AULA 10

Método guloso

Sec 4.1 e 4.2 do KT e 16.1 do CLRS

Algoritmos gulosos

Algoritmo guloso

- procura ótimo local e acaba obtendo ótimo global

Algoritmos gulosos

Algoritmo guloso

- procura ótimo local e acaba obtendo ótimo global

Costuma ser

- muito simples e intuitivo
- muito eficiente
- difícil de provar que está correto

Algoritmos gulosos

Algoritmo guloso

- procura ótimo local e acaba obtendo ótimo global

Costuma ser

- muito simples e intuitivo
- muito eficiente
- difícil de provar que está correto

Problema precisa ter

- subestrutura ótima (como na programação dinâmica)
- propriedade da escolha gulosa (*greedy-choice property*)

Problema dos intervalos disjuntos

Problema: Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$,
encontrar uma coleção máxima de intervalos dois a dois disjuntos.

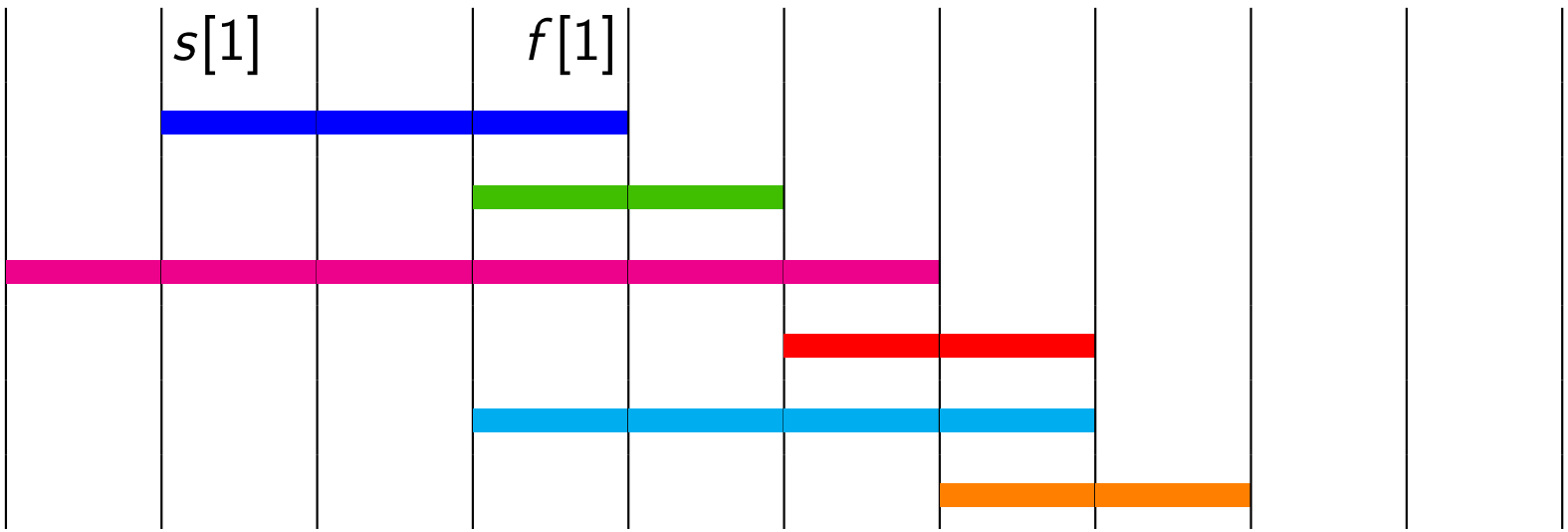
Solução é um subconjunto A de $\{1, \ldots, n\}$.

Problema dos intervalos disjuntos

Problema: Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$,
encontrar uma coleção máxima de intervalos dois a dois disjuntos.

Solução é um subconjunto A de $\{1, \ldots, n\}$.

Exemplo:

