UNIVERSIDADE FEDERAL DE VIÇOSA
DEPARTAMENTO DE INFORMÁTICA

Classificação de Problemas NP-Completude

INF 332

Prof. José Elias C. Arroyo

Outline

- Introdução
- Problemas P
- Algoritmos Não-Determinísticos
- Problemas NP
- Problemas NP-Completo
- Problemas NP-Difícil

Introdução

Problemas Tratáveis:

Problemas que podem ser resolvidos por algoritmos de tempo polinomial.

Problemas Intratáveis:

Problemas para os quais **não se conhece** um **algoritmo de tempo polinomial** que o resolva.

Os melhores algoritmos conhecidos para resolver estes problemas são de tempo exponencial.

Introdução

Problemas tratáveis

- Ordenação,
- Caminho mínimo,
- Árvore geradora mínima,
- Designação,
- Programação Linear, etc.

Problemas intratáveis

- Torres de Hanói
- Caixeiro viajante
- Mochila 0/1
- Cobertura mínima de vértices
- Clique máxima
- SAT, etc.
- Programação Linear Inteira

Introdução

- A teoria de complexidade a ser apresentada não mostra como obter algoritmos de tempo polinomial para problemas que demandam tempo exponencial, nem afirmam que não existem.

- É possível mostrar que os problemas intratáveis (para os quais não se conhece a existência de algoritmo de tempo polinomial) são computacionalmente relacionados.

Problemas da Classe P

- Um problema que pode ser resolvido em **tempo polinomial**, informalmente, podemos pensar como um problema da **classe *P***.

- Uma definição mais formal desta classe de problemas inclui **problemas de decisão**.

Problemas da Classe P

Problemas de Decisão (“Sem ou Não”):

- São problemas que questionam a existência de uma solução.
- **Existe uma solução para o problema?**
- A resposta (saída) para estes problemas pode ser SIM ou NÃO.

***Exemplo*:**

Dado um grafo valorado e um número k.

Deseja-se saber se existe ou não um *Ciclo Hamiltoniano* com custo $\leq k$.

Problemas da Classe P

Problemas de Otimização:

Estes problemas consistem em **encontrar a melhor solução** do problema (uma solução que minimize ou maximize uma função objetivo).

Problemas da Classe P

Problemas de Otimização:

Estes problemas consistem em **encontrar a melhor solução** do problema (uma solução que minimize ou maximize uma função objetivo).

Muitos problemas possuem versões de **decisão** e **otimização**.

Exemplo: PROBLEMA do CAMINHO MÍNIMO.
Dado um grafo e dois de seus vértices s e t.

Otimização: Encontre o caminho de menor custo de s até t.

Decisão: Existe um caminho de s até t de custo $\leq k$?

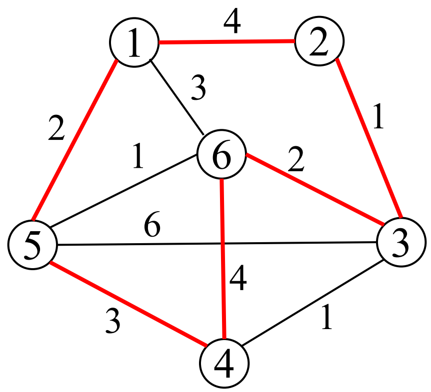
Problemas da Classe P

CICLO HAMILTONIANO

•***Problema de Decisão***:

Dado um grafo ponderado completo e um número *k*.

Existe um ciclo Hamiltoniano de custo $\leq k$?



$4 + 1 + 2 + 4 + 3 + 2 = 16$

$k = 17$: SIM
$k = 10$: NÃO

•***Problema de Otimização***: Encontre o ciclo Hamiltoniano de custo mínimo.

Problemas da Classe P

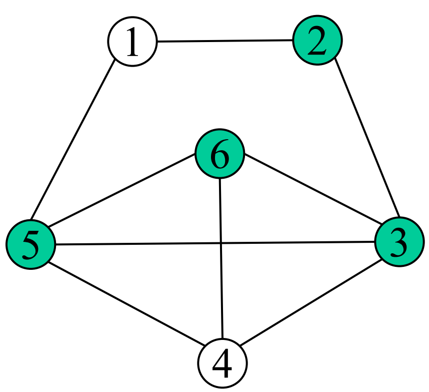
COBERTURA DE VÉRTICES

Dado um grafo $G = (V, A)$ com n vértices.

Uma *cobertura de vértices* de G é um **subconjunto** W **de vértices** tal que toda aresta de G tenha pelo menos uma extremidade em W ($|W|$ = tamanho da cobertura).

***Problema de Decisão*:**

Para o grafo abaixo, existe uma cobertura de vértices de tamanho $\leq k$, $(1 \leq k \leq n)$?

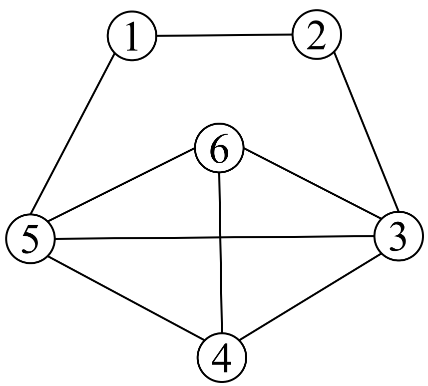


Se $k = 4$: SIM

Se $k = 3$: NÃO

Problemas da Classe P

Problema de Otimização **(Problema da Cobertura Mínima de Vértices)**: Determinar a menor cobertura de vértices do grafo.



