

Algoritmos gulosos (*greedy*)

CLRS Cap 16 e KT Cap 4

Problema dos intervalos disjuntos

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

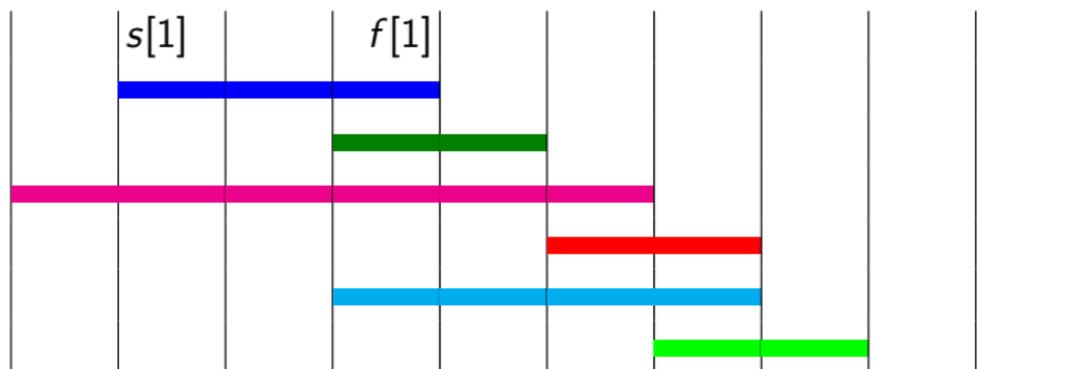
Solução é um subconjunto A de $\{1, \dots, n\}$.

Problema dos intervalos disjuntos

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

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

Exemplo:

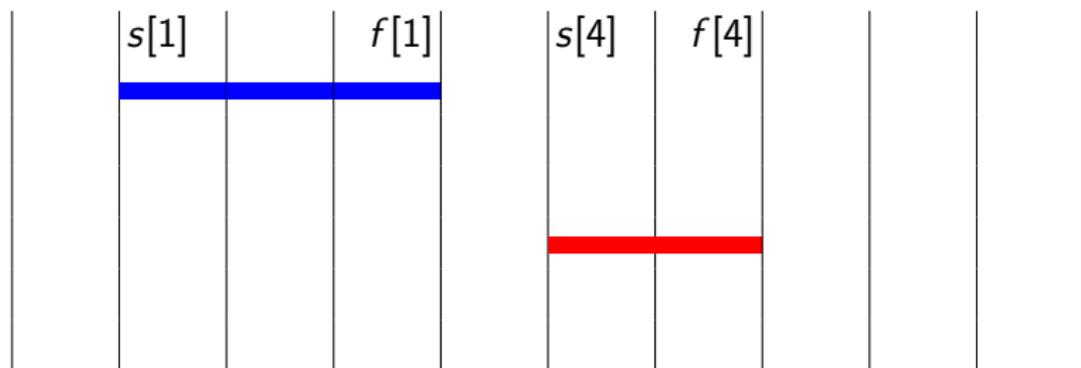


Problema dos intervalos disjuntos

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

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

Solução:

