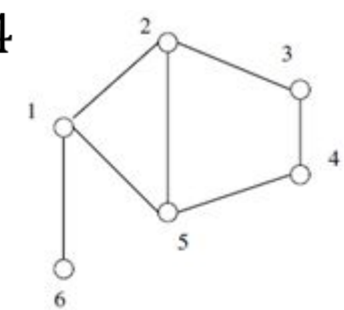
Teoria da Complexidade

Reduções

NP-completude

Conceitos básicos

- Problema
  - Uma pergunta a ser respondida por uma solução algorítmica.
    - Ex: encontrar um caminho de mínimo do vértice u ao vértice v em um grafo sem peso G.
- Instância
  - Uma instância de um problema é um conjunto de parâmetros de entrada específico para um dado problema
    - Ex: Seja o grafo G da figura ao lado, $u = 1\ e\ v = 4$
      - Solução para esta instância = $\langle 1,5,4 \rangle$



Redução

- Sejam os seguintes problemas
  - Problema A
    - Instância de entrada I_A
    - Solução S_A
  - Problema B
    - Instância de entrada I_B
    - Solução S_B
- Gostaríamos de resolver o problema A e sabemos como resolver o problema B (algoritmo π_B)
- Redução de um problema A para um problema B
  - T_I: transformação de uma instância I_A em uma instância I_B de B
  - T_S: transformação da solução S_B de B para I_B em uma solução S_A de A para I_A

Redução

$$
\begin{tikzcd}
\mathbf{A} & & \mathbf{B} \\
I_A \arrow[rr, "T_I"] & & I_B \arrow[d, "\pi_B"] \\
S_A & & S_B \arrow[ll, "T_S"']
\end{tikzcd}
$$

- Definição
  - Um problema A é redutível ao problema B em tempo $f(n)$ se existe a redução $A \propto_{f(n)} B$, onde $n = |I_A|$ e T_I e T_S são executáveis em $O(f(n))$

Exemplo de Redução

- Problema: Longest Increasing Subsequence (LIS)
  - Entrada: Uma sequência de inteiros ou caracteres $S = \langle s_1, \dots, s_n \rangle$ de tamanho n
  - Saída: Qual a subsequência crescente mais longa $\langle s_{p_1}, \dots, s_{p_k} \rangle$ tal que $p_i < p_{i+1}$ e $s_{p_i} < s_{p_{i+1}}$, sendo $1 \leq i < k \leq n$?

- Problema: Longest Common Subsequence (LCS)
  - Entrada: Sequências de inteiro/caracteres $S = \langle s_1, \dots, s_n \rangle$ e $T = \langle t_1, \dots, t_m \rangle$.
  - Saída: Qual a maior subsequência comum em S e T?

LIS $\propto$ LCS

- Como utilizar uma solução para LCS para resolver o problema LIS?
- Ideia:
  - Obter um S' a partir da ordenação de S
    - $S = \{-7, 10, 9, 2, 3, 8, 8, 1\}$
    - $S' = \{-7, 1,2,3,8,8,9,10\}$
  - Criar um conjunto T com elementos distintos de S'
    - $T = \{-7, 1,2,3,8,9,10\}$
  - Resolver $\text{LCS}(S, T)$
    - $S = \{-7, 10, 9, 2, 3, 8, 8, 1\}$
    - $T = \{-7, 1,2,3,8,9,10\}$
- Uma solução: O LCS é -7, 2,3,8, que é um LIS de S.

Corretude da redução

- Proposição: Se S tem uma subsequência crescente de tamanho k, então existe uma subsequência comum entre S e T de tamanho k.
- Prova:
  - Seja uma subsequência crescente $\langle s_{p_1}, \dots, s_{p_k} \rangle$ de S, tal que $s_{p_i} < s_{p_{i+1}}$ e $p_i < p_{i+1}$ para cada $1 \leq i < k \leq n$.
  - Seja T obtido a partir da redução.
  - Como os elementos de T são elementos distintos de S ordenados, então $\langle s_{p_1}, \dots, s_{p_k} \rangle$ representa uma subsequência comum entre S e T.
    - $\langle s_{p_1}, \dots, s_{p_k} \rangle$, por definição, é uma subsequência de S.
    - $\langle s_{p_1}, \dots, s_{p_k} \rangle$ também é subsequência de T, pois T contém todos os elementos distintos de S em ordem crescente!
- Proposição: Se existe uma subsequência comum entre S e T de tamanho k, então existe uma subsequência crescente de S de tamanho k.

Redução

```
LIS(S)
   S' = Ordena(S)
   T = elementosDistintos(S')
   Retorna(LCS(S,T));
```

- Tempo de execução
  - Ordenação em tempo $O(n \lg n)$
  - $|T| = m \leq n$
    - Geração de T em $O(n)$
  - LCS(S,T) feito em $O(nm) = O(n^2)$
  - Logo, tempo total é $O(n^2)$ para resolver o LIS através da redução, sendo $O(n \lg n)$ para as transformações.

