Melhores momentos

AULA PASSADA

Verificador polinomial para SIM

Um verificador polinomial para a resposta SIM a um problema Π é uma algoritmo polinomial ALG que recebe

uma instância I de Π e um objeto C, tal que $\langle C \rangle$ é $\mathrm{O}(\langle I \rangle^c)$ para alguma constante c e

$$\text{ALG}(I, C) = \text{SIM} \Leftrightarrow \Pi(I) = \text{SIM}$$

A constante c depende apenas do problema!

Verificador polinomial para NÃO

Um verificador polinomial para a resposta NÃO a um problema Π é uma algoritmo polinomial ALG que recebe

uma instância I de Π e um objeto C, tal que
$\langle C \rangle$ é $\mathrm{O}(\langle I \rangle^c)$ para alguma constante c e

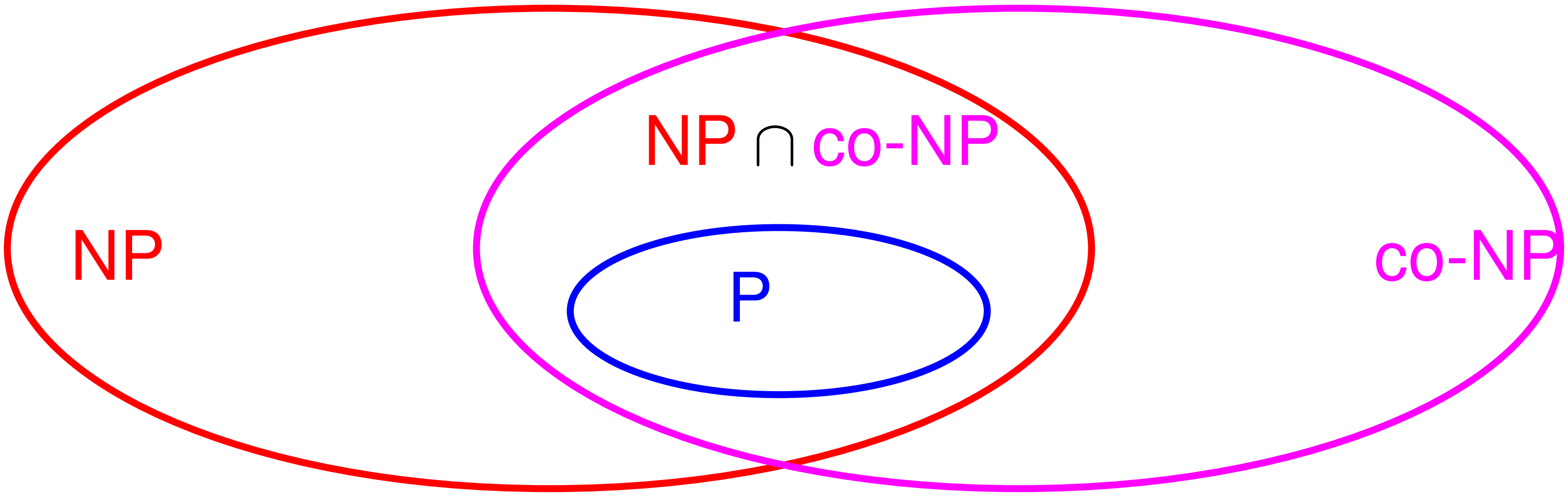
$$\text{ALG}(I, C) = \text{SIM} \Leftrightarrow \Pi(I) = \text{NÃO}$$

A constante c depende apenas do problema!

P, NP e co-NP

Classe P	formada por problemas de decisão que podem ser resolvidos em tempo polinomial
Classe NP	formada por problemas de decisão que possuem um verificador polinomial para a resposta SIM
Classe co-NP	formada por problemas de decisão que possuem um verificador polinomial para a resposta NÃO

P, NP e co-NP



P $\neq$ NP?

NP $\cap$ co-NP $\neq$ P?

NP $\neq$ co-NP?

Redução polinomial

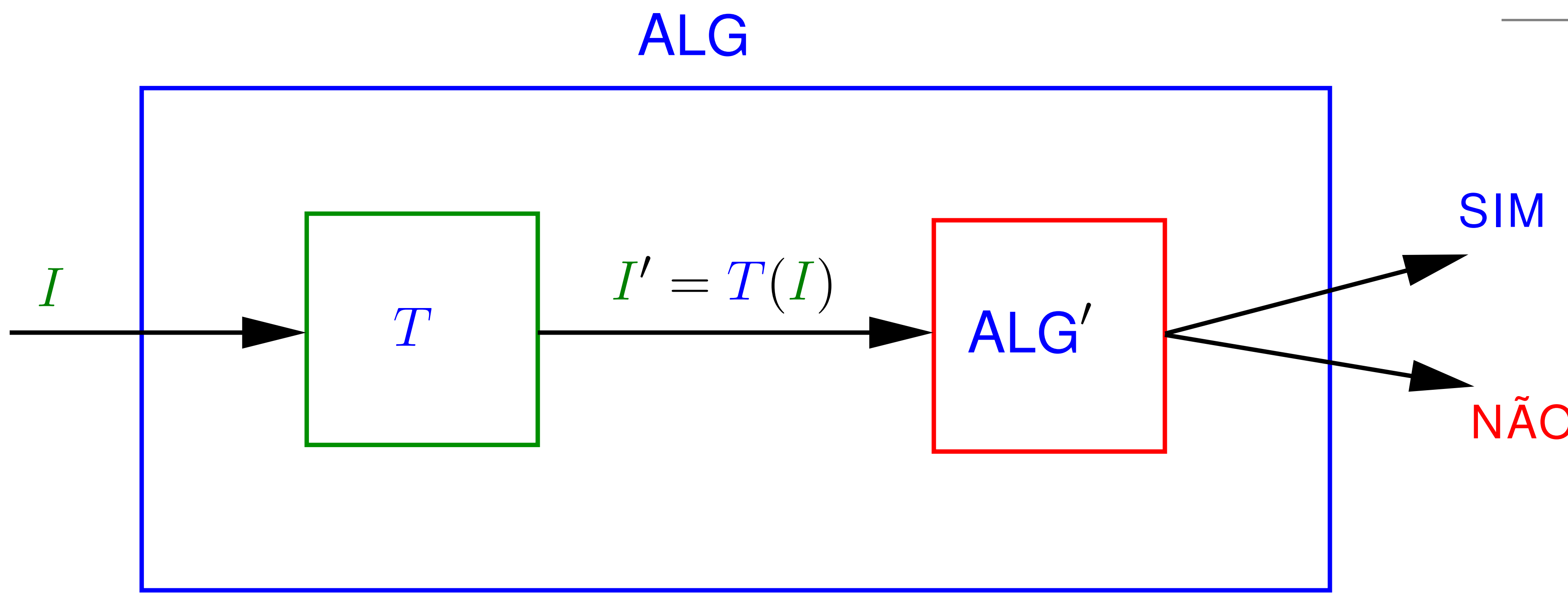
Permite comparar o "grau de complexidade" de problemas diferentes.

Redução (polinomial) de um problema Π a um problema Π' é um algoritmo ALG que resolve Π usando uma subrotina hipotética ALG$'$ que resolve Π', de tal forma que, se ALG$'$ é um algoritmo polinomial, então ALG é um algoritmo polinomial.

$\Pi \leq_P \Pi' =$ existe uma redução de Π a Π'.

Se $\Pi \leq_P \Pi'$ e Π' está em P, então Π está em P.

Esquema comum de reduções



Faz apenas uma chamada ao algoritmo ALG$'$.

T transforma uma instância I de Π em uma instância $I' = T(I)$ de Π' tal que

$$\Pi(I) = \text{SIM se e somente se } \Pi'(I') = \text{SIM}$$

T é uma espécie de "filtro" ou "compilador".

Reduções

Satisfatibilidade $\leq_P$ Sistemas lineares 0-1

Ciclo hamiltoniano $\leq_P$ Caminho hamiltoniano entre u e v

Caminho hamiltoniano entre u e v $\leq_P$ Caminho hamiltoniano

Hoje:

