

# 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  $Sol(I)$  de soluções viáveis e

uma função  $val(S)$  para cada solução  $S$  em  $Sol(I)$

# Problemas de Otimização Combinatória

conjunto de **instâncias**

para cada instância  $I$ ,

um conjunto  $Sol(I)$  de soluções viáveis e

uma função  $val(S)$  para cada solução  $S$  em  $Sol(I)$

Se  $Sol(I)$  é vazio,  $I$  é **inviável**, caso contrário,  $I$  é **viável**.

# Problemas de Otimização Combinatória

conjunto de **instâncias**

para cada instância  $I$ ,

um conjunto  $Sol(I)$  de soluções viáveis e

uma função  $val(S)$  para cada solução  $S$  em  $Sol(I)$

Se  $Sol(I)$  é vazio,  $I$  é **inviável**, caso contrário,  $I$  é **viável**.

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

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

# Problemas de Otimização Combinatória

conjunto de **instâncias**

para cada instância  $I$ ,

um conjunto  $Sol(I)$  de soluções viáveis e

uma função  $val(S)$  para cada solução  $S$  em  $Sol(I)$

Se  $Sol(I)$  é vazio,  $I$  é **inviável**, caso contrário,  $I$  é **viável**.

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

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

$opt(I)$ : 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

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

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

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

para toda instância  $I$  do problema (de minimização).

$A$  é uma  $\alpha$ -aproximação e

$\alpha$  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

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

para toda instância  $I$  do problema (de minimização).

$A$  é uma  $\alpha$ -aproximação e

$\alpha$  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

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

para toda instância  $I$  do problema (de minimização).

$A$  é uma  $\alpha$ -aproximação e

$\alpha$  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

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

para toda instância  $I$  do problema (de minimização).