Redução

$$
\begin{tikzcd}
\mathbf{A} & & \mathbf{B} \\
I_A \arrow[rr, "T_I"] & & I_B \arrow[d, "\pi_B"] \\
S_A & & S_B \arrow[ll, "T_S"']
\end{tikzcd}
$$

- Definição
  - Um problema A é redutível ao problema B em tempo $f(n)$ se existe a redução $A \propto_{f(n)} B$, onde $n = |I_A|$ e T_I e T_S são executáveis em $O(f(n))$

Reduções

- Existindo
  - $A \propto_{f(n)} B$
  - Algoritmo π_B para resolver B
  - Então, temos um algoritmo π_A que resolve A
    - $\pi_A = T_I \cdot \pi_B \cdot T_S$
- Se π_B tem complexidade $g(n)$, então o problema A pode ser resolvido em $O(f(n) + g(n))$
- Se o limite inferior do problema A é $\Omega(h(n))$, então $\Omega(h(n) - f(n))$ é também um limite inferior de B.
  - Se houvesse uma forma mais rápida de resolver B, então também violaria o limite inferior de A resolvendo através desta redução
  - Logo, se $f(n) = o(h(n))$, então $\Omega(h(n))$ é um limite inferior de B.

Redução

- Quando usar redução?

  1) Quero encontrar um algoritmo para A e conheço um algoritmo para B. Determino uma cota superior para A.

  2) Quero encontrar uma cota inferior para B e encontro uma cota inferior baseada na cota inferior para A.

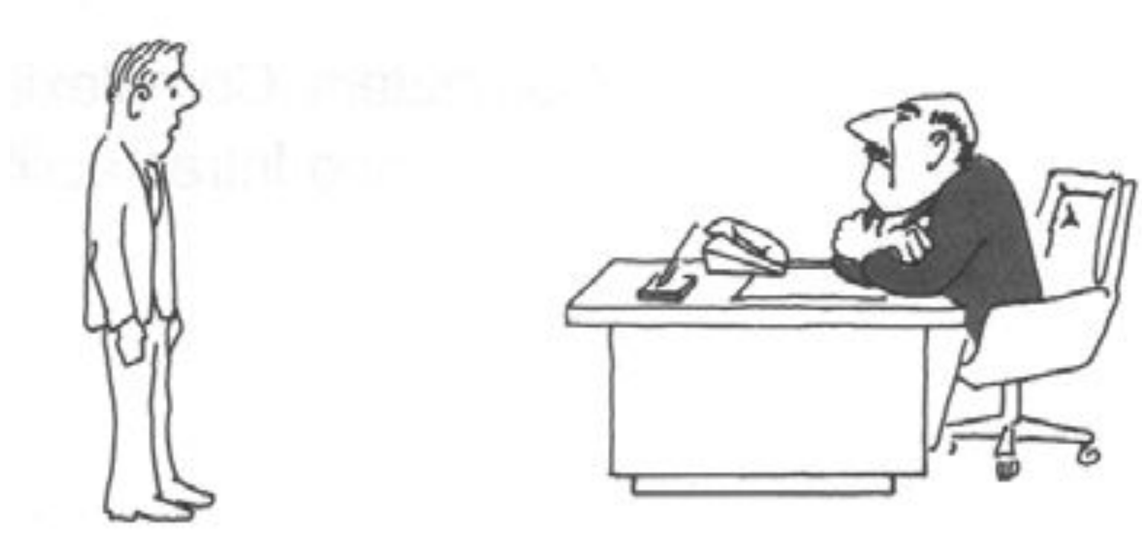
Introdução

- Muitos problemas não possuem solução eficiente conhecida
  - Problema do Caixeiro Viajante (TSP)
- Alguns problemas na vida real também são difíceis
  - Quais propriedades o tornam difíceis?
  - Conhecimento de problemas classificados como difíceis
  - Reduções

- Suponhamos que, mesmo após várias semanas de estudo e tentativas, você não consiga encontrar um algoritmo rápido para resolver uma tarefa atribuída pelo seu chefe. O que você poderia reportar ao seu chefe?

Resposta 1

- Eu acho que sou muito burro ...
  - Resposta perigosa!



"I can't find an efficient algorithm, I guess I'm just too dumb."

Resposta 2

- Não existe um algoritmo rápido!
  - Exige a demonstração de um limite inferior ao problema!

"I can't find an efficient algorithm, because no such algorithm is possible!"

Resposta 3

- Eu não consigo encontrar um algoritmo eficiente, mas nenhuma outra pessoa no mundo conseguiria.

"I can't find an efficient algorithm, but neither can all these famous people."

NP-Completude

- Demonstrar que o seu problema é “tão difícil” quanto um conjunto de problemas reconhecidamente difíceis, para os quais nenhuma solução eficiente é conhecida, indica que tentar encontrar uma solução eficiente para o seu problema pode não ser o melhor caminho
  - Projetar por um algoritmo de aproximação?

Teoria de NP-completude

- Vários problemas para os quais soluções eficiente (polinomial) não eram conhecidos, mas não se conseguia provar um limite inferior de tempo exponencial
- No início da década de 1970, S. Cook e R. Karp forneceram meios para mostrar que vários desses problemas considerados difíceis se tratavam do mesmo problema!

Problemas de decisão

- Problemas diferem no seu tipo de resposta
  - Ex: Problema do Caixeiro Viajante (TSP) retorna uma permutação de vértices
  - Outros problemas podem retornar números, etc.
- Problemas de decisão retornam **Verdadeiro** ou **Falso**.
  - Reduzir um problema de decisão a outro problema de decisão é conveniente, pois as respostas são do mesmo tipo

Problemas de decisão

- Os problemas interessantes de otimização podem ser convertidos em problemas de decisão
- Ex:
  - Problema: Problema de Decisão do Caixeiro Viajante
  - Entrada: Um grafo com pesos G e um inteiro k
  - Saída: Existe um tour TSP com custo ≤ k?
- Se existisse uma solução rápida ao Problema de decisão do Caixeiro Viajante, poderíamos fazer uma busca binária para diferentes valores de k.