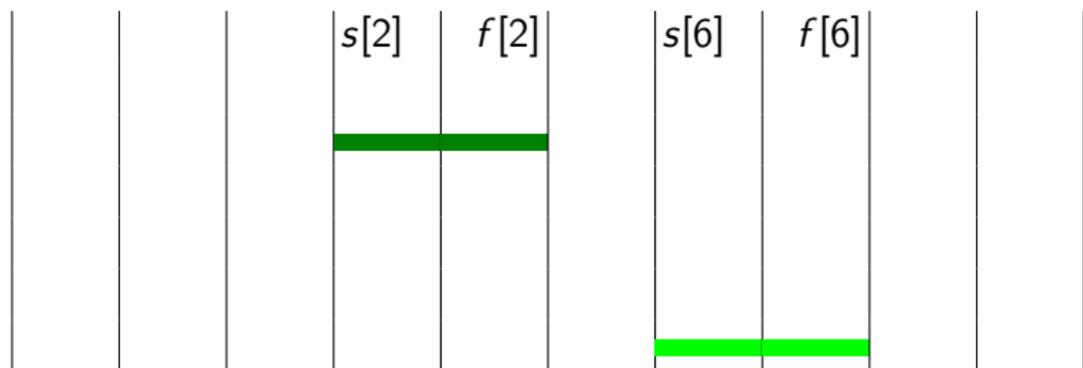


Problema dos intervalos disjuntos

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

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

Solução:



Motivação



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

Nome no CLRS: **Activity Selection Problem**

Subestrutura ótima

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

Suponha que

A é **coleção máxima** de intervalos disjuntos de S .

Se $i \notin A$

então

Subestrutura ótima

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

Suponha que

A é **coleção máxima** de intervalos disjuntos de S .

Se $i \notin A$

então A é **coleção máxima** de intervalos disjuntos de $S \setminus \{i\}$.

senão

Subestrutura ótima

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

Suponha que

A é **coleção máxima** de intervalos disjuntos de S .

Se $i \notin A$

então A é **coleção máxima** de intervalos disjuntos de $S \setminus \{i\}$.

senão $A \setminus \{i\}$ é **coleção máxima** de intervalos disjuntos de $S \setminus \{k : [s[k], f[k]) \cap [s[i], f[i]) \neq \emptyset\}$.

Demonstre a propriedade.

Subestrutura ótima II

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

Suponha que

A é **coleção máxima** de intervalos disjuntos de S .

Se $i \notin A$

então A é **coleção máxima** de intervalos disjuntos de $S \setminus \{i\}$.

senão suponha que i é o primeiro intervalo de A

Subestrutura ótima II

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

Suponha que

A é **coleção máxima** de intervalos disjuntos de S .

Se $i \notin A$

então A é **coleção máxima** de intervalos disjuntos de $S \setminus \{i\}$.

senão suponha que i é o primeiro intervalo de A

$A - \{i\}$ é **coleção máxima** de intervalos disjuntos de $\{k : s[k] \geq f[i]\}$.

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

Demonstre a propriedade.

Algoritmo de programação dinâmica

Suponha $s[1] \leq s[2] \leq \dots \leq s[n]$

$t[i]$ = tamanho de uma subcoleção
disjunta máxima de $\{i, \dots, n\}$

Algoritmo de programação dinâmica

Suponha $s[1] \leq s[2] \leq \dots \leq s[n]$

$t[i]$ = tamanho de uma subcoleção
disjunta máxima de $\{i, \dots, n\}$

$$t[n] = 1$$

$$t[i] = \max \{t[i+1], 1 + t[k]\} \quad \text{para } i = 1, \dots, n-1,$$

onde k é o menor índice tal que $s[k] \geq f[i]$.

Algoritmo de programação dinâmica

DYNAMIC-ACTIVITY-SELECTOR (s, f, n)

0 ordene s e f de forma que $s[1] \leq s[2] \leq \dots \leq s[n]$

1 $A[n + 1] \leftarrow \emptyset$

2 **para** $i \leftarrow n$ decrescendo até 1 **faça**

3 $A[i] \leftarrow A[i + 1]$

4 $k \leftarrow i + 1$

5 **enquanto** $k \leq n$ e $s[k] < f[i]$ **faça**

6 $k \leftarrow k + 1$

7 **se** $|A[i]| < 1 + |A[k]|$

8 **então** $A[i] \leftarrow \{i\} \cup A[k]$

9 **devolva** $A[1]$

Consumo de tempo é $\Theta(n^2)$.

Conclusão

Invariante: na linha 2 vale que

(i0) $A[k]$ é *coleção disjunta máxima* de $\{k, \dots, n\}$
para $k = i + 1, \dots, n$.