$A$  é uma  $\alpha$ -aproximação e

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



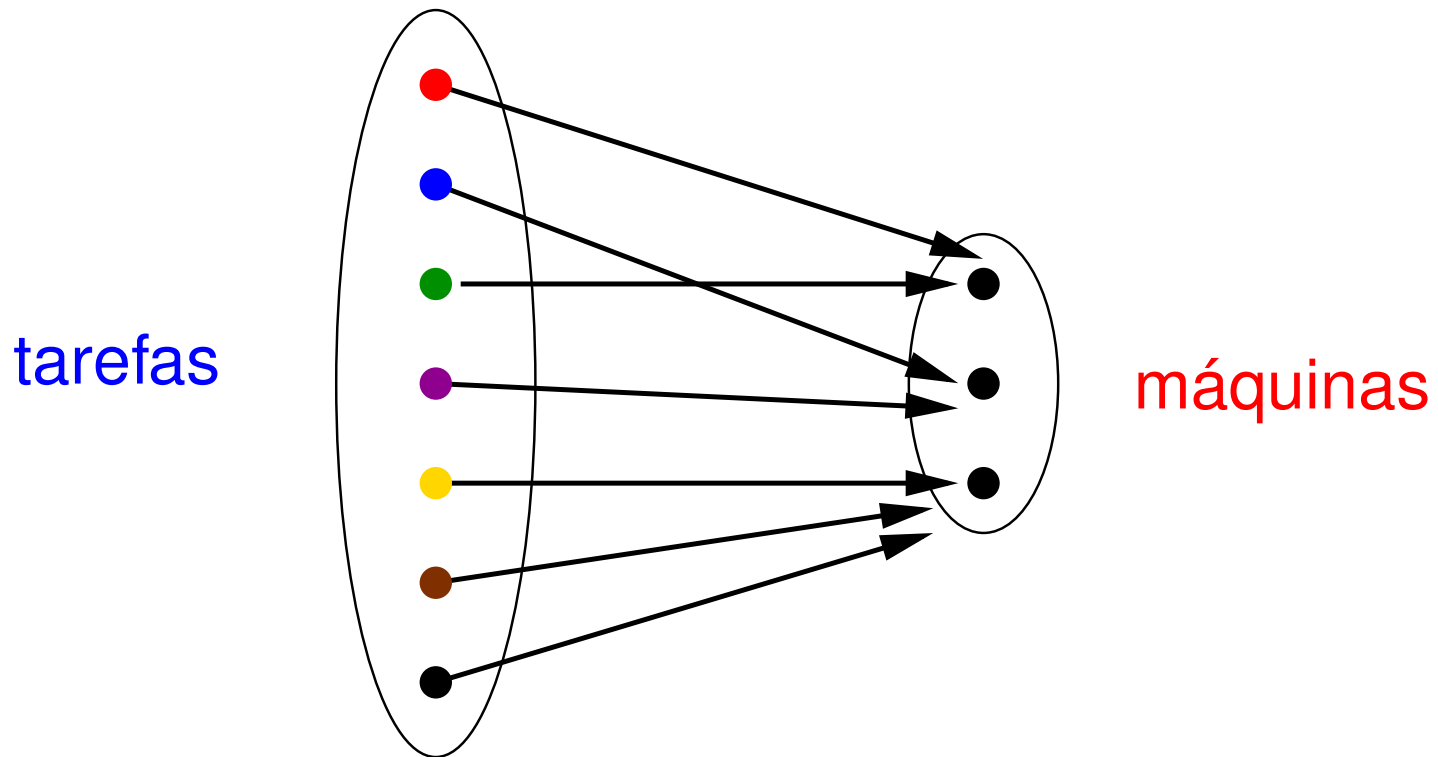
# Escalonamento de máquinas idênticas

Dados:  $q$  máquinas

$n$  tarefas








