HEURÍSTICAS CONSTRUTIVAS

Thiago Noronha (tfn@dcc.ufmg.br)

HEURÍSTICAS CONSTRUTIVAS

- Constroem uma solução para o problema
- Partem unicamente dos parâmetros da instância

HEURÍSTICAS CONSTRUTIVAS

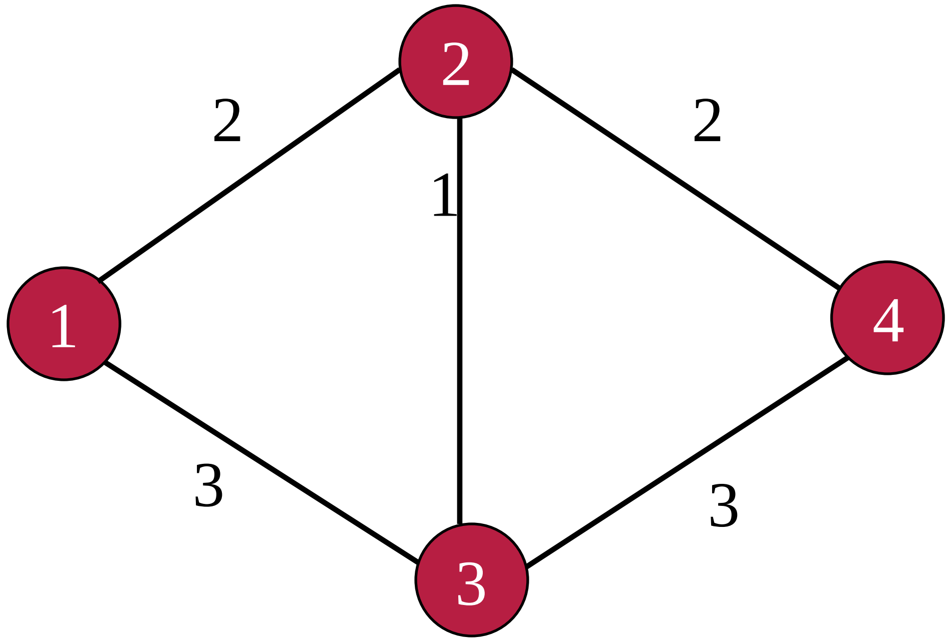
- Resolvem o problema de localização
  - associado ao problema de otimização
  - Garante a viabilidade, mas não a otimalidade
- Complexidade polinomial

O problema de localização associado pode ser NP-Difícil

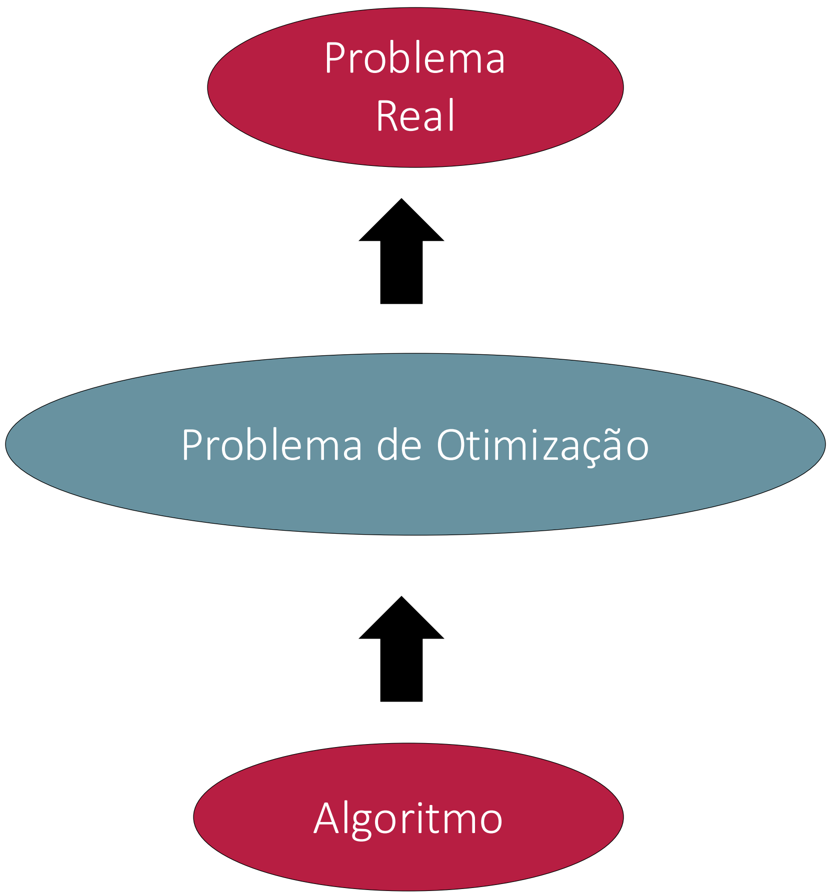
PROBLEMAS CUJA VERSÃO DE LOCALIZAÇÃO É NP-DIFÍCIL

Exemplo: Problema do caixeiro viajante em um grafo qualquer

- É NP-Completo decidir se existe um ciclo hamiltoniano em uma grafo qualquer



PROBLEMAS CUJA VERSÃO DE LOCALIZAÇÃO É NP-DIFÍCIL



Nem sempre é possível projetar uma heurística construtiva polinomial

PROBLEMAS CUJA VERSÃO DE LOCALIZAÇÃO É NP-DIFÍCIL

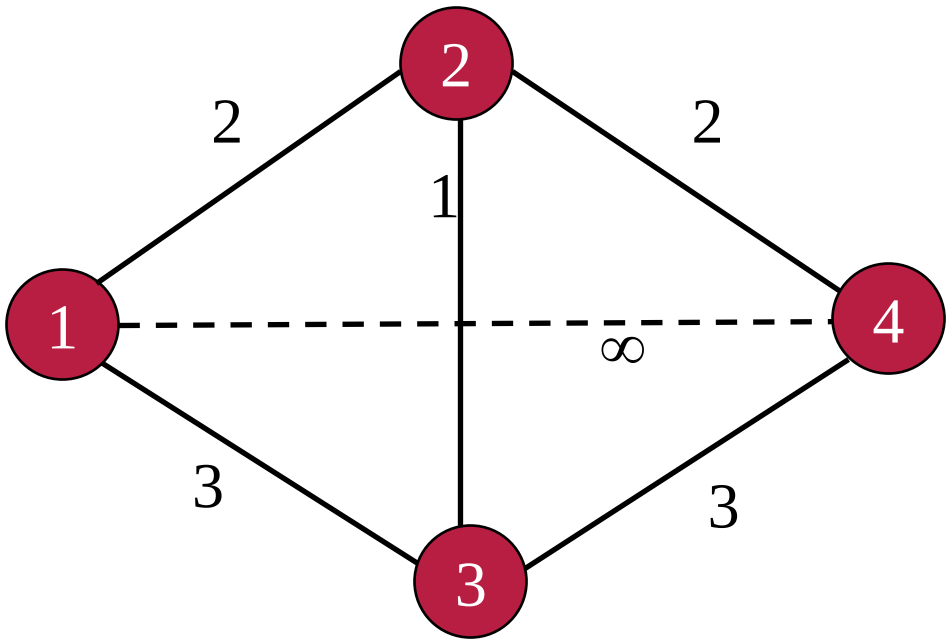
Quando necessário, ...

Sempre vamos transformar o nosso problema em um problema de otimização cuja versão de localização é polinomial

SOLUÇÃO

Problema do caixeiro viajante em um grafo qualquer

- Neste caso, reduz-se o problema a um grafo completo



CONSIDERAÇÕES FINAIS

- Não garantem a qualidade no pior caso
  - São projetadas para obter boas soluções no caso médio
- São projetadas *ad-hoc* para um problema específico
  - Não são facilmente generalizáveis

CONSIDERAÇÕES FINAIS

Podem ser projetadas a partir de qualquer paradigma de projeto

PARADIGMAS DE PROJETO DE ALGORITMOS

para problemas NPH

- Podem ser projetadas a partir de qualquer paradigma de projeto de algoritmos
  - Algoritmos gulosos
  - Programação dinâmica
  - Dividir e conquistar
  - Etc.

HEURÍSTICAS GULOSAS

Paradigma mais simples e mais utilizada no projeto de algoritmos

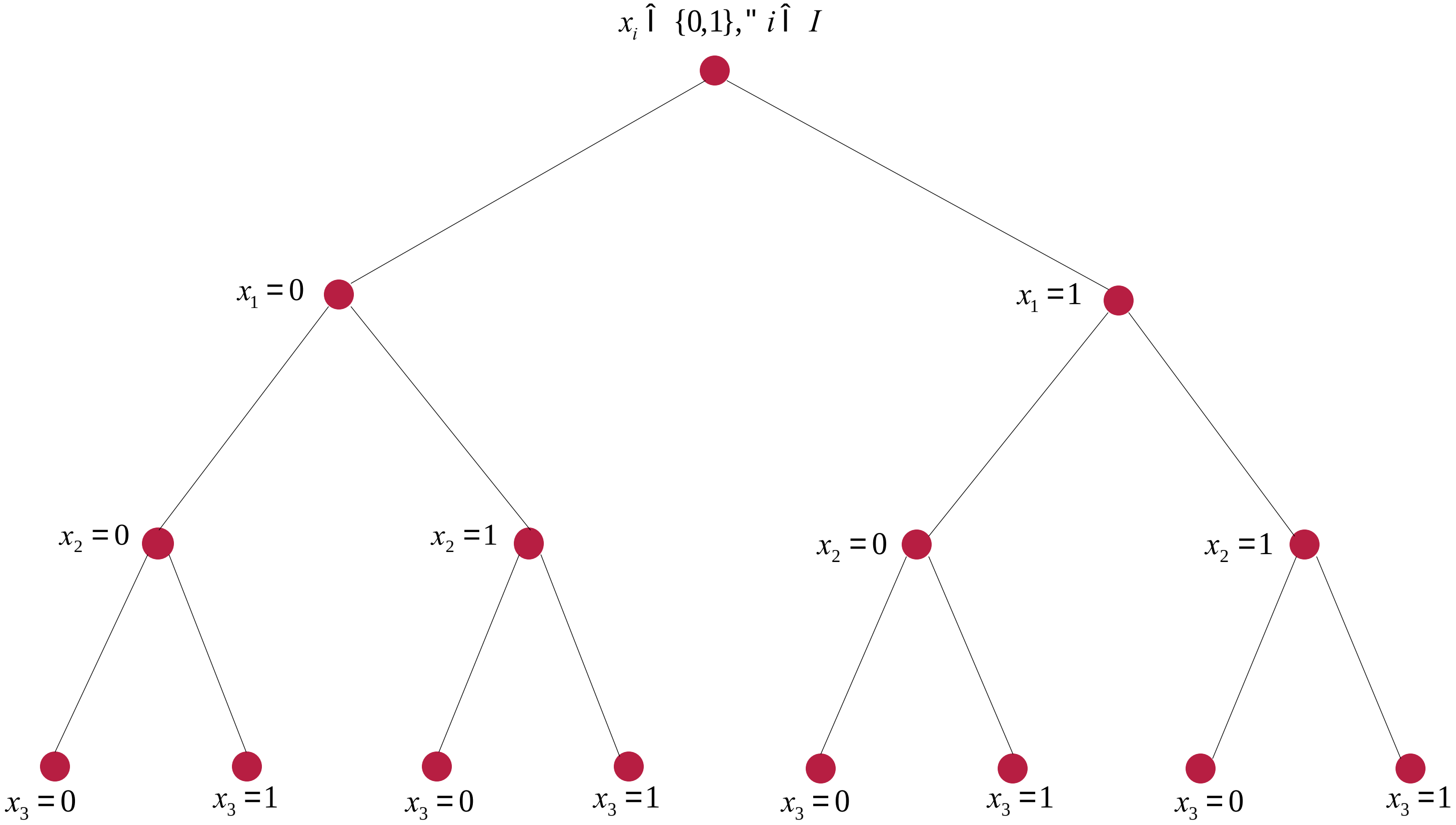
- São inerentemente recursivos
- Decompõem o problema em um único subproblema

- Uma decisão é tomada de forma gulosa
- As demais decisões são delegadas ao subproblema

- Nem sempre uma decisão que é ótima localmente é ótima globalmente
- Podem retornar soluções subótima