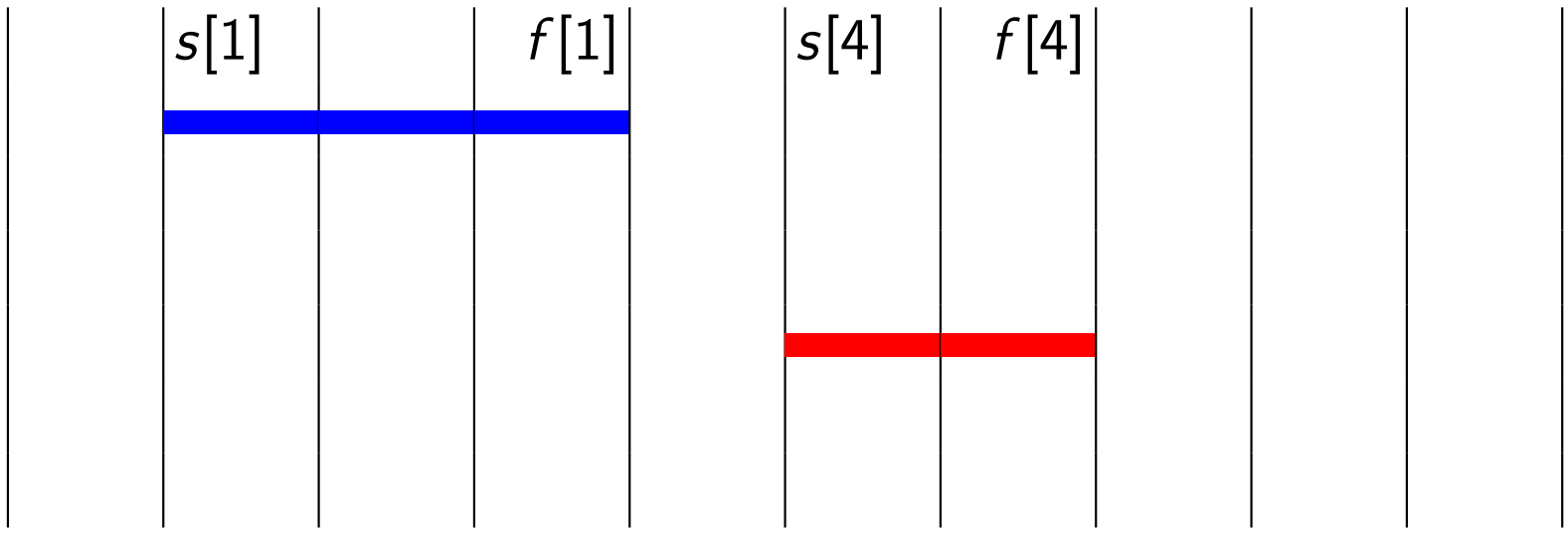


Problema dos intervalos disjuntos

Problema: Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$,
encontrar uma coleção máxima de intervalos dois a dois disjuntos.

Solução é um subconjunto A de $\{1, \ldots, n\}$.

Solução:



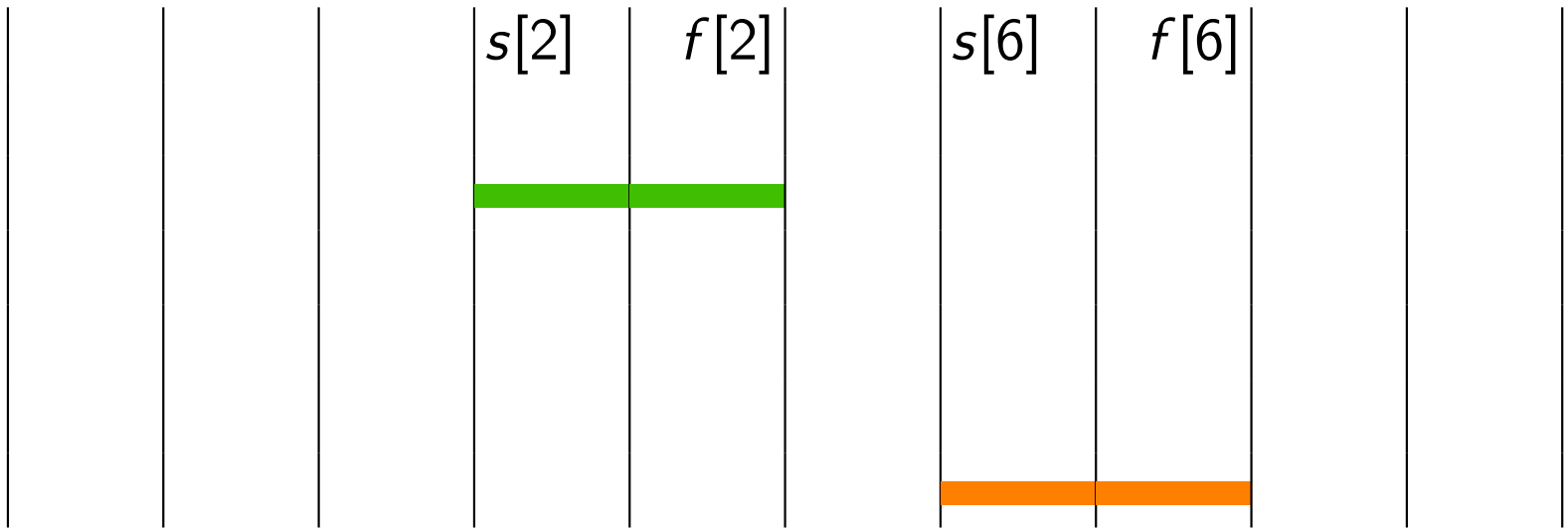
$A = \{1, 4\}$

Problema dos intervalos disjuntos

Problema: Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$,
encontrar uma coleção máxima de intervalos dois a dois disjuntos.

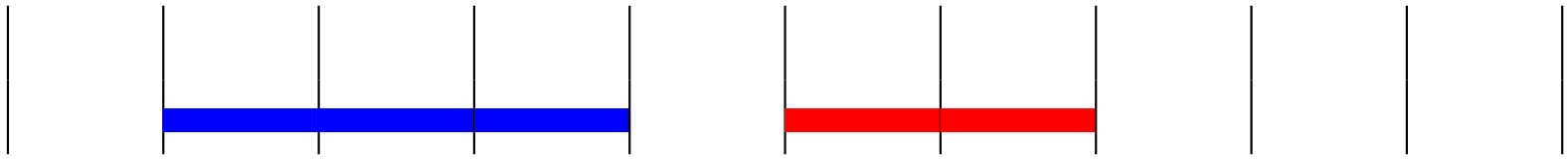
Solução é um subconjunto A de $\{1, \ldots, n\}$.

Solução:



$A = \{2, 6\}$

Motivação



Se cada intervalo é uma "atividade",
queremos coleção disjunta máxima de atividades compatíveis
(i e j com $s[i] \leq s[j]$ são compatíveis se $f[i] \leq s[j]$).

