Análise de Algoritmos

Parte destes slides são adaptações de slides

do Prof. Paulo Feofiloff e do Prof. José Coelho de Pina.

Problemas de Otimização Combinatória

conjunto de instâncias

para cada instância I,
um conjunto $\mathrm{Sol}(I)$ de soluções viáveis e
uma função $\mathrm{val}(S)$ para cada solução S em $\mathrm{Sol}(I)$

Problemas de Otimização Combinatória

conjunto de instâncias

para cada instância I,
  um conjunto $\mathrm{Sol}(I)$ de soluções viáveis e
  uma função $\mathrm{val}(S)$ para cada solução S em $\mathrm{Sol}(I)$

Se $\mathrm{Sol}(I)$ é vazio, I é inviável, caso contrário, I é viável.

Problemas de Otimização Combinatória

conjunto de instâncias

para cada instância I,
  um conjunto $\mathrm{Sol}(I)$ de soluções viáveis e
  uma função $\mathrm{val}(S)$ para cada solução S em $\mathrm{Sol}(I)$

Se $\mathrm{Sol}(I)$ é vazio, I é inviável, caso contrário, I é viável.

Problema de minimização: Dada uma instância I viável, encontrar uma solução S em $\mathrm{Sol}(I)$ tal que $\mathrm{val}(S)$ é mínimo.

Problema de maximização: Dada uma instância I viável, encontrar uma solução S em $\mathrm{Sol}(I)$ tal que $\mathrm{val}(S)$ é máximo.

Problemas de Otimização Combinatória

conjunto de instâncias

para cada instância I,
  um conjunto $\mathrm{Sol}(I)$ de soluções viáveis e
  uma função $\mathrm{val}(S)$ para cada solução S em $\mathrm{Sol}(I)$

Se $\mathrm{Sol}(I)$ é vazio, I é inviável, caso contrário, I é viável.

Problema de minimização: Dada uma instância I viável, encontrar uma solução S em $\mathrm{Sol}(I)$ tal que $\mathrm{val}(S)$ é mínimo.

Problema de maximização: Dada uma instância I viável, encontrar uma solução S em $\mathrm{Sol}(I)$ tal que $\mathrm{val}(S)$ é máximo.

$\mathrm{opt}(I)$: valor ótimo (valor de uma solução ótima)

Como lidar com problemas NP-difíceis?

Um dito em engenharia:

"Rápido. Barato. Confiável. Escolha dois."

Como lidar com problemas NP-difíceis?

Um dito em engenharia:

"Rápido. Barato. Confiável. Escolha dois."

Podemos ter algoritmos que (1) encontram soluções ótimas (2) em tempo polinomial (3) para qualquer instância.

Como lidar com problemas NP-difíceis?

Um dito em engenharia:

"Rápido. Barato. Confiável. Escolha dois."

Podemos ter algoritmos que (1) encontram soluções ótimas (2) em tempo polinomial (3) para qualquer instância.

Abrimos mão de (3) quando procuramos casos particulares do problema que conseguimos resolver eficientemente.

Como lidar com problemas NP-difíceis?

Um dito em engenharia:

"Rápido. Barato. Confiável. Escolha dois."

Podemos ter algoritmos que (1) encontram soluções ótimas (2) em tempo polinomial (3) para qualquer instância.

Abrimos mão de (3) quando procuramos casos particulares do problema que conseguimos resolver eficientemente.

Em programação inteira por exemplo abre-se mão de (2).

Como lidar com problemas NP-difíceis?

Um dito em engenharia:

"Rápido. Barato. Confiável. Escolha dois."

Podemos ter algoritmos que (1) encontram soluções ótimas (2) em tempo polinomial (3) para qualquer instância.

Abrimos mão de (3) quando procuramos casos particulares do problema que conseguimos resolver eficientemente.

Em programação inteira por exemplo abre-se mão de (2).

Em algoritmos de aproximação, abrimos mão de (1):

Como lidar com problemas NP-difíceis?

Um dito em engenharia:

"Rápido. Barato. Confiável. Escolha dois."

Podemos ter algoritmos que (1) encontram soluções ótimas (2) em tempo polinomial (3) para qualquer instância.

Abrimos mão de (3) quando procuramos casos particulares do problema que conseguimos resolver eficientemente.

Em programação inteira por exemplo abre-se mão de (2).

Em algoritmos de aproximação, abrimos mão de (1):

buscamos *boas* soluções que
possam ser obtidas eficientemente.

Algoritmos de aproximação

Algoritmo A é de aproximação se
é polinomial e existe $\alpha > 0$ tal que

$$\mathrm{val}(A(I)) \leq \alpha \cdot \mathrm{opt}\,(I)$$

para toda instância I do problema (de minimização).

Algoritmos de aproximação

Algoritmo A é de aproximação se
é polinomial e existe $\alpha > 0$ tal que

$$\mathrm{val}(A(I)) \leq \alpha \cdot \mathrm{opt}\,(I)$$

para toda instância I do problema (de minimização).

A é uma α-aproximação e

α é a razão de aproximação (ou garantia de performance).

Algoritmos de aproximação

Algoritmo A é de aproximação se
é polinomial e existe $\alpha > 0$ tal que

$$\mathrm{val}(A(I)) \leq \alpha \cdot \mathrm{opt}\,(I)$$

para toda instância I do problema (de minimização).

A é uma α-aproximação e

α é a razão de aproximação (ou garantia de performance).

$\alpha > 1$ para problemas de minimização

Algoritmos de aproximação

Algoritmo A é de aproximação se
é polinomial e existe $\alpha > 0$ tal que

$$\mathrm{val}(A(I)) \leq \alpha \cdot \mathrm{opt}\,(I)$$

para toda instância I do problema (de minimização).

A é uma α-aproximação e

α é a razão de aproximação (ou garantia de performance).

$\alpha > 1$ para problemas de minimização

Para problemas de maximização,
a desigualdade é invertida e $\alpha < 1$.

Algoritmos de aproximação

Algoritmo A é de aproximação se
é polinomial e existe $\alpha > 0$ tal que

$$\mathrm{val}(A(I)) \leq \alpha \cdot \mathrm{opt}\,(I)$$

para toda instância I do problema (de minimização).

A é uma α-aproximação e

α é a razão de aproximação (ou garantia de performance).

$\alpha > 1$ para problemas de minimização

Para problemas de maximização,
a desigualdade é invertida e $\alpha < 1$.

