Fontes principais

1. Cormem T. H.; Leiserson C. E.; Rivest R.: Stein C.. *Introduction to Algorithms*, 3^a edição, MIT Press, 2009

2. Análise de algoritmo - IME/USP (prof. Paulo Feofiloff) `http://www.ime.usp.br/~pf/analise_de_algoritmos`

Algoritmos gulosos

Algoritmos gulosos

Normalmente aplicado a problemas de otimização, em que queremos computar a melhor solução.

Em cada passo, o algoritmo sempre escolhe a melhor opção local viável, sem se preocupar com as conseqüências futuras (dizemos que ele é "míope")

Algoritmos gulosos

▷ Nem sempre produz a solução ótima

▷ Não existe backtracking envolvido

▷ Na maioria das vezes, projetar ou descrever um algoritmo guloso é "fácil", mas provar sua corretude é difícil.

Estratégia gulosa vs Programação dinâmica

Algoritmo guloso ("ganancioso"):

- ▷ "abocanha"a alternativa mais promissora, sem explorar outras
- ▷ a execução costuma a ser muito rápida
- ▷ nunca se arrepende da decisão tomada
- ▷ prova de corretude difícil

Estratégia gulosa vs Programação dinâmica

Algoritmo de programação dinâmica:

▷ explora todas as alternativas, e faz isso de maneira eficiente
▷ a execução é um tanto "lenta"
▷ a cada iteração pode se arrepender de decisões tomadas anteriormente (pode rever o "ótimo corrente")
▷ prova de corretude fácil (explora todas as possibilidades)

Exemplo: Máximo e Maximal

$\triangleright$ S é uma coleção de subconjuntos de $\{1 \cdots n\}$

$\triangleright$ $X \subset S$ é **máximo** se não existe $Y \subset S$ tal que $|Y| > |X|$

$\triangleright$ $X \subset S$ é **maximal** se não existe $Y \subset S$ tal que $X \subset Y$, ou seja, se nenhum elemento de S é superconjunto próprio de X.

Exemplo: Máximo e Maximal

$S = \{\{1,2\},\{2,3\},\{4,5\},\{1,2,3\},\{1,2,4\},\{2,3,4,5\},\{1,3,4,5\}\}$

$\triangleright$ Elementos máximos: $\{2,3,4,5\}$ e $\{1,3,4,5\}$
$\triangleright$ Elementos maximais: $\{1,2,3\}$, $\{1,2,4\}$, $\{2,3,4,5\}$ e $\{1,3,4,5\}$

Note que todo máximo é maximal, mas a recíproca não é verdade.

Máximo e Maximal

▷ Procurar um elemento máximo em S é computacionalmente pesado (examinar todos os elementos)

▷ Mas encontrar um elemento maximal de S é muito fácil, aplicando a estratégia gulosa.

Máximo e Maximal

Algoritmo guloso para o maximal:

escolha algum X em S
enquanto $X \subset Y$ para algum Y em S **faça**
$X \leftarrow Y$
devolva X

Verifique que o algoritmo funciona para o exemplo descrito anteriormente

Máximo e Maximal

É ainda mais fácil encontrar um elemento maximal se a coleção S tiver caráter hereditário, ou seja, se tiver a seguinte propriedade: para cada X em S, todos os subconjuntos de X também estão em S. Nesse caso, basta executar o algoritmo:

$X \leftarrow \{\}$
para cada k em $\{1 \cdots n\}$ faça
    $Y \leftarrow X \cup \{k\}$
    se Y está em S
        então $X \leftarrow Y$
devolva X

Problema da seleção de atividades

Problema da seleção de atividades

Considere um conjunto $\{1, 2, \cdots, n\}$ de n atividades que competem por um recurso, por exemplo uma sala de aula.

Cada atividade tem um início s_i e um término t_i, com $s_i \leq t_i$.