Nome no CLRS: Activity Selection Problem

Algoritmo guloso

```
GULOSO (s, f, n)
1   A ← ∅
2   R ← {1, ..., n}
3   enquanto R ≠ ∅ faça
4         escolha por um critério guloso um intervalo i de R
5         A ← A ∪ {i}
6         R ← R \ {j ∈ R : j intersecta i}
7   devolva A
```

Algoritmo guloso

GULOSO (s, f, n)
1 $A \leftarrow \emptyset$
2 $R \leftarrow \{1, \ldots, n\}$
3 **enquanto** $R \neq \emptyset$ **faça**
4 escolha por um critério guloso um intervalo i de R
5 $A \leftarrow A \cup \{i\}$
6 $R \leftarrow R \setminus \{j \in R : j \text{ intersecta } i\}$
7 **devolva** A

Invariante: todo intervalo de R é compatível com os de A.

Algoritmo guloso

GULOSO (s, f, n)
1 $A \leftarrow \emptyset$
2 $R \leftarrow \{1, \ldots, n\}$
3 **enquanto** $R \neq \emptyset$ **faça**
4 escolha por um critério guloso um intervalo i de R
5 $A \leftarrow A \cup \{i\}$
6 $R \leftarrow R \setminus \{j \in R : j \text{ intersecta } i\}$
7 **devolva** A

Invariante: todo intervalo de R é compatível com os de A.

Claro que A é uma coleção de intervalos compatíveis.

Algoritmo guloso

```
GULOSO (s, f, n)
1   A ← ∅
2   R ← {1, ..., n}
3   enquanto R ≠ ∅ faça
4         escolha por um critério guloso um intervalo i de R
5         A ← A ∪ {i}
6         R ← R \ {j ∈ R : j intersecta i}
7   devolva A
```

Invariante: todo intervalo de R é compatível com os de A.

Claro que A é uma coleção de intervalos compatíveis.

Qual critério usamos para escolher um intervalo de R?

Algoritmo guloso

```
GULOSO (s, f, n)
1   ORDENE(s, f, n)                          ▷ f[1] ≤ f[2] ≤ ··· ≤ f[n]
2   A ← ∅
3   f_A ← 0                                  ▷ fim do último intervalo em A
4   para i ← 1 até n faça
5         se s[i] ≥ f_A                      ▷ intervalo i é compatível com A?
6               então A ← A ∪ {i}
7                     f_A ← f[i]
8   devolva A
```

Algoritmo guloso

```
GULOSO (s, f, n)
1   ORDENE(s, f, n)                    ▷ f[1] ≤ f[2] ≤ ··· ≤ f[n]
2   A ← ∅
3   f_A ← 0                            ▷ fim do último intervalo em A
4   para i ← 1 até n faça
5         se s[i] ≥ f_A                ▷ intervalo i é compatível com A?
6               então A ← A ∪ {i}
7                     f_A ← f[i]
8   devolva A
```

Consumo de tempo: $\mathrm{O}(n \lg n)$

Algoritmo guloso

```
GULOSO (s, f, n)
1   ORDENE(s, f, n)                    ▷ f[1] ≤ f[2] ≤ ··· ≤ f[n]
2   A ← ∅
3   f_A ← 0                            ▷ fim do último intervalo em A
4   para i ← 1 até n faça
5         se s[i] ≥ f_A                ▷ intervalo i é compatível com A?
6               então A ← A ∪ {i}
7                     f_A ← f[i]
8   devolva A
```

Consumo de tempo: $\mathrm{O}(n \lg n)$

Formalizando...

Por que A é coleção máxima de intervalos compatíveis?

Subestrutura ótima

Intervalos $S := \{1, \ldots, n\}$

Suponha que A é coleção máxima de intervalos disjunto de S.

Subestrutura ótima

Intervalos $S := \{1, \ldots, n\}$

Suponha que A é coleção máxima de intervalos disjunto de S.

Demonstre a seguinte propriedade:

Se $i \in A$ é tal que $f[i]$ é mínimo ▷ intervalo i é o mais à esquerda de A

Subestrutura ótima

Intervalos $S := \{1, \ldots, n\}$

Suponha que A é coleção máxima de intervalos disjunto de S.

Demonstre a seguinte propriedade:

Se $i \in A$ é tal que $f[i]$ é mínimo ▷ intervalo i é o mais à esquerda de A

então $A - \{i\}$ é coleção máxima de intervalos disjuntos

de $\{k : s[k] \geq f[i]\}$. ▷ intervalos compatíveis com i

$\{k : s[k] \geq f[i]\}$ = todos intervalos "à direita" de i.

No lugar de i, podemos colocar o intervalo j de S com $f[j]$ mínimo.

Algoritmo guloso

```
GULOSO (s, f, n)
1   ORDENE(s, f, n)                          ▷ f[1] ≤ f[2] ≤ ··· ≤ f[n]
2   A ← ∅
3   f_A ← 0                                  ▷ fim do último intervalo em A
4   para i ← 1 até n faça
5       se s[i] ≥ f_A                        ▷ intervalo i é compatível com A?
6           então A ← A ∪ {i}
7                 f_A ← f[i]
8   devolva A
```

Consumo de tempo: $\mathrm{O}(n \lg n)$

Do que argumentamos, vale a propriedade da escolha gulosa, logo o algoritmo devolve uma coleção máxima de intervalos disjuntos.

Coloração de intervalos

Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$ e um inteiro positivo k, uma k-coloração dos intervalos é uma partição deles em k coleções de intervalos disjuntos.

