Projeto e Análise de Algoritmos II

Classes de Complexidade de Problemas

Prof. Dr. Osvaldo Luiz de Oliveira

Estas anotações devem ser complementadas por apontamentos em aula.

Tempo polinomial

Um algoritmo A, com entrada de tamanho igual a n, é **polinomia**l se a sua complexidade (tempo, pior caso) é $O(n^k)$ pata algum $k \geq 0$.

Problemas

- Todo problema para o qual existe um algoritmo polinomial é dito ser **tratável**.

- Inversamente, o problema é dito ser **intratável**.

Por que o tempo polinomial é o divisor de águas?

- As diferenças entre tempo polinomial e super-polinomial são astronômicas.

- Embora $O(n^{1000})$ seja um limite superior muito grande, existem poucos problemas práticos que admitem algoritmos polinomiais com alto grau polinomial.

"Deus não é tão cruel assim".

Problemas de decisão

- Problema cuja resposta é 1 (“sim”) ou 0 (“não”).

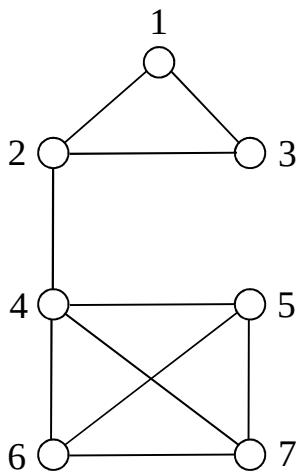
- A teoria estudada restringe a atenção a problemas de decisão.

CLIQUE

Dado um grafo não orientado G = (V, E) e uma constante $k \geq 0$.

Problema: determinar se G tem uma clique de tamanho igual a *k*.

Uma clique de G = (V, E) é um conjunto de vértices $C \subseteq V$ tal que existe aresta em E entre dois quaisquer vértices de C.



Cliques de tamanho 2: {2, 4}, {5, 7} etc. .

Cliques de tamanho 3: {1, 2, 3}, {4, 5, 6} etc.

Clique de tamanho 4: {4, 5, 6, 7}.

COBERTURA DE VÉRTICES (VC)

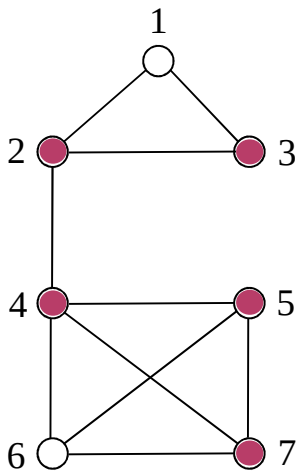
Dado um grafo não orientado G = (V, E) e uma constante $k \geq 0$.

Problema: determinar se G tem VC de tamanho igual a k.

> C $\subseteq$ V é VC de G = (V, E) se e somente se para toda aresta $(u, v) \in$ E ocorre de $u \in$ C ou $v \in$ C.

Uma VC de tamanho igual a 7:
C = {1, 2, 3, 4, 5, 6, 7}.

Uma VC de tamanho igual a 5:
C = {2, 3, 4, 5, 7}.

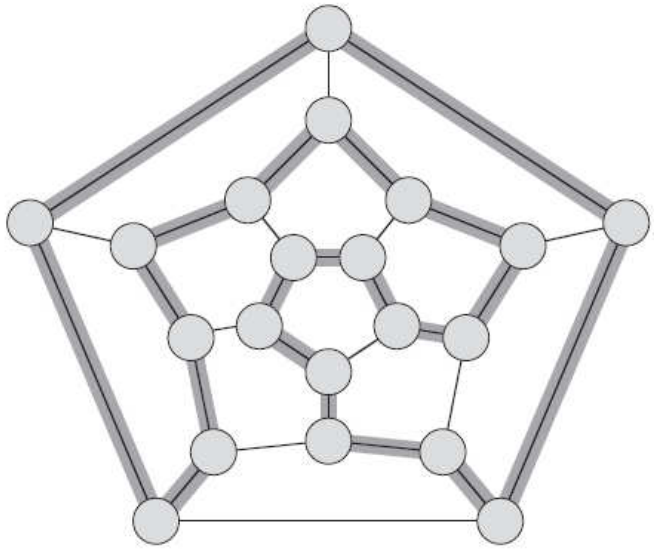


CICLO HAMILTONIANO

Dado um grafo não orientado G = (V, E).

Problema: determinar se G tem um ciclo hamiltoniano.

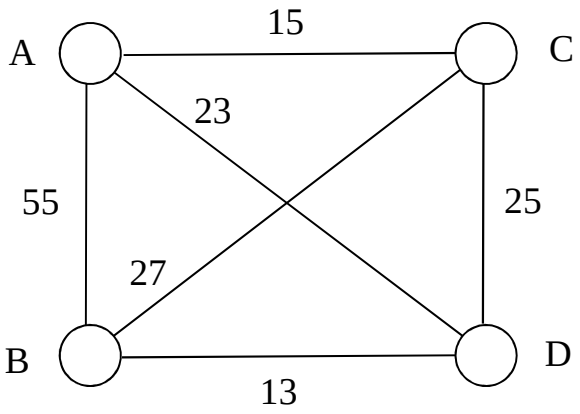
Um ciclo hamiltoniano é um circuito (simples) que contém cada vértice de G exatamente uma vez.



CAIXEIRO VIAJANTE (TRAVELING SALESMAN – TSP)

Dado um grafo não orientado G = (V, E), completo, com custo nas arestas, e um número W.

Problema: determinar se G tem um ciclo hamiltoniano de custo ≤ W.



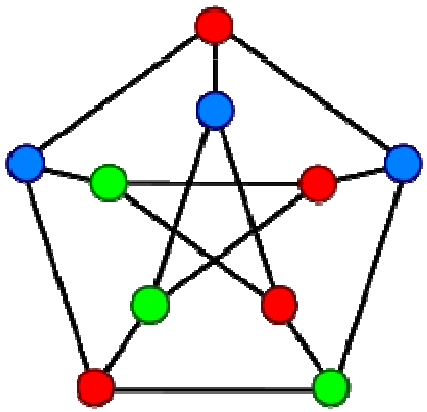
W = 80

Um ciclo hamiltoniano de custo ≤ 80:
A, D, B, C, A

COLORAÇÃO COM TRÊS CORES (3-COLORING)

Dado um grafo não orientado G = (V, E).

