Algoritmos

Pedro Hokama

Fontes

- [*clrs*] Algoritmos: Teoria e Prática (Terceira Edição) Thomas H. Cormen, Charles Eric Leiserson, Ronald Rivest, Ronald L. Rivest e Clifford Stein.
- [*timr*] Algorithms Illuminated Series, Tim Roughgarden

Apresentação Baseada:

- Stanford Algorithms
  `https://www.youtube.com/playlist?list=PLXFMmlk03Dt7Q0xr1PIAriY5623cKiH7V`
  `https://www.youtube.com/playlist?list=PLXFMmlk03Dt5EMI2s2WQBsLsZl7A5HEK6`
- Conjunto de Slides dos Professores Cid C. de Souza, Cândida N. da Silva, Orlando Lee, Pedro J. de Rezende
- Conjunto de Slides do Professores Cid C. de Souza para a disciplina MO420

Qualquer erro é de minha responsabilidade.

Classes de Complexidade

- Formalmente as classes de complexidade P, NP, NP-completo e outras, são melhor definidas sobre os problemas de decisão, para os quais a resposta é simplesmente **SIM** ou **NÃO**. Mas os problemas tem uma forte relação entre si.
  - Caminho mínimo: Dado G, um vértice s e um número k existe um caminho de s para qualquer outro vértice com custo no máximo k?
  - MST: Dado G e um inteiro k, existe uma Árvore geradora mínima de custo no máximo k?
  - Compressão de Texto: Dado um texto T e um número k, existe uma compressão desse texto com no máximo k bits?

Classes de Complexidade

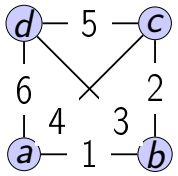
- O problemas que podem ser decididos em tempo polinomial formam a classe de complexidade

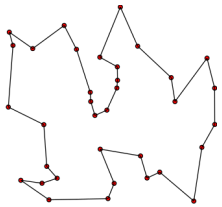
$$P$$

O Problema do Caixeiro Viajante

Travelling Salesman Problem - TSP

- Dado um grafo com custos nas arestas, encontrar um ciclo que passa por todos os vértices exatamente uma vez de custo mínimo.





- Versão de decisão: Dado um grafo com custos nas arestas e um valor W, decidir se existe um ciclo que passa por todos os vértices exatamente uma vez de custo $\leq W$.

A classe *NP*

- **Definição:** Um problema A está em NP se para qualquer instância x de A para o qual a resposta seja "sim", existe um certificado y de tamanho polinomial, que pode ser verificado em tempo polinomial, provando que a resposta de x de fato é "sim".

A classe NP

- Dado um grafo G, existe uma árvore geradora de custo $\leq W$
  - Um certificado, é a própria árvore
- Dado um texto T, existe uma compressão que usa $\leq B$ bits
  - Um certificado é o próprio texto comprimido
- Todo problema em P está em NP

- Dado um grafo G, decidir se existe um ciclo hamiltoniano.
  - Um certificado é o próprio ciclo
- Dado um conjunto de itens I e uma mochila de capacidade W, decidir se cabe na mochila um subconjunto de itens de valor $\geq K$.
  - Um certificado é a própria coleção de itens.

A classe NP

- Todo problema em NP pode ser resolvido por força-bruta em tempo exponencial.
- Gerar um número exponencial de soluções e verificar cada uma em tempo polinomial.
- A maioria dos problemas (computáveis) que encontramos está em NP

- **Definição:** Dada uma classe de problemas C um problema é C-**difícil** se ele é tão difícil quanto qualquer outro problema em C. Ou seja, existe uma redução de tempo polinomial de qualquer problema em C para ele.
- Um problema é C-Completo se é C-difícil e pertence a C

A classe NP-Completo

- Será que existem problemas NP-Completos?
- E se existirem será que existe um algoritmo polinomial para resolve-los?
- A maioria dos Ciências da Computação acredita que tal algoritmo não existe.

- Será que existem problemas *NP*-Completos?
- E se existirem será que existe um algoritmo polinomial para resolve-los?
- A maioria dos Ciências da Computação acredita que tal algoritmo não existe.