Coloração de intervalos

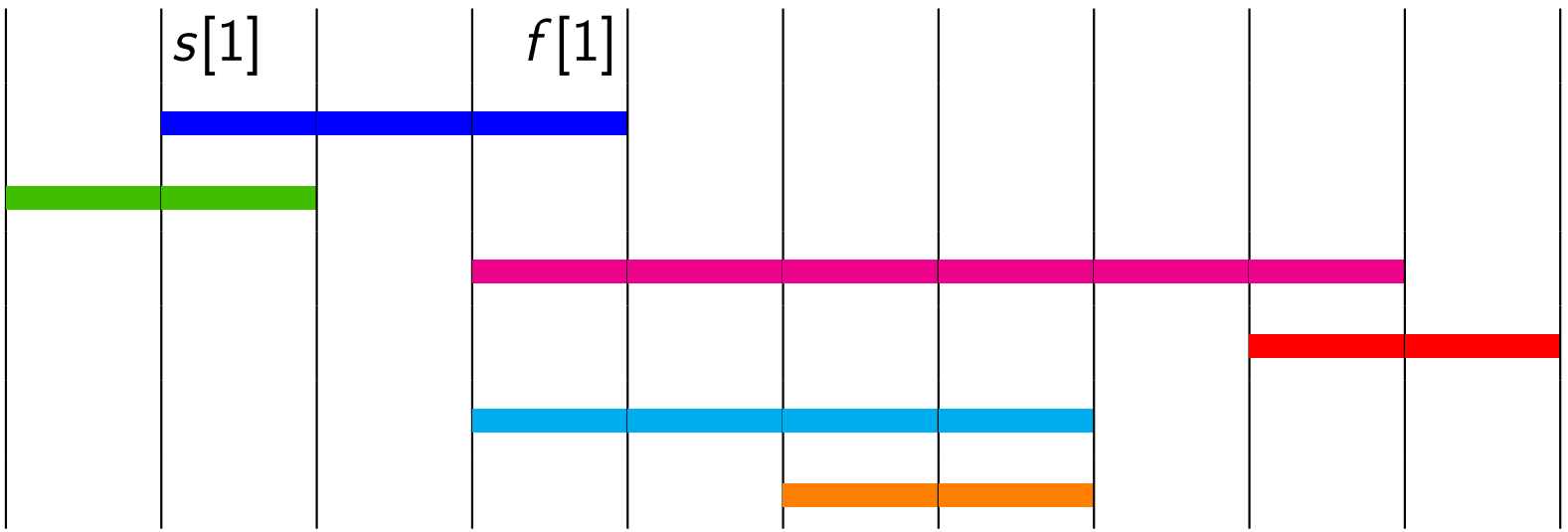
Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$ e um inteiro positivo k,
uma k-coloração dos intervalos é uma partição deles
em k coleções de intervalos disjuntos.

Problema: Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$,
encontrar uma k-coloração dos intervalos com k o menor possível.

Coloração de intervalos

Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$ e um inteiro positivo k, uma k-coloração dos intervalos é uma partição deles em k coleções de intervalos disjuntos.

Problema: Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$, encontrar uma k-coloração dos intervalos com k o menor possível.

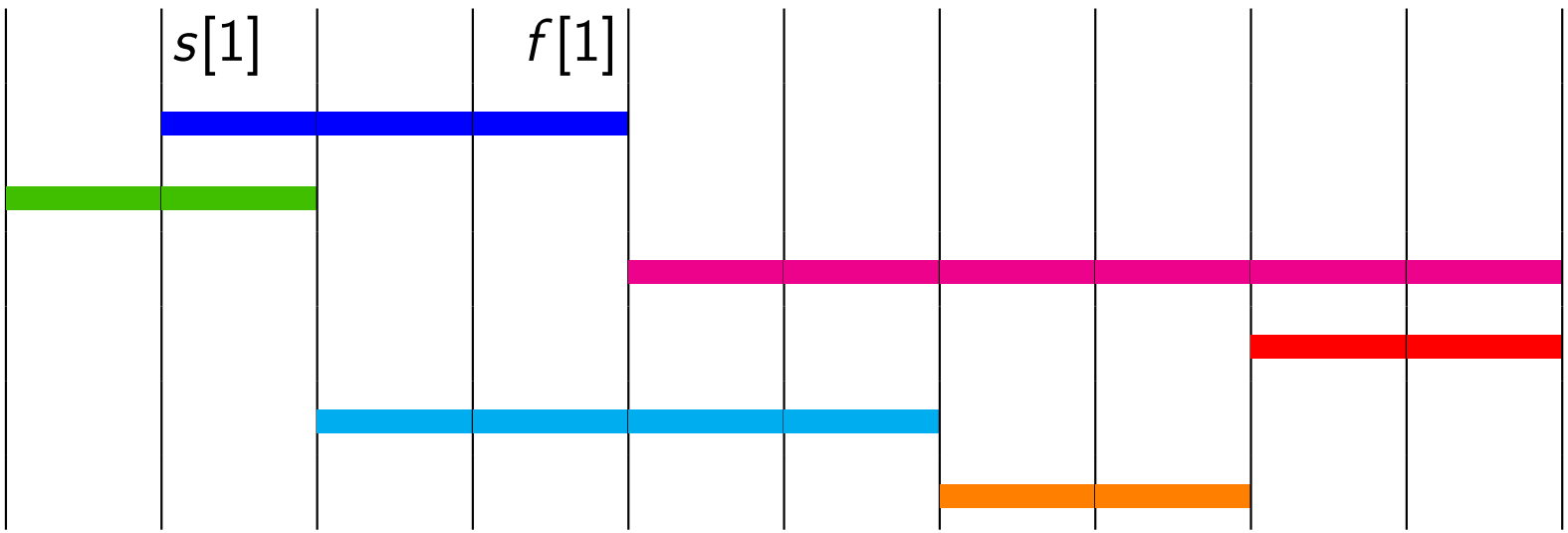


Quantas cores precisa?

Coloração de intervalos

Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$ e um inteiro positivo k, uma k-coloração dos intervalos é uma partição deles em k coleções de intervalos disjuntos.