Problema: determinar se G pode ser colorido com três cores.



Grafo de Petersen.

COLORAÇÃO COM *K* CORES (*K*-COLORING)

Dado um grafo não orientado G = (V, E) e um inteiro *k*.

Problema: determinar se G pode ser colorido com *k* cores.

Em outras palavras, determinar se existe função $f : V \rightarrow \{1, 2, ..., k\}$ tal que para toda aresta $(u, v) \in E$ ocorre que $f(u) \neq f(v)$.

SOMA DO SUBCONJUNTO (SUBSET-SUM)

Dado conjunto S de *n* inteiros e um inteiro *k*.

Problema: determinar se existe subconjunto S´ de S cuja soma seja igual a *k*.

Exemplo: sejam S = {2, 5, 8, 9, 15, 18} e *k* = 22.

Neste caso existe S´ = {2, 5, 15}.

PARTIÇÃO

Dado um conjunto S = $\{s_1, s_2, ..., s_n\}$ de números.

Problema: determinar se existe subconjunto T de S tal que

$$\sum_{s_i \in T} s_i = \sum_{s_i \in S-T} s_i .$$

Exemplo: seja S = {1, 3, 8, 9, 15, 18}.

Neste caso existe T = {1, 3, 8, 15}, uma vez que
S – T = {9, 18}.

MOCHILA (*KNAPSACK*)

Dado um conjunto S de objetos numerados de 1 a *n*. Cada objeto *i* tem associado um inteiro s_i e um valor w_i. Também são dados dois valores *C* e *W*.

Problema: determinar se existe subconjunto T de S tal que

$$\sum_{i \in T} s_i \leq C \quad e \quad \sum_{i \in T} w_i \geq W.$$

Exemplo: sejam

	1	2	3	4	5	6	7
s	5	3	7	2	6	4	8
w	5.2	3.1	2.2	2.3	3.3	1.1	2.4

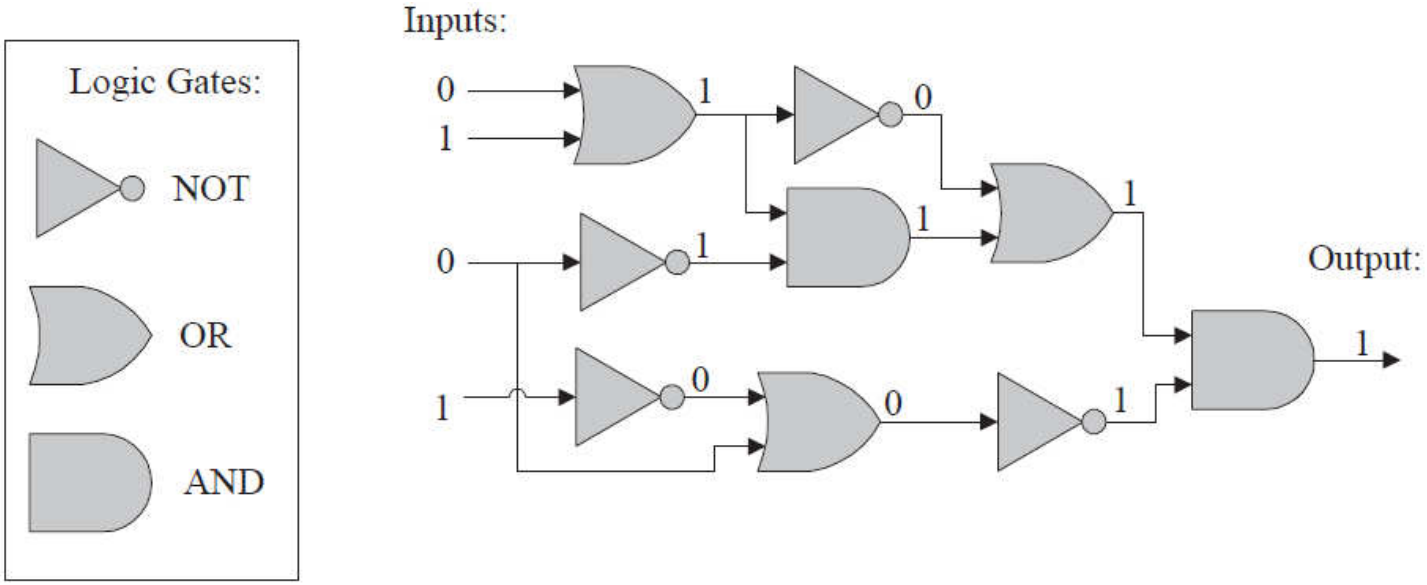
, C = 10 e W = 6

Neste caso existe $T \subseteq S$, T = {2, 4, 6}.

“SATISFATIBILIDADE” DE CIRCUITO (CIRCUIT-SAT)

Dado um circuito composto por portas NOT, OR e AND com um único pino de saída.

Problema: determinar se existe uma atribuição de valores para as entradas de forma que a saída seja igual a 1.



“SATISFATIBILIDADE” DE FÓRMULA BOOLEANA (SAT)

Dado uma formula ϕ composta variáveis booleans x_1, x_2, ... x_n, conectivos lógicos ($\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$) e parênteses.

Problema: determinar se existe uma atribuição de valores para as variáveis de ϕ de forma que a fórmula seja avaliada igual a 1.

$\phi = (x_1 \wedge x_2)$ pode ser satisfeita com $x_1 = 1$ e $x_2 = 1$.

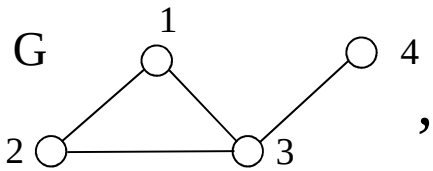
$$\phi = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$$

Pode ser satisfeita com $x1 = 0$, $x_2 = 0$, $x_3 = 1$ e $x_4 = 1$.

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

Codificação da entrada

- Cada problema de decisão possui uma infinidade de instâncias de problema.
- Uma instância do problema de decisão CLIQUE:



, $k = 3$

O grafo possui uma clique de tamanho igual a 3?

- Cada instância está associada a uma cadeia (codificação). Exemplo:

x = #grafo G#3#

↓

1,2,3,4$(1,2),(1,3),(2,3),(3,4)

x = #1,2,3,4$(1,2),(1,3),(2,3),(3,4)#3#

Um pouco de linguagens formais

- Σ (alfabeto): conjunto finito de símbolos.

