# Problema da Mochila: Recorrência

Problema (Knapsack Problem):
Dados $(w, v, n, W)$, encontrar uma mochila boolena ótima.

Seja $t[i, Y]$ o valor da expressão $x \cdot v$ sujeito à restrição $x \cdot w \leq Y$.

$$
\begin{array}{lcll}
t[0, Y] & = & 0 & \text{para todo } Y \\
t[i, 0] & = & 0 & \text{para todo } i \\
t[i, Y] & = & t[i-1, Y] & \text{se } w[i] > Y \\
t[i, Y] & = & \max\{t[i-1, Y], t[i-1, Y-w[i]] + v[i]\} & \text{se } w[i] \leq Y
\end{array}
$$

# Algoritmo de programação dinâmica

Devolve o valor de uma mochila booleana ótima para $(w, v, n, W)$.

```
MOCHILA-BOOLEANA (w, v, n, W)
 1  para Y ← 0 até W faça
 2      t[0, Y] ← 0
 3      para i ← 1 até n faça
 4          a ← t[i−1, Y]
 5          se w[i] > Y
 6              então b ← 0
 7              senão b ← t[i−1, Y − w[i]] + v[i]
 8          t[i, Y] ← max{a, b}
 9  devolva t[n, W]
```

Consumo de tempo é $\Theta(nW)$.

# Algoritmo de programação dinâmica

Devolve o valor de uma mochila booleana ótima para $(w, v, n, W)$.

```
MOCHILA-BOOLEANA (w, v, n, W)
 1  para Y ← 0 até W faça
 2      t[0, Y] ← 0
 3      para i ← 1 até n faça
 4          a ← t[i−1, Y]
 5          se w[i] > Y
 6              então b ← 0
 7              senão b ← t[i−1, Y − w[i]] + v[i]
 8          t[i, Y] ← max{a, b}
 9  devolva t[n, W]
```

Consumo de tempo é $\Theta(nW)$.

Como encontrar uma mochila ótima?

# Obtenção da mochila

```
MOCHILA (w, n, W, t)
1  Y ← W
2  para i ← n decrescendo até 1 faça
3      se t[i, Y] = t[i−1, Y]
4          então x[i] ← 0
5          senão x[i] ← 1
6                Y ← Y − w[i]
7  devolva x
```

# Obtenção da mochila

```
MOCHILA (w, n, W, t)
1  Y ← W
2  para i ← n decrescendo até 1 faça
3      se t[i, Y] = t[i−1, Y]
4          então x[i] ← 0
5          senão x[i] ← 1
6                Y ← Y − w[i]
7  devolva x
```

Consumo de tempo é $\Theta(n)$.

# Versão recursiva

```
MEMOIZED-MOCHILA-BOOLEANA (w, v, n, W)
1  para i ← 0 até n faça
2      para Y ← 0 até W faça
3          t[i, Y] ← ∞
4  devolva LOOKUP-MOC (w, v, n, W)
```

# Versão recursiva

```
MEMOIZED-MOCHILA-BOOLEANA (w, v, n, W)
1  para i ← 0 até n faça
2      para Y ← 0 até W faça
3          t[i, Y] ← ∞
4  devolva LOOKUP-MOC (w, v, n, W)

LOOKUP-MOC (w, v, i, Y)
1  se t[i, Y] < ∞
2      então devolva t[i, Y]
3  se i = 0 ou Y = 0 então t[i, Y] ← 0
```

# Versão recursiva

```
MEMOIZED-MOCHILA-BOOLEANA (w, v, n, W)
1  para i ← 0 até n faça
2      para Y ← 0 até W faça
3          t[i, Y] ← ∞
4  devolva LOOKUP-MOC (w, v, n, W)

LOOKUP-MOC (w, v, i, Y)
1  se t[i, Y] < ∞
2      então devolva t[i, Y]
3  se i = 0 ou Y = 0 então t[i, Y] ← 0
4  senão se w[i] > Y
5              então t[i, Y] ← LOOKUP-MOC (w, v, i−1, Y)
6              senão a ← LOOKUP-MOC (w, v, i−1, Y)
7                    b ← LOOKUP-MOC (w, v, i−1, Y−w[i]) + v[i]
8                    t[i, Y] ← max {a, b}
9  devolva t[i, Y]
```

# Problema dos caramelos

Problema K em:

`https://maratona.sbc.org.br/primfase24/contest-pf2024/maratona.pdf`

Primeiramente temos que mastigar um pouco esse problema...

Suponha que temos uma partição das sacolas de caramelos que deixa exatamente metade dos caramelos para cada um deles.
(Se isso não existir, não há nada a fazer.)

Será que, neste caso, existe uma ordem que deixe cada um dos dois com exatamente um dos conjuntos dessa partição?

Agora... será que conseguimos decidir se existe uma tal partição?
Como podemos usar o problema da mochila para isso?