Problema: Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$, encontrar uma k-coloração dos intervalos com k o menor possível.

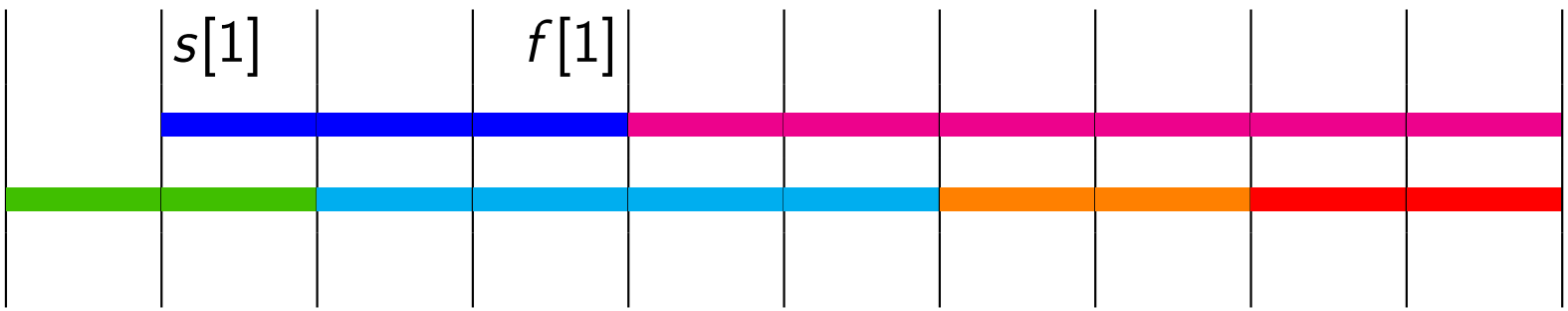


Quantas cores precisa?

Coloração de intervalos

Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$ e um inteiro positivo k, uma k-coloração dos intervalos é uma partição deles em k coleções de intervalos disjuntos.

Problema: Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$, encontrar uma k-coloração dos intervalos com k o menor possível.

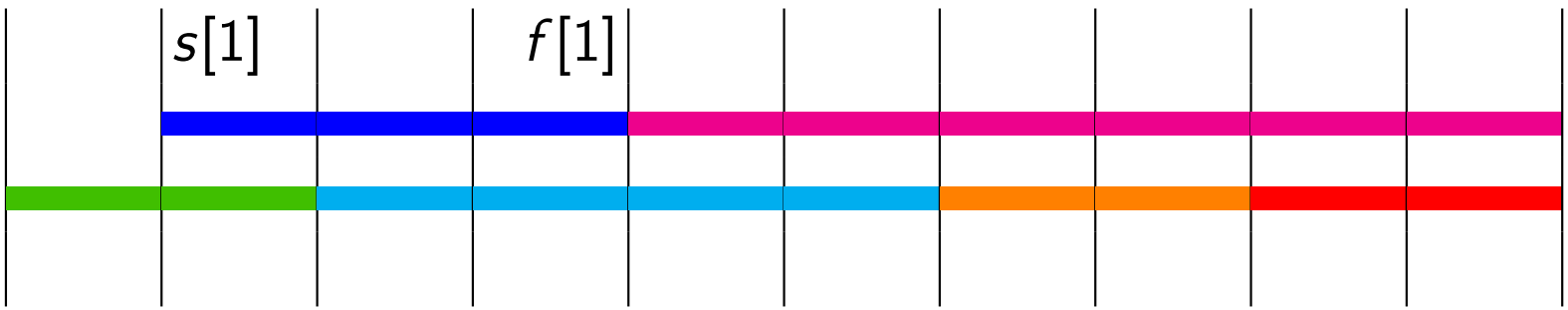


Solução: 2-coloração.

Coloração de intervalos

Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$ e um inteiro positivo k, uma k-coloração dos intervalos é uma partição deles em k coleções de intervalos disjuntos.

Problema: Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$, encontrar uma k-coloração dos intervalos com k o menor possível.



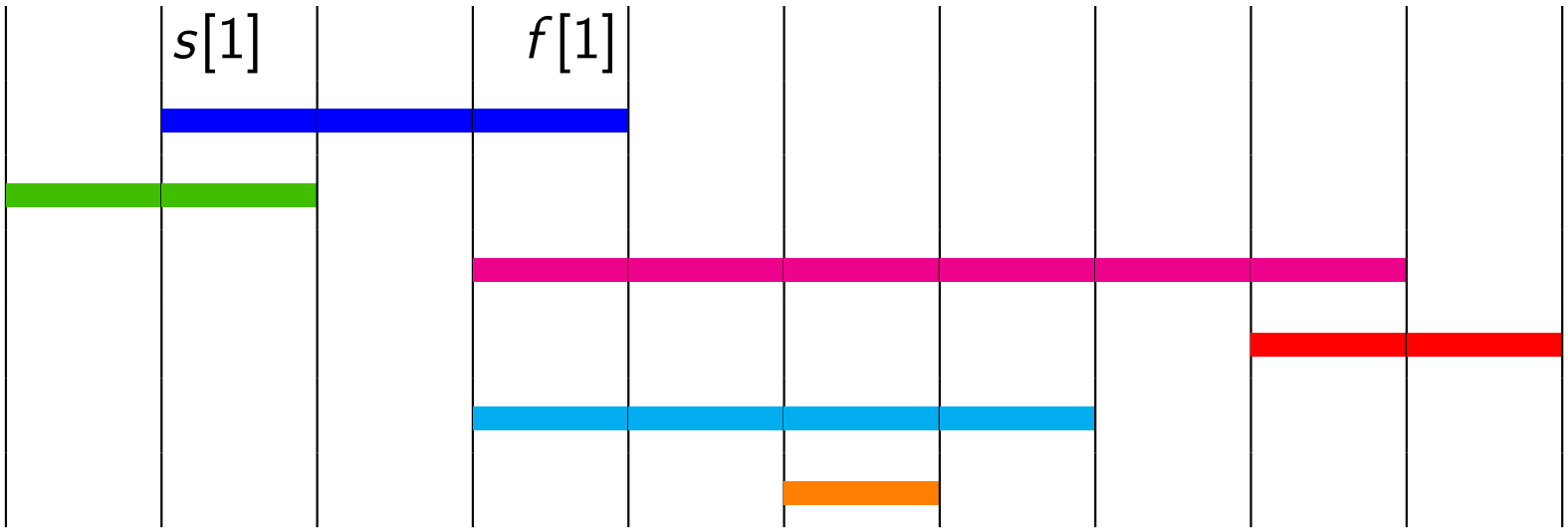
Solução: 2-coloração.

