# Melhores momentos

AULA 25

# Verificador polinomial para SIM

Um verificador polinomial para a resposta SIM a um problema $\Pi$ é uma algoritmo polinomial ALG que recebe

uma instância $I$ de $\Pi$ 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!

# 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 |

# Redução polinomial

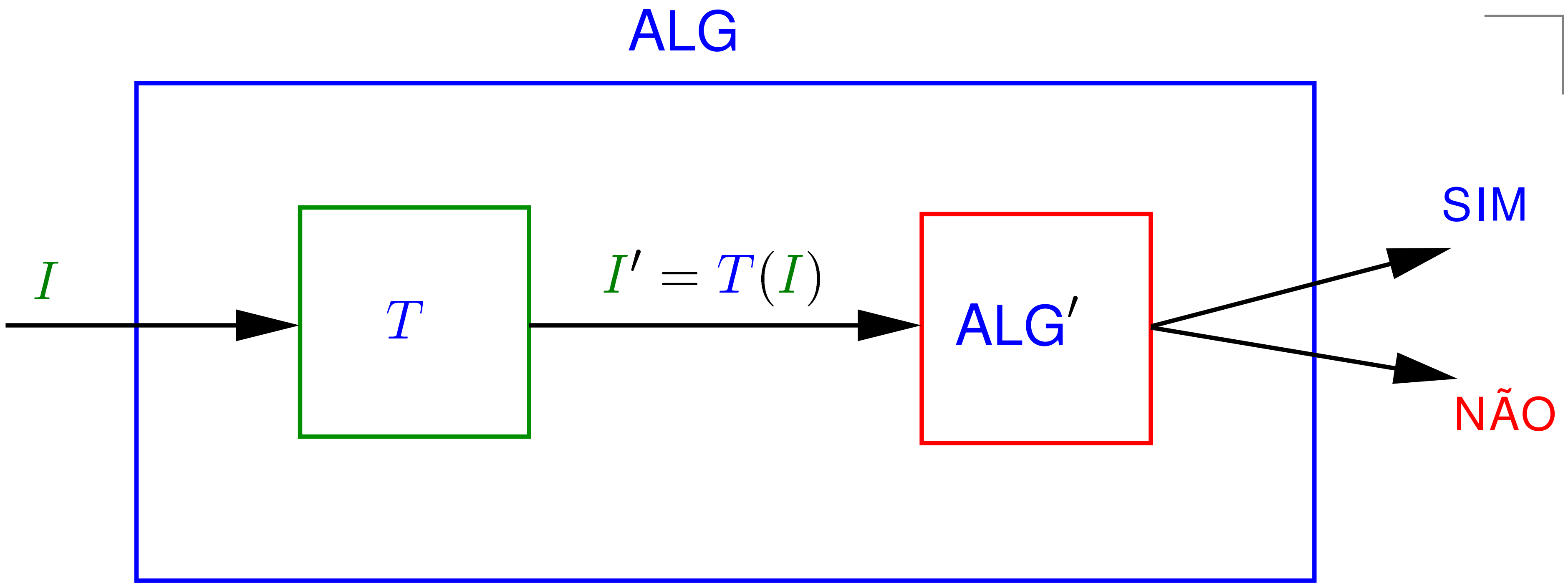
Permite comparar o “grau de complexidade” de problemas diferentes.

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

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

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

# Esquema comum de redução




Faz apenas uma chamada ao algoritmo ALG$'$.

$T$ transforma uma instância $I$ de $\Pi$ em uma instância $I' = T(I)$ de $\Pi'$ 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

Caminho hamiltoniano $\leq_P$ Satisfatibilidade

Satisfatibilidade $\leq_P$ 3-Satisfatibilidade

Hoje: 3-Satisfatibilidade $\leq_P$ Clique

# Problemas completos em NP

Um problema $\Pi$ em NP é NP-completo se cada problema em NP pode ser reduzido a $\Pi$.

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

Se $\Pi \leq_P \Pi'$ e $\Pi$ é NP-completo, então $\Pi'$ 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 $\Pi'$ é NP-completo podemos utilizar o Teorema de Cook e Levin.

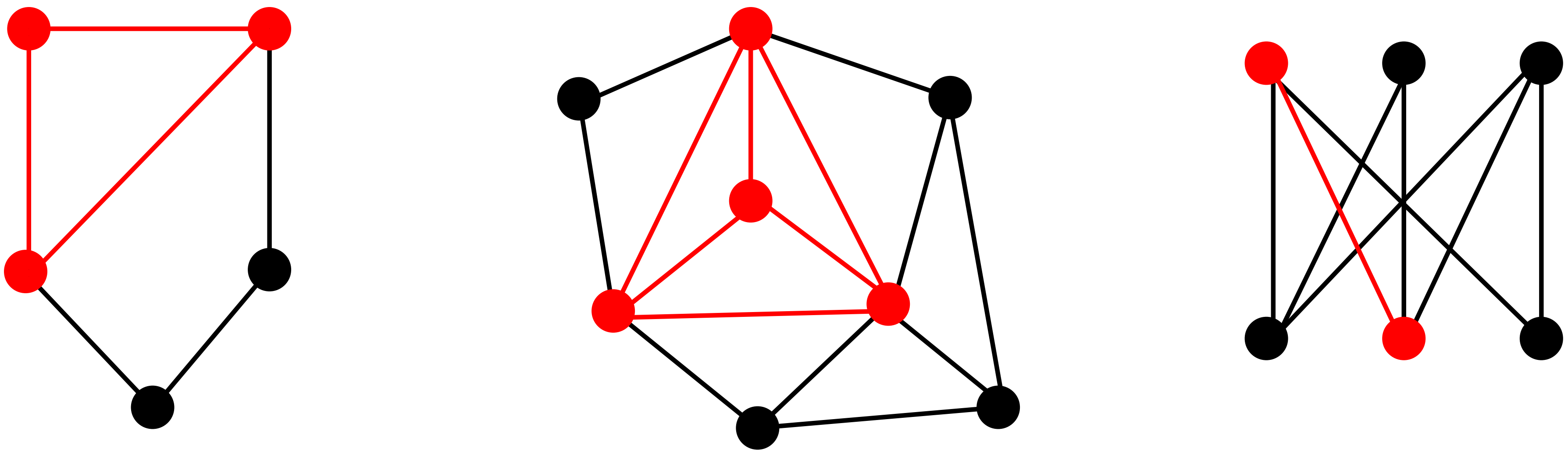
Para isto devemos:

- Demonstrar que $\Pi'$ está en NP.
- Escolher um problema $\Pi$ 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 $\phi$ com $k$ claúsulas e exatamente 3 literais por claúsula e devolve um grafo $G$ tais que