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

escalonamento: partição  $\{M[1], \dots, M[q]\}$  de  $\{1, \dots, n\}$



# Exemplo 1

$$q = 3 \quad n = 7$$

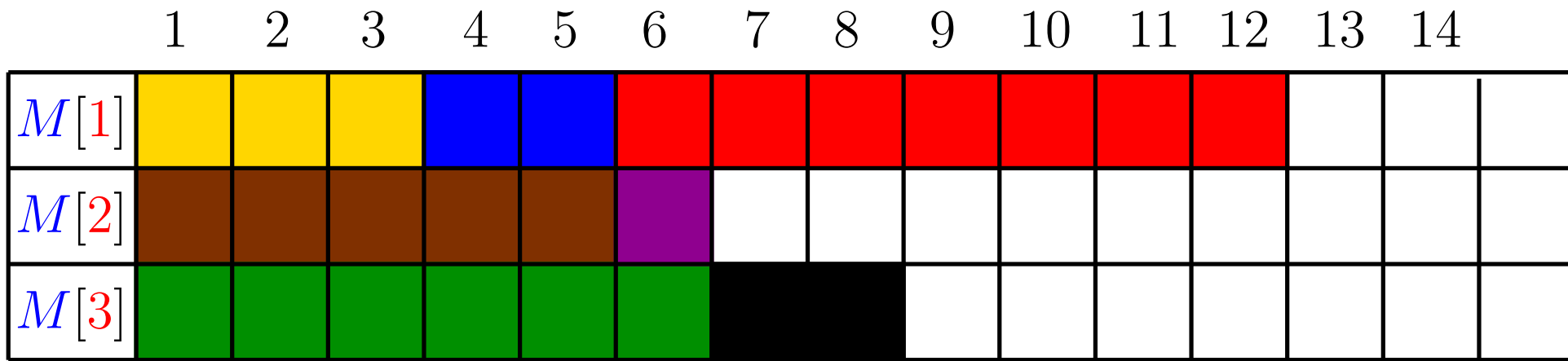
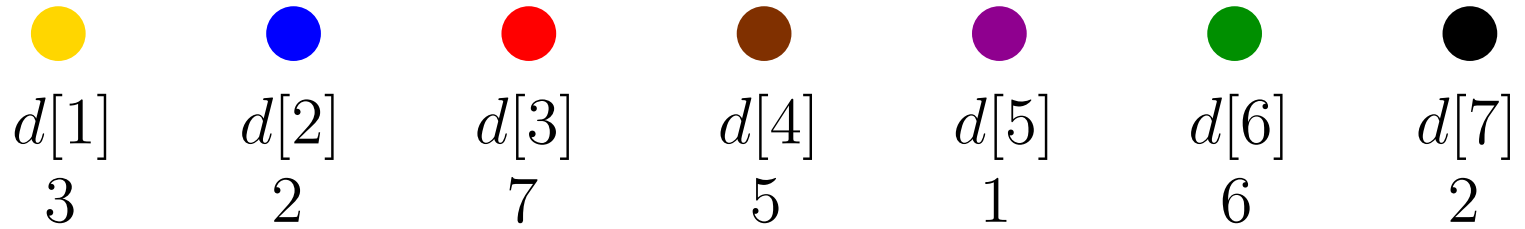
						