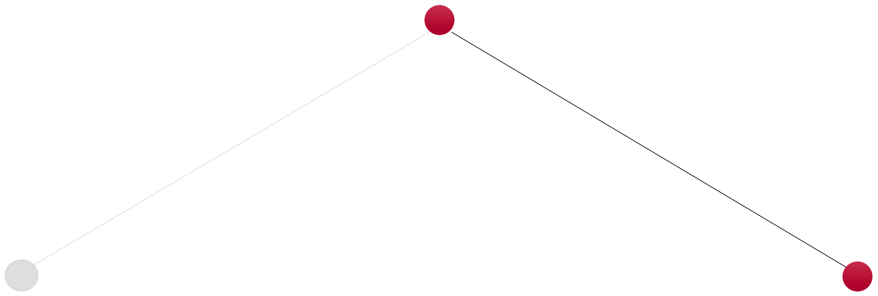
ÁRVORE DE DECISÃO

Exemplo: problemas com |*I*| variáveis binárias

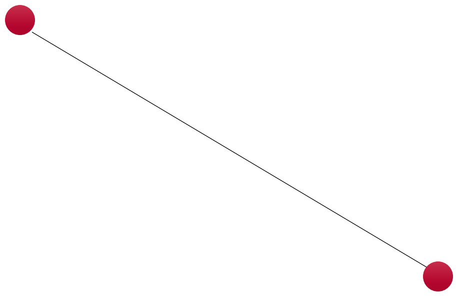


ALGORITMOS GULOSOS

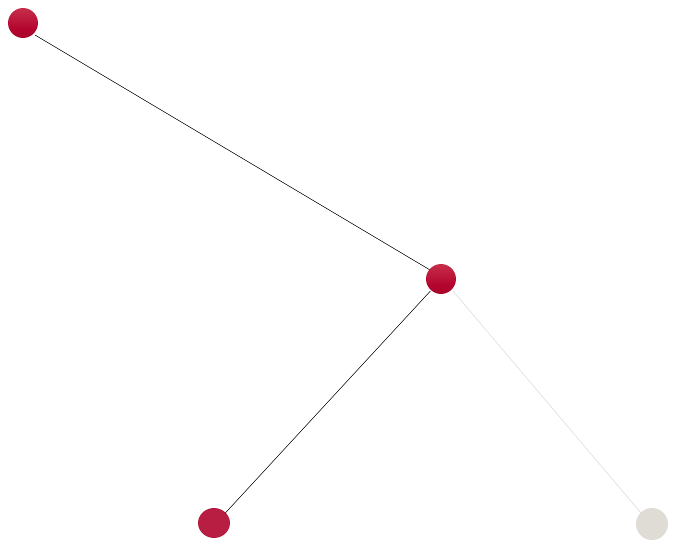
ALGORITMOS GULOSOS



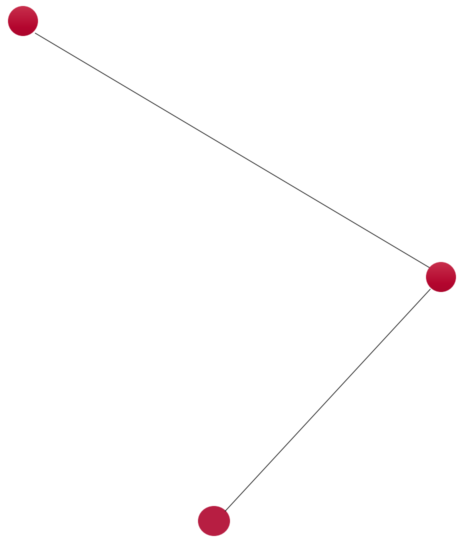
ALGORITMOS GULOSOS



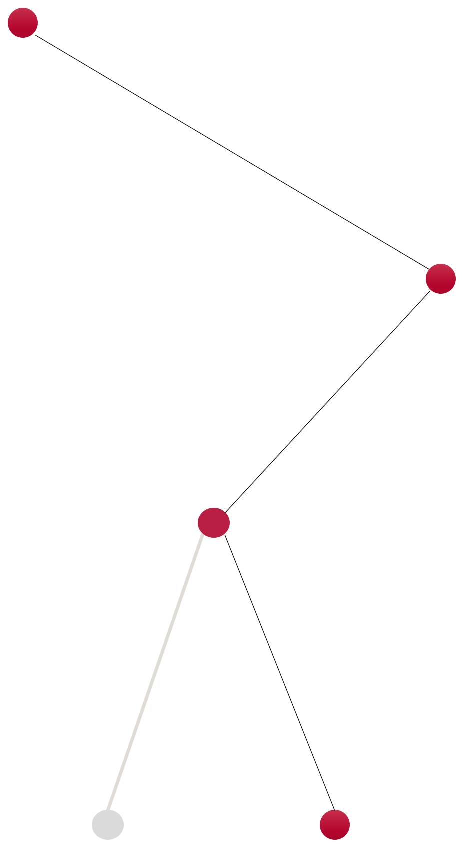
ALGORITMOS GULOSOS



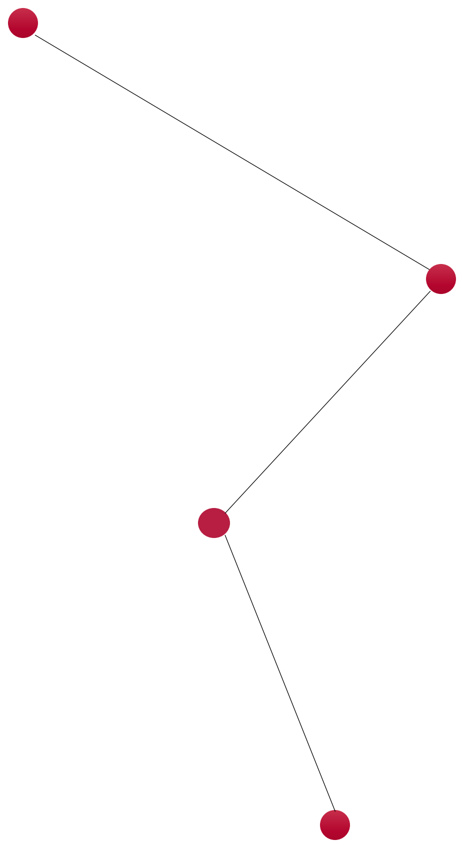
ALGORITMOS GULOSOS



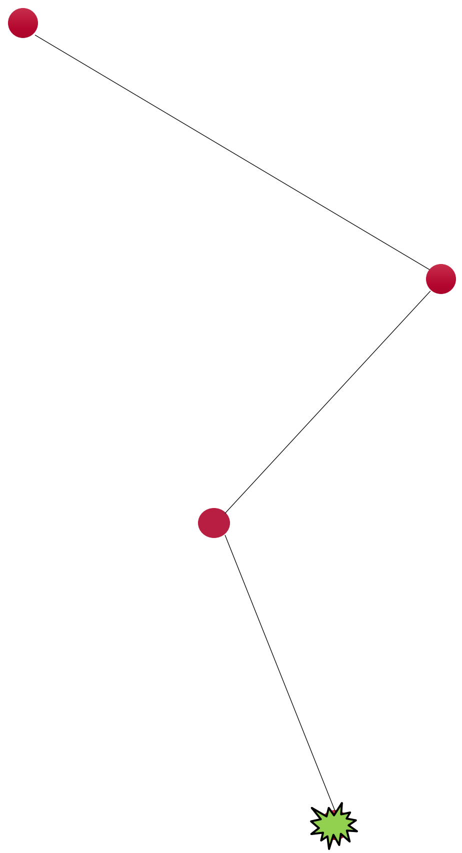
ALGORITMOS GULOSOS



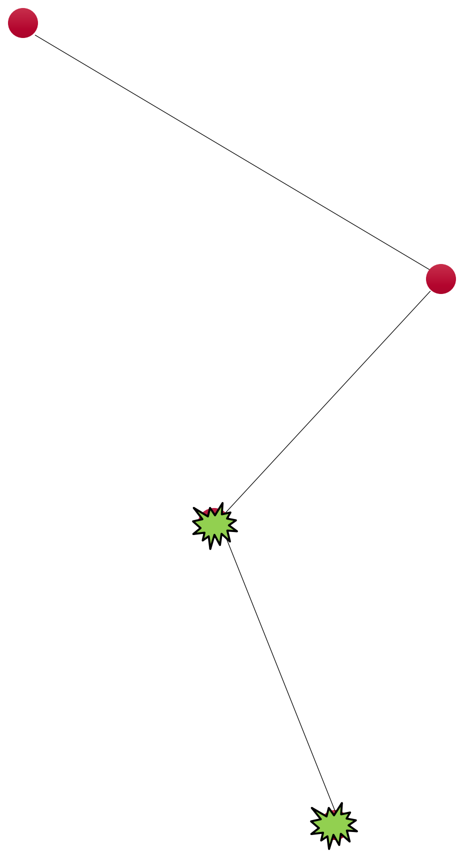
ALGORITMOS GULOSOS



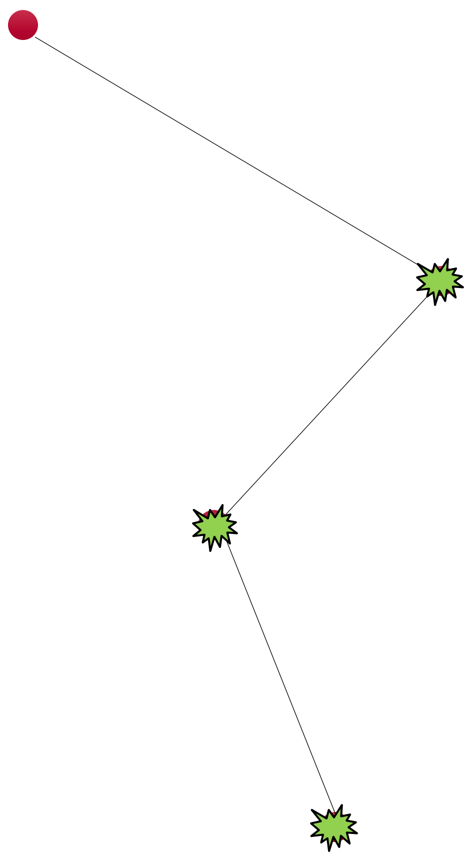
ALGORITMOS GULOSOS



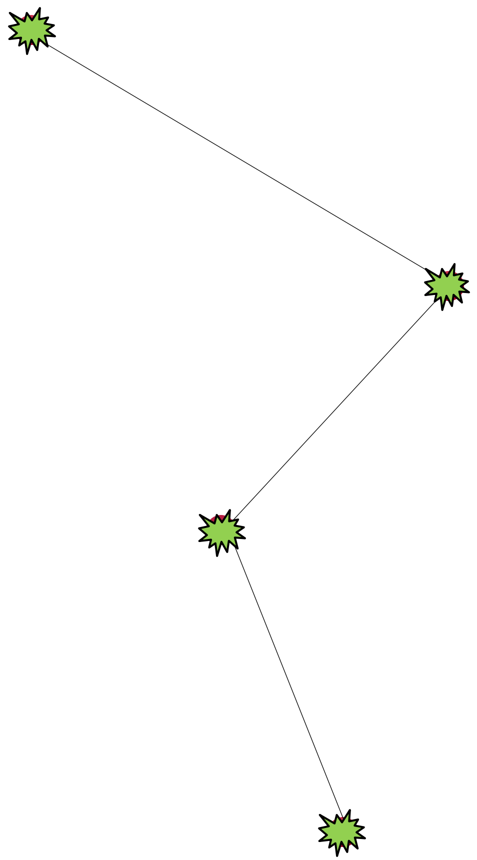
ALGORITMOS GULOSOS



ALGORITMOS GULOSOS



ALGORITMOS GULOSOS



ALGORITMOS GULOSOS

Etapas fundamentais

- Decisão
- Recursão
- Combinação

ALGORITMOS GULOSOS

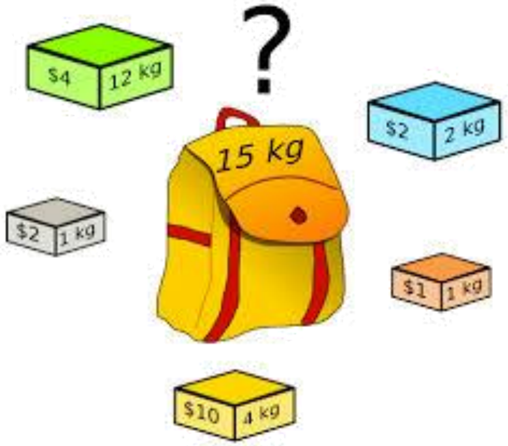
Para o problema da Mochila Binária

$$I = \{1, 2, \dots, |I|\}$$

$$a_i \in N$$

$$c_i \in N$$

$$B \in N$$



ALGORITMOS GULOSOS

Para o problema da Mochila Binária

- Decisão
  - $f(I, B)$: função gulosa
  - Retorna o item k com maior benefício/custo
  - $\{k\} = f(I, B) = \underset{i \in I : a_i \leq B}{\operatorname{argmax}} \frac{c_i}{a_i}$

ALGORITMOS GULOSOS

Para o problema da Mochila Binária

- Recursão
  - Se $\{k\} = \emptyset$, retorne $\emptyset$
  - Senão, retorne $\mathrm{AG}(\mathrm{I} \backslash \{\mathrm{k}\}, \mathrm{B} - \mathrm{a_k})$

- Combinação
  - Retorne $\{\mathrm{k}\} \cup AG(\mathrm{I} \backslash \{\mathrm{k}\}, B - a_k)$

HEURÍSTICA GULOSA

Para o problema da mochila

- Exemplo:

$$I = \{1, 2, 3, 4, 5\}$$
$$c_i = [20, 29, 38, 94, 5]$$
$$a_i = [20, 30, 40, 100, 10]$$
$$B = 100$$

$$v_i = [1.00, 0.97, 0.95, 0.94, 0.50]$$

ALGORITMOS GULOSOS RECURSIVOS

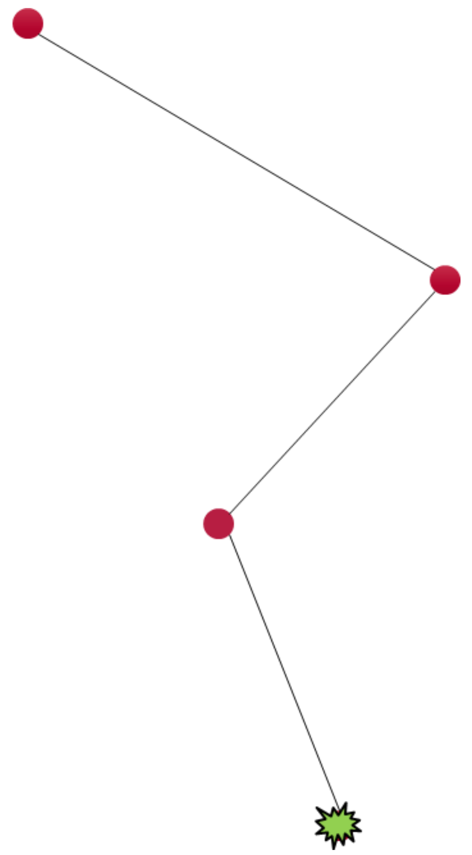
Etapas fundamentais

- Decisão
- Recursão
- Combinação

ALGORITMOS GULOSOS ITERATIVOS

Etapas do projeto de uma HCG

- Parte de uma solução parcialmente viável S
  - E.g. $S = \emptyset$
- Enquanto S não for viável e maximal:
  - Incrementa S



PRINCÍPIOS DE PROJETO

PRINCÍPIOS DE PROJETO DE HEURÍSTICAS GULOSAS

- Definição
- Exemplo
  - Corretude
  - Complexidade
  - Aproximação

PRINCÍPIO DE PROJETO

Inserir primeiro os elementos com o melhor custo benefício

EXEMPLO

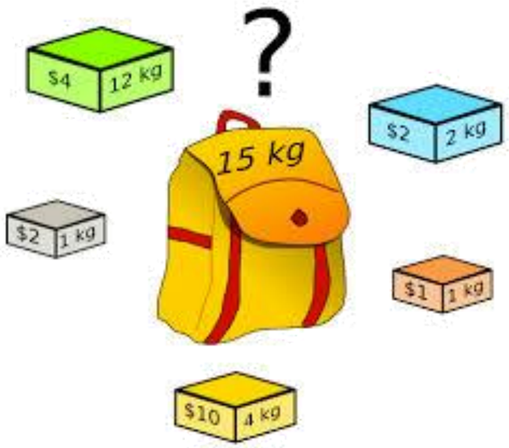
Problema da mochila

$$I = \{1, 2, \ldots, |I|\}$$

$$a_i \in N$$

$$c_i \in N$$

$$B \in N$$



PSEUDOCÓDIGO

Da Heurística gulosa

- Parte de uma solução parcial $S = \emptyset$
- Enquanto $\{\mathrm{k}\} \neq \emptyset$
  - Seleciona $\{\mathrm{k}\} = \underset{i \in I \setminus \mathrm{S}: a_i \leq B - a(S)}{\operatorname{argmax}} \frac{c_i}{a_i}$
  - Insere $\{\mathrm{k}\}$ em S
- Retorna S

PSEUDOCÓDIGO

Da Heurística gulosa Modificada

- Parte de uma solução parcial $S = \emptyset$
- Enquanto $a_k \leq B - a(S \cup \{\mathrm{k}\})$
  - Seleciona $\{\mathrm{k}\} = \underset{i \in I \backslash S}{\operatorname{argmax}} \frac{c_i}{a_i}$
  - Insere $\{\mathrm{k}\}$ em S se $a_k \leq B - a(S \cup \{\mathrm{k}\})$
- Retorna $\operatorname{argmax}(c(S), c(\{j\}))$
  - Onde $\{\mathrm{j}\} = \underset{i \in I \backslash S}{\operatorname{argmax}} \frac{c_i}{a_i}$

HEURÍSTICA GULOSA

Para o problema da mochila

- Exemplo:

$$I = \{1, 2, 3, 4, 5\}$$
$$c_i = [20, 29, 38, 94, 5]$$
$$a_i = [20, 30, 40, 100, 10]$$
$$B = 100$$

$$v_i = [1.00, 0.97, 0.95, 0.94, 0.50]$$

CORRETUDE

Da Heurística Gulosa

- $S = \emptyset$ é viável
- Para $\{k\} \neq \emptyset$, $a_k \leq B - a(S)$
  - Portanto, $S = S \cup \{k\}$ é viável
- Para $\{k\} = \emptyset$, retorna-se S
  - Que é viável por definição

- Trivial
  - $T(n) = O(n) \cdot O(n) = O(n^2)$

- Ordenando-se os itens
  - $T(n) = O(n\log n) + O(n) \cdot O(1)$
  - $T(n) = O(n\log n) + O(n)$
  - $T(n) = O(n\log n)$

- A Heurística Gulosa Modificada é ½-aproximativa

APROXIMAÇÃO

A Heurística Gulosa Modificada é ½-aproximativa

- A solução ótima da mochila fracionária é pelo menos tão boa quanto a da mochila binária
  - $C_F^* \geq C_B^*$

- HGM é ótimo para o problema da mochila fracionária
  - $C_F^* = c(S) + \varepsilon \cdot c_j$
    - $0 \leq \varepsilon < 1$

APROXIMAÇÃO

A Heurística Gulosa Modificada é ½-aproximativa

- HGM retorna $\max(S, \{j\})$
  - $C^{HGM} = \max(c(S), c_j)$

APROXIMAÇÃO

A Heurística Gulosa Modificada é ½-aproximativa

- $C^{HGM} = \max\left(c(S), c_j\right)$
- $2 \cdot C^{HGM} = 2 \cdot \max(c(S), c_j) \geq c(S) + c_j$
- $2 \cdot C^{HGM} \geq c(S) + c_j \geq c(S) + \varepsilon \cdot c_j$
- $2 \cdot C^{HGM} \geq c(S) + \varepsilon \cdot c_j = C_F^*$
- $2 \cdot C^{HGM} \geq C_F^* \geq C_B^*$
- $C^{HGM} \geq \frac{1}{2} C_B^*$