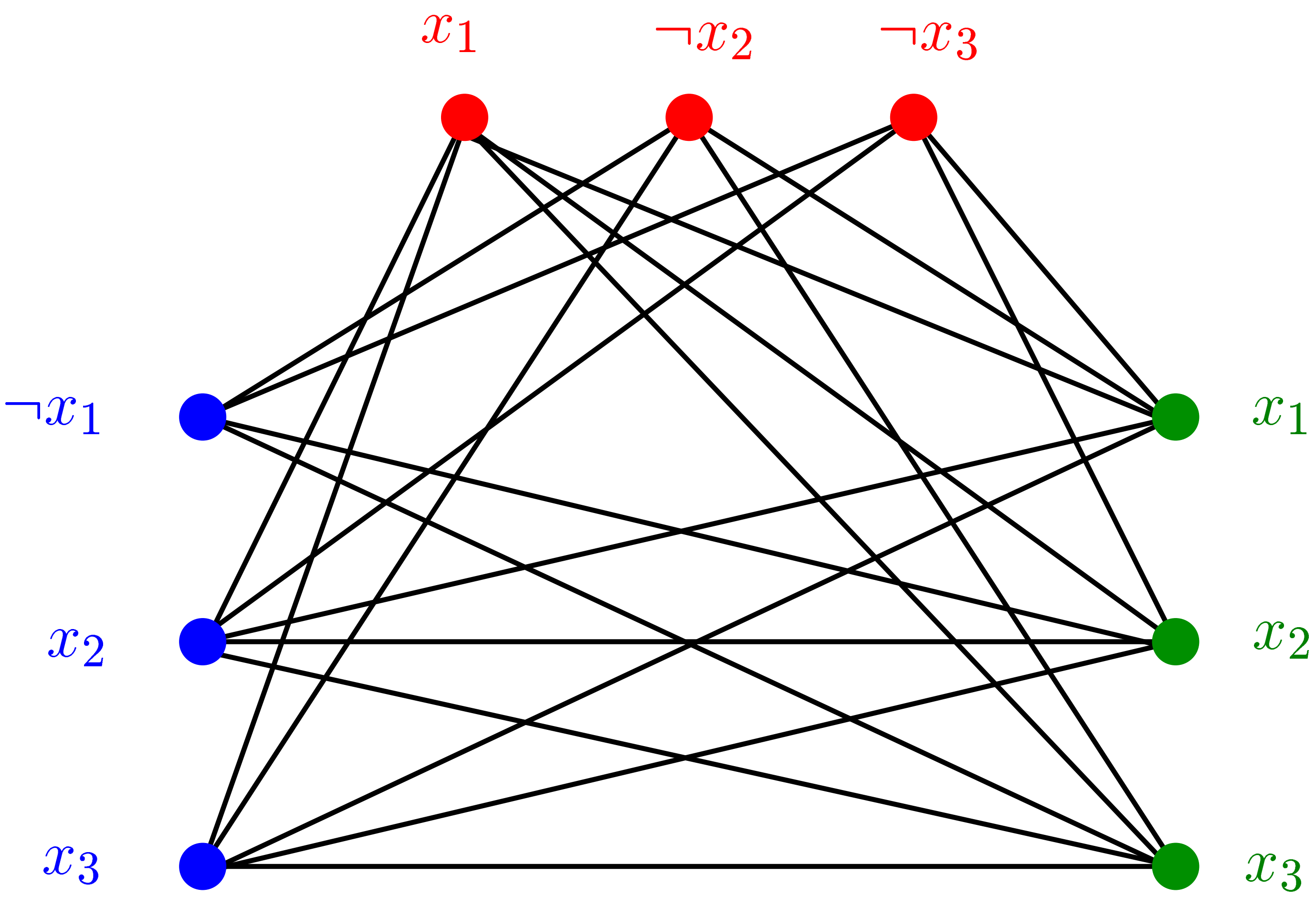
$$\phi \text{ é satisfatível} \Leftrightarrow G \text{ possui um clique} \geq k.$$

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.)

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




Verifique: $\phi$ é satisfatível $\Leftrightarrow$ $G$ possui um clique $\geq k$.

# Problemas NP-difíceis

Um problema $\Pi$, não necessariamente em NP, é NP-díficil se a existência de um algoritmo polinomial para $\Pi$ 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.

# AULA 26

# Algoritmos de aproximação

CLRS 35

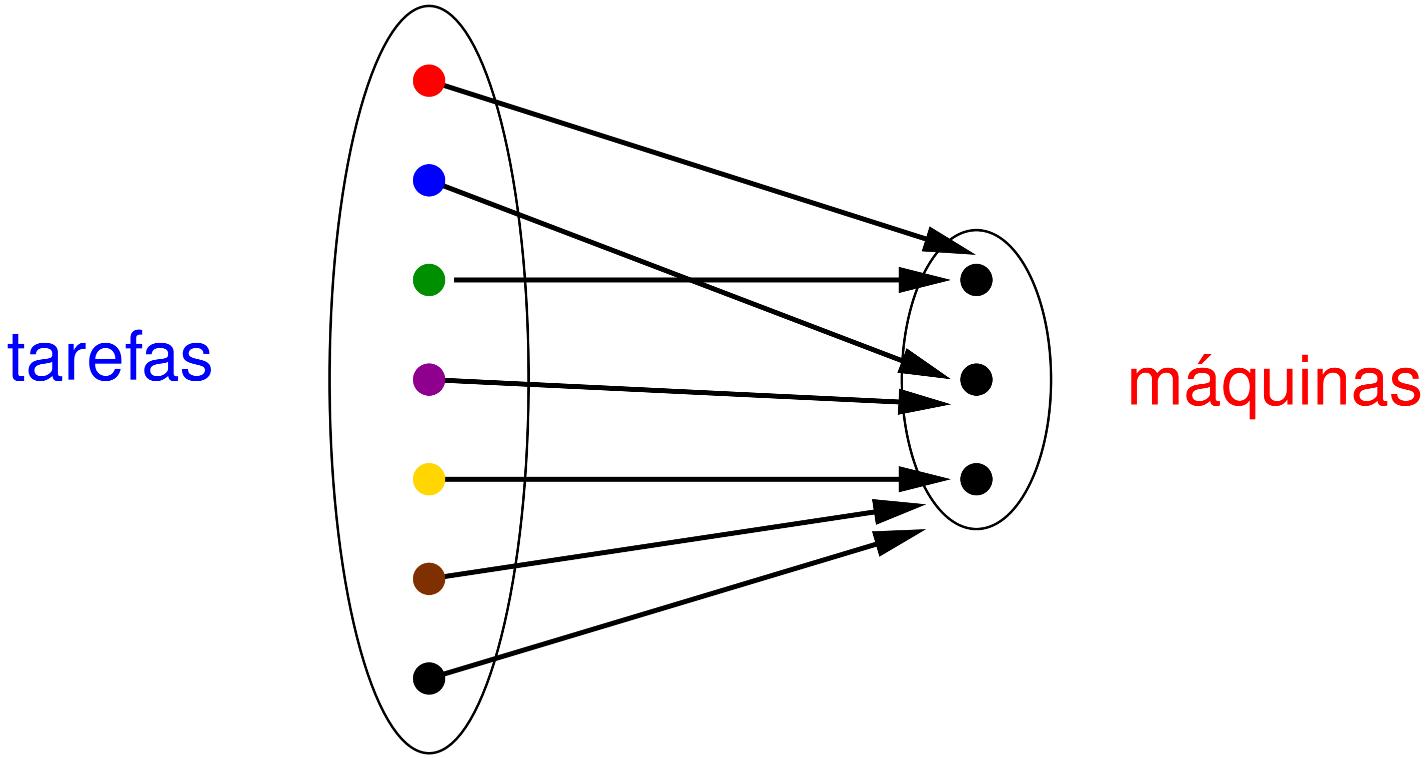
Transparências de Cristina Gomes Fernandes

MAC5727 Algoritmos de Aproximação

# Escalonamento de máquinas idênticas

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

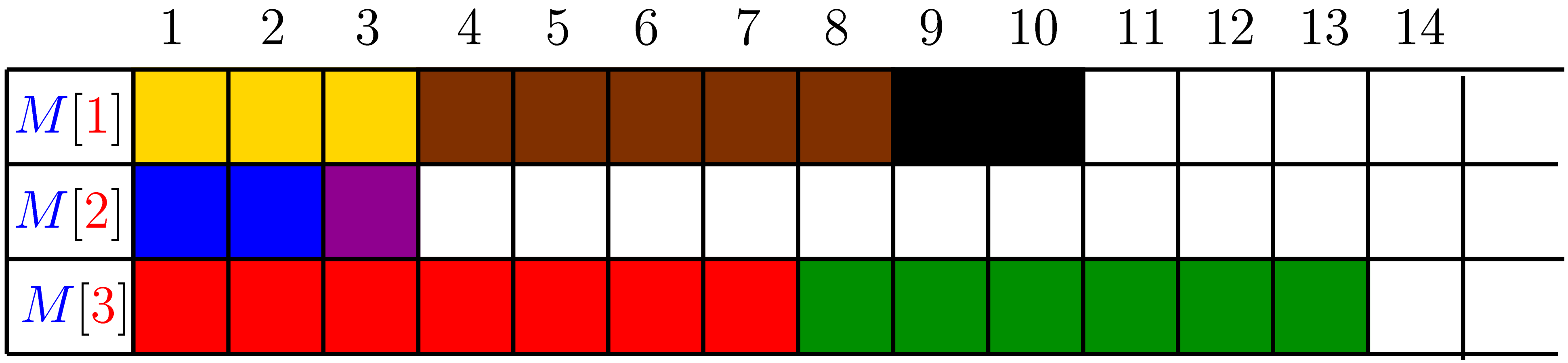
um escalonamento é uma partição $\{M[1], \ldots, M[m]\}$
de $\{1, \ldots, t\}$



