com aresta (i, j) se:

1. Existir um caminho de tamanho 2 de i a j.
2. Existir a aresta (i, j).

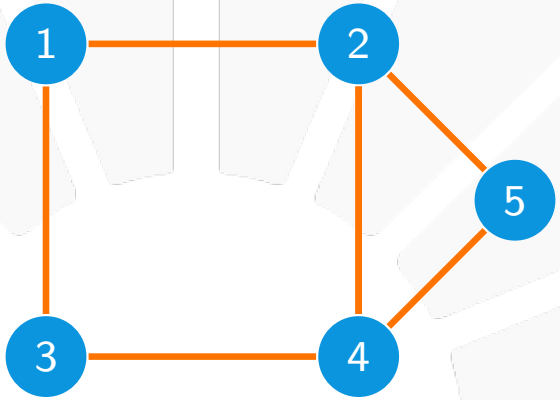
Algoritmo 36: REDUÇÃO-PET-MMQ(A, n)

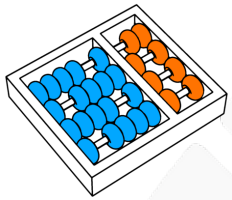
```
1 A² ← ALG_MMQ(A, n)
2 para i = 1 até n
3     para j = 1 até n
4         se a²_ij > 0 e a_ij = 1
5             devolva SIM
6 devolva NAO
```

- **Tempo da redução:** $O(n^2)$.
- **Tempo total:** $O(n^{2{,}3728639})$.

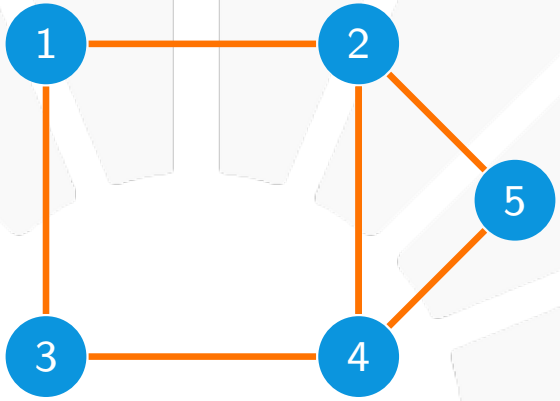


PET ≤ MMQ

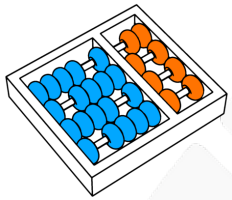




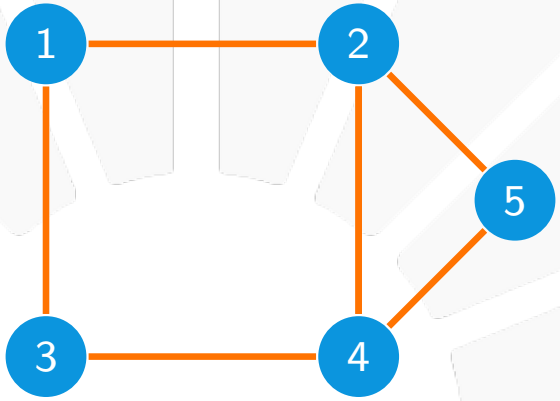
PET ≤ MMQ



A	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	1	1
3	1	0	0	1	0
4	0	1	1	0	1
5	0	1	0	1	0

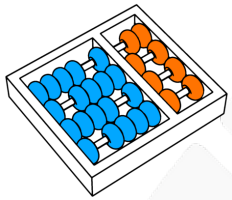


PET $\propto$ MMQ



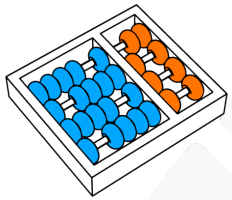
A	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	1	1
3	1	0	0	1	0
4	0	1	1	0	1
5	0	1	0	1	0

A^2	1	2	3	4	5
1	2	0	0	2	1
2	0	3	2	1	1
3	0	2	2	0	1
4	2	1	0	3	1
5	1	1	1	1	2



Considerando o caso particular

Considere um caso particular de MMQ:



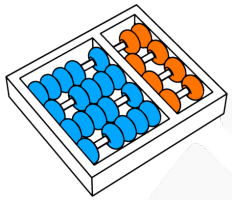
Considerando o caso particular

Considere um caso particular de MMQ:

Problema (Multiplicação de Matrizes Simétricas (MMS))

Entrada: *Uma matriz* ***SIMÉTRICA*** *quadrada A de ordem m e uma matriz* ***SIMÉTRICA*** *quadrada B de ordem m.*

Saída: *O produto $P = A \times B$.*



Considerando o caso particular

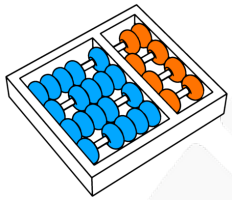
Considere um caso particular de MMQ:

Problema (Multiplicação de Matrizes Simétricas (MMS))

Entrada: *Uma matriz* ***SIMÉTRICA*** *quadrada A de ordem m e uma matriz* ***SIMÉTRICA*** *quadrada B de ordem m.*

Saída: *O produto $P = A \times B$.*

Claro que MMS $\preccurlyeq_{m^2}$ MMQ.



Considerando o caso particular

Considere um caso particular de MMQ:

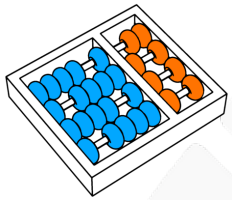
Problema (Multiplicação de Matrizes Simétricas (MMS))

Entrada: *Uma matriz* ***SIMÉTRICA*** *quadrada A de ordem m e uma matriz* ***SIMÉTRICA*** *quadrada B de ordem m.*

Saída: *O produto $P = A \times B$.*

Claro que MMS $\preccurlyeq_{m^2}$ MMQ.

- Portanto, MMQ é pelo menos tão difícil quanto MMS.



Considerando o caso particular

Considere um caso particular de MMQ:

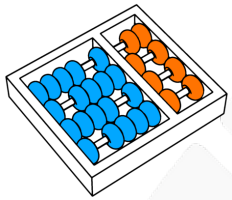
Problema (Multiplicação de Matrizes Simétricas (MMS))

Entrada: *Uma matriz* ***SIMÉTRICA*** *quadrada A de ordem m e uma matriz* ***SIMÉTRICA*** *quadrada B de ordem m.*

Saída: *O produto $P = A \times B$.*

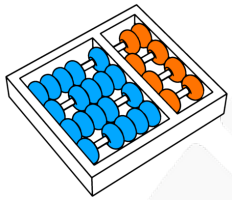
Claro que MMS $\preccurlyeq_{m^2}$ MMQ.

- Portanto, MMQ é pelo menos tão difícil quanto MMS.
- Será que MMS também é pelo menos tão difícil quanto MMQ?



Reduzindo MMQ $\propto$ MMS

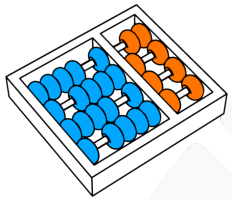
1. Considere uma instância de MMQ, $I_{MMQ} = (A, B, n)$.



Reduzindo MMQ $\preceq$ MMS

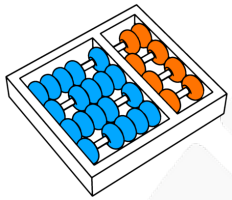
1. Considere uma instância de MMQ, $I_{MMQ} = (A, B, n)$.
2. Construa uma instância de MMS, $I_{MMS} = (A', B', 2n)$, em que

$$A' = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \quad \text{e} \quad B' = \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix}.$$



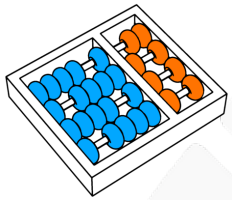
Reduzindo MMQ $\preceq$ MMS

1. Considere uma instância de MMQ, $I_{MMQ} = (A, B, n)$.
2. Construa uma instância de MMS, $I_{MMS} = (A', B', 2n)$, em que
$$A' = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \quad \text{e} \quad B' = \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix}.$$
3. A solução de MMS é:
$$P' = A'B' = \begin{bmatrix} AB & 0 \\ 0 & A^T B^T \end{bmatrix}.$$



Reduzindo MMQ $\preceq$ MMS

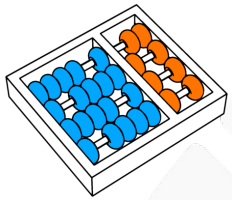
1. Considere uma instância de MMQ, $I_{MMQ} = (A, B, n)$.
2. Construa uma instância de MMS, $I_{MMS} = (A', B', 2n)$, em que
$$A' = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \quad \text{e} \quad B' = \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix}.$$
3. A solução de MMS é:
$$P' = A'B' = \begin{bmatrix} AB & 0 \\ 0 & A^T B^T \end{bmatrix}.$$
4. Devolva o primeiro bloco da matriz P'.



Reduzindo MMQ $\preceq$ MMS

1. Considere uma instância de MMQ, $I_{MMQ} = (A, B, n)$.
2. Construa uma instância de MMS, $I_{MMS} = (A', B', 2n)$, em que
$$A' = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \quad \text{e} \quad B' = \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix}.$$
3. A solução de MMS é:
$$P' = A'B' = \begin{bmatrix} AB & 0 \\ 0 & A^T B^T \end{bmatrix}.$$
4. Devolva o primeiro bloco da matriz P'.

Tempo da redução: $O(n^2)$.

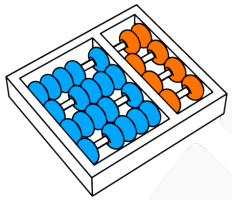


Reduzindo MMQ $\preceq$ MMS

1. Considere uma instância de MMQ, $I_{MMQ} = (A, B, n)$.
2. Construa uma instância de MMS, $I_{MMS} = (A', B', 2n)$, em que
$$A' = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \quad \text{e} \quad B' = \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix}.$$
3. A solução de MMS é:
$$P' = A'B' = \begin{bmatrix} AB & 0 \\ 0 & A^T B^T \end{bmatrix}.$$
4. Devolva o primeiro bloco da matriz P'.

Tempo da redução: $O(n^2)$.

- Construir I_{MMQ} leva tempo $O(n^2)$.

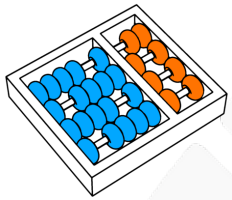


Reduzindo MMQ $\preceq$ MMS

1. Considere uma instância de MMQ, $I_{MMQ} = (A, B, n)$.
2. Construa uma instância de MMS, $I_{MMS} = (A', B', 2n)$, em que
$$A' = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \quad \text{e} \quad B' = \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix}.$$
3. A solução de MMS é:
$$P' = A'B' = \begin{bmatrix} AB & 0 \\ 0 & A^T B^T \end{bmatrix}.$$
4. Devolva o primeiro bloco da matriz P'.

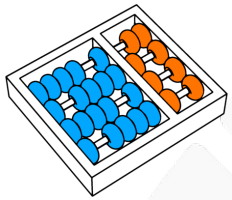
Tempo da redução: $O(n^2)$.

- Construir I_{MMQ} leva tempo $O(n^2)$.
- Copiar o bloco e P' leva tempo $O(n^2)$.



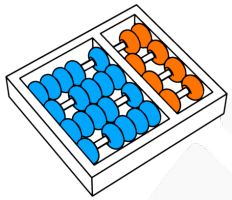
Interpretando os fatos

- Suponha que exista um algoritmo para MMS de tempo **O(T(m))** para algum polinômio $T(m)$.



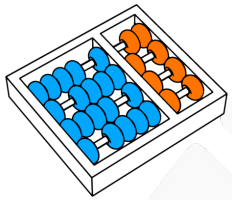
Interpretando os fatos

- Suponha que exista um algoritmo para MMS de tempo **O(T(m))** para algum polinômio $T(m)$.
  - Lembre que m é a ordem das matrizes A' e B'.



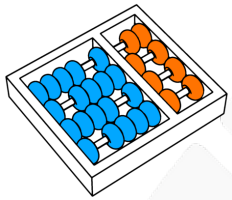
Interpretando os fatos

- Suponha que exista um algoritmo para MMS de tempo **O(T(m))** para algum polinômio $T(m)$.
  - Lembre que m é a ordem das matrizes A' e B'.
- Quão pequeno pode ser $T(m)$?



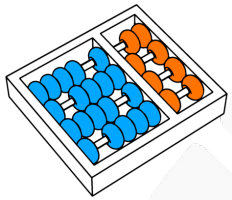
Interpretando os fatos

- Suponha que exista um algoritmo para MMS de tempo **O(T(m))** para algum polinômio $T(m)$.
  - Lembre que m é a ordem das matrizes A' e B'.
- Quão pequeno pode ser $T(m)$?
  - É claro que $T(m) = \Omega(m^2)$, pois é preciso ler a entrada.



Interpretando os fatos

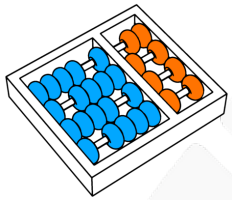
- Suponha que exista um algoritmo para MMS de tempo **O(T(m))** para algum polinômio $T(m)$.
  - Lembre que m é a ordem das matrizes A' e B'.
- Quão pequeno pode ser $T(m)$?
  - É claro que $T(m) = \Omega(m^2)$, pois é preciso ler a entrada.
  - Será que pode ser mais rápido que $o(m^{2,3728639})$?



Interpretando os fatos

- Suponha que exista um algoritmo para MMS de tempo **O(T(m))** para algum polinômio $T(m)$.
  - Lembre que m é a ordem das matrizes A' e B'.
- Quão pequeno pode ser $T(m)$?
  - É claro que $T(m) = \Omega(m^2)$, pois é preciso ler a entrada.
  - Será que pode ser mais rápido que $o(m^{2,3728639})$?

Para responder isso, usamos a redução MMQ $\preccurlyeq_{n^2}$ MMS:

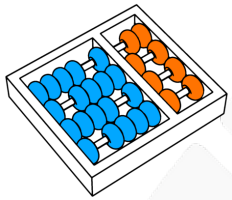


Interpretando os fatos

- Suponha que exista um algoritmo para MMS de tempo **O(T(m))** para algum polinômio $T(m)$.
  - Lembre que m é a ordem das matrizes A' e B'.
- Quão pequeno pode ser $T(m)$?
  - É claro que $T(m) = \Omega(m^2)$, pois é preciso ler a entrada.
  - Será que pode ser mais rápido que $o(m^{2,3728639})$?

Para responder isso, usamos a redução MMQ $\preccurlyeq_{n^2}$ MMS:

- Ela implica em um algoritmo de tempo total $O(T(m) + n^2)$.

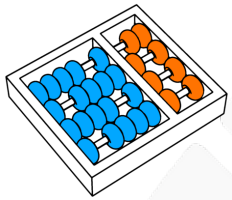


Interpretando os fatos

- Suponha que exista um algoritmo para MMS de tempo **O(T(m))** para algum polinômio $T(m)$.
  - Lembre que m é a ordem das matrizes A' e B'.
- Quão pequeno pode ser $T(m)$?
  - É claro que $T(m) = \Omega(m^2)$, pois é preciso ler a entrada.
  - Será que pode ser mais rápido que $o(m^{2,3728639})$?