Maximizar a utilização dos recursos disponíveis

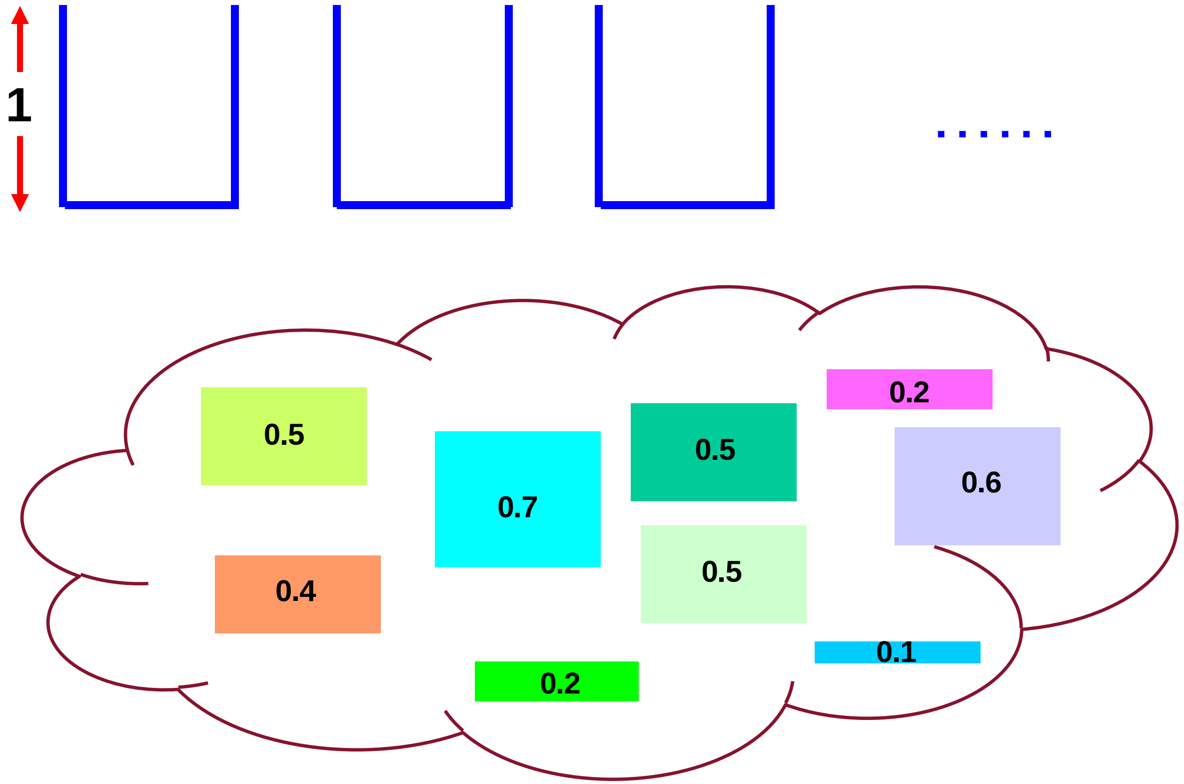
EXEMPLO

Problema de Empacotamento (Bin Packing)

- Dados
  - $I = \{1, \cdots, n\}$
  - $0 \leq a_i < 1$
- Empacotar os itens no menor número de caixas
- NP-Difícil

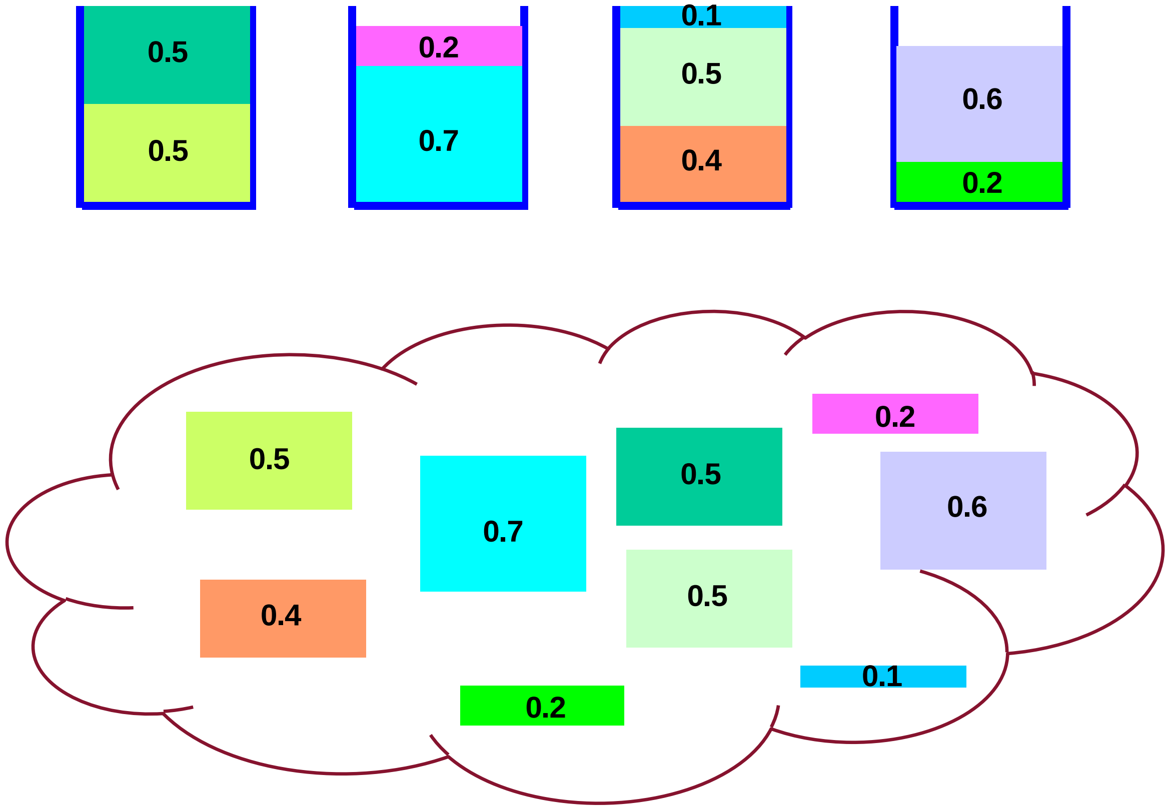
EXEMPLO

Problema de Empacotamento (Bin Packing)



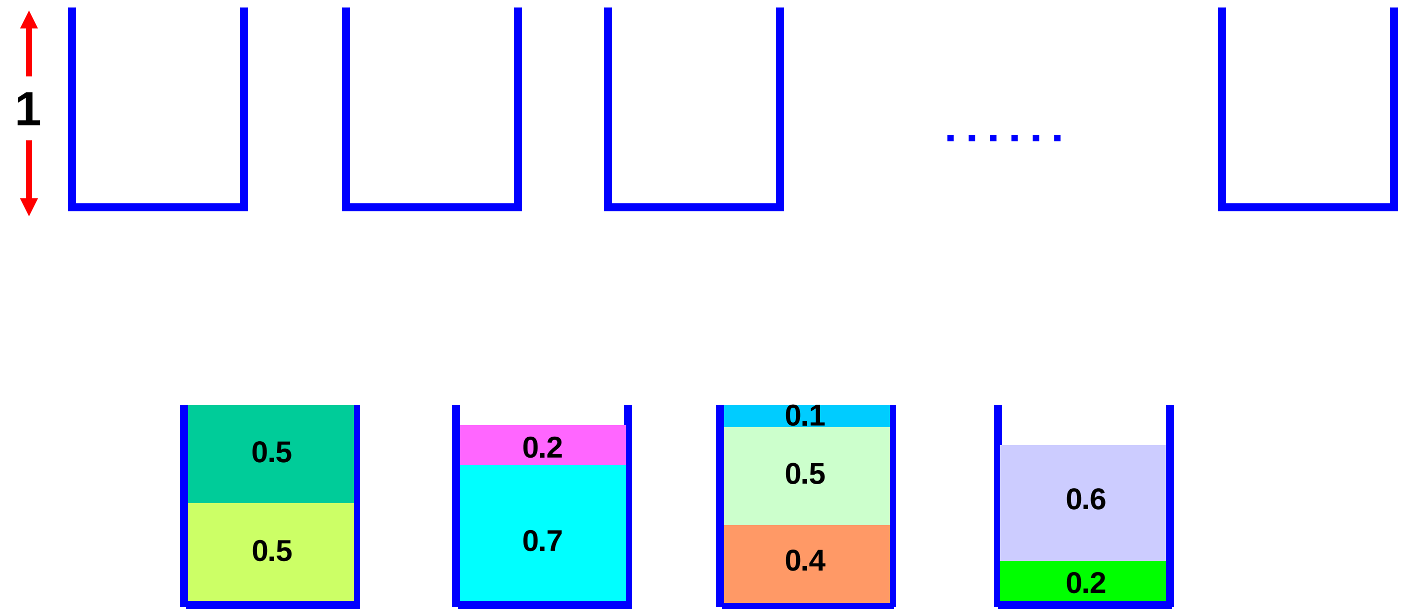
EXEMPLO

Problema de Empacotamento (Bin Packing)



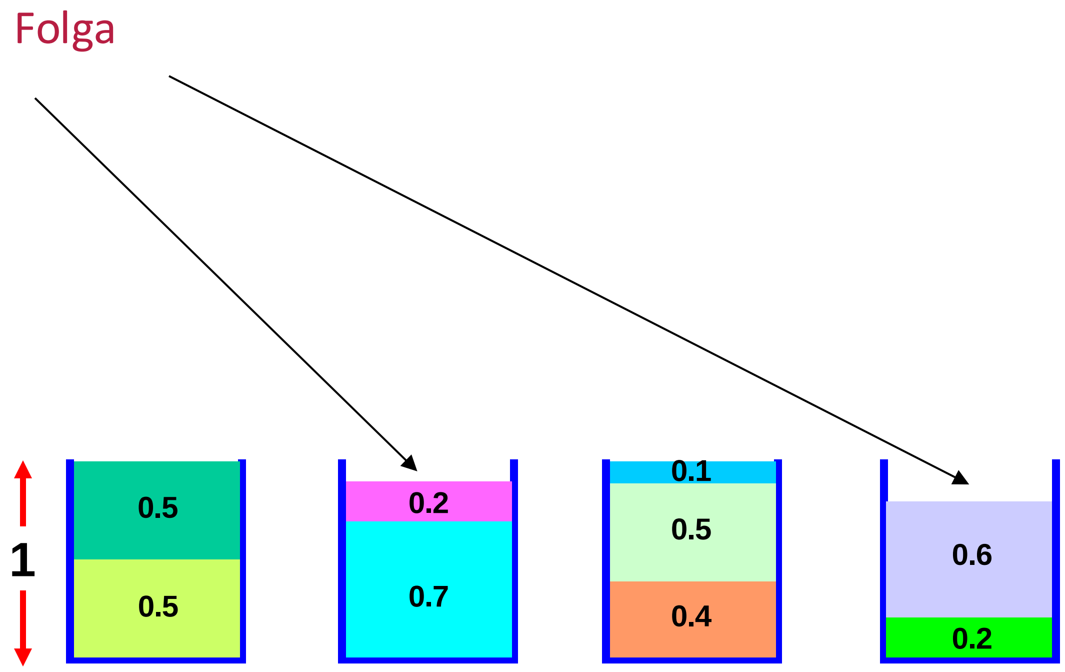
EXEMPLO

Problema de Empacotamento (Bin Packing)



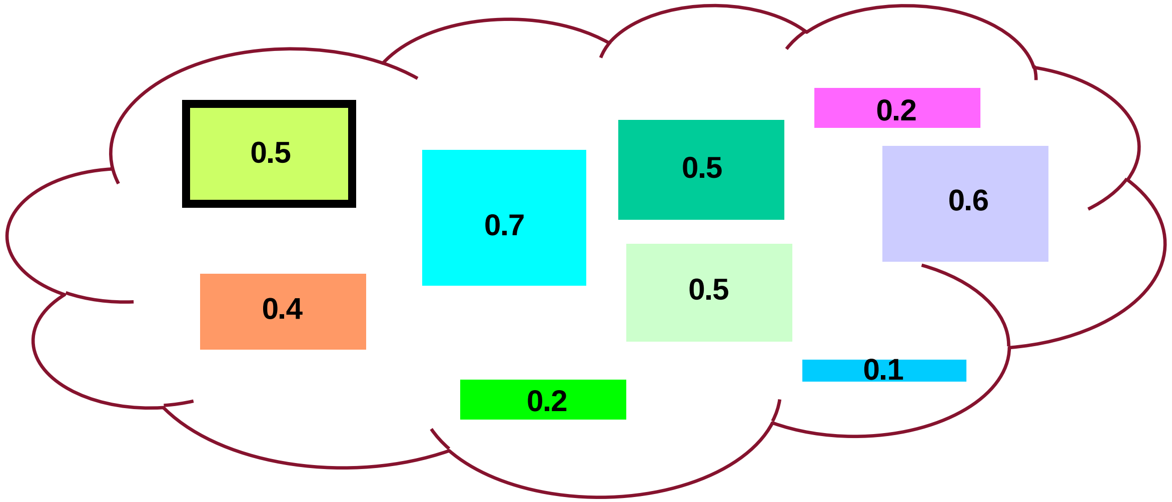
EXEMPLO

Problema de Empacotamento (Bin Packing)



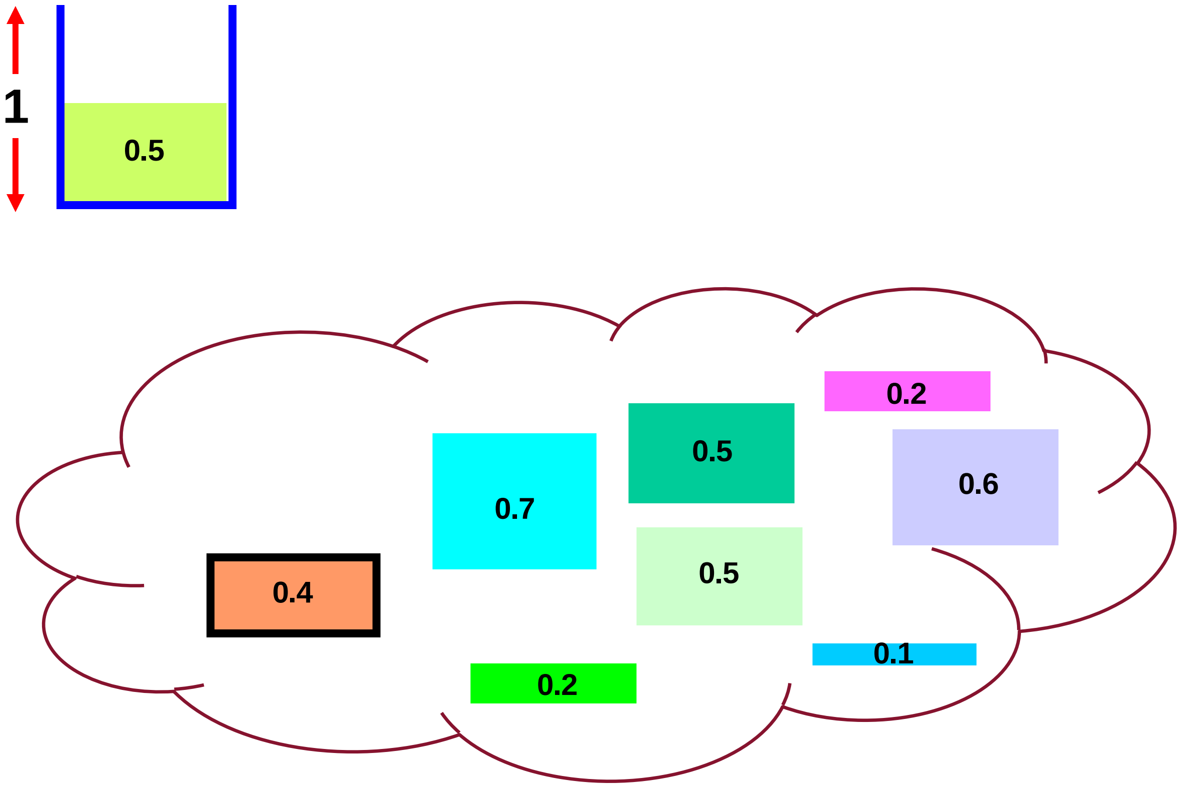
HEURÍSTICA DE FIRST-FIT (FF)

Para o problema de empacotamento



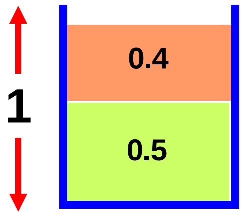
HEURÍSTICA DE FIRST-FIT (FF)

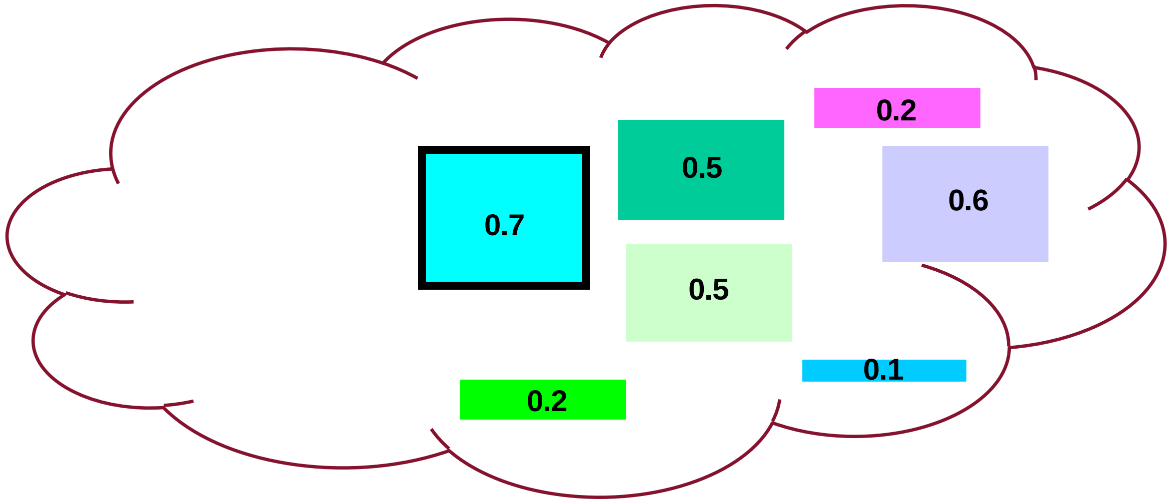
Para o problema de empacotamento



HEURÍSTICA DE FIRST-FIT (FF)