Problemas da Classe P

- Ao resolver um problema de *Otimização* estaremos resolvendo um problema de *Decisão*, pois obtendo a solução ótima estamos respondendo se existe uma solução.

- Se um *Problema de Decisão* **não** puder ser resolvido em tempo polinomial, o *Problema de Otimização* associado, também **não** poderá ser resolvido em tempo polinomial.

Problemas da Classe P

- Para o estudo teórico de **intratabilidade de problemas (complexidade de problemas)** é conveniente considerar problemas de *Decisão* que são **problemas mais simples**.

- As classes mais importantes de problemas P, NP, e NP-Completo, são definidas para *Problemas de Decisão*.

Problemas da Classe P

- **Classe P**

  É formada por *Problemas de Decisão* que podem ser resolvidos por ***algoritmos determinísticos*** em tempo ***P****olinomial*: $O(P(n))$, onde $P(n)$ é um polinômio.

Problemas da Classe P

❑ Classe P

É formada por *Problemas de Decisão* que podem ser resolvidos por ***algoritmos determinísticos*** em tempo ***P****olinomial*: $O(P(n))$, onde $P(n)$ é um polinômio.

Algoritmo Determinístico:

É um algoritmo que, em termos informais, se comporta de forma previsível. Para uma **entrada específica**, **o algoritmo sempre produzirá a mesma saída** (saída única).

Problemas da Classe P

- **Exemplos de problemas da Classe P**

1. **Problema do ciclo Euleriano**:

   Dado um grafo conexo. Determinar se o grafo possui um ciclo Euleriano (um ciclo que passa por todas as arestas do grafo uma única vez).

2. **Problema do caminho mínimo**:

   Dado um grafo, dois de seus vértices *s* e *t*, e um número *k*. Determine se existe um caminho de *s* até *t* de custo $\leq k$.

3. **Problema da árvore geradora mínima**.

   Dado um grafo conexo valorado e um valor *k*. Determine se o grafo possui uma árvore geradora de custo $\leq k$.

Problemas da Classe P

- É natural questionar, **todo problema de decisão pode ser resolvido em tempo polinomial**?

  A resposta é **não**.

- Na verdade, **alguns problemas de decisão não podem ser resolvidos por nenhum algoritmo**.
- Esses problemas são chamados de indecidíveis.

Problemas da Classe P

- **Exemplo**: ***Problema de decisão do caixeiro viajante***:

  Dado um grafo valorado completo e um número k.
  Determine se existe um ciclo hamiltoniano de custo $\leq k$.

  Nenhum algoritmo de tempo polinomial foi proposto para resolver este problema, mas pode existir.

Algoritmos Não-Determinísticos

Algoritmo Não-Determinístico (ND):

- Usam operações cujo resultado não é unicamente definido.

- Estes algoritmos, a cada passo, são capazes de escolher "*arbitrariamente*" uma dentre as várias alternativas possíveis, produzindo **saídas diferentes**.

Algoritmos Não-Determinísticos

- Um ***algoritmo ND*** faz uso das seguintes operações:

- **Escolhe**:
  - Escolhe *arbitrariamente* um elemento para **construir** uma solução do problema.
  - A escolha é feita em **tempo constante** $O(1)$.

- **Verifica**:
  - Verifica se a solução construída é válida.
  - A verificação é feita usando um **algoritmo determinístico**.

Algoritmos Não-Determinísticos

- Um algoritmo ND resolve um Problema de Decisão (ou seja, retorna uma *resposta SIM ou NÃO*) se e somente se **em alguma execução** do algoritmo é **determinada uma solução válida**.

- Os Algoritmos ND são interessantes na solução de *problemas que possuem várias alternativas* a serem testadas.

- Um algoritmo é Não-determinístico Polinomial se o tempo da operação de **verificação** for **polinomial**.

Algoritmos Não-Determinísticos

Exemplos de Algoritmos ND

AlgoritmoND 1: //Busca de um elemento x em uma lista L

Entrada: $L[1..n]$ e x

Saída: SIM (achou o elemento x) ou NÃO (caso $x \notin$ L)

Início

$j \leftarrow$ **Escolhe**(L);

Se $L[j] == x$ então *//Verifica*

imprima(j);

retorne **SIM**;

Senão imprima(0); retorne **NÃO**;

Fim.

Tempo: $O(1)$

Algoritmos Não-Determinísticos

AlgoritmoND 2: //Ordenação **crescente** de n elementos de uma lista

Entrada: $L[1..\ n]$;

Saída: SIM (lista A ordenada) ou NÃO (falha).

Início

*//**Constrói** uma lista A com elementos de L:*

$ne = n$

Para $i \leftarrow 1$ até n faça:

$j \leftarrow$ **Escolhe**$(1..ne)$; *//escolhe índice ou posição*

$A[i] \leftarrow L[j]$; $A[j] \leftarrow L[ne]$; $ne = ne - 1$;

Para $i \leftarrow 1$ até $n-1$ faça: *//**Verifica***

Se $A[i] > A[i+1]$ então: **retorne NÃO**;

Imprima(A);

retorne SIM;

Fim.