O consumo de tempo do algoritmo
DYNAMIC-ACTIVITY-SELECTOR é $\Theta(n^2)$.

Algoritmos gulosos

“A *greedy algorithm* starts with a solution to a very small subproblem and augments it successively to a solution for the big problem. The augmentation is done in a “greedy” fashion, that is, paying attention to short-term or local gain, without regard to whether it will lead to a good long-term or global solution. As in real life, greedy algorithms sometimes lead to the best solution, sometimes lead to pretty good solutions, and sometimes lead to lousy solutions. The trick is to determine when to be greedy.”

“One thing you will notice about greedy algorithms is that they are usually easy to design, easy to implement, easy to analyse, and they are very fast, but they are *almost always difficult to prove correct.*”

I. Parberry, *Problems on Algorithms*, Prentice Hall, 1995.

Algoritmos gulosos

Algoritmo guloso

- ▶ procura ótimo local e acaba obtendo ótimo global

Costuma ser

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

Problema precisa ter

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

Algoritmo guloso

Considere o problema da coleção máxima de intervalos disjuntos.

GULOSO (s, f, n)

```
1   $A \leftarrow \emptyset$ 
2   $R \leftarrow \{1, \dots, n\}$ 
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  $A$ 
```

Claro que A é uma coleção de intervalos compatíveis.

Qual critério usamos para escolher um intervalo de R ?

Possíveis critérios gulosos

- ▶ escolher o intervalo i em R com menor $s[i]$;

Possíveis critérios gulosos

- ▶ escolher o intervalo i em R com menor $s[i]$;
- ▶ escolher o intervalo i em R com menor $f[i] - s[i]$;

Possíveis critérios gulosos

- ▶ escolher o intervalo i em R com menor $s[i]$;
- ▶ escolher o intervalo i em R com menor $f[i] - s[i]$;
- ▶ escolher o intervalo i tal que

$$|\{j \in R : j \text{ intersecta } i\}|$$

é o menor possível;

Possíveis critérios gulosos

- ▶ escolher o intervalo i em R com menor $s[i]$;
- ▶ escolher o intervalo i em R com menor $f[i] - s[i]$;
- ▶ escolher o intervalo i tal que

$$|\{j \in R : j \text{ intersecta } i\}|$$

é o menor possível;

- ▶ escolher o intervalo i em R com menor $f[i]$;

Escolha gulosa

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

Se $f[i]$ é mínimo em S ,

então **EXISTE** uma solução ótima A tal que $i \in A$.

Demonstre a propriedade.

Algoritmo guloso

Devolve uma coleção **máxima de intervalos** disjuntos dois a dois.

INTERVALOS-DISJUNTOS (s, f, n)

0 ordene s e f de forma que $f[1] \leq f[2] \leq \dots \leq f[n]$

1 $A \leftarrow \{1\}$

2 $i \leftarrow 1$

3 **para** $j \leftarrow 2$ **até** n **faça**

4 **se** $s[j] \geq f[i]$

5 **então** $A \leftarrow A \cup \{j\}$

6 $i \leftarrow j$

7 **devolva** A

Consumo de tempo da linha 0 é $\Theta(n \lg n)$.

Consumo de tempo das linhas 1–7 é $\Theta(n)$.

Conclusão

Na linha 3 vale que

(i0) A é uma coleção máxima de intervalos disjuntos de $(s, f, j-1)$

O consumo de tempo do algoritmo
INTERVALOS-DISJUNTOS é $\Theta(n \lg n)$.

Problema fracionário da mochila

Problema: Dados (w, v, n, W) , encontrar uma **mochila ótima**.

Problema fracionário da mochila

Problema: Dados (w, v, n, W) , encontrar uma mochila ótima.

Exemplo: $W = 50, n = 4$

	1	2	3	4
w	40	30	20	10
v	840	600	400	100
x	1	0	0	0
x	1	0	0	1
x	0	1	1	0
x	1	1/3	0	0

valor = 840

valor = 940

valor = 1000

valor = 1040

Subestrutura ótima

Suponha que $x[1..n]$ é **mochila ótima** para o problema (w, v, n, W) .