Classes P e NP

- Classe P
  - Classe de problemas com soluções **polinomiais** conhecidas. P=*polynomial-time*.
  - Ex: caminho mínimo, árvore geradora mínima, etc.
- Classe NP
  - Classe de problemas que podem ser **verificados** em tempo polinomial. Soluções aos problemas em NP **não são necessariamente polinomiais** (NP=*nondeterministic polynomial-time*), mas problemas de P estão dentro de NP, pois se conseguimos resolver o problema em tempo polinomial, podemos verificar uma solução em tempo polinomial também: podemos resolver o problema e verificar se essa solução é boa o suficiente.
  - Ex: SAT, TSP, vertex cover, etc.

Verificação

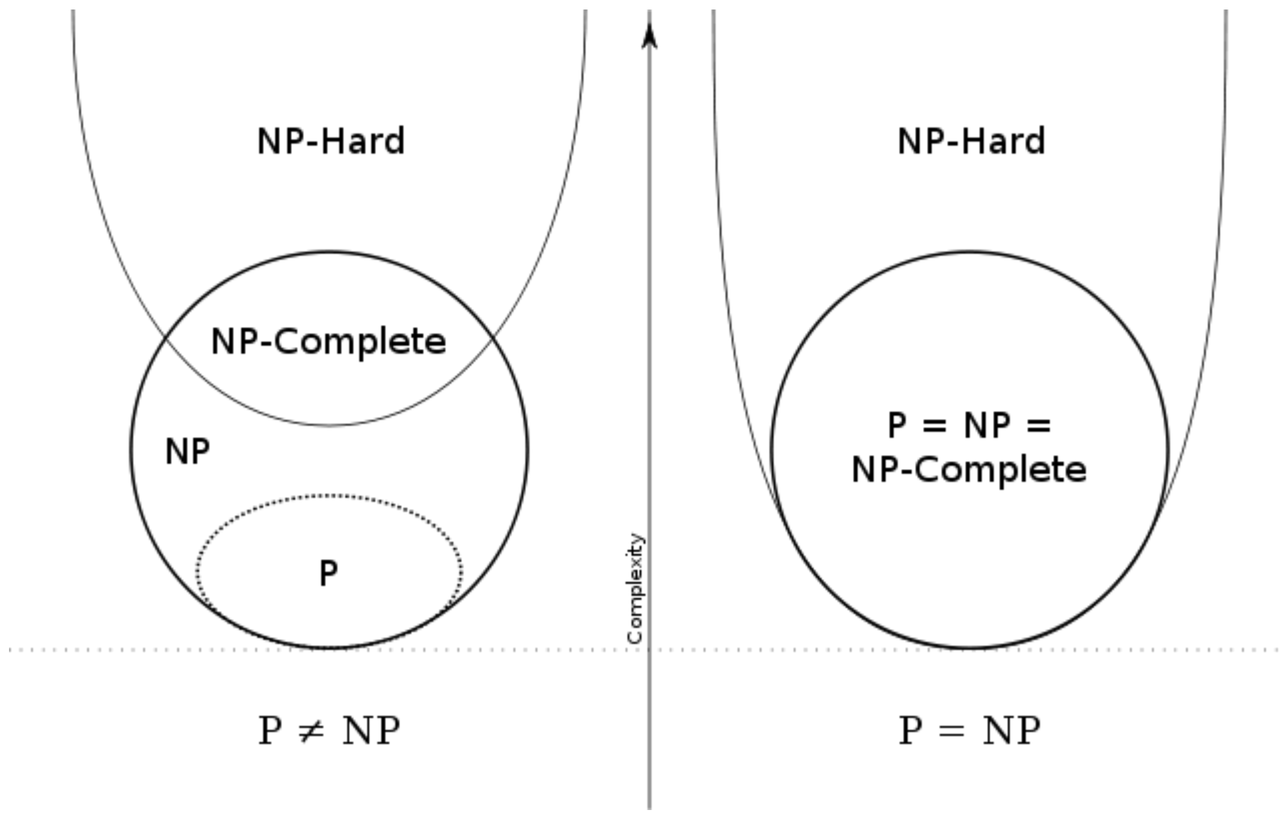
- Verificação é uma tarefa realmente mais fácil do que a descoberta ou busca da solução?
- Para problemas de decisão NP-completo:
  - SAT: Podemos verificar que um conjunto de atribuições V/F representa uma solução a uma instância do problema SAT?
    - Sim, basta verificar se cada cláusula possui pelo menos 1 literal verdadeiro.
  - TSP: Podemos verificar se um dado grafo tem um tour TSP de peso no máximo k dada a ordem de vértices do tour?
    - Sim, basta somarmos os pesos das arestas entre os vértices do tour e mostrar que é no máximo k.

P vs NP

- Um dos problemas abertos mais intrigantes da computação é se “P=NP?”
  - Existem problemas em NP que não são membros de P?
    - Se não existir, então P=NP.
    - Se existir pelo menos um, então P≠NP. É preciso provar que “não posso encontrar um algoritmo rápido o suficiente” para esse problema.

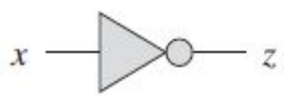
NP-hard vs NP-completo

- NP-hard (NP-difícil): um problema é NP-hard se é tão difícil quanto qualquer problema em NP.
  - Ex: SAT
- NP-completo: um problema é NP-completo se ele é NP-hard e também está em NP.
  - Em geral, problemas NP-hard também são NP-completo
- Existem alguns problemas que parecem ser NP-hard, mas não NP-completo. Ou seja, podem ser mais difíceis que NP-completo.
  - Ex: Xadrez. Para verificar checkmate após a primeira jogada, seria necessário criar uma árvore de possíveis jogadas com número exponencial de nós. Não pode ser verificado em tempo polinomial, portanto o problema não está em NP.