Como resolver esse problema?

Ideias?

Ordenar pelo s e pintar com a menor cor possível?

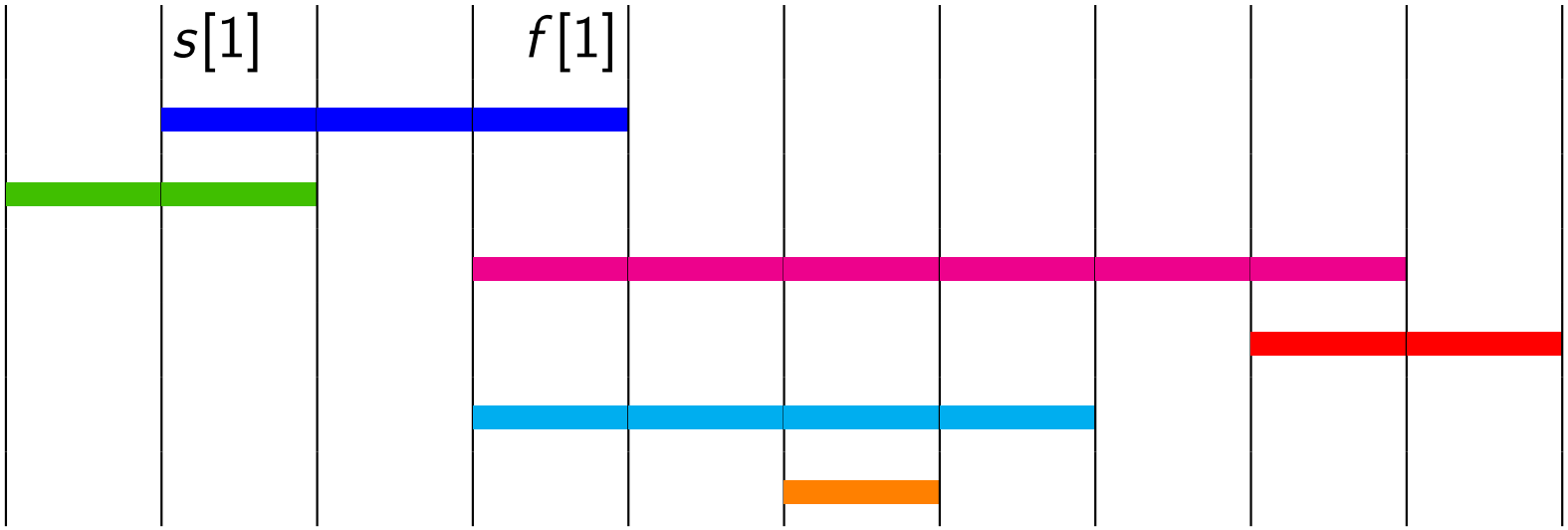
Funciona?



Ideias?

Ordenar pelo s e pintar com a menor cor possível?

Funciona?



Funciona! Produz um certificado!

Algoritmo guloso

```
GULOSO (s, f, n)
 1   ORDENE(s, f, n)                          ▷ s[1] ≤ s[2] ≤ ··· ≤ s[n]
 2   m ← 0                                    ▷ número de cores usadas
 3   ℓ[1] ← 0                                 ▷ término corrente daquela cor
 4   para k ← 1 até n faça
 5         j ← 1                              ▷ procura menor cor compatível para k
 6         enquanto ℓ[j] > s[k] faça
 7               j ← j + 1
 8         se j > m                           ▷ precisa de mais uma cor?
 9               então m ← j
10                     ℓ[m + 1] ← 0
11         ℓ[j] ← f[k]
12         c[k] ← j
13   devolva c
```

Um problema de escalonamento

d_i: deadline da tarefa i

t_i: tempo de processamento da tarefa i

Escalonamento: permutação π, onde
$\pi(i)$ denota a posição da tarefa i na ordem de execução.

Um problema de escalonamento

d_i: deadline da tarefa i

t_i: tempo de processamento da tarefa i

Escalonamento: permutação π, onde
$\pi(i)$ denota a posição da tarefa i na ordem de execução.

Dado π, o instante de início da tarefa i é

$$s_i = \sum_{j=1}^{\pi(i)-1} t_{\pi^{-1}(j)},$$

Um problema de escalonamento

d_i: deadline da tarefa i

t_i: tempo de processamento da tarefa i

Escalonamento: permutação π, onde
$\pi(i)$ denota a posição da tarefa i na ordem de execução.

Dado π, o instante de início da tarefa i é

$$s_i = \sum_{j=1}^{\pi(i)-1} t_{\pi^{-1}(j)},$$

ou seja, é a soma das durações das tarefas anteriores a i.