História

- Jack Edmonds é um Matemático e Cientista da Computação Estadunidense
- Graduado em 1957 na George Washington University e Pós-graduado na University of Maryland em 1959.

História

- Trabalhou no National Institute of Standards and Technology de 1959 até 1969
- Em 1965 em um artigo chamado "Paths, Trees and Flowers" conjecturou que não existe um algoritmo em tempo polinomial para o TSP.

História

- Stephen Arthur Cook é um cientista da computação e matemático Estadunidense-Canadense
  - Obteve em 1962 e 1966 seu mestrado e doutorado pela Universidade de Harvard
  - Em 1970 mostrou que existem Problemas *NP*-completos

História

- Leonid Levin é um matemático e cientista da computação Soviético
  - Obteve seu mestrado e doutorado em 1970 e 1972 na universidade de Moscou
  - Em 1972 mostrou a existência dos problemas *NP*-completos.
- O teorema de Cook-Levin afirma que o problema da satisfabilidade boleana é NP-Completo.

História

- Richard Karp é um cientista da computação Estadunidense.
- Obteve seu mestrado e doutorado em matemática aplicada em 1956 e 1959 em Harvard.

História

- Trabalhou na IBM e foi professor de ciência da computação e Pesquisa Operacional na Universidade da Califórnia, Berkeley.
- Mostrou 21 problemas que eram NP-completos. E a partir desses milhares foram provados.‘

Problemas *NP*-Completos

- Suponha então que você se deparou com um problema π, e tentou todas as técnicas que você aprendeu, mas não obteve um algoritmo polinomial.
- Mais uma ferramenta essencial para a nossa caixa: talvez seu problema seja *NP*-Completo. Receita para provar:
  1. Provar que ele pertence a *NP*
  2. Provar que ele é *NP*-difícil,
     - ⋆ Escolha um problema π' que sabidamente é *NP*-completo
     - ⋆ Faça uma redução (polinomial) de π' para π.

Problemas *NP*-Completos

- Qual problema π' escolher?
- 21 problemas de Karp
- Cap. 34 do CLRS
- Garey e Johnson, Computers and Intractability, 1979

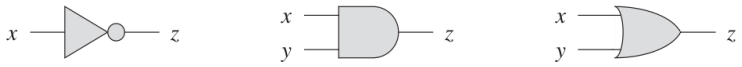
Um primeiro problema *NP*-completo

Satisfabilidade de Circuito

- O teorema de Cook-Levin é um pouco complicado (para mim, pelo menos).
- Ao invés disso vamos argumentar (um pouco informalmente) que de fato existe um problema *NP*-completo.
- O nosso primeiro problema será o problema da satisfabilidade de circuitos.

Satisfabilidade de Circuito

- Um circuito é formado por portas lógicas ligadas por fios.
- Em um extremo temos os fios de entrada e no outro temos os fios de saída.
- Existem três portas básicas. a porta NOT, AND e OR



x	$\neg x$
0	1
1	0

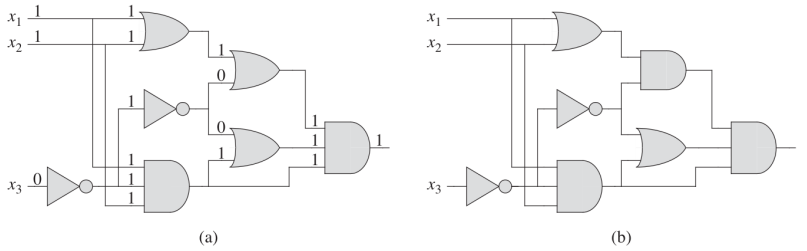
x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

Satisfabilidade de Circuito

Problema da Satisfabilidade de Circuito

Dado um circuito C com fios de entrada e uma saída. Existe alguma atribuição dos valores de entrada que tornam a saída verdadeira?



- O primeiro circuito tem uma atribuição que a torna verdadeira

Satisfabilidade de Circuito

Teorema

O Problema da Satisfabilidade de Circuitos é NP-completo.

- Primeiro provar que o Problema da Satisfabilidade de Circuitos é *NP*
- Depois provar que o Problema da Satisfabilidade de Circuitos é *NP*-Difícil.