Caminho hamiltoniano $\leq_P$ Satisfatibilidade

Satisfatibilidade $\leq_P$ 3-Satisfatibilidade

3-Satisfatibilidade $\leq_P$ Clique

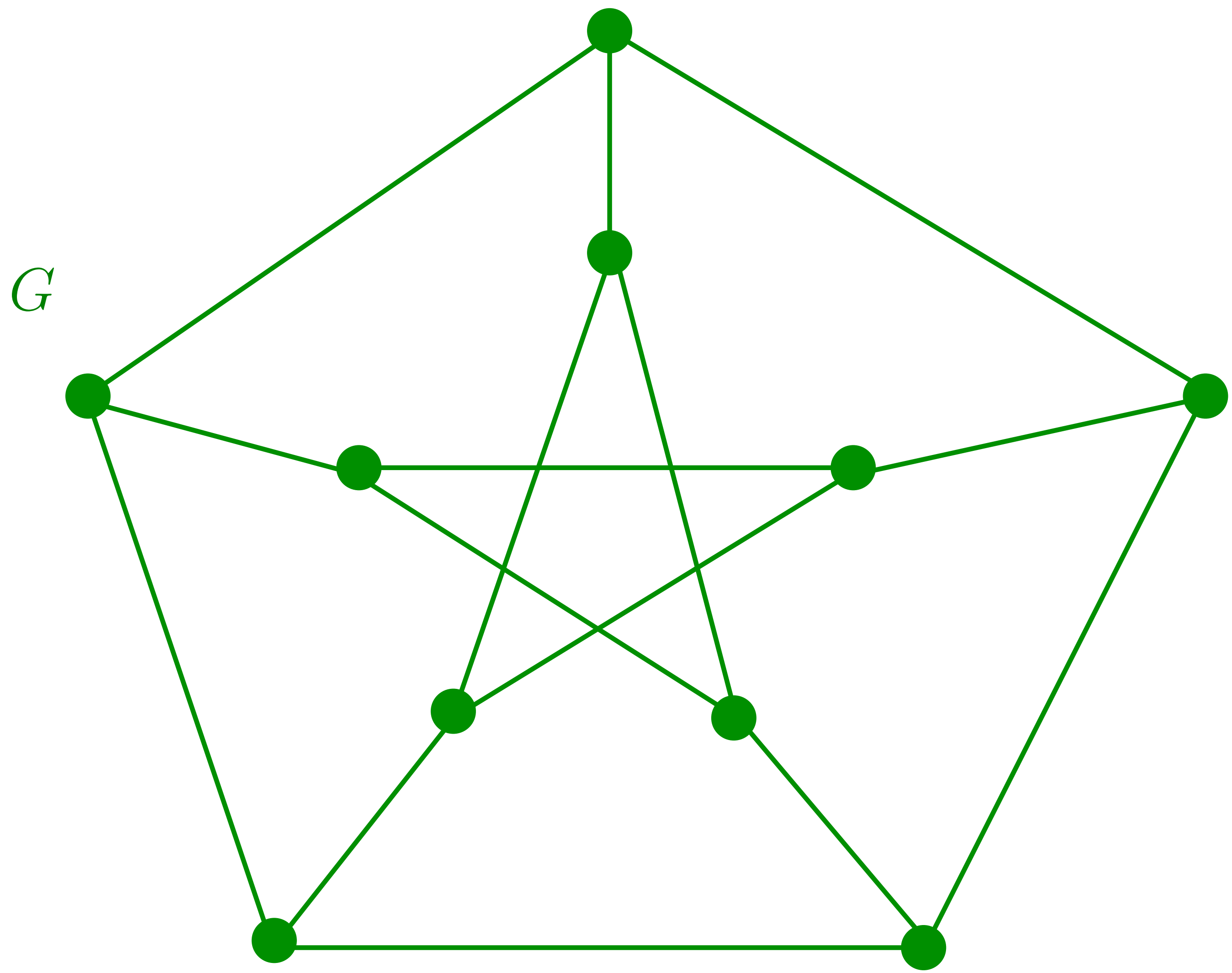
AULA 23

Mais complexidade computacional ainda

CLR 36 ou CLRS 34

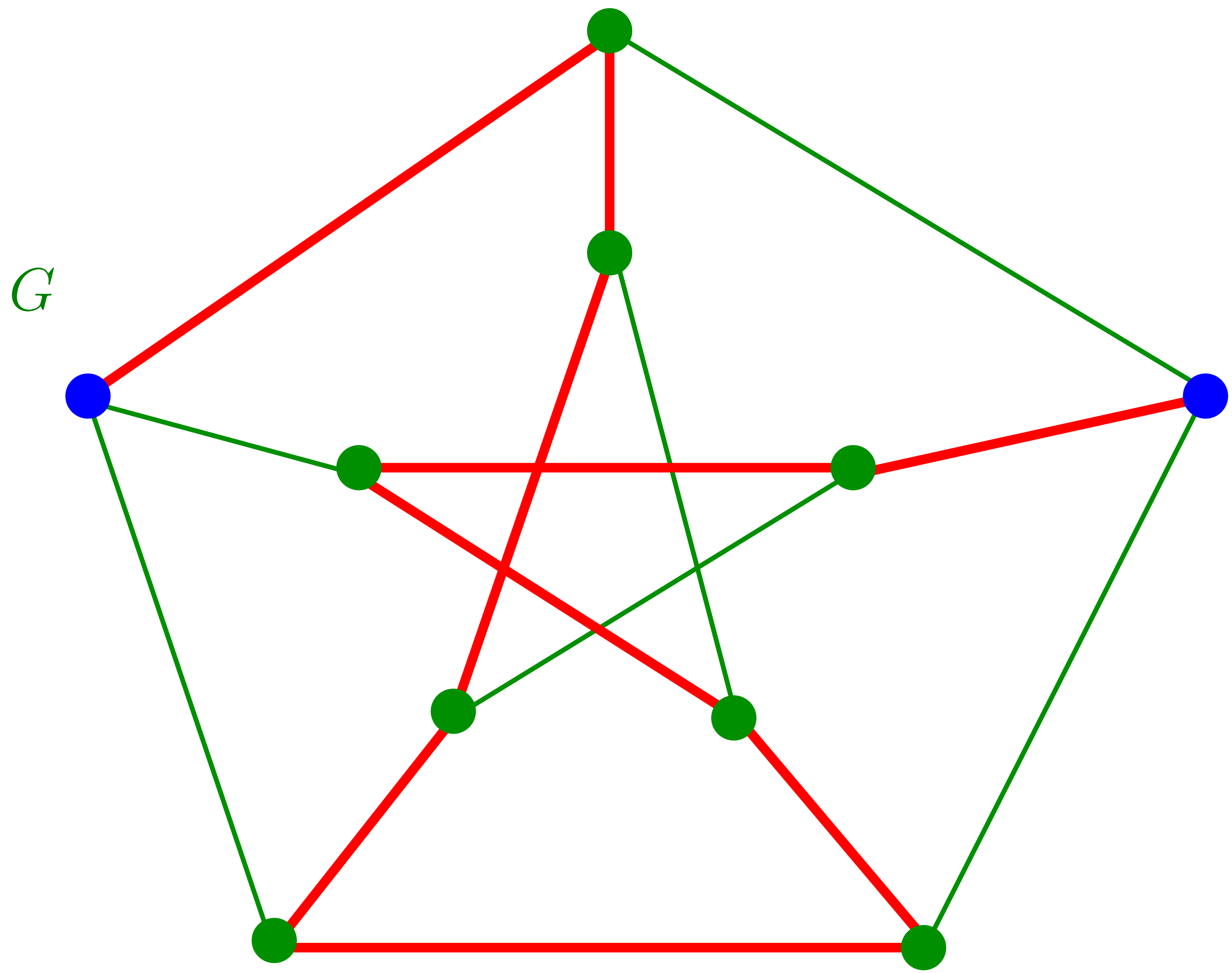
Caminhos hamiltonianos

Problema: um dado grafo possui um caminho hamiltoniano?



Caminhos hamiltonianos

Problema: um dado grafo possui um caminho hamiltoniano?



Satisfatibilidade

Problema: Dada um fórmula booleana ϕ nas variáveis $x_1, \ldots, x_n$, existe uma atribuição

$$t : \{x_1, \ldots, x_n\} \rightarrow \{\text{VERDADE}, \text{FALSO}\}$$

que torna ϕ verdadeira?

Exemplo:

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

Se $t(x_1) = \text{VERDADE}, t(x_2) = \text{FALSO}, t(x_3) = \text{FALSO}$, então $t(\phi) = \text{VERDADE}$

Se $t(x_1) = \text{VERDADE}, t(x_2) = \text{VERDADE}, t(x_3) = \text{FALSO}$, então $t(\phi) = \text{FALSO}$

Exemplo 3

Caminho hamiltoniano $\leq_P$ Satisfatibilidade

Descreveremos um algoritmo polinomial T que recebe um grafo G e devolve uma fórmula booleana $\phi(G)$ tais que

G tem caminho hamiltoniano $\Leftrightarrow$ $\phi(G)$ é satisfatível.

Suponha que G tem vértices $1, \ldots, n$.

$\phi(G)$ tem n^2 variáveis $x_{i,j}$, $1 \leq i, j \leq n$.

Interpretação: $x_{i,j} = \text{VERDADE} \Leftrightarrow$ vértice j é o i-ésimo vértice do caminho.

Exemplo 3 (cont.)

Claúsulas de $\phi(G)$:

- “vértice j faz parte do caminho:

$$(x_{1,j} \lor x_{2,j} \lor \cdots \lor x_{n,j})$$

  para cada j (n claúsulas).

- “vértice j não está em duas posições do caminho:

$$(\neg x_{i,j} \lor \neg x_{k,j})$$

  para cada j e $i \neq k$ ($\mathrm{O}(n^3)$ claúsulas).

- “algum vértice é o i-ésimo do caminho”:

$$(x_{i,1} \lor x_{i,2} \lor \cdots \lor x_{i,n})$$

  para cada i (n claúsulas).

Exemplo 3 (cont.)

Mais claúsulas de $\phi(G)$:

- "dois vértices não podem ser o i-ésimo":

  $$(\neg x_{i,j} \vee \neg x_{i,k})$$

  para cada i e $j \neq k$ ($\mathrm{O}(n^3)$ claúsulas).

- "se ij não é aresta, j não pode seguir i no caminho":

  $$(\neg x_{k,i} \vee \neg x_{k+1,j})$$

  para cada ij que não é aresta ($\mathrm{O}(n^3)$ claúsulas).

A fórmula $\phi(G)$ tem $\mathrm{O}(n^3)$ claúsulas e cada claúsula tem $\leq n$ literais. Logo, $\langle\phi(G)\rangle$ é $\mathrm{O}(n^4)$.

Não é difícil construir o algoritmo polinomial T.

Exemplo 3 (cont.)

$\phi(G)$ satisfatível $\Rightarrow$ G tem caminho hamiltoniano.

Prova: Seja $t : \{\text{variáveis}\} \rightarrow \{\text{VERDADE}, \text{FALSO}\}$ tal que $t(\phi(G)) = \text{VERDADE}$.

Para cada i existe um único j tal que $t(x_{i,j}) = \text{VERDADE}$.
Para cada j existe um único i tal que $t(x_{i,j}) = \text{VERDADE}$.
Logo, t é a codificadação de uma permutação

$$\pi(1), \pi(2), \ldots, \pi(n)$$

dos vértices de G, onde

$$\pi(i) = j \Leftrightarrow t(x_{i,j}) = \text{VERDADE}.$$

Para cada k, $(\pi(k), \pi(k+1))$ é uma aresta de G.

Logo, $(\pi(1), \pi(2), \ldots, \pi(n))$ é um caminho hamiltoniano.

Exemplo 3 (cont.)

G tem caminho hamiltoniano $\Rightarrow \phi(G)$ satisfatível.

Suponha que $(\pi(1), \pi(2), \ldots, \pi(n))$ é um caminho hamiltoniano, onde π é uma pernutação dos vértices de G. Então

$t(x_{i,j})$ = VERDADE se $\pi(i) = j$ e

$t(x_{i,j})$ = FALSO se $\pi(i) \neq j$,

é uma atribuição de valores que satisfaz todas as claúsulas de $\phi(G)$.

3-Satisfatibilidade

Problema: Dada um fórmula booleana ϕ nas variáveis $x_1, \ldots, x_n$ em que cada claúsula tem exatamente 3 literais, existe uma atribuição

$$t : \{x_1, \ldots, x_n\} \rightarrow \{\text{VERDADE}, \text{FALSO}\}$$

que torna ϕ verdadeira?

Exemplo:

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

Um literal é uma variável x ou sua negação $\neg x$.

Exemplo 4

Satisfatibilidade $\leq_P$ 3-Satisfatibilidade

Descreveremos um algoritmo polinomial T que recebe um fórmula booleana ϕ e devolve uma fórmula booleana ϕ' com exatamente 3 literais por claúsula tais que

$$\phi \text{ é satisfatível} \Leftrightarrow \phi' \text{ é satisfatível.}$$

A transformação consiste em substituir cada claúsula de ϕ por uma coleção de claúsulas com exatamente 3 literais cada e equivalente a ϕ.

Exemplo 4 (cont.)

Seja $(l_1 \vee l_2 \vee \cdots \vee l_k)$ uma claúsula de ϕ.

Caso 1. $k = 1$

Troque (l_1) por

$$(l_1 \vee y_1 \vee y_2)\,(l_1 \vee \neg y_1 \vee y_2)\,(l_1 \vee y_1 \vee \neg y_2)\,(l_1 \vee \neg y_1 \vee \neg y_2)$$

onde y_1 e y_2 são variáveis novas.

Caso 2. $k = 2$

Troque $(l_1 \vee l_2)$ por $(l_1 \vee l_2 \vee y)\,(l_1 \vee l_2 \vee \neg y)$. onde y é uma variáveis nova.

Caso 3. $k = 3$

Mantenha $(l_1 \vee l_2 \vee l_3)$.

Exemplo 4 (cont.)

Caso 4. $k > 3$

Troque $(l_1 \vee l_2 \vee \cdots \vee l_k)$ por

$$(l_1 \vee l_2 \vee y_1)$$
$$(\neg y_1 \vee l_3 \vee y_2)\,(\neg y_2 \vee l_4 \vee y_3)\,(\neg y_3 \vee l_5 \vee y_4)\ \ldots$$
$$(\neg y_{k-3} \vee l_{k-1} \vee l_k)$$

onde $y_1, y_2, \ldots, y_{k-3}$ são variáveis novas

Verifique que ϕ é satisfátivel $\Leftrightarrow$ nova fórmula é satisfatível.

O tamanho da nova fórmula é $\mathrm{O}(m)$, onde m é o número de literais que ocorrem em ϕ (contando-se as repetições).

Problemas completos em NP

Um problema Π em NP é NP-completo se cada problema em NP pode ser reduzido a Π.

Teorema de S. Cook e L.A. Levin: Satisfatibilidade é NP-completo.

Se $\Pi \leq_P \Pi'$ e Π é NP-completo, então Π' é NP-completo.

Existe um algoritmo polinomial para um problema NP-completo se e somente se P $=$ NP.

Demonstração de NP-completude

Para demonstrar que um problema Π' é NP-completo podemos utilizar o Teorema de Cook e Levin.

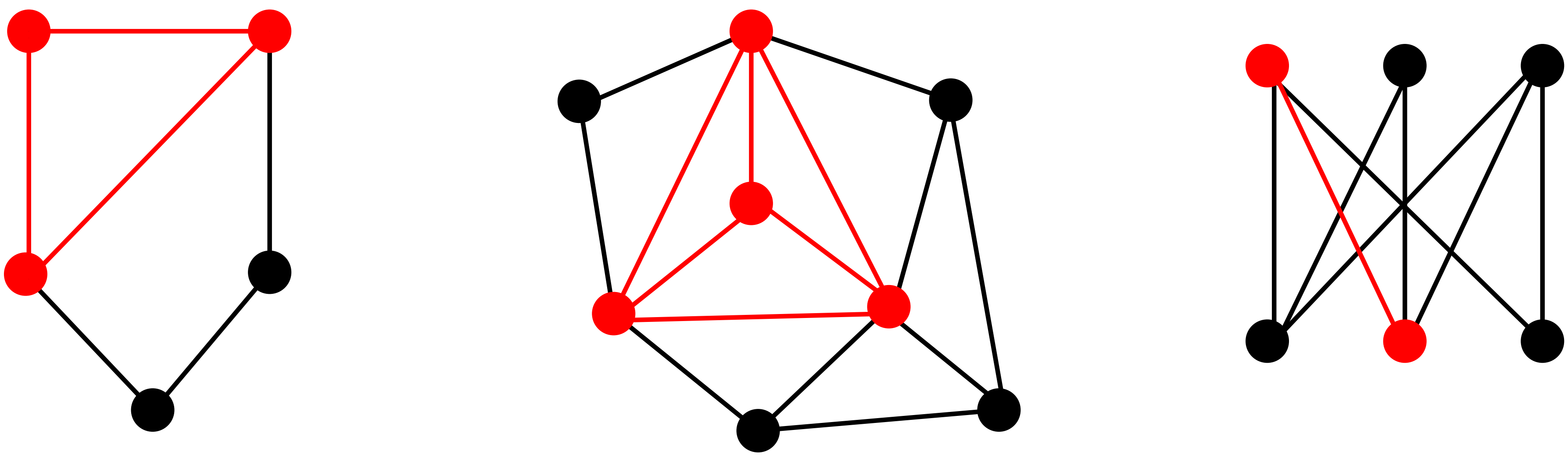
Para isto devemos:

- Demonstrar que Π' está en NP.
- Escolher um problema Π sabidamente NP-completo.
- Demonstrar que $\Pi \leq_P \Pi'$.