Um problema de escalonamento

d_i: deadline da tarefa i

t_i: tempo de processamento da tarefa i

Escalonamento: permutação π, onde
$\pi(i)$ denota a posição da tarefa i na ordem de execução.

Dado π, o instante de início da tarefa i é

$$s_i = \sum_{j=1}^{\pi(i)-1} t_{\pi^{-1}(j)},$$

ou seja, é a soma das durações das tarefas anteriores a i.

Instante de término: $f_i = s_i + t_i$.

Um problema de escalonamento

d_i: deadline da tarefa i

t_i: tempo de processamento da tarefa i

Escalonamento: permutação π, onde
$\pi(i)$ denota a posição da tarefa i na ordem de execução.

Dado π, o instante de início da tarefa i é

$$s_i = \sum_{j=1}^{\pi(i)-1} t_{\pi^{-1}(j)},$$

ou seja, é a soma das durações das tarefas anteriores a i.

Instante de término: $f_i = s_i + t_i$.

Atraso ℓ_i: 0 se $f_i \leq d_i$ e $f_i - d_i$ se $f_i > d_i$.

Mais um problema de escalonamento

d_i: deadline da tarefa i

t_i: tempo de processamento da tarefa i

Dado escalonamento π,

s_i: início do processamento da tarefa i

f_i: fim do processamento da tarefa i

ℓ_i: atraso da tarefa i

Mais um problema de escalonamento

d_i: deadline da tarefa i
t_i: tempo de processamento da tarefa i

Dado escalonamento π,
s_i: início do processamento da tarefa i
f_i: fim do processamento da tarefa i
ℓ_i: atraso da tarefa i

Problema: Dados d e t, encontrar π cujo atraso máximo é mínimo.

Mais um problema de escalonamento

d_i: deadline da tarefa i
t_i: tempo de processamento da tarefa i

Dado escalonamento π,
s_i: início do processamento da tarefa i
f_i: fim do processamento da tarefa i
ℓ_i: atraso da tarefa i

Problema: Dados d e t, encontrar π cujo atraso máximo é mínimo.

Exemplo 1: $t_1 = 1$ e $d_1 = 2$, $t_2 = 2$ e $d_2 = 4$, $t_3 = 3$ e $d_3 = 6$.

Mais um problema de escalonamento

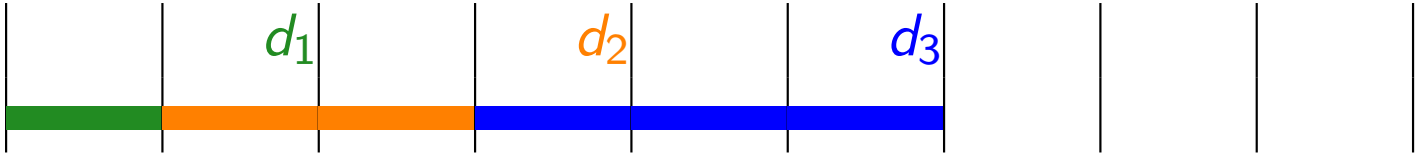
d_i: deadline da tarefa i
t_i: tempo de processamento da tarefa i

Dado escalonamento π,
s_i: início do processamento da tarefa i
f_i: fim do processamento da tarefa i
ℓ_i: atraso da tarefa i

Problema: Dados d e t, encontrar π cujo atraso máximo é mínimo.

Exemplo 1: $t_1 = 1$ e $d_1 = 2$, $t_2 = 2$ e $d_2 = 4$, $t_3 = 3$ e $d_3 = 6$.

Escalonamento com atraso mínimo: $(1, 2, 3)$



Atrasos: $\ell_1 = \ell_2 = \ell_3 = 0$.

Mais um problema de escalonamento

d_i: deadline da tarefa i
t_i: tempo de processamento da tarefa i

Dado escalonamento π,
s_i: início do processamento da tarefa i
f_i: fim do processamento da tarefa i
ℓ_i: atraso da tarefa i

Problema: Dados d e t, encontrar π cujo atraso máximo é mínimo.

Exemplo 2: $t_1 = 1$ e $d_1 = 4$, $t_2 = 2$ e $d_2 = 5$, $t_3 = 3$ e $d_3 = 3$.

Mais um problema de escalonamento

d_i: deadline da tarefa i
t_i: tempo de processamento da tarefa i

Dado escalonamento π,
s_i: início do processamento da tarefa i
f_i: fim do processamento da tarefa i
ℓ_i: atraso da tarefa i

Problema: Dados d e t, encontrar π cujo atraso máximo é mínimo.

Exemplo 2: $t_1 = 1$ e $d_1 = 4$, $t_2 = 2$ e $d_2 = 5$, $t_3 = 3$ e $d_3 = 3$.

Escalonamento com atraso mínimo: $(3, 1, 2)$

Atrasos: $\ell_1 = 0$, $\ell_2 = 1$, e $\ell_3 = 0$.

Possíveis critérios gulosos

- SPT - shortest processing time (menor t_i primeiro)

  Funciona?

Possíveis critérios gulosos

- SPT - shortest processing time (menor t_i primeiro)

  Funciona? Não...

  Exemplo: $d_1 = 9$ e $t_1 = 1$, $d_2 = 8$ e $t_2 = 8$.

Possíveis critérios gulosos

- SPT - shortest processing time (menor t_i primeiro)

  Funciona? Não...

  Exemplo: $d_1 = 9$ e $t_1 = 1$, $d_2 = 8$ e $t_2 = 8$.

Possíveis critérios gulosos

- SPT - shortest processing time (menor t_i primeiro)

  Funciona? Não...