O intervalo requerido pela atividade é $[s_i, t_i)$.

Problema da seleção de atividades

Duas atividades i e j são compatíveis se os intervalos $[s_i, t_i)$ $[s_j, t_j)$ não se interceptam ($s_i \geq t_j$ ou $s_j \geq t_i$).

Problema: encontrar o conjunto de atividades mutuamente compatíveis de tamanho máximo.

Problema da seleção de atividades

Exemplo: Considerando 11 atividades em 14 unidades de tempo

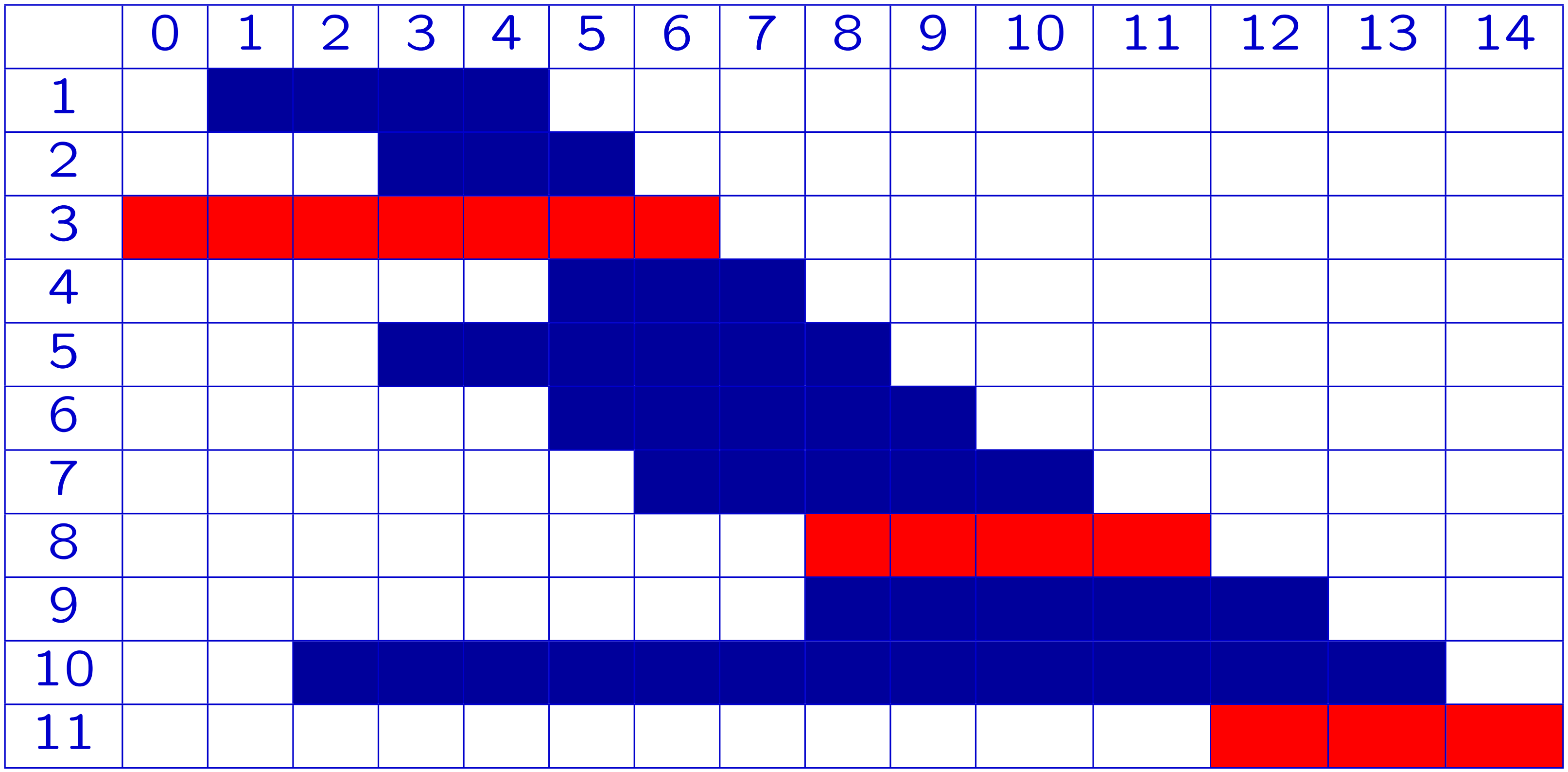
Podemos verificar três estratégias:

$\triangleright$ 1^a tentativa: Escolher primeiro as atividades que começam primeiro.
$\triangleright$ 2^a tentativa: Escolher primeiro as atividades que demoram menos tempo.
$\triangleright$ 3^a tentativa: Escolher primeiro as atividades que terminam primeiro.

Problema da seleção de atividades

1^a tentativa: Escolher primeiro as atividades que começam primeiro.

Exemplo: Considerando 11 atividades em 14 unidades de tempo

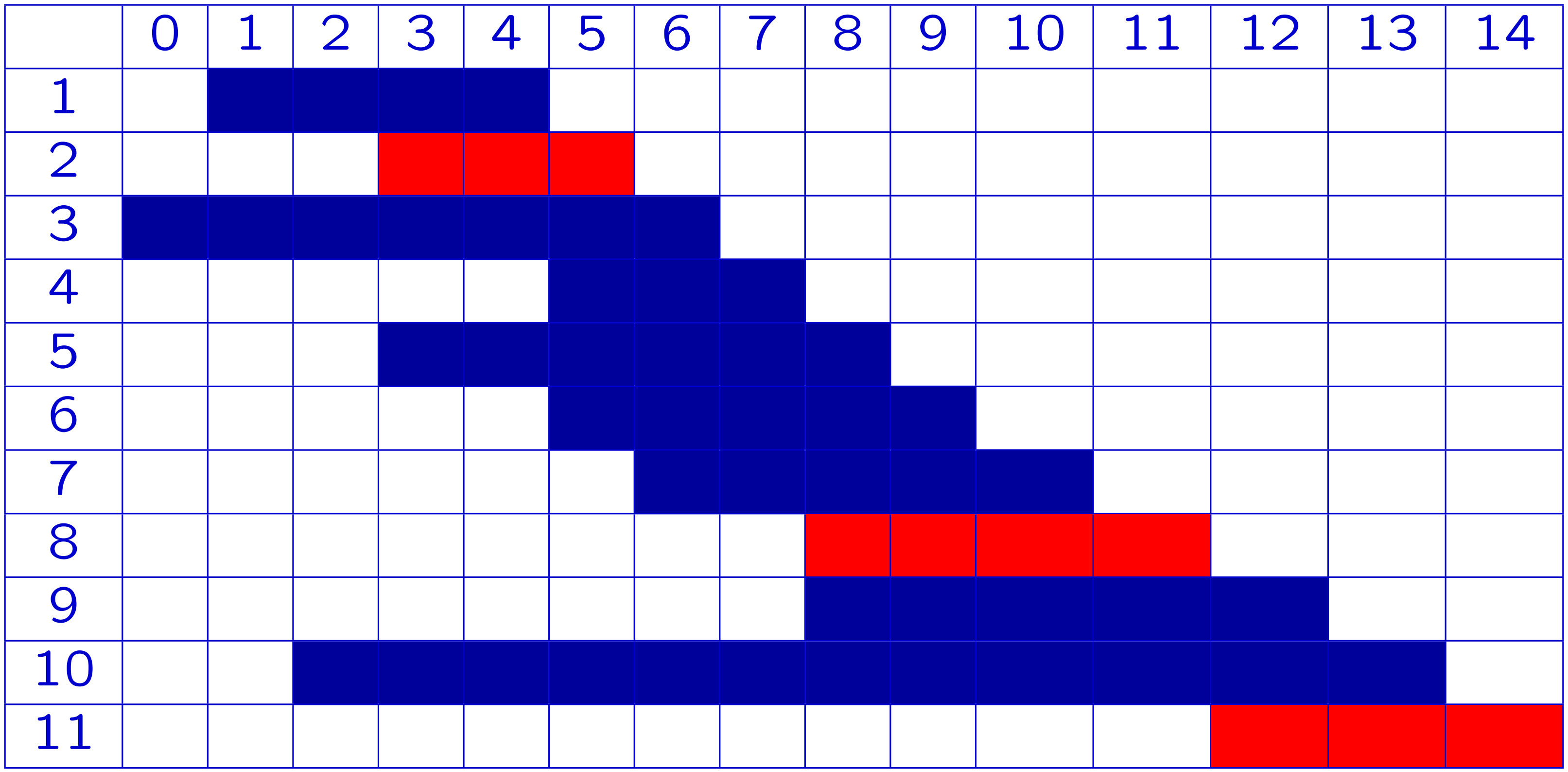


1ª tentativa: Escolher primeiro as atividades que começam primeiro. Escolhemos as atividades 3, 8 e 11.

Problema da seleção de atividades

2^a tentativa: Escolher primeiro as atividades que demoram menos tempo.