# Exemplo 1

$m = 3 \qquad t = 7$

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

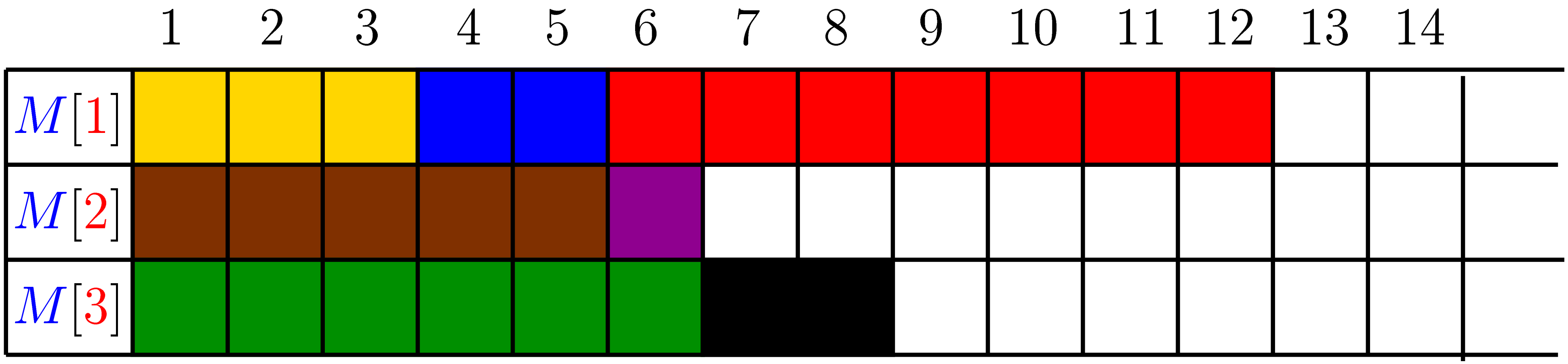



$\{\{1,4,7\},\{2,5\},\{3,6\}\} \Rightarrow$ Tempo de conclusão $= 13$

# Exemplo 2

$m = 3 \qquad t = 7$

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

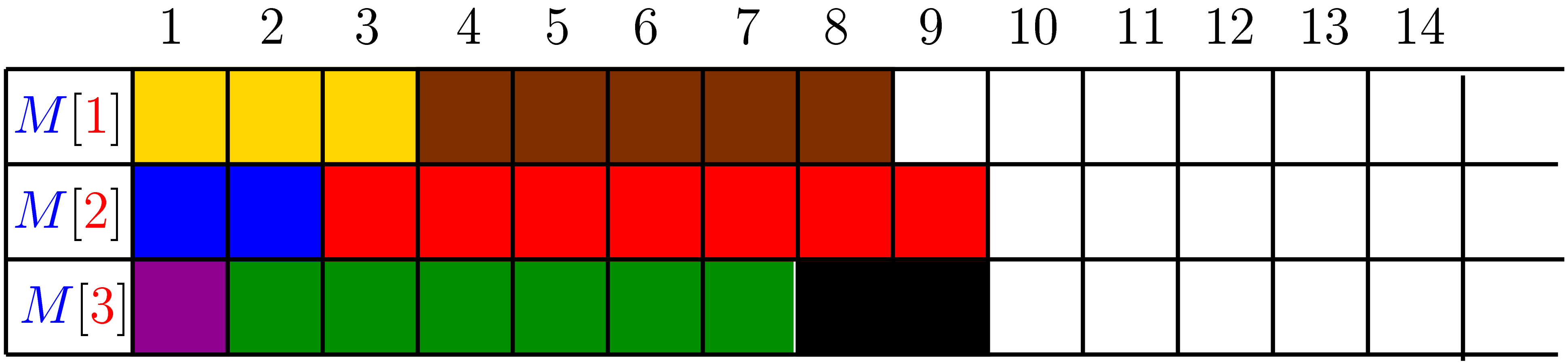



$\{\{1,2,3\},\{4,5\},\{6,7\}\} \Rightarrow$ Tempo de conclusão $= 12$

# Problema

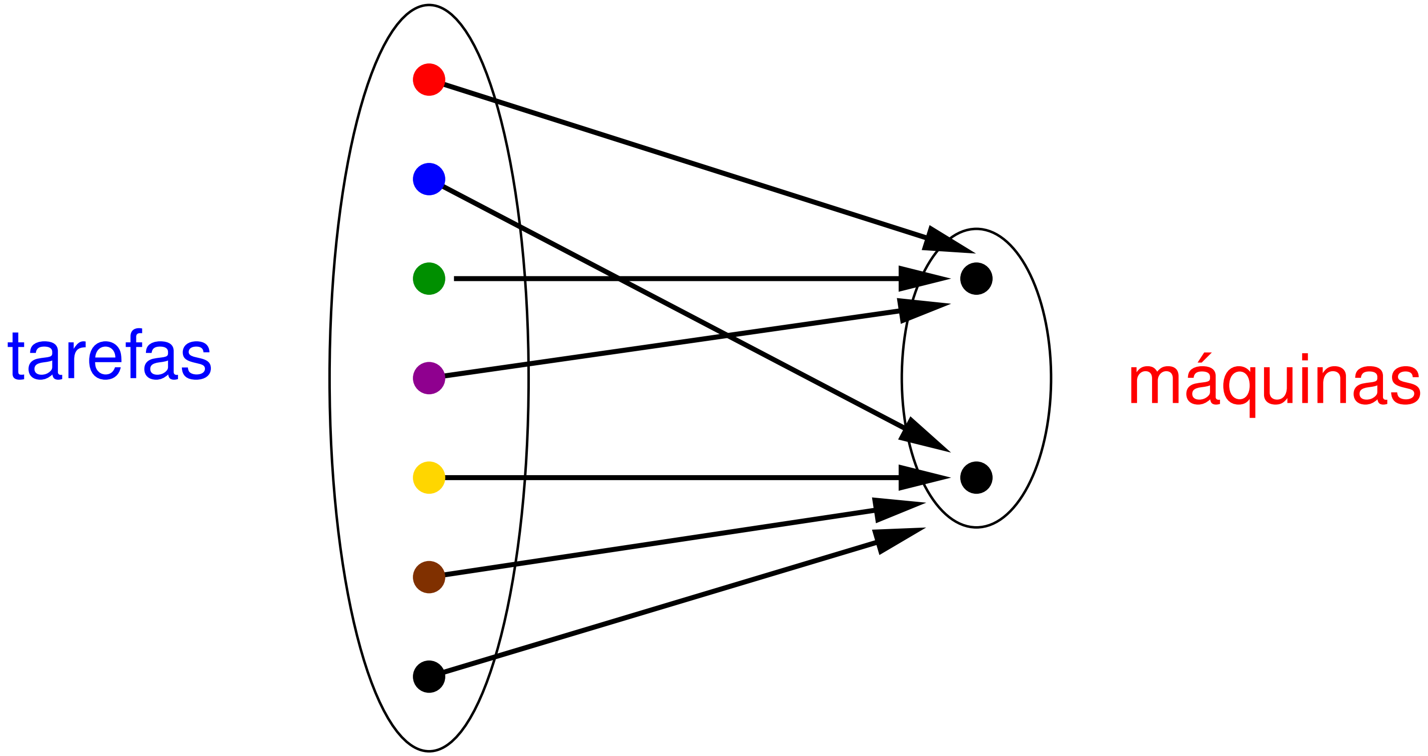
Encontrar um escalonamento com tempo de conclusão mínimo.

| $d[1]$ | $d[2]$ | $d[3]$ | $d[4]$ | $d[5]$ | $d[6]$ | $d[7]$ |
|---|---|---|---|---|---|---|
| 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.