  Σ = { #, $, ', (,), 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

- $\Sigma . \Sigma = \Sigma^2$ (concatenação de Σ com Σ)

  $\Sigma . \Sigma$ = { ##, #$, #', #(, #), #0, #1, ... $#, $$, $', ... }

- $\Sigma . \Sigma^2 = \Sigma^3$

  $\Sigma . \Sigma^2$ = { ###, ##$, ##', ##(, ... #$#, #$$, #$', ... }

- Σ^0 = { ε }, onde ε representa a cadeia vazia.

Um pouco de linguagens formais

- Σ^* (conjunto de todas as cadeias que podem ser construídas com o alfabeto Σ, ou seja,

  $\Sigma^* = \Sigma^0 \cup \Sigma \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$.

Palavras de comprimento

$\Sigma^* = \{$ ε, ⟵ 0

#, $, ', (,), 0, ..., ⟵ 1

##, #$, #', #(, #), #0, #1, ..., ⟵ 2

###, ##$, ##', ##(, ... ⟵ 3

...

}

Um pouco de linguagens formais

- Linguagem L: subconjunto de cadeias de $\sum^*$.

$L_1 = \varnothing = \{\ \}$ (linguagem vazia).

$L_2 = \{$ #, ##, #$# $\}$

$L_3 = \{$ #1,2,3,4$(1,2),(1,3),(2,3),(3,4)#3# $\}$

$L_4 = \{\ \varepsilon\ \}$ (linguagem que contém apenas a cadeia vazia)

Um pouco de linguagens formais

- Algumas operações sobre linguagens. Sejam L, L_1 e L_2 linguagens sobre o alfabeto $\sum$.

  - União: $L_1 \cup L_2 = \{ x \in \sum^* \mid x \in L_1 \text{ou } x \in L_2 \}$

  - Intersecção: $L_1 \cap L_2 = \{ x \in \sum^* \mid x \in L_1 \text{ e } x \in L_2 \}$

  - Concatenação: $L_1 . L_2 = \{ x_1 x_2 \in \sum^* \mid x_1 \in L_1 \text{ e } x_2 \in L_2 \}$

  - Concatenação de L *k* vezes

    $L^k = L . L^{k-1}$, para $k > 0$

    $L^0 = \{ \varepsilon \}$

  - L^* (fecho reflexivo e transitivo de L ou fecho de Kleene)

    $L^* = \{ \varepsilon \} \cup L \cup L^2 \cup L^3 \cup ...$

  - Complemento: $\overline{L} = \sum^* - L = \{ x \in \sum^* \mid x \notin L \}$

Problema de decisão e linguagem

- Problema de decisão: problema cuja solução tem como resposta 1 (“sim”) ou 0 (“não).
- Instâncias de um problema de decisão Q podem ser codificadas sobre um alfabeto $\sum$.
- Uma linguagem L sobre $\sum$ pode representar as instâncias de um problema de decisão.

$$L = \{ x \in \sum^* \mid Q(x) = 1 \}$$

Exemplo
Problema de decisão CLIQUE e Linguagem CLIQUE

- Problema de decisão CLIQUE: dado grafo G=(V, E) e uma constante *k*, responder 1 (“sim”) caso G tenha uma clique de tamanho igual a *k* e 0 (“não”), caso contrário.

- Linguagem CLIQUE (Lc)

  Lc = { $x \in \sum^*$, x = #grafo G#*k*# |

  existe clique de tamanho igual a *k* em G }

  x = #1, 2, 3, 4 $ (1, 2), (1, 3), (2, 3), (3, 4)#3# $\in$ Lc.

  x = #1, 2, 3, 4 $ (1, 2), (1, 3), (2, 3), (3, 4)#4# $\notin$ Lc.

- Observe que $\overline{\text{Lc}}$ é formado por cadeias cujo problema de decisão tem resposta 0 (“não”) e também aquelas com formato impróprio.

Algoritmo de decisão e linguagem aceita

- Algoritmo de decisão: algoritmo que recebe uma cadeia x e retorna 1 (“sim”) ou 0 (“não”).
- Linguagem L aceita por um algoritmo de decisão A.

$$L = \{ x \in \Sigma^* \mid A(x) = 1 \}$$

- Linguagem M rejeitada por um algoritmo de decisão A.

$$M = \{ x \in \Sigma^* \mid A(x) = 0 \}$$

- $L \cup M$ pode ser diferente de Σ^*, pois para A pode entrar em looping para alguma cadeia e, assim, nem aceitar, nem rejeitar.

Linguagem decidida por um algoritmo

- Um algoritmo A decide uma linguagem L se, para toda cadeia $x \in \sum^*$, A (x) = 1 ou A(x) = 0.

Tamanho da entrada (n)

- Quantidade de símbolos (ou de bits) utilizados para codificar uma instância do problema

x = #1,2,3,4$(1,2),(1,3),(2,3),(3,4)#2#

$n = |x| = 35$ ou

$x = 35 . 8 = 280$ bits para uma certa codificação em que cada símbolo é representado por 8 bits.

Classe de complexidade **P**

- **P** = { linguagens L | existe algoritmo A que **aceita** L em tempo polinomial ao tamanho da entrada, no pior caso }

  Em outras palavras: conjunto de todos os problemas de decisão para os quais existe algoritmo polinomial ao tamanho da entrada, no pior caso.

- Observe que a definição da classe **P**, nada é dito sobre o tempo de execução para rejeitar uma cadeia *x* não pertencente a L.

Fechos da classe **P**

A classe **P** é fechada para, união, intersecção, complemento, concatenação e fecho de kleene. Ou seja, se $L, L_1, L_2 \in \mathbf{P}$, então**:**

- $L_1 \cup L_2 \in \mathbf{P}$;
- $L_1 \cap L_2 \in \mathbf{P}$;
- $\overline{L} \in \mathbf{P}$;
- $L_1 . L_2 \in \mathbf{P}$;
- $L^* \in \mathbf{P}$.