$d[1]$	$d[2]$	$d[3]$	$d[4]$	$d[5]$	$d[6]$	$d[7]$
3	2	7	5	1	6	2

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$M[1]$	yellow	yellow	yellow	brown	brown	brown	brown	brown	black	black				
$M[2]$	blue	blue	purple											
$M[3]$	red	red	red	red	red	red	red	green	green	green	green	green	green	

$\{\{1, 4, 7\}, \{2, 5\}, \{3, 6\}\} \Rightarrow$  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**.



$d[1]$   
3



$d[2]$   
2



$d[3]$   
7



$d[4]$   
5



$d[5]$   
1



$d[6]$   
6



$d[7]$   
2

1

2

3

4

5

6

7

8

9

10

11

12

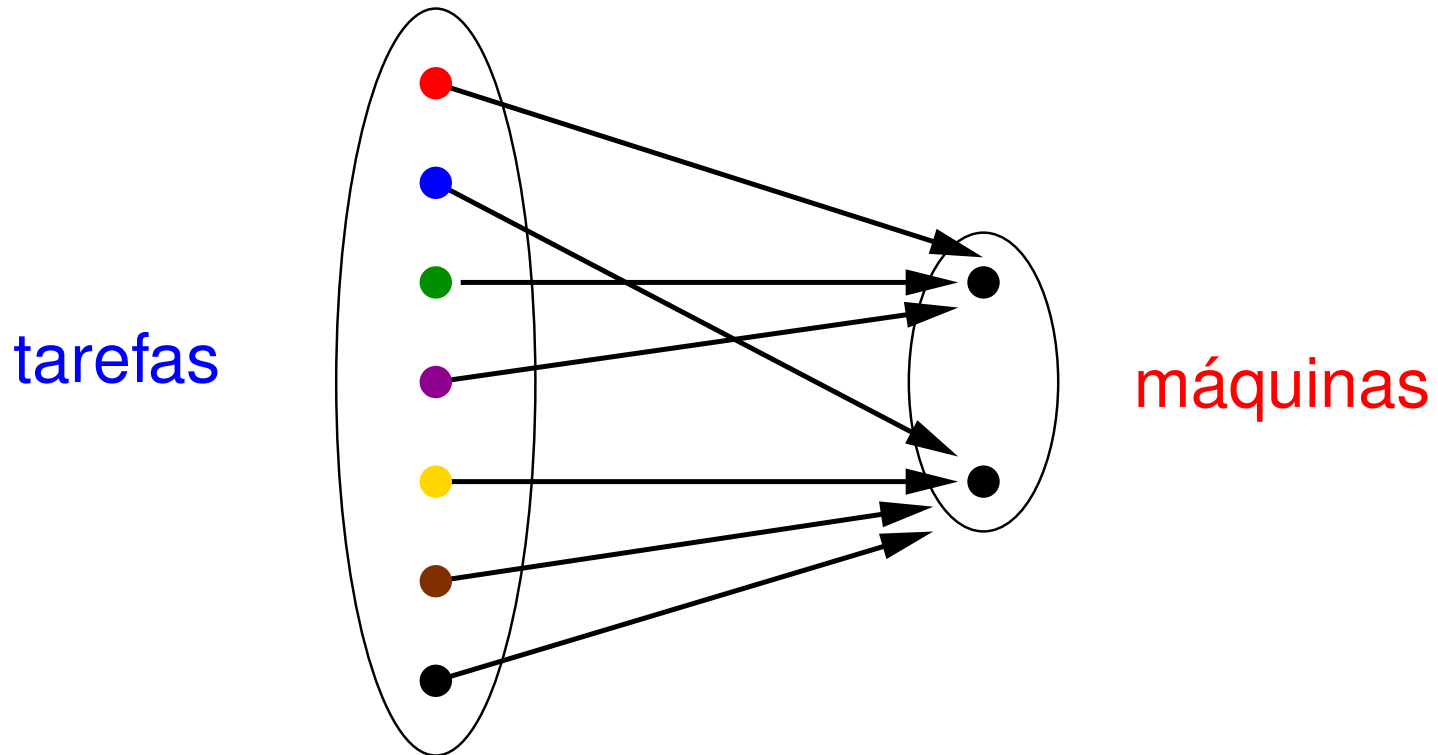
13

14

$M[1]$	Yellow	Yellow	Yellow	Brown	Brown	Brown	Brown	Brown							
$M[2]$	Blue	Blue	Red	Red	Red	Red	Red	Red	Red						
$M[3]$	Purple	Green	Green	Green	Green	Green	Green	Black	Black						

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

# NP-difícil mesmo para $q = 2$



**Algoritmo:** testa todo  $M[1] \subseteq \{1, \dots, n\}$  e escolhe melhor  $2^n$  subconjuntos  $\Rightarrow$  **exponencial**

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

# Algoritmo de Graham

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



$d[1]$   
3



$d[2]$   
2



$d[3]$   
7



$d[4]$   
5



$d[5]$   
1



$d[6]$   
6



$d[7]$   
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.



$d[1]$   
3



$d[2]$   
2



$d[3]$   
7



$d[4]$   
5



$d[5]$   
1



$d[6]$   
6



$d[7]$   
2

1

2

3

4

5

6

7

8

9

10

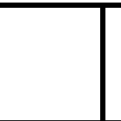
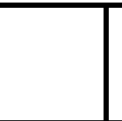
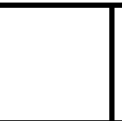
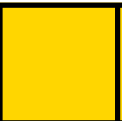
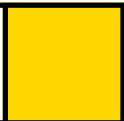
11

12

13

14

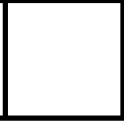
$M[1]$



$M[2]$



$M[3]$



# Algoritmo de Graham

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



$d[1]$   
3



$d[2]$   
2



$d[3]$   
7



$d[4]$   
5



$d[5]$   
1



$d[6]$   
6



$d[7]$   
2

1

2

3

4

5

6

7

8

9

10

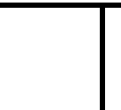
11

12

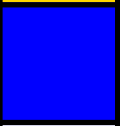
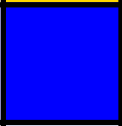
13

14

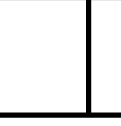
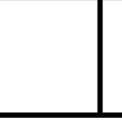
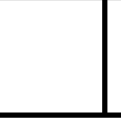
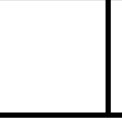
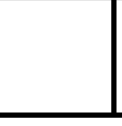
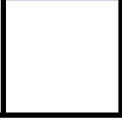
$M[1]$