Exemplo: Considerando 11 atividades em 14 unidades de tempo

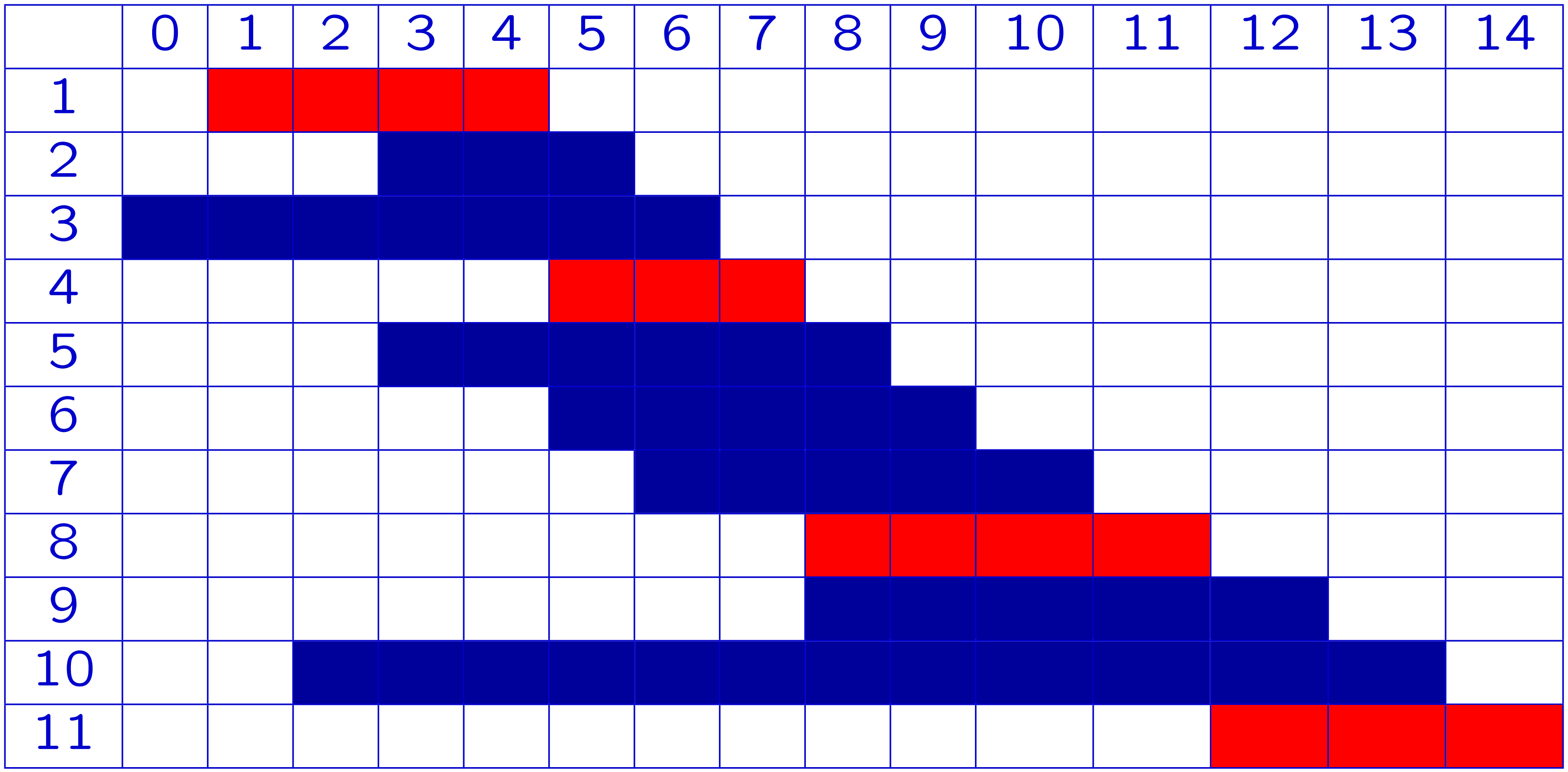


2^a tentativa: Escolher primeiro as atividades que demoram menos tempo. Escolhemos as atividades 2, 8 e 11.

Problema da seleção de atividades

3^a tentativa: Escolher primeiro as atividades que terminam primeiro.

Exemplo: Considerando 11 atividades em 14 unidades de tempo



3^a tentativa: Escolher primeiro as atividades que terminam primeiro. Escolhemos as atividades 1, 4, 8 e 11.

Problema da seleção de atividades

Exemplo: Considerando 11 atividades em 14 unidades de tempo

▷ 1^a tentativa: Escolher primeiro as atividades que começam primeiro.
▷ 2^a tentativa: Escolher primeiro as atividades que demoram menos tempo.
▷ 3^a tentativa: Escolher primeiro as atividades que terminam primeiro.

A 3^a tentativa apresentou a melhor estratégia de seleção de atividade. Dizemos que a solução é **ótima** para esta instância!

Problema da seleção de atividades

seleciona_atividades_guloso(s, t, n)

ordene s e t de tal forma que
$t[1] \leq t[2] \leq \cdots \leq t[n]$
$A \leftarrow \{1\}$ $\quad \rhd$ $t_1 \leq t_2 \leq \cdots \leq t_n$
$j \leftarrow 1$
para $i = 2$ **até** n
    faça se $s_i \geq t_j$ $\quad \rhd$ i e j são compatíveis
        então $A \leftarrow A \cup \{i\}$
            $j \leftarrow i$
retorne A

Complexidade do algoritmo: $O(n \lg n)$

Estratégia gulosa

Ingredientes chaves de um algoritmo guloso:

- ▷ subestrutura ótima
- ▷ característica gulosa: Solução ótima global pode ser produzida a partir de uma escolha ótima local.

O problema do troco

O problema do troco

Dados os valores de moedas (cédulas e moedas) de um país, determinar o mínimo de moedas para dar um valor de troco.

O problema do troco

Escolha gananciosa: devolver o número mínimo de moedas de mais alto valor cuja soma total resulta no valor de determinado troco. O algoritmo guloso encontra a solução ótima, isto é, o troco com o menor número de moedas.

O problema do troco

Exemplo: Dadas as moedas $= \{100, 50, 25, 10, 1\}$ e o valor do troco $= 37$, qual o número mínimo de moedas necessárias para resultar o valor do troco ?

