Algoritmos gulosos (*greedy*)

CLRS 16.2

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 atention 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."

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 atention 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.

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	valor $= 840$
x	1	0	0	1	valor $= 940$
x	0	1	1	0	valor $= 1000$
x	1	$1/3$	0	0	valor $= 1040$

A propósito . . .

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

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

A propósito . . .

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

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

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

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.

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":

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

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":

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

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] ≤ v[2]/w[2] ≤ ··· ≤ v[n]/w[n]
1  para i ← n decrescendo até 1 faça
2     se w[i] ≤ W
3        então x[i] ← 1
4              W ← W − w[i]
5        senão x[i] ← W/w[i]
6              W ← 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] ≤ v[2]/w[2] ≤ ··· ≤ v[n]/w[n]
 1  para i ← n decrescendo até 1 faça
 2     se w[i] ≤ W
 3        então x[i] ← 1
 4              W ← W − w[i]
 5        senão x[i] ← W/w[i]
 6              W ← 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

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

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

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

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

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

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 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 provar que está correto

Problema precisa ter

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

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*)

Exercício: O problema da mochila booleana pode ser resolvido por um algoritmo guloso?

Problema dos intervalos disjuntos

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

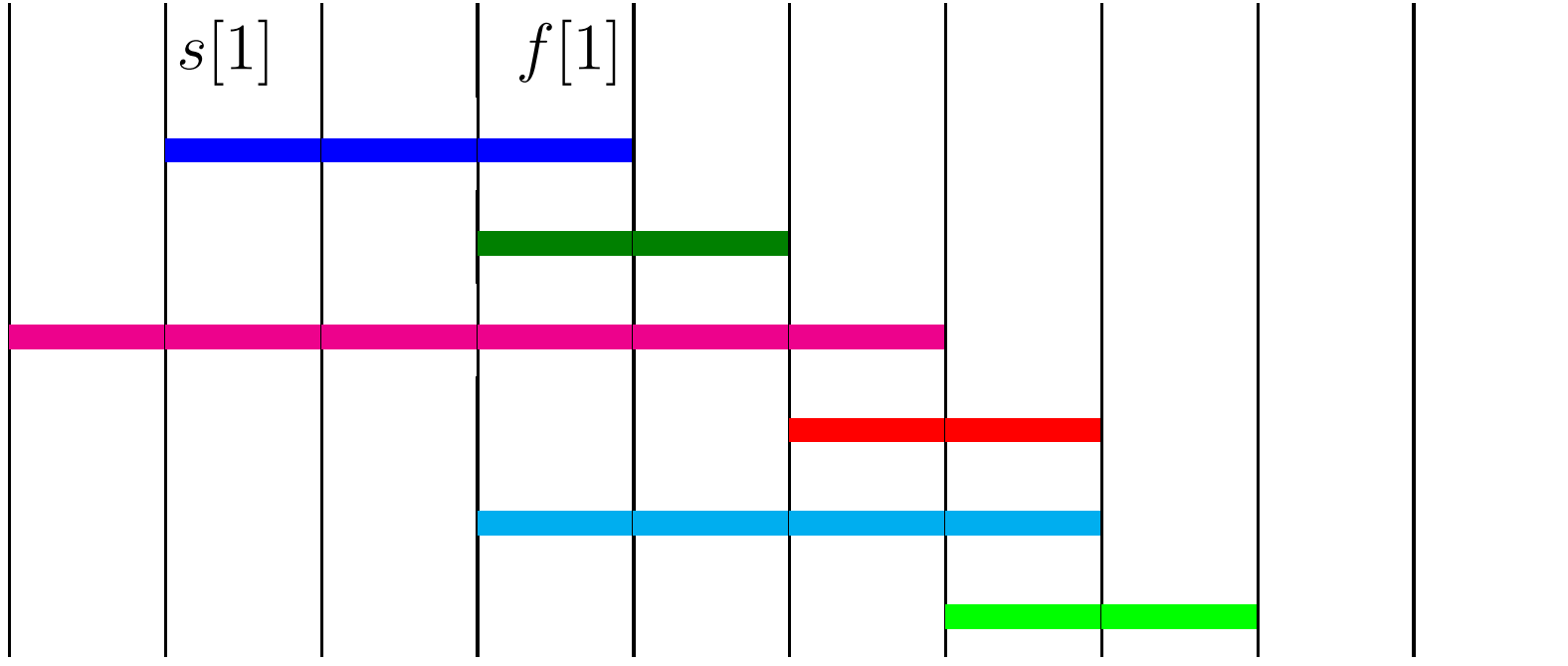
Solução é um subconjunto A de $\{1, \ldots, n\}$.