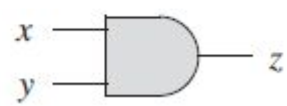

NP-Hard
NP-Complete
NP
P
Complexity
P ≠ NP
NP-Hard
P = NP =
NP-Complete
P = NP

Circuitos Lógicos Combinacionais

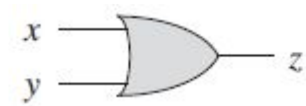
- Portas lógicas



0	1
1	0



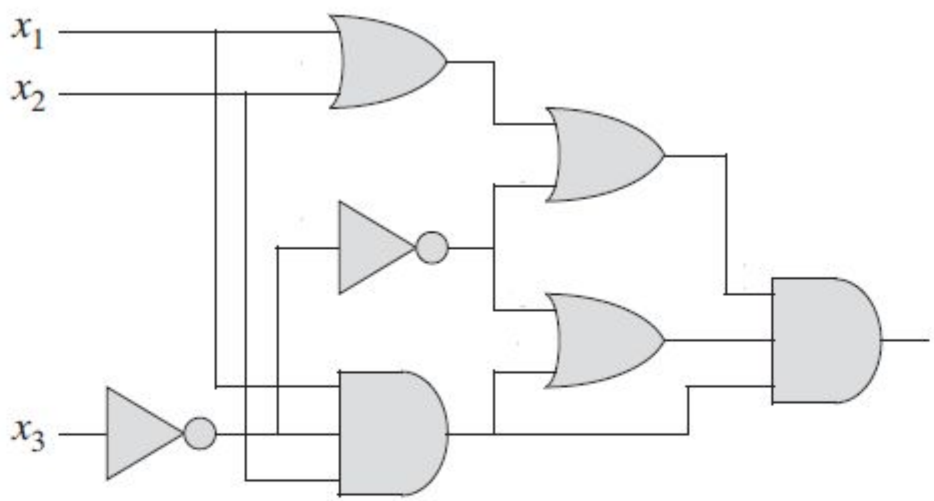
0	0	0
0	1	0
1	0	0
1	1	1



0	0	0
0	1	1
1	0	1
1	1	1

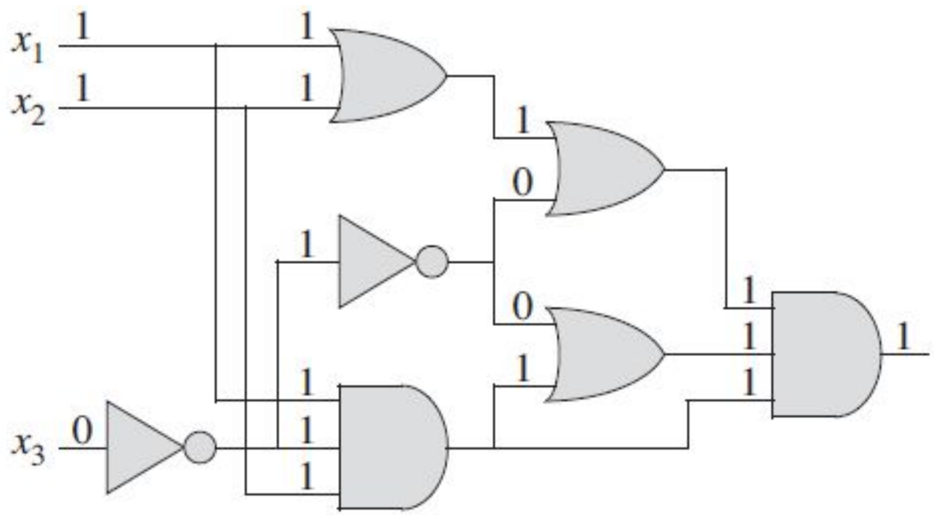
Satisfazendo circuitos lógicos

- Existe um conjunto de valores de entrada (x_1, x_2, x_3) que resulte em uma saída 1?



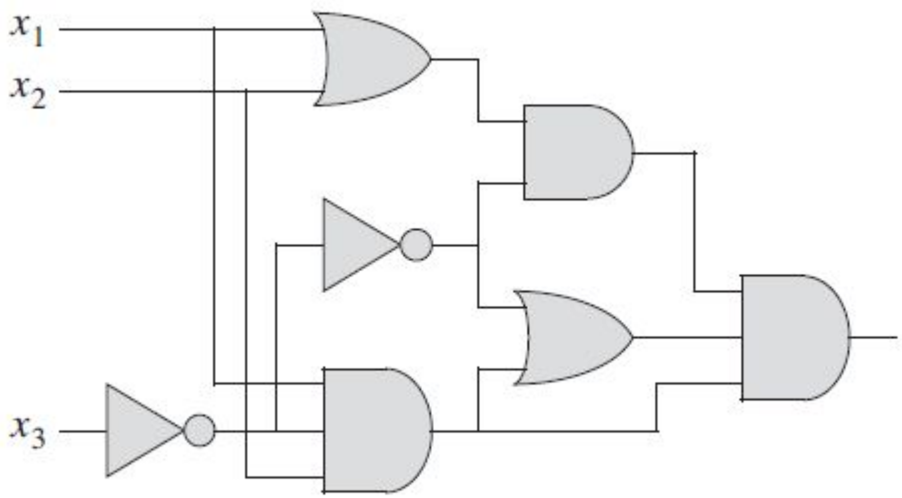
Satisfazendo circuitos lógicos

- Existe um conjunto de valores de entrada (x_1, x_2, x_3) que resulte em uma saída 1?