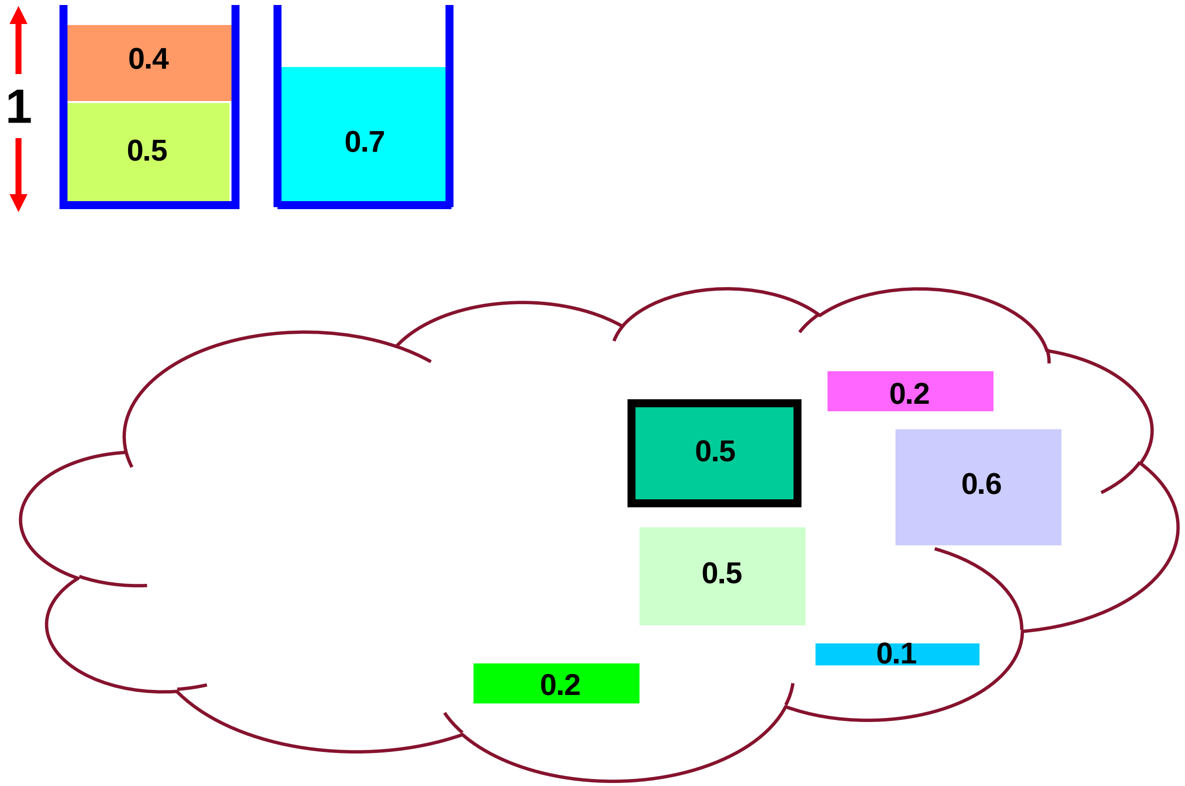
Para o problema de empacotamento





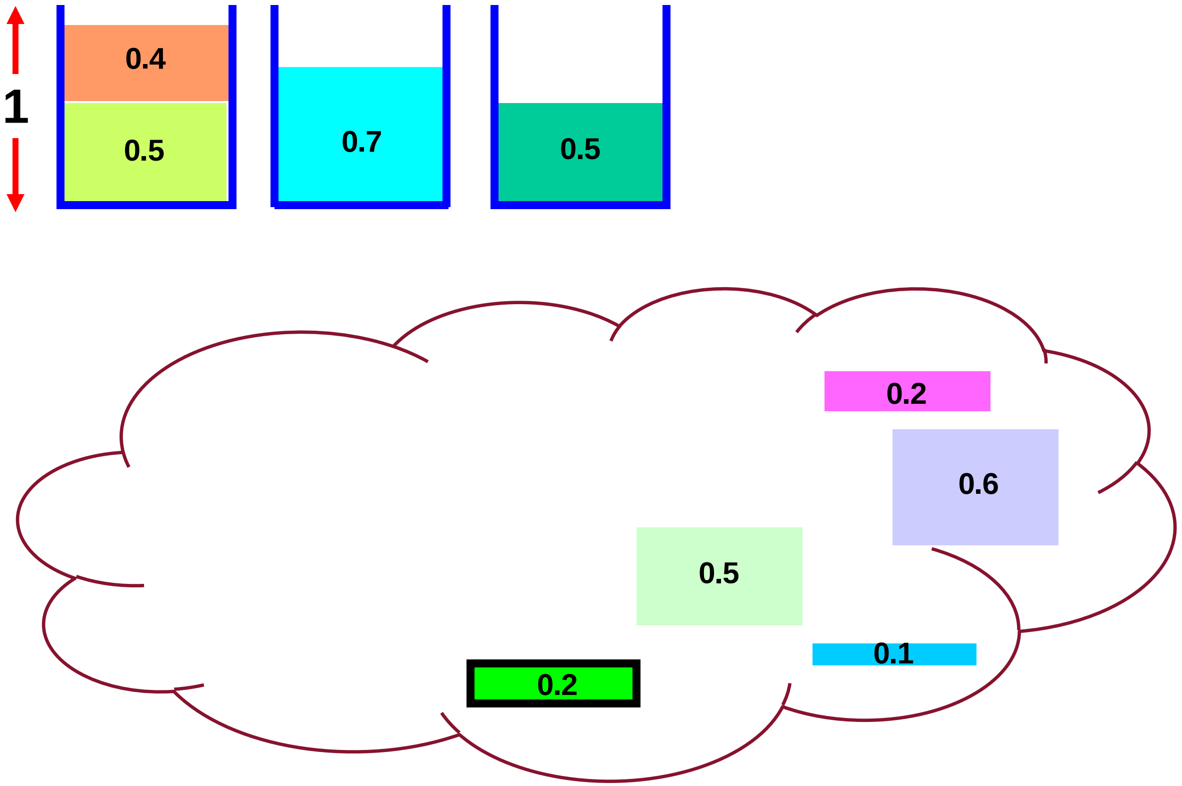
HEURÍSTICA DE FIRST-FIT (FF)

Para o problema de empacotamento



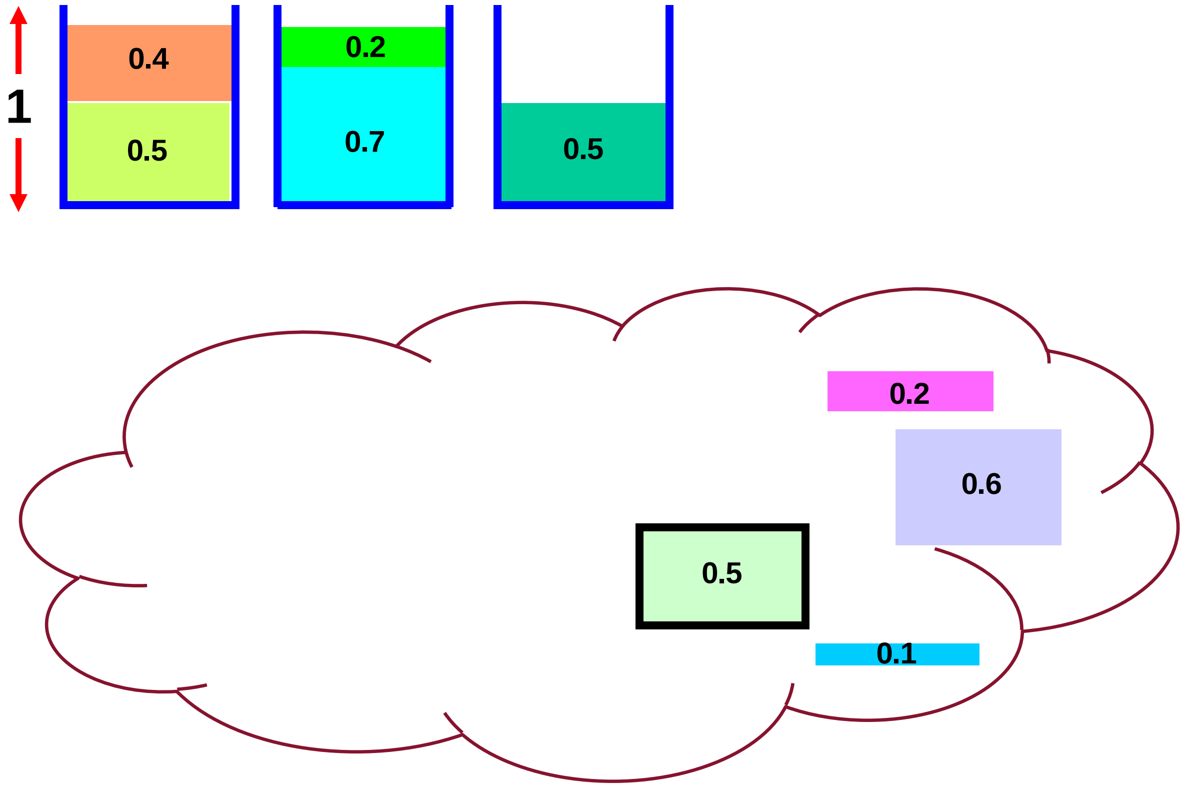
HEURÍSTICA DE FIRST-FIT (FF)

Para o problema de empacotamento



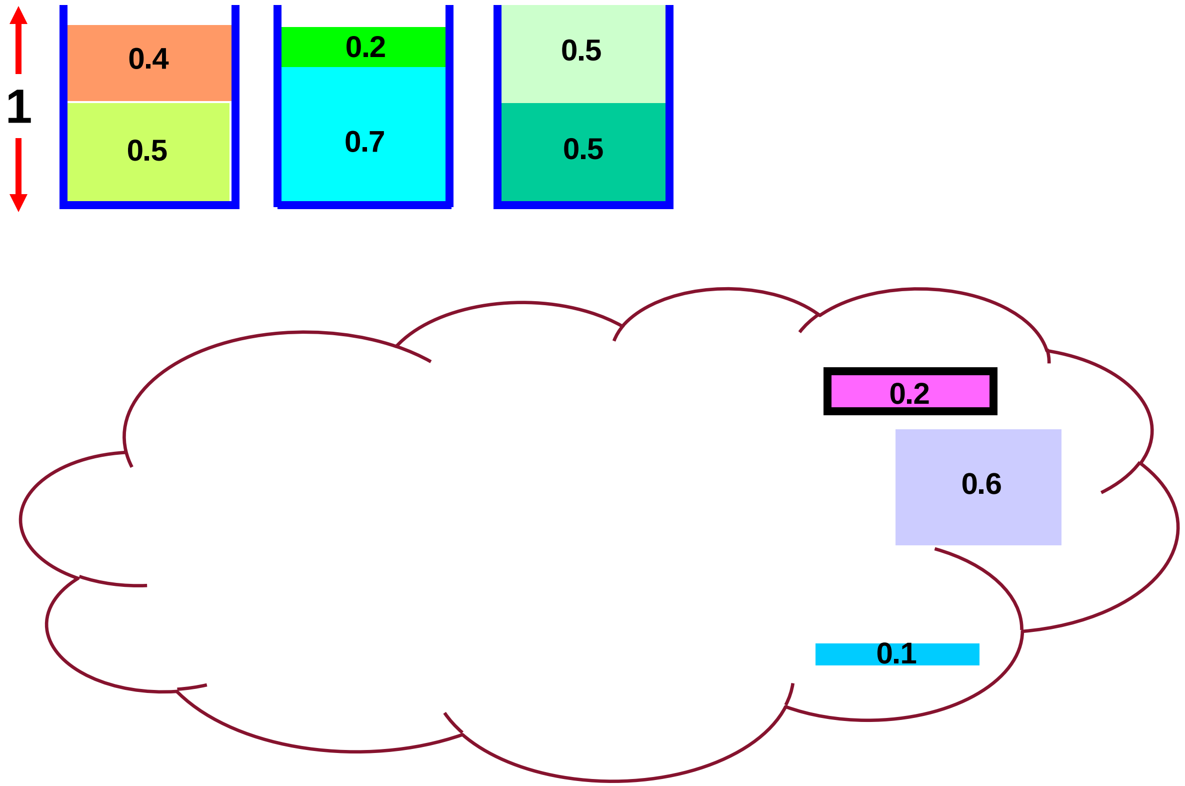
HEURÍSTICA DE FIRST-FIT (FF)

Para o problema de empacotamento



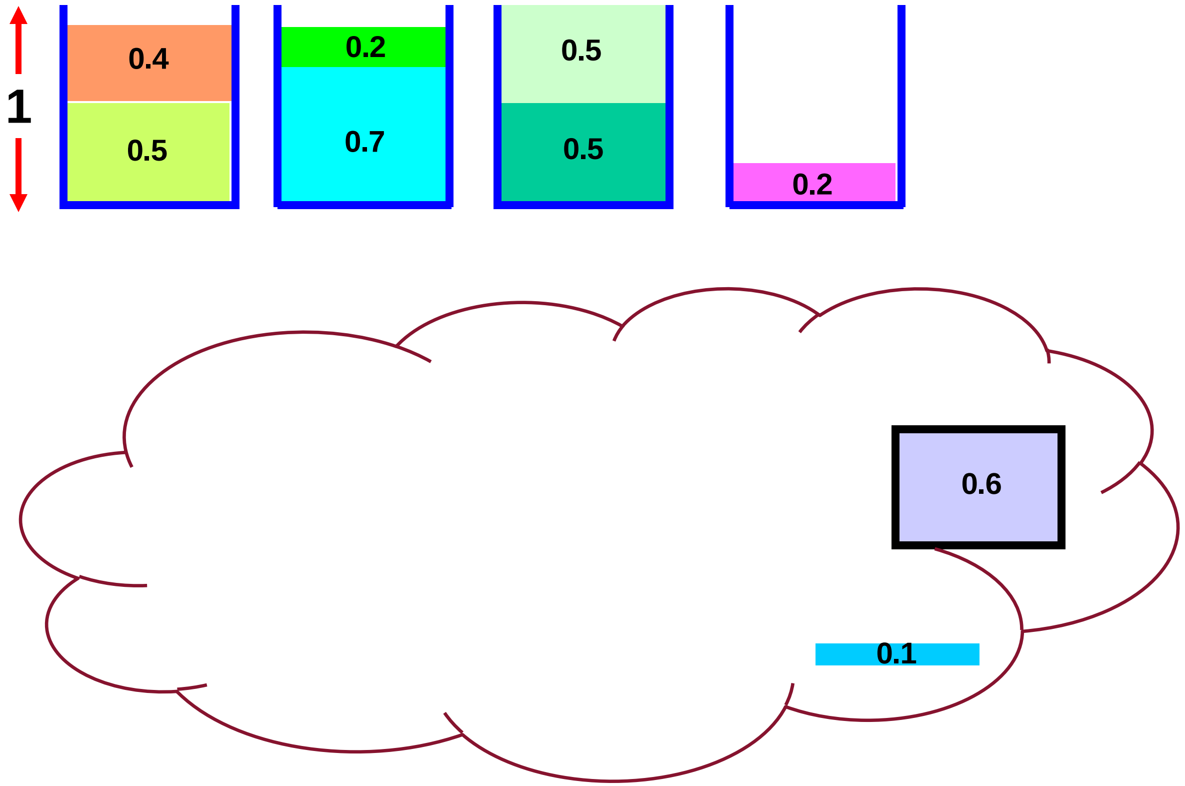
HEURÍSTICA DE FIRST-FIT (FF)