Objetivo: α tão perto de 1 quanto possível

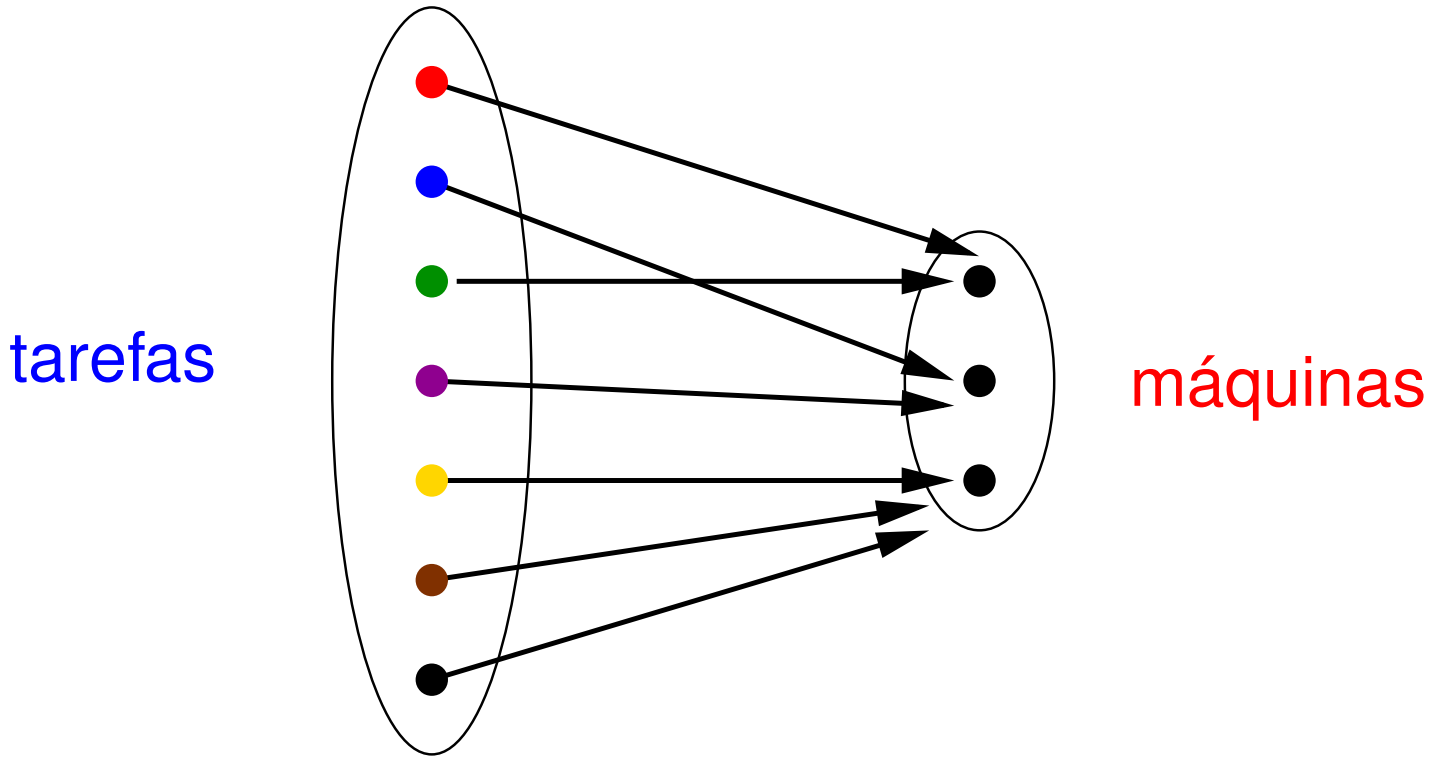
Escalonamento de máquinas idênticas

Dados: q máquinas

n tarefas

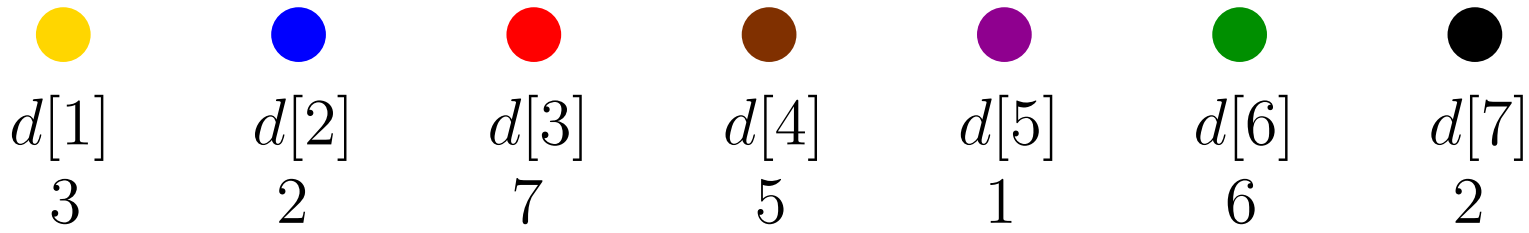
duração $d[i]$ da tarefa i $\quad (i = 1, \ldots, n)$

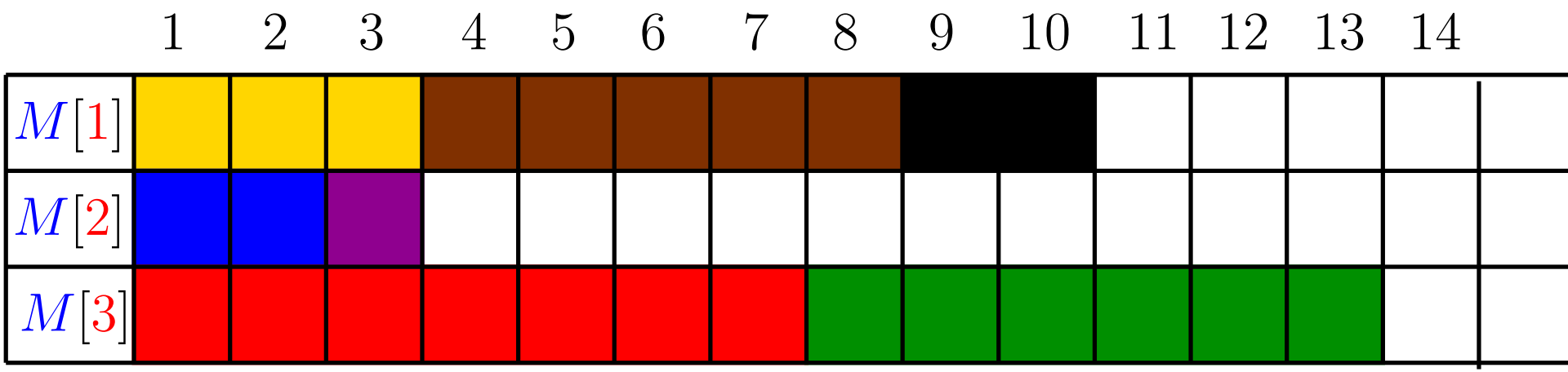
um escalonamento é uma partição $\{M[1], \ldots, M[q]\}$
de $\{1, \ldots, n\}$