Para responder isso, usamos a redução MMQ $\preccurlyeq_{n^2}$ MMS:

- Ela implica em um algoritmo de tempo total $O(T(m) + n^2)$.
- Como $m = 2n$, tempo é $O(T(2n) + n^2) = O(T(n) + n^2)$.

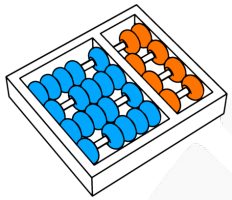


Interpretando os fatos

- Suponha que exista um algoritmo para MMS de tempo **O(T(m))** para algum polinômio $T(m)$.
  - Lembre que m é a ordem das matrizes A' e B'.
- Quão pequeno pode ser $T(m)$?
  - É claro que $T(m) = \Omega(m^2)$, pois é preciso ler a entrada.
  - Será que pode ser mais rápido que $o(m^{2,3728639})$?

Para responder isso, usamos a redução MMQ $\preccurlyeq_{n^2}$ MMS:

- Ela implica em um algoritmo de tempo total $O(T(m) + n^2)$.
- Como $m = 2n$, tempo é $O(T(2n) + n^2) = O(T(n) + n^2)$.
- Como $T(n)$ domina n^2, o tempo é simplesmente $O(T(n))$.



Interpretando os fatos

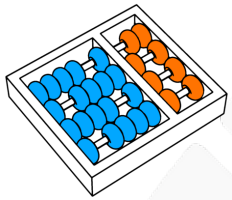
- Suponha que exista um algoritmo para MMS de tempo **O(T(m))** para algum polinômio $T(m)$.
  - Lembre que m é a ordem das matrizes A' e B'.
- Quão pequeno pode ser $T(m)$?
  - É claro que $T(m) = \Omega(m^2)$, pois é preciso ler a entrada.
  - Será que pode ser mais rápido que $o(m^{2,3728639})$?

Para responder isso, usamos a redução MMQ $\preccurlyeq_{n^2}$ MMS:

- Ela implica em um algoritmo de tempo total $O(T(m) + n^2)$.
- Como $m = 2n$, tempo é $O(T(2n) + n^2) = O(T(n) + n^2)$.
- Como $T(n)$ domina n^2, o tempo é simplesmente $O(T(n))$.

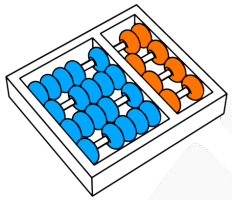
Ou seja: Um algoritmo com complexidade $T(m)$ para MMS implica em um algoritmo com complexidade $T(n)$ para MMQ!

Reduções para obtenção de cota inferior



Uma redução $A \preccurlyeq_{f(n)} B$

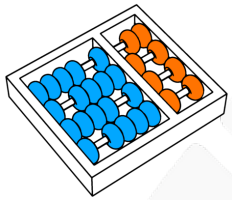
Suponha que:



Uma redução $A \preccurlyeq_{f(n)} B$

Suponha que:

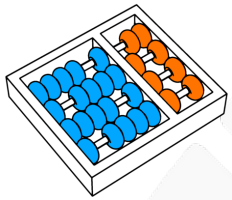
- A tem cota inferior $h(n)$.



Uma redução $A \preccurlyeq_{f(n)} B$

Suponha que:

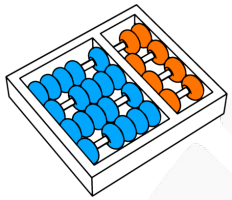
- A tem cota inferior $h(n)$.
- $\textsc{Alg}_B$ resolve B em tempo $g(n)$.



Uma redução $A \preccurlyeq_{f(n)} B$

Suponha que:

- A tem cota inferior $h(n)$.
- $\textsc{Alg}_B$ resolve B em tempo $g(n)$.
- A redução gasta tempo $f(n) \leq \frac{h(n)}{2}$.

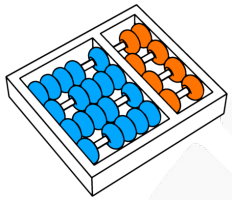


Uma redução $A \preccurlyeq_{f(n)} B$

Suponha que:

- A tem cota inferior $h(n)$.
- $\textsc{Alg}_B$ resolve B em tempo $g(n)$.
- A redução gasta tempo $f(n) \leq \frac{h(n)}{2}$.

Então, um algoritmo baseado na redução $A \preccurlyeq_{f(n)} B$ tem tmepo:



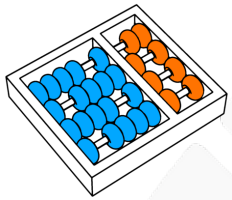
Uma redução $A \preccurlyeq_{f(n)} B$

Suponha que:

- A tem cota inferior $h(n)$.
- $\textsc{Alg}_B$ resolve B em tempo $g(n)$.
- A redução gasta tempo $f(n) \leq \frac{h(n)}{2}$.

Então, um algoritmo baseado na redução $A \preccurlyeq_{f(n)} B$ tem tmepo:

$f(n) + g(n)$



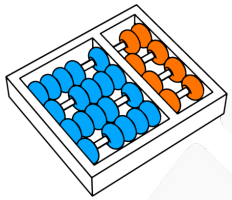
Uma redução $A \preccurlyeq_{f(n)} B$

Suponha que:

- A tem cota inferior $h(n)$.
- $\textsc{Alg}_B$ resolve B em tempo $g(n)$.
- A redução gasta tempo $f(n) \leq \frac{h(n)}{2}$.

Então, um algoritmo baseado na redução $A \preccurlyeq_{f(n)} B$ tem tmepo:

$$f(n) + g(n) \geq h(n)$$



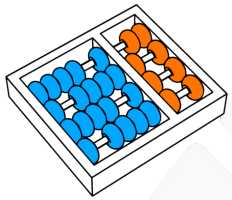
Uma redução $A \preccurlyeq_{f(n)} B$

Suponha que:

- A tem cota inferior $h(n)$.
- ALG_B resolve B em tempo $g(n)$.
- A redução gasta tempo $f(n) \leq \frac{h(n)}{2}$.

Então, um algoritmo baseado na redução $A \preccurlyeq_{f(n)} B$ tem tmepo:

$f(n) + g(n) \geq h(n) \Rightarrow$



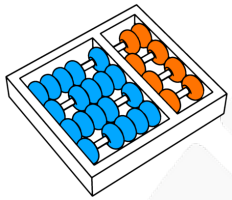
Uma redução $A \preccurlyeq_{f(n)} B$

Suponha que:

- A tem cota inferior $h(n)$.
- ALG_B resolve B em tempo $g(n)$.
- A redução gasta tempo $f(n) \leq \frac{h(n)}{2}$.

Então, um algoritmo baseado na redução $A \preccurlyeq_{f(n)} B$ tem tmepo:

$$f(n) + g(n) \geq h(n) \Rightarrow g(n) \geq h(n) - f(n)$$



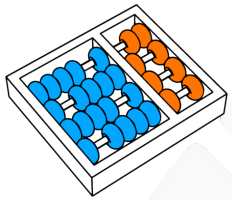
Uma redução $A \preccurlyeq_{f(n)} B$

Suponha que:

- A tem cota inferior $h(n)$.
- ALG_B resolve B em tempo $g(n)$.
- A redução gasta tempo $f(n) \leq \frac{h(n)}{2}$.

Então, um algoritmo baseado na redução $A \preccurlyeq_{f(n)} B$ tem tmepo:

$$f(n) + g(n) \geq h(n) \Rightarrow g(n) \geq h(n) - f(n) \geq h(n) - \frac{h(n)}{2}$$



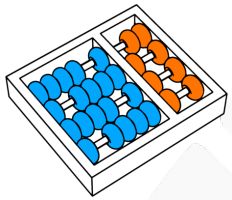
Uma redução $A \preccurlyeq_{f(n)} B$

Suponha que:

- A tem cota inferior $h(n)$.
- ALG_B resolve B em tempo $g(n)$.
- A redução gasta tempo $f(n) \leq \frac{h(n)}{2}$.

Então, um algoritmo baseado na redução $A \preccurlyeq_{f(n)} B$ tem tmepo:

$$f(n) + g(n) \geq h(n) \Rightarrow g(n) \geq h(n) - f(n) \geq h(n) - \frac{h(n)}{2} =$$



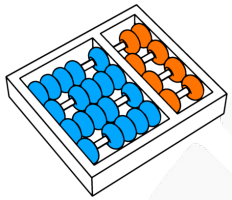
Uma redução $A \preccurlyeq_{f(n)} B$

Suponha que:

- A tem cota inferior $h(n)$.
- ALG_B resolve B em tempo $g(n)$.
- A redução gasta tempo $f(n) \leq \frac{h(n)}{2}$.

Então, um algoritmo baseado na redução $A \preccurlyeq_{f(n)} B$ tem tmepo:

$$f(n) + g(n) \geq h(n) \Rightarrow g(n) \geq h(n) - f(n) \geq h(n) - \frac{h(n)}{2} = \frac{h(n)}{2}$$



Uma redução $A \preccurlyeq_{f(n)} B$

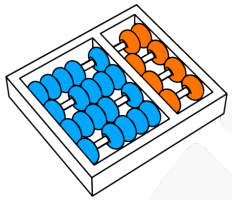
Suponha que:

- A tem cota inferior $h(n)$.
- ALG_B resolve B em tempo $g(n)$.
- A redução gasta tempo $f(n) \leq \frac{h(n)}{2}$.

Então, um algoritmo baseado na redução $A \preccurlyeq_{f(n)} B$ tem tmepo:

$$f(n) + g(n) \geq h(n) \Rightarrow g(n) \geq h(n) - f(n) \geq h(n) - \frac{h(n)}{2} = \frac{h(n)}{2}$$

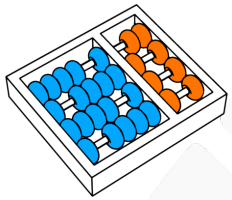
Conclusão: $g(n) \geq \Omega(h(n))$.



Transferindo cotas inferiores

Teorema

Considere dois problemas A e B e suponha que

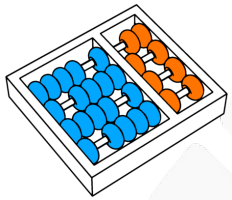


Transferindo cotas inferiores

Teorema

Considere dois problemas A e B e suponha que

1. *$h(n)$ é cota inferior para A e*

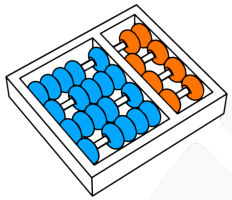


Transferindo cotas inferiores

Teorema

Considere dois problemas A e B e suponha que

1. *$h(n)$ é cota inferior para A e*
2. *$A \preccurlyeq_{f(n)} B$,*

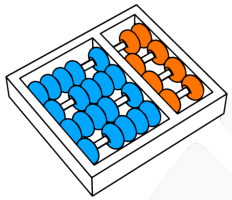


Transferindo cotas inferiores

Teorema

Considere dois problemas A e B e suponha que

1. $h(n)$ *é cota inferior para* A *e*
2. $A \preccurlyeq_{f(n)} B$,
3. $f(n) = o(h(n))$.



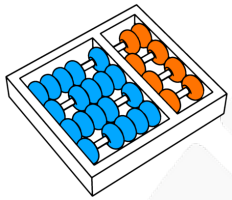
Transferindo cotas inferiores

Teorema

Considere dois problemas A e B e suponha que

1. *$h(n)$ é cota inferior para A e*
2. *$A \preccurlyeq_{f(n)} B$,*
3. *$f(n) = o(h(n))$.*

Então $h(n)$ é cota inferior para B.



Transferindo cotas inferiores

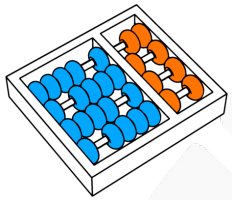
Teorema

Considere dois problemas A e B e suponha que

1. *$h(n)$ é cota inferior para A e*
2. *$A \preccurlyeq_{f(n)} B$,*
3. *$f(n) = o(h(n))$.*

Então $h(n)$ é cota inferior para B.

Observações:



Transferindo cotas inferiores

Teorema

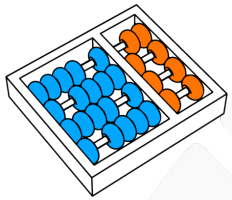
Considere dois problemas A e B e suponha que

1. *$h(n)$ é cota inferior para A e*
2. *$A \preccurlyeq_{f(n)} B$,*
3. *$f(n) = o(h(n))$.*

Então $h(n)$ é cota inferior para B.

Observações:

- A cota inferior depende do **MODELO DE COMPUTAÇÃO**.



Transferindo cotas inferiores

Teorema

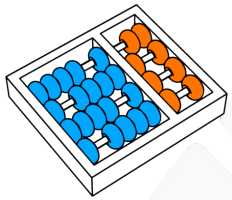
Considere dois problemas A e B e suponha que

1. *$h(n)$ é cota inferior para A e*
2. *$A \preccurlyeq_{f(n)} B$,*
3. *$f(n) = o(h(n))$.*

Então $h(n)$ é cota inferior para B.

Observações:

- A cota inferior depende do **MODELO DE COMPUTAÇÃO**.
- Supomos o mesmo modelo para ambos problemas.

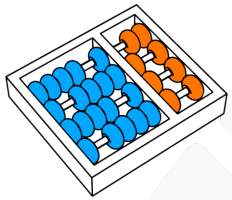


Problema de origem

Problema (Ordenação (ORD))

Entrada: *Uma sequência* $X = (x_1, x_2, \ldots, x_n)$ *de* n *elementos comparáveis.*

Saída: *Uma permutação de* X *cujos elementos estejam ordenados.*



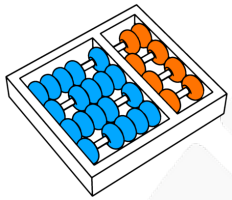
Problema de origem

Problema (Ordenação (ORD))

Entrada: *Uma sequência* $X = (x_1, x_2, \ldots, x_n)$ *de* n *elementos comparáveis.*

Saída: *Uma permutação de* X *cujos elementos estejam ordenados.*

Observações:



Problema de origem

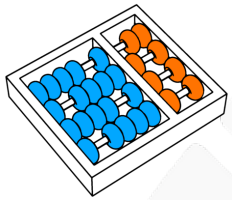
Problema (Ordenação (ORD))

Entrada: *Uma sequência* $X = (x_1, x_2, \ldots, x_n)$ *de* n *elementos comparáveis.*

Saída: *Uma permutação de* X *cujos elementos estejam ordenados.*

Observações:

- Só podemos comparar dois elementos por uma sub-rotina caixa-preta de tempo constante (chamada **ORÁCULO**).



Problema de origem

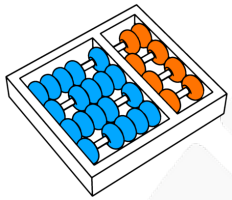
Problema (Ordenação (ORD))

Entrada: *Uma sequência* $X = (x_1, x_2, \ldots, x_n)$ *de* n *elementos comparáveis.*

Saída: *Uma permutação de* X *cujos elementos estejam ordenados.*

Observações:

- Só podemos comparar dois elementos por uma sub-rotina caixa-preta de tempo constante (chamada **ORÁCULO**).
- Esse problema tem cota inferior $\Omega(n \log n)$.