Lema

O Problema da Satisfabilidade de Circuitos $\in$ NP.

Prova: Dado um circuito C e uma atribuição dos valores de cada fio de C. Um algoritmo A pode verificar cada porta lógica e verificar se os fios de entrada estão correspondendo corretamente ao fio de saída daquela porta. E verifica se o fio de saída do circuito é 1.
Se C é viável ele tem um certificado, se C não é viável nenhum certificado pode enganar A. O algoritmo A roda em tempo polinomial. Então verificamos que o Problema da Satisfabilidade de Circuitos $\in$ *NP*.

Satisfabilidade de Circuito

Lema

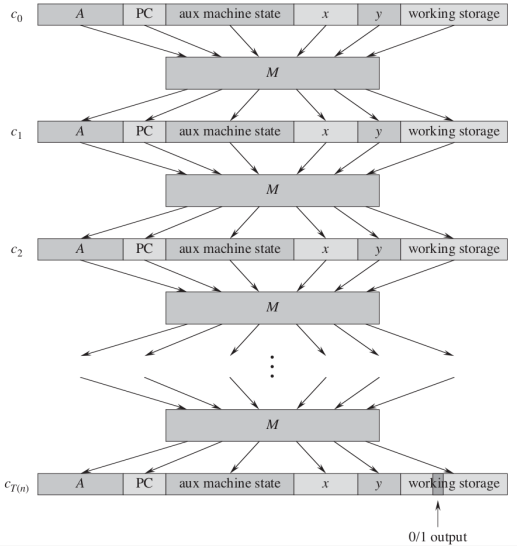
O Problema da Satisfabilidade de Circuitos é NP-Difícil.

- Primeiramente vamos relembrar de como funciona um computador.
- Um programa de computador é uma sequencia de instruções guardadas na memória do computador.
- Uma instrução tipicamente é composta da operação a ser executada, do endereço da memória dos operadores e do endereço onde vai ser armazenado o resultado.

- Um pedaço especial da memória é o **contador de programa** que sabe qual é a próxima instrução a ser executada.
- Sempre que uma instrução é executada, o contador é incrementado ou sobrescrito (no caso de um desvio)
- A qualquer momento, a memoria do computador (RAM, contadores, registradores, etc) guarda uma **configuração** completa de um passo na execução do programa.

Satisfabilidade de Circuito

- Agora considere um problema *NP* qualquer.
- Por definição existe um algoritmo A que recebe uma instância x e um certificado y e devolve SIM (1) se aquele certificado comprova que a solução de x é SIM.
- Então vamos simular a execução desse algoritmo!



- Mas o computador M nada mais é do que um circuito lógico.
- O número de réplicas é polinomial no tamanho de x
- Então se removermos o y, e deixar entradas livres. Temos um circuito que só é satisfeito com a entrada adequada.

3-CNF-SAT

- Uma formula booleana é composta
  - de variáveis booleanas $x_1, x_2, \ldots, x_n$,
  - de conectivos $\neg$ (NOT), $\vee$ (OR), $\wedge$ (AND),
  - Parenteses.
- Uma variável ou sua negação são chamadas **literais**.
- Uma fórmula booleana está na forma normal 3-conjuntiva se
  - está dividido em clausuras
  - cada clausura tem 3 literais conectas por ORs
  - as clausuras estão conectadas por ANDs.

$(x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x1 \vee \neg x3 \vee \neg x4)$

3-CNF-SAT

3-CNF-SAT

Dado uma fórmula booleana na forma normal 3-conjuntiva, decidir se existe alguma atribuição das variáveis que torna a formula verdadeira.

Teorema

*O 3-CNF-SAT é **NP**-Completo*

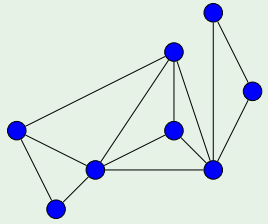
Prova: Veremos depois.

Problema do CLICK

Problema do CLICK

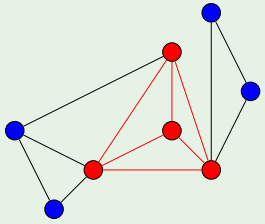
Dado um grafo não orientado $G = (V, E)$, e um inteiro k decidir se existe um subgrafo G' induzido de G que tenha k vértices e seja completo.