Subestrutura ótima

Suponha que $x[1..n]$ é **mochila ótima** para o problema (w, v, n, W) .

Se $x[n] = \delta$

então $x[1..n-1]$ é **mochila ótima** para
 $(w, v, n-1, W - \delta w[n])$

Subestrutura ótima

Suponha que $x[1..n]$ é **mochila ótima** para o problema (w, v, n, W) .

Se $x[n] = \delta$

então $x[1..n-1]$ é **mochila ótima** para
 $(w, v, n-1, W - \delta w[n])$

NOTA. Não há nada de especial acerca do índice n .
Uma afirmação semelhante vale para qualquer índice i .

Subestrutura ótima

Suponha que $x[1..n]$ é **mochila ótima** para o problema (w, v, n, W) .

Se $x[n] = \delta$

então $x[1..n-1]$ é **mochila ótima** para
 $(w, v, n-1, W - \delta w[n])$

NOTA. Não há nada de especial acerca do índice n .
Uma afirmação semelhante vale para qualquer índice i .

Então podemos fazer um algoritmo de PD como no caso anterior.
Mas não dá para fazer algo mais simples, melhor?

Escolha gulosa

Suponha $w[i] \neq 0$ para todo i .

Escolha gulosa

Suponha $w[i] \neq 0$ para todo i .

Se $v[n]/w[n] \geq v[i]/w[i]$ para todo i

então **EXISTE** uma mochila ótima $x[1..n]$ tal que

$$x[n] = \min \left\{ 1, \frac{W}{w[n]} \right\}$$

Algoritmo guloso

Esta **propriedade da escolha gulosa** sugere um algoritmo que atribui os valores de $x[1..n]$ supondo que os dados estejam em ordem decrescente de “valor específico”:

$$\frac{v[1]}{w[1]} \leq \frac{v[2]}{w[2]} \leq \dots \leq \frac{v[n]}{w[n]}$$

Algoritmo guloso

Esta **propriedade da escolha gulosa** sugere um algoritmo que atribui os valores de $x[1..n]$ supondo que os dados estejam em ordem decrescente de “valor específico”:

$$\frac{v[1]}{w[1]} \leq \frac{v[2]}{w[2]} \leq \dots \leq \frac{v[n]}{w[n]}$$

É nessa ordem “**mágica**” que está o **segredo do funcionamento** do algoritmo.

Algoritmo guloso

Devolve uma **mochila ótima** para (w, v, n, W) .

MOCHILA-FRACIONÁRIA (w, v, n, W)

- 0 ordene w e v de tal forma que
$$v[1]/w[1] \leq v[2]/w[2] \leq \dots \leq v[n]/w[n]$$
- 1 **para** $i \leftarrow n$ decrescendo até 1 **faça**
- 2 **se** $w[i] \leq W$
- 3 **então** $x[i] \leftarrow 1$
- 4 $W \leftarrow W - w[i]$
- 5 **senão** $x[i] \leftarrow W/w[i]$
- 6 $W \leftarrow 0$
- 7 **devolva** x

Algoritmo guloso

Devolve uma **mochila ótima** para (w, v, n, W) .

MOCHILA-FRACIONÁRIA (w, v, n, W)

- 0 ordene w e v de tal forma que
$$v[1]/w[1] \leq v[2]/w[2] \leq \dots \leq v[n]/w[n]$$
- 1 **para** $i \leftarrow n$ decrescendo até 1 **faça**
- 2 **se** $w[i] \leq W$
- 3 **então** $x[i] \leftarrow 1$
- 4 $W \leftarrow W - w[i]$
- 5 **senão** $x[i] \leftarrow W/w[i]$
- 6 $W \leftarrow 0$
- 7 **devolva** x

Consumo de tempo da linha 0 é $\Theta(n \lg n)$.