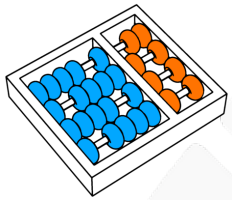


Problema de destino

Problema (Envoltória Convexa (EC))

Entrada: *Um conjunto* $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ *de n pontos no plano.*

Saída: ***MENOR POLÍGONO CONVEXO*** *que contém os n pontos.*

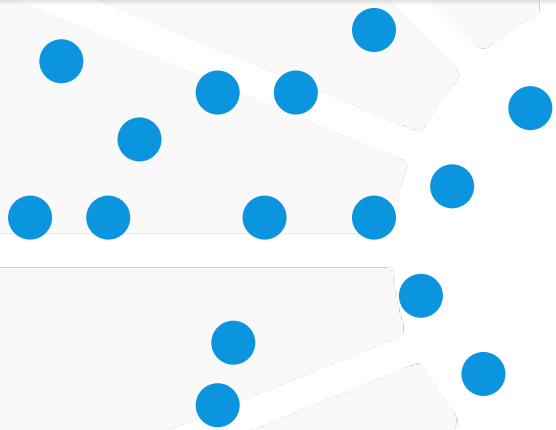


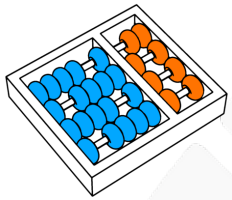
Problema de destino

Problema (Envoltória Convexa (EC))

Entrada: *Um conjunto* $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ *de n pontos no plano.*

Saída: ***MENOR POLÍGONO CONVEXO*** *que contém os n pontos.*



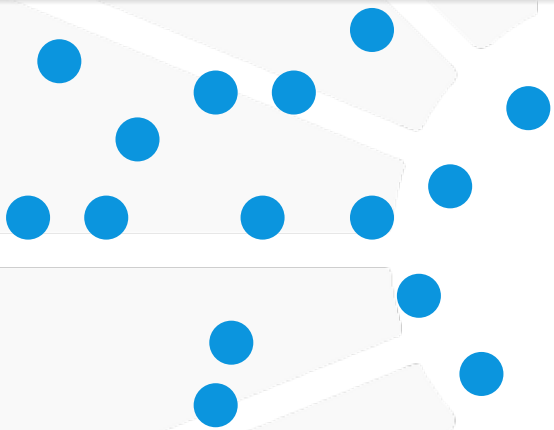


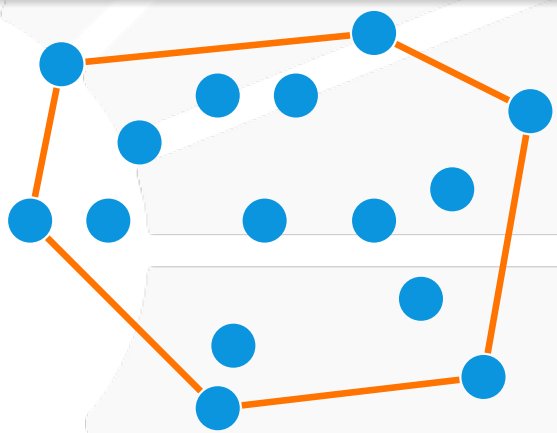
Problema de destino

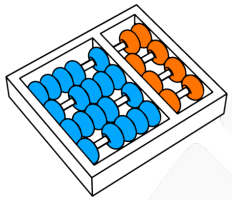
Problema (Envoltória Convexa (EC))

Entrada: *Um conjunto* $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ *de n pontos no plano.*

Saída: ***MENOR POLÍGONO CONVEXO*** *que contém os n pontos.*





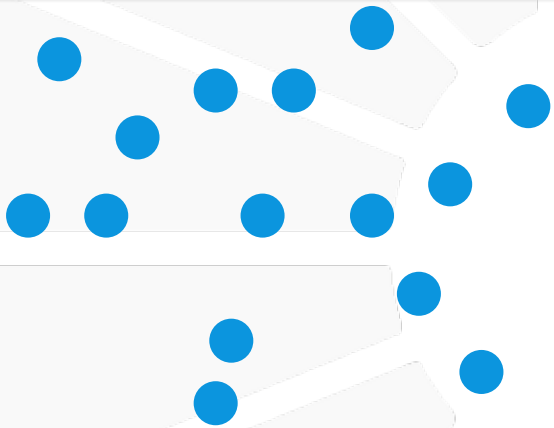


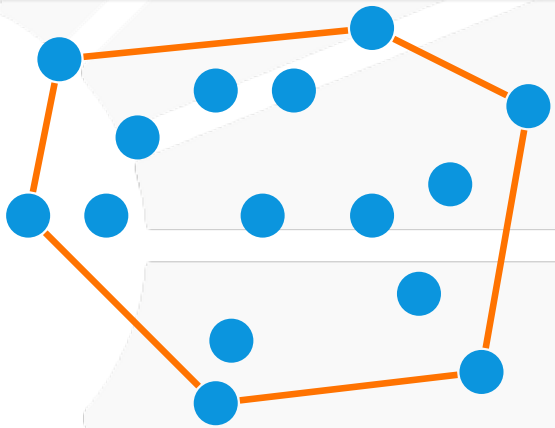
Problema de destino

Problema (Envoltória Convexa (EC))

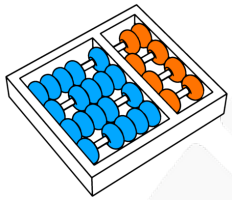
Entrada: *Um conjunto* $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ *de n pontos no plano.*

Saída: ***MENOR POLÍGONO CONVEXO*** *que contém os n pontos.*





Observações:

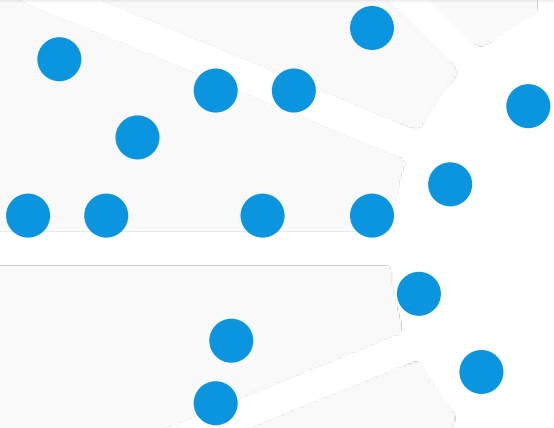


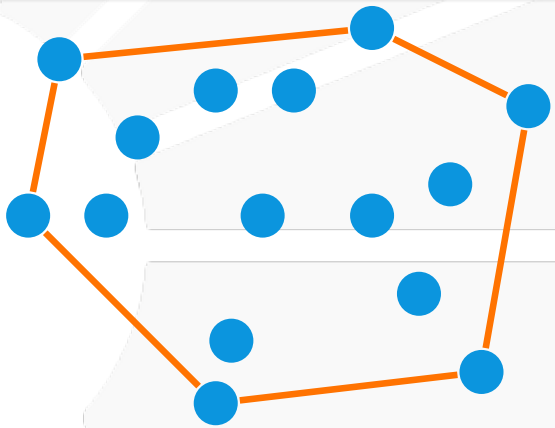
Problema de destino

Problema (Envoltória Convexa (EC))

Entrada: *Um conjunto* $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ *de n pontos no plano.*

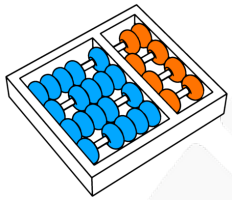
Saída: ***MENOR POLÍGONO CONVEXO*** *que contém os n pontos.*





Observações:

- Os vértices são representados em ordem anti-horária.

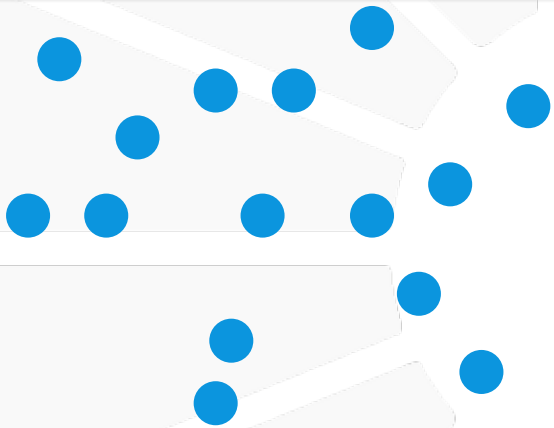


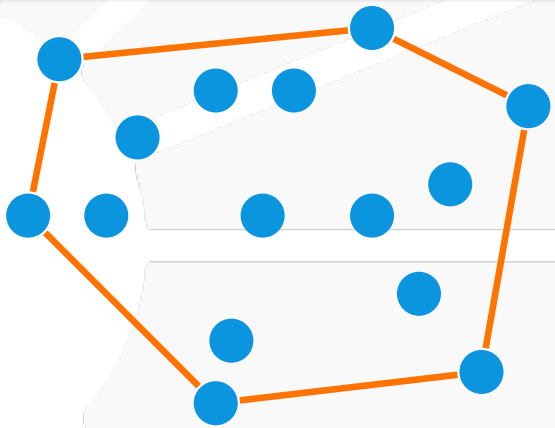
Problema de destino

Problema (Envoltória Convexa (EC))

Entrada: *Um conjunto* $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ *de n pontos no plano.*

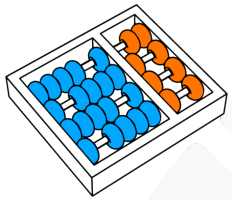
Saída: ***MENOR POLÍGONO CONVEXO*** *que contém os n pontos.*





Observações:

- Os vértices são representados em ordem anti-horária.
- Problema clássico de Geometria Computacional.

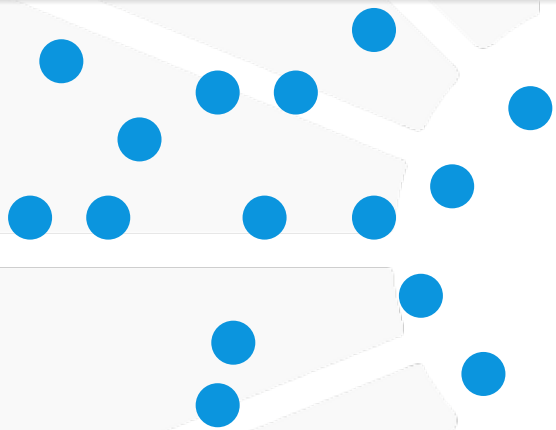


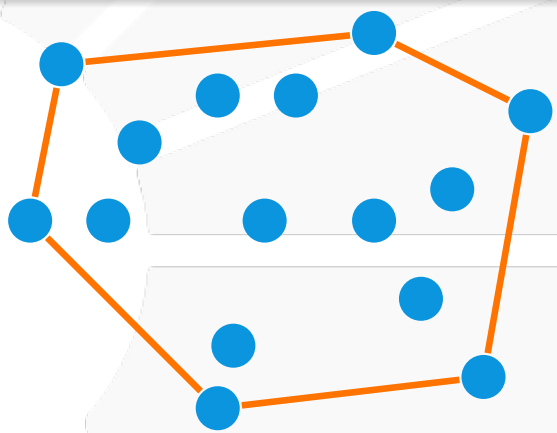
Problema de destino

Problema (Envoltória Convexa (EC))

Entrada: *Um conjunto* $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ *de* n *pontos no plano.*

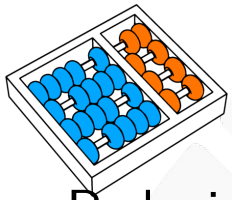
Saída: ***MENOR POLÍGONO CONVEXO*** *que contém os* n *pontos.*





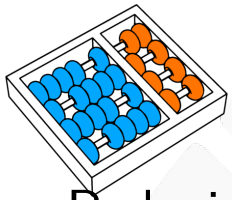
Observações:

- Os vértices são representados em ordem anti-horária.
- Problema clássico de Geometria Computacional.
- Pode ser resolvido em tempo $O(n \log n)$.



ORD $\preceq_n$ EC

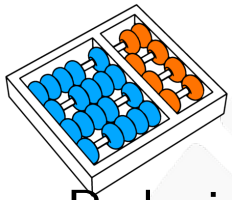
Reduzindo ORD $\preceq_n$ EC:



ORD $\preccurlyeq_n$ EC

Reduzindo ORD $\preccurlyeq_n$ EC:

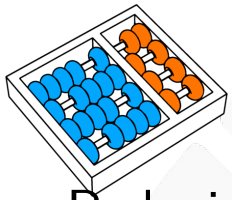
1. Considere uma instância de $I_{ORD} = (x_1, x_2 \ldots, x_n)$.



ORD $\preccurlyeq_n$ EC

Reduzindo ORD $\preccurlyeq_n$ EC:

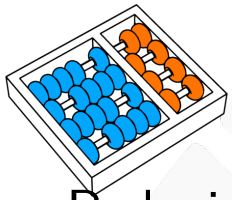
1. Considere uma instância de $I_{ORD} = (x_1, x_2 \ldots, x_n)$.
2. Construa instância $I_{EC} = \{(x_1, x_1^2), (x_2, x_2^2), \ldots, (x_n, x_n^2)\}$.



ORD $\preccurlyeq_n$ EC

Reduzindo ORD $\preccurlyeq_n$ EC:

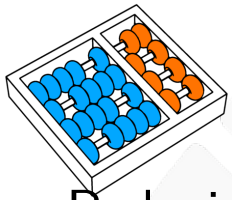
1. Considere uma instância de $I_{ORD} = (x_1, x_2 \ldots, x_n)$.
2. Construa instância $I_{EC} = \{(x_1, x_1^2), (x_2, x_2^2), \ldots, (x_n, x_n^2)\}$.



ORD $\preccurlyeq_n$ EC

Reduzindo ORD $\preccurlyeq_n$ EC:

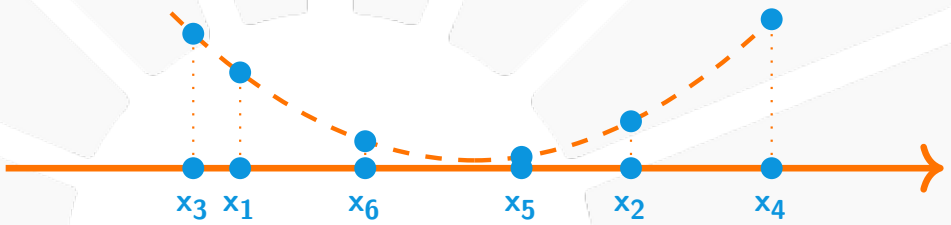
1. Considere uma instância de $I_{ORD} = (x_1, x_2 \ldots, x_n)$.
2. Construa instância $I_{EC} = \{(x_1, x_1^2), (x_2, x_2^2), \ldots, (x_n, x_n^2)\}$.

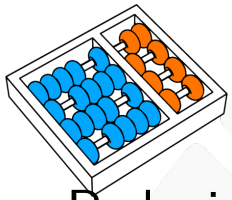


ORD $\preccurlyeq_n$ EC

Reduzindo ORD $\preccurlyeq_n$ EC:

1. Considere uma instância de $I_{ORD} = (x_1, x_2 \ldots, x_n)$.
2. Construa instância $I_{EC} = \{(x_1, x_1^2), (x_2, x_2^2), \ldots, (x_n, x_n^2)\}$.