Outra definição da classe de complexidade **P**

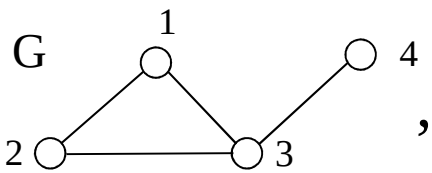
Como **P** é fechado sobre o complemento, então podemos dizer que

P = { linguagens L | existe algoritmo A que **decide** L em tempo polinomial ao tamanho da entrada, no pior caso }

Algoritmos de verificação

- Suponha que alguém lhe ofereceu um conjunto de vértices que ele diz solucionar o problema da CLIQUE igual a *k* em um grafo G.

Por exemplo, alguém diz $y = \{1, 2, 3\}$ para esta instância do problema



, $k = 3$

y é chamado de **certificado**

- É fácil verificar se o certificado $y = \{1, 2, 3\}$ é uma clique em G de tamanho igual 3.

Algoritmo de verificação

- Um algoritmo de verificação toma duas entradas: *x* (a cadeia a ser verificada) e *y* (o certificado).

Algoritmo A (*x*, *y*)

Entrada: $x \in \sum^*$, *x* = #grafo G#*k*# e *y* (certificado), um conjunto de vértices.

Saída: 1 se *y* é uma clique de G de tamanho igual a *k* e 0, caso contrário.

```
{
    se o tamanho de y for igual a k
        se para cada vértice em y houver em G aresta para os outros vértices de y
            retornar 1
        senão
            retornar 0
    senão
        retornar 0
}
```

Classe de complexidade **NP**

- **NP** = { linguagens L | existe algoritmo de verificação A que **aceita** L em tempo polinomial ao tamanho da entrada, no pior caso }

  Em outras palavras: conjunto de todos os problemas de decisão para os quais existe algoritmo de verificação polinomial ao tamanho da entrada, no pior caso.

- Observe que a definição da classe **NP**, nada é dito sobre o tempo de execução para rejeitar uma cadeia *x* não pertencente a L.

Obs.: o nome **NP** vem de *nondeterministic polynomial*. Esta classe foi originalmente estudada no contexto do não determinismo. A definição que estamos usando é equivalente a uma outra que diz que **NP** é a classe das linguagens que definem problemas de decisão para as quais existe algoritmo não determinístico que executa em um tempo polinomial, no pior caso.

Linguagem CLIQUE Lc ∈ **NP**

Algoritmo A (*x*, *y*)
Entrada: $x \in \sum^*$, *x* = #grafo G#*k*# e *y* (certificado), um conjunto de vértices.
Saída: 1 se *y* é uma clique de G de tamanho igual a *k* e 0, caso contrário.

```
{
    se o tamanho de y for igual a k
        se para cada vértice em y houver em G aresta para os outros vértices de y
            retornar 1
        senão
            retornar 0
    senão
        retornar 0
}
```

$O(n^2)$

1 - O certificado $|y| = O(n)$

2 - Algoritmo A aceita Lc em tempo polinomial.

Logo, Lc ∈ NP.

O que também significa que o problema de decisão CLIQUE ∈ NP.

A questão **P = NP**

- Não se sabe se **P = NP**.
- **P** $\subseteq$ **NP**.

Prova

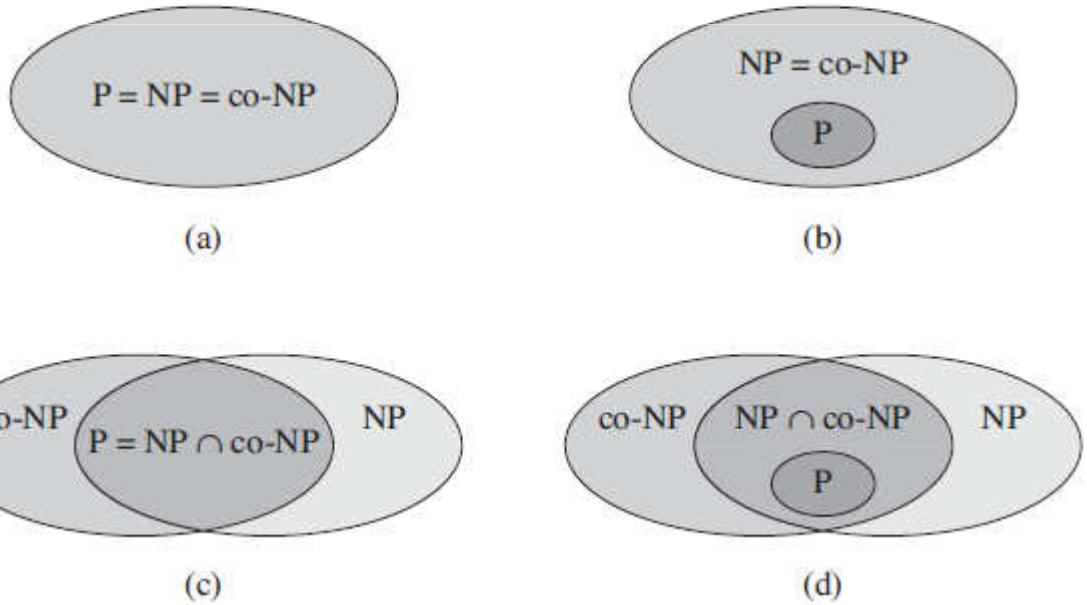
Seja L $\in$ P. Então existe um algoritmo A que decide L em tempo polinomial. Podemos usar A para desenvolver um algoritmo de verificação B que aceita L em tempo polinomial.

```
Algoritmo B (x, y)
{
    retornar A (x) // Ignora o certificado y.
}
```

O algoritmo de verificação B aceita *x* se e somente se A aceita *x*, assim B aceita L. Além disso B executa em tempo polinomial, pois A executa em tempo polinomial. Logo L $\in$ **NP.** Ou seja P $\subseteq$ NP.

Possíveis cenários

- **Não** se sabe se **NP** é fechado pelo complemento, isto é, se $L \in$ **NP** implica que $\overline{L} \in$ **NP**.
- **co-NP** = $\{ L \in NP \mid \overline{L} \in NP \}$.