Exemplo 1

$q = 3 \quad n = 7$

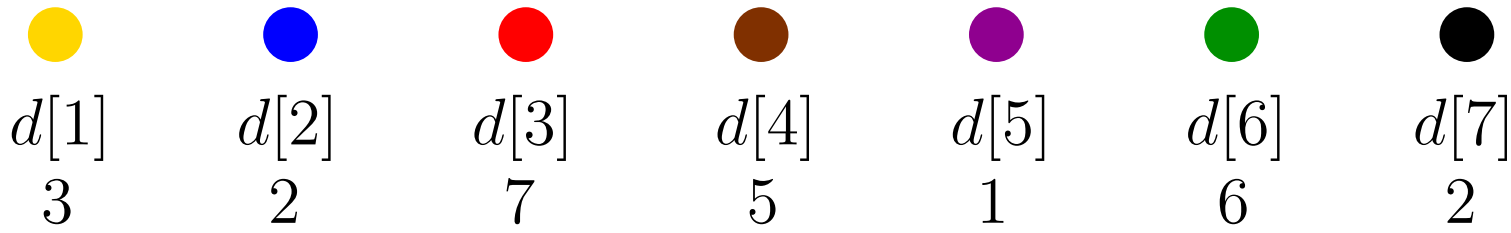


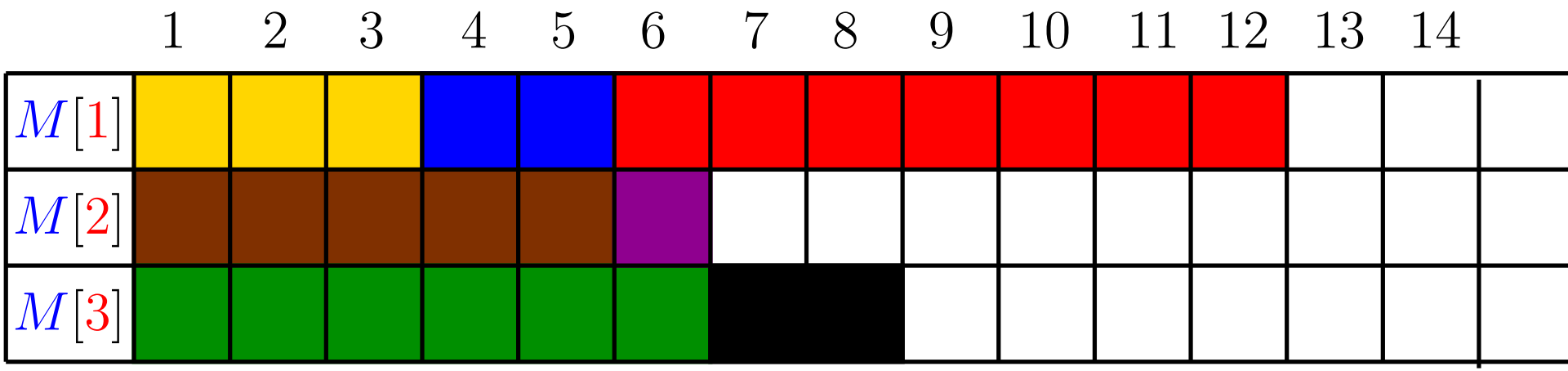


$\{\{1, 4, 7\}, \{2, 5\}, \{3, 6\}\} \Rightarrow$ Tempo de conclusão $= 13$

Exemplo 2

$q = 3 \quad n = 7$

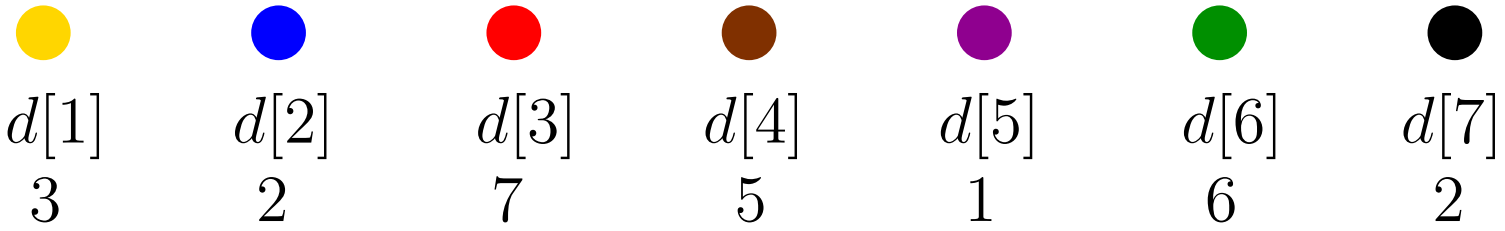


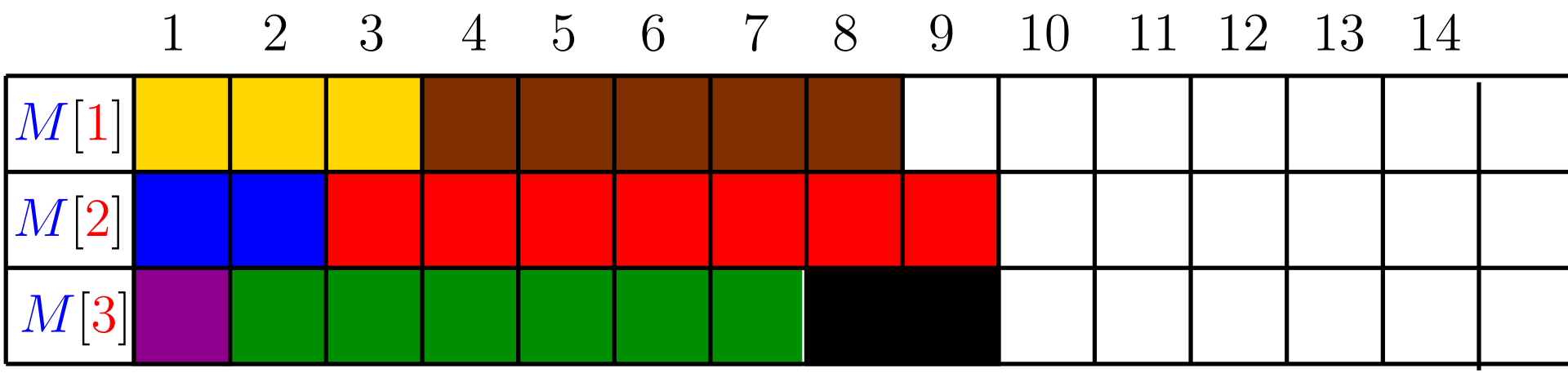


$\{\{1,2,3\},\{4,5\},\{6,7\}\} \Rightarrow$ Tempo de conclusão $= 12$

Problema

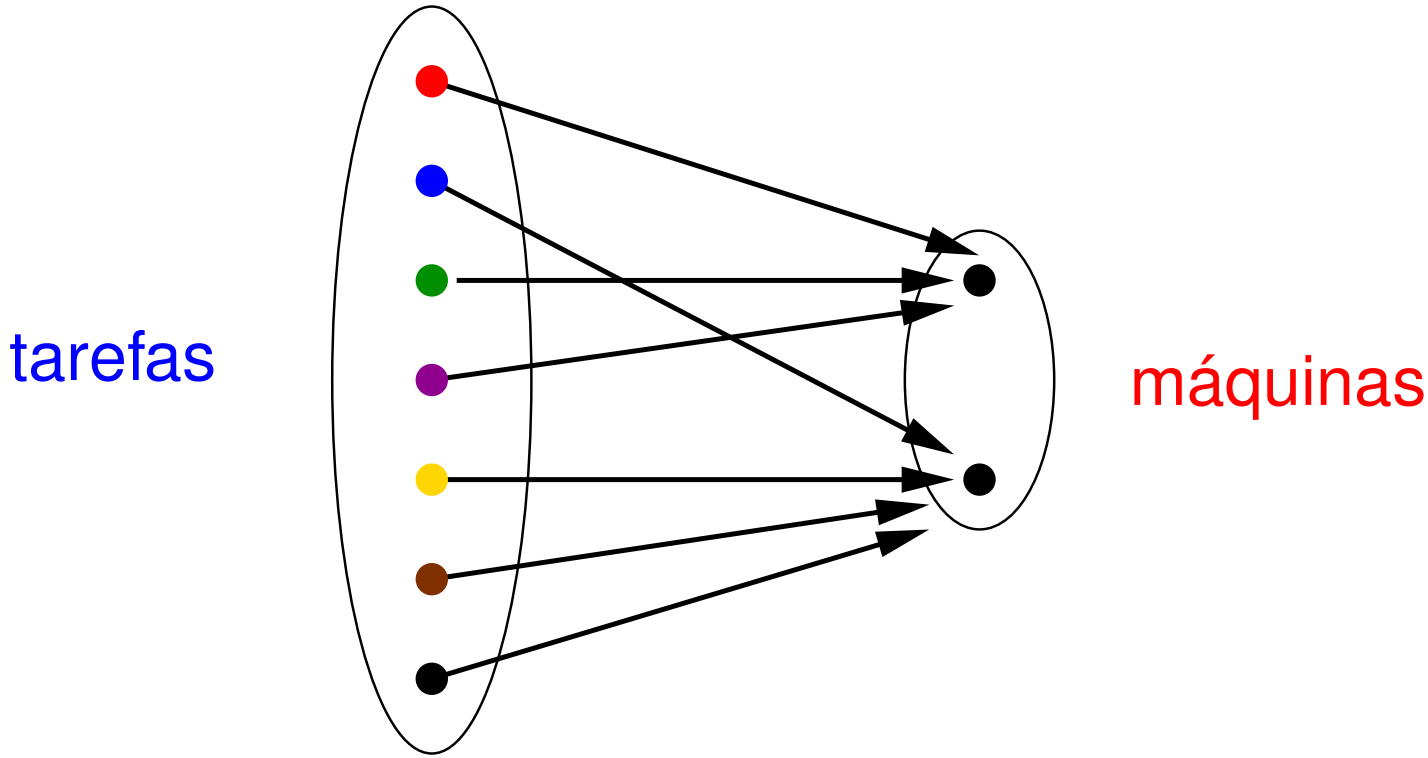
Encontrar um escalonamento com tempo de conclusão mínimo.





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

NP-difícil mesmo para $q = 2$



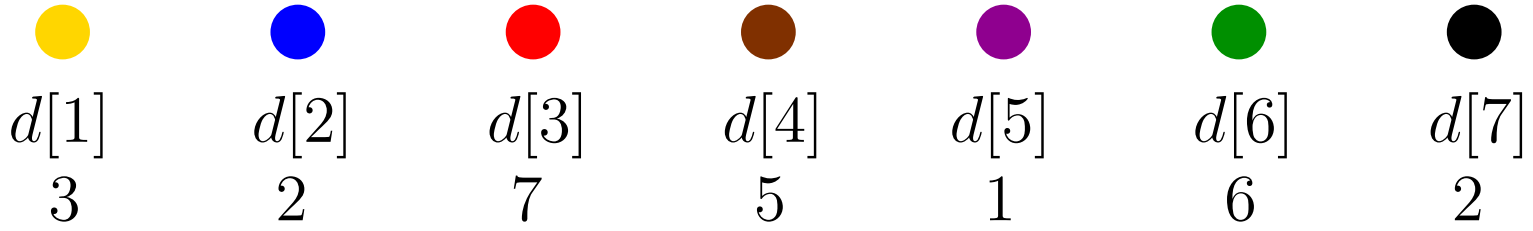
Algoritmo: testa todo $M[1] \subseteq \{1, \ldots, n\}$ e escolhe melhor

2^n subconjuntos $\Rightarrow$ exponencial

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

Algoritmo de Graham

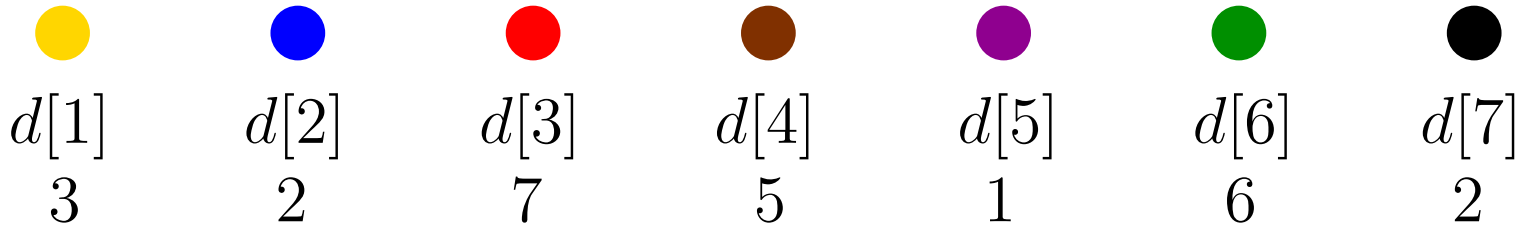
Atribui, uma a uma, cada tarefa à máquina menos ocupada.

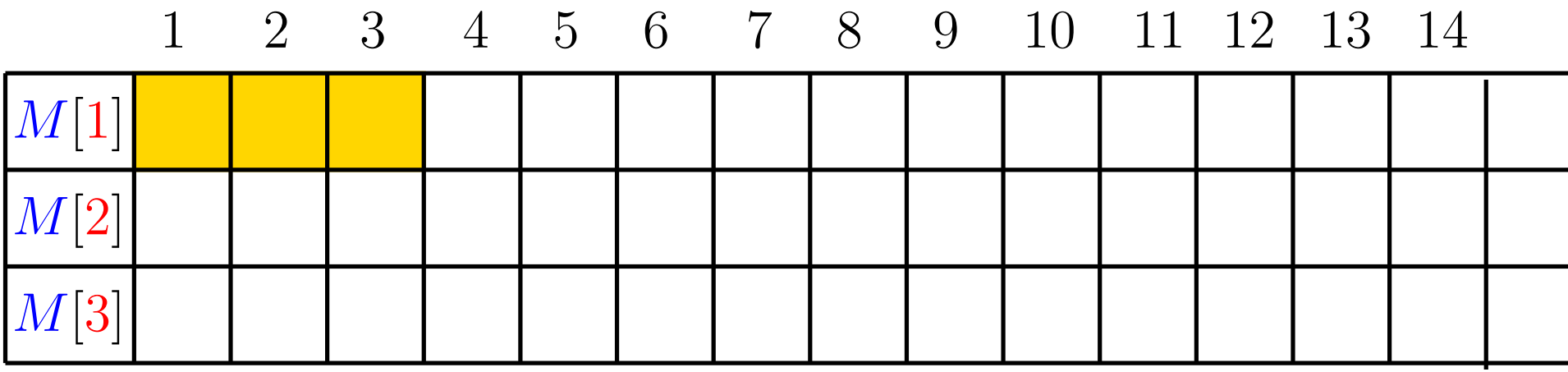


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
$M[1]$															
$M[2]$															
$M[3]$															

Algoritmo de Graham

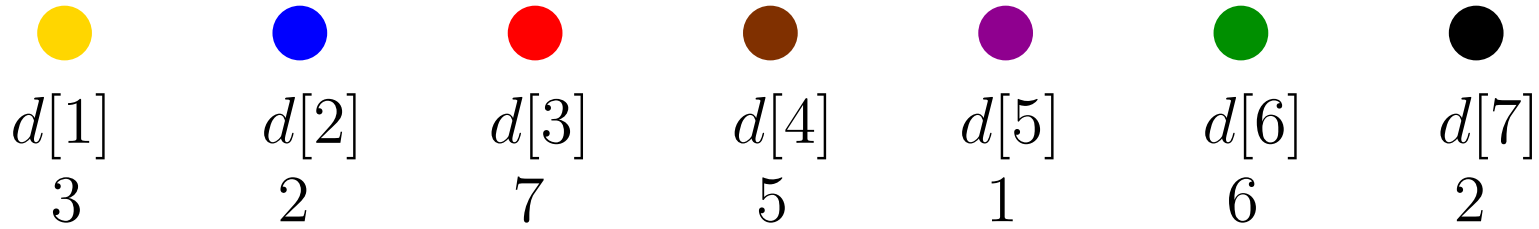
Atribui, uma a uma, cada tarefa à máquina menos ocupada.