$M[2]$



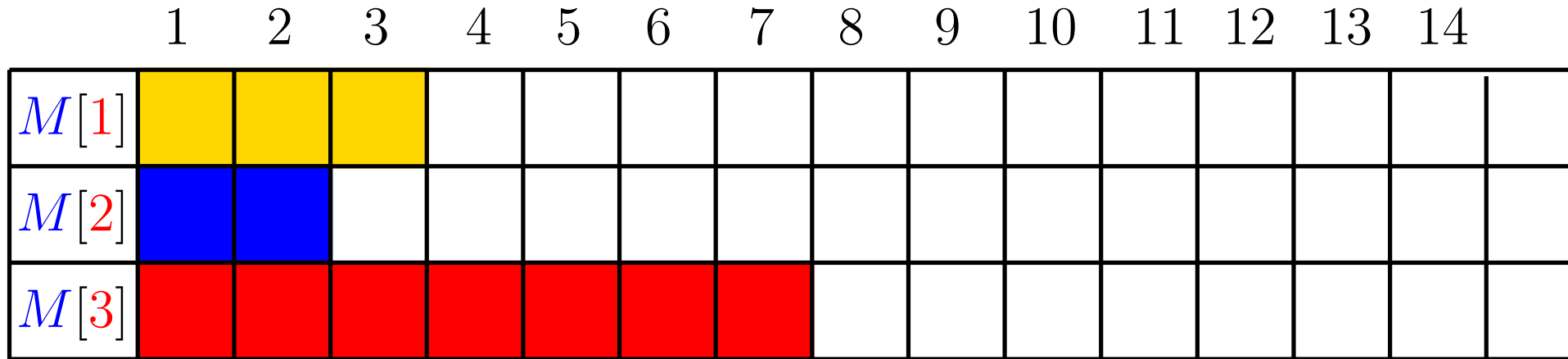
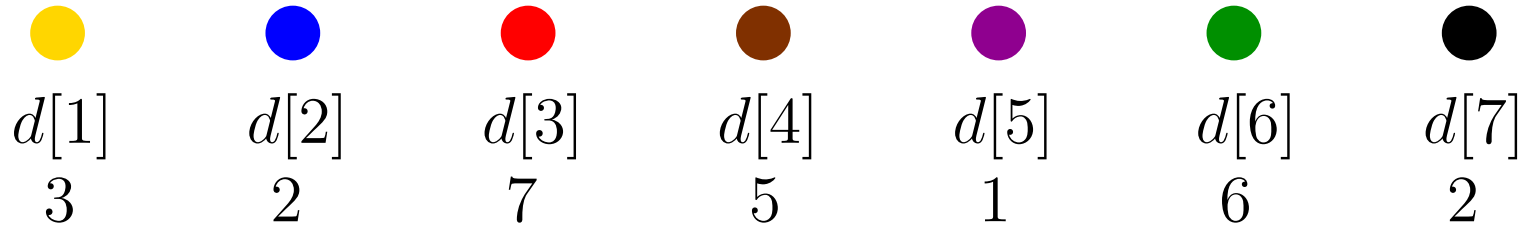
$M[3]$