Exemplo



Existe uma click de tamanho 4?

Exemplo



Existe uma click de tamanho 4?

O Problema do CLICK

Será que CLICK é NP-Completo?

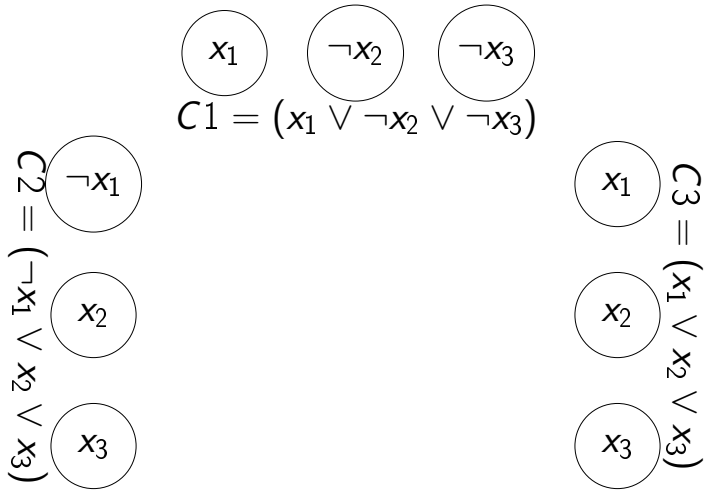
- CLICK $\in$ NP, basta apresentar o conjunto de vértices
- Vamos reduzir o 3-CNF-SAT para CLICK
  - Devemos converter qualquer instancia do 3-CNF-SAT para o problema do CLICK
  - Essa redução deve ter tempo polinomial
  - Aqui vamos mostrar o algoritmo da redução em um exemplo, mas é necessário garantir que funciona para qualquer instância.

3-CNF-SAT $\leq_p$ CLICK

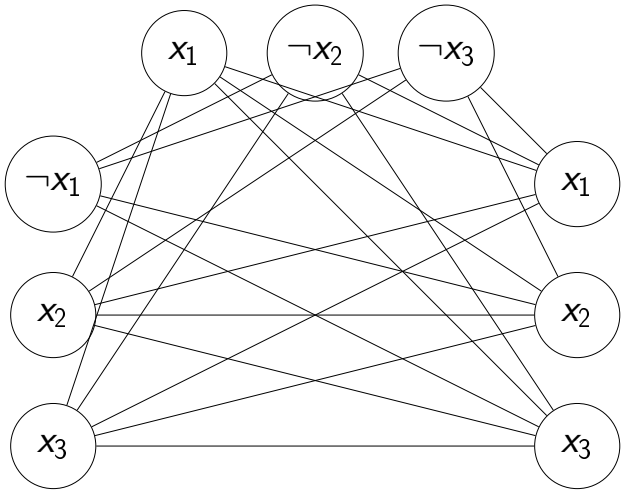
Considere uma formula booleana com k clausulas na forma normal 3-conjuntiva.

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$

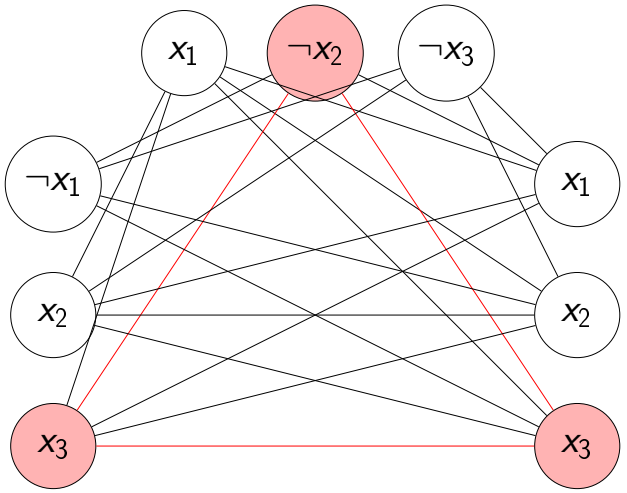
$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$