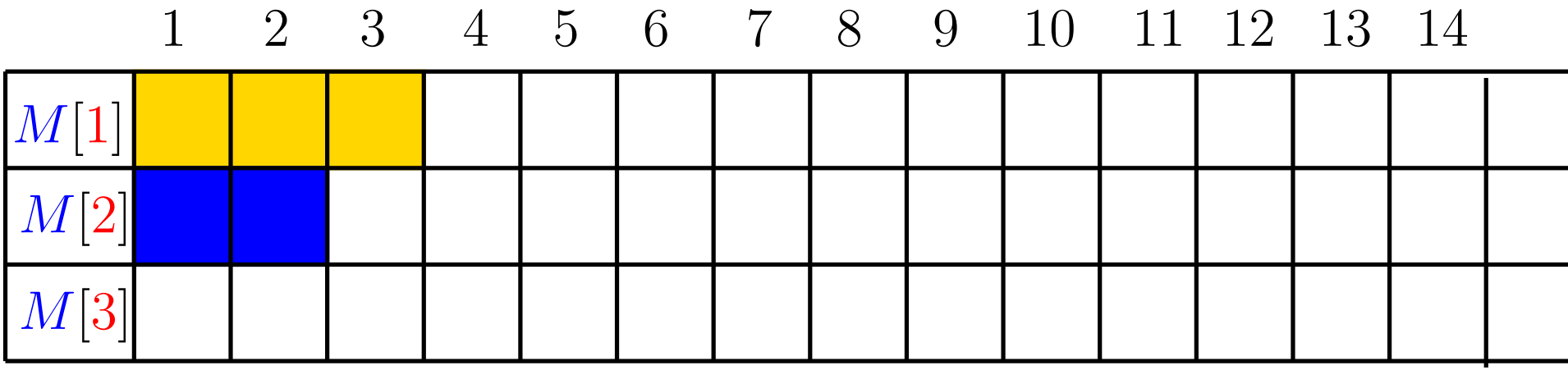




Algoritmo de Graham

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

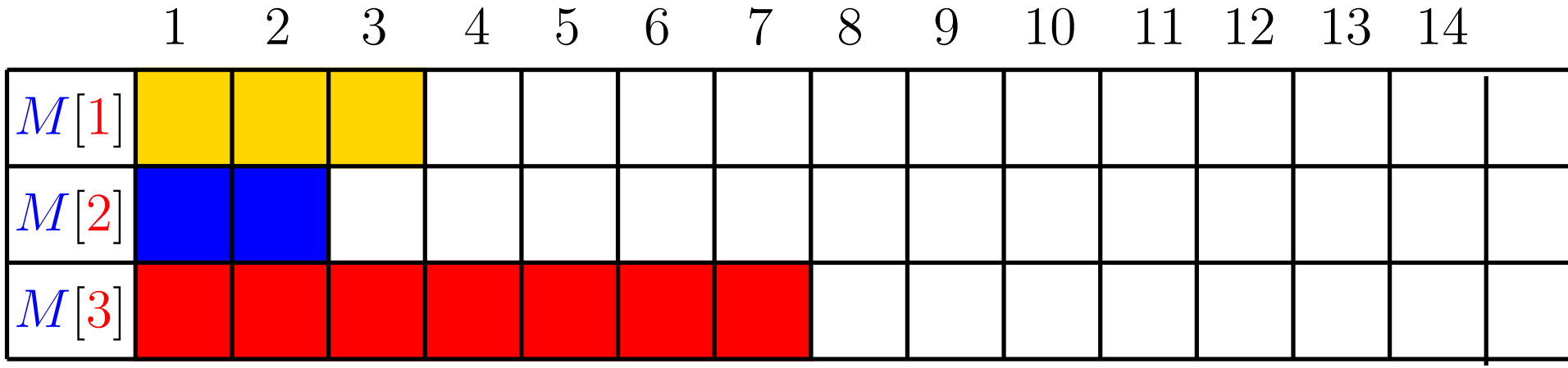




Algoritmo de Graham

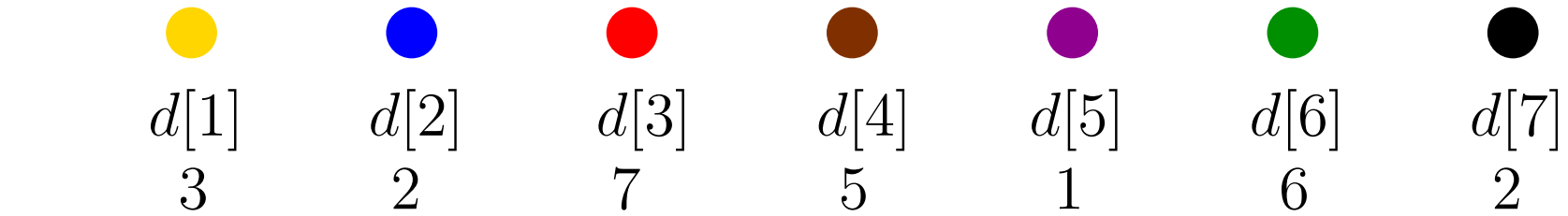
Atribui, uma a uma, cada tarefa à máquina menos ocupada.

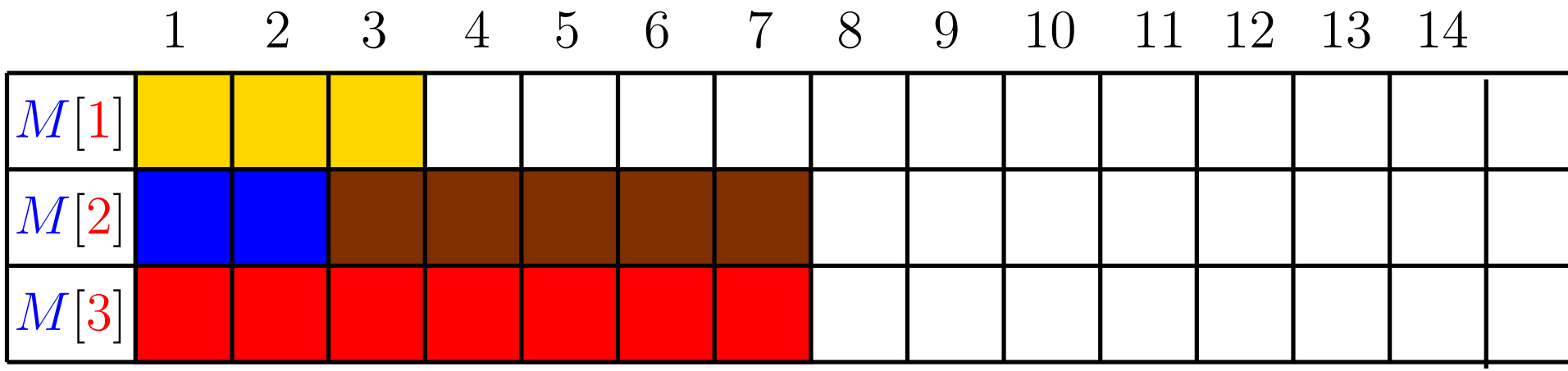
$d[1]$	$d[2]$	$d[3]$	$d[4]$	$d[5]$	$d[6]$	$d[7]$
3	2	7	5	1	6	2



Algoritmo de Graham

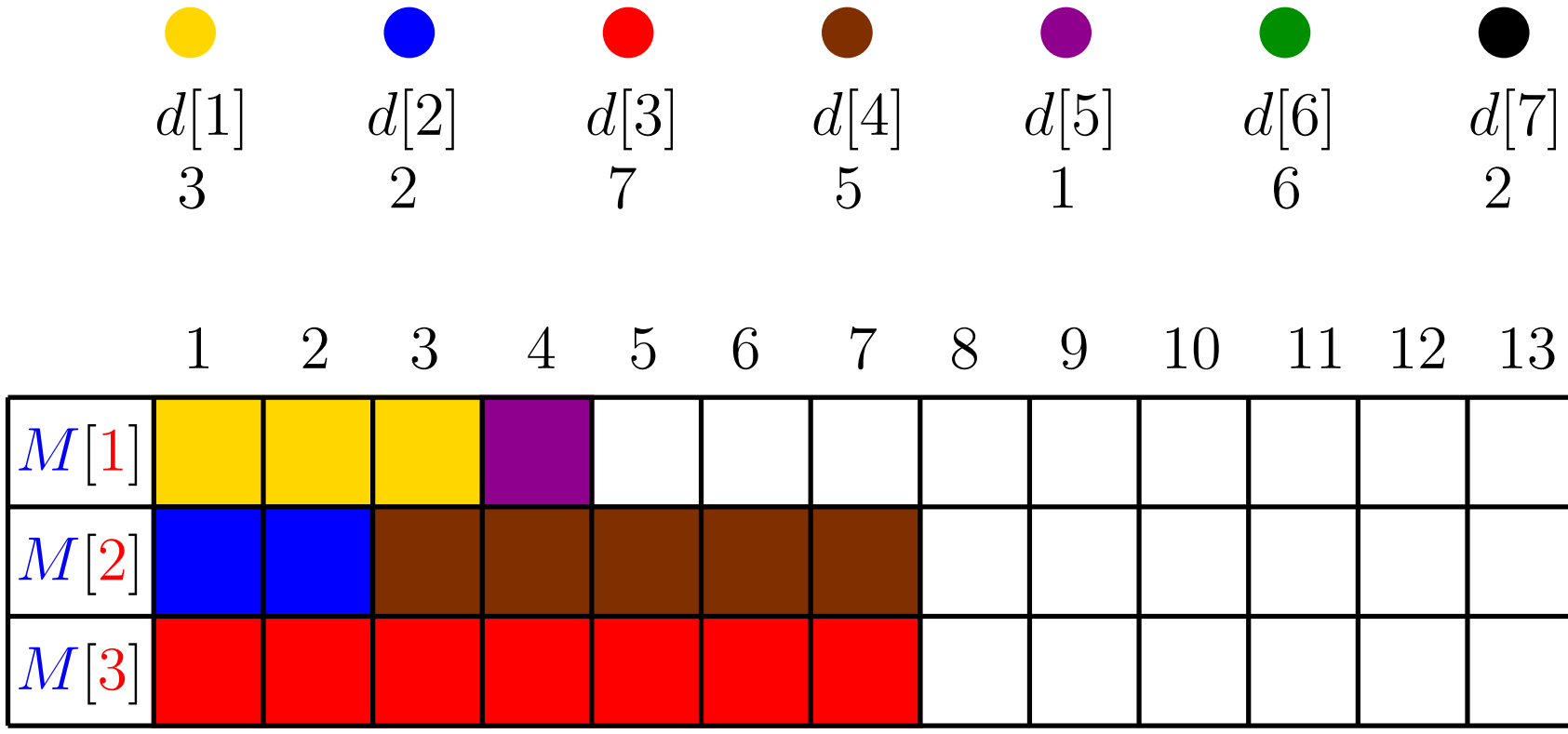
Atribui, uma a uma, cada tarefa à máquina menos ocupada.