Satisfazendo circuitos lógicos

- Existe um conjunto de valores de entrada (x_1, x_2, x_3) que resulte em uma saída 1?



Definição: um circuito pode ser **satisfeito** se existe um conjunto de atribuições das variáveis de entrada que resultam em uma saída 1.

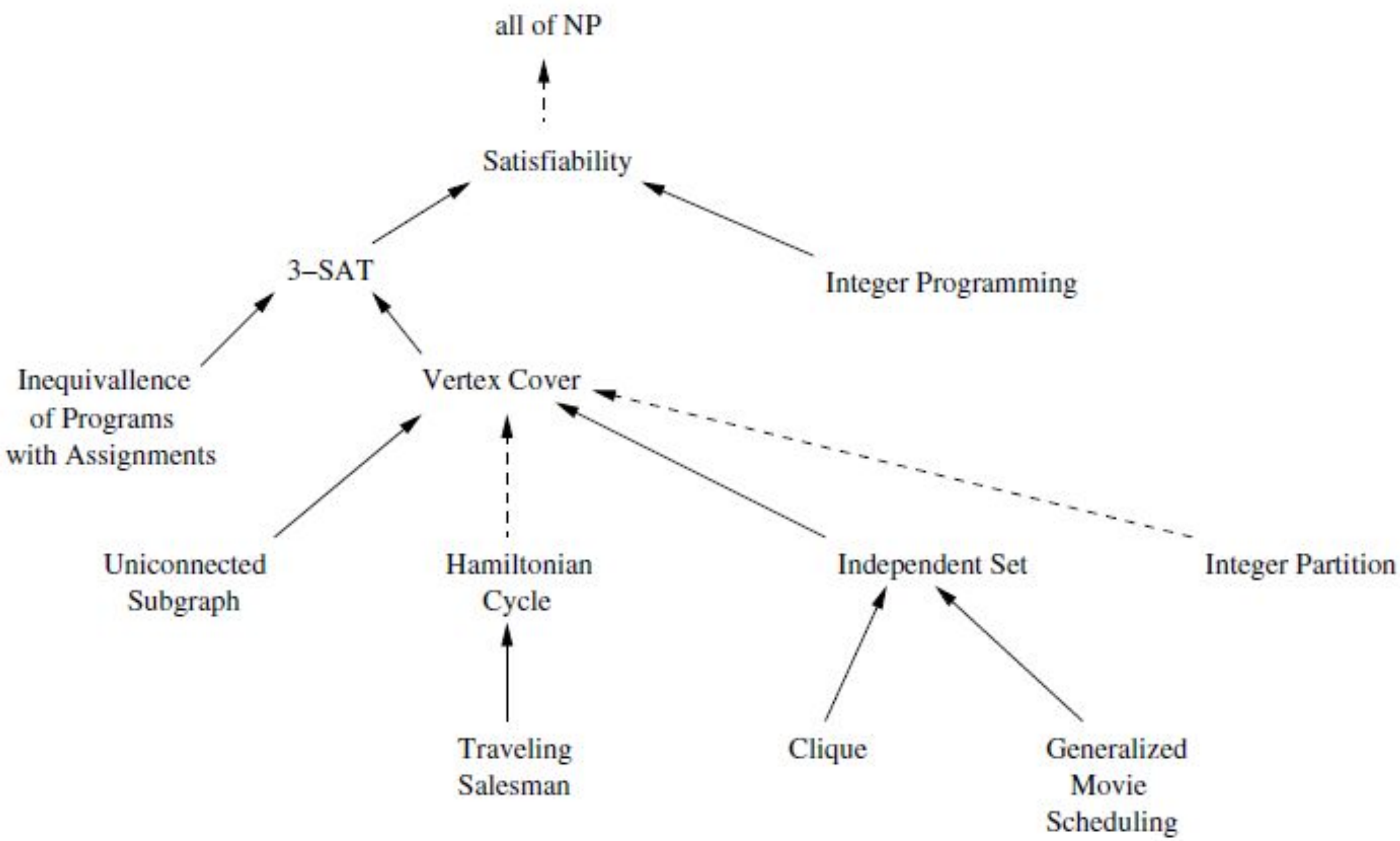
Circuit Satisfiability

- Problema CircuitSAT: dado um circuito de portas AND, OR e NOT, determinar se ele pode ser satisfeito

- Este problema é NP-completo
  - CircuitSAT está em NP, pois pode de ser verificado em tempo polinomial.
  - CircuitSAT é NP-hard
    - Qualquer problema em NP podem ser reduzido a CircuitSAT em tempo polinomial

Provando NP-completude

- Para mostrar que um problema B é NP-completo
  - Mostrar que B está em NP
  - Selecionar um problema NP-completo conhecido A
  - Construir uma transformação (redução) de A para B
  - Provar que a redução é polinomial



Árvore de reduções para alguns problemas NP-completo.

Satisfiability (SAT)

- Problema reconhecidamente difícil, NP-completo
  - Problema: Seja um conjunto de n variáveis booleanas $x_1, x_2, \ldots, x_n$ e um conjunto de cláusulas com conectivos booleanos tais como $\wedge$(AND), $\vee$(OR), $\neg$ (NOT), $\rightarrow$ (implicação), ($\leftrightarrow$) se e somente se e parênteses, determinar se a fórmula pode ser satisfeita por algum conjunto de atribuições de valores para as variáveis.
  - Entrada: atribuições {0,1} para as n variáveis
  - Saída: Existe uma atribuição de valores booleanos de forma que a resposta seja verdadeira (1)?