Clique

Problema: Dado um grafo G e um inteiro k, G possui um clique com $\geq k$ vértices?

Exemplos:



clique com k vértices $=$ subgrafo completo com k vértices

Clique é NP-completo

Clique está em NP e 3-Satisfatibilidade $\leq_P$ Clique.

Descreveremos um algoritmo polinomial T que recebe um fórmula booleana ϕ com k claúsulas e exatamente 3 literais por claúsula e devolve um grafo G tais que

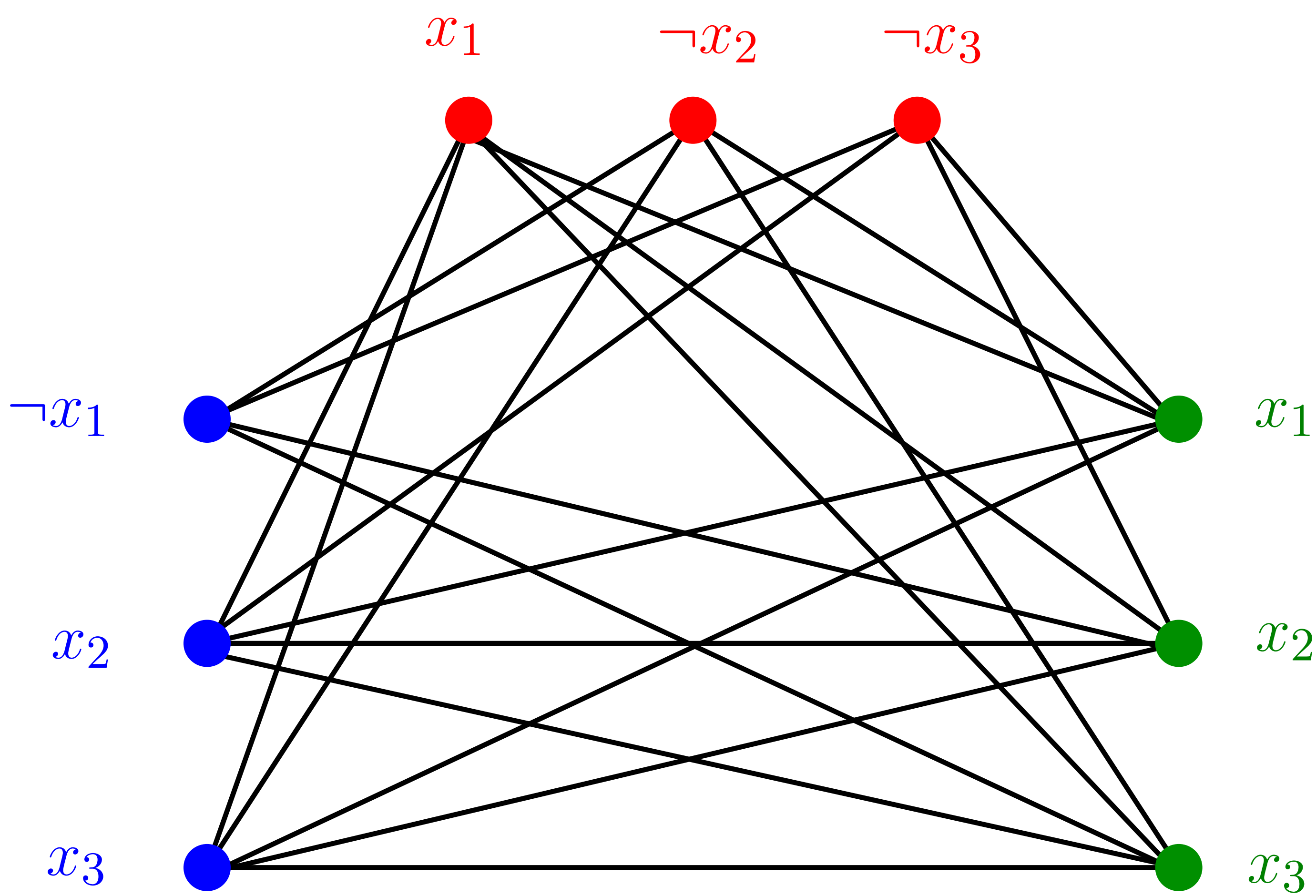
$$\phi \text{ é satisfatível} \Leftrightarrow G \text{ possui um clique} \geq k.$$

Para cada claúsula o grafo G terá três vértices, um correspondente a cada literal da cláusula. Logo, G terá $3k$ vértices. Teremos uma aresta ligando vértices u e v se

- u e v são vértices que correspondem a literais em diferentes claúsula; e
- se u corresponde a um literal x então v não corresponde ao literal $\neg x$.

Clique é NP-completo (cont.)

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



Problemas NP-difíceis

Um problema Π, não necessariamente em NP, é NP-díficil se a existência de um algoritmo polinomial para Π implica em P $=$ NP.

Todo problema NP-completo é NP-difícil.

Exemplos:

- Encontrar um ciclo hamiltoniano é NP-difícil, mas não é NP-completo, pois não é um problema de decisão e portanto não está em NP.
- Satisfabilidade é NP-completo e NP-difícil.

Mais problemas NP-difíceis

Os seguintes problema são NP-difíceis:

- mochila booleana
- caminho máximo
- caminho hamiltoniano
- escalonamento de tarefas
- subset-sum
- clique máximo
- cobertura por vértices
- sistemas 0-1

e mais um montão deles . . .

Exercícios

Exercício 25.A
Suponha que os algoritmos ALG e ALG$'$ transformam cadeias de caracteres em outras cadeias de caracteres. O algoritmo ALG consome $\mathrm{O}(n^2)$ unidades de tempo e o algoritmo ALG$'$ consome $\mathrm{O}(n^4)$ unidades de tempo, onde n é o número de caracteres da cadeia de entrada. Considere agora o algoritmo ALGALG$'$ que consiste na composição de ALG e ALG$'$, com ALG$'$ recebendo como entrada a saída de ALG. Qual o consumo de tempo de ALGALG$'$?

Exercício 25.B [CLRS 34.1-4]
O algoritmo MOCHILA-BOOLEANA é polinomial? Justifique a sua resposta.

Exercício 25.C [CLRS 34.1-5]
Seja ALG um algoritmo que faz um número constante de chamadas a um algoritmo ALG$'$. Suponha que se o consumo de tempo de ALG$'$ é constante então o consumo de tempo de ALG é polinomial.

1. Mostre que se o consumo de tempo de ALG$'$ é polinomial então o consumo de tempo de ALG é polinomial.
2. Mostre que se o consumo de tempo de ALG$'$ é polinomial e ALG faz um número polinomial de chamadas a ALG$'$, então é possível que o consumo de tempo de ALG seja exponencial.

Mais exercícios

Exercício 25.D [CLRS 34.2-1]
Mostre que o problema de decidir se dois grafos dados são isomorfos está em NP.

Exercício 25.E [CLRS 34.2-2]
Mostre que um grafo bipartido com um número ímpar de vértices não é hamiltoniano ($=$ possui um ciclo hamiltoniano).

Exercício 25.F [CLRS 34.2-3]
Mostre que se o problema do Ciclo hamiltoniano está em P, então o problema de listar os vértices de um ciclo hamiltoniano, na ordem em que eles ocorrem no ciclo, pode ser resolvido em tempo polinomial.

Exercício 25.G [CLRS 34.2-5]
Mostre que qualquer problema em NP pode ser resolvido por um algoritmo de consumo de tempo $2^{\mathrm{O}(n^c)}$, onde n é o tamanho da entrada e c é uma constante.

Exercício 25.H [CLRS 34.2-6]
Mostre que o problema do Caminho hamiltoniano está em NP.

Exercício 25.I [CLRS 34.2-7]
Mostre que o problema do caminho hamiltoniano pode ser resolvido em tempo polinomial em grafos orientado acíclicos.

Mais exercícios

Exercício 25.J [CLRS 34.2-8]
Uma fórmula booleana ϕ é uma tautologia se $t(\phi) = \text{VERDADE}$ para toda atribuição de $t : \{\text{variáveis}\} \rightarrow \{\text{VERDADE}, \text{FALSO}\}$. Mostre que o problema de decidir se uma dada fórmula booleana é uma tautologia está em co-NP.

Exercício 25.K [CLRS 34.2-9]
Prove que P $\subseteq$ co-NP.

Exercício 25.L [CLRS 34.2-10]
Prove que se NP $\neq$ co-NP, então P $\neq$ NP.

Exercício 25.M [CLRS 34.2-11]
Se G é um grafo conexo com pelo menos 3 vértices, então G^3 é o grafo que se obtém a partir de G ligando-se por uma aresta todos os pares de vértices que estão conectados em G por um caminho com no máximo três arestas. Mostre que G^3 é hamiltoniano.

Exercício 25.N [CLRS 34.3-2]
Mostre que se $\Pi_1 \leq_P \Pi_2$ e $\Pi_2 \leq_P \Pi_3$, então $\Pi_1 \leq_P \Pi_3$.

Mais exercícios

Exercício 25.O [CLRS 34.3-7]
Suponha que Π e Π' são problemas de decisão sobre o mesmo conjunto de instâncias e que $\Pi(I) =$ SIM se e somente se $\Pi'(I) =$ NÃO. Mostre que Π é NP-completo se e somente se Π' é co-NP-completo.
(Um problema Π' é co-NP-completo se Π' está em co-NP e $\Pi \leq_P \Pi'$ para todo problema Π em co-NP.)

Exercício 25.P [CLRS 34.4-4]
Mostre que o problema de decidir se uma fórmula boolena é uma tautologia é co-NP-completo. (Dica: veja o exercício 25.O.)

Exercício 25.Q [CLRS 34.4-6]
Suponha que ALG$'$ é um algoritmo polinomial para Satisfatibilidade. Descreva um algoritmo polinomial ALG que recebe um fórmula booleana ϕ e devolve uma atribuição t : {variáveis} $\rightarrow$ {VERDADE, FALSO} tal que $t(\phi) =$ VERDADE.

Exercício 25.Q [CLRS 34.5-3]
Prove que o problema Sistemas lineares 0-1 é NP-completo.

Exercício 25.R [CLRS 34.5-6]
Mostre que o problema C aminho hamiltoniano é NP-completo.

Mais um exercício

Exercício 25.S [CLRS 34.5-7]

Mostre que o problema de encontrar um ciclo de comprimento máximo é NP-completo.