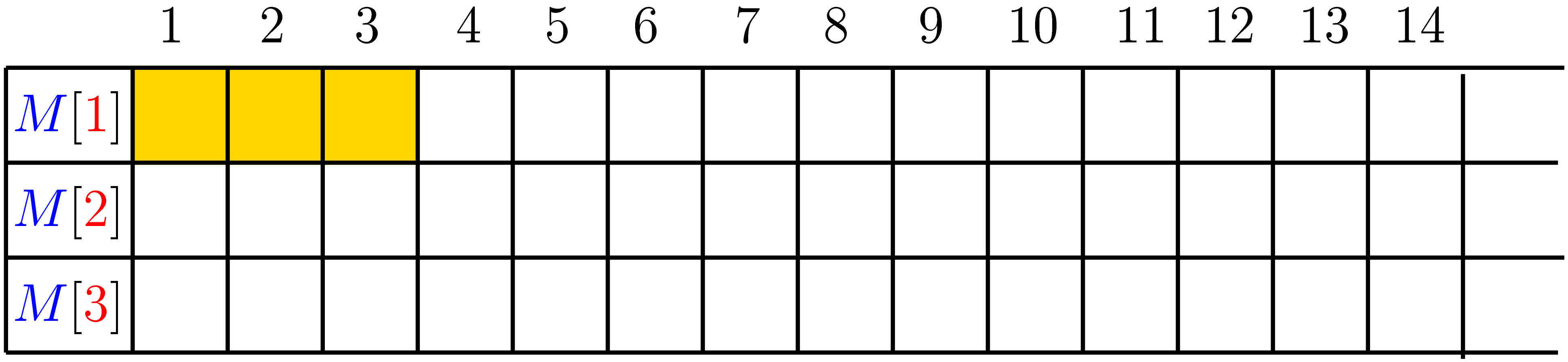
| $d[1]$ | $d[2]$ | $d[3]$ | $d[4]$ | $d[5]$ | $d[6]$ | $d[7]$ |
|---|---|---|---|---|---|---|
| 3 | 2 | 7 | 5 | 1 | 6 | 2 |

| | 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.

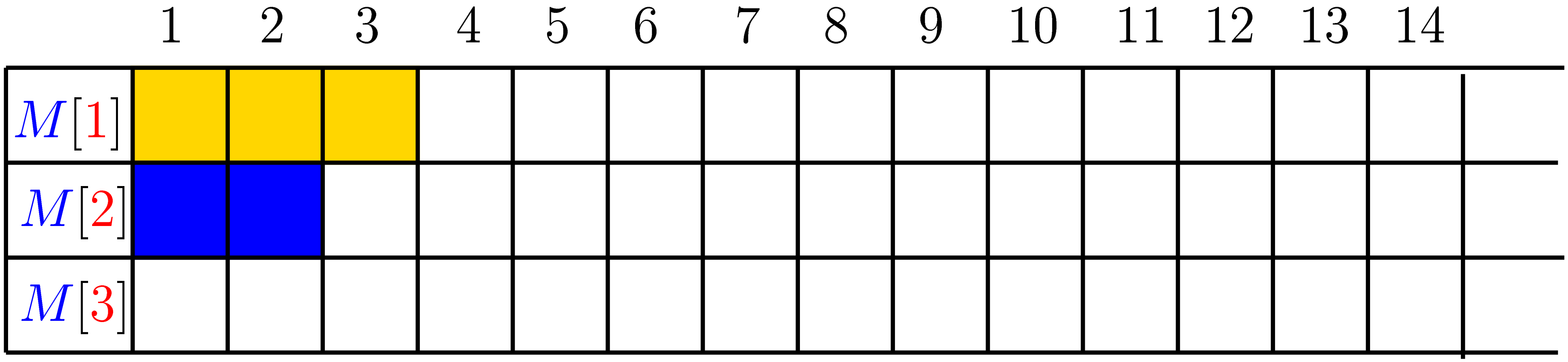
| $d[1]$ | $d[2]$ | $d[3]$ | $d[4]$ | $d[5]$ | $d[6]$ | $d[7]$ |
|---|---|---|---|---|---|---|
| 3 | 2 | 7 | 5 | 1 | 6 | 2 |



# Algoritmo de Graham

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

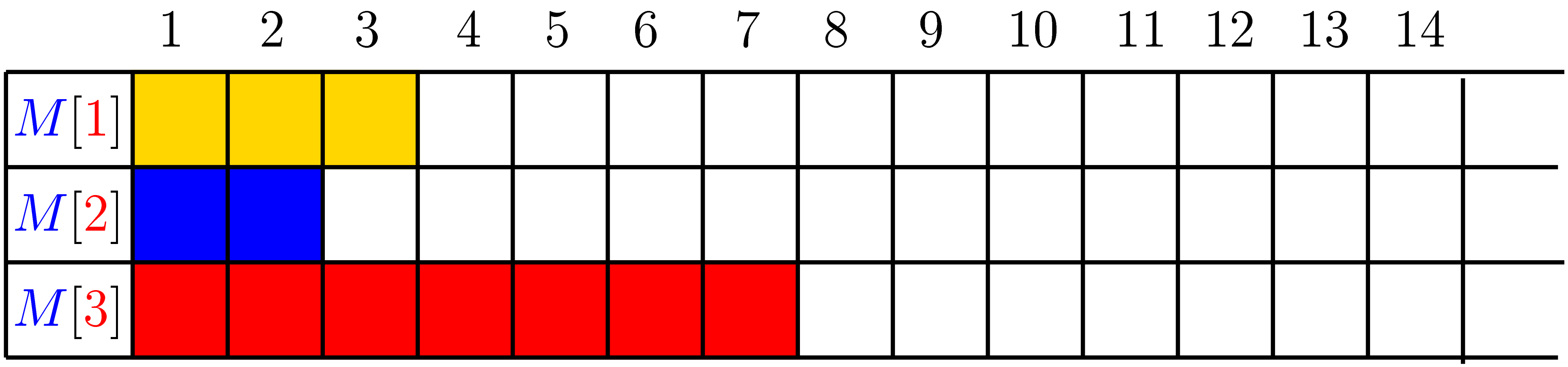
| $d[1]$ | $d[2]$ | $d[3]$ | $d[4]$ | $d[5]$ | $d[6]$ | $d[7]$ |
|---|---|---|---|---|---|---|
| 3 | 2 | 7 | 5 | 1 | 6 | 2 |



# Algoritmo de Graham

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

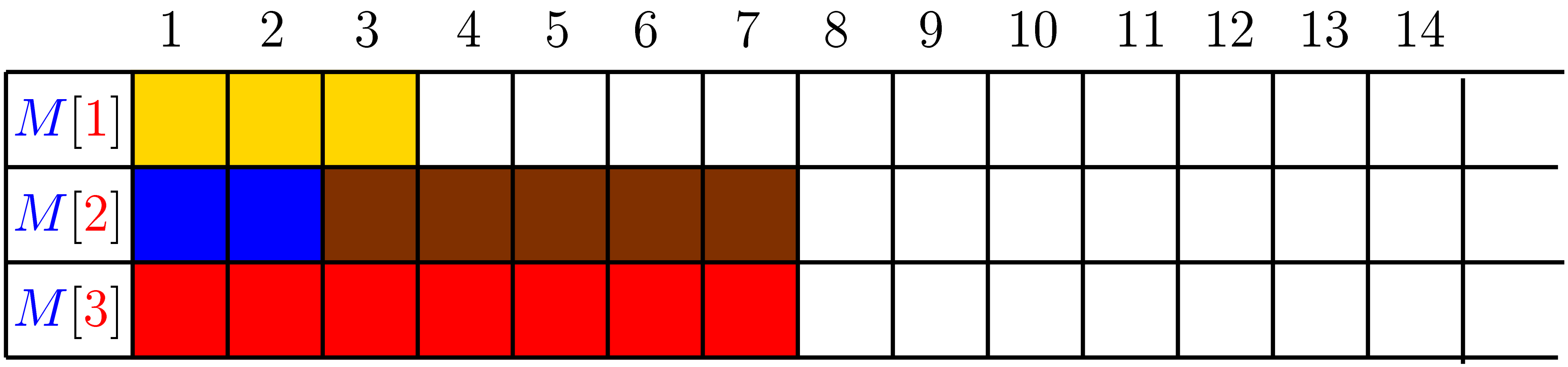
| $d[1]$ | $d[2]$ | $d[3]$ | $d[4]$ | $d[5]$ | $d[6]$ | $d[7]$ |
|---|---|---|---|---|---|---|
| 3 | 2 | 7 | 5 | 1 | 6 | 2 |