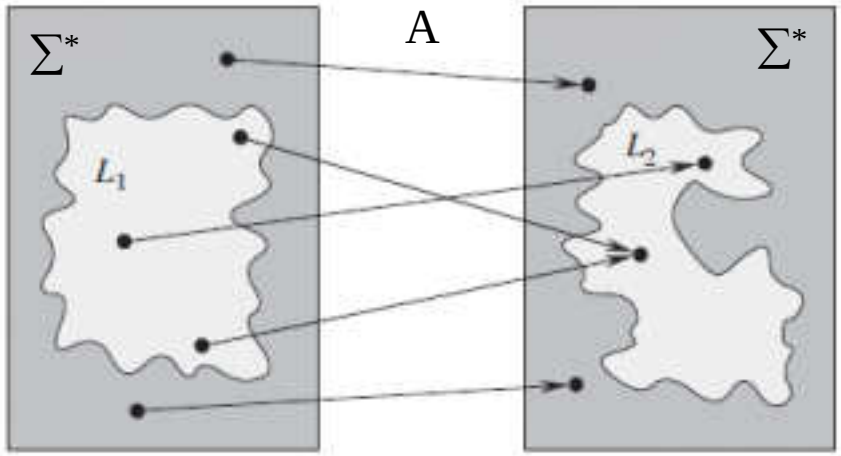
Reduções em tempo polinomial

- Um problema Q pode ser reduzido a um problema R se Q puder ser “refraseado” em termos de R.

O problema de resolver uma equação linear $a\,x + b = 0$ pode ser reduzido ao problema de resolver uma equação do segundo grau $0x^2 + a\,x + b = 0$.

Reduções em tempo polinomial

- Uma linguagem L_1 é redutível em tempo polinomial a uma linguagem L_2, escreve-se $\mathbf{L_1 \leq_p L_2}$, se existe um algoritmo polinomial A tal que, para todo $x \in \Sigma^*$:
  - $A(x) = y$ (i. e., A transforma x em y)
  - $x \in L_1$ se e somente se $y \in L_2$.



Implicação importante das reduções polinomiais

Se $L_A \leq_p L_B$ e existe algoritmo B que decide L_B em tempo polinomial
então
existe algoritmo A que decide L_A em tempo polinomial.

```
Algoritmo A (x)
{
    y := Reduz_LA_LB (x); // Obtém uma instância y do problema B a partir de
                          // uma instância x do problema A.

    retornar B (y) // Retorna 1 ("sim") se B retornar 1 e 0 ("não"), caso contrário.
}
```

O (n^k), i.e, polinomial

O (n^c), i.e, polinomial

Complexidade de A: $T(n) = O(n^k) + O(n^c) = O(n^l)$, i.e, polinomial.

Obs. l = max (k, c), l constante

Classe de complexidade **NP-difícil** (NP-hard)

NP-difícil =

{ linguagens L | para todo L' $\in$ NP ocorre que L' $\leq_p$ L }

Isto é, L' não é mais difícil, por um fator polinomial , do que L.

Classe de complexidade **NP-completo**

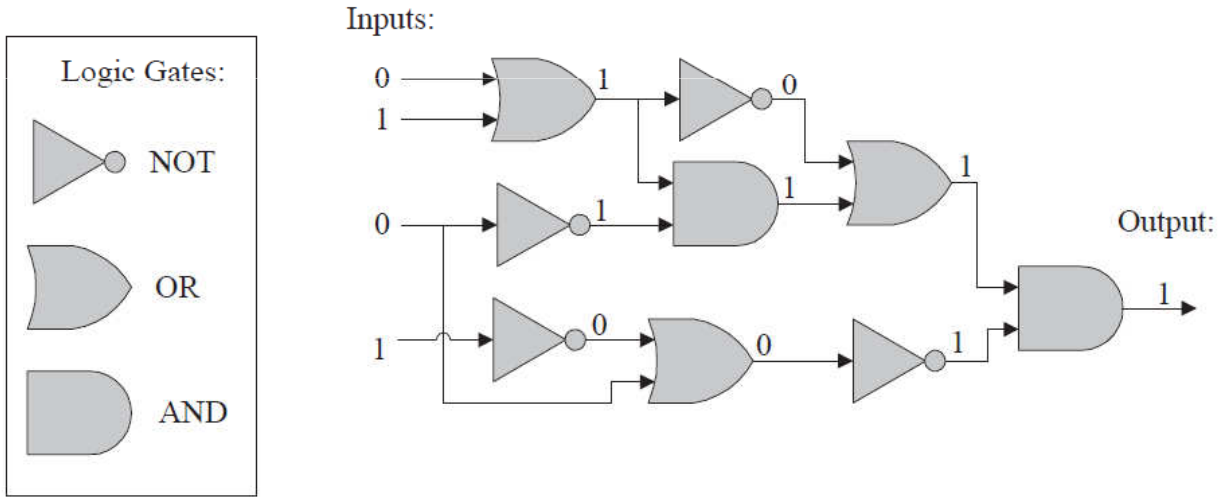
NP-completo =

{ linguagens L | L $\in$ **NP** e L $\in$ **NP-difícil**}

Primeiro problema **NP-completo**

$L_{CIRCUIT\text{-}SAT}$ =

{ #circuito C# | existe uma atribuição de valores para a entrada de forma que a saída seja igual a 1 }



Obs.: C é uma codificação do circuito usando símbolos de um alfabeto ∑.

Primeiro problema **NP-completo**

Teorema de Cook-Levin:

$L_{\text{CIRCUIT-SAT}} \in$ **NP-completo**.

Cook mostrou em 1971que CNF-SAT é **NP-completo**. Levin formulou a noção de NP completude de forma independente de Cook, quase na mesma época. Garey e Johnson (indicado na bibliografia da disciplina) é um catálogo para muitos problemas NP-completos. Cormen (livro texto) fornece uma prova simplificada do teorema.

Stephen Cook. The complexity of theorem proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

L. A. Levin. Universal sorting problems. *Problemy Peredachi Informatsii*, 9(3):265–266, 1973. In Russian.

Outra definição para a classe **NP-completo**

NP-completo =

{ linguagens L | L $\in$ **NP** e

L' $\leq_p$ L para algum L' $\in$ **NP-completo** }

Esta definição simplifica a prova de que um problema de decisão é NP-completo. Não há necessidade de provar L $\in$ **NP-difícil**, i.e, que para todo L' $\in$ NP ocorre de L' $\leq_p$ L. Basta provar que existe L' $\in$ **NP-completo** tal que L' $\leq_p$ L.

Isto decorre do fato: se existe tal L' então todo L'' $\in$ NP, L'' $\leq_p$ L'.
Então, por transitividade L'' $\leq_p$ L' $\leq_p$ L.

Prova de que uma linguagem L $\in$ **NP-completo**

1 - Mostrar L $\in$ **NP**.