Tempo: $O(n)$

Algoritmos Não-Determinísticos

AlgoritmoND 3:

//Dado um Grafo $G = (V, E)$. Deseja-se determinar uma
//cobertura de vértices W com $|W| \leq k$.

<u>*Entrada*</u>: $G = (V, E), k$.

<u>*Saída*</u>: SIM (W é cobertura) ou NÃO (W não é cobertura)

Início

$W \leftarrow \varnothing$;

$q \leftarrow$ **Escolhe**$(1, k)$; *//escolhe um valor entre 1 e k*

<u>Para</u> $i \leftarrow 1$ até q <u>faça:</u>

$v \leftarrow$ **Escolhe**(V); *//escolhe um vértice*

$W \leftarrow W \cup \{v\}$;

<u>Para</u> cada $(u, w) \in E$ <u>faça:</u> ***//Verifica***

<u>Se</u> $u \notin W$ e $w \notin W$ <u>então:</u>

Retorne NÃO;

Retorne SIM;

Fim.

Tempo : $O(k + |E|)$

AlgoritmoND 4: Problema da Mochila 0/1

//Determinar uma solução viável de custo $\geq R$.

Entrada: n, $c = (c_1, c_2, \ldots, c_n)$, R, $w = (w_1, w_2, \ldots, w_n)$, W.

Saída: SIM (achou solução viável) ou NÃO (não achou)

Início

Para $i \leftarrow 1$ até n faça

$x[i] \leftarrow$ **Escolhe**(0, 1);

//Verifica:

Se $\left(\sum_{i=1}^{n} w[i] * x[i] \leq W\right)$ e $\left(\sum_{i=1}^{n} c[i] * x[i] \geq R\right)$ então

Retorne SIM;

Senão **Retorne NÃO**;

Fim.

Tempo : $O(n)$

Algoritmos Não-Determinísticos

Problema da Satisfabilidade (SAT)

- ***Expressão booleana*** *(ou Fórmula Lógica)* ***na forma Normal Conjuntiva***:

$$E = (x \lor y \lor \neg z) \land \underbrace{(\neg x \lor y \lor z)}_{\text{Cláusula}} \land (\neg x \lor \neg y).$$

Onde: $\land$ e $\lor$ são os **operadores** **E** e **Ou**, respectivamente.

x, y, z, etc. são chamadas de **literais**. $\neg x$ é a negação de x.

Algoritmos Não-Determinísticos

Problema da Satisfabilidade (SAT)

- ***Expressão booleana*** *(ou Fórmula Lógica)* ***na forma Normal Conjuntiva***:

$$E = (x \vee y \vee \neg z) \wedge \underbrace{(\neg x \vee y \vee z)}_{\text{Cláusula}} \wedge (\neg x \vee \neg y).$$

Onde: $\wedge$ e $\vee$ são os **operadores** **E** e **Ou**, respectivamente.

x, y, z, etc. são chamadas de **literais**. $\neg x$ é a negação de x.

- **O problema da SAT** consiste em determinar se uma dada expressão booleana *E* é ou não *satisfatível*, ou seja, determinar um conjunto de valores (falso/verdadeiro ou 0/1) para os literais de modo que *E* seja *verdadeira*.

Algoritmos Não-Determinísticos

- **Exemplo:**

$E = (x \vee y \vee \neg z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee \neg y)$.

$E = 1$ (**satisfatível**) para $x = 1$, $y = 0$, $z = 1$.

- Qual é o tamanho do **espaço de busca** deste problema?

  $\Rightarrow$ 2^n, onde n é o número de literais da instância do problema (cada literal pode assumir valores 0 ou 1).

Algoritmos Não-Determinísticos

- **Exemplo:**

$E = (x \vee y \vee \neg z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee \neg y).$

$E = 1$ (**satisfatível**) para $x = 1$, $y = 0$, $z = 1$.

- Qual é o tamanho do **espaço de busca** deste problema?

  $\Rightarrow 2^n$, onde n é o número de literais da instância do problema (cada literal pode assumir valores 0 ou 1).

- **Definição:** k-SAT

  É um caso particular do **SAT**, no qual **cada cláusula** da expressão booleana tem no máximo *k* literais.

Algoritmos Não-Determinísticos

AlgoritmoND 5: SAT

Entrada: Expressão E com n literais $x_1, x_2, ..., x_n$

Saída: SIM (E é satisfatível) ou NÃO (E não é satisfatível)

Início

Para i ← 1 até n faça

$x_i \leftarrow$ **Escolhe**(0, 1); //Falso ou Verdadeiro

//Verifica:

Se $E(x_1, x_2, ..., x_n)$ é *Verdadeira* então retorne **SIM**;

Senão retorne **NÃO**;

Fim.

Tempo : $O(n)$

Algoritmos Não-Determinísticos

SUBSET-SUM (Soma de Subconjunto): Dados um conjunto de *n* números naturais $X = \{x_1, \ldots, x_n\}$ e um número *k*. Decidir se existe um subconjunto *S* de *X* tal que a soma dos elementos de *S* seja igual a *k*.

Escreva um algoritmo Não-Determinístico de tempo polinomial para o problema de decisão SUBSET-SUM.