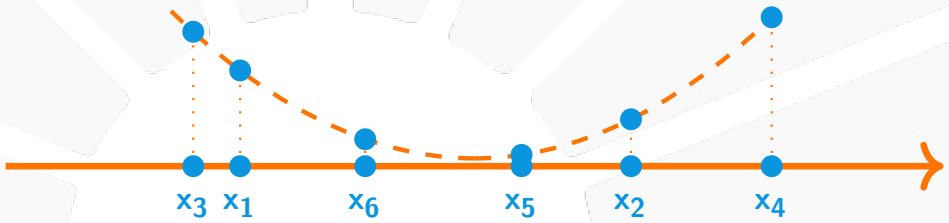




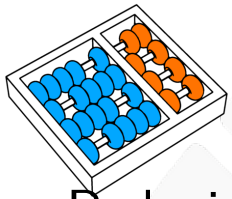
ORD $\preccurlyeq_n$ EC

Reduzindo ORD $\preccurlyeq_n$ EC:

1. Considere uma instância de $I_{ORD} = (x_1, x_2 \ldots, x_n)$.
2. Construa instância $I_{EC} = \{(x_1, x_1^2), (x_2, x_2^2), \ldots, (x_n, x_n^2)\}$.



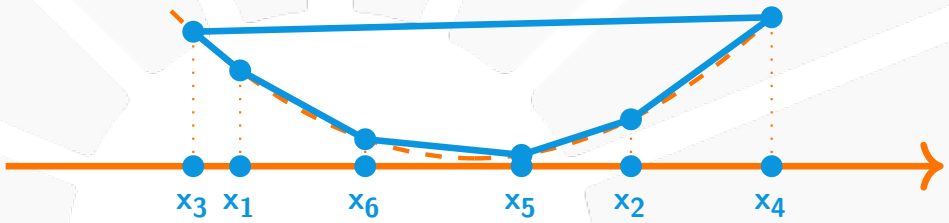
3. Resolva I_{EC} e obtenha solução S_{EC}, que é uma lista **CÍCLICA** dos vértices do polígono.



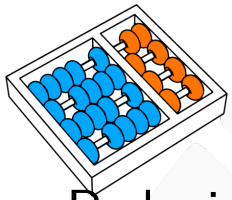
ORD $\preccurlyeq_n$ EC

Reduzindo ORD $\preccurlyeq_n$ EC:

1. Considere uma instância de $I_{ORD} = (x_1, x_2 \ldots, x_n)$.
2. Construa instância $I_{EC} = \{(x_1, x_1^2), (x_2, x_2^2), \ldots, (x_n, x_n^2)\}$.



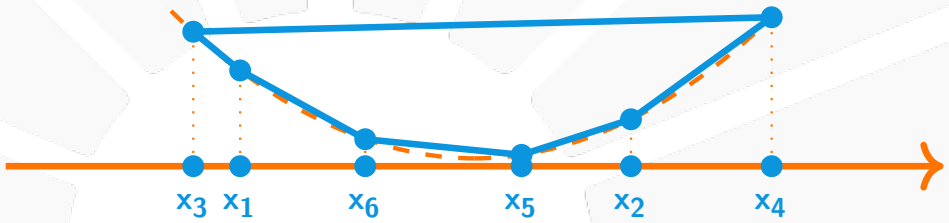
3. Resolva I_{EC} e obtenha solução S_{EC}, que é uma lista **CÍCLICA** dos vértices do polígono.



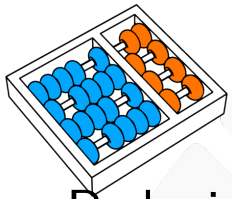
ORD $\preccurlyeq_n$ EC

Reduzindo ORD $\preccurlyeq_n$ EC:

1. Considere uma instância de $I_{ORD} = (x_1, x_2 \ldots, x_n)$.
2. Construa instância $I_{EC} = \{(x_1, x_1^2), (x_2, x_2^2), \ldots, (x_n, x_n^2)\}$.



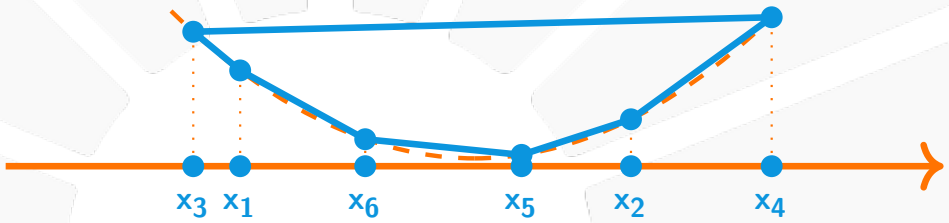
3. Resolva I_{EC} e obtenha solução S_{EC}, que é uma lista **CÍCLICA** dos vértices do polígono.
4. Determine índice i de S_{EC} do ponto com menor abcissa.



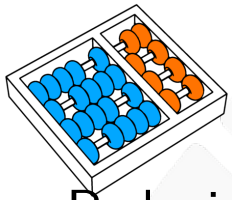
ORD $\preccurlyeq_n$ EC

Reduzindo ORD $\preccurlyeq_n$ EC:

1. Considere uma instância de $I_{ORD} = (x_1, x_2 \ldots, x_n)$.
2. Construa instância $I_{EC} = \{(x_1, x_1^2), (x_2, x_2^2), \ldots, (x_n, x_n^2)\}$.



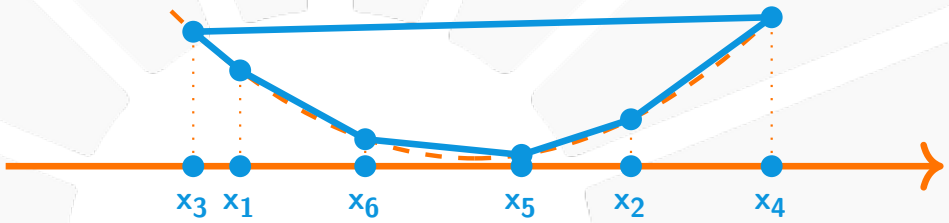
3. Resolva I_{EC} e obtenha solução S_{EC}, que é uma lista **CÍCLICA** dos vértices do polígono.
4. Determine índice i de S_{EC} do ponto com menor abcissa.
5. Liste os todos os índices a partir de i.



ORD $\preccurlyeq_n$ EC

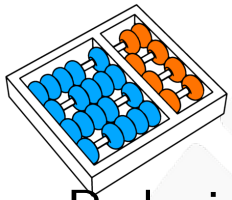
Reduzindo ORD $\preccurlyeq_n$ EC:

1. Considere uma instância de $I_{ORD} = (x_1, x_2 \ldots, x_n)$.
2. Construa instância $I_{EC} = \{(x_1, x_1^2), (x_2, x_2^2), \ldots, (x_n, x_n^2)\}$.



3. Resolva I_{EC} e obtenha solução S_{EC}, que é uma lista **CÍCLICA** dos vértices do polígono.
4. Determine índice i de S_{EC} do ponto com menor abcissa.
5. Liste os todos os índices a partir de i.

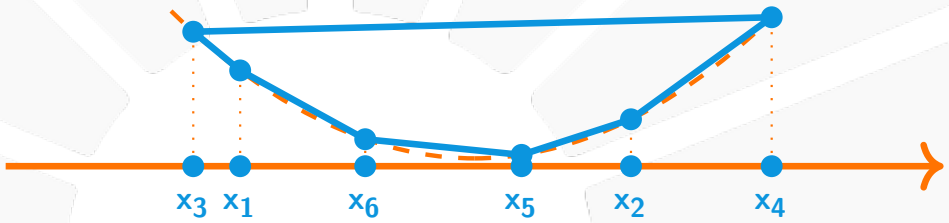
- O tempo da redução é $O(n)$.



ORD $\preccurlyeq_n$ EC

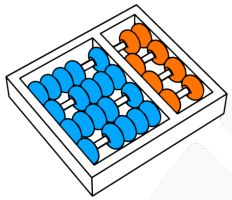
Reduzindo ORD $\preccurlyeq_n$ EC:

1. Considere uma instância de $I_{ORD} = (x_1, x_2 \ldots, x_n)$.
2. Construa instância $I_{EC} = \{(x_1, x_1^2), (x_2, x_2^2), \ldots, (x_n, x_n^2)\}$.



3. Resolva I_{EC} e obtenha solução S_{EC}, que é uma lista **CÍCLICA** dos vértices do polígono.
4. Determine índice i de S_{EC} do ponto com menor abcissa.
5. Liste os todos os índices a partir de i.

- O tempo da redução é $O(n)$.
- Portanto $\Omega(n \log n)$ também é **COTA INFERIOR** para EC.

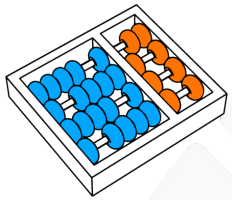


Problema de origem

Problema (Unicidade de Elementos (UE))

Entrada: *Uma sequência* $X = (x_1, x_2, \ldots, x_n)$ *de* n *elementos comparáveis.*

Saída: *Decidir se os elementos são* ***TODOS*** *distintos.*



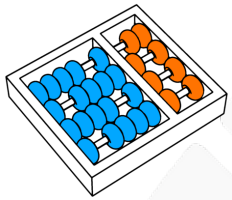
Problema de origem

Problema (Unicidade de Elementos (UE))

Entrada: *Uma sequência* $X = (x_1, x_2, \ldots, x_n)$ *de* n *elementos comparáveis.*

Saída: *Decidir se os elementos são* ***TODOS*** *distintos.*

Observações:



Problema de origem

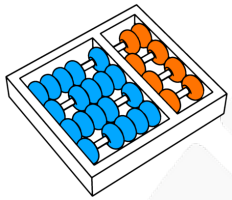
Problema (Unicidade de Elementos (UE))

Entrada: *Uma sequência* $X = (x_1, x_2, \ldots, x_n)$ *de* n *elementos comparáveis.*

Saída: *Decidir se os elementos são* ***TODOS*** *distintos.*

Observações:

- Só podemos comparar dois elementos por uma sub-rotina caixa-preta de tempo constante (chamada **ORÁCULO**).



Problema de origem

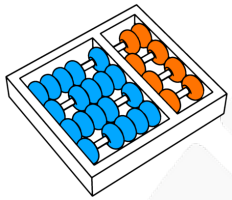
Problema (Unicidade de Elementos (UE))

Entrada: *Uma sequência* $X = (x_1, x_2, \ldots, x_n)$ *de* n *elementos comparáveis.*

Saída: *Decidir se os elementos são* ***TODOS*** *distintos.*

Observações:

- Só podemos comparar dois elementos por uma sub-rotina caixa-preta de tempo constante (chamada **ORÁCULO**).
- Esse problema tem cota inferior $\Omega(n \log n)$.



Problema de origem

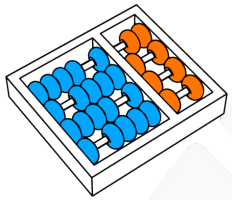
Problema (Unicidade de Elementos (UE))

Entrada: *Uma sequência* $X = (x_1, x_2, \ldots, x_n)$ *de* n *elementos comparáveis.*

Saída: *Decidir se os elementos são* ***TODOS*** *distintos.*

Observações:

- Só podemos comparar dois elementos por uma sub-rotina caixa-preta de tempo constante (chamada **ORÁCULO**).
- Esse problema tem cota inferior $\Omega(n \log n)$.
- O problema pode ser resolvido em tempo $O(n \log n)$ (como?).

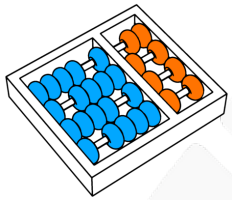


Problema de destino

Problema (Par Mais Próximo (PMP))

Entrada: *Uma coleção* $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ *de* n *pontos no plano.*

Saída: *Par de pontos* i *e* j *que estejam* ***A MENOR DISTÂNCIA****.*

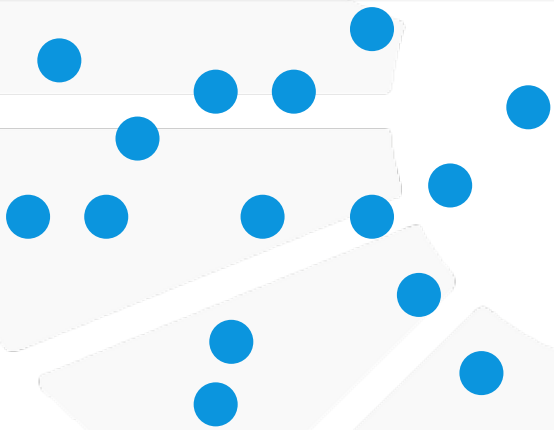


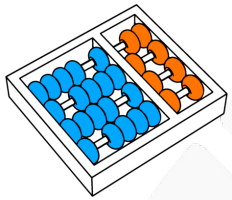
Problema de destino

Problema (Par Mais Próximo (PMP))

Entrada: *Uma coleção* $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ *de* n *pontos no plano.*

Saída: *Par de pontos* i *e* j *que estejam* ***A MENOR DISTÂNCIA****.*



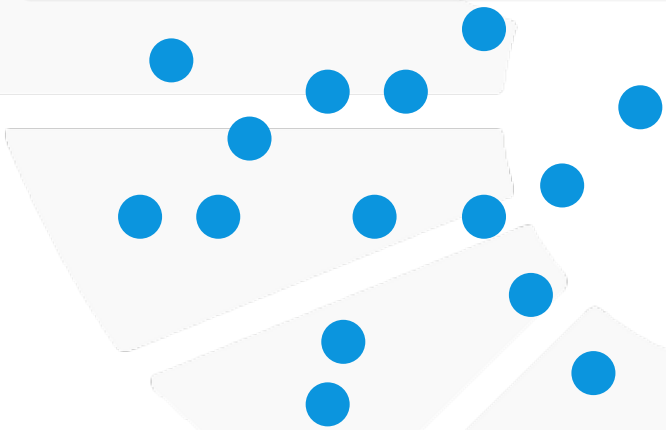


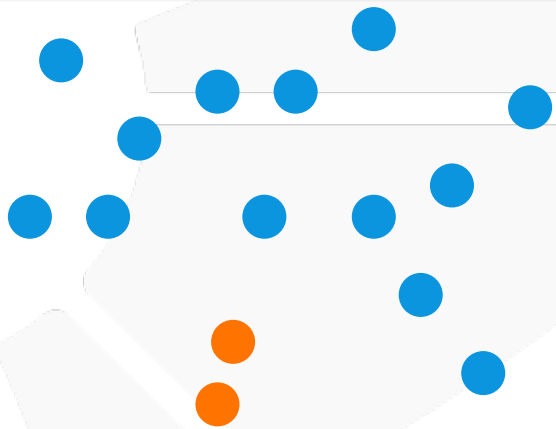
Problema de destino

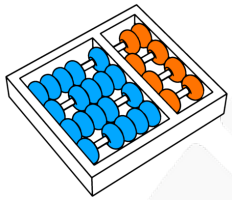
Problema (Par Mais Próximo (PMP))

Entrada: *Uma coleção* $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ *de* n *pontos no plano.*