$\triangleright$ **entrada:** conjunto $C = \{100, 50, 25, 10, 1\}$ de moedas em ordem decrescente e o valor do troco.
$\triangleright$ **saída:** conjunto S com as moedas utilizadas.

O problema do troco

Exemplo: Dadas as moedas $= \{100, 50, 25, 10, 1\}$ e o valor do troco = 37, qual o número mínimo de moedas necessárias para resultar o valor do troco ?

Idéia: Em cada estágio adicionamos a moeda de maior valor possível, de forma a não ultrapassar a quantidade necessária para o valor do troco.

O problema do troco

Exemplo: Dadas as moedas $= \{100, 50, 25, 10, 1\}$ e o valor do troco = 37, qual o número mínimo de moedas necessárias para resultar o valor do troco ?

Podemos fazer:

- $\triangleright$ $37 - 25 = 12$
- $\triangleright$ $12 - 10 = 2$
- $\triangleright$ $2 - 1 = 1$
- $\triangleright$ $1 - 1 = 0$

O total de 4 moedas: $\{25, 10, 1, 1\}$ **(solução ótima)**

O problema do troco

troco$(C, valor)$

$S \leftarrow \{\}$ $\quad soma \leftarrow 0$
enquanto $(soma < valor)$ **e** (C não vazio) **faça**
    $m \leftarrow$ moeda de maior valor em C
    se $soma + m \leq valor$ **então**
        $soma \leftarrow soma + m$
        $S \leftarrow S \cup \{m\}$
    senão
        $C \leftarrow C - \{m\}$
se $soma = valor$ **então**
    retorne S
senão
    retorne “não encontrei a solução”

Problema da mochila

Problema da mochila

Dados:

▷ Uma mochila que possui uma certa capacidade W (admite um peso).

▷ Um conjunto de n objetos distintos, enumerados de 1 a n, cada um com um certo valor $v_1, v_2, \cdots, v_n$ e peso $w_1, w_2, \cdots, w_n$.

Objetivo:

▷ Maximizar o valor do conjunto de objetos dentro da mochila respeitando a capacidade.

Problema da mochila

Há duas variações para o problema da mochila

- ▷ Mochila fracionária
- ▷ Mochila binária ou $0-1$

Problema da mochila fracionária

Os objetos podem ser particionados, ou seja, você pode colocar uma fração do objeto na mochila.

ex.: ouro em pó

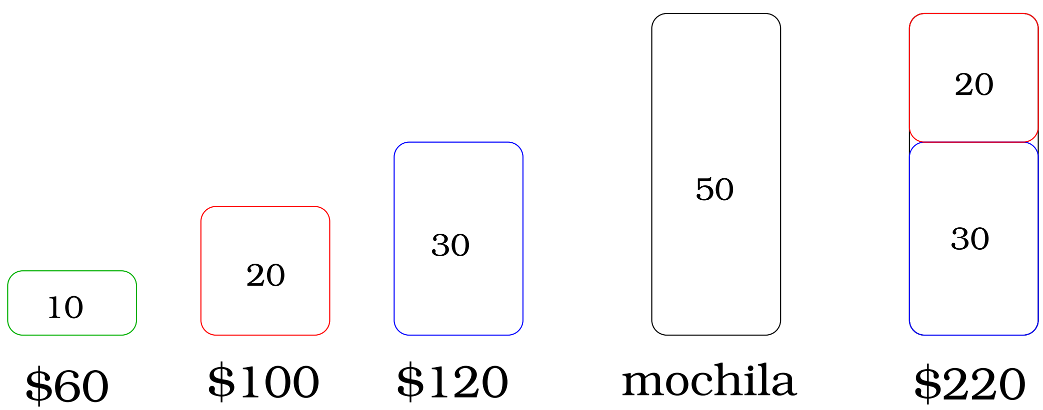
Problema da mochila binária

Os objetos não podem ser particionados, você só pode colocar itens inteiros e apenas uma vez.

ex.: ouro em barra

Problema da mochila

- Ambos tem sub-estrutura ótima.
- O problema da mochila fracionária tem solução gulosa, o problema da mochila binária não.