Classe NP dos Problemas de Decisão

- **Classe NP**

  Formada por *problemas de decisão* que podem ser resolvidos por *algoritmos **N**ão-determinísticos* de *tempo **P**olinomial* (ou seja, classe de problemas de decisão cujas soluções construídas podem ser verificadas em tempo polinomial).

  Cuidado!: NP não é abreviatura de não polinomial!
  NP é abreviatura de ***N**ondeterministic **P**olynomial*.

Classe NP dos Problemas de Decisão

- Para verificar se um dado *Problema de Decisão* pertence à *classe NP* deve-se apresentar um algoritmo ND cuja **etapa de verificação** tem *tempo polinomial*.

Classe NP dos Problemas de Decisão

- Para verificar se um dado *Problema de Decisão* pertence à *classe NP* deve-se apresentar um algoritmo ND cuja **etapa de verificação** tem *tempo polinomial.*
- Os seguintes **problemas** (na versão de **decisão**) pertencem à classe **NP** :
  - Ordenação,
  - Caminho mínimo
  - Árvore geradora mínima,
  - Cobertura de vértices,
  - Caixeiro Viajante,
  - Mochila 0/1,
  - SAT, etc.

  Para estes problemas de decisão existem algoritmos NDs cuja verificação é feita em tempo polinomial.

Classe NP dos Problemas de Decisão

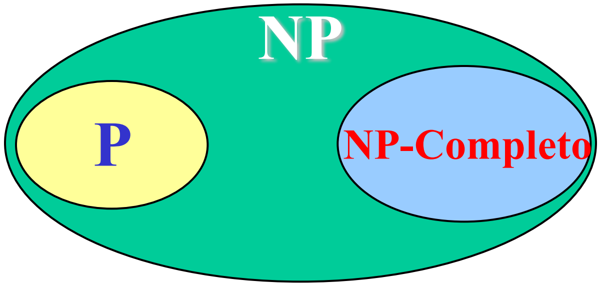
- Problema em aberto da Computação: $P = NP$?
- $P \subset NP$
  - Um *algoritmo determinístico* pode ser visto como um caso particular de um *algoritmo ND*.
  - Para qualquer problema da classe P existe um algoritmo ND de tempo polinomial.

- Grande pergunta: $NP \subset P$?

  Conjectura: existe uma forte suspeita de que isso NÃO SEJA VERDADE!!

  Isto é, $P \neq NP$.

Problemas NP-Completos



Classe NP-Completo

- É a classe dos problemas **mais difíceis** de NP (NP-Completo $\subseteq$ NP).
- Para "tentar" resolver P = NP, estudam-se os problemas NP-Completos.

" Se for encontrado um algoritmo de tempo polinomial para um *Problema NP-Completo*, então TODOS os problemas da classe NP podem ser resolvidos em tempo polinomial, ou seja, P = NP."

Como verificar que um problema pertence à classe NP-Completo?

Problemas NP-Completos

Redução de Problemas

Um problema $\mathcal{P}$ é ***redutível*** a um problema $\mathcal{Q}$ ($\mathcal{P} \propto \mathcal{Q}$) se:

- existe um algoritmo $\mathbf{f}_1$ que transforma instâncias de $\mathcal{P}$ em instâncias de $\mathcal{Q}$:

$$I_{\mathcal{Q}} \leftarrow \mathbf{f}_1(I_{\mathcal{P}});$$

- existe um algoritmo $\mathbf{A}_{\mathcal{Q}}$ que resolve $\mathcal{Q}$ obtendo uma saída $S_{\mathcal{Q}}$

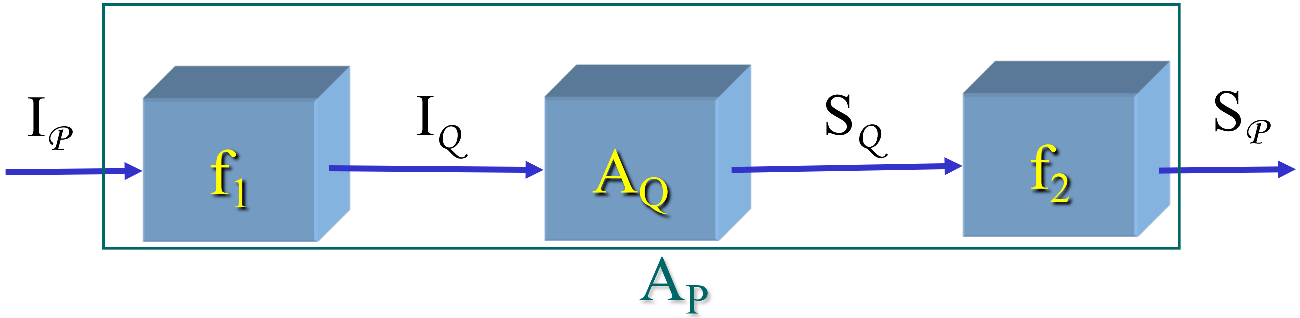
$$S_{\mathcal{Q}} \leftarrow \mathbf{A}_{\mathcal{Q}}(I_{\mathcal{Q}});$$

- existe um algoritmo $\mathbf{f}_2$ que transforma a solução de $\mathcal{Q}$ em solução de $\mathcal{P}$:

$$S_{\mathcal{P}} \leftarrow \mathbf{f}_2(S_{\mathcal{Q}});$$

Problemas NP-Completos

Redução de Problemas



Tempo da Redução $\propto$: $T(f_1) + T(f_2)$

Tempo de A_P: $T(f_1) + T(A_Q) + T(f_2)$

Problemas NP-Completos

Redução de Problemas

- Se $\mathcal{P} \propto Q \Rightarrow \mathcal{P}$ pode ser considerado um *caso particular* de Q.