Algoritmos de aproximação

CLRS 35

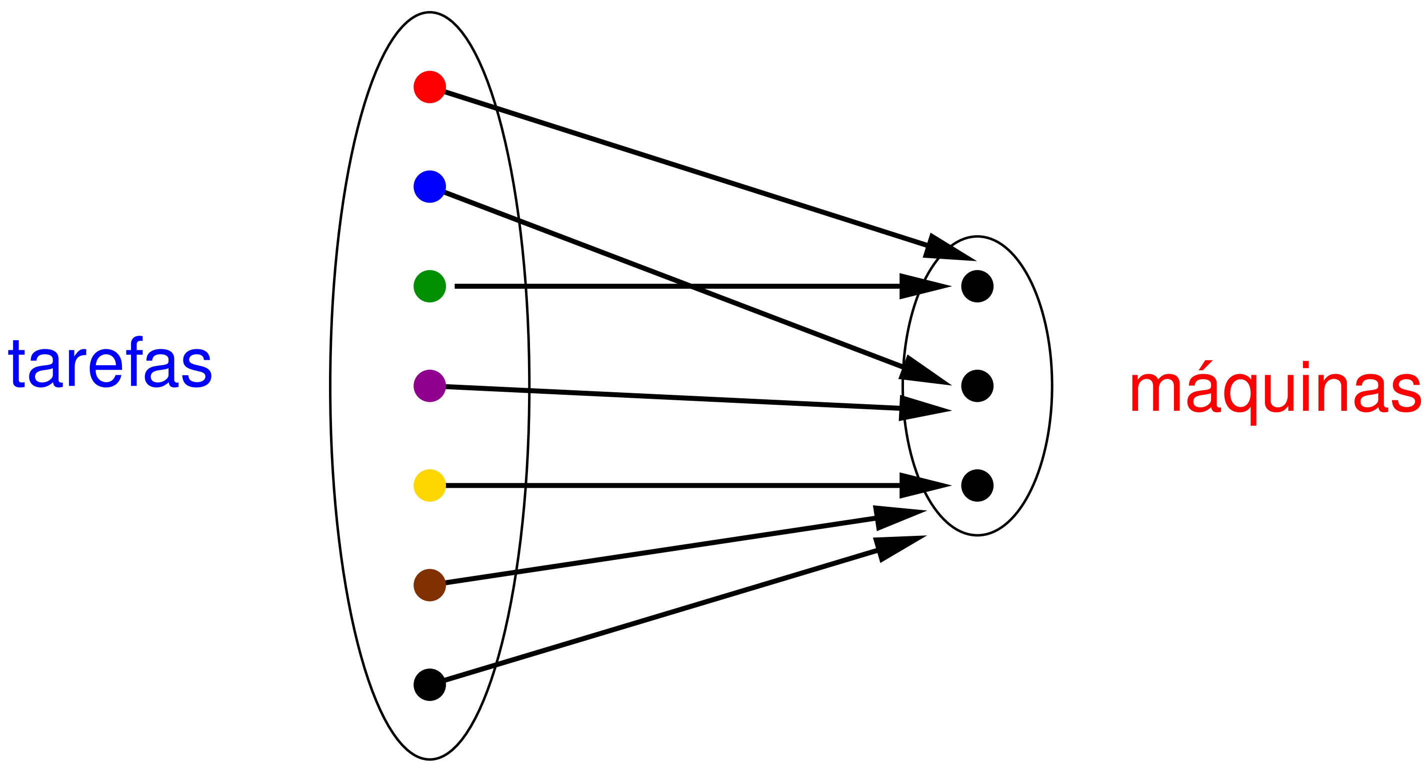
Transparências de Cristina Gomes Fernandes

MAC5727 Algoritmos de Aproximação

Escalonamento de máquinas idênticas

Dados: m máquinas
t tarefas
duração $d[i]$ da tarefa i $(i = 1, \ldots, t)$

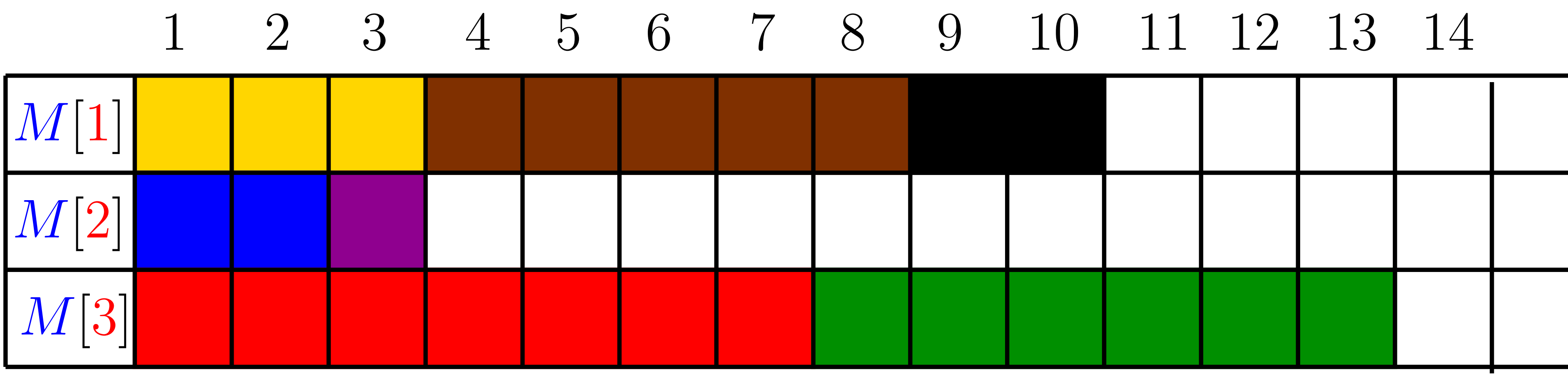
um escalonamento é uma partição $\{M[1], \ldots, M[m]\}$ de $\{1, \ldots, t\}$



Exemplo 1

$m = 3 \quad t = 7$

$d[1]$	$d[2]$	$d[3]$	$d[4]$	$d[5]$	$d[6]$	$d[7]$
3	2	7	5	1	6	2

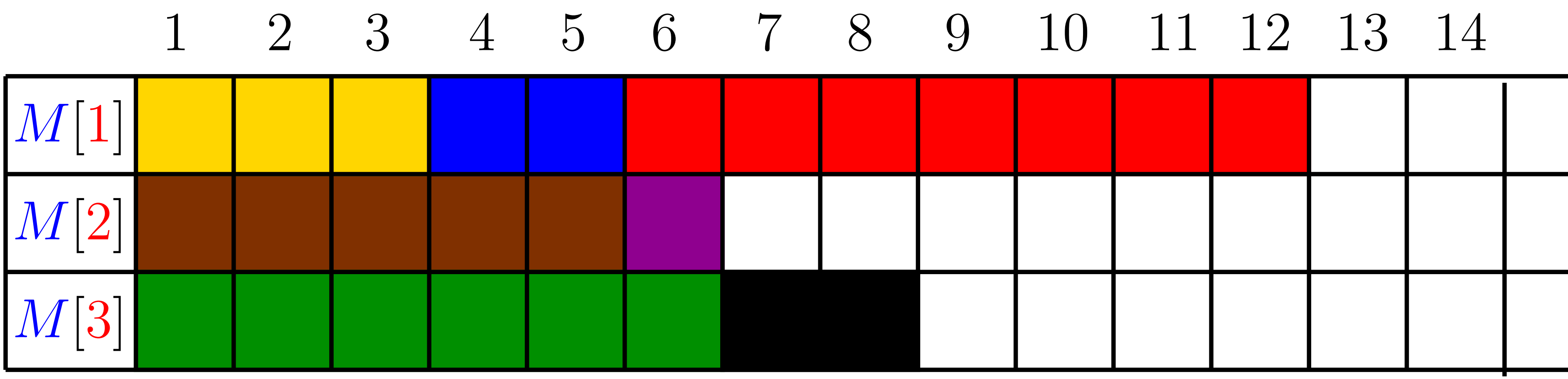


$\{\{1,4,7\},\{2,5\},\{3,6\}\} \Rightarrow$ Tempo de conclusão $= 13$

Exemplo 2

$m = 3 \quad t = 7$

$d[1]$	$d[2]$	$d[3]$	$d[4]$	$d[5]$	$d[6]$	$d[7]$
3	2	7	5	1	6	2

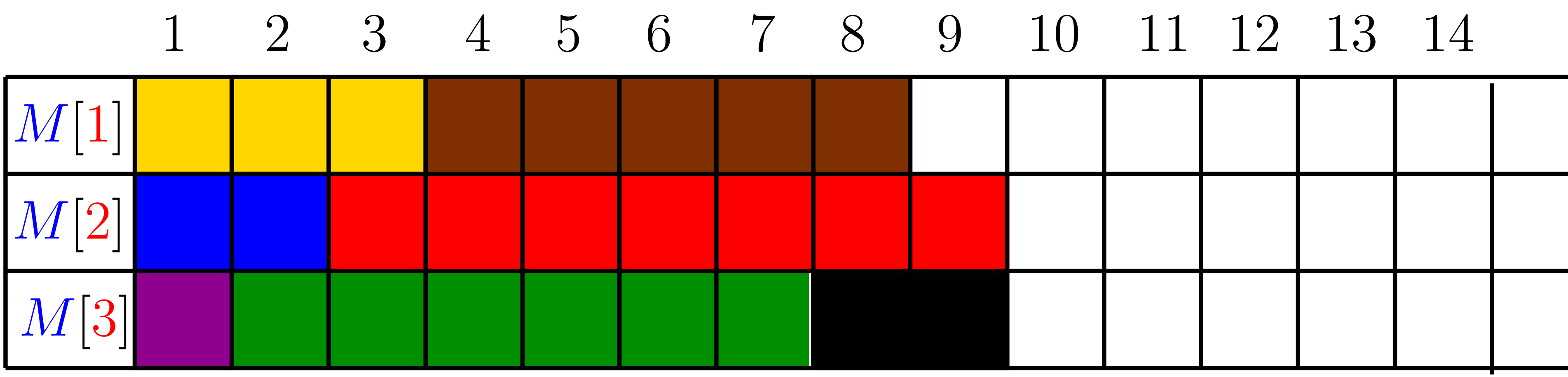


$\{\{1,2,3\},\{4,5\},\{6,7\}\} \Rightarrow$ Tempo de conclusão $= 12$

Problema

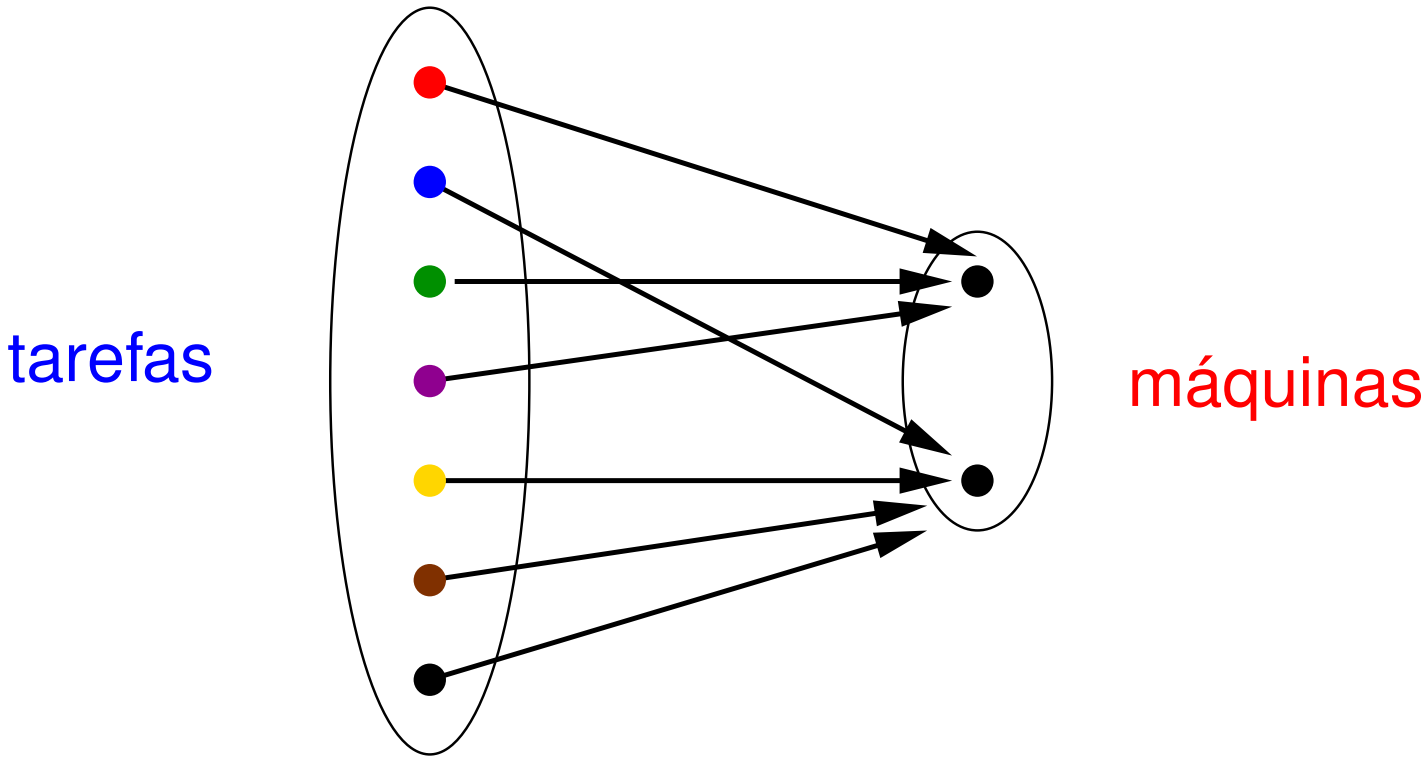
Encontrar um escalonamento com tempo de conclusão mínimo.