# Algoritmo de Graham

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

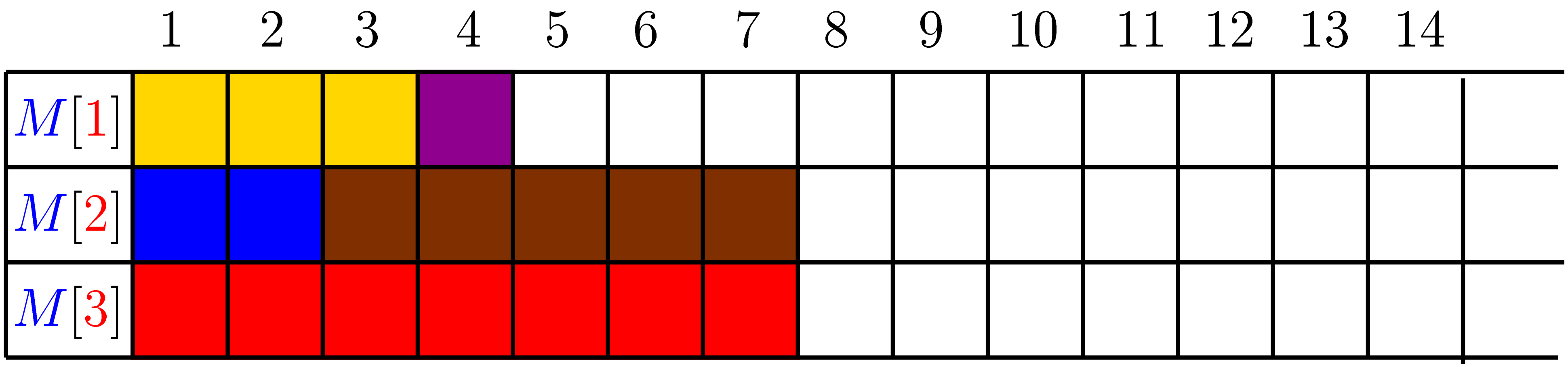
| $d[1]$ | $d[2]$ | $d[3]$ | $d[4]$ | $d[5]$ | $d[6]$ | $d[7]$ |
|---|---|---|---|---|---|---|
| 3 | 2 | 7 | 5 | 1 | 6 | 2 |



# Algoritmo de Graham

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

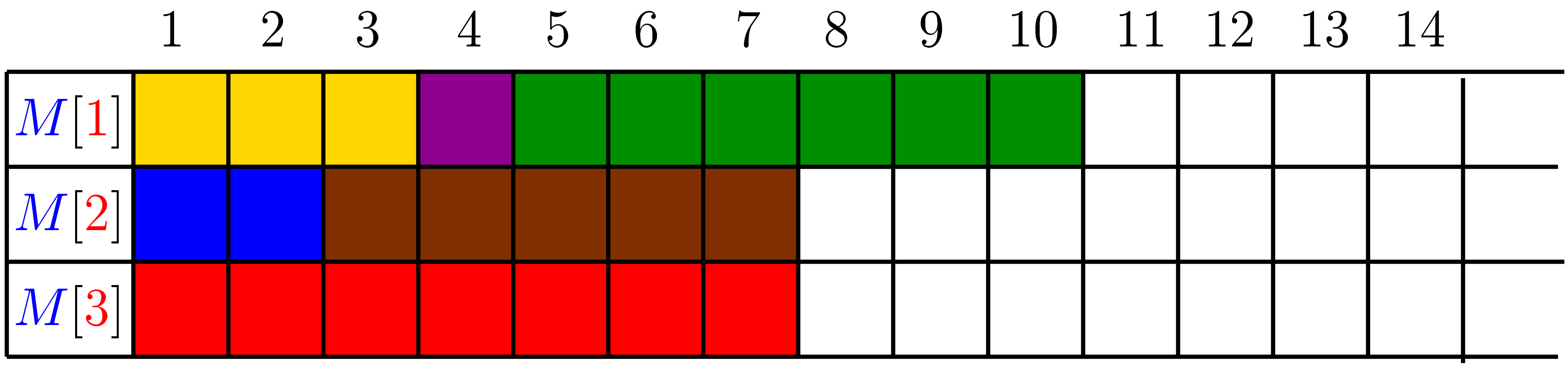
| $d[1]$ | $d[2]$ | $d[3]$ | $d[4]$ | $d[5]$ | $d[6]$ | $d[7]$ |
|---|---|---|---|---|---|---|
| 3 | 2 | 7 | 5 | 1 | 6 | 2 |



# Algoritmo de Graham

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

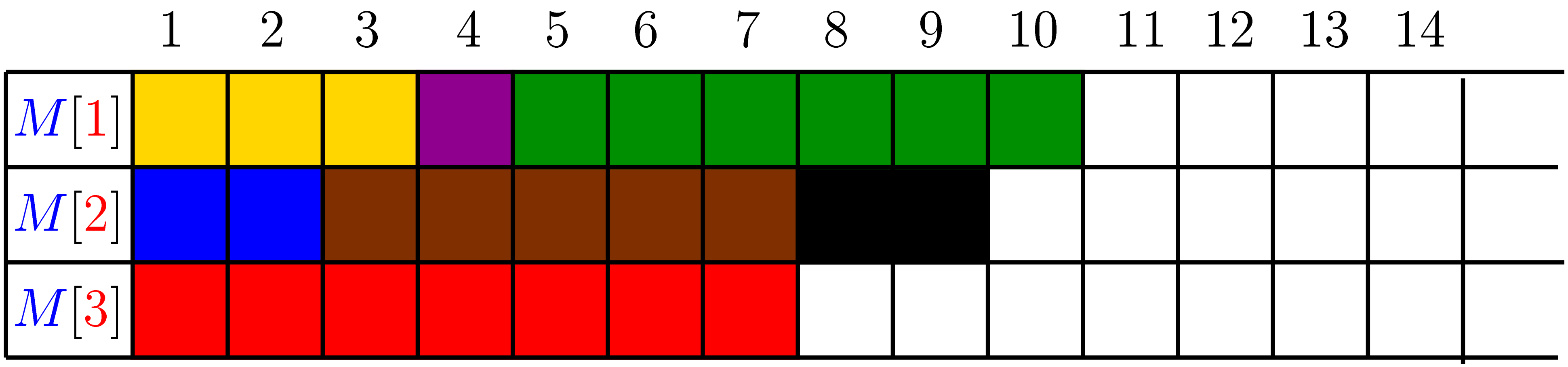
| $d[1]$ | $d[2]$ | $d[3]$ | $d[4]$ | $d[5]$ | $d[6]$ | $d[7]$ |
|---|---|---|---|---|---|---|
| 3 | 2 | 7 | 5 | 1 | 6 | 2 |



# Algoritmo de Graham

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

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



# 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 $\mathrm{OPT}$

$\mathrm{OPT}$ = menor tempo de conclusão de um escalonamento

- Duração da tarefa mais longa:

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

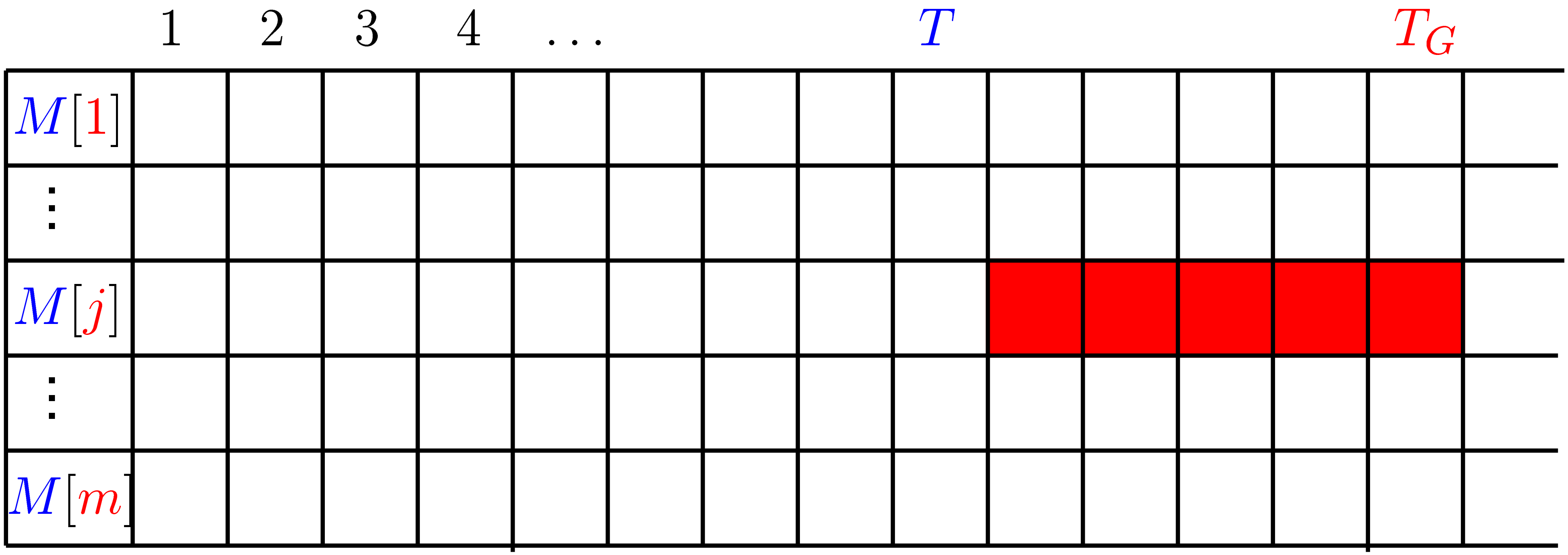
- Distribuição balanceada:

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

# Fator de aproximação

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

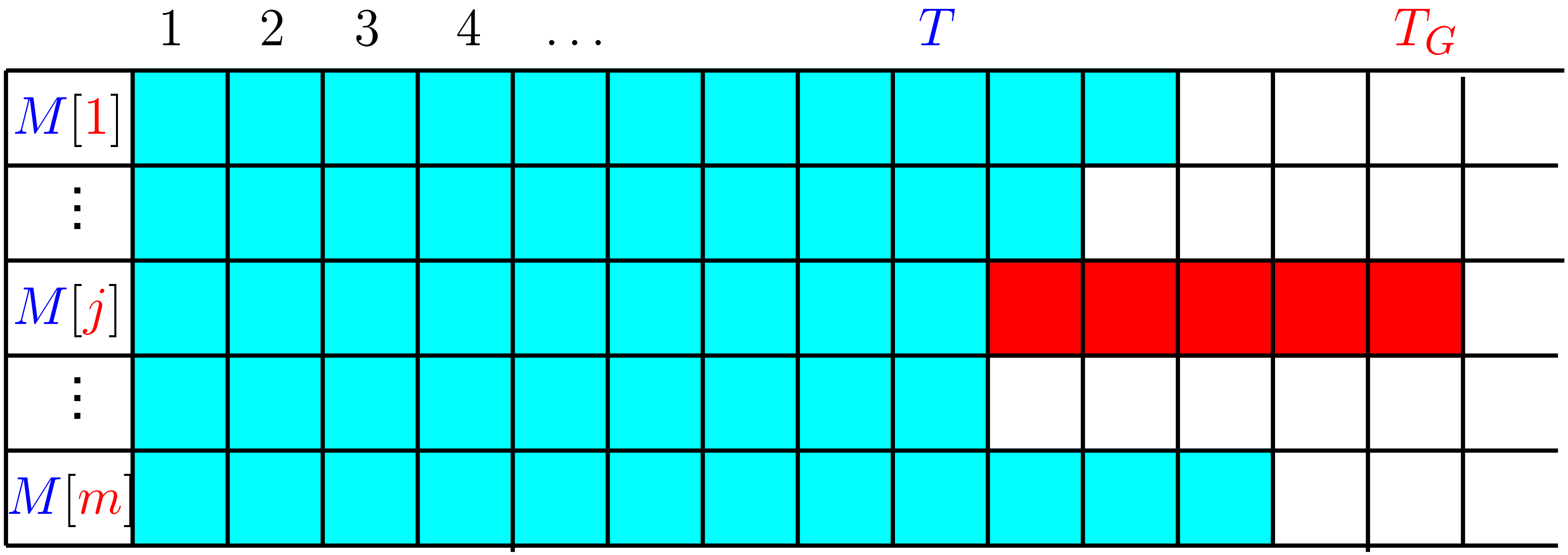
$T_G =$ conclusão do escalonamento obtido pelo algoritmo



# Fator de aproximação (cont.)

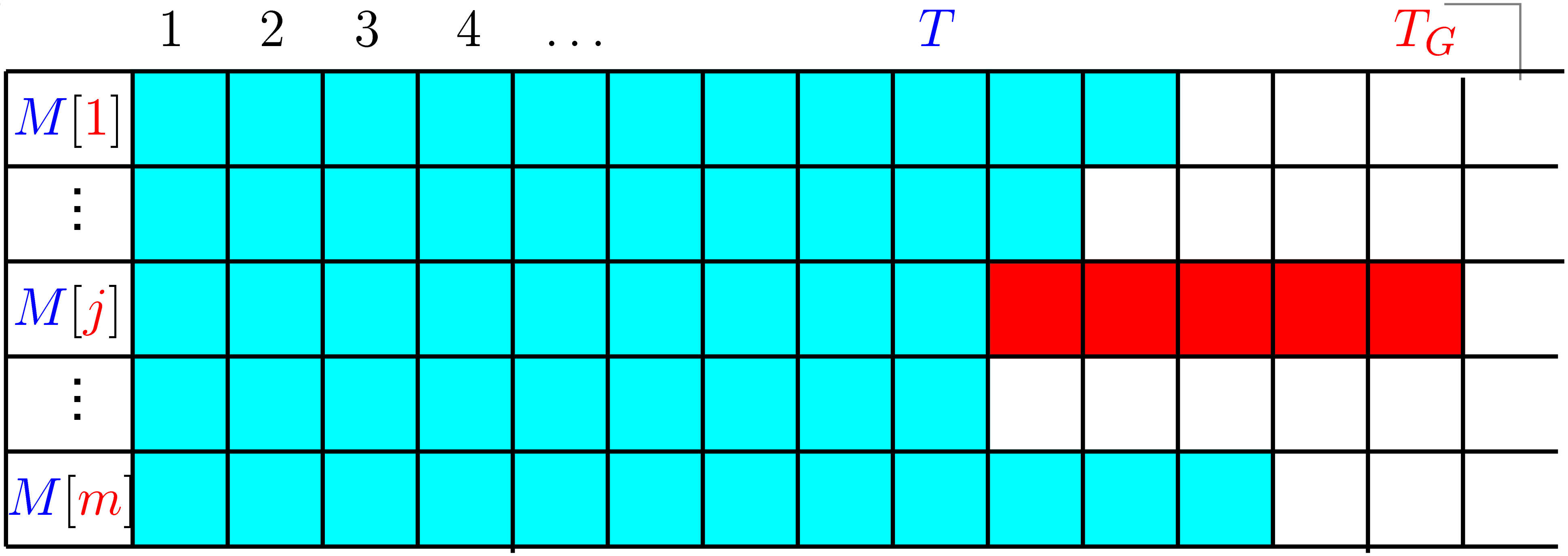
tarefa $t$ é executada na máquina $j$ a partir do instante $T$

$T_G =$ 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.)