Consumo de tempo das linhas 1–7 é $\Theta(n)$.

Invariante

Seja W_0 o valor original de W .

No início de cada execução da linha 1 vale que

Invariante

Seja W_0 o valor original de W .

No início de cada execução da linha 1 vale que

(i0) $x' = x[i+1..n]$ é **mochila ótima** para

$$(w', v', n', W_0)$$

onde

$$w' = w[i+1..n]$$

$$v' = v[i+1..n]$$

$$n' = n - i$$

Invariante

Seja W_0 o valor original de W .

No início de cada execução da linha 1 vale que

(i0) $x' = x[i+1..n]$ é **mochila ótima** para

$$(w', v', n', W_0)$$

onde

$$w' = w[i+1..n]$$

$$v' = v[i+1..n]$$

$$n' = n - i$$

Na última iteração $i = 0$ e

portanto $x[1..n]$ é **mochila ótima** para (w, v, n, W_0) .

Conclusão

O consumo de tempo do algoritmo
MOCHILA-FRACIONÁRIA é $\Theta(n \lg n)$.

Escolha gulosa

Precisamos mostrar que
se $x[1..n]$ é uma **mochila ótima**,
então podemos supor que

$$x[n] = \alpha := \min \left\{ 1, \frac{W}{w[n]} \right\}.$$

Escolha gulosa

Precisamos mostrar que
se $x[1..n]$ é uma **mochila ótima**,
então podemos supor que

$$x[n] = \alpha := \min \left\{ 1, \frac{W}{w[n]} \right\}.$$

Depois de mostrar isto, indução faz o resto do serviço.

Escolha gulosa

Precisamos mostrar que
se $x[1..n]$ é uma **mochila ótima**,
então podemos supor que

$$x[n] = \alpha := \min \left\{ 1, \frac{W}{w[n]} \right\}.$$

Depois de mostrar isto, indução faz o resto do serviço.

Técnica:

transformar uma **solução ótima** em uma **solução ótima 'gulosa'**.

Escolha gulosa

Precisamos mostrar que
se $x[1..n]$ é uma **mochila ótima**,
então podemos supor que

$$x[n] = \alpha := \min \left\{ 1, \frac{W}{w[n]} \right\}.$$

Depois de mostrar isto, indução faz o resto do serviço.

Técnica:

transformar uma **solução ótima** em uma **solução ótima 'gulosa'**.

Esta transformação é semelhante ao processo de pivotação do algoritmo **SIMPLEX** para programação linear.

Escolha gulosa

Seja $x[1..n]$ uma **mochila ótima** para (w, v, n, W)
tal que $x[n]$ é **máximo**. Se $x[n] = \alpha$, não há o que mostrar.

Escolha gulosa

Seja $x[1..n]$ uma **mochila ótima** para (w, v, n, W)
tal que $x[n]$ é **máximo**. Se $x[n] = \alpha$, não há o que mostrar.

Suponha $x[n] < \alpha$.

Escolha gulosa

Seja $x[1..n]$ uma **mochila ótima** para (w, v, n, W)
tal que $x[n]$ é **máximo**. Se $x[n] = \alpha$, não há o que mostrar.

Suponha $x[n] < \alpha$.

Podemos supor que $x \cdot w = W$. (Podemos mesmo?)

Escolha gulosa

Seja $x[1..n]$ uma **mochila ótima** para (w, v, n, W)
tal que $x[n]$ é **máximo**. Se $x[n] = \alpha$, não há o que mostrar.

Suponha $x[n] < \alpha$.

Podemos supor que $x \cdot w = W$. (Podemos mesmo?)

Seja i em $[1..n-1]$ tal que $x[i] > 0$.
(Quem garante que existe um tal i ?).

Escolha gulosa

Seja $x[1..n]$ uma mochila ótima para (w, v, n, W)
tal que $x[n]$ é máximo. Se $x[n] = \alpha$, não há o que mostrar.