$d[1]$	$d[2]$	$d[3]$	$d[4]$	$d[5]$	$d[6]$	$d[7]$
3	2	7	5	1	6	2



$\{\{1,4\},\{2,3\},\{5,6,7\}\} \Rightarrow$ Tempo de conclusão $= 9$

NP-difícil mesmo para $m = 2$



Algoritmo: testa todo $M[1] \subseteq \{1, \ldots, t\}$ e escolhe melhor

2^t subconjuntos $\Rightarrow$ exponencial

NP-difícil $\Rightarrow$ é improvável que exista algoritmo polinomial que resolva o problema (se exitir, P $=$ NP)

Algoritmo de Graham

Atribui, uma a uma, cada tarefa à máquina menos ocupada.

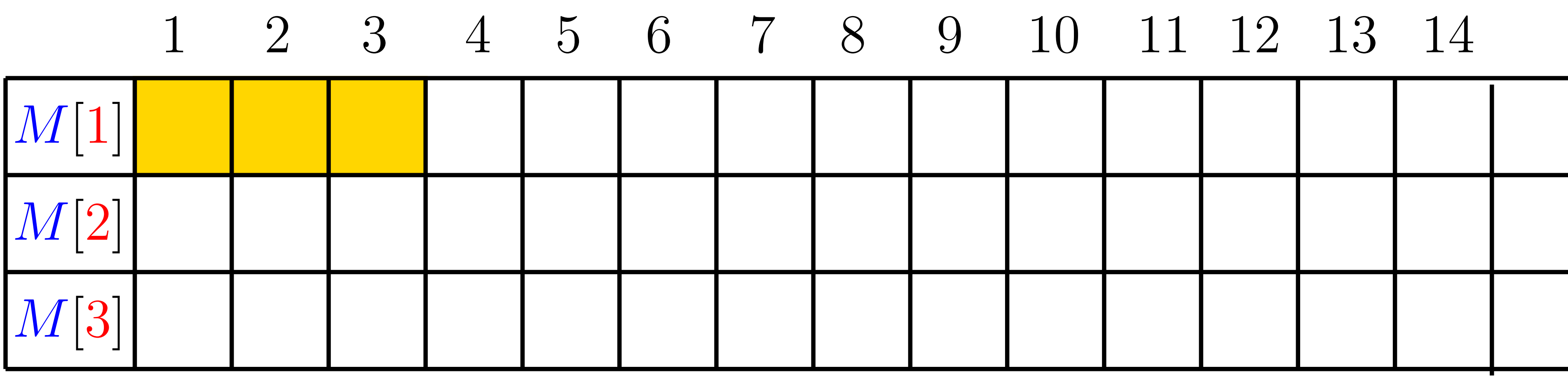
$d[1]$	$d[2]$	$d[3]$	$d[4]$	$d[5]$	$d[6]$	$d[7]$
3	2	7	5	1	6	2

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$M[1]$														
$M[2]$														
$M[3]$														

Algoritmo de Graham

Atribui, uma a uma, cada tarefa à máquina menos ocupada.

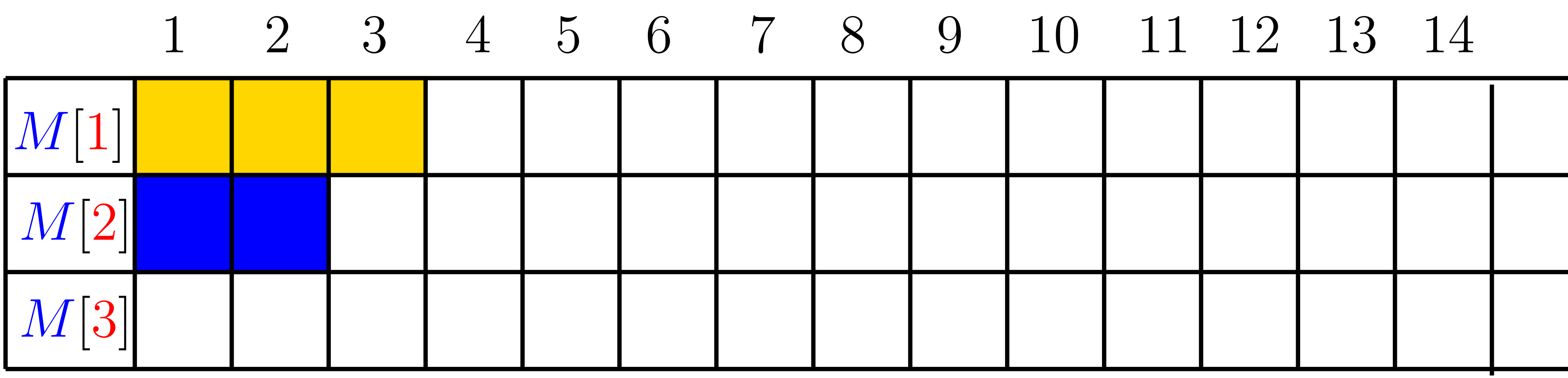
$d[1]$	$d[2]$	$d[3]$	$d[4]$	$d[5]$	$d[6]$	$d[7]$
3	2	7	5	1	6	2



Algoritmo de Graham

Atribui, uma a uma, cada tarefa à máquina menos ocupada.

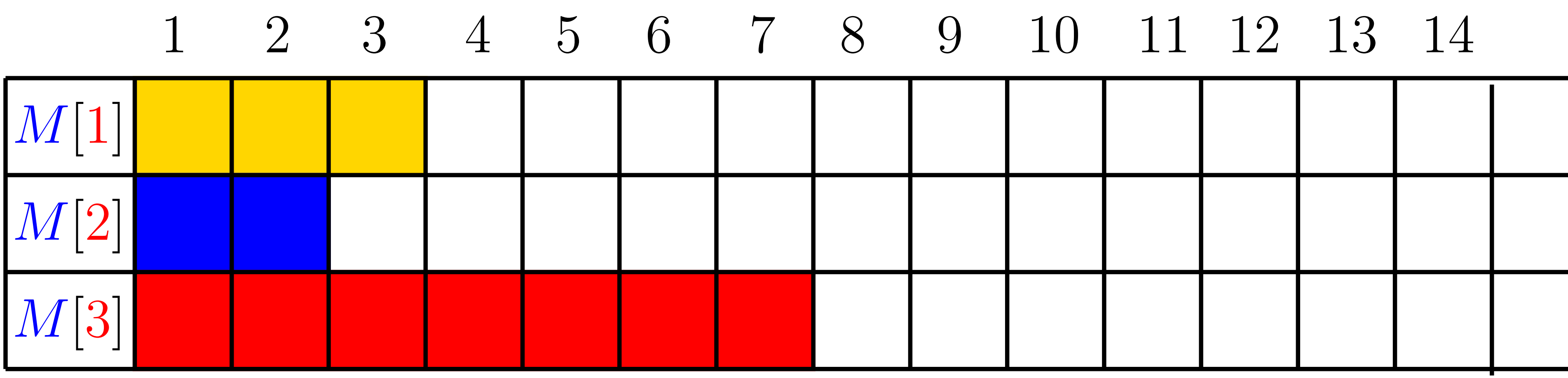
$d[1]$	$d[2]$	$d[3]$	$d[4]$	$d[5]$	$d[6]$	$d[7]$
3	2	7	5	1	6	2



Algoritmo de Graham

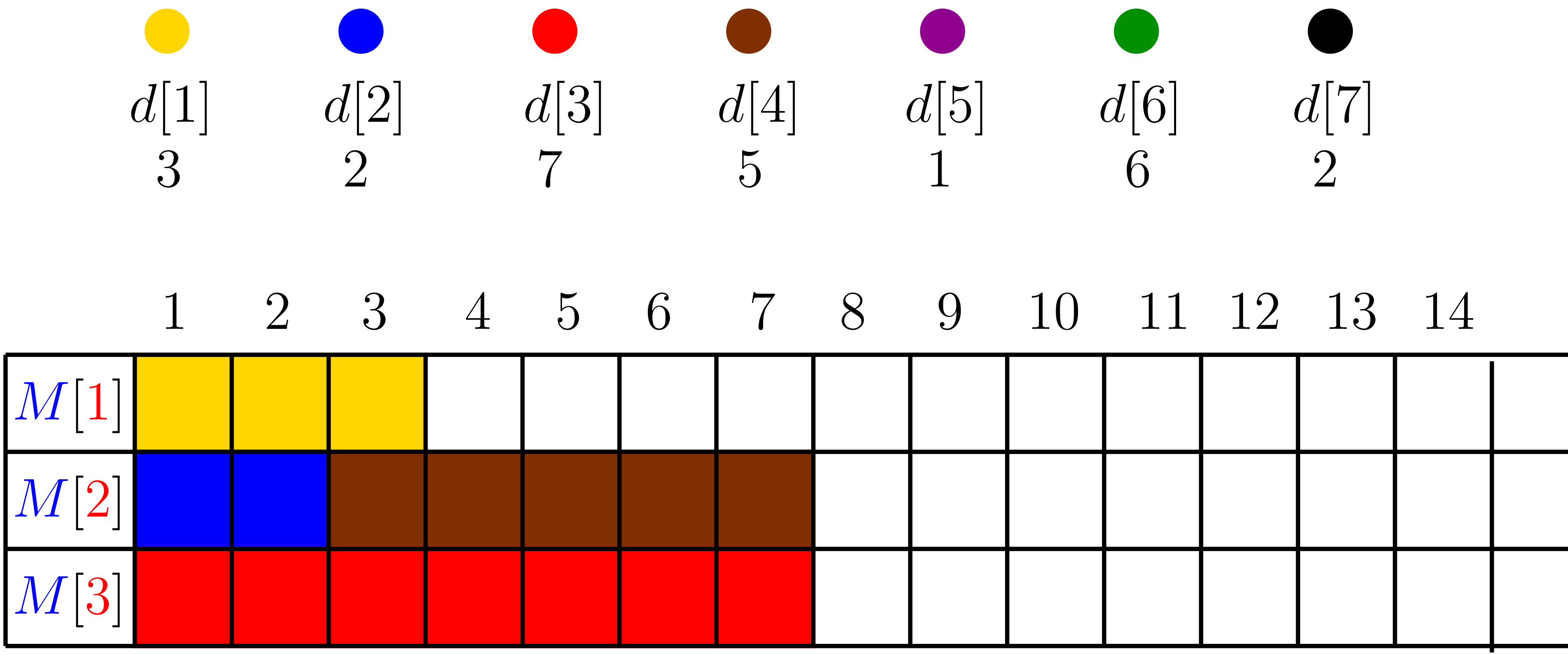
Atribui, uma a uma, cada tarefa à máquina menos ocupada.

$d[1]$	$d[2]$	$d[3]$	$d[4]$	$d[5]$	$d[6]$	$d[7]$
3	2	7	5	1	6	2



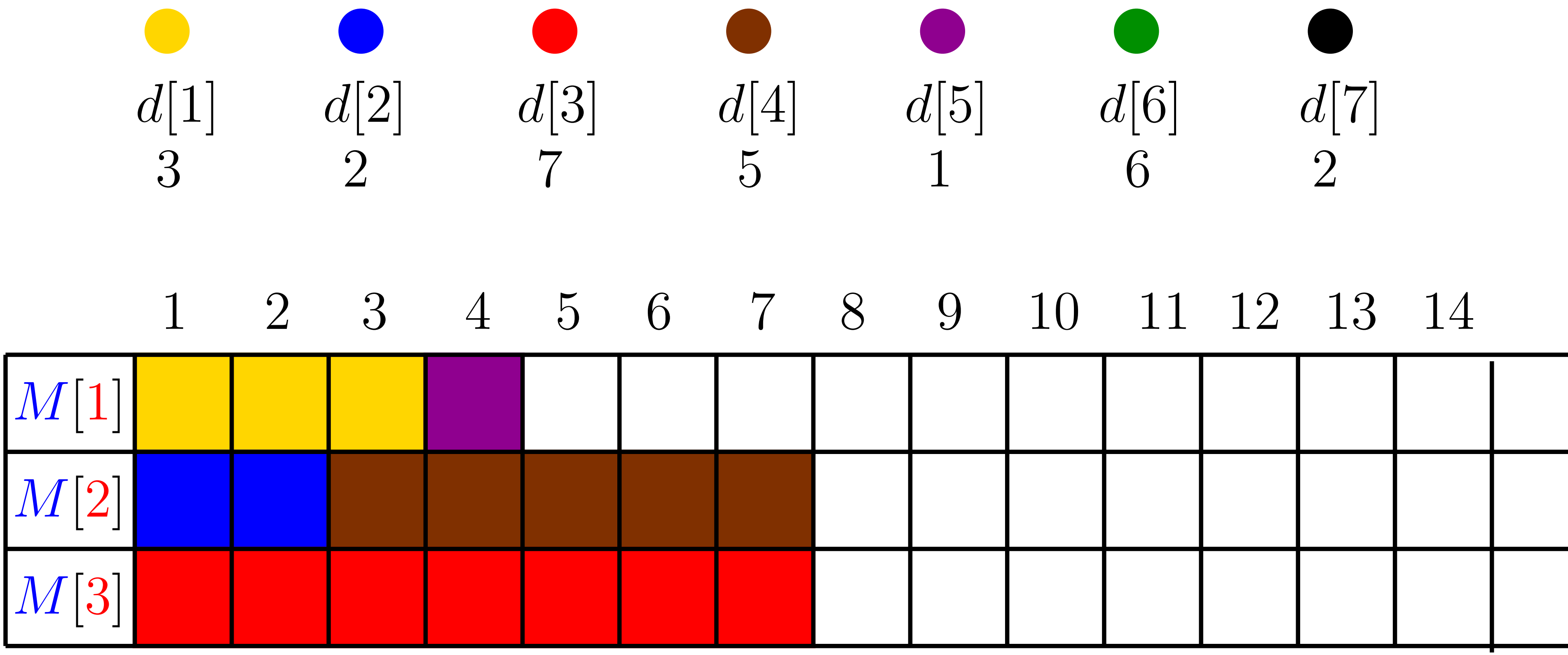
Algoritmo de Graham

Atribui, uma a uma, cada tarefa à máquina menos ocupada.