Saída: *Par de pontos* i *e* j *que estejam* ***A MENOR DISTÂNCIA****.*





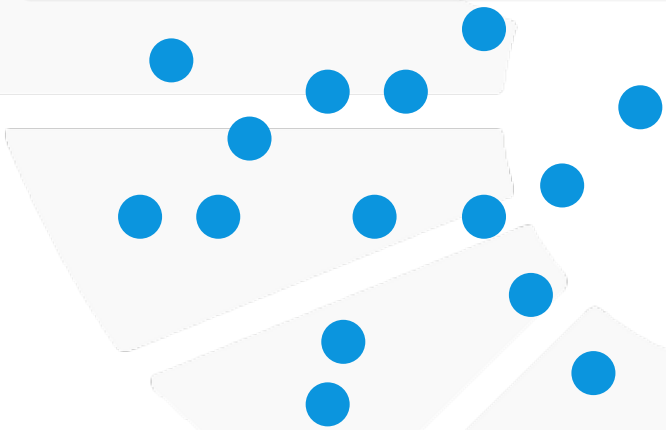


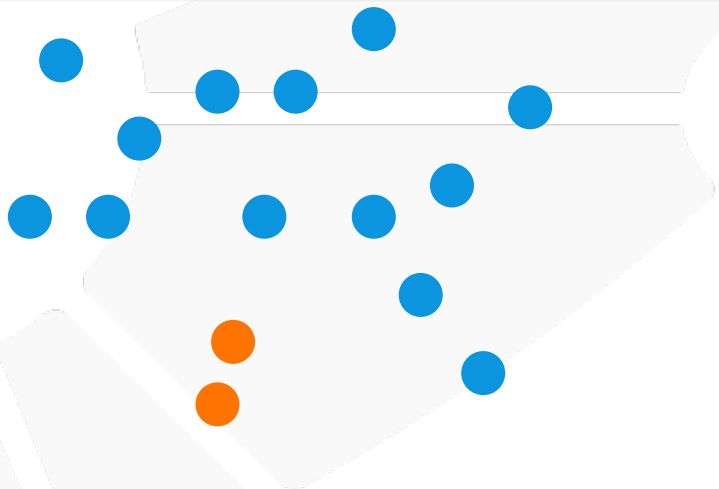
Problema de destino

Problema (Par Mais Próximo (PMP))

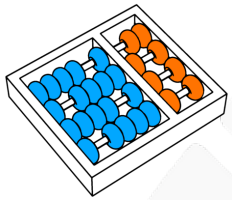
Entrada: *Uma coleção* $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ *de* n *pontos no plano.*

Saída: *Par de pontos* i *e* j *que estejam* ***A MENOR DISTÂNCIA****.*





Observação:

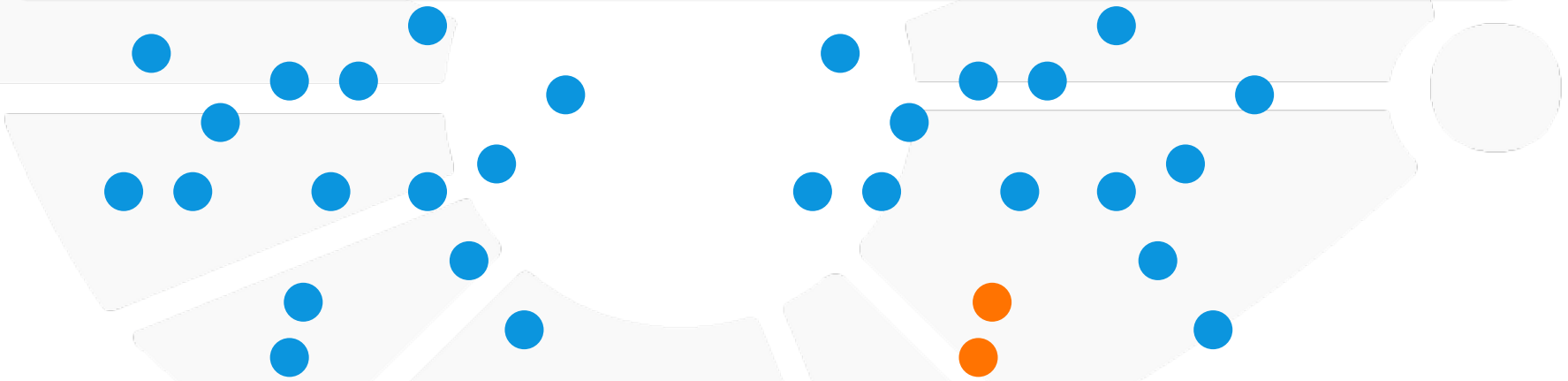


Problema de destino

Problema (Par Mais Próximo (PMP))

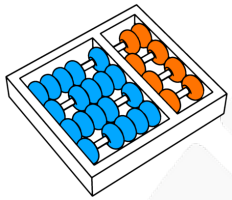
Entrada: *Uma coleção* $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ *de* n *pontos no plano.*

Saída: *Par de pontos* i *e* j *que estejam* ***A MENOR DISTÂNCIA****.*



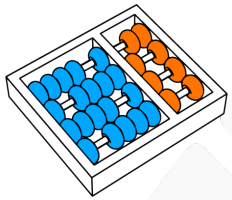
Observação:

- Pode ser resolvido em tempo $O(n \log n)$



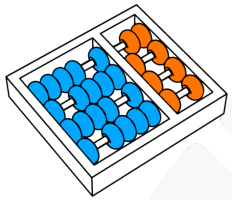
Reduzindo UE $\propto_n$ PMP

1. Considere instância $I_{UE} = (x_1, x_2, \ldots, x_n)$.



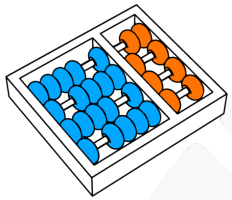
Reduzindo UE $\preccurlyeq_n$ PMP

1. Considere instância $I_{UE} = (x_1, x_2, \ldots, x_n)$.
2. Construa instância $I_{PMP} = \{(x_1, 0), (x_2, 0), \ldots, (x_n, 0)\}$.



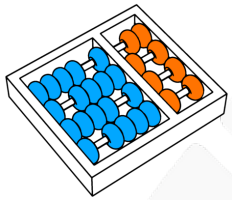
Reduzindo UE $\preceq_n$ PMP

1. Considere instância $I_{UE} = (x_1, x_2, \ldots, x_n)$.
2. Construa instância $I_{PMP} = \{(x_1, 0), (x_2, 0), \ldots, (x_n, 0)\}$.



Reduzindo UE $\propto_n$ PMP

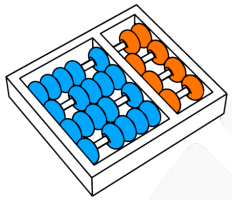
1. Considere instância $I_{UE} = (x_1, x_2, \ldots, x_n)$.
2. Construa instância $I_{PMP} = \{(x_1, 0), (x_2, 0), \ldots, (x_n, 0)\}$.



Reduzindo UE $\preceq_n$ PMP

1. Considere instância $I_{UE} = (x_1, x_2, \ldots, x_n)$.
2. Construa instância $I_{PMP} = \{(x_1, 0), (x_2, 0), \ldots, (x_n, 0)\}$.

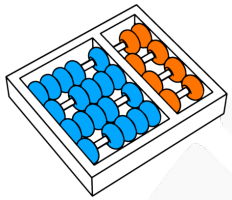
3. Resolva I_{PMP} e obtenha par de pontos $(x_i, 0), (x_j, 0)$.



Reduzindo UE $\preceq_n$ PMP

1. Considere instância $I_{UE} = (x_1, x_2, \ldots, x_n)$.
2. Construa instância $I_{PMP} = \{(x_1, 0), (x_2, 0), \ldots, (x_n, 0)\}$.

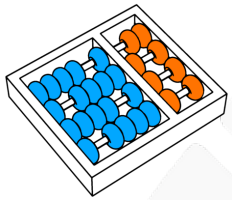
3. Resolva I_{PMP} e obtenha par de pontos $(x_i, 0), (x_j, 0)$.



Reduzindo UE $\preceq_n$ PMP

1. Considere instância $I_{UE} = (x_1, x_2, \ldots, x_n)$.
2. Construa instância $I_{PMP} = \{(x_1, 0), (x_2, 0), \ldots, (x_n, 0)\}$.

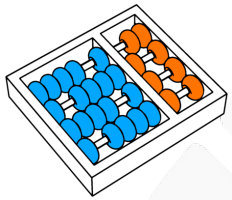
3. Resolva I_{PMP} e obtenha par de pontos $(x_i, 0), (x_j, 0)$.
4. Calcule a **DISTÂNCIA** d entre os pontos:



Reduzindo UE $\preccurlyeq_n$ PMP

1. Considere instância $I_{UE} = (x_1, x_2, \ldots, x_n)$.
2. Construa instância $I_{PMP} = \{(x_1, 0), (x_2, 0), \ldots, (x_n, 0)\}$.

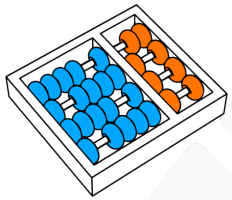
3. Resolva I_{PMP} e obtenha par de pontos $(x_i, 0), (x_j, 0)$.
4. Calcule a **DISTÂNCIA** d entre os pontos:
   (a) Se $d = 0$, então devolva NAO.



Reduzindo UE $\preccurlyeq_n$ PMP

1. Considere instância $I_{UE} = (x_1, x_2, \ldots, x_n)$.
2. Construa instância $I_{PMP} = \{(x_1, 0), (x_2, 0), \ldots, (x_n, 0)\}$.

3. Resolva I_{PMP} e obtenha par de pontos $(x_i, 0), (x_j, 0)$.
4. Calcule a **DISTÂNCIA** d entre os pontos:
   (a) Se $d = 0$, então devolva NAO.
   (b) Se $d > 0$, devolva SIM.

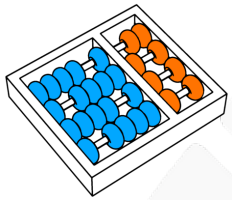


Reduzindo UE $\preceq_n$ PMP

1. Considere instância $I_{UE} = (x_1, x_2, \ldots, x_n)$.
2. Construa instância $I_{PMP} = \{(x_1, 0), (x_2, 0), \ldots, (x_n, 0)\}$.

3. Resolva I_{PMP} e obtenha par de pontos $(x_i, 0), (x_j, 0)$.
4. Calcule a **DISTÂNCIA** d entre os pontos:
   (a) Se $d = 0$, então devolva NAO.
   (b) Se $d > 0$, devolva SIM.

- O tempo da redução é $O(n)$.

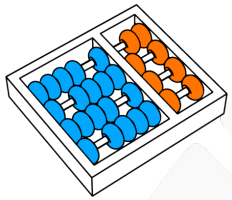


Reduzindo UE $\preceq_n$ PMP

1. Considere instância $I_{UE} = (x_1, x_2, \ldots, x_n)$.
2. Construa instância $I_{PMP} = \{(x_1, 0), (x_2, 0), \ldots, (x_n, 0)\}$.

3. Resolva I_{PMP} e obtenha par de pontos $(x_i, 0), (x_j, 0)$.
4. Calcule a **DISTÂNCIA** d entre os pontos:
   (a) Se $d = 0$, então devolva NAO.
   (b) Se $d > 0$, devolva SIM.

- O tempo da redução é $O(n)$.
- Portanto $\Omega(n \log n)$ também é **COTA INFERIOR** para PMP.

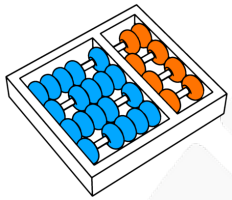


Problema de origem

Problema (3-Soma (3SUM))

Entrada: *Uma sequência* $X = (x_1, x_2, \ldots, x_n)$ *de* n *reais.*
Saída: *Determinar se existem índices distintos* i,j *e* k *tais que:*

$$x_i + x_j + x_k = 0.$$



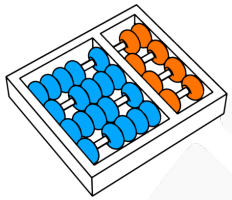
Problema de origem

Problema (3-Soma (3SUM))

Entrada: *Uma sequência* $X = (x_1, x_2, \ldots, x_n)$ *de* n *reais.*
Saída: *Determinar se existem índices distintos* i,j *e* k *tais que:*

$$x_i + x_j + x_k = 0.$$

Exemplo:



Problema de origem

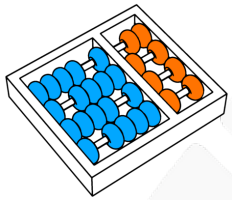
Problema (3-Soma (3SUM))

Entrada: *Uma sequência* $X = (x_1, x_2, \ldots, x_n)$ *de* n *reais.*
Saída: *Determinar se existem índices distintos* i,j *e* k *tais que:*

$$x_i + x_j + x_k = 0.$$

Exemplo:

- Instância $X = (\underline{4}, -6, \underline{1}, 8, 7, \underline{-5})$



Problema de origem

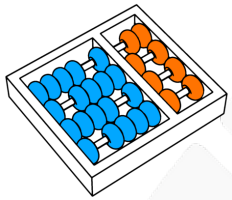
Problema (3-Soma (3SUM))

Entrada: *Uma sequência* $X = (x_1, x_2, \ldots, x_n)$ *de* n *reais.*
Saída: *Determinar se existem índices distintos* i,j *e* k *tais que:*

$$x_i + x_j + x_k = 0.$$

Exemplo:

- Instância $X = (\underline{4}, -6, \underline{1}, 8, 7, \underline{-5})$
- Solução $i = 1$, $j = 3$ e $k = 6$



Problema de origem

Problema (3-Soma (3SUM))

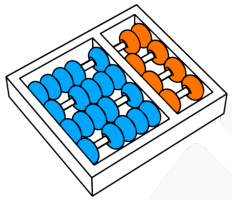
Entrada: *Uma sequência* $X = (x_1, x_2, \ldots, x_n)$ *de* n *reais.*
Saída: *Determinar se existem índices distintos* i,j *e* k *tais que:*

$$x_i + x_j + x_k = 0.$$

Exemplo:

- Instância $X = (\underline{4}, -6, \underline{1}, 8, 7, \underline{-5})$
- Solução $i = 1$, $j = 3$ e $k = 6$

Observações:



Problema de origem

Problema (3-Soma (3SUM))

Entrada: *Uma sequência* $X = (x_1, x_2, \ldots, x_n)$ *de* n *reais.*
Saída: *Determinar se existem índices distintos* i,j *e* k *tais que:*

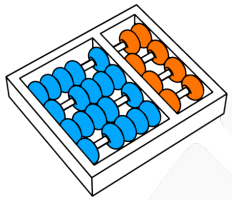
$$x_i + x_j + x_k = 0.$$

Exemplo:

- Instância $X = (\underline{4}, -6, \underline{1}, 8, 7, \underline{-5})$
- Solução $i = 1$, $j = 3$ e $k = 6$

Observações:

- Pode ser resolvido em $O(n^2)$.



Problema de origem

Problema (3-Soma (3SUM))

Entrada: *Uma sequência* $X = (x_1, x_2, \ldots, x_n)$ *de* n *reais.*
Saída: *Determinar se existem índices distintos* i,j *e* k *tais que:*

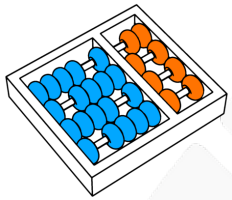
$$x_i + x_j + x_k = 0.$$

Exemplo:

- Instância $X = (\underline{4}, -6, \underline{1}, 8, 7, \underline{-5})$
- Solução $i = 1$, $j = 3$ e $k = 6$

Observações:

- Pode ser resolvido em $O(n^2)$ (como?).