Problema dos intervalos disjuntos

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

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

Exemplo:

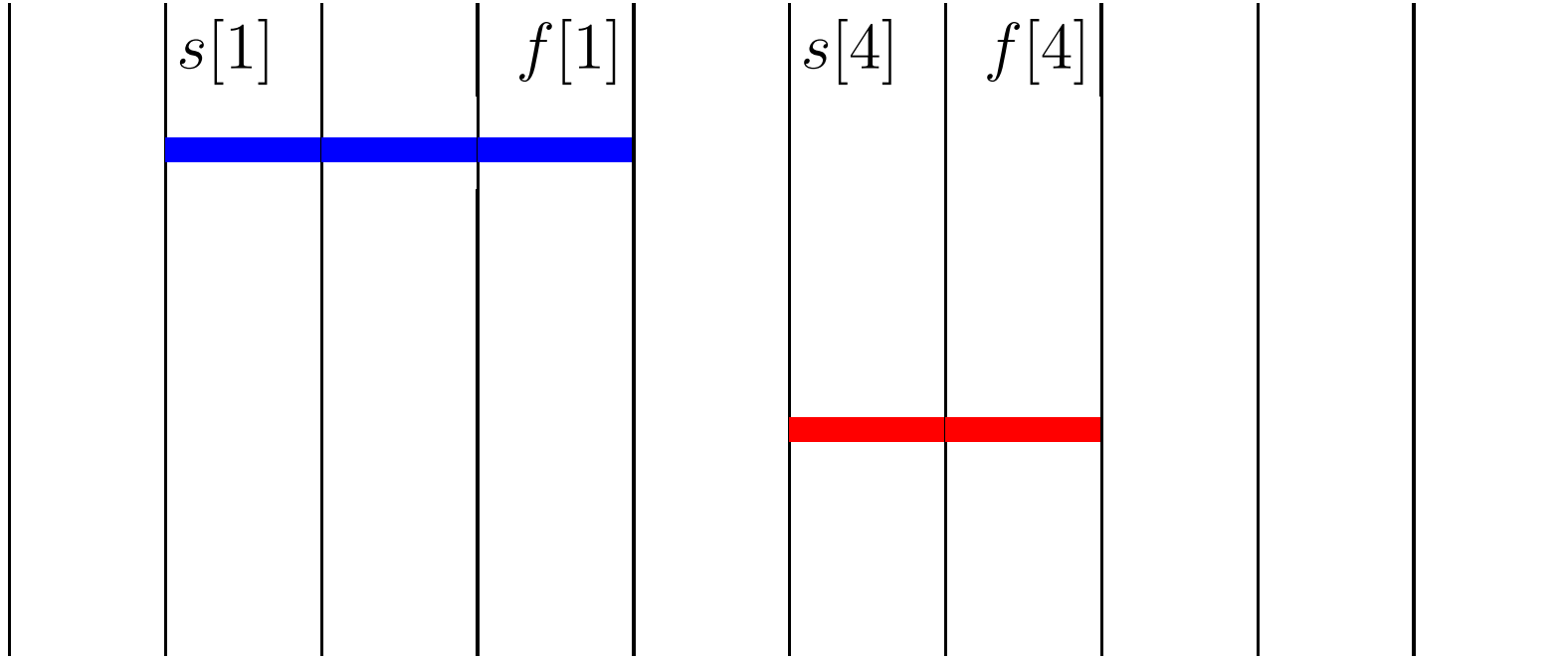


Problema dos intervalos disjuntos

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

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

Solução:

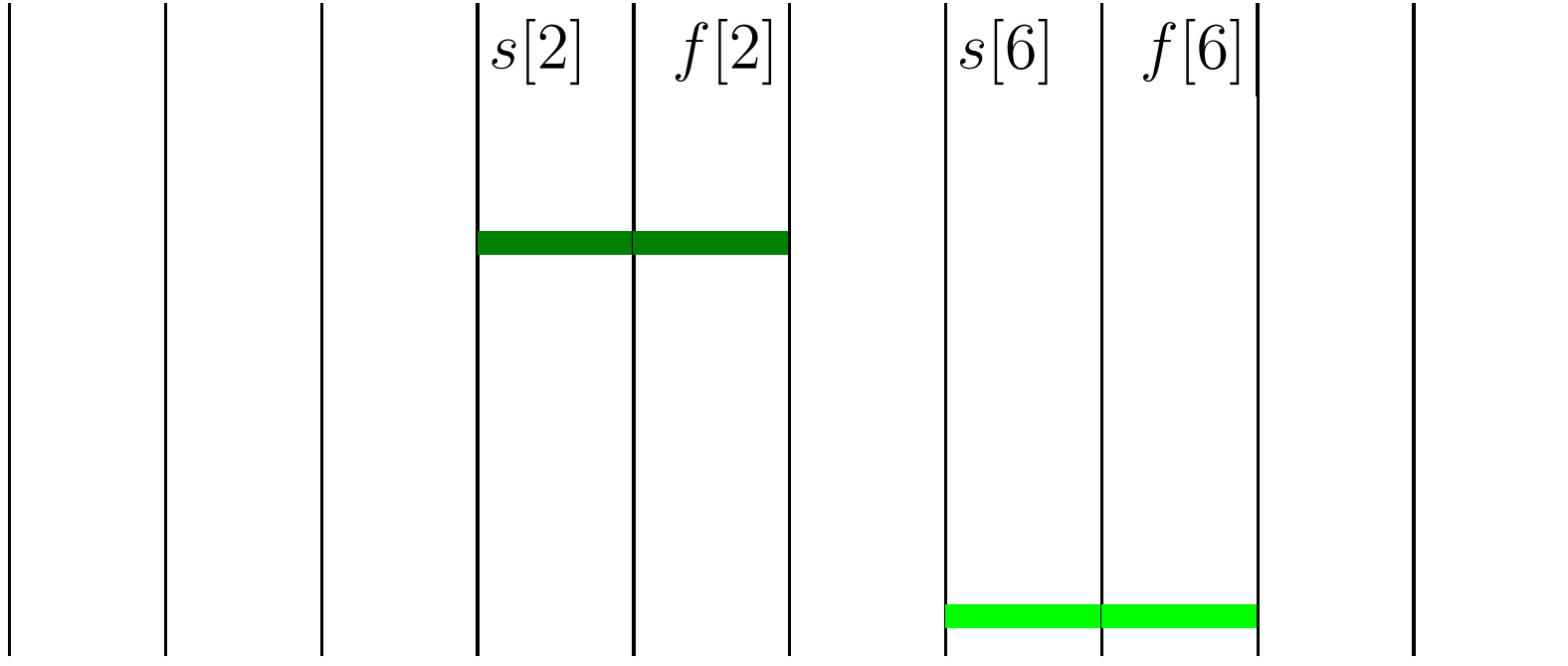


Problema dos intervalos disjuntos

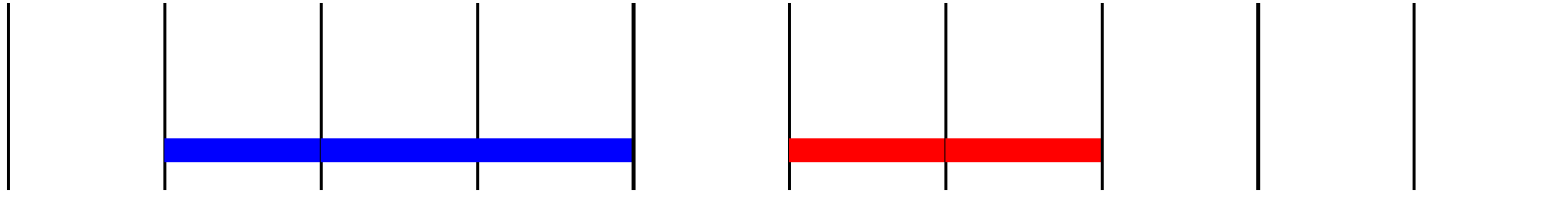
Problema: Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$, encontrar uma coleção máxima de intervalos disjuntos dois a dois.

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

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

Suponha que A é coleção máxima de intervalos de S disjuntos dois a dois.

Se $i \in A$

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

senão A é coleção máxima de intervalos disjuntos de $S - \{i\}$.

Demonstre a propriedade.

Subestrutura ótima II

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

Suponha que A é coleção máxima de intervalos de S disjuntos dois a dois.

Se $i \in A$ é tal que $f[i]$ é mínimo

então $A - \{i\}$ 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 $s[1] \leq s[2] \leq \cdots \leq s[n]$

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

$$t[n] = 1$$

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

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 tal forma que
$$s[1] \leq s[2] \leq \cdots \leq s[n]$$
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 $\{k, \ldots, n\}$ para $k = i + 1, \ldots, n$.

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

Escolha gulosa

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

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 tal forma que
$$f[1] \leq f[2] \leq \cdots \leq f[n]$$

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 $\Theta(n \lg n)$.

Coloração de intervalos

Problema: Dados intervalos de tempo $[s_1, f_2), \ldots, [s_n, f_n)$, encontrar uma coloração dos intervalos com o menor número possível de cores em que dois intervalos de mesma cor sempre sejam disjuntos.

Solução: uma partição de $\{1, \ldots, n\}$ em coleções de intervalos dois a dois disjuntos.

Motivação

Queremos distribuir um conjunto de atividades no menor número possivel de salas.

Cada atividade a_i ocupa um certo intervalo de tempo $[s_i, f_i)$ e duas atividades podem ser programadas para a mesma sala somente se os correspondentes intervalos são disjuntos.

Cada sala corresponde a uma cor. Queremos usar o menor número possível de cores para pintar todos os intervalos.

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 $f[1] \leq f[2] \leq \cdots \leq f[n]$ e pinte uma a uma nesta ordem sempre usando a menor cor possível para aquela atividade.
- Ordene as atividades de maneira que $s[1] \leq s[2] \leq \cdots \leq s[n]$ e pinte uma a uma nesta ordem sempre usando a menor cor possível para aquela atividade.

Quais destas estratégias funcionam?
Quais não funcionam?