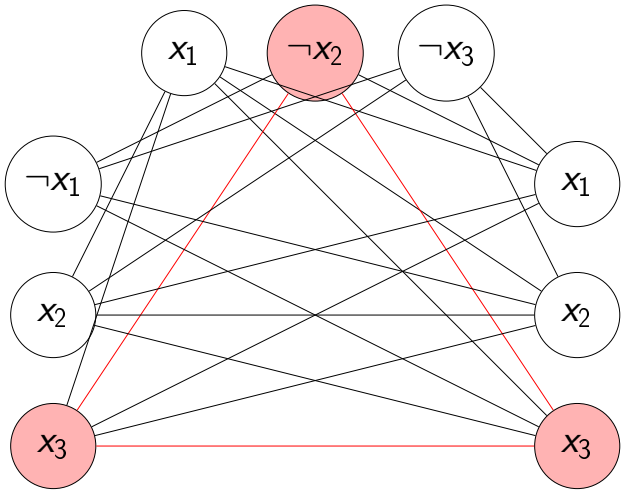
$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$



$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$

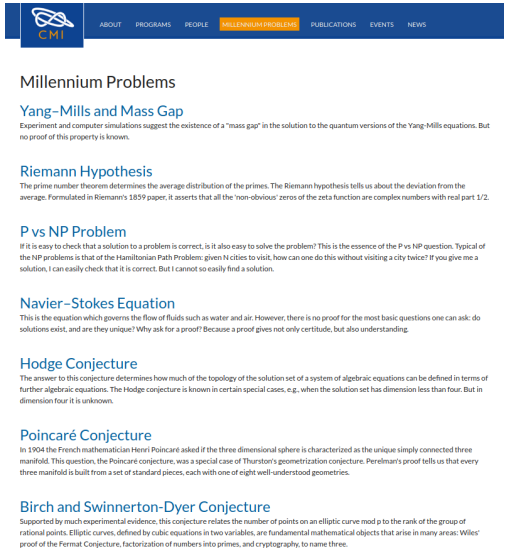


$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$



A questão P vs NP

- Seria $P = NP$?
- Bastaria mostrar 1 algoritmo com tempo de execução polinomial para 1 problema NP-completo.
- Conjectura-se (fortemente) que $P \neq NP$.
- Mas também não foi provado.



O nome NP

- O nome NP deriva de Non-deterministic Polynomial Time
- São problemas que podem ser resolvidos em tempo polinomial em uma máquina de Turing não determinística.
- Informalmente, considere uma máquina que conseguisse testar todas as soluções simultaneamente.

Problemas Intratáveis

- O que podemos fazer se um problema é *NP*-completo? (sem ser desistir)
- Na verdade muita coisa pode ser feita, e é disso que trataremos nessa disciplina.
- Veremos algumas estratégias:

Casos Especiais

- Muitas vezes um problema geral pode ser *NP*-completo mas alguns casos especiais podem ser mais fáceis de resolver
  1. Encontrar um conjunto independente máximo é *NP*-completo. Mas no grafo caminho é fácil.
  2. O problema da Mochila binária é *NP*-completo, mas o da mochila fracionária é fácil.
  3. Na mochila binária, se a capacidade for polinomial no número de itens o problema também é fácil
  4. Satisfabilidade de fórmulas booleanas é *NP*-completo, mas o 2-CNF-SAT é fácil

Heurísticas

- Podemos abrir mão de encontrar a solução ótima, e tentar encontrar uma solução boa o bastante.
- Normalmente heurísticas são rápidas.
- Geralmente não possuem garantias do quão próximas estão da solução ótima

Algoritmos Aproximados

- Podemos abrir mão de encontrar a solução ótima, e tentar encontrar uma solução boa o bastante.
- Executam em tempo polinomial.
- Possuem garantias do quão próximas estão da solução ótima.
- Por exemplo: A solução de um algoritmo aproximado vai estar no máximo a um fator α da solução ótima.

Algoritmos Exatos

- Busca a solução ótima.
- Desiste do tempo polinomial.
- A ideia é reduzir ao máximo a complexidade, e aumentar ao máximo o tamanho das instâncias que conseguimos resolver.
  - Branch-and-Bound
  - Programação Linear Inteira
  - Programação por restrições