- $\mathcal{P}$ e Q são **polinomialmente equivalentes** se e somente se $\mathcal{P} \propto Q$ e $Q \propto \mathcal{P}$.

- **Propriedade** "A relação $\propto$ é transitiva":
  Se $\mathcal{X} \propto \mathcal{Y}$ e $\mathcal{Y} \propto \mathcal{Z}$ então $\mathcal{X} \propto \mathcal{Z}$

Problemas NP-Completos

Exemplo 1 (*Redução de Problemas*):

$\mathcal{P}$ = Resolver uma equação de primeiro grau:

$$px + q = 0$$

$\mathcal{Q}$ = Resolver uma equação de segundo grau:

$$ax^2 + bx + c = 0$$

- $\mathcal{P} \propto \mathcal{Q}$

  $\mathrm{I}_{\mathcal{P}}$: p e q

  $\mathrm{I}_{\mathcal{Q}} \leftarrow \mathrm{f}_1(\mathrm{I}_{\mathcal{P}})$: $a = 0$, $b = p$, $c = q$

  $\mathrm{A}_{\mathcal{Q}}$: um algoritmo para resolver a equação de segundo grau

  f_2 : faça $\mathrm{S}_{\mathcal{P}} \leftarrow \mathrm{S}_{\mathcal{Q}}$

- ***Tempo da Redução*** $\propto$: $\mathrm{T}(\mathrm{f}_1) + \mathrm{T}(\mathrm{f}_2) = O(1) + O(1) = O(1)$
- ***Tempo para resolver*** $\mathcal{P}$: $\mathrm{T}(\mathrm{f}_1) + \mathrm{T}(\mathrm{f}_2) + \mathrm{T}(\mathrm{A}_{\mathcal{Q}}) = O(1) + O(1)$.

Problemas NP-Completos

Exemplo 2 (*Redução de Problemas*):

$\mathcal{P}$ = Problema da mediana: dada uma lista com n elementos não ordenados, encontrar a mediana.

$$n = 7,\ \ L = [36, 15, 56, 85, 76, 42, 32] \ \Rightarrow\ \ mediana = 42.$$

$\mathcal{Q}$ = Problema do k-ésimo: dada uma lista com n elementos, e um número k. Encontrar o elemento x tal que k-1 elementos sejam <= x.

- $\mathcal{P} \propto \mathcal{Q}$

  $I_{\mathcal{P}}$: lista com n elementos

  $I_{Q} \leftarrow f_1(I_{\mathcal{P}})$: vetor V com n elementos e $k = \lfloor n/2 \rfloor$;

  A_Q: um algoritmo para resolver o problema do k-ésimo.

  f_2 : faça $S_{\mathcal{P}} \leftarrow S_Q$.

- ***Tempo da Redução*** $\propto$: $T(f_1) + T(f_2) = O(1) + O(1) = O(1)$
- ***Tempo para resolver*** $\mathcal{P}$: $T(f_1) + T(f_2) + T(A_Q) = O(1) + O(n\log n)$.

Problemas NP-Completos

- Um problema de decisão $\mathcal{X}$ é **NP-Completo** sss:

1. $\mathcal{X} \in$ NP
2. Todo problema da classe NP pode ser reduzido em *tempo polinomial* ao problema $\mathcal{X}$,

   ou seja, $\forall \mathcal{Y} \in$ NP, $\mathcal{Y} \propto_p \mathcal{X}$.

   ($\propto_p$: é uma redução em tempo polinomial)

Problemas NP-Completos

Teorema (maneira de provar que um problema $\mathcal{X}$ é NP-Completo)

$\mathcal{X}$ é NP-Completo <u>se</u> $\mathcal{X} \in$ NP e existe um problema $\mathcal{Y}$ NP-Completo tal que $\mathcal{Y} \propto_p \mathcal{X}$.

Demonstração:

Suponha que $\mathcal{Y}$ é NP-Completo

$\Rightarrow$ todo $\mathcal{Z} \in$ NP, $\mathcal{Z} \propto_p \mathcal{Y}$,

$\Rightarrow$ Como $\mathcal{Y} \propto_p \mathcal{X}$, pela transitividade, $\mathcal{Z} \propto_p \mathcal{X}$

$\Rightarrow$ Ou seja, todo problema $\mathcal{Z} \in$ NP, $\mathcal{Z} \propto_p \mathcal{X}$

como $\mathcal{X} \in$ NP $\Rightarrow$ $\mathcal{X}$ é NP-Completo.

Problemas NP-Completos

Teorema de Cook (1982 Prêmio Turing = Prêmio Nobel da Computação)

"O problema **SAT** é NP-Completo"

Isto significa que: $\forall \mathcal{X} \in \mathrm{NP},\ \mathcal{X} \propto_p \mathrm{SAT}$

Problemas NP-Completos

Passos para provar que um **Problema de Decisão** X é NP-Completo**:**

1) Mostrar que $X \in$ NP
2) Selecionar um problema X' conhecido NP-Completo
3) Provar que $X' \propto_p X$.