Para o problema de empacotamento



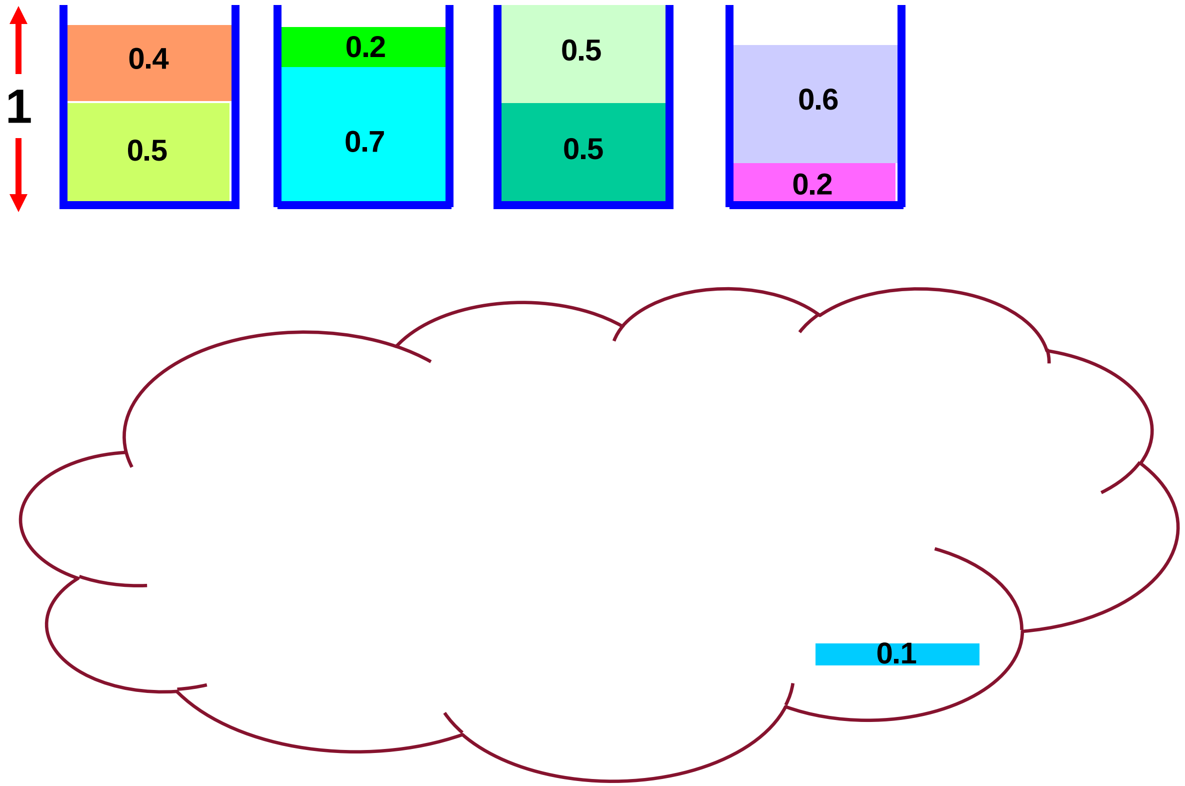
HEURÍSTICA DE FIRST-FIT (FF)

Para o problema de empacotamento



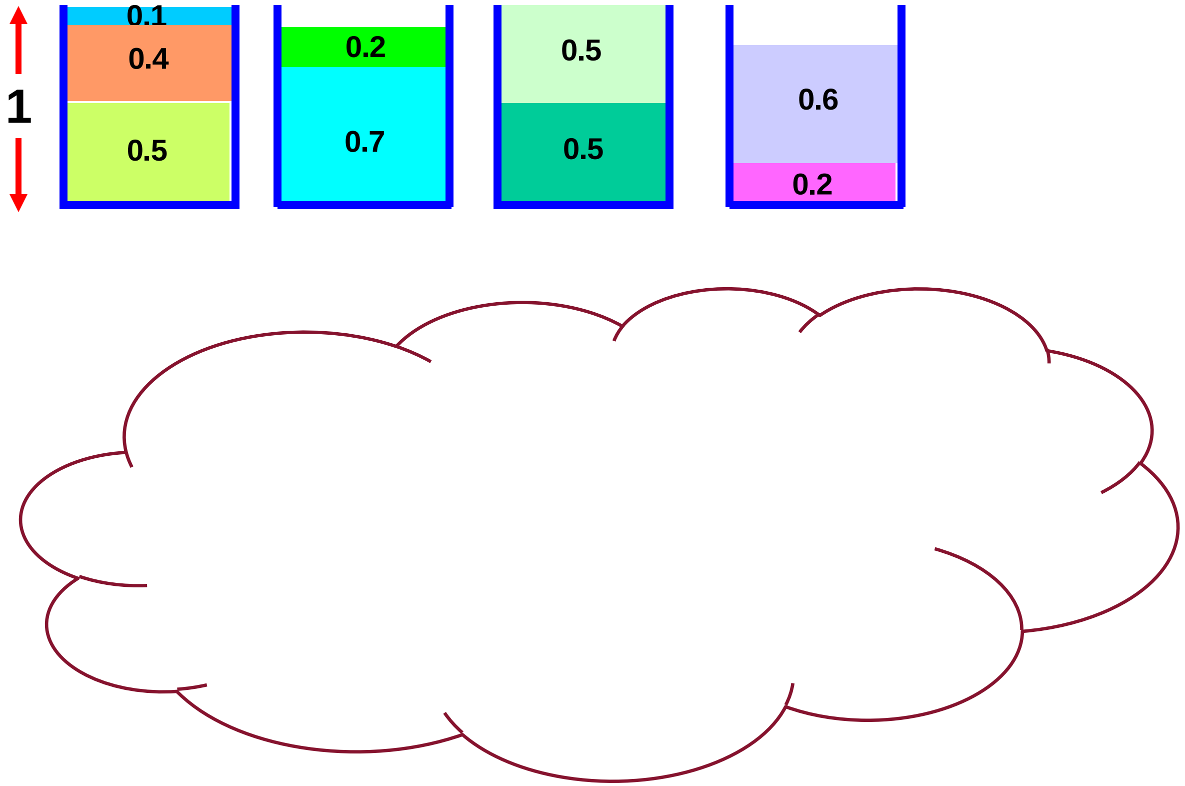
HEURÍSTICA DE FIRST-FIT (FF)

Para o problema de empacotamento



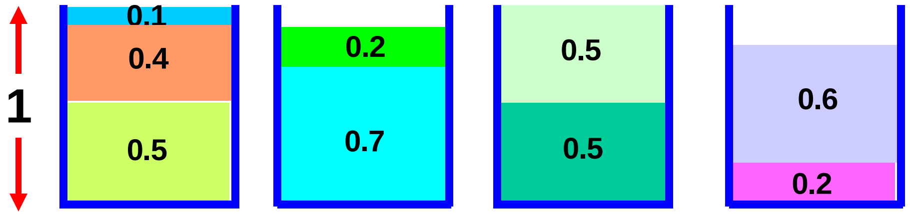
HEURÍSTICA DE FIRST-FIT (FF)

Para o problema de empacotamento



HEURÍSTICA DE FIRST-FIT (FF)

Para o problema de empacotamento



$$S = \begin{bmatrix} [0.1, 0.4, 0.5], \\ [0.2. 0.7], \\ [0.5, 0.5], \\ [0.6, 0.2] \end{bmatrix}$$

PSEUDOCÓDIGO

Da heurística First Fit

- Parte de uma solução parcial $S = []$
- Enquanto $I \backslash S \neq \emptyset$
  - Seleciona $\mathrm{k} \in I \backslash S$
  - Insere k na primeira caixa de S
    - Que comporta k
    - Se não existe nenhuma, adicione uma caixa vazia
- Retorne S

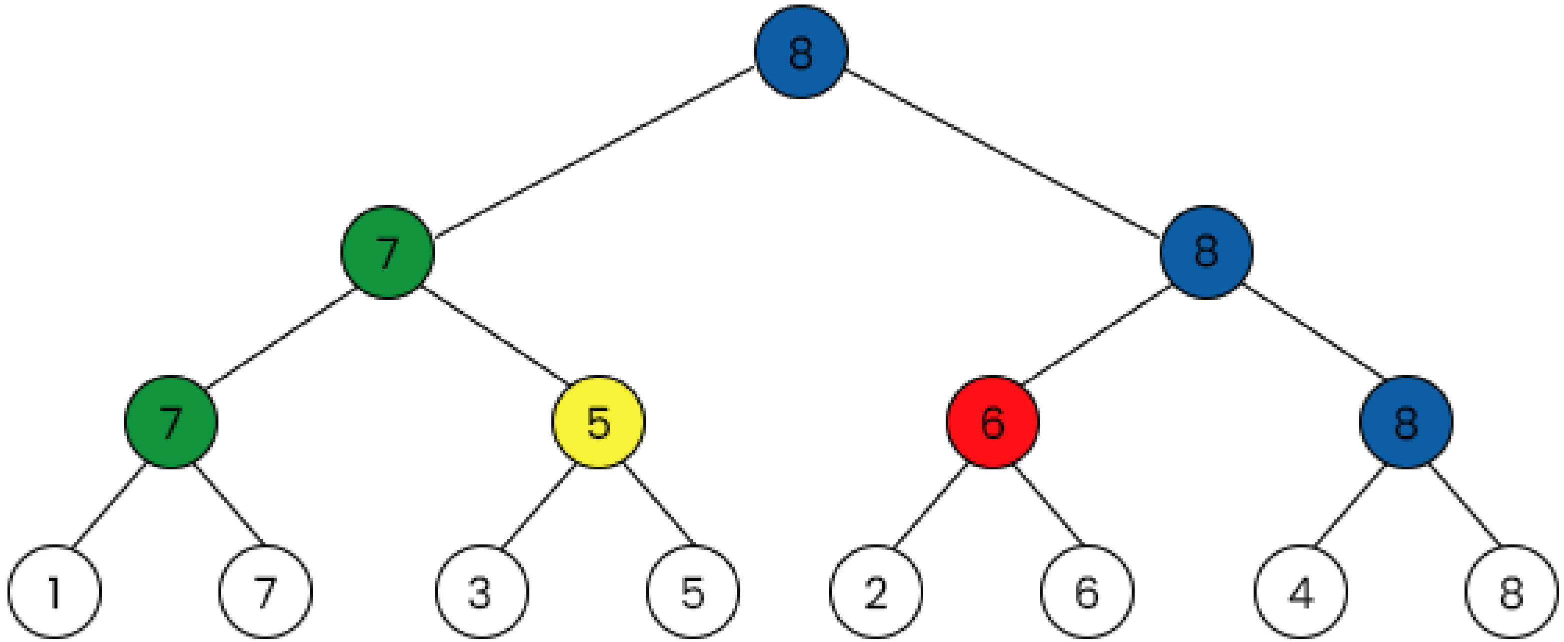
CORRETUDE

Da heurística First Fit

- $S = []$ é parcialmente viável
  - Nenhuma caixa violada
- $\mathrm{k} \in I \backslash \mathrm{S}$, por definição
  - é inserido em uma caixa que o comporta,
  - ou em uma nova caixa (que o comporta).
  - Portanto $\mathrm{S} \oplus k$ é parcialmente viável
- Para quando $I \backslash \mathrm{S} = \emptyset$
  - Portanto $\mathrm{S} = \mathrm{I}$, e S é viável

- Para uma seleção arbitrária de k
  - $T(n) = O(n) \cdot O(\Delta)$
  - $T(n) = O(n) \cdot O(n)$
  - $T(n) = O(n^2)$

- Usando *Árvores de Torneio*



Fone: https://www.javatpoint.com/tournament-tree

- Usando *Árvores de Torneio*
  - $T(n) = O(n) \cdot O(\log \Delta)$
  - $T(n) = O(n) \cdot O(\log n)$
  - $T(n) = O(n\log n)$

MELHOR APROXIMAÇÃO PARA FF

$$B \leq \frac{17}{10} B^* + 2$$

APROXIMAÇÃO

- FF é 2-aproximativa

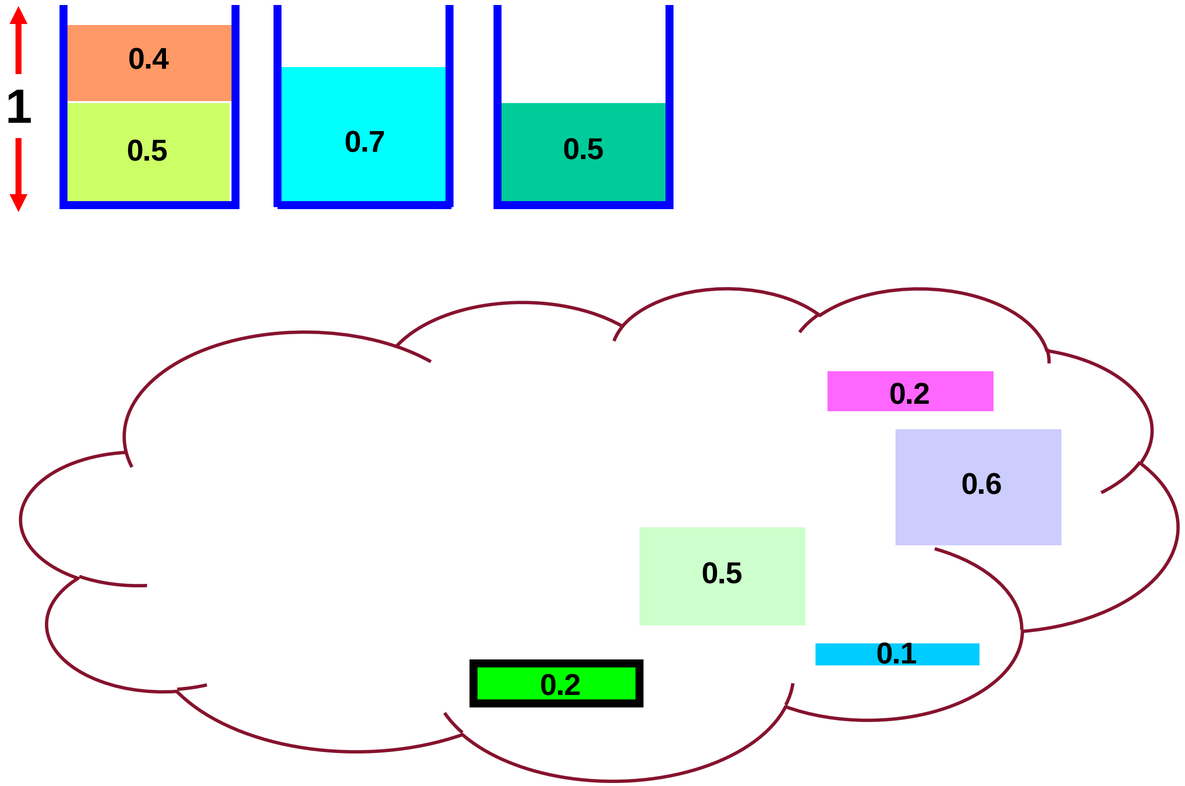
APROXIMAÇÃO

FF é 2-aproximativa

- Uma nova caixa só é criada se:
  - chega um item com $a_i \leq {}^1/_2$
    - e todas as caixas estão mais do que ${}^1/_2$ cheias, ou
  - chega um item com $a_i > {}^1/_2$

HEURÍSTICA DE FIRST-FIT (FF)

Para o problema de empacotamento



APROXIMAÇÃO

FF é 2-aproximativa

- Dada uma solução com B caixas,
  - pelo menos B - 1 delas estão mais da metade cheias
  - $(B-1)\,{}^1/_2 \leq \sum_{j=1}^{B-1} A(j) < \sum_{i \in I} a_i$
  - $(B-1)\,{}^1/_2 < \sum_{i \in I} a_i$

APROXIMAÇÃO

FF é 2-aproximativa

- $\sum_{i \in I} a_i$ é um limite inferior para o número ótimo de caixas é
  - $\sum_{i \in I} a_i \leq B^*$

- Assim, temos que:
  - $(B-1)\,{}^{1}\!/_{2} < \sum_{i \in I} a_i \leq B^*$
  - $(B-1)\,{}^{1}\!/_{2} < B^*$
  - $B - 1 < 2B^*$
  - $B \leq 2B^*$

Inserir primeiro os elementos mais difíceis

HEURÍSTICA DE FIRST-FIT DECREASING (FFD)

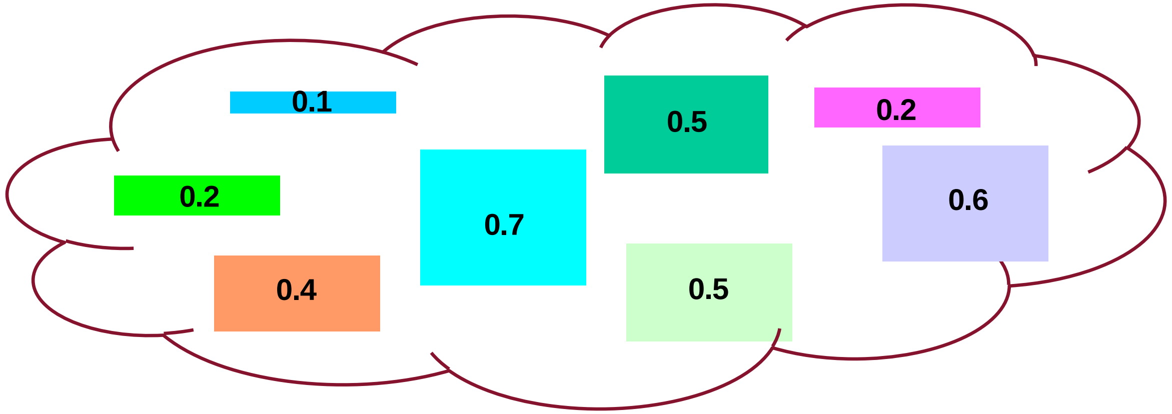
Para o problema de empacotamento

- Igual a heurísticas FF, mas ...
  - Insere os itens numa ordem específica
  - Do maior para o menor