Exemplo

- $\emptyset = ((x_1 \rightarrow x_2) \vee \neg\big((\neg x_1 \leftrightarrow x_3) \vee x_4\big)) \wedge \neg x_2$

- Para $x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 1$, então
- $\emptyset = \Big((0 \rightarrow 0) \vee \neg\big((\neg 0 \leftrightarrow 1) \vee 1\big)\Big) \wedge \neg 0$

$$= \big(1 \vee \neg(1 \vee 1)\big) \wedge 1$$
$$= (1 \vee 0) \wedge 1$$
$$= 1$$

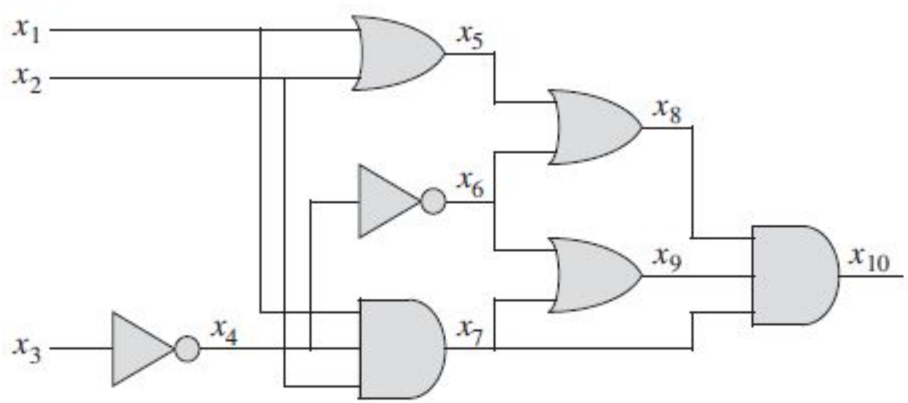
Solução para SAT

- Força bruta
  - Testar todas as possíveis 2^n atribuições às n variáveis
- Provando NP-completude
  - Mostrar que o problema está em NP
  - Mostrar que é NP-hard
    - Considerando o problema CircuitSAT como um problema NP-completo conhecido torna essa tarefa mais fácil do que mostrar reduções polinomiais de todos problemas em NP a SAT
    - Podemos mostrar a redução polinomial CircuitSAT $\propto_{\mathrm{P}}$ SAT

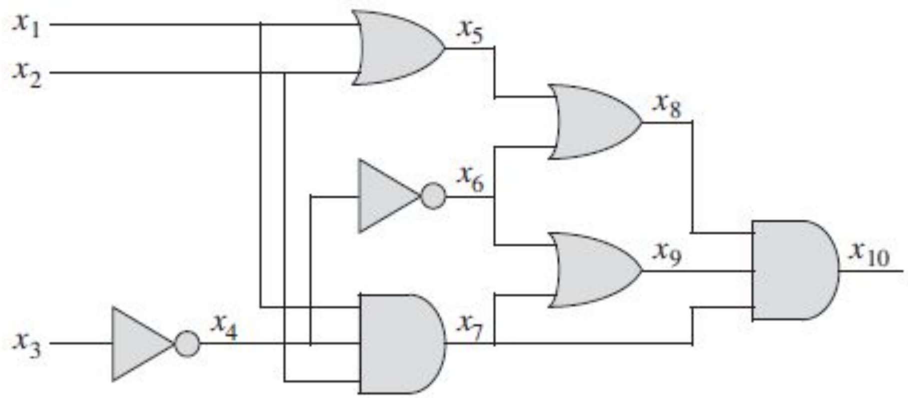
CircuitSAT $\propto_P$ SAT

- Podemos representar as portas lógicas como fórmulas e trocar as entradas das portas lógicas conectadas a saídas de outras pelas fórmulas
  - O tamanho da fórmula pode crescer exponencialmente

- Podemos representar a última porta como uma cláusula da forma
- $x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9)$
- Então, podemos representar o circuito acima através da fórmula



$$
\begin{aligned}
\emptyset = x_{10} &\wedge (x_4 \leftrightarrow \neg x_3) \\
&\wedge (x_5 \leftrightarrow (x_1 \vee x_2)) \\
&\wedge (x_6 \leftrightarrow \neg x_4) \\
&\wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)) \\
&\wedge (x_8 \leftrightarrow (x_5 \vee x_6)) \\
&\wedge (x_9 \leftrightarrow (x_6 \vee x_7)) \\
&\wedge (x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9))
\end{aligned}
$$



$$
\begin{aligned}
\emptyset = x_{10} &\wedge (x_4 \leftrightarrow \neg x_3) \\
&\wedge (x_5 \leftrightarrow (x_1 \vee x_2)) \\
&\wedge (x_6 \leftrightarrow \neg x_4) \\
&\wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)) \\
&\wedge (x_8 \leftrightarrow (x_5 \vee x_6)) \\
&\wedge (x_9 \leftrightarrow (x_6 \vee x_7)) \\
&\wedge (x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9))
\end{aligned}
$$

- Se o circuito tem um conjunto de atribuições que o satisfaz, então esses valores produzem 1 na última variável. Portanto, se utilizarmos esses valores em $\emptyset$, cada cláusula terá valor 1. Desta forma, se o circuito pode ser satisfeito, então a fórmula também.
- Se alguma atribuição de valores leva $\emptyset$ a um valor 1, o circuito também pode ser satisfeito usando um argumento análogo.