Suponha $x[n] < \alpha$.

Podemos supor que $x \cdot w = W$. (Podemos mesmo?)

Seja i em $[1..n-1]$ tal que $x[i] > 0$.
(Quem garante que existe um tal i ?).

Seja

$$\delta := \min \left\{ x[i], (\alpha - x[n]) \frac{w[n]}{w[i]} \right\}$$

e

$$\beta := \delta \frac{w[i]}{w[n]}.$$

Escolha gulosa

Seja $x[1..n]$ uma mochila ótima para (w, v, n, W)
tal que $x[n]$ é máximo. Se $x[n] = \alpha$, não há o que mostrar.

Suponha $x[n] < \alpha$.

Podemos supor que $x \cdot w = W$. (Podemos mesmo?)

Seja i em $[1..n-1]$ tal que $x[i] > 0$.
(Quem garante que existe um tal i ?).

Seja

$$\delta := \min \left\{ x[i], (\alpha - x[n]) \frac{w[n]}{w[i]} \right\}$$

e

$$\beta := \delta \frac{w[i]}{w[n]}.$$

Note que $\delta > 0$ e $\beta > 0$.

Mais escolha gulosa

Seja $x'[1..n]$ tal que

$$x'[j] := \begin{cases} x[j] & \text{se } j \neq i \text{ e } j \neq n, \\ x[i] - \delta & \text{se } j = i, \\ x[n] + \beta & \text{se } j = n. \end{cases}$$

Mais escolha gulosa

Seja $x'[1..n]$ tal que

$$x'[j] := \begin{cases} x[j] & \text{se } j \neq i \text{ e } j \neq n, \\ x[i] - \delta & \text{se } j = i, \\ x[n] + \beta & \text{se } j = n. \end{cases}$$

Verifique que $0 \leq x'[j] \leq 1$ para todo j .

Mais escolha gulosa

Seja $x'[1..n]$ tal que

$$x'[j] := \begin{cases} x[j] & \text{se } j \neq i \text{ e } j \neq n, \\ x[i] - \delta & \text{se } j = i, \\ x[n] + \beta & \text{se } j = n. \end{cases}$$

Verifique que $0 \leq x[j] \leq 1$ para todo j .

Além disso, temos que

$$\begin{aligned} x' \cdot w &= x'[1]w[1] + \cdots + x'[i]w[i] + \cdots + x'[n]w[n] \\ &= x[1]w[1] + \cdots + (x[i] - \delta)w[i] + \cdots + (x[n] + \beta)w[n] \\ &= x[1]w[1] + \cdots + (x[i] - \delta)w[i] + \cdots + \left(x[n] + \delta \frac{w[i]}{w[n]}\right) w[n] \\ &= x[1]w[1] + \cdots + x[i]w[i] - \delta w[i] + \cdots + x[n]w[n] + \delta w[i] \\ &= W. \end{aligned}$$

Mais escolha gulosa ainda

Temos ainda que

$$\begin{aligned}x' \cdot v &= x'[1]v[1] + \cdots + x'[i]v[i] + \cdots + x'[n]v[n] \\&= x[1]v[1] + \cdots + (x[i] - \delta)v[i] + \cdots + (x[n] + \beta)v[n] \\&= x[1]v[1] + \cdots + (x[i] - \delta)v[i] + \cdots + \left(x[n] + \delta \frac{w[i]}{w[n]}\right)v[n] \\&= x[1]v[1] + \cdots + x[i]v[i] - \delta v[i] + \cdots + x[n]v[n] + \delta w[i] \frac{v[n]}{w[n]} \\&= x \cdot v + \delta \left(w[i] \frac{v[n]}{w[n]} - v[i]\right) \\&\geq x \cdot v + \delta \left(w[i] \frac{v[i]}{w[i]} - v[i]\right) \quad (\text{devido à escolha gulosa!}) \\&= x \cdot v.\end{aligned}$$

Escolha gulosa: epílogo

Assim, x' é uma mochila tal que $x' \cdot v \geq x \cdot v$.