HEURÍSTICA DE FIRST-FIT DECREASING (FFD)

Para o problema de empacotamento

FF



FFD

HEURÍSTICA DE FIRST-FIT DECREASING (FFD)

Para o problema de empacotamento

FFD

PSEUDOCÓDIGO

Da heurística FFD

- Executa FF com os elementos ordenados do maior para o menor valor de a_i

CORRETUDE

De FFD

- Só muda a ordem em que os elementos são inseridos
- Consequentemente, a prova de corretude de FF é válida para FFD

COMPLEXIDADE

De FFD

- $T^{FF}(n) = O(n\log n)$
- $T^{FFD}(n) = O(n\log n) + T^{FF}(n) = O(n\log n)$

APROXIMAÇÃO

- FFD é 2-aproximativa
- Pois a prova de FF continua válida

MELHOR APROXIMAÇÃO PARA FFD

$$B \leq \frac{11}{9} B^* + \frac{6}{9}$$

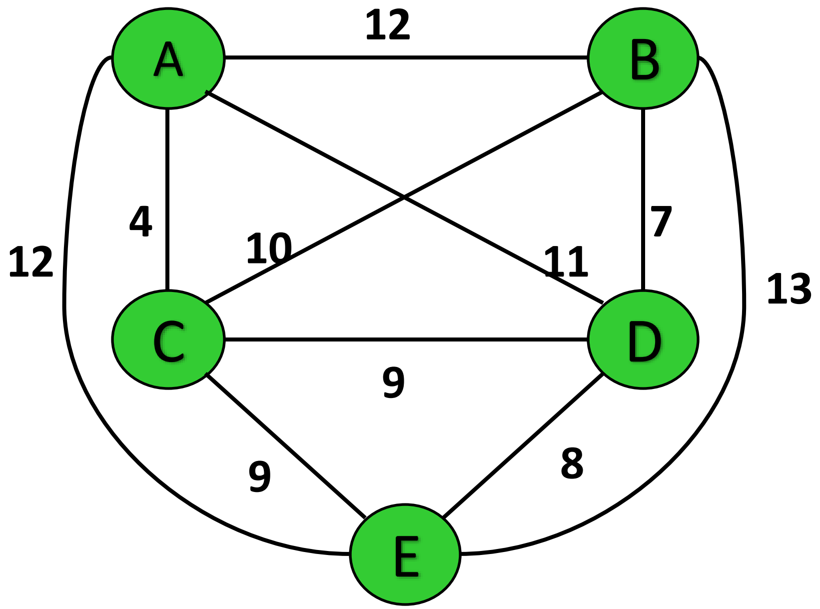
PRINCÍPIO DE PROJETO

de heurísticas construtivas

Inserir primeiro os elementos que resultam no menor incremento* da função objetivo

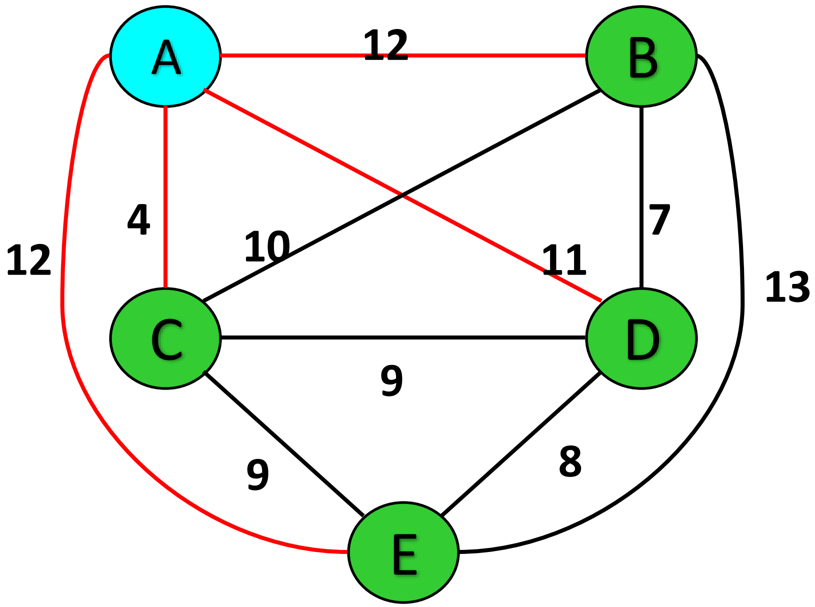
HEURÍSTICA DO VIZINHO MAIS PRÓXIMO

Para para o problema do caixeiro viajante



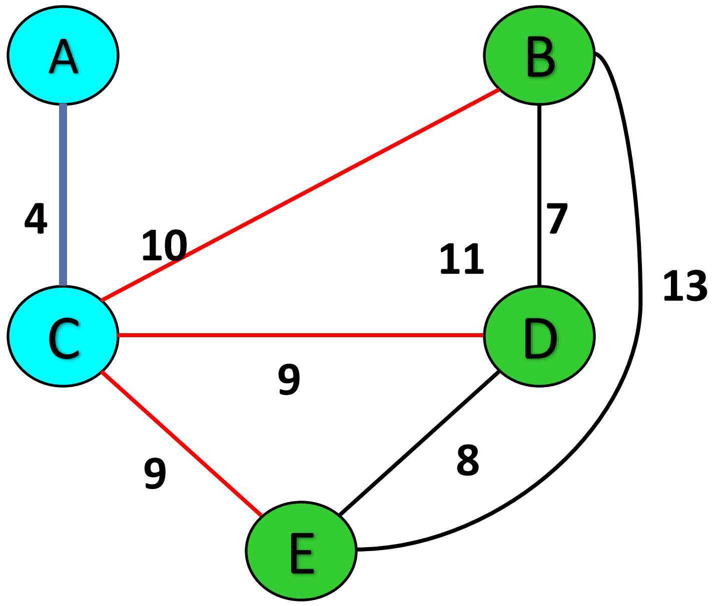
HEURÍSTICA DO VIZINHO MAIS PRÓXIMO

Para para o problema do caixeiro viajante



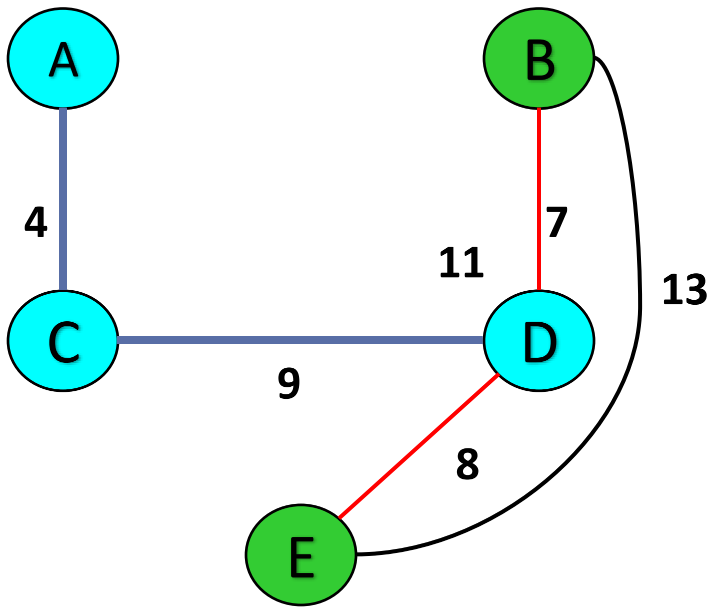
HEURÍSTICA DO VIZINHO MAIS PRÓXIMO

Para para o problema do caixeiro viajante



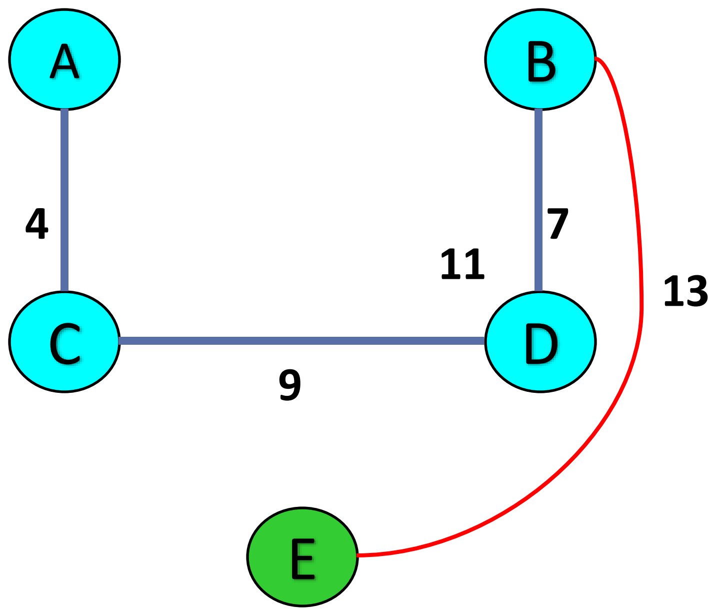
HEURÍSTICA DO VIZINHO MAIS PRÓXIMO

Para para o problema do caixeiro viajante



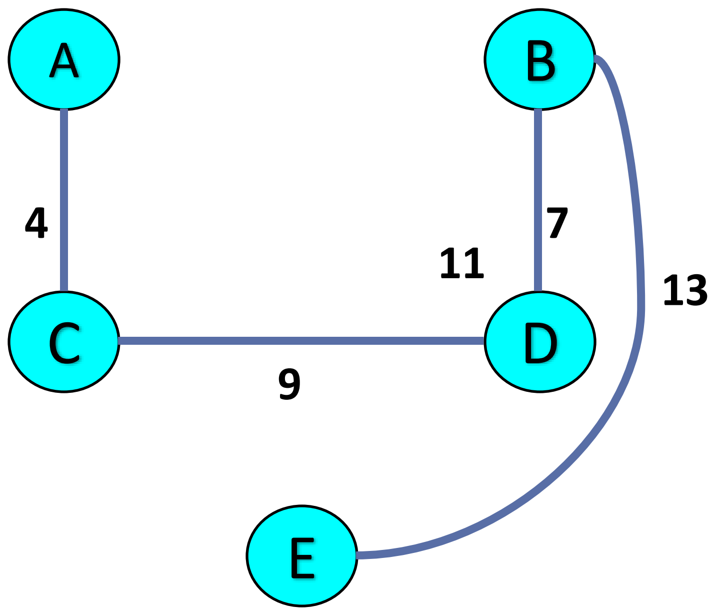
HEURÍSTICA DO VIZINHO MAIS PRÓXIMO

Para para o problema do caixeiro viajante



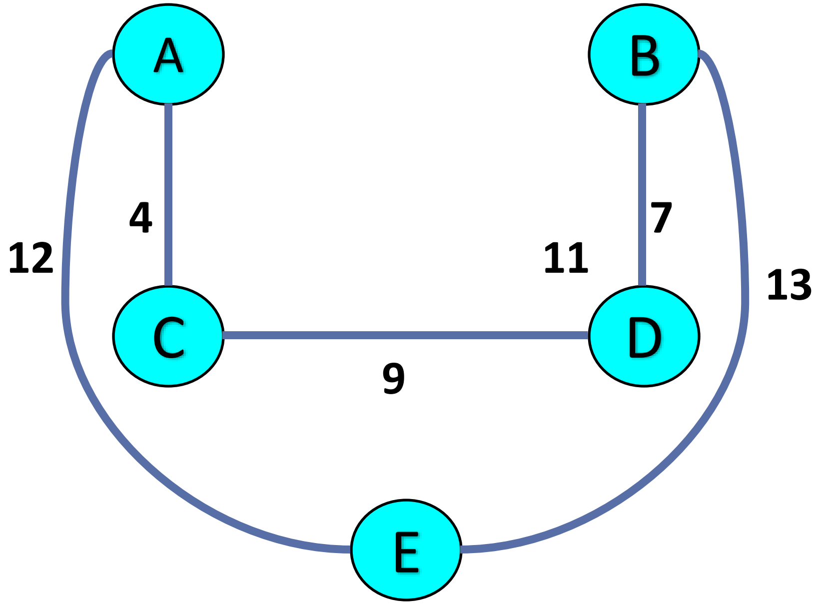
HEURÍSTICA DO VIZINHO MAIS PRÓXIMO

Para para o problema do caixeiro viajante



HEURÍSTICA DO VIZINHO MAIS PRÓXIMO

Para para o problema do caixeiro viajante



PSEUDOCÓDIGO

Da heurística do Vizinho Mais Próximo

- Parte de uma solução parcial $S = [0]$
- Enquanto $V \backslash S \neq \emptyset$
  - Seleciona o vizinho mais próximo $\mathrm{k} \in V \backslash S$
  - Insere este nó em S
- Retorne S

CORRETUDE

Para grafos completos

- $S = [0]$ é parcialmente viável
  - Ciclo de tamanho m.
- $(k, 0) \in E, \forall k \in V,$
  - Portanto, $S = [S$:k] é parcialmente viável
- Para quando $V \backslash \mathrm{S} = \emptyset$
  - Portanto $\mathrm{S} = \mathrm{V}$, e S é viável

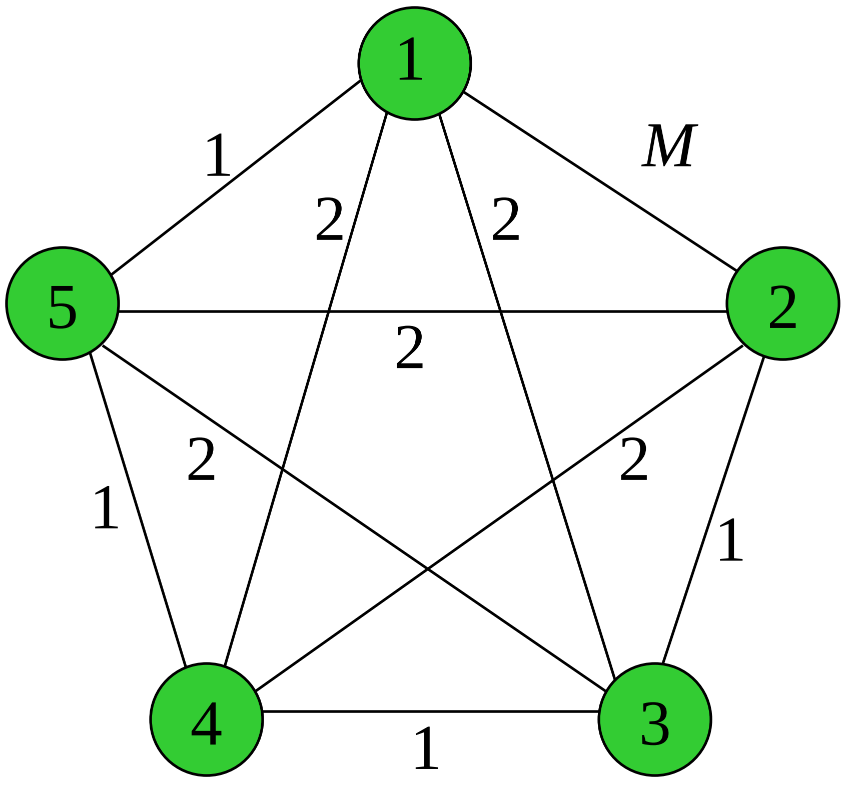
COMPLEXIDADE

Da heurística do Vizinho Mais Próximo

- $T(n) = O(n) \cdot O(n) = O(n^2)$

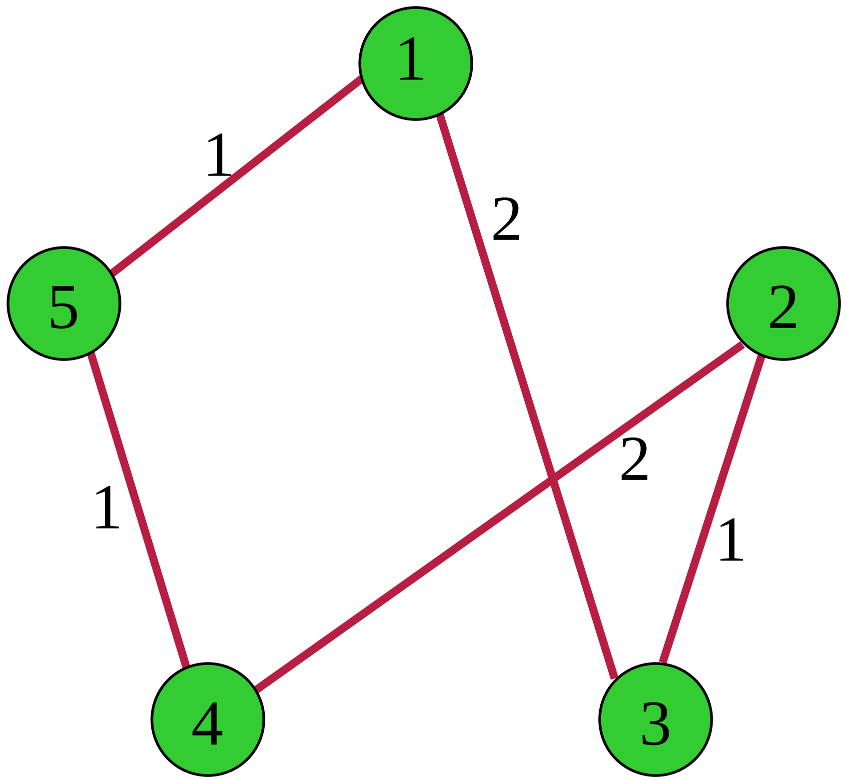
APROXIMAÇÃO

HVMP encontra soluções arbitrariamente ruins



APROXIMAÇÃO

HVMP encontra soluções arbitrariamente ruins



HEURÍSTICAS DE PROGRAMAÇÃO DINÂMICA

O que fazer quando não conhecemos um algoritmo guloso para o problema de localização associado?