2 – Mostrar que L $\in$ **NP-difícil**

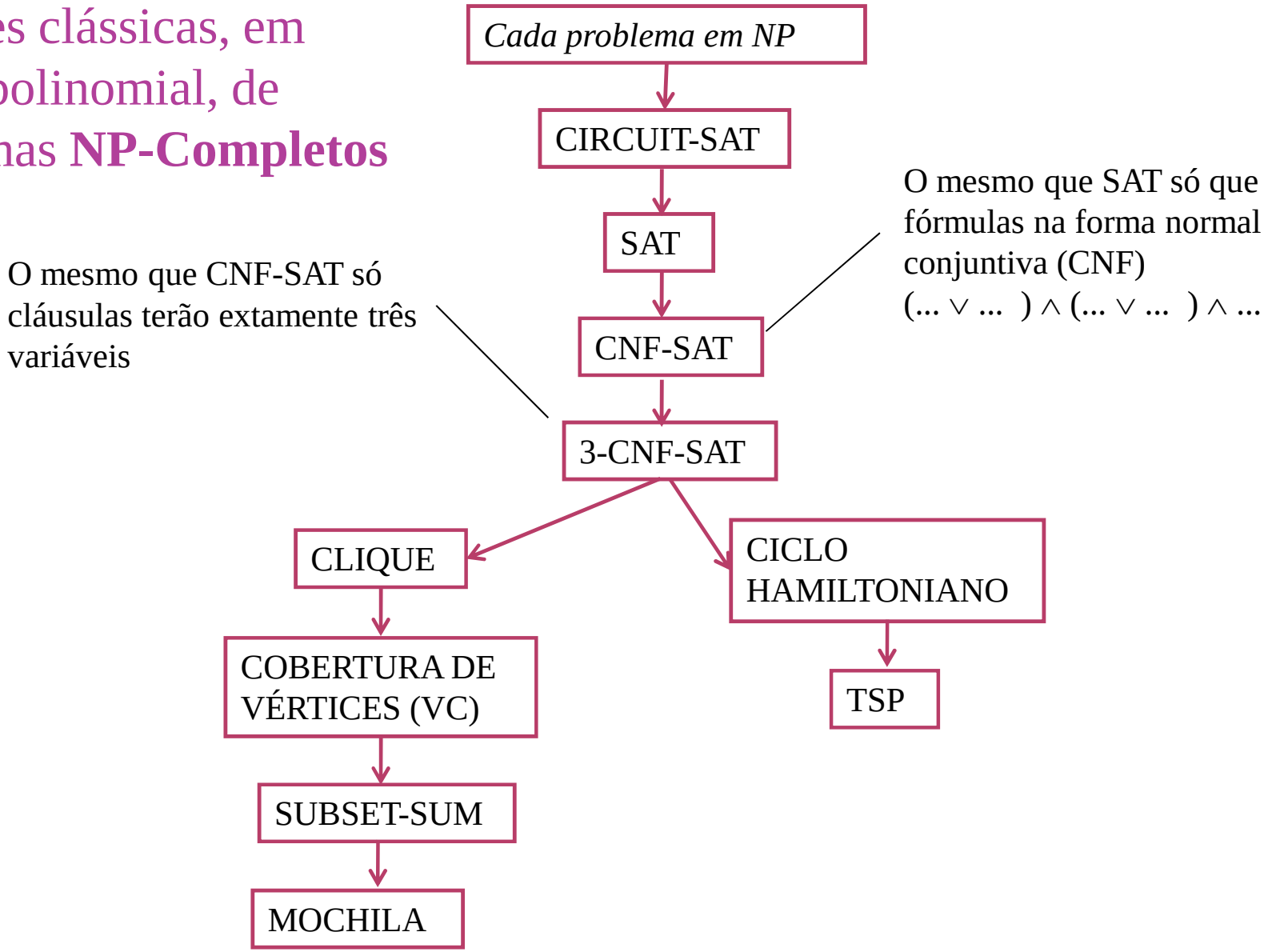
Achar uma linguagem L' $\in$ **NP-completo** tal que L' $\leq_p$ L.

2.1 – Descrever um algoritmo A(x) que reduz em tempo polinomial uma instância x $\in \sum^*$ de L' a uma instância y de L.

2.2 – Mostrar que o algoritmo A executa em tempo polinomial.

2.3 – Mostrar que $x \in$ L' se e somente se $y \in$ L.

Um pequeno quadro de reduções clássicas, em tempo polinomial, de problemas **NP-Completos**



Prova de que $L_{TSP} \in$ **NP-completo**

L_{TSP} = { $x \in \sum^*$, x = #grafo G#W# | existe ciclo hamiltoniano de custo $\leq$ W no grafo completo G }

Precisamos mostrar que:

1 - $L_{TSP} \in$ **NP**.

2 – Que existe L' $\in$ **NP-completo** tal que L' $\leq_p$ L_{TSP}.

1 – Mostrando que $L_{TSP} \in$ **NP**.

Isto é, temos que mostrar que existe algoritmo de verificação A que **aceita** L_{TSP} em tempo polinomial ao tamanho da entrada, no pior caso.

Continuação

Algoritmo A (x, y)
Entrada: $x \in \sum^*$, x = #grafo G#W# e y (certificado), uma sequência de vértices.
Saída: 1 se y é um ciclo hamiltoniano de custo custo $\leq$ W em G .
{
se (y é um circuito simples que passa por todos os vértices de G e tem custo $\leq$ W)
retornar 1
senão
retornar 0
}

$O(n^2)$

- O certificado $| y | = O(n)$.
- Algoritmo A aceita L_{TSP} em tempo polinomial ($T(n) = O(n^2)$).

Logo, $L_{TSP} \in$ **NP**.

Continuação

2 – Mostrando que existe L’ ∈ **NP-completo** tal que L’ $\leq_p L_{TSP}$.

$L_{CH} \in$ NP-completo (admitimos que isto já foi provado).

Mostraremos que $L_{CH} \leq_p L_{TSP}$

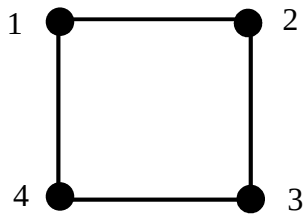
$L_{CH} = \{ x \in \Sigma^*, x = \#\text{grafo G'}\# \mid \text{existe ciclo hamiltoniano em G'} \}$

Precisamos mostrar que $L_{CH} \leq_p L_{TSP}$.