Escolha gulosa: epílogo

Assim, x' é uma mochila tal que $x' \cdot v \geq x \cdot v$.

Como x é **mochila ótima**, concluímos que $x' \cdot v = x \cdot v$ e que x' é uma mochila ótima que contradiz a nossa escolha de x , já que

$$x'[n] = x[n] + \beta > x[n].$$

Escolha gulosa: epílogo

Assim, x' é uma mochila tal que $x' \cdot v \geq x \cdot v$.

Como x é **mochila ótima**, concluímos que $x' \cdot v = x \cdot v$ e que x' é uma mochila ótima que contradiz a nossa escolha de x , já que

$$x'[n] = x[n] + \beta > x[n].$$

Conclusão

Se $v[n]/w[n] \geq v[i]/w[i]$ para todo i
e x é uma **mochila ótima** para (w, v, n, W)
com $x[n]$ **máximo**, então

$$x[n] = \min \left\{ 1, \frac{W}{w[n]} \right\}.$$

A propósito ...

O **problema fracionário da mochila** é um problema de **programação linear (PL)**: encontrar um vetor x que

$$\begin{array}{ll} \text{maximize} & x \cdot v \\ \text{sob as restrições} & x \cdot w \leq W \\ & x[i] \geq 0 \quad \text{para } i = 1, \dots, n \\ & x[i] \leq 1 \quad \text{para } i = 1, \dots, n \end{array}$$

A propósito ...

O problema fracionário da mochila é um problema de programação linear (PL): encontrar um vetor x que

$$\begin{aligned} & \text{maximize} && x \cdot v \\ & \text{sob as restrições} && x \cdot w \leq W \\ & && x[i] \geq 0 \quad \text{para } i = 1, \dots, n \\ & && x[i] \leq 1 \quad \text{para } i = 1, \dots, n \end{aligned}$$

PL's podem ser resolvidos por

SIMPLEX: no pior caso consome tempo exponencial
na prática é muito rápido

ELIPSÓIDES: consome tempo polinomial na prática é lento

PONTOS-INTERIORES: consome tempo polinomial
na prática é rápido

Coloração de intervalos

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

Coloração de intervalos

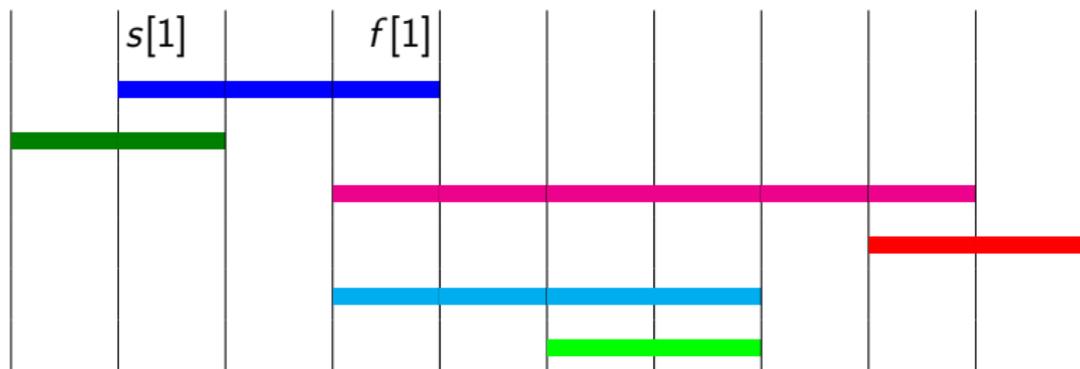
Dados intervalos $[s[1], f[1]), \dots, [s[n], f[n])$ e um inteiro positivo k , uma **k -coloração** dos intervalos é uma partição deles em k **coleções de intervalos disjuntos**.