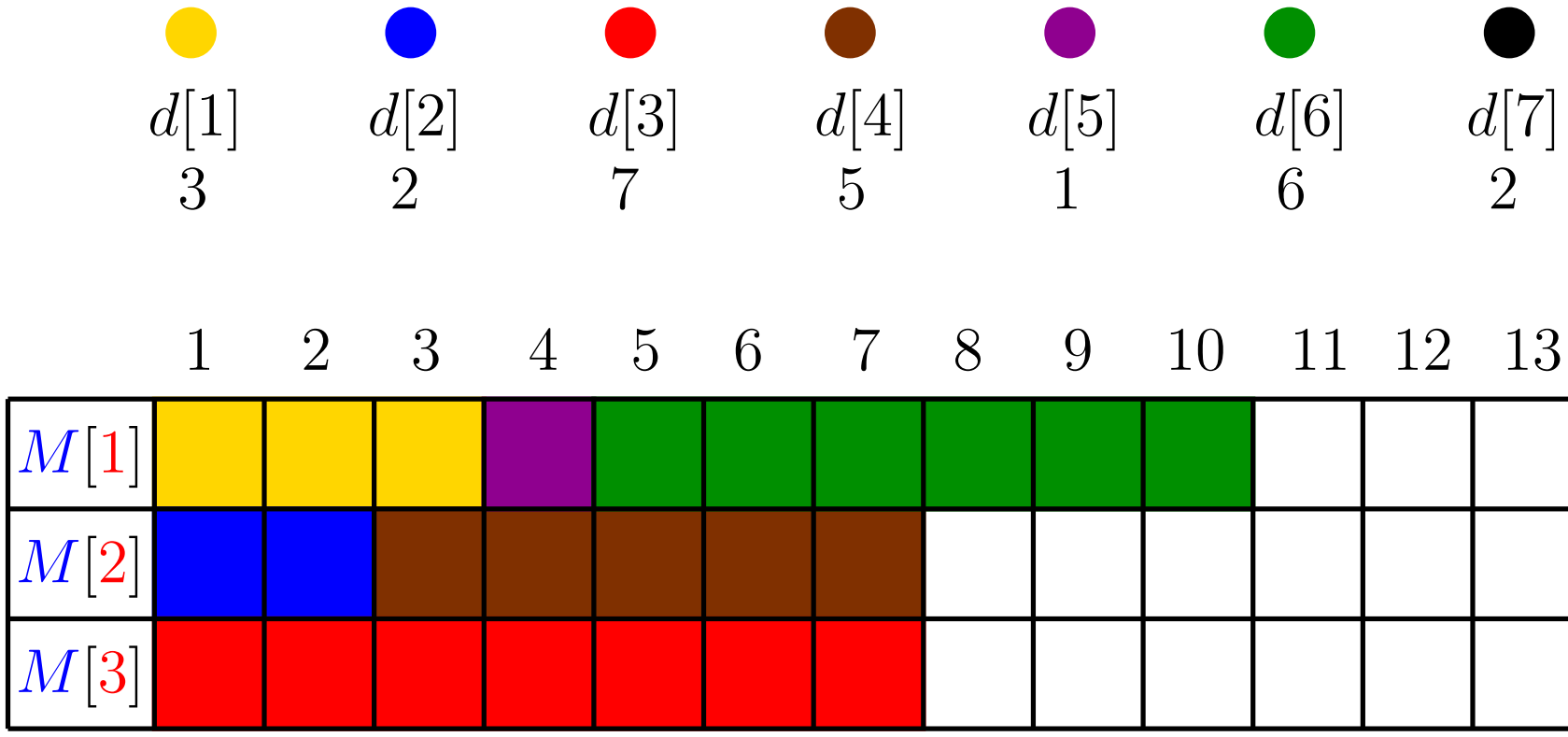
Algoritmo de Graham

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



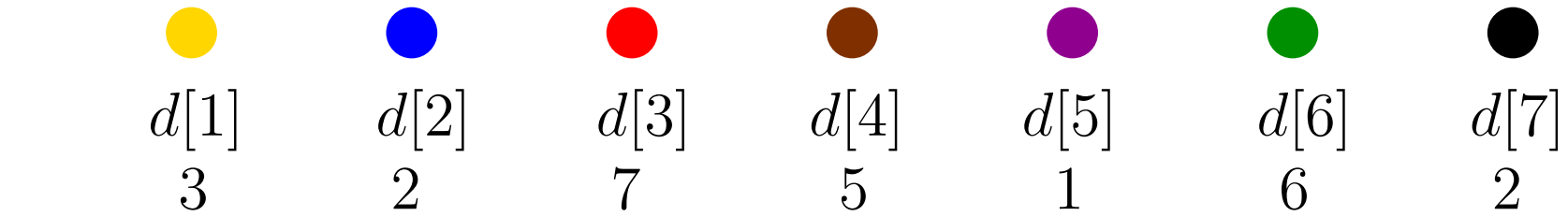
Algoritmo de Graham

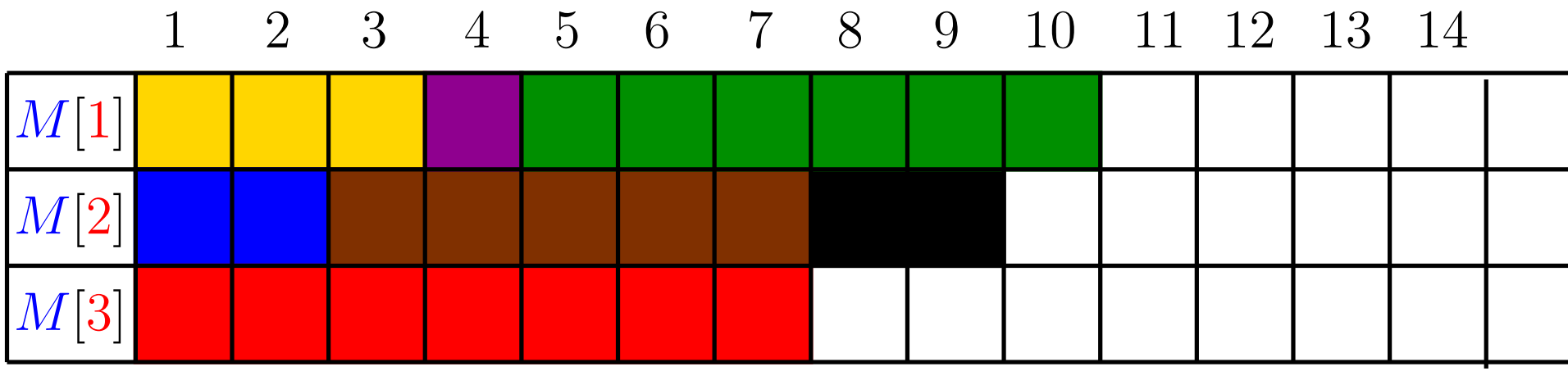
Atribui, uma a uma, cada tarefa à máquina menos ocupada.



Algoritmo de Graham

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





Escalonamento-Graham

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

ESCALONAMENTO-GRAHAM (q, n, d)

```
1   para j ← 1 até q faça
2       M[j] ← ∅
3       T[j] ← 0
4   para i ← 1 até n faça
5       seja k tal que T[k] é mínimo
6       M[k] ← M[k] ∪ {i}
7       T[k] ← T[k] + d[i]
8   devolva {M[1], ..., M[q]}
```

Delimitações para opt

opt = menor tempo de conclusão de um escalonamento

- Duração da tarefa mais longa:

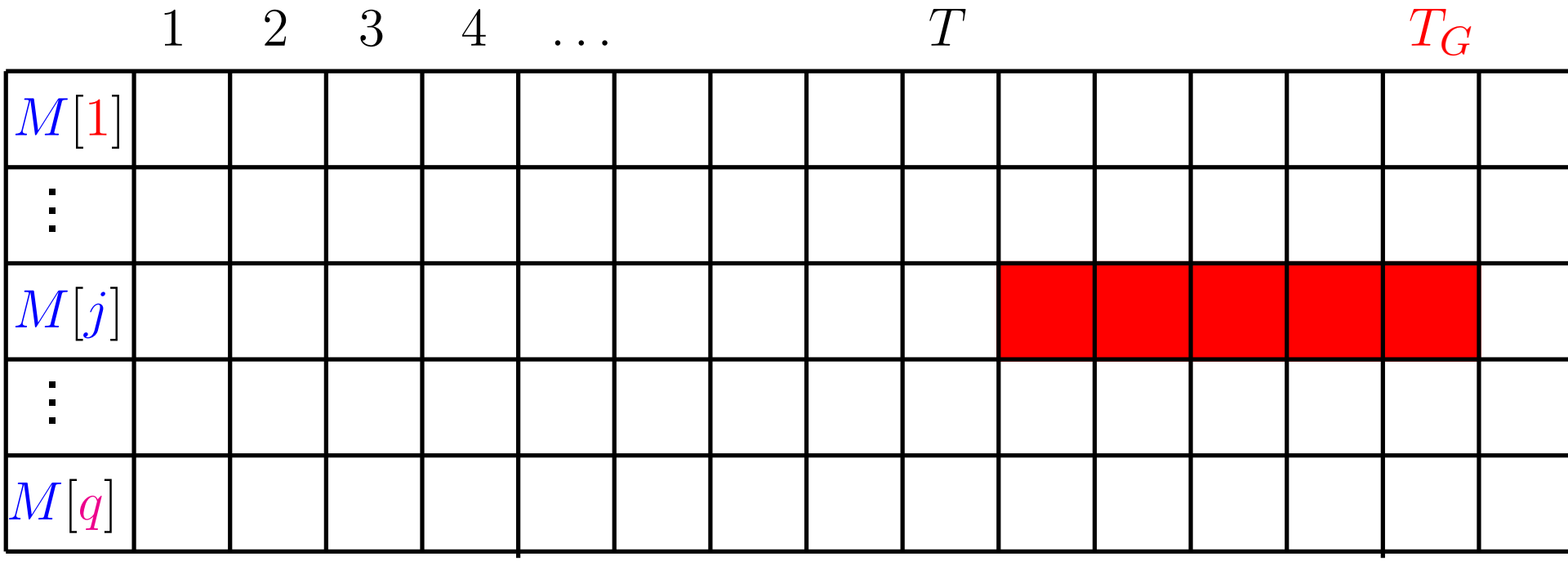
$$\mathrm{opt} \geq \max\{d[1], d[2], \ldots, d[n]\}$$

- Distribuição balanceada:

$$\mathrm{opt} \geq \frac{d[1] + d[2] + \cdots + d[n]}{q}$$

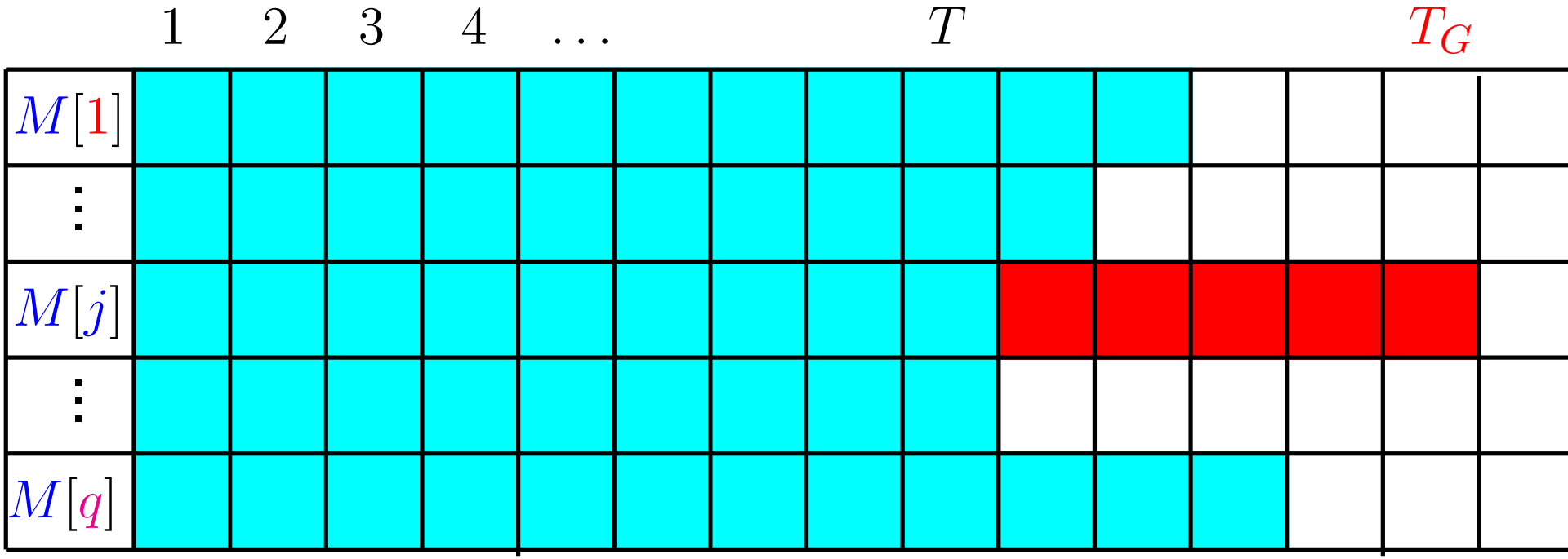
Fator de aproximação

T_G: conclusão do escalonamento obtido pelo algoritmo
i: última tarefa a concluir
j: máquina em que i está, a partir do instante T



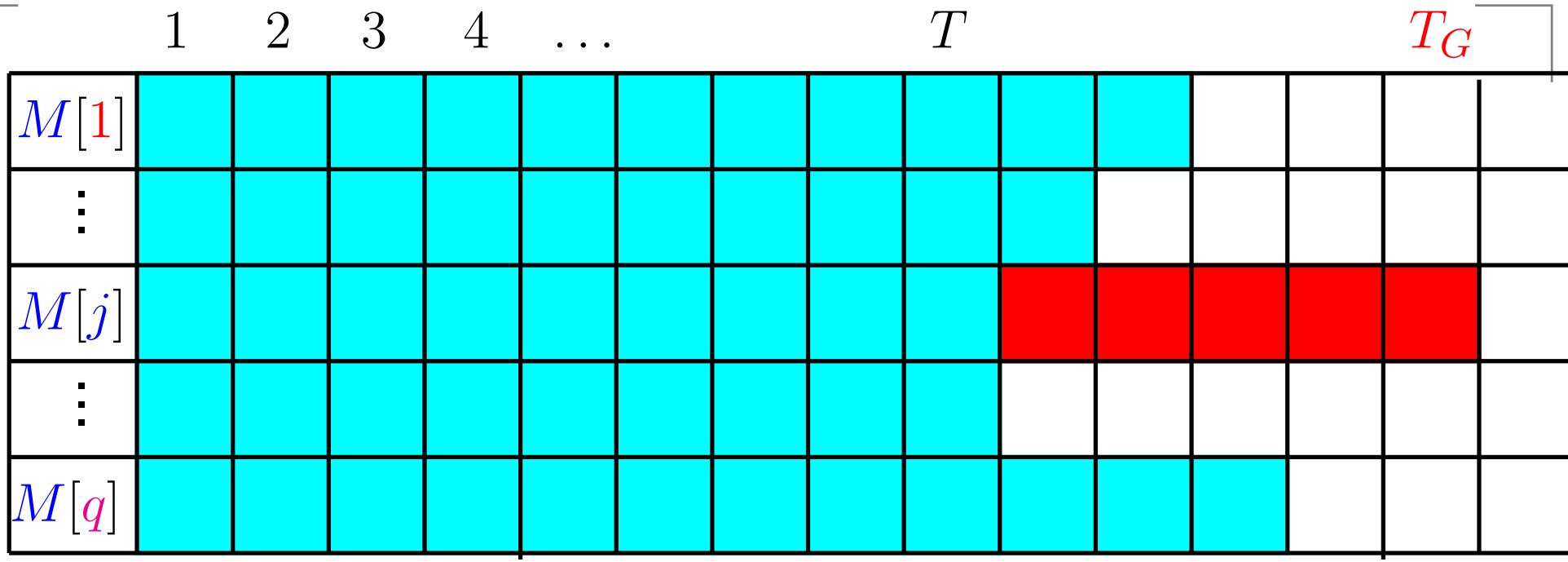
Fator de aproximação (cont.)

T_G: conclusão do escalonamento obtido pelo algoritmo
i: última tarefa a concluir
j: máquina em que i está, a partir do instante T



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

Fator de aproximação (cont.)



$$
\begin{aligned}
T_G &= T + d[i] \\
&\leq \frac{d[1] + \cdots + d[n]}{q} + d[i] \\
&\leq \frac{d[1] + \cdots + d[n]}{q} + \max\{d[1], \ldots, d[n]\} \\
&\leq \text{opt} + \text{opt} = 2\,\text{opt}
\end{aligned}
$$

Conclusão

ESCALONAMENTO-GRAHAM (q, n, d)

```
1   para j ← 1 até q faça
2       M[j] ← ∅
3       T[j] ← 0
4   para i ← 1 até n faça
5       seja k tal que T[k] é mínimo
6       M[k] ← M[k] ∪ {i}
7       T[k] ← T[k] + d[i]
8   devolva {M[1], ..., M[q]}
```

Teorema: O algoritmo ESCALONAMENTO-GRAHAM é uma 2-aproximação.

Uma melhora no algoritmo

Regra LPT: longest processing time rule

Ordene as tarefas pelo tempo de processamento, com as mais longas primeiro, e aplique ESCALONAMENTO-GRAHAM.

Uma melhora no algoritmo

Regra LPT: longest processing time rule

Ordene as tarefas pelo tempo de processamento, com as mais longas primeiro, e aplique ESCALONAMENTO-GRAHAM.

Mostre que o algoritmo resultante é uma $\frac{3}{2}$-aproximação.

Uma melhora no algoritmo

Regra LPT: longest processing time rule

Ordene as tarefas pelo tempo de processamento,
com as mais longas primeiro, e aplique
ESCALONAMENTO-GRAHAM.

Mostre que o algoritmo resultante é uma $\frac{3}{2}$-aproximação.