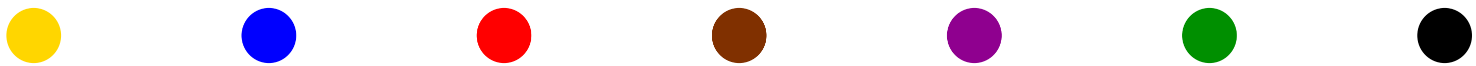
Algoritmo de Graham

Atribui, uma a uma, cada tarefa à máquina menos ocupada.

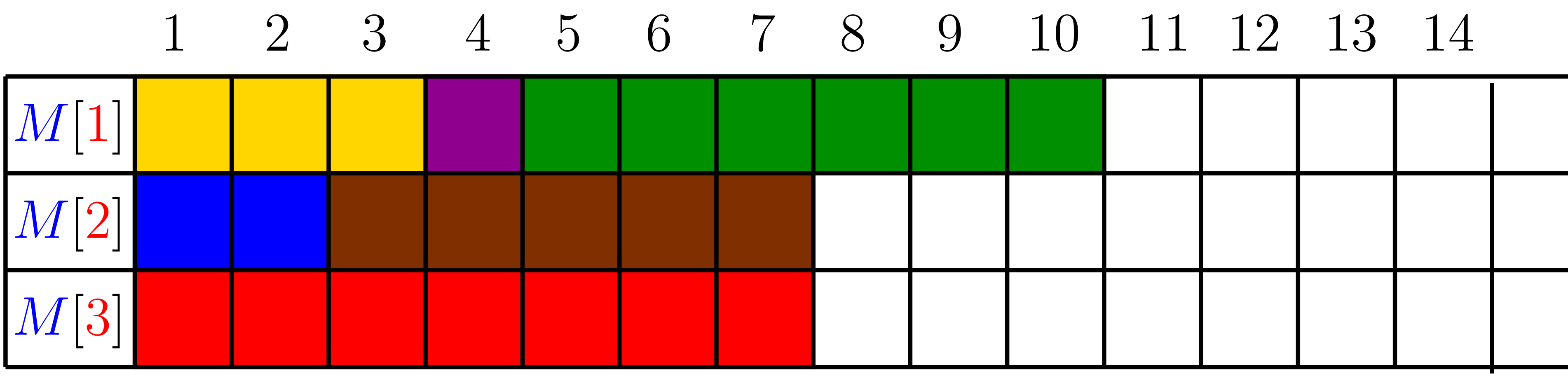


Algoritmo de Graham

Atribui, uma a uma, cada tarefa à máquina menos ocupada.

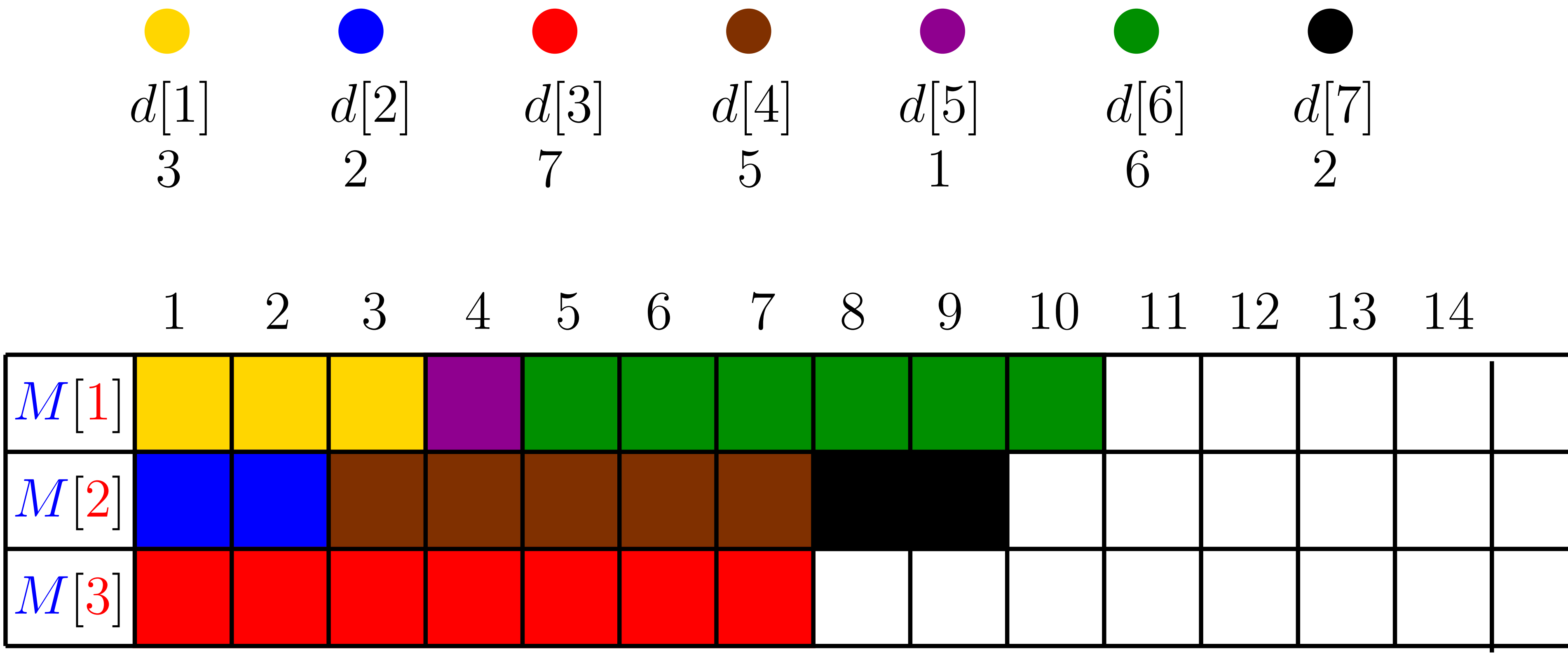


$d[1]$	$d[2]$	$d[3]$	$d[4]$	$d[5]$	$d[6]$	$d[7]$
3	2	7	5	1	6	2



Algoritmo de Graham

Atribui, uma a uma, cada tarefa à máquina menos ocupada.



Escalonamento-Graham

Recebe números inteiros positivos m e t e um vetor $d[1\,..\,t]$ e devolve um escalomento de $\{1, \ldots, t\}$ em m máquinas.

ESCALONAMENTO-GRAHAM (m, t, d)

para $j \leftarrow 1$ **até** m **faça**
    $M[j] \leftarrow \emptyset$
    $T[j] \leftarrow 0$
para $i \leftarrow 1$ **até** t **faça**
    seja k tal que $T[k]$ é mínimo
    $M[k] \leftarrow M[k] \cup \{i\}$
    $T[k] \leftarrow T[k] + d[i]$
devolva $\{M[1], \ldots, M[m]\}$

Consumo de tempo

linha	consumo de todas as execuções da linha
1	$\Theta(m)$
2	$\Theta(m)$
3	$\Theta(m)$
4	$\Theta(t)$
5	$\Theta(mt)$
6	$\Theta(t)$
7	$\Theta(t)$
8	$\Theta(t)$
total	$3\Theta(m) + \Theta(mt) + 3\Theta(t) = \Theta(mt)$

Escalonamento-Graham

Se utilizarmos uma fila com prioridades (min-heap) para representar as m máquinas onde a prioridade da máquina i é $T[i]$ teremos:

ESCALONAMENTO-GRAHAM (m, t, d)

para $j \leftarrow 1$ **até** m **faça**
    $M[j] \leftarrow \emptyset$
    $T[j] \leftarrow 0$
BUILD-MIN-HEAP (T, m)
para $i \leftarrow 1$ **até** t **faça**
    $k \leftarrow$ HEAP-MIN (T, m)
    $M[k] \leftarrow M[k] \cup \{i\}$
    HEAP-INCREASE-KEY $(T, k, T[k] + d[i])$
devolva $\{M[1], \ldots, M[m]\}$

Consumo de tempo

linha	consumo de todas as execuções da linha
1-4	$\Theta(m)$
5	$\Theta(t)$
6	$\Theta(t)$
7	$\Theta(t)$
8	$\mathrm{O}(t \lg m)$
9	$\Theta(t)$
total	$\Theta(m) + 4\Theta(t) + \mathrm{O}(t \lg m) = \mathrm{O}(m + t \lg m)$

O consumo de tempo do algoritmo ESCALONAMENTO-GRAHAM é $\mathrm{O}(m + t \lg m)$.

Delimitações para OPT

OPT = menor tempo de conclusão de um escalonamento

- Duração da tarefa mais longa:

$$\mathrm{OPT} \geq \max\{d[1], d[2], \ldots, d[t]\}$$

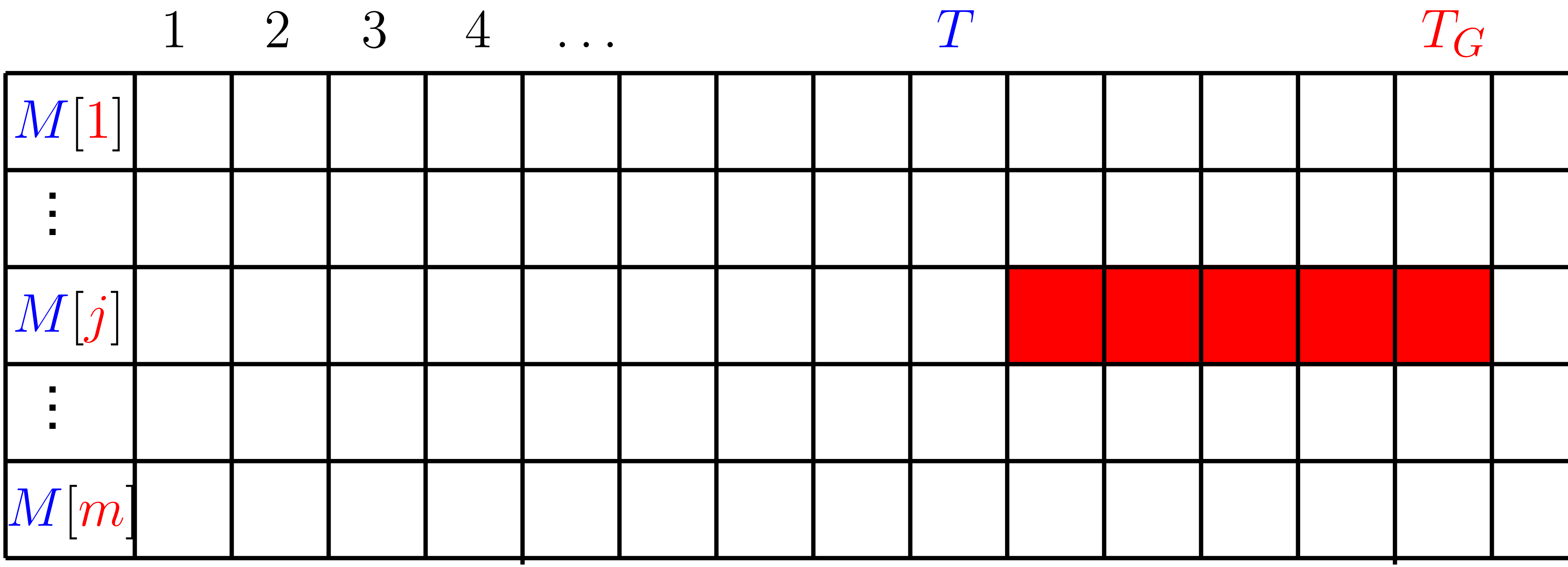
- Distribuição balanceada:

$$\mathrm{OPT} \geq \frac{d[1] + d[2] + \cdots + d[t]}{m}$$

Fator de aproximação

tarefa t é executada na máquina j a partir do instante T

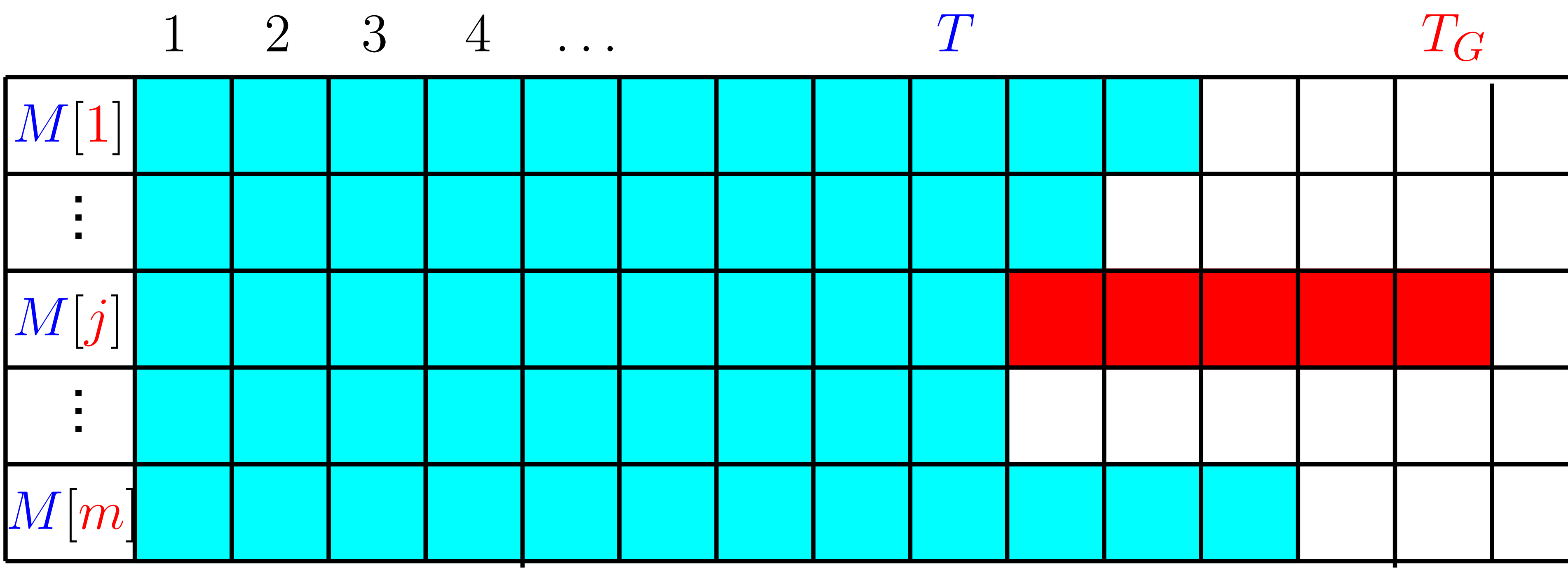
$T_G =$ conclusão do escalonamento obtido pelo algoritmo



Fator de aproximação (cont.)

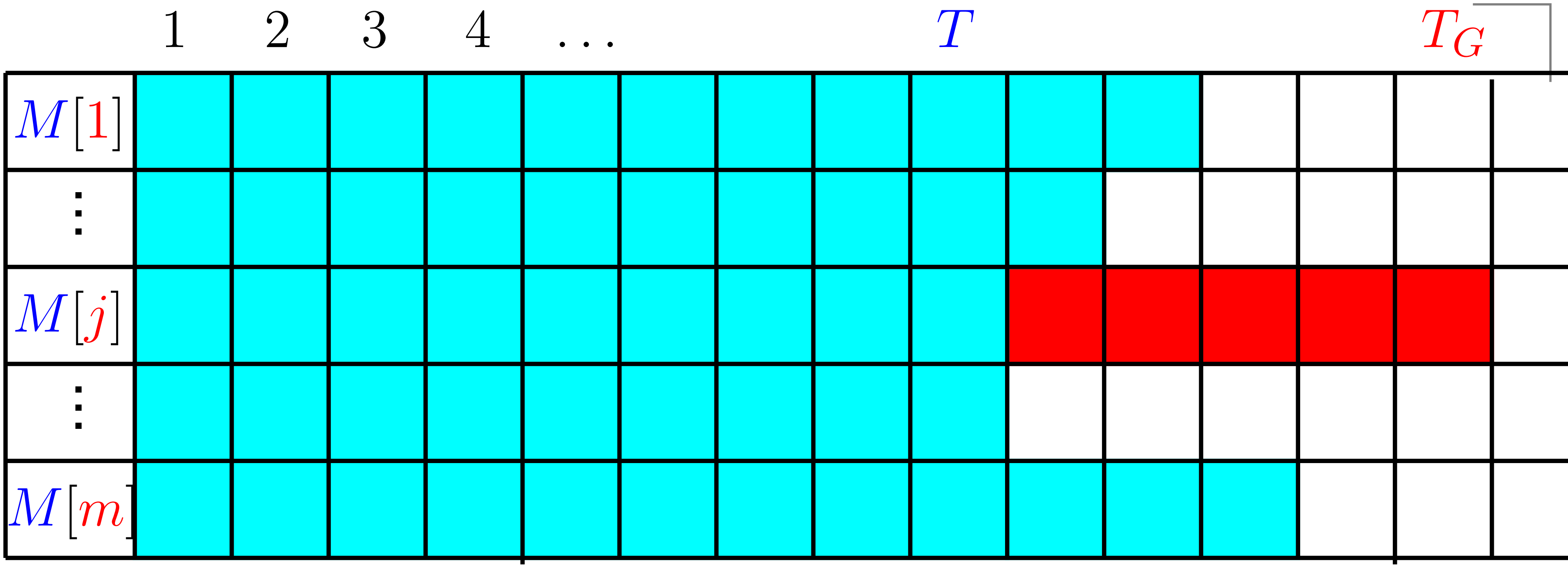
tarefa t é executada na máquina j a partir do instante T

$T_G =$ conclusão do escalonamento obtido pelo algoritmo



$$T \cdot m \leq d[1] + \cdots + d[t] \quad \Rightarrow \quad T \leq \frac{d[1] + \cdots + d[t]}{m}$$

Fator de aproximação (cont.)



$$
\begin{aligned}
T_G &= T + d[t] \\
&\leq \frac{d[1] + \cdots + d[t]}{m} + d[t] \\
&\leq \frac{d[1] + \cdots + d[t]}{m} + \max\{d[1], \ldots, d[t]\} \\
&\leq \mathrm{OPT} + \mathrm{OPT} = 2\,\mathrm{OPT}
\end{aligned}
$$

Algoritmos de aproximação

Algoritmo ALG é de aproximação se existe $\alpha > 0$ tal que

$$\text{valor de ALG}(I) \leq \alpha \cdot \mathrm{OPT}\,(I)$$

para toda instância I do problema.

α é o fator de aproximação

Objetivo: α tão perto de 1 quanto possível

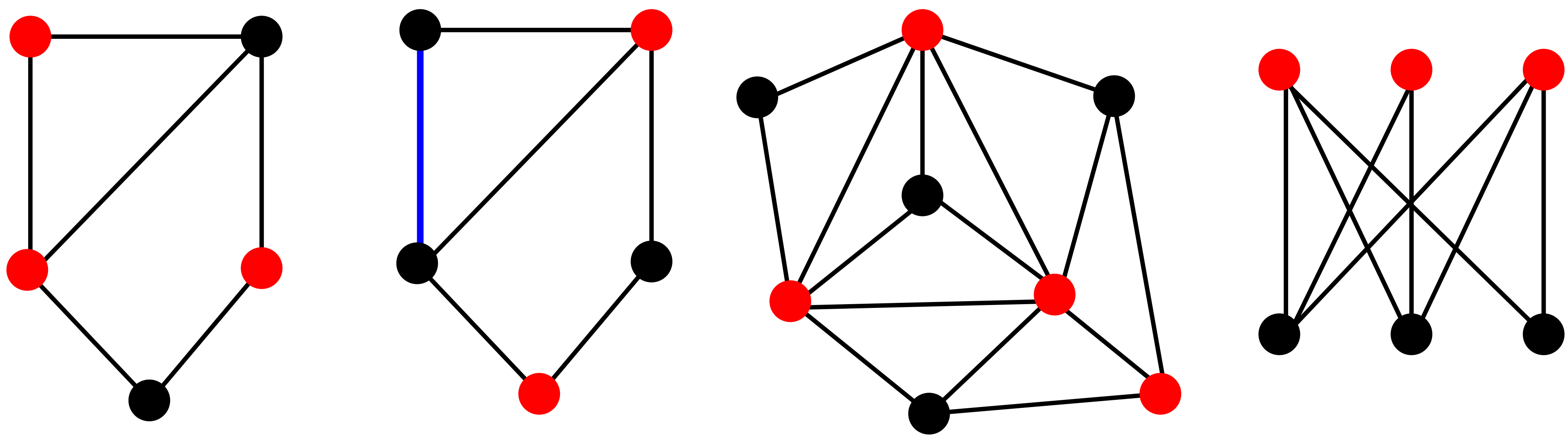
Conclusão

O algoritmo ESCALONAMENTO-GRAHAM é uma 2-aproximação polinomial.

Cobertura por vértices

Um conjunto de vértices C é uma cobertura por vértices de G se cada aresta tem pelo menos uma ponta em C

Exemplos:



não é

Cobertura por vértices

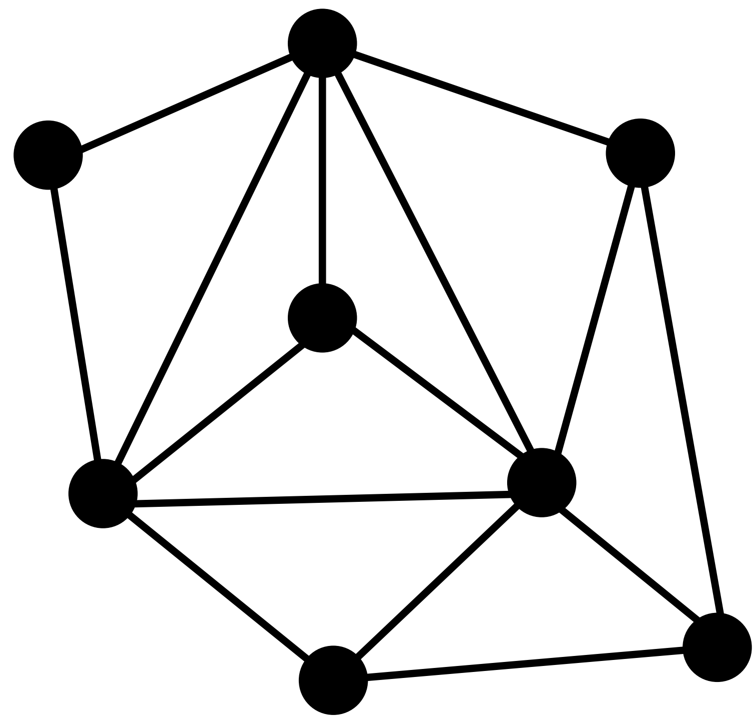
Problema: Dado um grafo G, encontrar um cobertura mínima.

Problema é NP-difícil.