Problema de origem

Problema (3-Soma (3SUM))

Entrada: *Uma sequência* $X = (x_1, x_2, \ldots, x_n)$ *de* n *reais.*
Saída: *Determinar se existem índices distintos* i,j *e* k *tais que:*

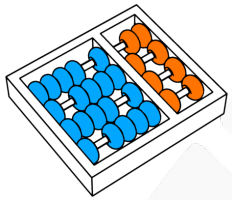
$$x_i + x_j + x_k = 0.$$

Exemplo:

- Instância $X = (\underline{4}, -6, \underline{1}, 8, 7, \underline{-5})$
- Solução $i = 1$, $j = 3$ e $k = 6$

Observações:

- Pode ser resolvido em $O(n^2)$ (como?).
- Acreditava-se que $\Omega(n^2)$ era **COTA INFERIOR**.



Problema de origem

Problema (3-Soma (3SUM))

Entrada: *Uma sequência* $X = (x_1, x_2, \ldots, x_n)$ *de* n *reais.*
Saída: *Determinar se existem índices distintos* i,j *e* k *tais que:*

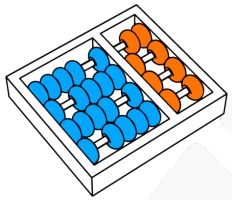
$$x_i + x_j + x_k = 0.$$

Exemplo:

- Instância $X = (\underline{4}, -6, \underline{1}, 8, 7, \underline{-5})$
- Solução $i = 1$, $j = 3$ e $k = 6$

Observações:

- Pode ser resolvido em $O(n^2)$ (como?).
- Acreditava-se que $\Omega(n^2)$ era **COTA INFERIOR**.
- Pode ser resolvido em $o(n^2)$ (Grønlund e Pettie, 2014).



Problema de origem

Problema (3-Soma (3SUM))

Entrada: *Uma sequência* $X = (x_1, x_2, \ldots, x_n)$ *de* n *reais.*
Saída: *Determinar se existem índices distintos* i,j *e* k *tais que:*

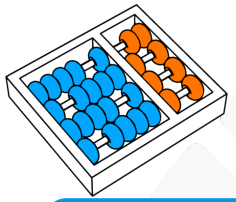
$$x_i + x_j + x_k = 0.$$

Exemplo:

- Instância $X = (\underline{4}, -6, \underline{1}, 8, 7, \underline{-5})$
- Solução $i = 1$, $j = 3$ e $k = 6$

Observações:

- Pode ser resolvido em $O(n^2)$ (como?).
- Acreditava-se que $\Omega(n^2)$ era **COTA INFERIOR**.
- Pode ser resolvido em $o(n^2)$ (Grønlund e Pettie, 2014).
- Ainda se acredita que não dá pra fazer melhor que $n^{2-\Omega(1)}$.

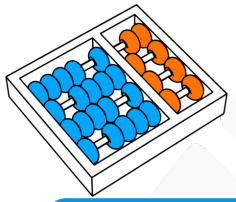


Problema de destino

Problema (Colinearidade Não Horizontal (COL))

Entrada: *Um conjunto* $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ *de* n *pontos no plano.*

Saída: *Determinar se três pontos estão em alguma* ***RETA NÃO HORIZONTAL****.*

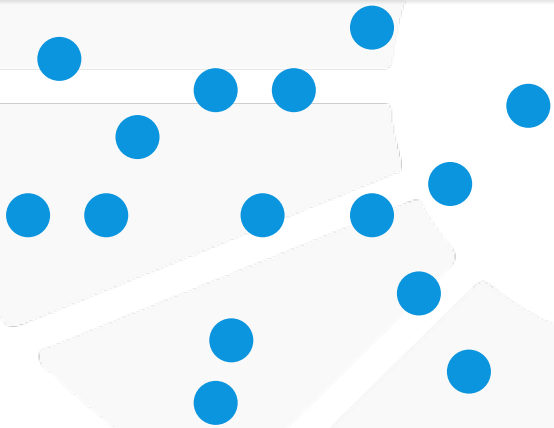


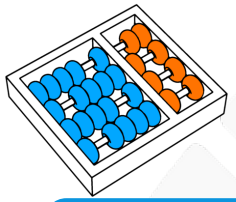
Problema de destino

Problema (Colinearidade Não Horizontal (COL))

Entrada: *Um conjunto* $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ *de* n *pontos no plano.*

Saída: *Determinar se três pontos estão em alguma* ***RETA NÃO HORIZONTAL****.*



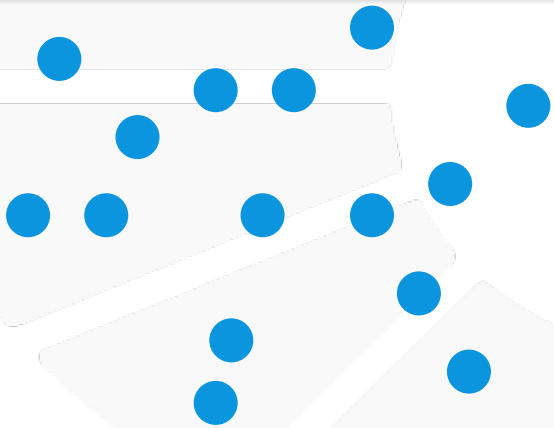


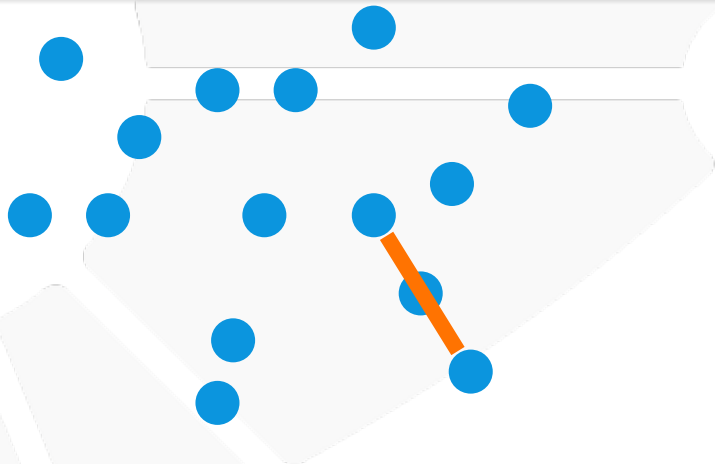
Problema de destino

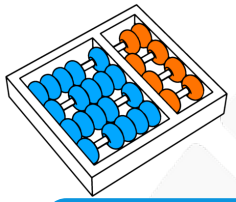
Problema (Colinearidade Não Horizontal (COL))

Entrada: *Um conjunto* $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ *de* n *pontos no plano.*

Saída: *Determinar se três pontos estão em alguma* ***RETA NÃO HORIZONTAL****.*





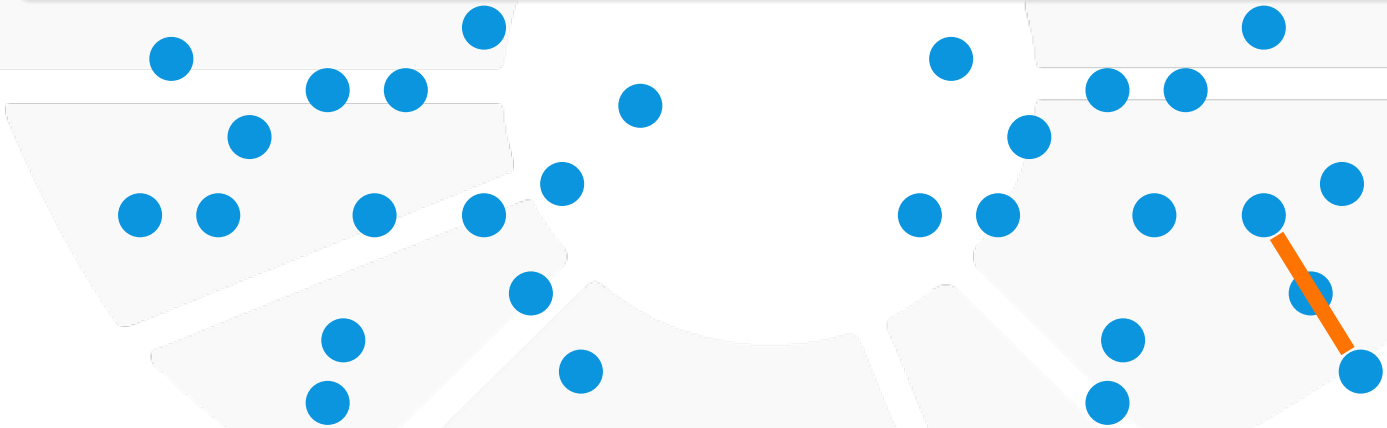


Problema de destino

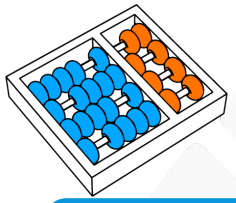
Problema (Colinearidade Não Horizontal (COL))

Entrada: *Um conjunto* $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ *de n pontos no plano.*

Saída: *Determinar se três pontos estão em alguma* ***RETA NÃO HORIZONTAL****.*



Observações:

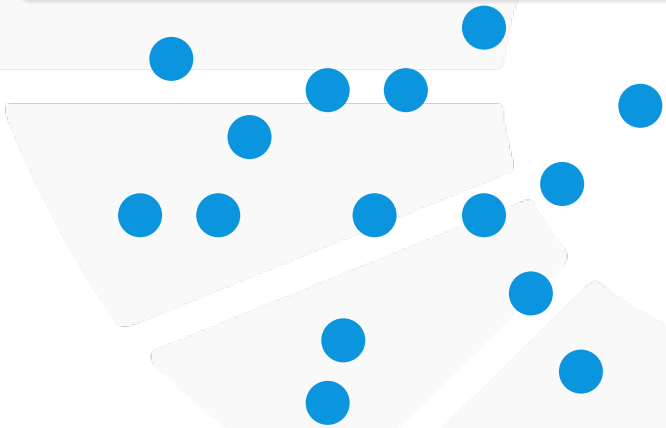


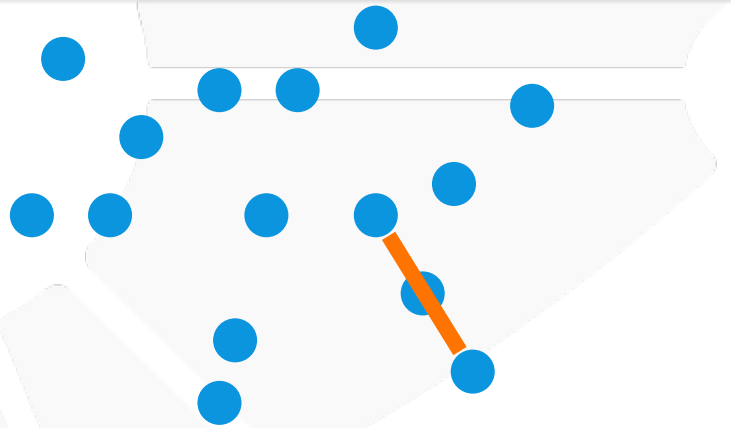
Problema de destino

Problema (Colinearidade Não Horizontal (COL))

Entrada: *Um conjunto* $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ *de* n *pontos no plano.*

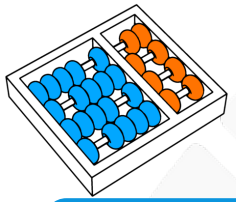
Saída: *Determinar se três pontos estão em alguma* ***RETA NÃO HORIZONTAL****.*





Observações:

- Pode ser resolvido em tempo $O(n^2)$.

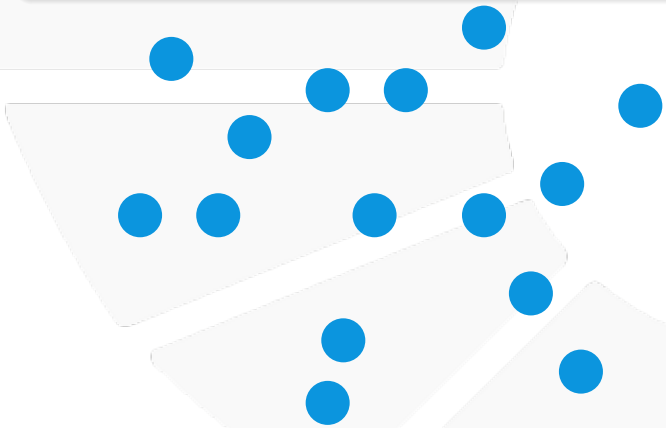


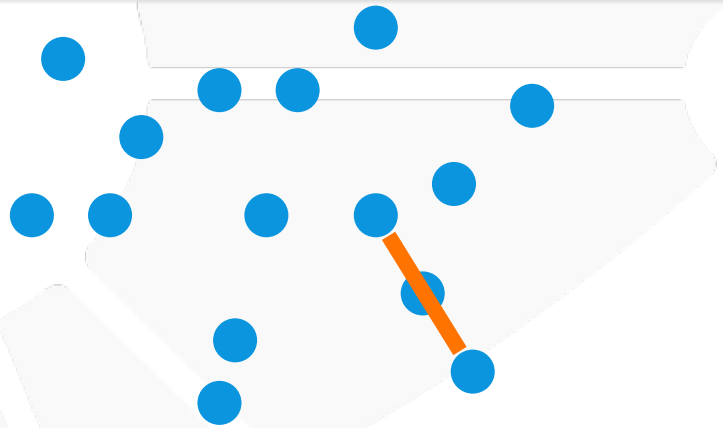
Problema de destino

Problema (Colinearidade Não Horizontal (COL))

Entrada: *Um conjunto* $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ *de* n *pontos no plano.*

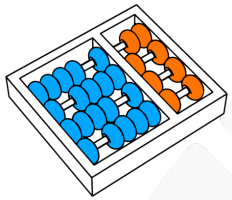
Saída: *Determinar se três pontos estão em alguma* ***RETA NÃO HORIZONTAL****.*





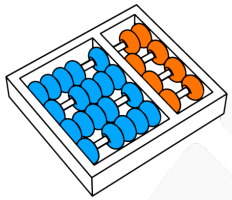
Observações:

- Pode ser resolvido em tempo $O(n^2)$.
- Acredita-se que $\Omega(n^2)$ é **COTA INFERIOR**.



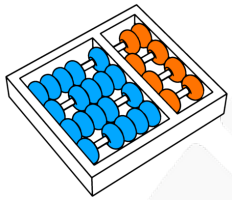
Reduzindo 3SUM $\prec_n$ COL

1. Considere instância $I_{3SUM} = (x_1, x_2, \ldots, x_n)$.



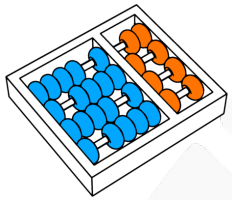
Reduzindo 3SUM $\preceq_n$ COL

1. Considere instância $I_{3SUM} = (x_1, x_2, \ldots, x_n)$.
2. Construa instância
   $I_{COL} = \{(x_i, 0), (-x_i/2, 1), (x_i, 2) : i = 1, 2, \ldots, n\}$.