  Exemplo: $d_1 = 9$ e $t_1 = 1$, $d_2 = 8$ e $t_2 = 8$.

- LST - least slack time (menor $d_i - t_i$ primeiro)

  Funciona?

Possíveis critérios gulosos

- SPT - shortest processing time (menor t_i primeiro)

  Funciona? Não...

  Exemplo: $d_1 = 9$ e $t_1 = 1$, $d_2 = 8$ e $t_2 = 8$.

- LST - least slack time (menor $d_i - t_i$ primeiro)

  Funciona? Também não...

  Exemplo: $d_1 = 2$ e $t_1 = 1$, $d_2 = 8$ e $t_2 = 8$.

Possíveis critérios gulosos

- SPT - shortest processing time (menor t_i primeiro)

  Funciona? Não...

  Exemplo: $d_1 = 9$ e $t_1 = 1$, $d_2 = 8$ e $t_2 = 8$.

- LST - least slack time (menor $d_i - t_i$ primeiro)

  Funciona? Também não...

  Exemplo: $d_1 = 2$ e $t_1 = 1$, $d_2 = 8$ e $t_2 = 8$.

Possíveis critérios gulosos

- LST - least slack time (menor $d_i - t_i$ primeiro)

  Funciona? Também não...

  Exemplo: $d_1 = 2$ e $t_1 = 1$, $d_2 = 8$ e $t_2 = 8$.

- EDD - earliest due date (menor d_i primeiro)

  Funciona?

Algoritmo resultante

GULOSO (d, t, n)

1 seja π permutação tq $d[\pi^{-1}(1)] \leq \cdots \leq d[\pi^{-1}(n)]$

2 **devolva** π

Nem olhamos o t... Será que isso funciona?

Algoritmo resultante

GULOSO (d, t, n)

1 seja π permutação tq $d[\pi^{-1}(1)] \leq \cdots \leq d[\pi^{-1}(n)]$

2 **devolva** π

Note que não há tempo ocioso em uma solução.

Algoritmo resultante

GULOSO (d, t, n)
1 seja π permutação tq $d[\pi^{-1}(1)] \leq \cdots \leq d[\pi^{-1}(n)]$
2 **devolva** π

Note que não há tempo ocioso em uma solução.

Dado um escalonamento,
uma inversão é um par de índices i e j tais que $\pi(i) < \pi(j)$ e $d_i > d_j$.

Algoritmo resultante

GULOSO (d, t, n)
1 seja π permutação tq $d[\pi^{-1}(1)] \leq \cdots \leq d[\pi^{-1}(n)]$
2 **devolva** π

Note que não há tempo ocioso em uma solução.

Dado um escalonamento,
uma inversão é um par de índices i e j tais que $\pi(i) < \pi(j)$ e $d_i > d_j$.

Afirmação 1:
Dois escalonamentos sem inversão têm o mesmo atraso máximo.

Algoritmo resultante

GULOSO (d, t, n)
1 seja π permutação tq $d[\pi^{-1}(1)] \leq \cdots \leq d[\pi^{-1}(n)]$
2 **devolva** π

Note que não há tempo ocioso em uma solução.

Dado um escalonamento,
uma inversão é um par de índices i e j tais que $\pi(i) < \pi(j)$ e $d_i > d_j$.

Afirmação 1:
Dois escalonamentos sem inversão têm o mesmo atraso máximo.

Prova: Qualquer que seja a ordem das tarefas com deadline d,
elas, juntas, consomem o mesmo tempo para serem executadas.
Em cada bloco de tarefas com um mesmo deadline d,
a mais atrasada é a que foi escalonada por último.
O atraso máximo nesse bloco é sempre o atraso da última.

Algoritmo resultante

GULOSO (d, t, n)
1 seja π permutação tq $d[\pi^{-1}(1)] \leq \cdots \leq d[\pi^{-1}(n)]$
2 **devolva** π

Note que não há tempo ocioso em uma solução.

Dado um escalonamento,
uma inversão é um par de índices i e j tais que $\pi(i) < \pi(j)$ e $d_i > d_j$.

Afirmação 1:
Dois escalonamentos sem inversão têm o mesmo atraso máximo.

Afirmação 2:
Existe uma solução que não tem nenhuma inversão.

Algoritmo resultante

GULOSO (d, t, n)
1 seja π permutação tq $d[\pi^{-1}(1)] \leq \cdots \leq d[\pi^{-1}(n)]$
2 **devolva** π

Note que não há tempo ocioso em uma solução.

Dado um escalonamento,
uma inversão é um par de índices i e j tais que $\pi(i) < \pi(j)$ e $d_i > d_j$.

Afirmação 1:
Dois escalonamentos sem inversão têm o mesmo atraso máximo.

Afirmação 2:
Existe uma solução que não tem nenhuma inversão.

Prova: Se houver uma inversão, há uma entre tarefas consecutivas.
Inverta duas tarefas consecutivas que formam uma inversão.
Isso não aumenta o atraso máximo. Repita até não haver inversão.

Algoritmo resultante

GULOSO (d, t, n)
1 seja π permutação tq $d[\pi^{-1}(1)] \leq \cdots \leq d[\pi^{-1}(n)]$
2 **devolva** π

Note que não há tempo ocioso em uma solução.

Dado um escalonamento,
uma inversão é um par de índices i e j tais que $\pi(i) < \pi(j)$ e $d_i > d_j$.

Afirmação 1:
Dois escalonamentos sem inversão têm o mesmo atraso máximo.

Afirmação 2:
Existe uma solução que não tem nenhuma inversão.

Disso segue que o algoritmo funciona.