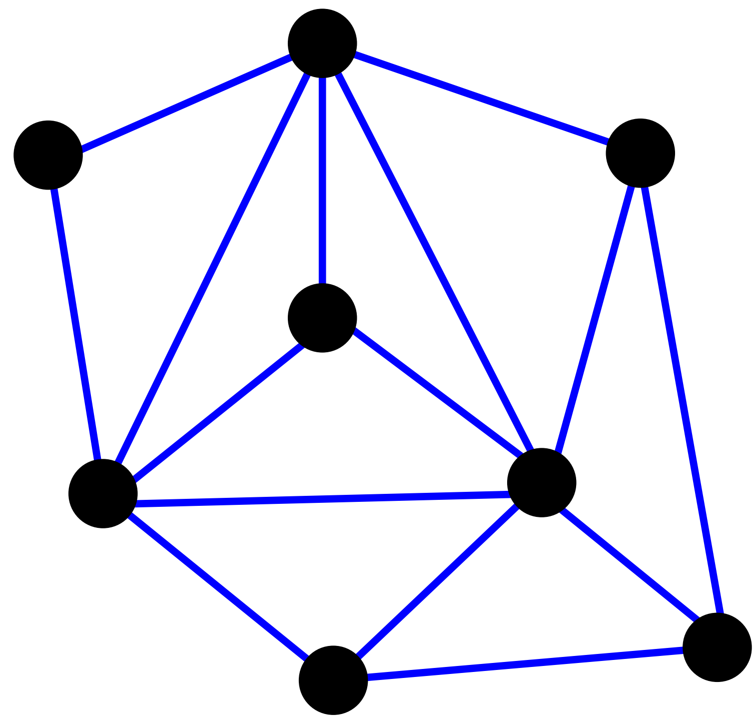
APPROX-VERTEX-COVER (G)

$C \leftarrow \emptyset$
$E \leftarrow$ {arestas de G}
enquanto $E \neq \emptyset$ **faça**
    seja (u, v) uma aresta qualquer em E
    $C \leftarrow C \cup \{u, v\}$
    $E \leftarrow E -$ {arestas incidentes a u ou v}
devolva C

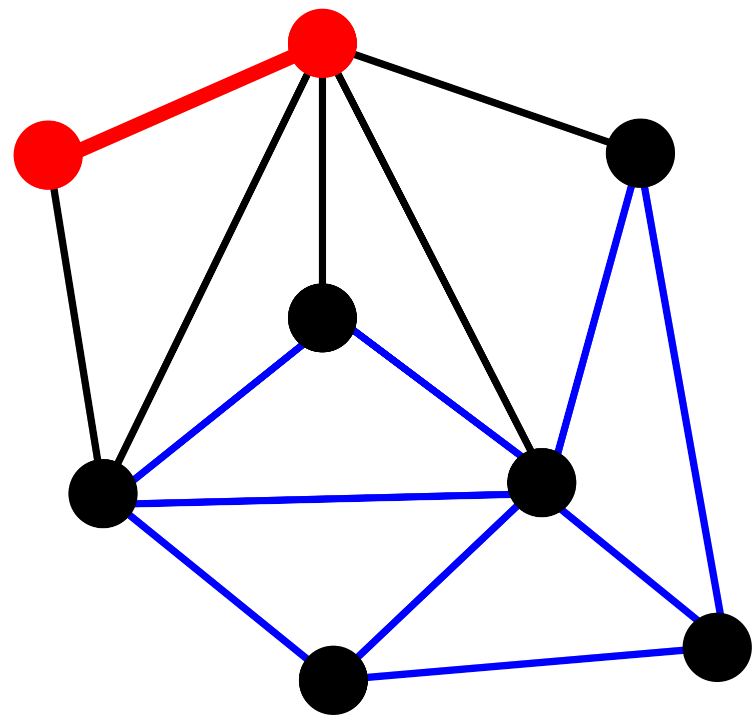
Exemplo



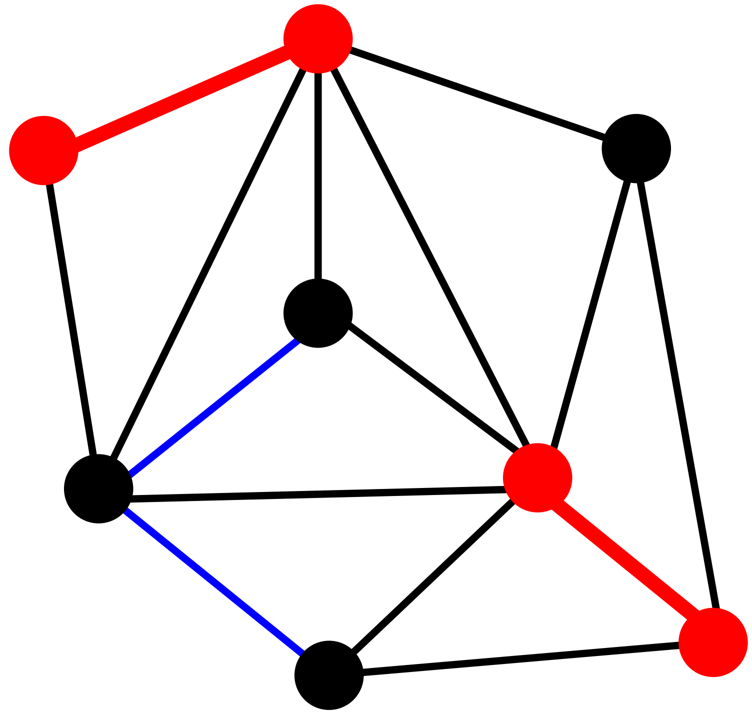
Exemplo



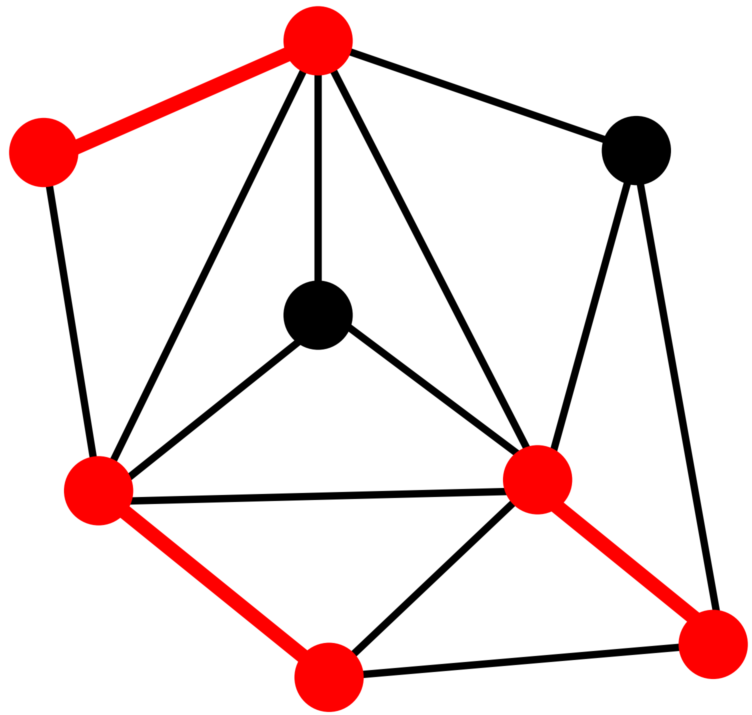
Exemplo



Exemplo



Exemplo

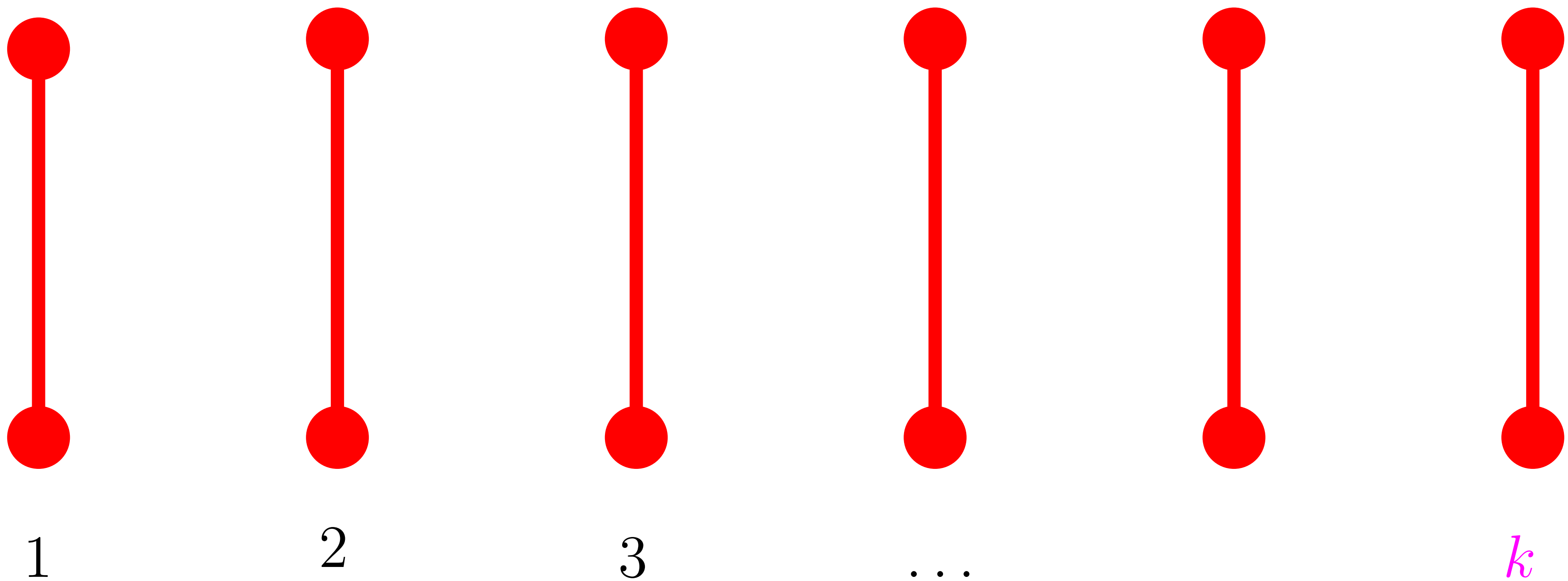


Consumo de tempo

E representado através de listas de adjacência.
n = número de vértices
m = número de arestas

linha	consumo de todas as execuções da linha
1	$\Theta(1)$
2	$\Theta(n+m)$
3	$O(m)$
4	$O(m)$
5	$O(n)$
6	$O(m)$
7	$O(n)$
total	$\Theta(1) + \Theta(n+m) + 3O(m) + 2O(n) = \Theta(n+m)$

Delimitações para OPT

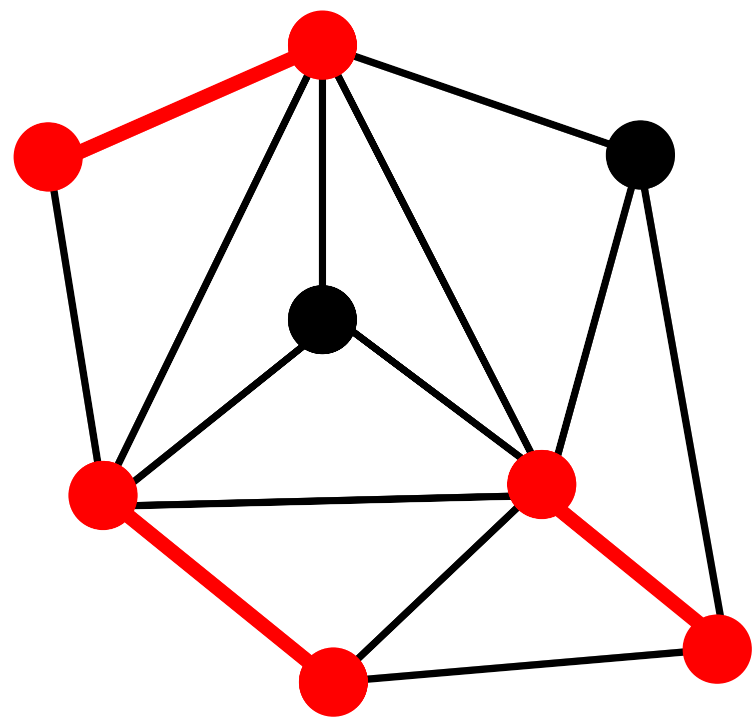


Se G possui um emparelhamento com k arestas, então

$$\mathrm{OPT} \geq k.$$

$\mathrm{OPT} =$ número mínimo de vértices de uma cobertura por vértices

Fator de aproximação



C = cobertura devolvida pelo algoritmo.
G possui um emparelhamento com $|C|/2$ arestas.
Logo,

$$\frac{|C|}{2} \leq \mathrm{OPT} \quad \Rightarrow \quad |C| \leq 2\,\mathrm{OPT}.$$

Conclusão

O algoritmo APPROX-VERTEX-COVER é uma 2-aproximação polinomial.