Ideia do algoritmo que reduz em tempo polinomial L_{CH} a L_{TSP}.

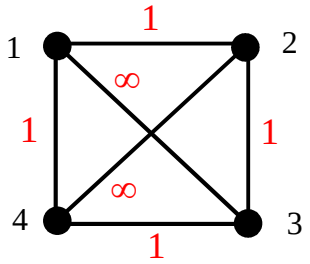
Instância do problema CH

G’ = (V’, E’)



Instância do problema TSP

G = (V’, E), completo, com custos.



W = | E’ | = 4

Continuação

Algoritmo Reduz_L_{CH}_L_{TSP} (x)
Entrada: $x \in \sum^*$, x = #grafo G' = (V', E')#
Saída: $z \in \sum^*$, z = #grafo G = (V', E)#W#
{
1 – W := | E' |;
2 – E := E' ∪ conjunto de arestas que faltam em E' para que G seja um grafo completo;
3 – Inserir peso igual a 1 para as arestas de G que também são arestas de G' ;
4 – Inserir peso ∞ para as arestas de G que não são arestas de G' ;
5 – retornar z
}

O(n)
O(n^2)
O(n^2)

T(n) = O(n^2). O algoritmo de redução é polinomial em relação ao tamanho (n) de x.

Continuação

Precisamos mostrar que: $x \in L_{CH}$ se e somente se $z \in L_{TSP}$.

i) Mostrando: Se $x \in L_{CH} \Rightarrow z \in L_{TSP}$.

Seja $x \in L_{CH}$, x = #grafo G' = (V', E')#.

$\Rightarrow$ Existe um ciclo hamiltoniano em G'.

As arestas de G = (V', E) que também são arestas em G' = (V', E') têm peso igual a 1. A soma dos pesos destas arestas é W = | E' |.

$\Rightarrow$ Assim existe um ciclo hamiltoniano em G de custo igual a W.

$\Rightarrow z \in L_{TSP}$.

Continuação

ii) Mostrando: $z \in L_{TSP} \Rightarrow$ Se $x \in L_{CH}$.

Seja $z \in L_{TSP}$, z = #grafo G = (V', E)#W#.

$\Rightarrow$ Existe um ciclo hamiltoniano em G, grafo completo, de custo igual a W.

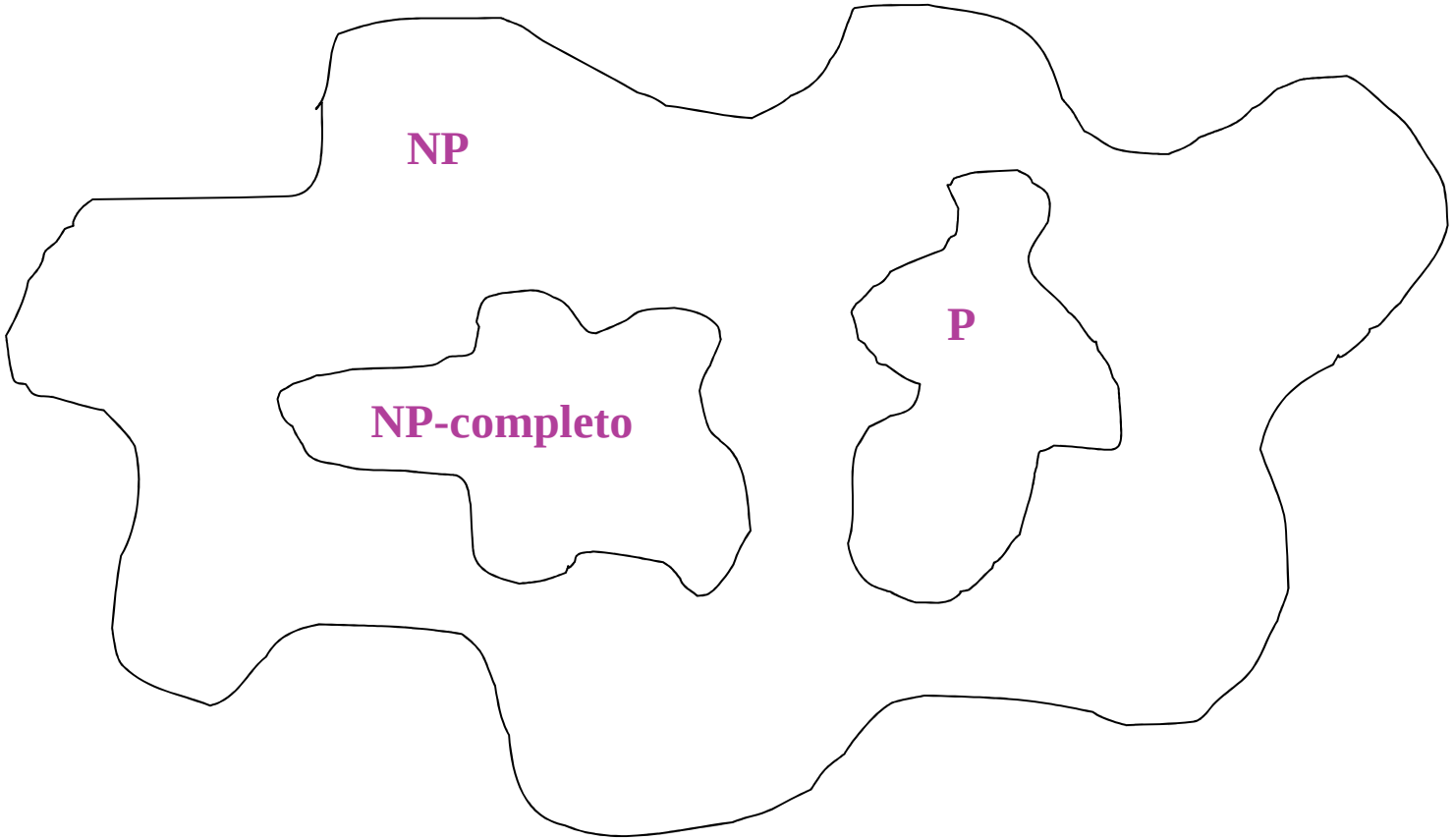
As arestas de G com peso ∞ não participam do ciclo, pois o custo do ciclo é igual a W.