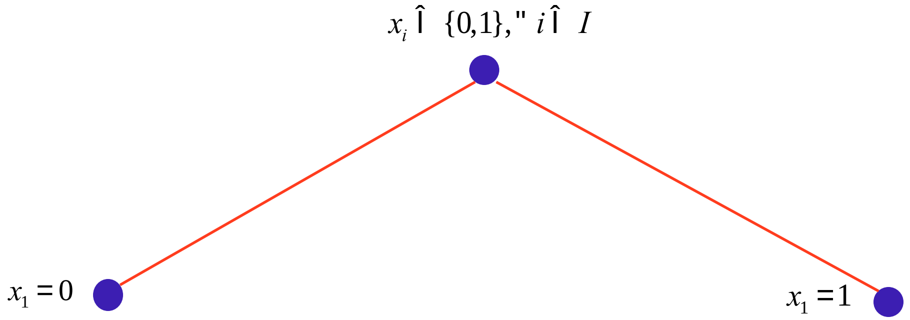
Utilizamos outro

paradigma de programação

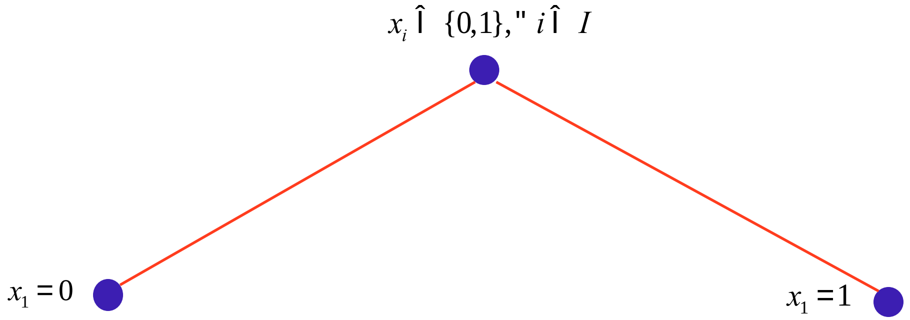
O problema de localização associado pode ser NP-Difícil

PROGRAMAÇÃO DINÂMICA

- Paradigma baseado em dividir e conquistar

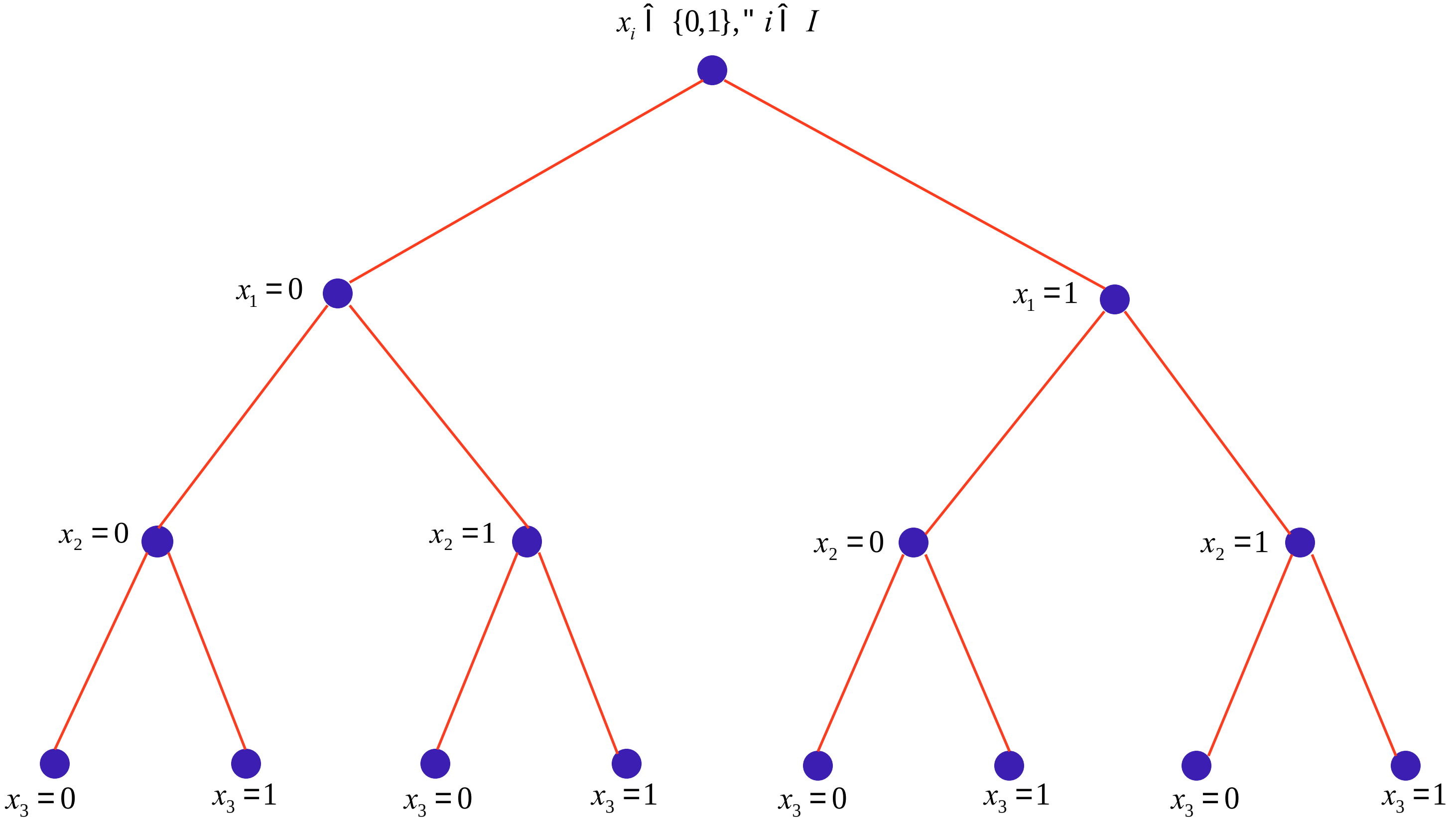


- Retorna a solução ótima, caso ela possa ser construída a partir da solução ótima dos subproblemas



PROGRAMAÇÃO DINÂMICA

O número de subproblema cresce exponencialmente com a altura da árvore

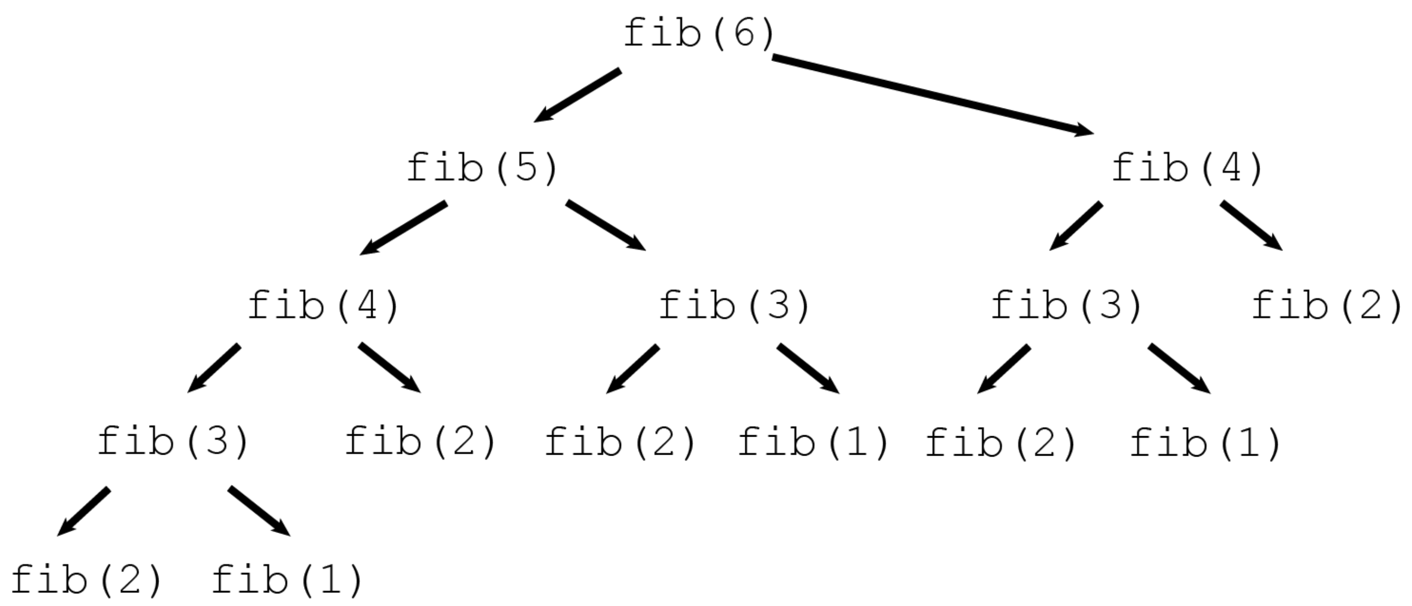


PROGRAMAÇÃO DINÂMICA

- Eficiente quando existem subproblemas repetidos
- Armazenamos as soluções dos subproblemas já resolvidos
  - Memorização
  - Tabulação

PROGRAMAÇÃO DINÂMICA: MEMORIZAÇÃO

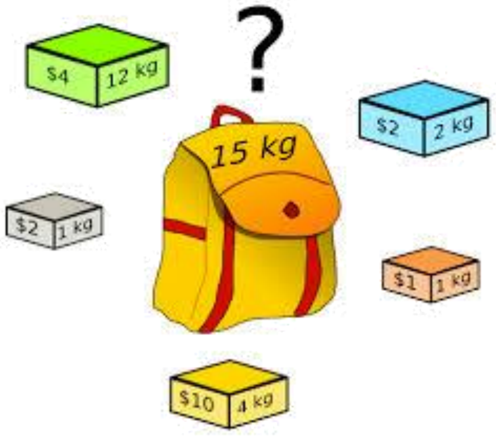
Exemplo: Computar `fib(n)`, onde a Sequencia de Fibonacci é 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233

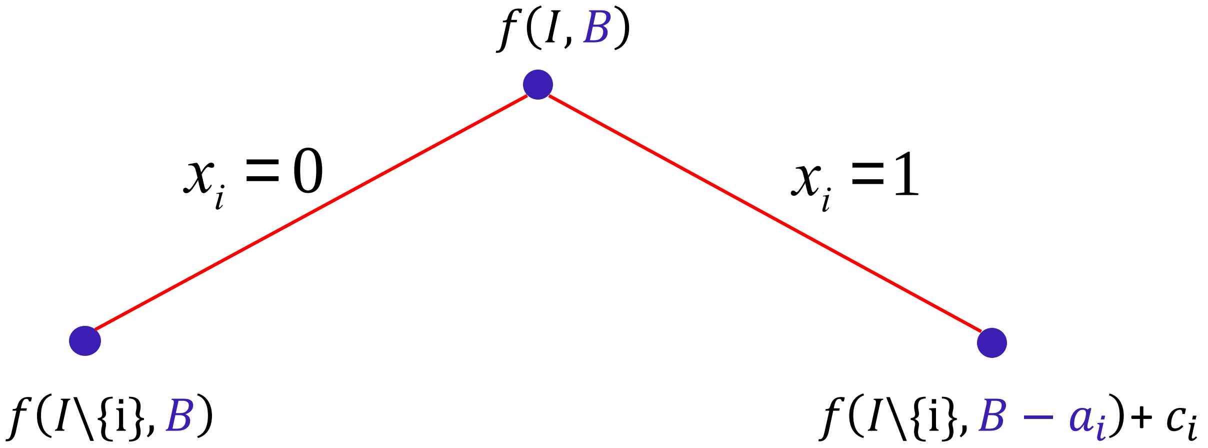


Fonte: https://willrosenbaum.com/teaching/2021s-cosc-112/notes/recursive-fibonacci/