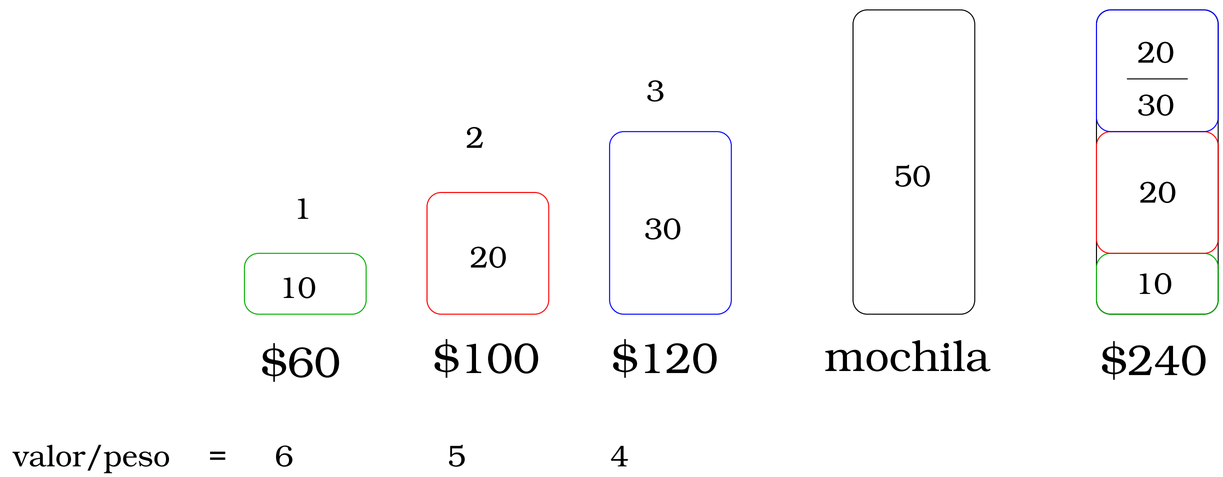
Problema da mochila fracionária

Ideia do algoritmo:

- ▷ Ordene os itens por $valor/peso$ em ordem decrescente.
- ▷ Começando em $i = 1$ coloque na mochila o máximo do item i que estiver disponível e for possível.
- ▷ Se a capacidade da mochila permitir passe para o próximo item.

Problema da mochila fracionária

Item	Valor	Peso	Valor/Peso
1	$60	10	6
2	$100	20	5
3	$120	30	4



Problema da mochila fracionária

```
mochila_fracionaria(w, v, n, W)
1    ordene w e v de tal forma que
     v[1]/w[1] ≥ v[2]/w[2] ≥ ··· v[n]/w[n]
2    para i = 1 até n faça
3        se w[i] ≤ W
4                então  x[i] ← 1
5                       W ← W − w[i]
6                senão  x[i] ← W/w[i]
7                       W ← 0
8    retorne x
```

Consumo de tempo na linha 1: $\Theta(n \lg n)$.

Consumo de tempo nas linhas 2-8: $\Theta(n)$.

Invariante

No início de cada execução da linha 2 vale que

$x' = x[1 \cdots i-1]$ é mochila ótima para $(w', v', i-1, W)$

onde

$$w' = w[1 \cdots i-1]$$
$$v' = v[1 \cdots i-1]$$

Na última iteração $i = n$ e portanto $x[1 \cdots n]$ é mochila ótima para (w, v, n, W)

Conclusão

$\boxed{\text{O consumo de tempo do algoritmo mochila\_fracionaria é } \Theta(n \lg n).}$

Obrigado