Problemas NP-Completos

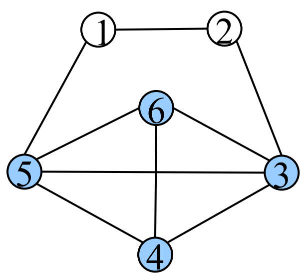
3SAT é NP-Completo

- O problema de decisão **3SAT** é uma versão do **SAT** na qual cada cláusula da expressão booleana na Forma Normal Conjuntiva tem no máximo 3 literais (variáveis) distintos.

- 3SAT $\in$ NP
- Provar que **SAT** $\propto_p$ **3SAT**

Problema da Clique

Dado um grafo conexo $G = (V, A)$, deseja-se saber se G possui uma **clique** (subgrafo completo) de tamanho k, isto é, um subconjunto de k vértices todos conectados.



o subgrafo formado pelos vértices 3, 4,5 e 6 é uma clique de tamanho $k = 4$

Problema da Clique é NP-Completo

1) Clique $\in$ NP (*Existe um algoritmo não-determinístico polinomial*)
2) 3SAT $\propto_p$ Clique

$\Rightarrow$ pelo Teorema 1, Clique é NP-Completo.

Problemas NP-Completos

Provar que 3SAT $\propto_p$ Clique:

- I_{3SAT} : seja $E = \wedge_{j=1}^{m} C_j$ uma expressão booleana na Forma Normal Conjuntiva com no máximo 3 literais distintos em cada clausula.

- $I_{Cliq} \leftarrow f_1(I_{3SAT})$: Transformar a entrada do problema SAT numa entrada do problema da Clique.

  A partir de I_{3SAT} construir um grafo $G = (V, A)$ que possua uma clique de tamanho $k = m$ (número de clausulas).

  O grafo pode ser definido por:

  $V = \{(x, i) \,/\, x$ é literal na cláusula $i\}$,

  cada literal corresponde a um vértice

  $A = \{((x, i), (y, j)) \,/\, x \neq y,\ y \neq \neg x, i \neq j\ \}$

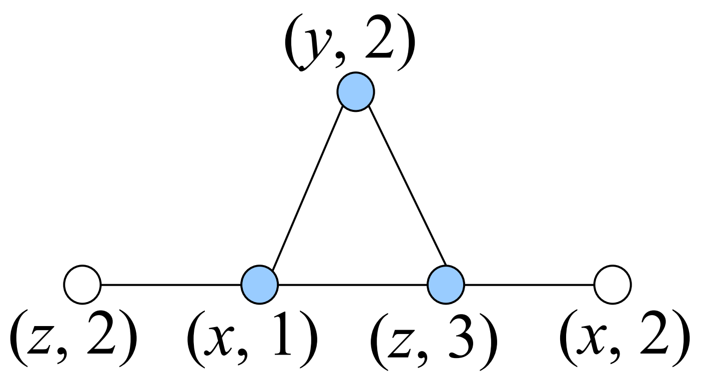
  $x_i, x_j \in C_k$ não formam aresta.

  $x_i \in C_k$ e $\neg x_i \in C_j$ não formam aresta.

Problemas NP-Completos

Exemplo: $E = (x) \wedge (x \vee y \vee z) \wedge (z)$

$$V = \{(x, 1),\ (x, 2), (y, 2), (z, 2), (z, 3)\}$$



Se a expressão booleana E é *verdadeira*, haverá pelo menos um literal x em cada cláusula que é verdadeira.

$\Rightarrow$ O grafo construído sempre possuirá uma **clique** de tamanho $k = m$ (número de clausulas):

$$S = \{(x, i) \ / \ x \text{ é verdadeiro em } C_i\} : S_{\text{Cliq}}$$

Problemas NP-Completos

$S_{3SAT} \leftarrow f_2(S_{Cliq})$

Seja $G = (V, A)$ um grafo qualquer que possui uma clique V' de tamanho k.

Construir uma saída para S_{3SAT}

"A saída do SAT é um conjunto de valores para as variáveis que torne E verdadeira."

Seja $S' = \{x \,/\, (x, i) \in V'$ para algum $i\}$ o conjunto de literais correspondentes aos vértices da clique V'.

S' não pode conter ao mesmo tempo um literal x e $\neg x$ (pois não há aresta conectando (x, i) e $(\neg x, j)$).

Problemas NP-Completos

Fazemos $x = 1$ (***true***) <u>se</u> $x \in S'$, e $x = 0$ (***false***) <u>se</u> $\neg x \in S'$.

Atribuir valores arbitrários para os literais que não estejam em S'.

$\Rightarrow E = (\ldots \vee x \vee \ldots) \wedge (\ldots \vee y \vee \ldots) \wedge (\ldots \vee z \vee \ldots)$, tem k cláusulas e **é verdadeira**.