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

# 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.

$\alpha$ é o fator de aproximação

Objetivo: $\alpha$ tão perto de 1 quanto possível

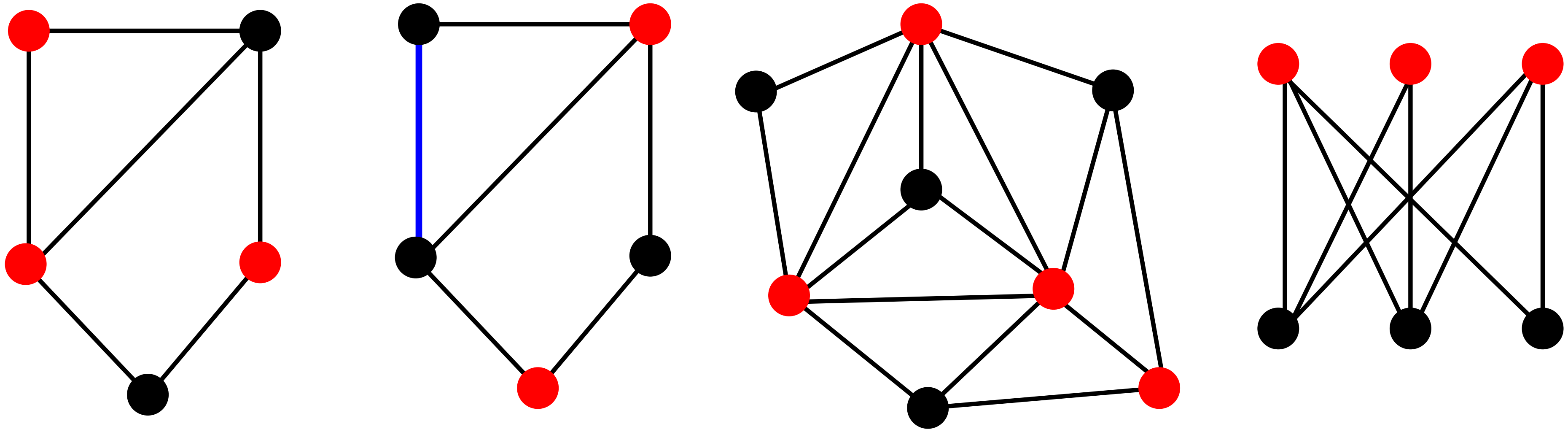
# Conclusão

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

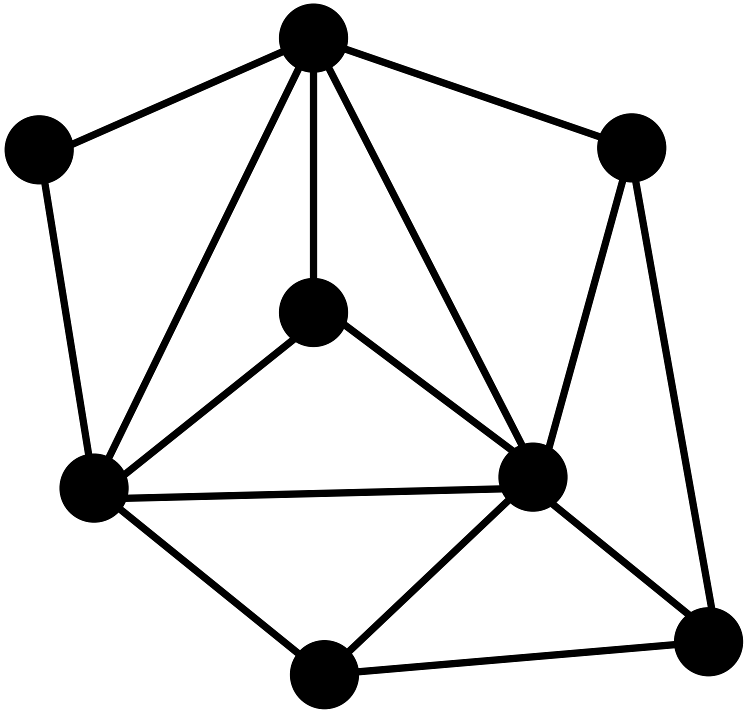
# Cobertura por vértices

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

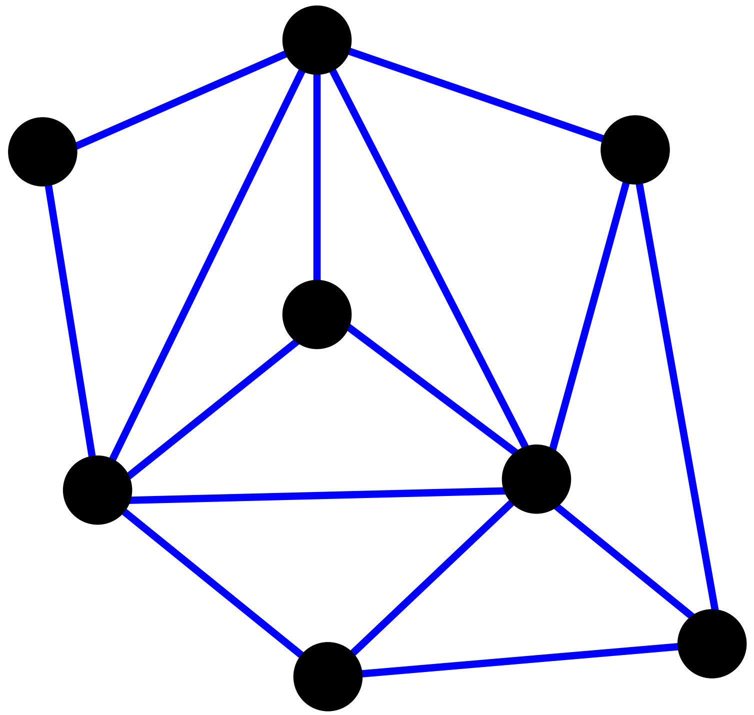
Problema é NP-difícil.

```
APPROX-VERTEX-COVER (G)
C ← ∅
E ← {arestas de G}
enquanto E ≠ ∅ faça
        seja (u, v) uma aresta qualquer em E
        C ← C ∪ {u, v}
        E ← E − {arestas incidentes a u ou v}
devolva C
```