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

Coloração de intervalos

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

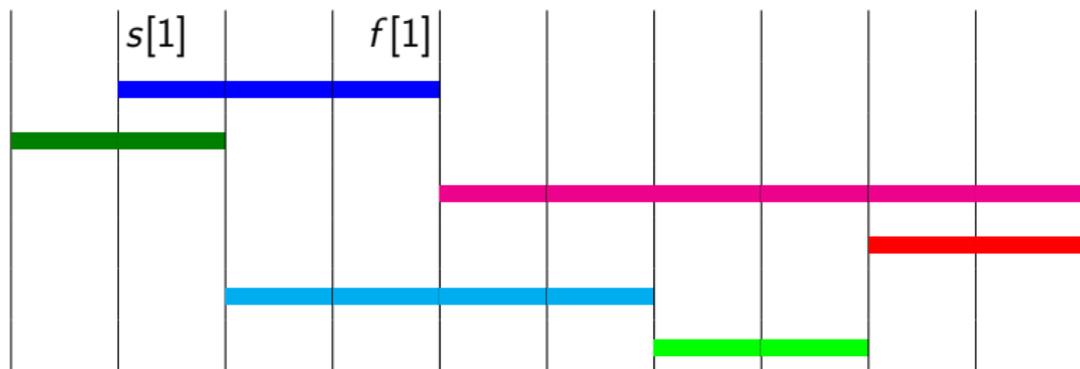
Problema: Dados intervalos $[s[1], f[1]), \dots, [s[n], f[n])$, encontrar uma k -coloração dos intervalos com k o menor possível.



Coloração de intervalos

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

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



Coloração de intervalos

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

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



Solução: 2-coloração.

Coloração de intervalos

Estratégias gulosas:

- ▶ Encontre uma coleção disjunta máxima de intervalos, pinte com a próxima cor disponível e repita a idéia para os intervalos restantes.
- ▶ Ordene as atividades de maneira que $f[1] \leq f[2] \leq \dots \leq f[n]$ e pinte uma a uma nesta ordem sempre usando a menor cor possível para aquela atividade.
- ▶ Ordene as atividades de maneira que $s[1] \leq s[2] \leq \dots \leq s[n]$ e pinte uma a uma nesta ordem sempre usando a menor cor possível para aquela atividade.

Quais destas estratégias funcionam?

Quais não funcionam?

Algoritmo guloso

GULOSO (s, f, n)

```
1  ORDENE( $s, f, n$ )  ▷  $s[1] \leq s[2] \leq \dots \leq s[n]$ 
2   $m \leftarrow 0$ 
3   $\ell[1] \leftarrow 0$ 
4  para  $k \leftarrow 1$  até  $n$  faça
5       $j \leftarrow 1$ 
6      enquanto  $\ell[j] > s[k]$  faça
7           $j \leftarrow j + 1$ 
8      se  $j > m$ 
9          então  $m \leftarrow j$ 
10          $\ell[m + 1] \leftarrow 0$ 
11      $\ell[j] \leftarrow f[k]$ 
12      $c[k] \leftarrow j$ 
13  devolva  $c$ 
```

Exercícios

1. Mostre um exemplo para os três primeiros critérios gulosos apresentados para o primeiro problema que prove que o algoritmo obtido usando estes critérios pode produzir um conjunto A que não é máximo.
2. Considere o algoritmo do slide anterior para o segundo problema. Modifique-o para que, além de c , ele devolva um conjunto S de m intervalos e um instante t tal que $s[i] \leq t < f[i]$ para todo i em S .
3. Numa versão anterior dos slides desta aula, o algoritmo do segundo problema ordenava os intervalos pelo valor de f em vez de pelo valor de s . Isso faz diferença? Ou seja, a versão do algoritmo da página anterior que ordena os intervalos por f em vez de s também dá uma resposta correta? Prove ou dê um contra-exemplo.