Problemas NP-Completos

O Problema de Cobertura de Vértices é NP-Completo

- **Provar que Clique $\propto_p$ Cobertura de Vértices (CV)**
- I_{Cliq}: $G = (V, A)$ um grafo conexo, $|V| = n$, k (tamanho da clique) .
- $I_{CV} \leftarrow f_1(I_{Cliq})$:

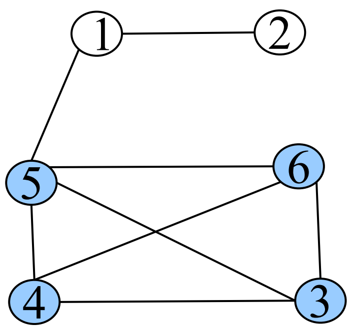
  A partir de G vamos construir um grafo G' que tenha uma CV de tamanho $n-k$ (desde que G possuir uma clique de tamanho k).
- O grafo G' pode ser definido como:

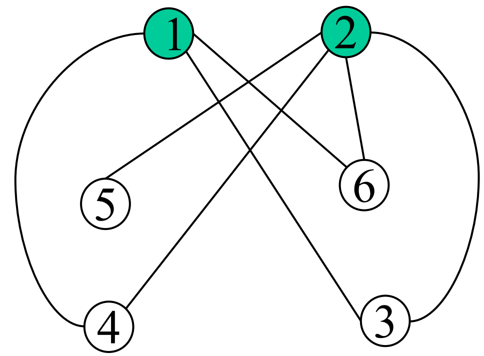
  $G' = (V, A')$ onde $A' = \{(u, v) \,/\, u,v \in V$ e $(u, v) \notin A\}$.

Problemas NP-Completos

- **Exemplo:**

$G = (V, A)$ $\xrightarrow{f_1}$ $G' = (V, A')$





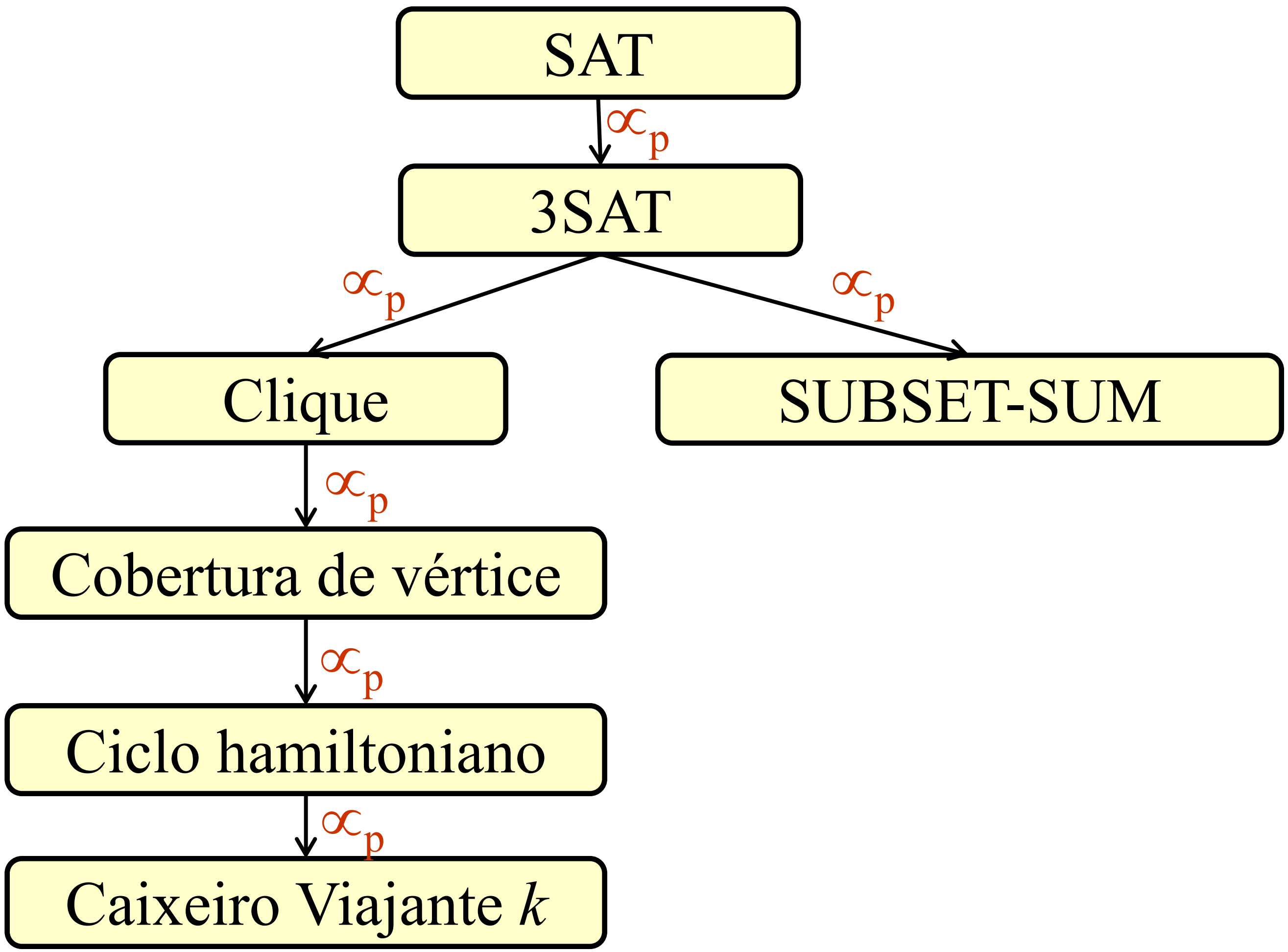
Os vértices da clique estão conectados aos $n-k$ vértices restantes

- $S_{Cliq} \leftarrow f_2(S_{CV})$:

  Se S é uma CV para G' então $V-S$ forma uma **clique** em G.