- Esta redução pode ser feita facilmente em tempo polinomial.

- Portanto, mostramos que CircuitSAT $\propto_{\mathrm{P}}$ SAT.

Satisfiability (SAT)

- Versão restrita com operadores AND, OR e NOT (FNC: Forma Normal Conjuntiva)
  - Problema: Satisfiability (SAT)
  - Entrada: Um conjunto de variáveis booleanas V e um conjunto de cláusulas C sobre V
    - Cada cláusula é formada por disjunção (OR) de um conjunto de literais (ex: v_1 ou $\bar{v}_1$)
    - A fórmula é formada pela conjunção (AND) de cláusulas
  - Saída: Existe uma atribuição de valores booleanos para C de forma que cada cláusula contém pelo menos um literal verdadeiro?

SAT

- Ex 1: $C = \{\{v_1, \bar{v}_2\}, \{\bar{v}_1, v_2\}\}, V = \{v_1, v_2\}$
  - Solução: $v_1 = v_2 = \text{True}$ ou $v_1 = v_2 = \text{False}$
- Ex 2: $C = \{\{v_1, v_2\}, \{v_1, \bar{v}_2\}, \{\bar{v}_1\}\}$
  - Para satisfazer a terceira cláusula, v_1 deve ser falso. Logo, para satisfazer a segunda cláusula, v_2 deve ser falso também, tornando a primeira cláusula não satisfeita!
- Não existe algoritmo conhecido com pior caso polinomial que resolva o problema SAT

3-SAT

- O problema SAT é difícil, porém alguns casos especiais de SAT não são.
  - Se cada cláusula possui apenas um literal, basta atribuir um valor àquela variável para satisfazer cada cláusula.
- A partir de 3 literais por cláusula o problema se torna difícil
- Problema: 3-Satisfiability (3-SAT)
  - Entrada: Um conjunto de variáveis booleanas V e um conjunto de cláusulas C sobre V onde cada cláusula possui **exatamente 3 literais**
  - Saída: Existe uma atribuição de valores booleanos para C de forma que cada cláusula é satisfeita?

3-SAT

- Redução de SAT para 3-SAT
  - Transformar cada cláusula transformando-a em cláusulas de 3 literais
- Para cláusula C_i com k literais:
  - $k = 1, C_i = \{z_1\}$: criamos 2 novas variáveis v_1, v_2 e 4 novas cláusulas: $\{\{v_1, v_2, z_1\}, \{v_1, \bar{v}_2, z_1\}, \{\bar{v}_1, v_2, z_1\}, \{\bar{v}_1, \bar{v}_2, z_1\}\}$
  - $k = 2, C_i = \{z_1, z_2\}$: criamos 1 nova variável v_1 e 2 novas cláusulas: $\{\{v_1, z_1, z_2\}, \{\bar{v}_1, z_1, z_2\}\}$
  - $k = 3, C_i = \{z_1, z_2, z_3\}$: apenas copiamos C_i inalterado
  - $k > 3, C_i = \{z_1, z_2, \ldots, z_n\}$: criamos $n-3$ novas variáveis e $n-2$ novas cláusulas em uma cadeia: para $2 \leq j \leq n-3$, $C_{i,j} = \left\{v_{i,j-1}, z_{j+1}, \bar{v}_{i,j}\right\}, C_{i,1} = \left\{z_1, z_2, \bar{v}_{i,1}\right\}$ e $C_{i,n-2} = \left\{v_{i,n-3}, z_{n-1}, z_n\right\}$.
    - Para satisfazer $C_{i,1}$ se todas literais de C_i forem falsas, implica em $v_{i,1} = \text{False}$, o que implica $v_{i,2} = \text{False}$ e assim por diante. Porém, em $C_{i,n-2}$ não haveria uma nova variável adicional para satisfazer a cláusula.

- Para n cláusulas e total de m literais, esta transformação pode ser feita em $O(m + n)$.
- Cada solução de SAT também satisfaz a instância de 3-SAT e cada solução de 3-SAT descreve como podemos atribuir valores às variáveis para obter uma solução a SAT. Portanto, o problema transformado é equivalente ao original.

Clique

- Um clique em um grafo não-direcionado $G = (V, E)$ é um subconjunto $V' \subseteq V$ de vértices em que cada par de vértices está conectado por uma aresta em E.
  - O problema do clique consiste em determinar o clique de maior tamanho em G
- Vamos considerar o problema de decisão para um tamanho k
  - Problema CLIQUE: Dada um grafo G, determinar se existe um clique de tamanho k.