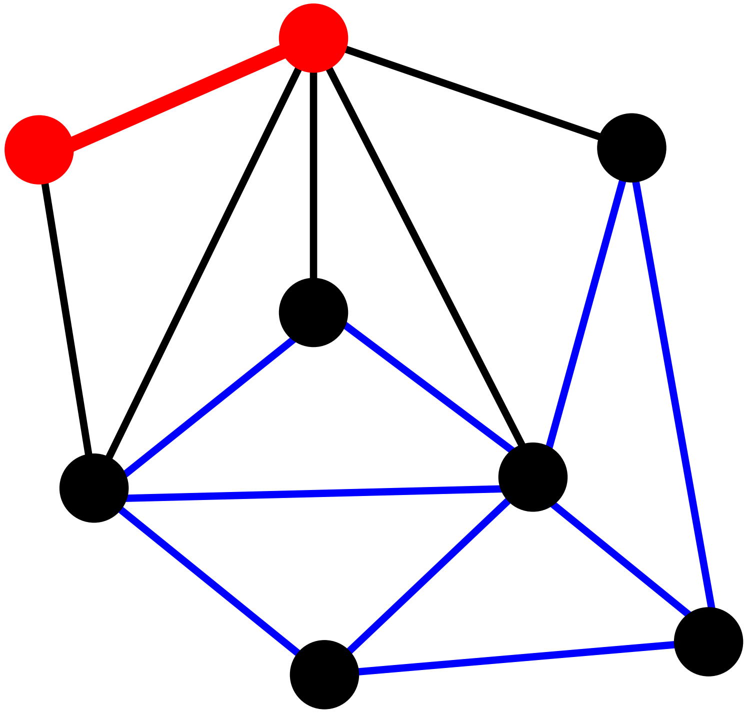
# Exemplo



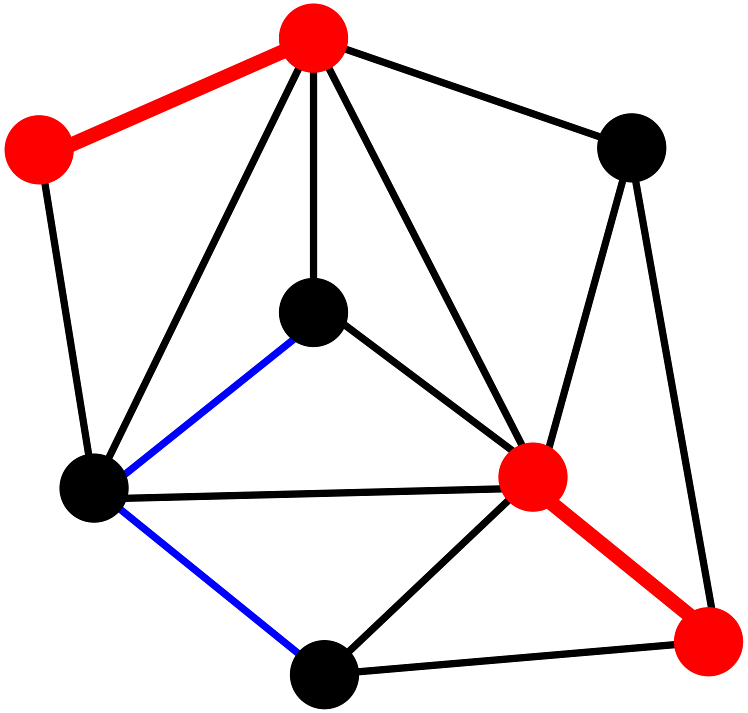
# Exemplo



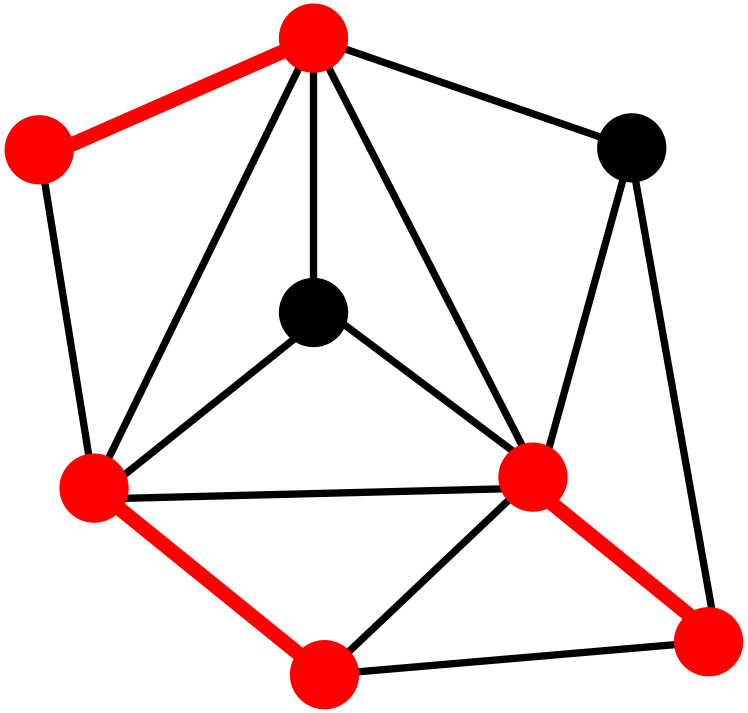
# Exemplo



# Exemplo



# Exemplo

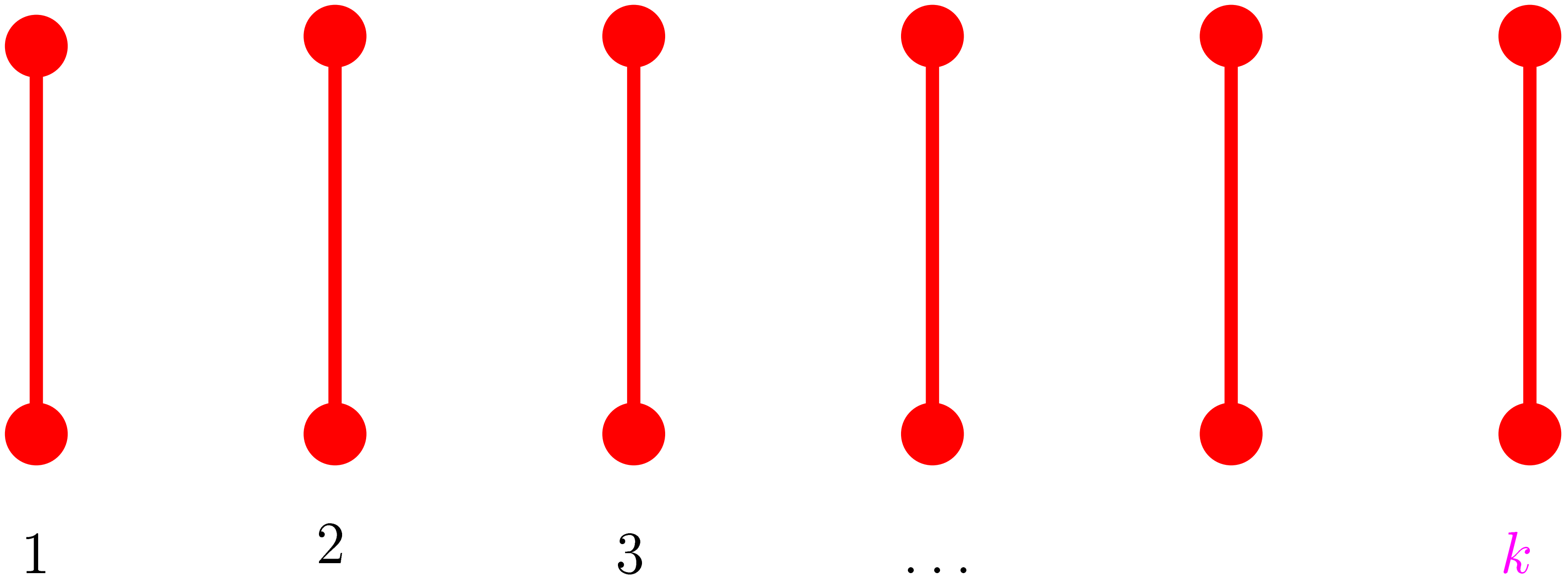


# Consumo de tempo

$E$ 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 | $\mathrm{O}(m)$ |
| 4 | $\mathrm{O}(m)$ |
| 5 | $\mathrm{O}(n)$ |
| 6 | $\mathrm{O}(m)$ |
| 7 | $\mathrm{O}(n)$ |
| total | $\Theta(1) + \Theta(n+m) + 3\mathrm{O}(m) + 2\mathrm{O}(n) = \Theta(n+m)$ |

# **Delimitações para** OPT

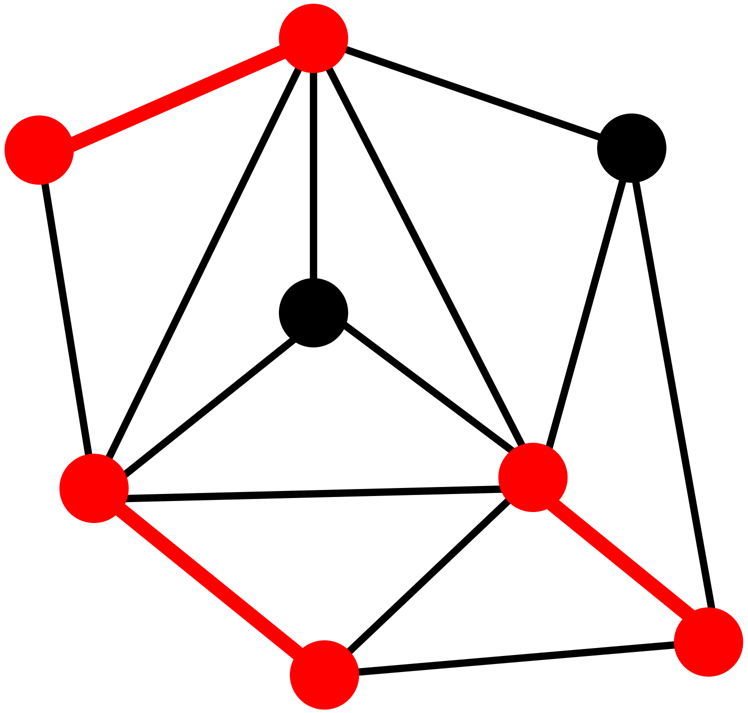



Se $G$ possui um emparelhamento com $k$ arestas, então

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

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

# Fator de aproximação



$C$ = cobertura devolvida pelo algoritmo.
$G$ possui um emparelamento 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.