Problemas NP-Completos

Sequencia de provas de problemas NP-Completos (provas baseadas em reduções):

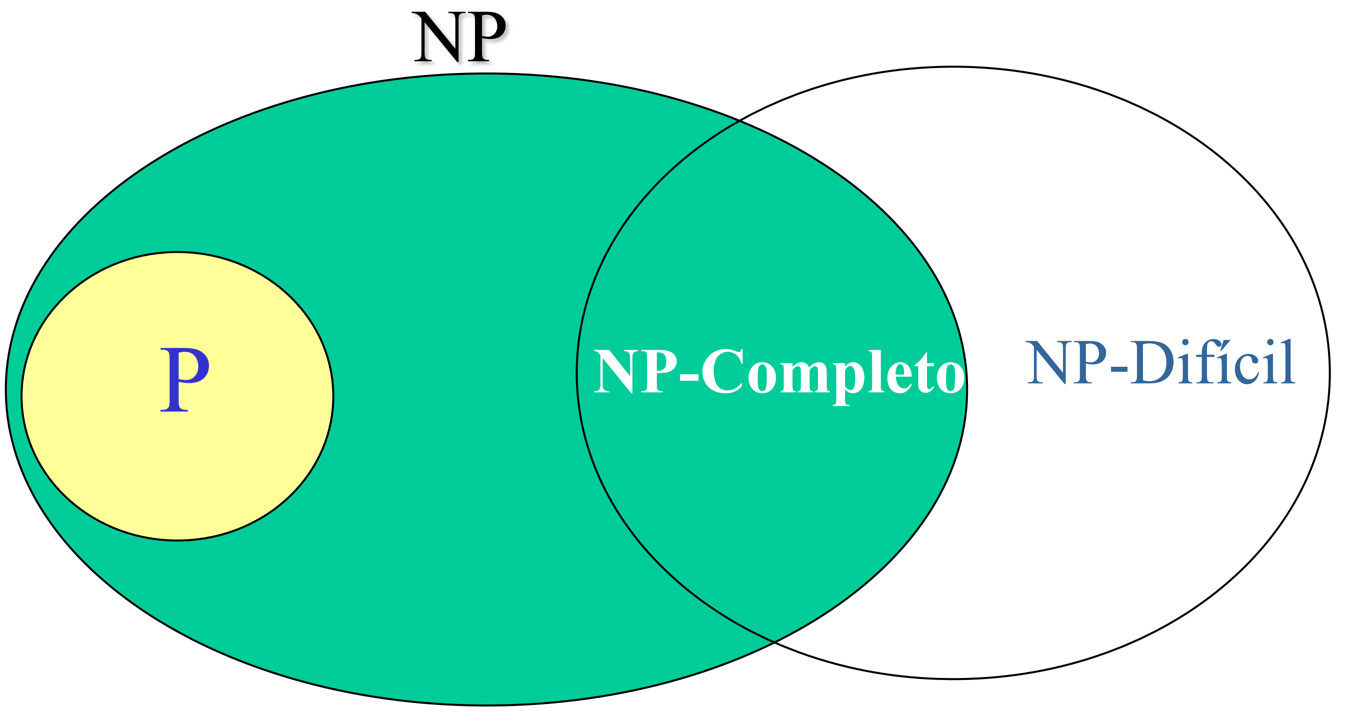


Problemas NP-Difícil

- Um problema $\mathcal{X}$ é **NP-Difícil** sss todo problema da classe NP é redutível a $\mathcal{X}$ em tempo polinomial, ou seja, $\forall\ \mathcal{Y} \in \mathrm{NP},\ \ \mathcal{Y} \propto_p \mathcal{X}$.
- $\mathcal{X}$ é **NP-Difícil** sss existe um problema NP-Completo $\mathcal{Y}$ tal que $\mathcal{Y} \propto_p \mathcal{X}$.
- Um problema NP-Difícil é pelo menos tão difícil quanto os problemas NP-Completo.
- Existe uma **classe intermediária** entre P e NP constituída por problemas que não são da classe P nem da classe NP-Completo, ou seja, ninguém conseguiu uma redução polinomial de um problema NP-Completo para eles.

Problemas NP-Difícil

- Conjetura:



Problemas NP-Difícil

- Todo problema NP-Completo é NP-Difícil
- **Exemplo**:
  - SAT é NP-completo e NP-difícil.
- Apenas **problemas de decisão** ("sim/não") podem ser NP-Completo.
- **Problemas de otimização** podem ser NP-difícil.

Problemas NP-Difícil

- Há problemas que são NP-difíceis, mas não NP-completos.
  - O **problema da parada (*PP*)** é NP-difícil, mas não é NP-completo,

Problemas NP-Difícil

- O *PP* é um problema clássico que consiste em determinar se **um dado algoritmo** (determinístico) sempre vai parar (ou seja, terminar sua execução) **para uma dada entrada**) ou se vai executar infinitamente.
- Alan Turing provou, em 1936, que é **impossível resolver** o *PP* generalizando para qualquer par *algoritmo-entrada*.
- O *PP* é indecidível. Não há algoritmo de qualquer complexidade para resolve-lo.