(Na verdade, ele é uma $\frac{4}{3}$-aproximação.)

Uma melhora no algoritmo

Regra LPT: longest processing time rule

Ordene as tarefas pelo tempo de processamento,
com as mais longas primeiro, e aplique
ESCALONAMENTO-GRAHAM.

Mostre que o algoritmo resultante é uma $\frac{3}{2}$-aproximação.

(Na verdade, ele é uma $\frac{4}{3}$-aproximação.)

Existe um PTAS para este problema.

PTAS: polynomial-time approximation scheme

Problema dos k-centros

Dados: grafo completo $G = (V, E)$, inteiro $k > 0$ e distâncias d_{ij} para cada i e j em V tais que $d_{ii} = 0$ para todo i, $d_{ij} = d_{ji}$ para todo i e j, e $d_{ij} \leq d_{i\ell} + d_{\ell j}$.

Problema dos k-centros

Dados: grafo completo $G = (V, E)$, inteiro $k > 0$ e distâncias d_{ij} para cada i e j em V tais que $d_{ii} = 0$ para todo i, $d_{ij} = d_{ji}$ para todo i e j, e $d_{ij} \leq d_{i\ell} + d_{\ell j}$.

S: conjunto não vazio de vértices

$d(i, S) := \min\{d(i, j) : j \in S\}$

raio de S: $\max\{d(i, S) : i \in V\}$

Problema dos k-centros

Dados: grafo completo $G = (V, E)$, inteiro $k > 0$ e distâncias d_{ij} para cada i e j em V tais que $d_{ii} = 0$ para todo i, $d_{ij} = d_{ji}$ para todo i e j, e $d_{ij} \leq d_{i\ell} + d_{\ell j}$.

S: conjunto não vazio de vértices

$d(i, S) := \min\{d(i, j) : j \in S\}$

raio de S: $\max\{d(i, S) : i \in V\}$

Objetivo: encontrar conjunto S com k vértices de V com raio mínimo.

Problema dos k-centros

Dados: grafo completo $G = (V, E)$, inteiro $k > 0$ e distâncias d_{ij} para cada i e j em V tais que $d_{ii} = 0$ para todo i, $d_{ij} = d_{ji}$ para todo i e j, e $d_{ij} \leq d_{i\ell} + d_{\ell j}$.

S: conjunto não vazio de vértices

$d(i, S) := \min\{d(i, j) : j \in S\}$

raio de S: $\max\{d(i, S) : i \in V\}$

Objetivo: encontrar conjunto S com k vértices de V com raio mínimo.

Vamos mostrar uma 2-aproximação para o problema.

Se soubéssemos o valor ótimo...

Vamos supor que $V = \{1, \ldots, n\}$ e seja r o valor ótimo.

INFORMADO (n, k, d)

1 $S \leftarrow \emptyset$ $V' \leftarrow V$
2 **enquanto** $V' \neq \emptyset$ **faça**
3 seja s um elemento arbitrário de V'
4 $S \leftarrow S \cup \{s\}$
5 remova de V' os vértices a distância até $2r$ de s
6 **devolva** S

Se soubéssemos o valor ótimo...

Vamos supor que $V = \{1, \ldots, n\}$ e seja r o valor ótimo.

INFORMADO (n, k, d)

1 $S \leftarrow \emptyset$ $V' \leftarrow V$
2 **enquanto** $V' \neq \emptyset$ **faça**
3 seja s um elemento arbitrário de V'
4 $S \leftarrow S \cup \{s\}$
5 remova de V' os vértices a distância até $2r$ de s
6 **devolva** S

O algoritmo acima usaria k centros.

Prova feita em aula.

Se soubéssemos o valor ótimo...

Vamos supor que $V = \{1, \ldots, n\}$ e seja r o valor ótimo.

INFORMADO (n, k, d)

1 $S \leftarrow \emptyset$ $V' \leftarrow V$
2 **enquanto** $V' \neq \emptyset$ **faça**
3 seja s um elemento arbitrário de V'
4 $S \leftarrow S \cup \{s\}$
5 remova de V' os vértices a distância até $2r$ de s
6 **devolva** S

O algoritmo acima usaria k centros.

Prova feita em aula.

Faça uma busca binária pelo valor ótimo.

Mas não precisa do valor ótimo...

Vamos supor que $V = \{1, \ldots, n\}$.

GULOSO (n, k, d)

1 $S \leftarrow \{1\}$
2 **enquanto** $|S| < k$ **faça**
3 $\quad j \leftarrow \arg\max\{d(\ell, S) : \ell \in V\}$
4 $\quad S \leftarrow S \cup \{j\}$
5 **devolva** S

Mas não precisa do valor ótimo...

Vamos supor que $V = \{1, \ldots, n\}$.

GULOSO (n, k, d)

```
1   S ← {1}
2   enquanto |S| < k faça
3       j ← arg max{d(ℓ, S) : ℓ ∈ V}
4       S ← S ∪ {j}
5   devolva S
```

Teorema: GULOSO é uma 2-aproximação para o problema dos k-centros.

Prova feita em aula.

Problema dos k-centros

Teorema: Se existe uma α-aproximação para o problema dos k-centros com $\alpha < 2$, então P $=$ NP.

Problema dos k-centros

Teorema: Se existe uma α-aproximação para o problema dos k-centros com $\alpha < 2$, então P $=$ NP.

Prova: Redução do conjunto dominante.

Problema dos k-centros

Teorema: Se existe uma α-aproximação para o problema dos k-centros com $\alpha < 2$, então P $=$ NP.

Prova: Redução do conjunto dominante.

Prova feita em aula.

Cobertura por conjuntos

Instância:

- conjunto base finito E
- coleção $\mathcal{S} = \{S_1, \ldots, S_m\}$ de subconjuntos de E
- custo $c_j > 0$ para cada S_j em $\mathcal{S}$

Cobertura por conjuntos

Instância:

- conjunto base finito E
- coleção $\mathcal{S} = \{S_1, \ldots, S_m\}$ de subconjuntos de E
- custo $c_j > 0$ para cada S_j em $\mathcal{S}$

Objetivo: encontrar cobertura $\mathcal{S}' \subseteq \mathcal{S}$ de E de custo mínimo.

Cobertura por conjuntos

Instância:

- conjunto base finito E
- coleção $\mathcal{S} = \{S_1, \ldots, S_m\}$ de subconjuntos de E
- custo $c_j > 0$ para cada S_j em $\mathcal{S}$

Objetivo: encontrar cobertura $\mathcal{S}' \subseteq \mathcal{S}$ de E de custo mínimo.

Relaxação linear: Dados E, $\mathcal{S}$ e c, encontrar x que

minimize $\sum_j c_j\, x_j$

sujeito a $\sum_{j:e \in S_j} x_j \geq 1$ para cada e em E

$x_j \geq 0$ para $j = 1, \ldots, m$

Cobertura por conjuntos

Instância:

- conjunto base finito E
- coleção $\mathcal{S} = \{S_1, \ldots, S_m\}$ de subconjuntos de E
- custo $c_j > 0$ para cada S_j em $\mathcal{S}$

Objetivo: encontrar cobertura $\mathcal{S}' \subseteq \mathcal{S}$ de E de custo mínimo.

Relaxação linear: Dados E, $\mathcal{S}$ e c, encontrar x que

minimize $\sum_j c_j\, x_j$

sujeito a $\sum_{j:e \in S_j} x_j \geq 1$ para cada e em E

$x_j \geq 0$ para $j = 1, \ldots, m$

Este é o programa primal, que denotaremos por (P).

Arredondamento determinístico

ARREDDET $(E, \mathcal{S}, c)$ $\quad \triangleright\ \mathcal{S} = \{S_1, \ldots, S_m\}$

1 $x^* \leftarrow$ solução da relaxação linear (P)
2 **para** cada e em E **faça**
3 $\quad f_e \leftarrow |\{j : e \in S_j\}|$
4 $f \leftarrow \max\{f_e : e \in E\}$
5 $I \leftarrow \{j : x^*_j \geq 1/f\}$
6 **devolva** I

Arredondamento determinístico

ARREDDET $(E, \mathcal{S}, c)$ $\quad \triangleright\ \mathcal{S} = \{S_1, \ldots, S_m\}$

1 $x^* \leftarrow$ solução da relaxação linear (P)
2 **para** cada e em E **faça**
3 $\quad f_e \leftarrow |\{j : e \in S_j\}|$
4 $f \leftarrow \max\{f_e : e \in E\}$
5 $I \leftarrow \{j : x^*_j \geq 1/f\}$
6 **devolva** I

Como (P) pode ser resolvido em tempo polinomial,
o consumo de tempo do algoritmo é polinomial.

Arredondamento determinístico

ARREDDET $(E, \mathcal{S}, c)$ $\quad \triangleright\ \mathcal{S} = \{S_1, \ldots, S_m\}$

1 $x^* \leftarrow$ solução da relaxação linear (P)
2 **para** cada e em E **faça**
3 $\quad f_e \leftarrow |\{j : e \in S_j\}|$
4 $f \leftarrow \max\{f_e : e \in E\}$
5 $I \leftarrow \{j : x^*_j \geq 1/f\}$
6 **devolva** I

Como (P) pode ser resolvido em tempo polinomial, o consumo de tempo do algoritmo é polinomial.

Mas I é mesmo uma cobertura?

Arredondamento determinístico

ARREDDET $(E, \mathcal{S}, c)$ $\quad \triangleright\ \mathcal{S} = \{S_1, \ldots, S_m\}$

1 $x^* \leftarrow$ solução da relaxação linear (P)
2 **para** cada e em E **faça**
3 $\quad f_e \leftarrow |\{j : e \in S_j\}|$
4 $f \leftarrow \max\{f_e : e \in E\}$
5 $I \leftarrow \{j : x^*_j \geq 1/f\}$
6 **devolva** I

Como (P) pode ser resolvido em tempo polinomial, o consumo de tempo do algoritmo é polinomial.