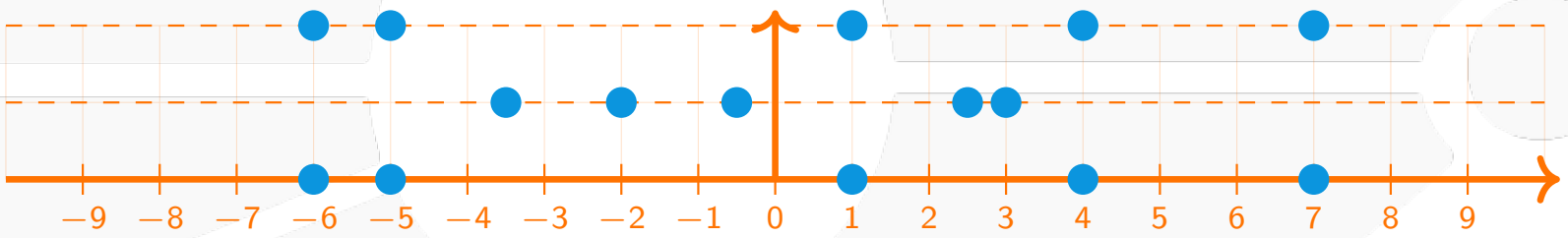
Reduzindo 3SUM $\preceq_n$ COL

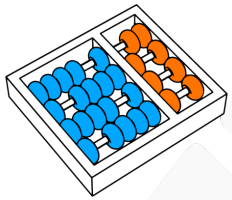
1. Considere instância $I_{3SUM} = (x_1, x_2, \ldots, x_n)$.
2. Construa instância
   $I_{COL} = \{(x_i, 0), (-x_i/2, 1), (x_i, 2) : i = 1, 2, \ldots, n\}$.
   Exemplo: $X = (4, -6, 1, 8, 7, -5)$:



Reduzindo 3SUM $\prec_n$ COL

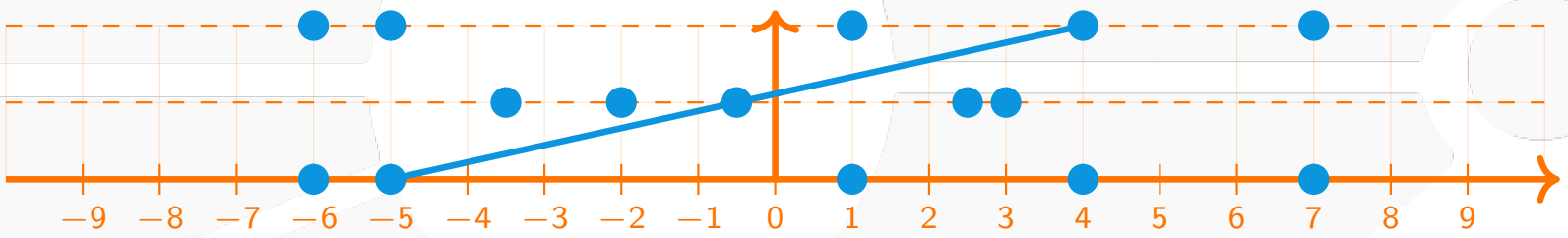
1. Considere instância $I_{3SUM} = (x_1, x_2, \ldots, x_n)$.
2. Construa instância
   $I_{COL} = \{(x_i, 0), (-x_i/2, 1), (x_i, 2) : i = 1, 2, \ldots, n\}$.
   Exemplo: $X = (4, -6, 1, 8, 7, -5)$:



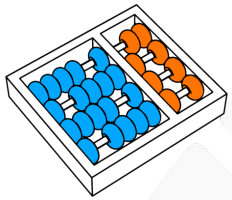


Reduzindo 3SUM $\preceq_n$ COL

1. Considere instância $I_{3SUM} = (x_1, x_2, \ldots, x_n)$.
2. Construa instância
   $I_{COL} = \{(x_i, 0), (-x_i/2, 1), (x_i, 2) : i = 1, 2, \ldots, n\}$.
   Exemplo: $X = (4, -6, 1, 8, 7, -5)$:

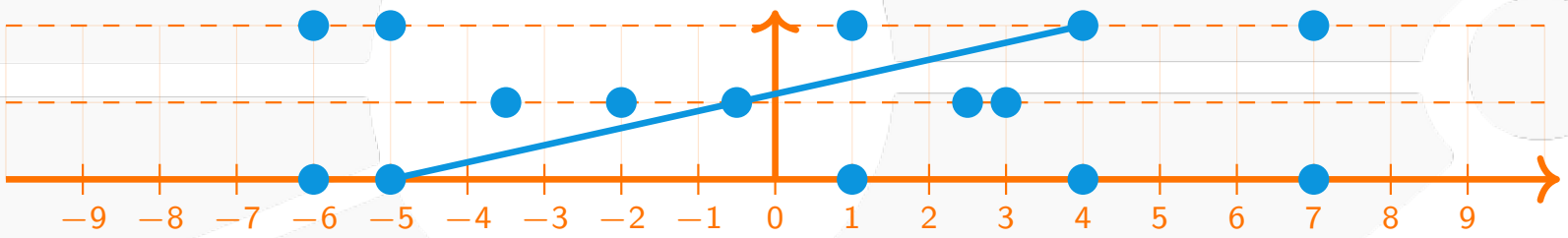


3. Resolva I_{COL} e obtenha S_{COL}.

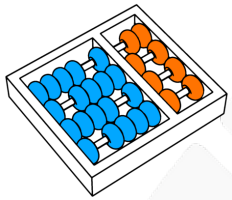


Reduzindo 3SUM $\preceq_n$ COL

1. Considere instância $I_{3SUM} = (x_1, x_2, \ldots, x_n)$.
2. Construa instância
   $I_{COL} = \{(x_i, 0), (-x_i/2, 1), (x_i, 2) : i = 1, 2, \ldots, n\}$.
   Exemplo: $X = (4, -6, 1, 8, 7, -5)$:



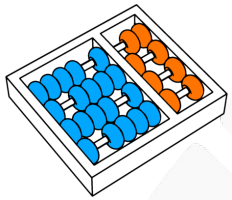
3. Resolva I_{COL} e obtenha S_{COL}.
4. Se a resposta S_{COL} for SIM, responda SIM,
   e se a resposta S_{COL} for NAO, responda NAO.



3SUM $\preccurlyeq_n$ COL (cont)

- A solução de I_{COL} (se houver) é uma tripla de pontos colineares. Claramente, cada um desses pontos deve estar em um dos eixos horizontais. Ou seja, tem a forma:

$$(x_i, 0), (-x_j/2, 1), (x_k, 2).$$

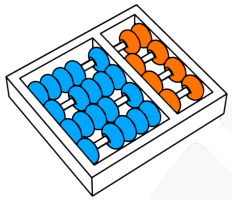


3SUM $\preccurlyeq_n$ COL (cont)

- A solução de I_{COL} (se houver) é uma tripla de pontos colineares. Claramente, cada um desses pontos deve estar em um dos eixos horizontais. Ou seja, tem a forma:

$$(x_i, 0), (-x_j/2, 1), (x_k, 2).$$

Portanto, $x_i + x_j + x_k = 0$.



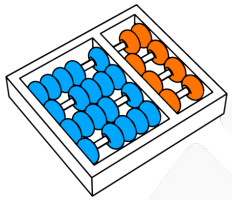
3SUM $\preccurlyeq_n$ COL (cont)

- A solução de I_{COL} (se houver) é uma tripla de pontos colineares. Claramente, cada um desses pontos deve estar em um dos eixos horizontais. Ou seja, tem a forma:

$$(x_i, 0), (-x_j/2, 1), (x_k, 2).$$

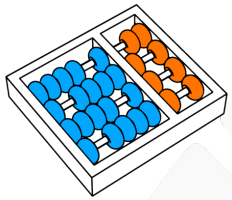
  Portanto, $x_i + x_j + x_k = 0$.

- Se $x_i + x_j + x_k = 0$, então, os pontos citados são colineares.



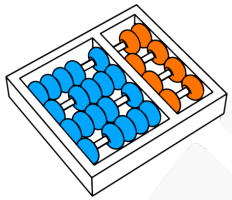
3SUM $\prec_n$ COL

- É claro que $\Omega(n)$ é uma cota inferior para 3SUM.



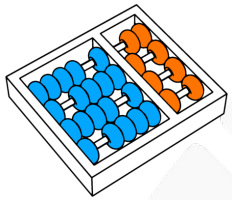
3SUM $\prec_n$ COL

- É claro que $\Omega(n)$ é uma cota inferior para 3SUM.
- E **SE** houver cota inferior $\Omega(h(n))$ maior para 3SUM?



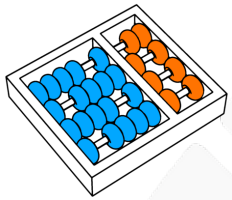
3SUM $\prec_n$ COL

- É claro que $\Omega(n)$ é uma cota inferior para 3SUM.
- E **SE** houver cota inferior $\Omega(h(n))$ maior para 3SUM?
  - A redução gasta tempo $f(n) = O(n)$.



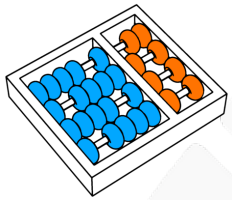
3SUM $\prec_n$ COL

- É claro que $\Omega(n)$ é uma cota inferior para 3SUM.
- E **SE** houver cota inferior $\Omega(h(n))$ maior para 3SUM?
  - A redução gasta tempo $f(n) = O(n)$.
  - Nesse caso, $f(n) = o(h(n))$.



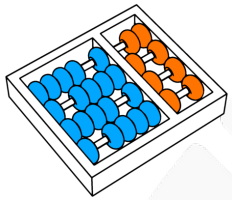
3SUM $\prec_n$ COL

- É claro que $\Omega(n)$ é uma cota inferior para 3SUM.
- E **SE** houver cota inferior $\Omega(h(n))$ maior para 3SUM?
  - A redução gasta tempo $f(n) = O(n)$.
  - Nesse caso, $f(n) = o(h(n))$.
  - Então, $\Omega(h(n))$ seria cota inferior para COL



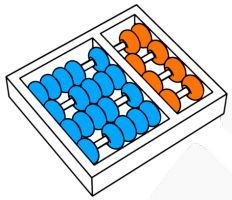
3SUM $\prec_n$ COL

- É claro que $\Omega(n)$ é uma cota inferior para 3SUM.
- E **SE** houver cota inferior $\Omega(h(n))$ maior para 3SUM?
  - A redução gasta tempo $f(n) = O(n)$.
  - Nesse caso, $f(n) = o(h(n))$.
  - Então, $\Omega(h(n))$ seria cota inferior para COL
- Mas só conhecemos a cota trivial $\Omega(n)$ para 3SUM.



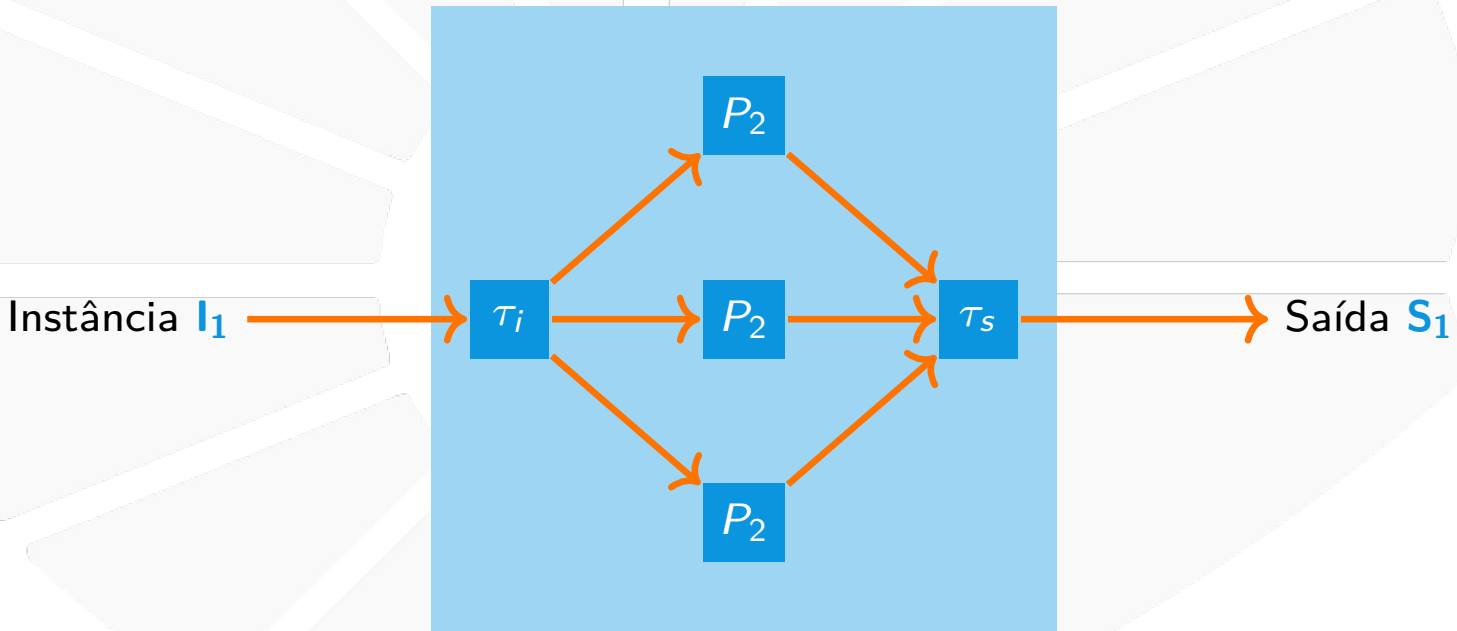
Redução de Turing

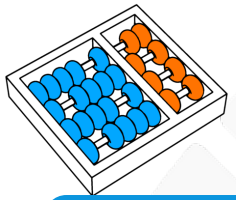
Podemos reduzir P_1 para P_2 fazendo várias aplicações de P_2.



Redução de Turing

Podemos reduzir P_1 para P_2 fazendo várias aplicações de P_2.



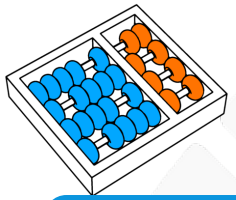


Exemplo de redução de Turing

Problema (Multiplicação de Inteiros)

Entrada: *Dois inteiros a e b.*

Saída: *O produto $a \cdot b$.*



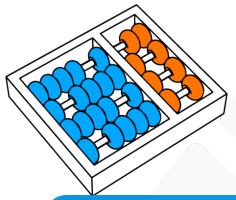
Exemplo de redução de Turing

Problema (Multiplicação de Inteiros)

Entrada: *Dois inteiros a e b.*
Saída: *O produto $a \cdot b$.*

Problema (Quadrado)

Entrada: *Um inteiro x.*
Saída: *O quadrado x^2.*



Exemplo de redução de Turing

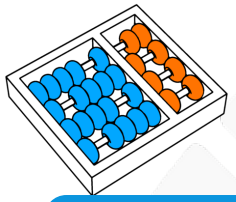
Problema (Multiplicação de Inteiros)

Entrada: *Dois inteiros a e b.*
Saída: *O produto $a \cdot b$.*

Problema (Quadrado)

Entrada: *Um inteiro x.*
Saída: *O quadrado x^2.*

Redução:



Exemplo de redução de Turing

Problema (Multiplicação de Inteiros)

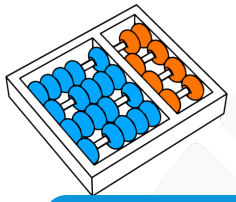
Entrada: *Dois inteiros a e b.*
Saída: *O produto $a \cdot b$.*

Problema (Quadrado)

Entrada: *Um inteiro x.*
Saída: *O quadrado x^2.*

Redução:

- Podemos reduzir Multiplicação de Inteiros para Quadrado.