As arestas de G que participam do ciclo são as arestas que também estão presentes em G'.

$\Rightarrow$ G' possui um ciclo hamiltoniano.

$\Rightarrow x \in L_{CH}$.

Muitos acreditam neste relacionamento



Extras

Conjuntos

Definições

Conjunto
Coleção de zero ou mais elementos distintos. ∅ denota um conjunto vazio (zero elemento).

a) **Pertinência**
Se um elemento *x* pertence a um conjunto A, denota-se por $x \in A$. Caso contrário escreve-se $x \notin A$.

b) **Subconjunto e subconjunto próprio**
- $A \subseteq B$: o conjunto A é subconjunto de B, i.e, para todo $x \in A$ ocorre de $x \in B$.
- $A \subset B$: o conjunto A é subconjunto próprio de B, i.e, para todo $x \in A$ ocorre de $x \in B$, mas existe pelo menos um $y \in B$ tal que $y \notin A$.

c) **Igualdade** entre conjuntos A e B
$A = B$ se e somente se $A \subseteq B$ e $B \subseteq A$.

Operações

a) União

$A \cup B = \{ x \mid x \in A \text{ ou } x \in B \}$

b) Intersecção

$A \cap B = \{ x \mid x \in A \text{ e } x \in B \}$

c) Diferença

$A - B = \{ x \mid x \in A \text{ e } x \notin B \}$

d) Complemento em relação a um conjunto universo **U** definido.

$\overline{A} = \{ x \mid x \in \mathbf{U} \text{ e } x \notin A \}$

e) Conjunto das partes

$\wp = 2^A = \{ S \mid S \subseteq A \}$

f) Produto cartesiano

$A \times B = \{ (x, y) \mid x \in A \text{ e } y \in B \}$

Exemplos

Sejam os conjuntos A = { 0, 1, 2 }, B = { 2, 3 } e **N** números naturais.

$A \cup B = \{ 0, 1, 2, 3\}$

$A \cap B = \{ 2 \}$

$A - B = \{ 0, 1\}$

$\overline{A} = \{ x \in \mathbf{N} \mid x > 2 \}$

$\wp = 2^B = \{ \varnothing, \{2\}, \{3\}, \{2, 3\} \}$

$A \times B = \{ (0, 2), (0, 3), (1, 2), (1, 3), (2, 2), (2, 3)\}$

Algumas propriedades

a) Idempotência

$A \cup A = A$

$A \cap A = A$

b) Comutatividade

$A \cup B = B \cup A$

$A \cap B = B \cap A$

c) Associatividade

$A \cup (B \cup C) = (A \cup B) \cup C$

$A \cap (B \cap C) = (A \cap B) \cap C$

d) Distributividade

$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$

e) Morgan

$$A \cup B = \overline{(\overline{A} \cap \overline{B})}$$

$$A \cap B = \overline{(\overline{A} \cup \overline{B})}$$

Lógica

Operadores

Conjunto lógico: { 0, 1} ou { F, V }.

a) Negação (Not): $\neg$

b) E (and): $\wedge$

c) Ou (Or): $\vee$

d) Se então: $\rightarrow$

e) Se e somente se: $\leftrightarrow$

Tabela verdade

x	y	$\neg x$	$x \vee y$	$x \wedge y$	$x \rightarrow y$	$x \leftrightarrow y$
0	0	1	0	0	1	1
0	1	1	1	0	1	0
1	0	0	1	0	0	0
1	1	0	1	1	1	1

Algumas propriedades

a) **Idempotência**

$x \vee x \leftrightarrow x$

$x \wedge x \leftrightarrow x$

b) **Comutatividade**

$x \vee y = y \vee x$

$x \wedge y = y \wedge x$

c) **Associatividade**

$x \vee (y \vee z) = (x \vee y) \vee z$

$x \wedge (y \wedge z) = (x \wedge y) \wedge z$

d) **Distributividade**

$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$

$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$

e) **Morgan**

$x \vee y = \neg (\neg x \wedge \neg y)$

$x \wedge y = \neg (\neg x \vee \neg y)$

Grafos

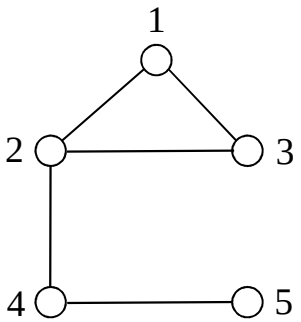
Definição e tipos

Definição

Um grafo G = (V, E) é um sistema matemático constituído por um conjunto de vértices V (ou nós) e um conjunto de arestas E. A cada aresta E corresponde um par de vértices.

Não orientado

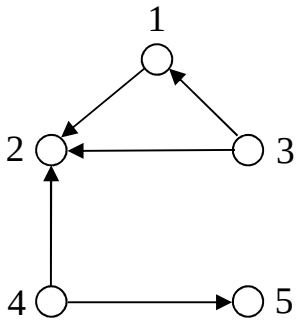
G = (V, E)



V = { 1, 2, 3, 4, 5 }
E = { {1, 2}, {1, 3}, {2, 3},
{2, 4}, {4, 5} }

Orientado

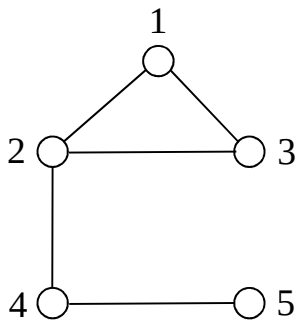
G = (V, E)



V = { 1, 2, 3, 4, 5 }
E = { (1, 2), (3, 1), (3, 2)},
(4, 2), (4, 5) }

Matriz de adjacências

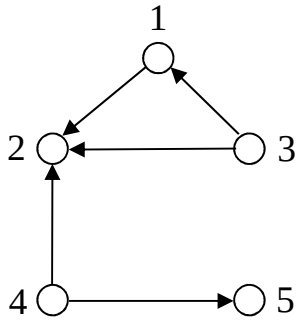
G = (V, E)



	1	2	3	4	5
1		1	1		
2	1		1	1	
3	1	1			
4		1			1
5				1	

M

G = (V, E)



	1	2	3	4	5
1		1			
2					
3	1	1			
4		1			1
5					

M

M [*v*, *w*] = 1: *existe aresta de v para w*

Lista de adjacências

G = (V, E)