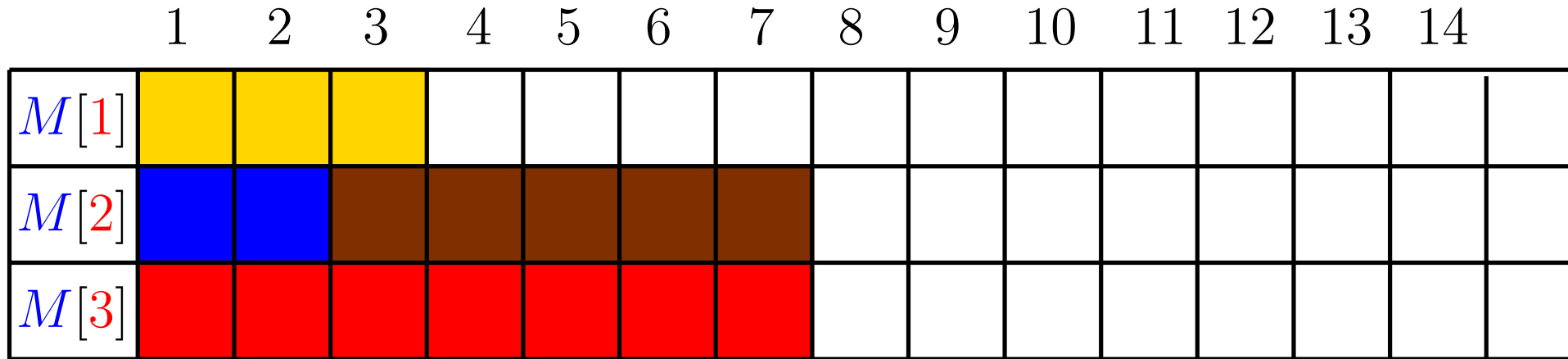
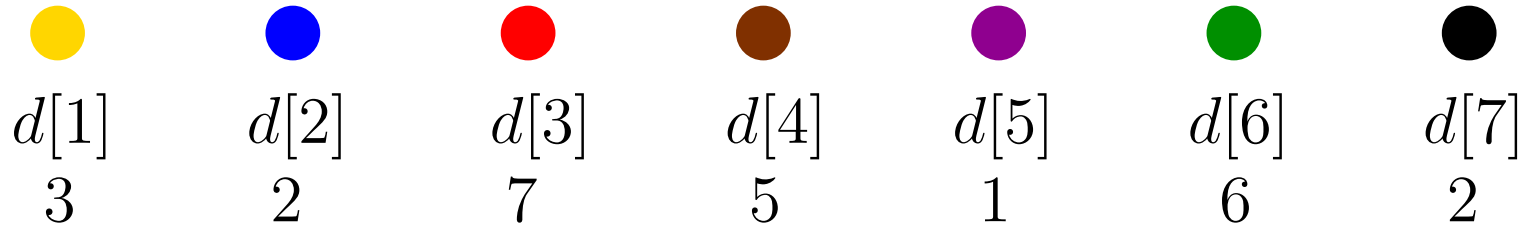
# Algoritmo de Graham