Exemplo de redução de Turing

Problema (Multiplicação de Inteiros)

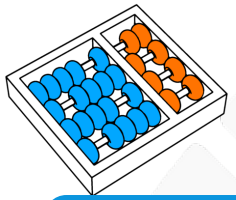
Entrada: *Dois inteiros a e b.*
Saída: *O produto $a \cdot b$.*

Problema (Quadrado)

Entrada: *Um inteiro x.*
Saída: *O quadrado x^2.*

Redução:

- Podemos reduzir Multiplicação de Inteiros para Quadrado.
- Fazemos apenas um número constante de somas, subtrações e divisão por dois



Exemplo de redução de Turing

Problema (Multiplicação de Inteiros)

Entrada: *Dois inteiros a e b.*
Saída: *O produto $a \cdot b$.*

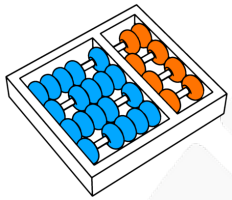
Problema (Quadrado)

Entrada: *Um inteiro x.*
Saída: *O quadrado x^2.*

Redução:

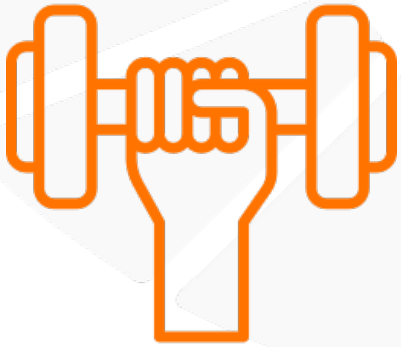
- Podemos reduzir Multiplicação de Inteiros para Quadrado.
- Fazemos apenas um número constante de somas, subtrações e divisão por dois

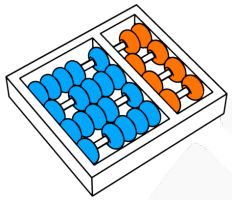
$$a \cdot b = \frac{(a+b)^2 - a^2 - b^2}{2}$$



Refletindo sobre reduções e cotas inferiores

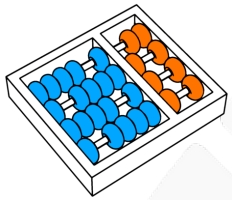
Vamos fazer alguns exercícios?





Exercício 1

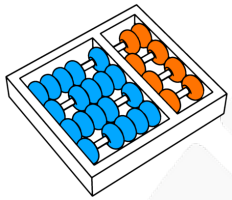
O problema 3SUMplus consiste em: dados uma sequência $X = (x_1, x_2, \ldots, x_n)$ de reais e um valor real b, determinar se existem três índices distintos i, j e k tais que $x_i + x_j + x_k = b$.



Exercício 1

O problema 3SUMplus consiste em: dados uma sequência $X = (x_1, x_2, \ldots, x_n)$ de reais e um valor real b, determinar se existem três índices distintos i, j e k tais que $x_i + x_j + x_k = b$.

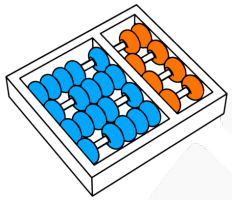
1. Mostre que 3SUM $\preccurlyeq_n$ 3SUMplus.



Exercício 1

O problema 3SUMplus consiste em: dados uma sequência $X = (x_1, x_2, \ldots, x_n)$ de reais e um valor real b, determinar se existem três índices distintos i, j e k tais que $x_i + x_j + x_k = b$.

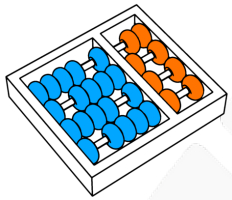
1. Mostre que 3SUM $\preccurlyeq_n$ 3SUMplus.
2. Mostre que 3SUMplus $\preccurlyeq_n$ 3SUM.



Exercício 1

O problema 3SUMplus consiste em: dados uma sequência $X = (x_1, x_2, \ldots, x_n)$ de reais e um valor real b, determinar se existem três índices distintos i, j e k tais que $x_i + x_j + x_k = b$.

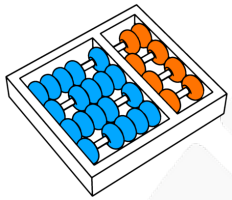
1. Mostre que 3SUM $\preccurlyeq_n$ 3SUMplus.
2. Mostre que 3SUMplus $\preccurlyeq_n$ 3SUM.
3. Suponha que o Professor Sabit Udo descobriu uma **COTA INFERIOR** de $\Omega(n^{1,9})$ para 3SUMplus. Nesse caso, quais das afirmações abaixo são verdadeiras?



Exercício 1

O problema 3SUMplus consiste em: dados uma sequência $X = (x_1, x_2, \ldots, x_n)$ de reais e um valor real b, determinar se existem três índices distintos i, j e k tais que $x_i + x_j + x_k = b$.

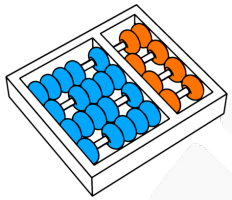
1. Mostre que 3SUM $\preccurlyeq_n$ 3SUMplus.
2. Mostre que 3SUMplus $\preccurlyeq_n$ 3SUM.
3. Suponha que o Professor Sabit Udo descobriu uma **COTA INFERIOR** de $\Omega(n^{1,9})$ para 3SUMplus. Nesse caso, quais das afirmações abaixo são verdadeiras?
   (i) Não existe algoritmo $O(n^{1,5})$ para 3SUMplus.



Exercício 1

O problema 3SUMplus consiste em: dados uma sequência $X = (x_1, x_2, \ldots, x_n)$ de reais e um valor real b, determinar se existem três índices distintos i, j e k tais que $x_i + x_j + x_k = b$.

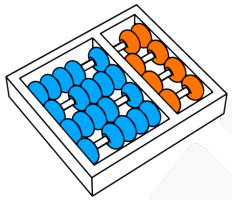
1. Mostre que 3SUM $\preccurlyeq_n$ 3SUMplus.
2. Mostre que 3SUMplus $\preccurlyeq_n$ 3SUM.
3. Suponha que o Professor Sabit Udo descobriu uma **COTA INFERIOR** de $\Omega(n^{1,9})$ para 3SUMplus. Nesse caso, quais das afirmações abaixo são verdadeiras?
   - (i) Não existe algoritmo $O(n^{1,5})$ para 3SUMplus.
   - (ii) Não existe algoritmo $O(n^{1,5})$ para 3SUM.



Exercício 1

O problema 3SUMplus consiste em: dados uma sequência $X = (x_1, x_2, \ldots, x_n)$ de reais e um valor real b, determinar se existem três índices distintos i, j e k tais que $x_i + x_j + x_k = b$.

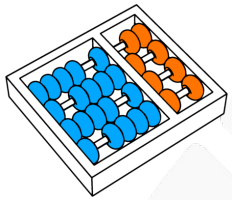
1. Mostre que 3SUM $\preccurlyeq_n$ 3SUMplus.
2. Mostre que 3SUMplus $\preccurlyeq_n$ 3SUM.
3. Suponha que o Professor Sabit Udo descobriu uma **COTA INFERIOR** de $\Omega(n^{1,9})$ para 3SUMplus. Nesse caso, quais das afirmações abaixo são verdadeiras?
   - (i) Não existe algoritmo $O(n^{1,5})$ para 3SUMplus.
   - (ii) Não existe algoritmo $O(n^{1,5})$ para 3SUM.
   - (iii) Existe um algoritmo $O(n^{1,9})$ para 3SUMplus.



Exercício 1

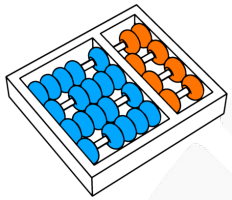
O problema 3SUMplus consiste em: dados uma sequência $X = (x_1, x_2, \ldots, x_n)$ de reais e um valor real b, determinar se existem três índices distintos i, j e k tais que $x_i + x_j + x_k = b$.

1. Mostre que 3SUM $\preccurlyeq_n$ 3SUMplus.
2. Mostre que 3SUMplus $\preccurlyeq_n$ 3SUM.
3. Suponha que o Professor Sabit Udo descobriu uma **COTA INFERIOR** de $\Omega(n^{1,9})$ para 3SUMplus. Nesse caso, quais das afirmações abaixo são verdadeiras?
   - (i) Não existe algoritmo $O(n^{1,5})$ para 3SUMplus.
   - (ii) Não existe algoritmo $O(n^{1,5})$ para 3SUM.
   - (iii) Existe um algoritmo $O(n^{1,9})$ para 3SUMplus.
   - (iv) Existe um algoritmo $O(n^{1,9})$ para 3SUM.



Exercício 2

Considere os problemas:



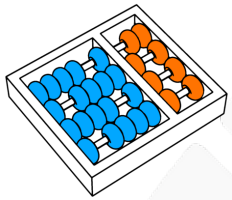
Exercício 2

Considere os problemas:

Problema (Sistema de Representantes Distintos (SRD))

Entrada: *Uma coleção de conjuntos* $S_1, \ldots, S_k$.

Saída: *Conjunto* $R = \{r_1, \ldots, r_k\}$ *tal que* $r_i \in S_i$ *para* $i = 1, \ldots, k$.



Exercício 2

Considere os problemas:

Problema (Sistema de Representantes Distintos (SRD))

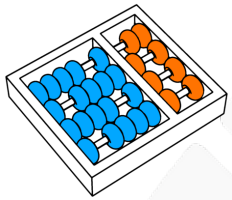
Entrada: *Uma coleção de conjuntos* $S_1, \ldots, S_k$.

Saída: *Conjunto* $R = \{r_1, \ldots, r_k\}$ *tal que* $r_i \in S_i$ *para* $i = 1, \ldots, k$.

Problema (Emparelhamento Máximo (EM))

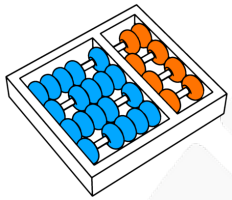
Entrada: *Um grafo bipartido* $G = (X \cup Y, E)$ *com bipartição* X *e* Y.

Saída: *Um subconjunto de arestas* M *que não compartilham vértices, tal que* $|M|$ *seja máximo.*



Exercício 2

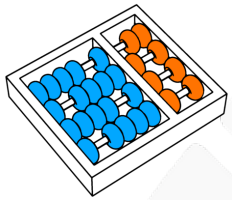
Mostre que SRD $\preccurlyeq$ EM.



Exercício 2

Mostre que SRD $\preccurlyeq$ EM.

Um fato útil, pode ser o seguinte teorema:



Exercício 2

Mostre que SRD $\preccurlyeq$ EM.

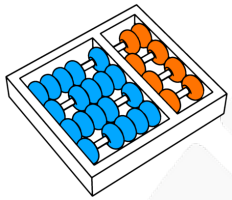
Um fato útil, pode ser o seguinte teorema:

Teorema (Teorema de Hall)

Uma coleção $S_1, \ldots, S_k$ tem um SRD se e somente se

$$\left|\{S_{i_1} \cup \cdots \cup S_{i_m}\}\right| \geq m$$

para ***TODA*** *coleção $\{i_1, \ldots, i_m\} \subseteq \{1, 2, 3, \ldots, k\}$.*

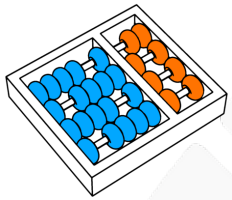


Exercício 3

Considere os seguintes problemas:

Problema (Edição de String)

Entrada: *Duas strings A e B.*
Saída: *Menor sequência de operações para transformar A em B, onde as possíveis operações são:* ***inserção*** *de um caractere,* ***remoção*** *de um caractere, ou* ***troca*** *de um caractere por outro.*



Exercício 3

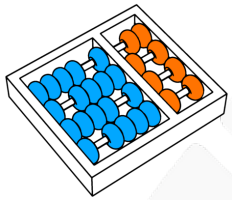
Considere os seguintes problemas:

Problema (Edição de String)

Entrada: *Duas strings A e B.*
Saída: *Menor sequência de operações para transformar A em B, onde as possíveis operações são:* ***inserção*** *de um caractere,* ***remoção*** *de um caractere, ou* ***troca*** *de um caractere por outro.*

Problema (Caminho Mínimo)

Entrada: *Um grafo direcionado $G(V, E)$, um peso $c_{ij} \geq 0$ para cada aresta $(i, j) \in E$, dois vértices s e t.*
Saída: *Um caminho de s a t em G de comprimento mínimo.*



Exercício 3

Considere os seguintes problemas:

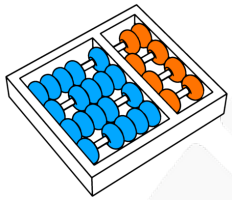
Problema (Edição de String)

Entrada: *Duas strings A e B.*
Saída: *Menor sequência de operações para transformar A em B, onde as possíveis operações são:* ***inserção*** *de um caractere,* ***remoção*** *de um caractere, ou* ***troca*** *de um caractere por outro.*

Problema (Caminho Mínimo)

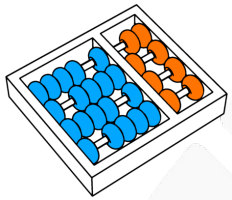
Entrada: *Um grafo direcionado $G(V, E)$, um peso $c_{ij} \geq 0$ para cada aresta $(i, j) \in E$, dois vértices s e t.*
Saída: *Um caminho de s a t em G de comprimento mínimo.*

Mostre como reduzir Edição de String a Caminho Mínimo.



Exercício 4

Considere os seguintes problemas:



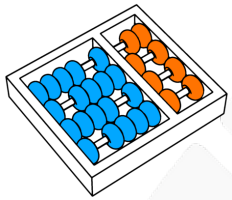
Exercício 4

Considere os seguintes problemas:

Problema (Ordenação)

Entrada: *Uma sequência de números naturais distintos* $x_1, x_2, \ldots, x_n$.

Saída: *Uma permutação ordenada dos números de entrada.*



Exercício 4

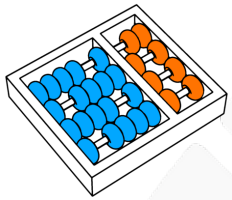
Considere os seguintes problemas:

Problema (Ordenação)

Entrada: *Uma sequência de números naturais distintos* $x_1, x_2, \ldots, x_n$.
Saída: *Uma permutação ordenada dos números de entrada.*

Problema (Codificação de Huffman)

Entrada: *Um alfabeto C e uma tabela de frequências f.*
Solução: *Uma codificação de comprimento variável que minimize o tamanho do texto codificado.*



Exercício 4

Considere os seguintes problemas:

Problema (Ordenação)

Entrada: *Uma sequência de números naturais distintos* $x_1, x_2, \ldots, x_n$.
Saída: *Uma permutação ordenada dos números de entrada.*

Problema (Codificação de Huffman)

Entrada: *Um alfabeto C e uma tabela de frequências f.*
Solução: *Uma codificação de comprimento variável que minimize o tamanho do texto codificado.*

Mostre como reduzir Ordeação para Codificação de Huffman.

Reduções

MC558 - Projeto e Análise de Algoritmos II

01/24

11

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

UNICAMP