Clique

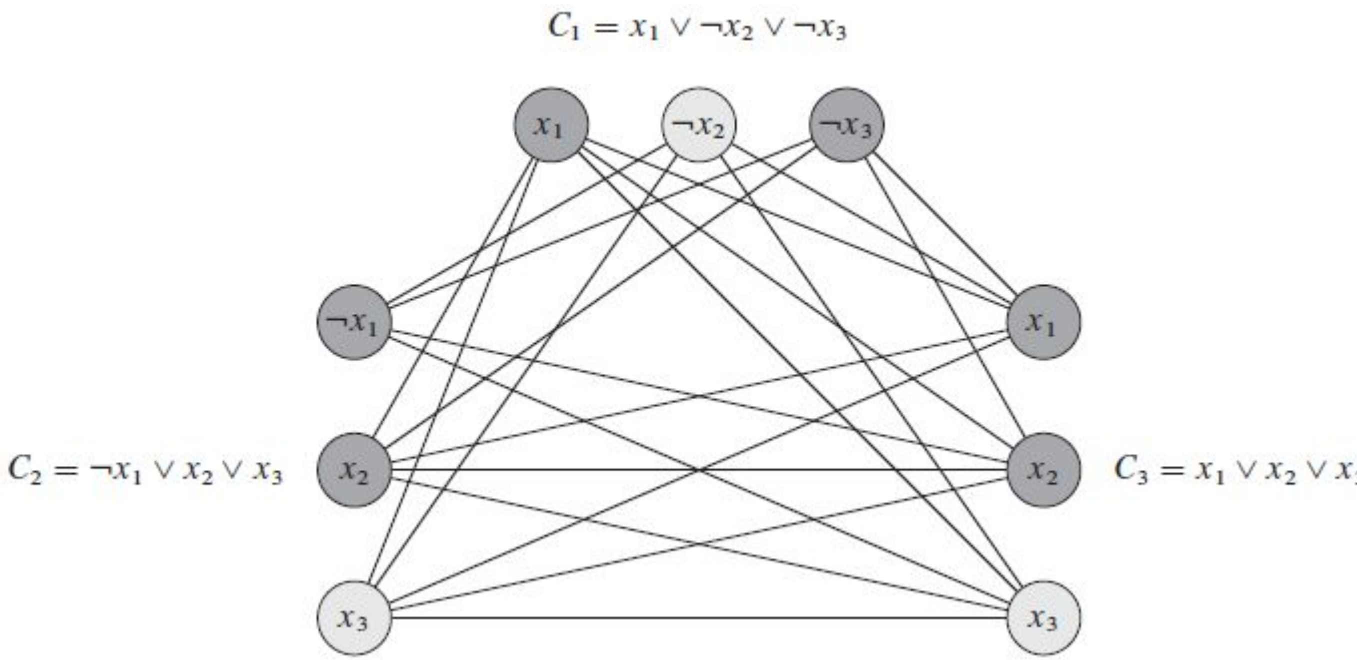
- O problema de decisão do clique é NP-completo
  - CLIQUE $\in$ NP, pois podemos verificar se um dado V' é subconjunto de V e se para cada par $u, v \in V'$ a aresta $(u, v) \in E$.
  - Mostrar que 3SAT $\propto_{\mathrm{P}}$ CLIQUE

Redução

- Seja $\emptyset = C_1 \wedge C_2 \wedge \cdots \wedge C_k$ a fórmula de 3-SAT com k cláusulas, sendo que cada cláusula tem exatamente 3 literais
- Construímos o grafo G da seguinte forma
  - Para cada cláusula, inserimos os 3 literais como vértices em V.
  - Inserimos uma aresta entre 2 vértices se essas literais estão em cláusulas diferentes e se uma não é a negação da outra
- Esta redução pode ser feita em tempo polinomial

Exemplo

- Para $C = \{\{x_1, \neg x_2, \neg x_3\}, \{\neg x_1, x_2, x_3\}, \{x_1, x_2, x_3\}\}$



- Se C pode ser satisfeita, então existe um clique em G de tamanho k.
  - Supondo que a fórmula C tem uma atribuição que a satisfaz, então cada uma das cláusulas possui pelo menos 1 literal de valor 1. Cada um desses literais é um vértice do grafo. Se pegarmos um literal de valor 1 de cada cláusula, teremos um subconjunto V' de k vértices. Para cada par de vértices de V', os seus literais têm valor 1, portanto um não pode ser a negação do outro, então temos um clique de tamanho k no grafo G com os vértices V'.

- Se G tem um clique de tamanho k, então C pode ser satisfeita.
  - Supondo que G tenha um clique V' de tamanho k. Nenhuma aresta em G conecta vértices da mesma cláusula, portanto V' contém exatamente 1 vértice de cada cláusula. Como o grafo não possui arestas ligando uma variável e sua negação, então podemos atribuir 1 para cada literal de V'.

Exercícios

1. Para cada questões abaixo, diga se a afirmação é verdadeira, falsa, verdadeira se P≠NP ou falsa se P≠NP. Justifique as suas respostas.

a) Existem problemas em NP que não estão em P.
b) Existem problemas em P que estão em NP-completo.
c) Não há problemas em P que estão em NP-completo.
d) Se A pode ser polinomialmente reduzido a B e B é NP-completo, então A é NP-completo.
e) Se A pode ser polinomialmente reduzido a B e B pertence a P, então A pertence a P.

Exercícios

2) Seja o problema Subset-Sum o problema de decidir se, dado como entrada um conjunto S de números inteiros, existe algum subconjunto X de S, tal que a soma de seus elementos seja igual a um número-alvo t. Prove que Subset-Sum é NP-completo. Dica: Tente reduzir 3SAT para este problema.

Exercícios

3) Seja o problema de decisão da Mochila o problema de decidir se, dados os inteiros positivos C e V existe um subconjunto $S' \subseteq S$ tal que a soma dos pesos dos elementos de S' seja menor ou igual a C e a soma dos valores dos elementos de S' seja maior ou igual a V. Mostre que o problema de decisão da Mochila é NP-completo. Dica: Tente reduzir de Subset-sum.

Referências

- CLRS, Introduction to Algorithms, 3rd ed.
  - Cap. 34, 34.3, 34.4, 34.5.1
- Skiena, The Algorithm Design Manual, 2nd ed.
  - Cap 9, 9.1, 9.2, 9.4, 9.9