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



# 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.



$d[1]$   
3



$d[2]$   
2



$d[3]$   
7



$d[4]$   
5



$d[5]$   
1



$d[6]$   
6



$d[7]$   
2

1

2

3

4

5

6

7

8

9

10

11

12

13

14

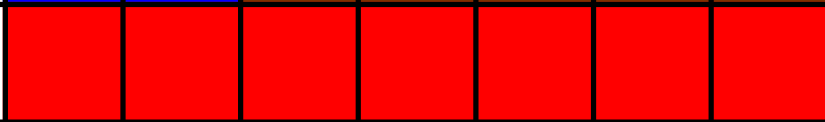
$M[1]$



$M[2]$

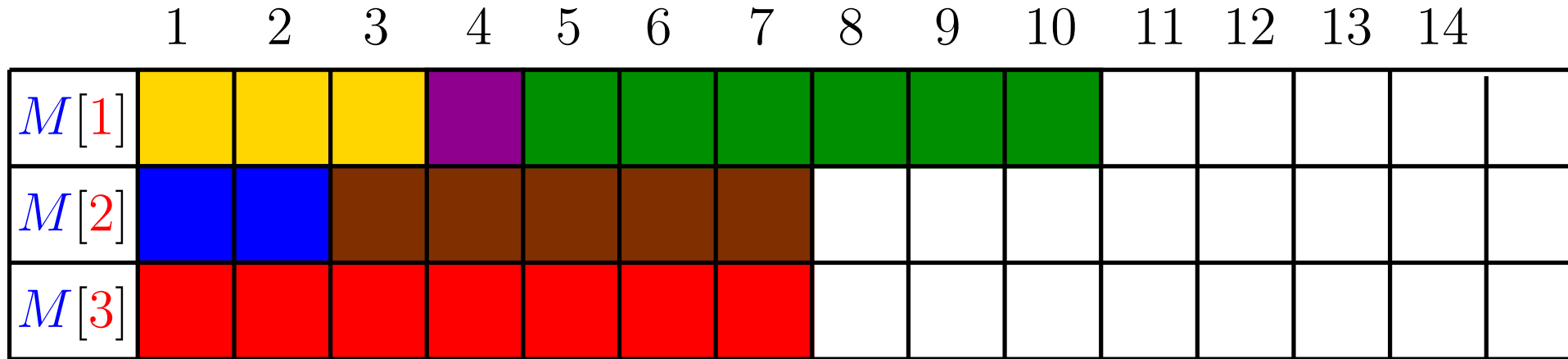
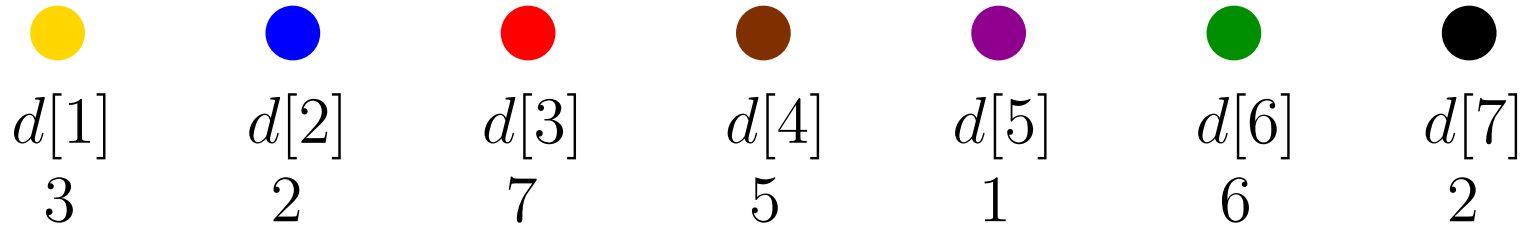


$M[3]$



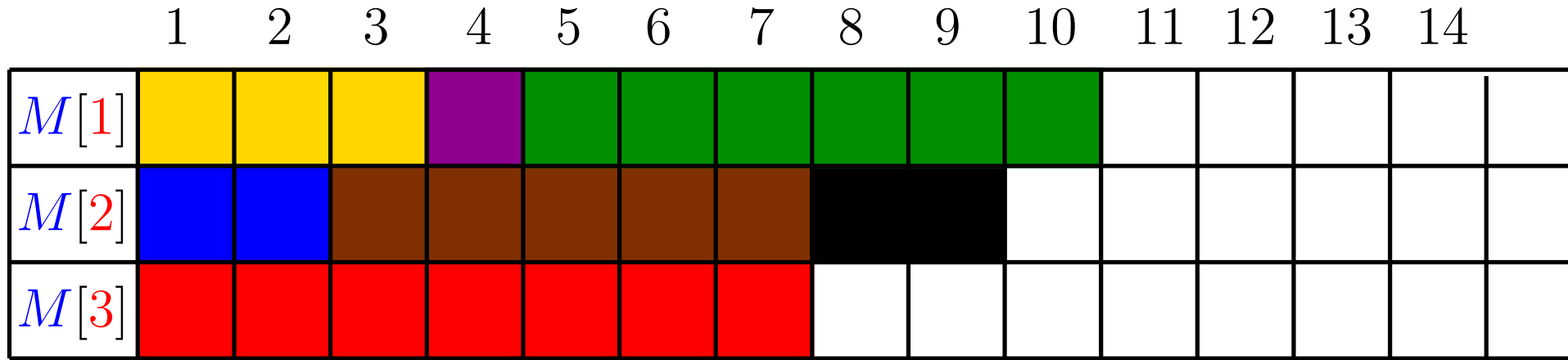
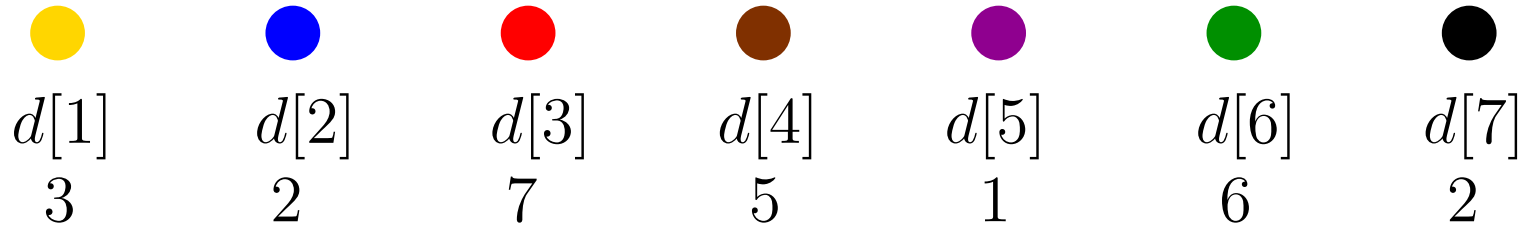
# Algoritmo de Graham