# Algoritmos gulosos (*greedy*)

CLRS Cap 16 Greedy Algorithms e KT Cap 4

# 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.

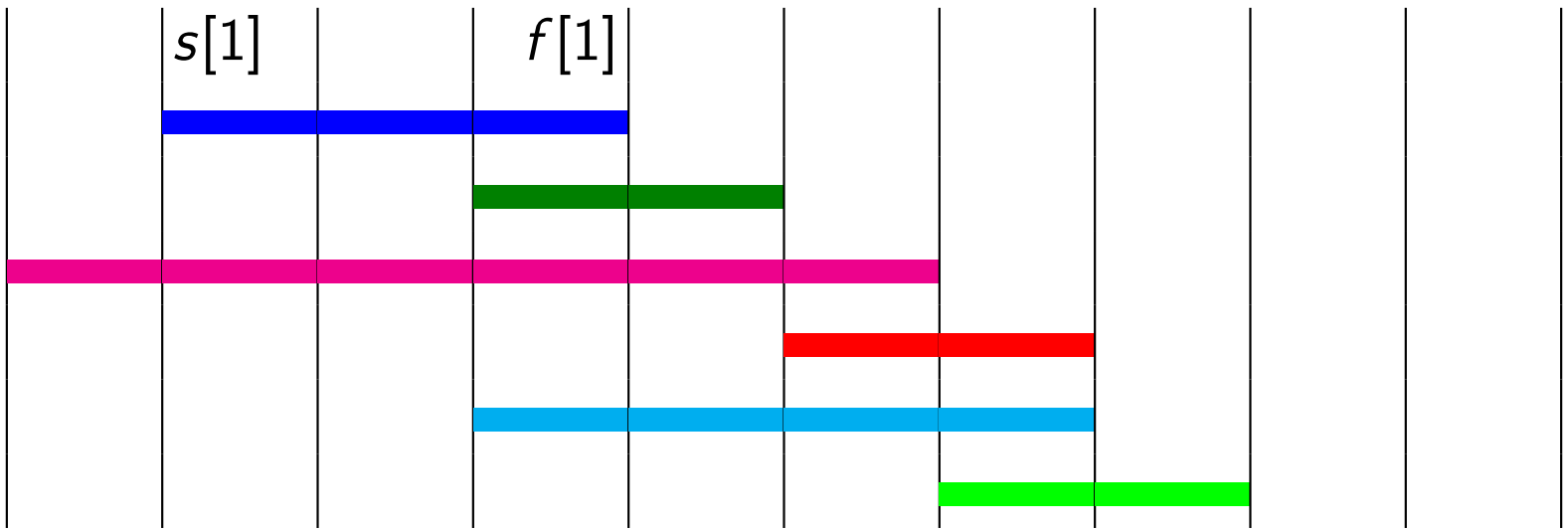
Solução é um subconjunto $A$ de $\{1, \ldots, n\}$.

# 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:

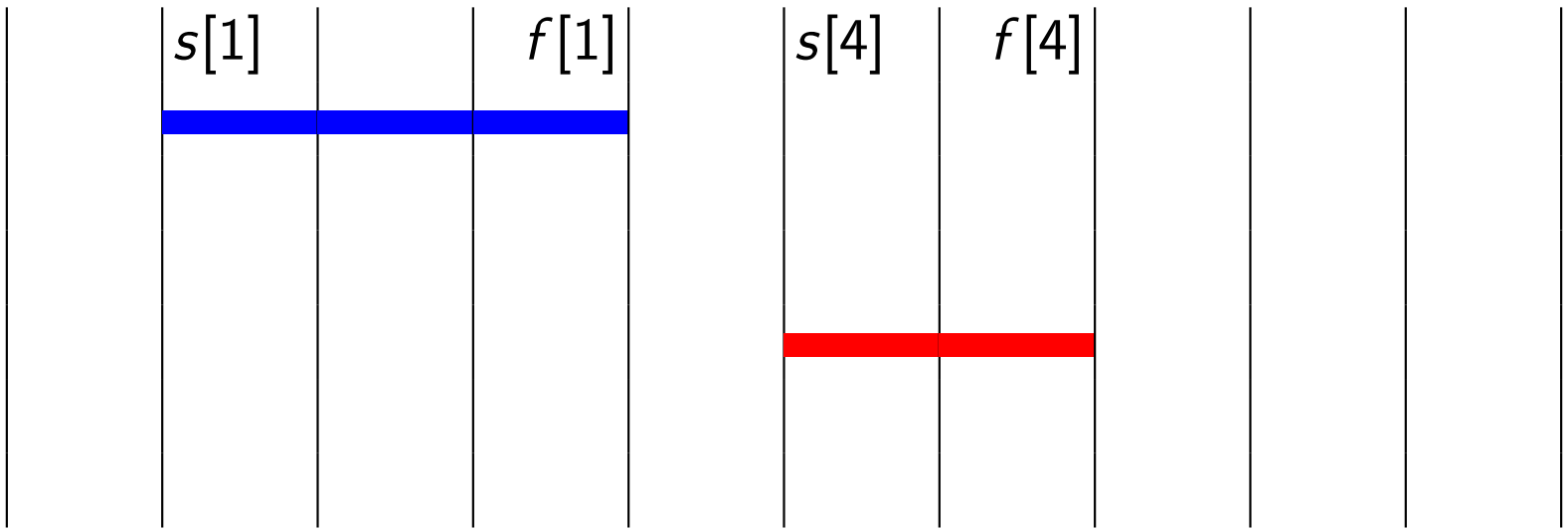


# Problema dos intervalos disjuntos

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:

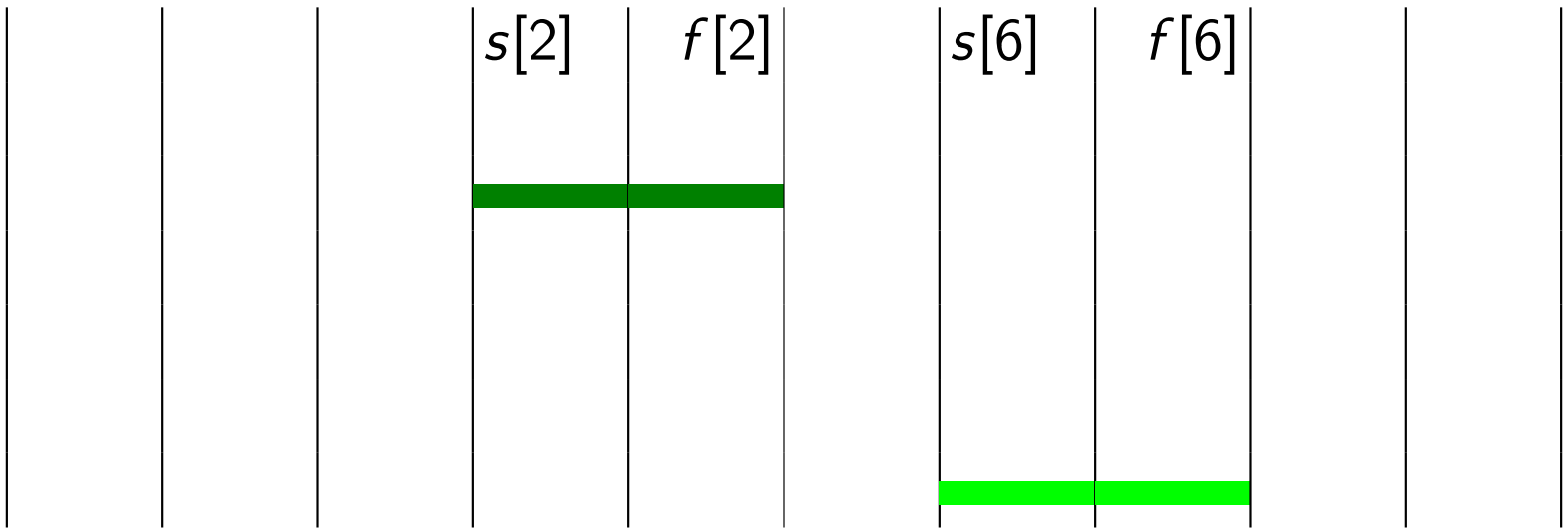


# Problema dos intervalos disjuntos

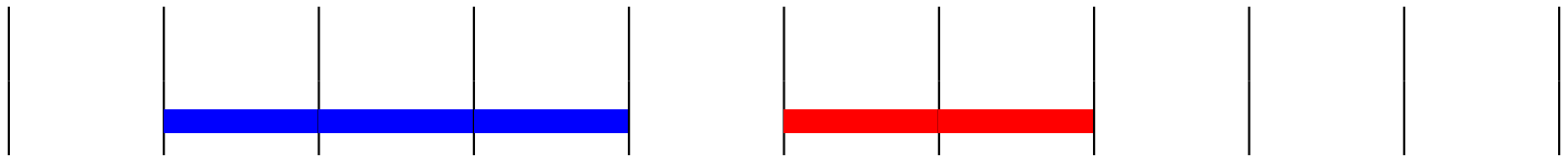
Problema: Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$,
encontrar uma coleção máxima de intervalos disjuntos dois a dois.

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 disjuntos de $S$.

Se $i \notin A$

então

# Subestrutura ótima

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

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

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

Suponha que

$A$ é coleção máxima de intervalos disjunto 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 $S := \{1, \ldots, n\}$

Suponha que

$A$ é coleção máxima de intervalos disjunto 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 $s[1] \leq s[2] \leq \cdots \leq s[n]$.

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

# 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 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 $\quad A[i] \leftarrow A[i+1]$
4 $\quad k \leftarrow i+1$
5 $\quad$ **enquanto** $k \leq n$ e $s[k] < f[i]$ **faça**
6 $\quad\quad k \leftarrow k+1$
7 $\quad$ **se** $|A[i]| < 1 + |A[k]|$
8 $\quad\quad$ **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 $\Theta(n^2)$.

# 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.

# 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 $R \leftarrow \{1, \ldots, n\}$
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** $A$

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

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

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

# 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 forma que f[1] ≤ f[2] ≤ ··· ≤ f[n]
1  A ← {1}
2  i ← 1
3  para j ← 2 até n faça
4      se s[j] ≥ f[i]
5          então A ← A ∪ {j}
6                i ← 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 $\Theta(n \lg n)$.