Mas I é mesmo uma cobertura?

Como $\sum_{j:e\in S_j} x^*_j \geq 1$ tem exatamente $f_e \leq f$ termos, um deles tem que valer pelo menos $1/f$.

Arredondamento determinístico

ARREDDET $(E, \mathcal{S}, c)$ $\quad \triangleright\ \mathcal{S} = \{S_1, \ldots, S_m\}$

1 $x^* \leftarrow$ solução da relaxação linear (P)
2 **para** cada e em E **faça**
3 $\quad f_e \leftarrow |\{j : e \in S_j\}|$
4 $f \leftarrow \max\{f_e : e \in E\}$
5 $I \leftarrow \{j : x^*_j \geq 1/f\}$
6 **devolva** I

Lema: I é uma cobertura.

Prova: Como $\sum_{j:e\in S_j} x^*_j \geq 1$ tem exatamente $f_e \leq f$ termos, um deles tem que valer pelo menos $1/f$. ∎

Arredondamento determinístico

ARREDDET $(E, \mathcal{S}, c)$ $\quad \triangleright\ \mathcal{S} = \{S_1, \ldots, S_m\}$

1 $x^* \leftarrow$ solução da relaxação linear (P)
2 **para** cada e em E **faça**
3 $\quad f_e \leftarrow |\{j : e \in S_j\}|$
4 $f \leftarrow \max\{f_e : e \in E\}$
5 $I \leftarrow \{j : x^*_j \geq 1/f\}$
6 **devolva** I

Lema: I é uma cobertura.

Prova: Como $\sum_{j:e\in S_j} x^*_j \geq 1$ tem exatamente $f_e \leq f$ termos, um deles tem que valer pelo menos $1/f$. ■

Teorema: ARREDDET é uma f-aproximação.

Prova: $c(I) = \sum_{j\in I} c_j \leq \sum_j (c_j\, f\, x^*_j) = f\, z^*_{\mathrm{LP}} \leq f\, \mathrm{opt}$. ■

Arredondamento de solução dual

ARREDDUAL $(E, \mathcal{S}, c)$ $\quad \triangleright\ \mathcal{S} = \{S_1, \ldots, S_m\}$

1 $y^* \leftarrow$ solução do dual da relaxação linear (P)

2 $I' \leftarrow \{j : \sum_{e \in S_j} y^*_e = c_j\}$

3 **devolva** I'

Arredondamento de solução dual

ARREDDUAL $(E, \mathcal{S}, c)$ $\quad \triangleright\ \mathcal{S} = \{S_1, \ldots, S_m\}$

1 $y^* \leftarrow$ solução do dual da relaxação linear (P)

2 $I' \leftarrow \{j : \sum_{e \in S_j} y^*_e = c_j\}$

3 **devolva** I'

O consumo de tempo é polinomial.

Arredondamento de solução dual

ARREDDUAL $(E, \mathcal{S}, c)$ $\quad \triangleright\ \mathcal{S} = \{S_1, \ldots, S_m\}$

1 $y^* \leftarrow$ solução do dual da relaxação linear (P)
2 $I' \leftarrow \{j : \sum_{e \in S_j} y^*_e = c_j\}$
3 **devolva** I'

O consumo de tempo é polinomial.

Lema: I é uma cobertura.

Prova feita na aula.

Arredondamento de solução dual

ARREDDUAL $(E, \mathcal{S}, c)$ $\quad \triangleright\ \mathcal{S} = \{S_1, \ldots, S_m\}$

1 $y^* \leftarrow$ solução do dual da relaxação linear (P)

2 $I' \leftarrow \{j : \sum_{e \in S_j} y^*_e = c_j\}$

3 **devolva** I'

O consumo de tempo é polinomial.

Lema: I é uma cobertura.

Prova feita na aula.

Teorema: ARREDDUAL é uma f-aproximação.

Prova feita na aula.

Algoritmo primal-dual

PRIMALDUAL $(E, \mathcal{S}, c)$ $\quad \triangleright\ \mathcal{S} = \{S_1, \ldots, S_m\}$

1 **para** cada e em E **faça** $y_e \leftarrow 0$
2 $I \leftarrow \{j : \sum_{e \in S_j} y_e = c_j\}$
3 **enquanto** existe e' que não é coberto por I **faça**
4 $\quad \epsilon \leftarrow \min\{c_j - \sum_{e \in S_j} y_e : j$ tal que $e' \in S_j\}$
5 $\quad y_{e'} \leftarrow y_{e'} + \epsilon$
6 $\quad I \leftarrow \{j : \sum_{e \in S_j} y_e = c_j\}$
7 **devolva** I

Algoritmo primal-dual

PRIMALDUAL $(E, \mathcal{S}, c)$ $\quad \triangleright\ \mathcal{S} = \{S_1, \ldots, S_m\}$

1 **para** cada e em E **faça** $y_e \leftarrow 0$
2 $I \leftarrow \{j : \sum_{e \in S_j} y_e = c_j\}$
3 **enquanto** existe e' que não é coberto por I **faça**
4 $\quad \epsilon \leftarrow \min\{c_j - \sum_{e \in S_j} y_e : j$ tal que $e' \in S_j\}$
5 $\quad y_{e'} \leftarrow y_{e'} + \epsilon$
6 $\quad I \leftarrow \{j : \sum_{e \in S_j} y_e = c_j\}$
7 **devolva** I

Consumo de tempo polinomial, sem resolução de PL!

Algoritmo primal-dual

PRIMALDUAL $(E, \mathcal{S}, c)$ $\quad \triangleright\ \mathcal{S} = \{S_1, \ldots, S_m\}$

1 **para** cada e em E **faça** $y_e \leftarrow 0$
2 $I \leftarrow \{j : \sum_{e \in S_j} y_e = c_j\}$
3 **enquanto** existe e' que não é coberto por I **faça**
4 $\quad \epsilon \leftarrow \min\{c_j - \sum_{e \in S_j} y_e : j$ tal que $e' \in S_j\}$
5 $\quad y_{e'} \leftarrow y_{e'} + \epsilon$
6 $\quad I \leftarrow \{j : \sum_{e \in S_j} y_e = c_j\}$
7 **devolva** I

Consumo de tempo polinomial, sem resolução de PL!

Claro que I é uma cobertura.

Algoritmo primal-dual

PRIMALDUAL $(E, \mathcal{S}, c)$ $\quad \triangleright\ \mathcal{S} = \{S_1, \ldots, S_m\}$

1 **para** cada e em E **faça** $y_e \leftarrow 0$
2 $I \leftarrow \{j : \sum_{e \in S_j} y_e = c_j\}$
3 **enquanto** existe e' que não é coberto por I **faça**
4 $\quad \epsilon \leftarrow \min\{c_j - \sum_{e \in S_j} y_e : j$ tal que $e' \in S_j\}$
5 $\quad y_{e'} \leftarrow y_{e'} + \epsilon$
6 $\quad I \leftarrow \{j : \sum_{e \in S_j} y_e = c_j\}$
7 **devolva** I

Consumo de tempo polinomial, sem resolução de PL!

Claro que I é uma cobertura.

Teorema: PRIMALDUAL é uma f-aproximação.

A mesma prova do ARREDDUAL funciona!

Algoritmo guloso

GULOSO $(E, \mathcal{S}, c)$ $\quad \triangleright\ \mathcal{S} = \{S_1, \ldots, S_m\}$

1 $I \leftarrow \emptyset$
2 **para** $j \leftarrow 1$ **até** m **faça** $\hat{S}_j \leftarrow S_j$
3 **enquanto** I não é uma cobertura **faça**
4 $\quad k \leftarrow \arg\min\{\frac{c_j}{|\hat{S}_j|} : j$ é tal que $\hat{S}_j \neq \emptyset\}$
5 $\quad I \leftarrow I \cup \{k\}$
6 $\quad$ **para** $j \leftarrow 1$ **até** m **faça**
7 $\quad\quad \hat{S}_j \leftarrow \hat{S}_j \setminus S_k$
8 **devolva** I

Algoritmo guloso

```
GULOSO (E, S, c)        ▷ S = {S_1, ..., S_m}
 1   I ← ∅
 2   para j ← 1 até m faça Ŝ_j ← S_j
 3   enquanto I não é uma cobertura faça
 4         k ← arg min{ c_j / |Ŝ_j| : j é tal que Ŝ_j ≠ ∅ }
 5         I ← I ∪ {k}
 6         para j ← 1 até m faça
 7               Ŝ_j ← Ŝ_j \ S_k
 8   devolva I
```

Consumo de tempo polinomial, sem resolução de PL!

Algoritmo guloso

```
GULOSO (E, S, c)        ▷ S = {S_1, ..., S_m}
 1   I ← ∅
 2   para j ← 1 até m faça Ŝ_j ← S_j
 3   enquanto I não é uma cobertura faça
 4         k ← arg min{ c_j / |Ŝ_j| : j é tal que Ŝ_j ≠ ∅ }
 5         I ← I ∪ {k}
 6         para j ← 1 até m faça
 7               Ŝ_j ← Ŝ_j \ S_k
 8   devolva I
```

Consumo de tempo polinomial, sem resolução de PL!

Claro que I é uma cobertura.

Algoritmo guloso

```
GULOSO (E, S, c)        ▷ S = {S_1, ..., S_m}
 1   I ← ∅
 2   para j ← 1 até m faça Ŝ_j ← S_j
 3   enquanto I não é uma cobertura faça
 4         k ← arg min{ c_j / |Ŝ_j| : j é tal que Ŝ_j ≠ ∅ }
 5         I ← I ∪ {k}
 6         para j ← 1 até m faça
 7               Ŝ_j ← Ŝ_j \ S_k
 8   devolva I
```

Consumo de tempo polinomial, sem resolução de PL!

Claro que I é uma cobertura.

Teorema: GULOSO é uma H_n-aproximação, onde $n = |E|$.