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



# 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 escalonamento de  $\{1, \dots, n\}$  em  $q$  máquinas.

## ESCALONAMENTO-GRAHAM $(q, n, d)$

```
1  para  $j \leftarrow 1$  até  $q$  faça
2       $M[j] \leftarrow \emptyset$ 
3       $T[j] \leftarrow 0$ 
4  para  $i \leftarrow 1$  até  $n$  faça
5      seja  $k$  tal que  $T[k]$  é mínimo
6       $M[k] \leftarrow M[k] \cup \{i\}$ 
7       $T[k] \leftarrow T[k] + d[i]$ 
8  devolva  $\{M[1], \dots, M[q]\}$ 
```

# Delimitações para $\text{opt}$

$\text{opt}$  = menor tempo de conclusão de um escalonamento

- Duração da tarefa mais longa:

$$\text{opt} \geq \max\{d[1], d[2], \dots, d[n]\}$$

- Distribuição balanceada:

$$\text{opt} \geq \frac{d[1] + d[2] + \dots + d[n]}{q}$$









# Conclusão

## ESCALONAMENTO-GRAHAM $(q, n, d)$

- 1 **para**  $j \leftarrow 1$  **até**  $q$  **faça**
- 2      $M[j] \leftarrow \emptyset$
- 3      $T[j] \leftarrow 0$
- 4 **para**  $i \leftarrow 1$  **até**  $n$  **faça**
- 5     **seja**  $k$  **tal que**  $T[k]$  **é mínimo**
- 6      $M[k] \leftarrow M[k] \cup \{i\}$
- 7      $T[k] \leftarrow T[k] + d[i]$
- 8 **devolva**  $\{M[1], \dots, M[q]\}$

**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**.

**Exercício:**

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**.

**Exercício:**

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**.

**Exercício:**

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  $d_{ij} \leq d_{il} + d_{lj}$ .

$S$ : 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  $d_{ij} \leq d_{il} + d_{lj}$ .

$S$ : 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  $V$  e 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  $d_{ij} \leq d_{il} + d_{lj}$ .

$S$ : conjunto não vazio de vértices

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

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

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

Vamos mostrar uma 2-aproximação para o problema.

# Se soubéssemos o valor ótimo...

Vamos supor que  $V = \{1, \dots, n\}$  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  $V = \{1, \dots, n\}$  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 no máximo  $k$  centros.

Prova feita em aula.

# Se soubéssemos o valor ótimo...

Vamos supor que  $V = \{1, \dots, n\}$  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 no máximo  $k$  centros.

Prova feita em aula.

Faça uma busca binária pelo valor ótimo.

# Mas não precisa do valor ótimo...

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

GULOSO ( $n, k, d$ )

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

# Mas não precisa do valor ótimo...

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

GULOSO ( $n, k, d$ )

- 1  $S \leftarrow \{1\}$
- 2 **enquanto**  $|S| < k$  **faça**
- 3      $j \leftarrow \arg \max\{d(\ell, S) : \ell \in V\}$
- 4      $S \leftarrow S \cup \{j\}$
- 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  $\alpha$ -aproximação para o problema dos  $k$ -centros com  $\alpha < 2$ , então  $P = NP$ .

# Problema dos $k$ -centros

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

**Prova:** Redução do conjunto dominante.

Prova feita em aula.