PROGRAMAÇÃO DINÂMICA: TABULAÇÃO

Exemplo: Problema da Mochila





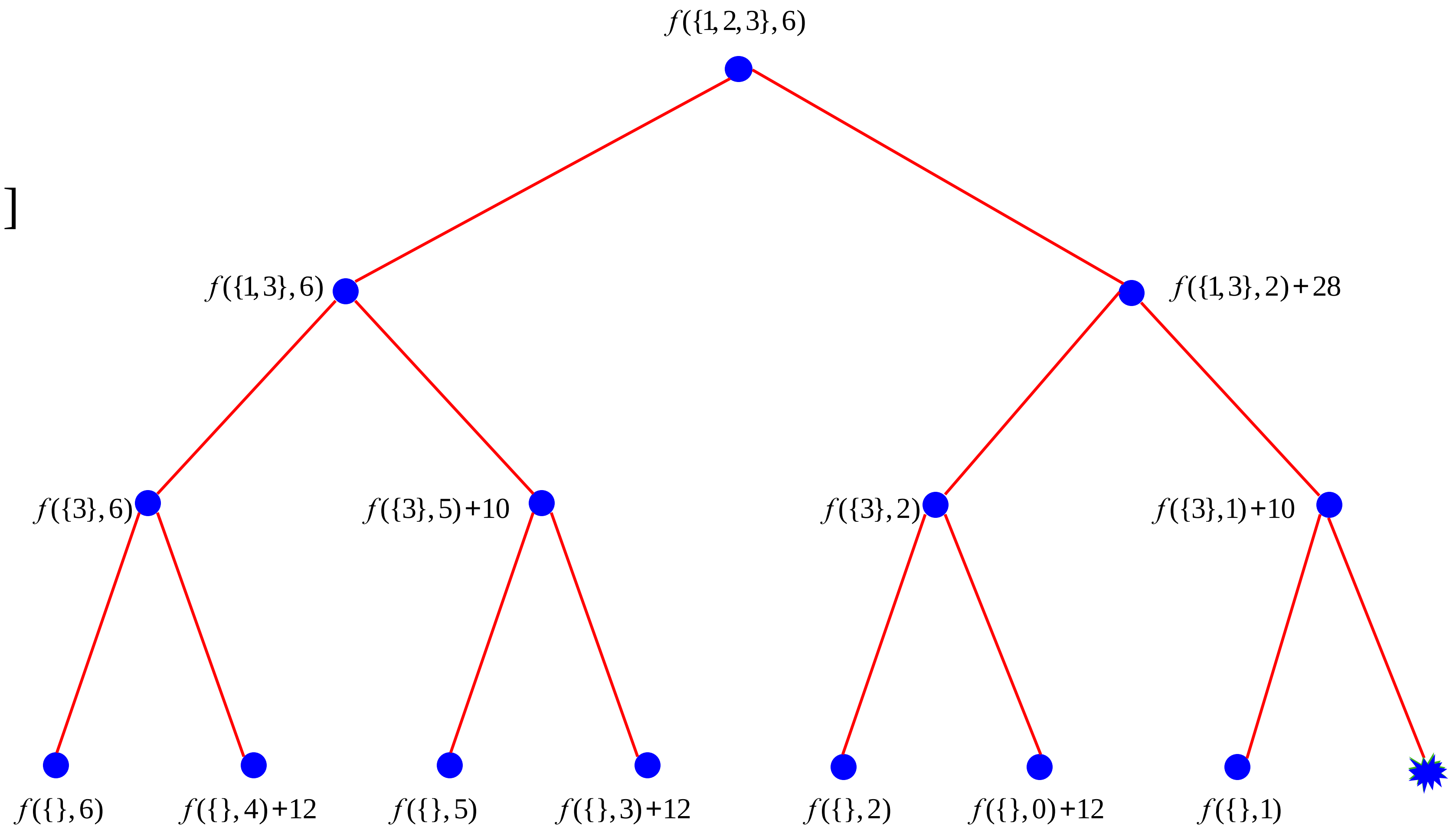
PROGRAMAÇÃO DINÂMICA

$I = \{1, 2, 3\}$

$a_i = [1, 4, 2]$

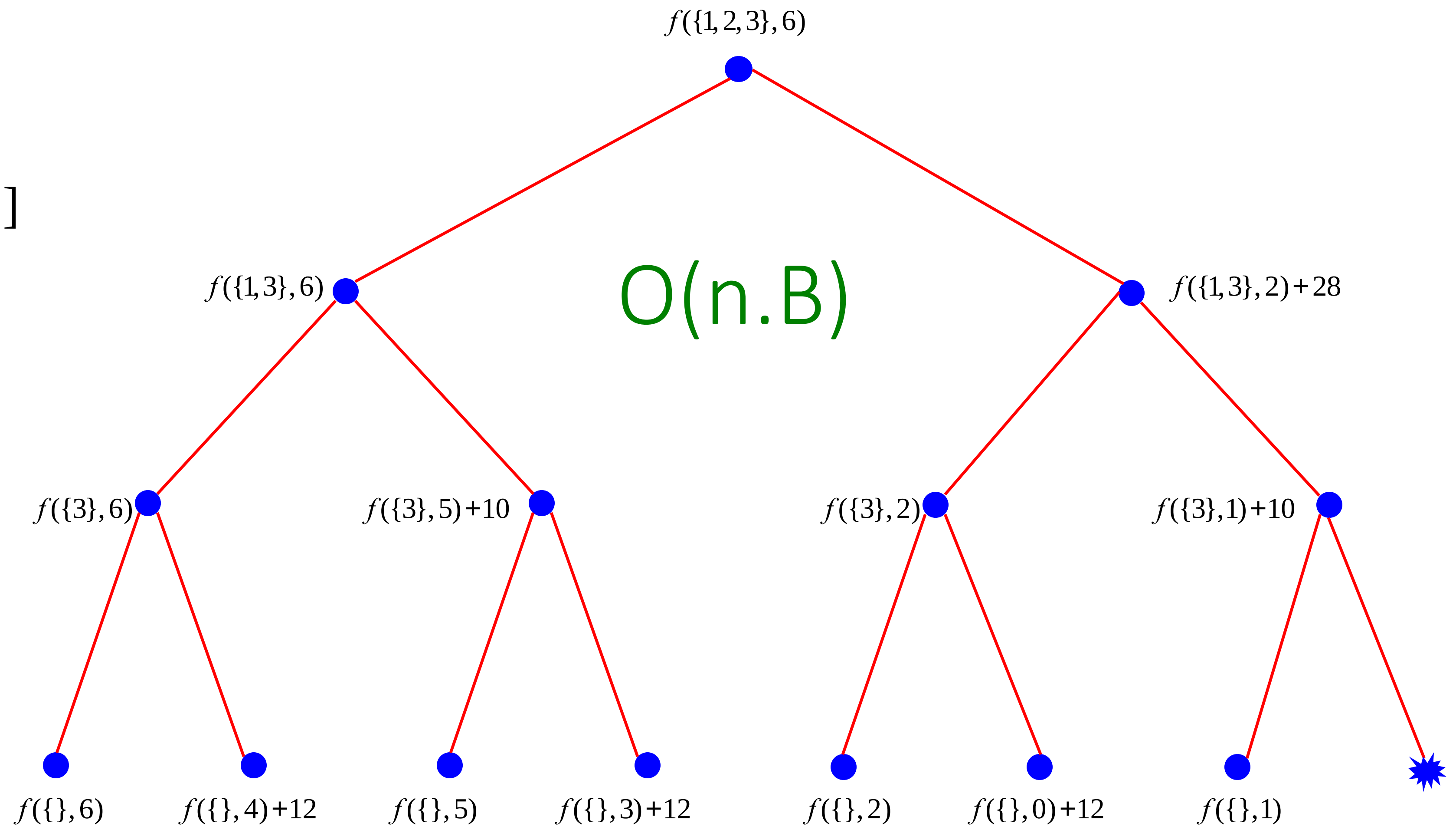
$c_i = [10, 28, 12]$

$B = 6$



PROGRAMAÇÃO DINÂMICA

$$I = \{1, 2, 3\}$$
$$a_i = [1, 4, 2]$$
$$c_i = [10, 28, 12]$$
$$B = 6$$



PROBLEMA DE OTIMIZAÇÃO DE PORTIFÓLIO (POP)

$I = \{1,2, \ldots, |I|\}$

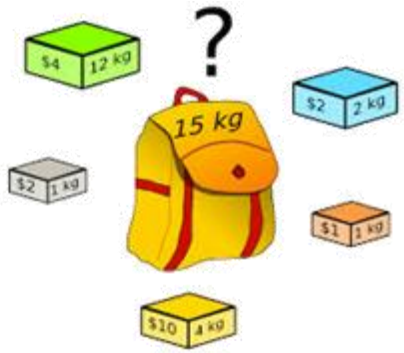
$\delta_{ij} \in \mathbb{N}$

$a_i \in \mathbb{N}$

$c_i \in \mathbb{N}$

$B \in \mathbb{N}$

$R \in \mathbb{N}$



$\min \sum_{i \in I} \delta_{ij} x_i \, x_j$ s.t.

$$\sum_{i \in I} c_i x_i \geq R$$

$$\sum_{i \in I} a_i x_i \leq B$$

$0 \leq x_i \leq 1$

$x_i \in \mathbb{Z}$

HEURÍSTICA DE PD PARA POP

- Executa a programação dinâmica da mochila
- Para cada solução $\bar{x}$ da mochila tal que $\sum_{i \in I} c_i \bar{x}_i \geq R$,
  - Computa $\sum_{i \in I} \delta_{ij} \bar{x}_i \bar{x}_j$ e armazena a melhor (x^*)
- Retorna x^*

CORRETUDE

- Retorna a solução ótima x^* da mochila , ou seja
  - $\sum_{i \in I} c_i \mathrm{x}_i^*$ é máximo, e
  - $\sum_{i \in I} a_i \mathrm{x}_i^* \leq B$
- Se existe $\bar{x}$ tal que $\sum_{i \in I} c_i \bar{x}_i \geq R$, e $\sum_{i \in I} a_i \bar{x} \leq B$, então
  - $\sum_{i \in I} c_i \mathrm{x}_i^* \geq R$
  - $\sum_{i \in I} a_i \mathrm{x}_i^* \leq B$

COMPLEXIDADE

- $T(n) = O(Bn) \cdot O(n^2)$
- $T(n) = O(Bn^3)$

$\min \sum_{i \in I} \delta_{ij} x_i \, x_j$ s.t.

$$\sum_{i \in I} c_i x_i \geq R$$

$$\sum_{i \in I} a_i x_i \leq B$$

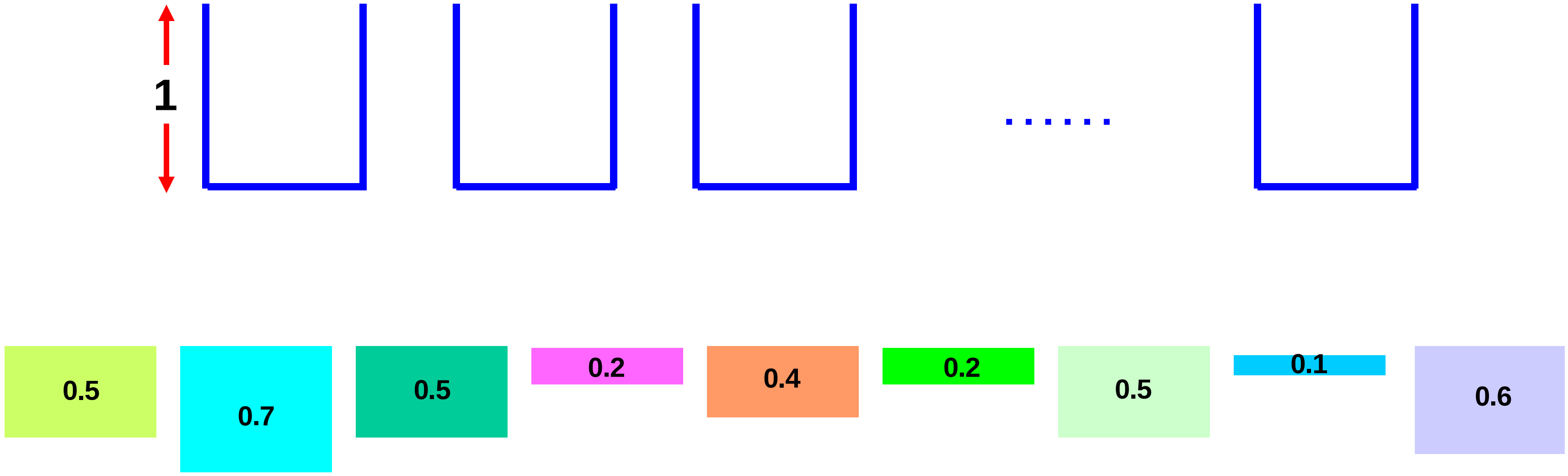
$0 \leq x_i \leq 1$

$x_i \in \mathbb{Z}$

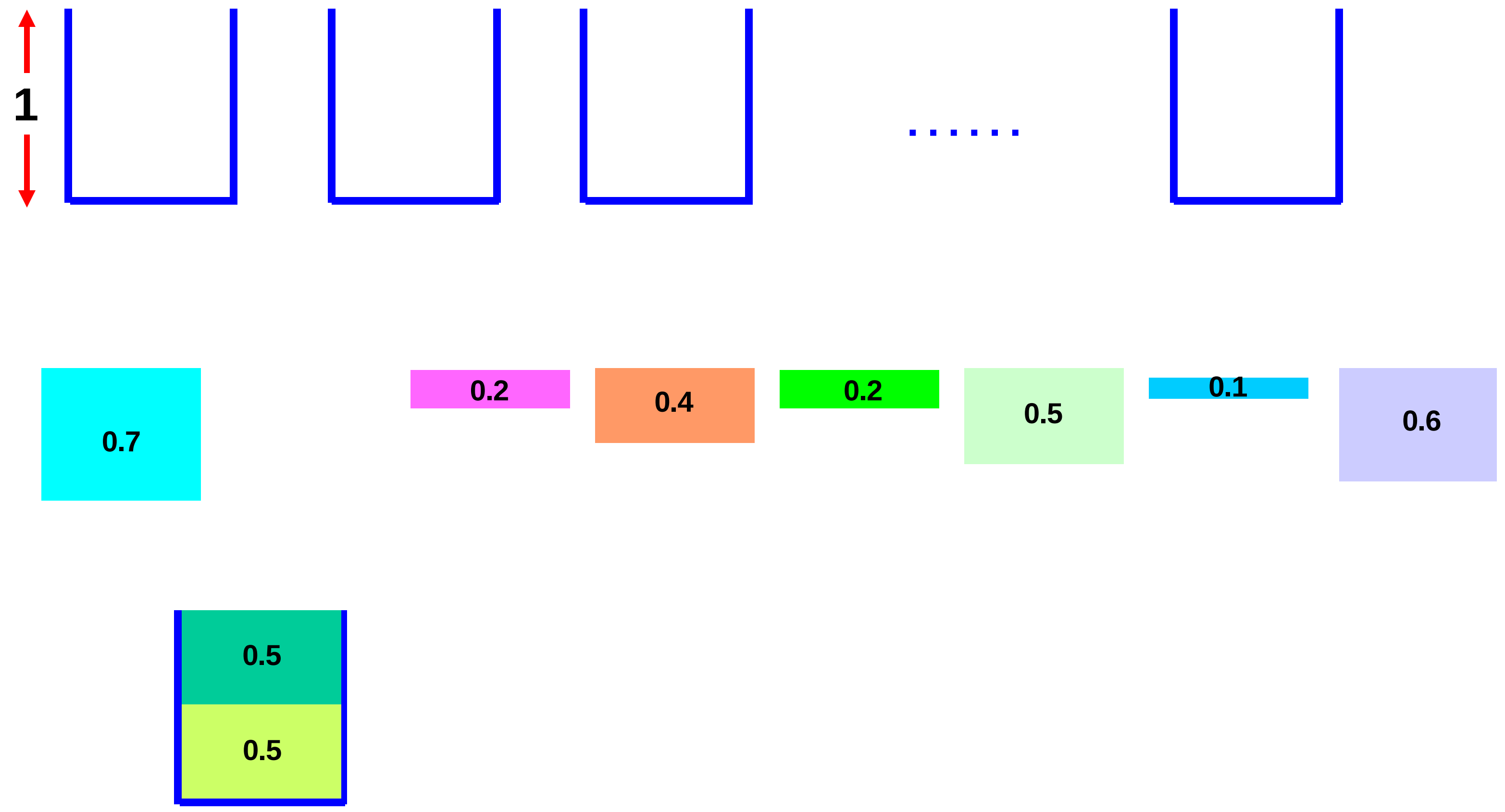
APROXIMAÇÃO

Resolve iterativamente subproblemas utilizando Programação Dinâmica

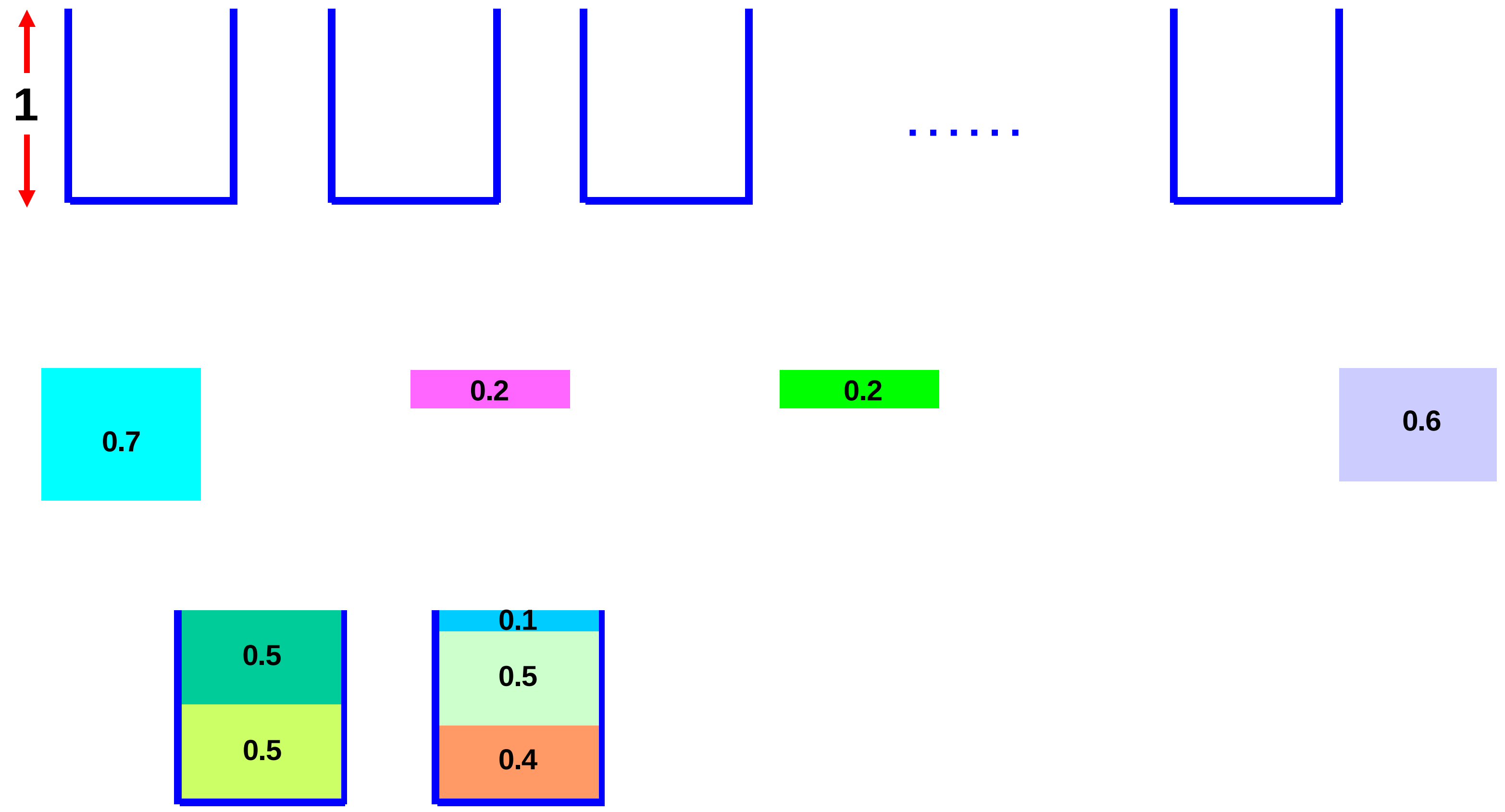
HPD PARA BIN PACKING



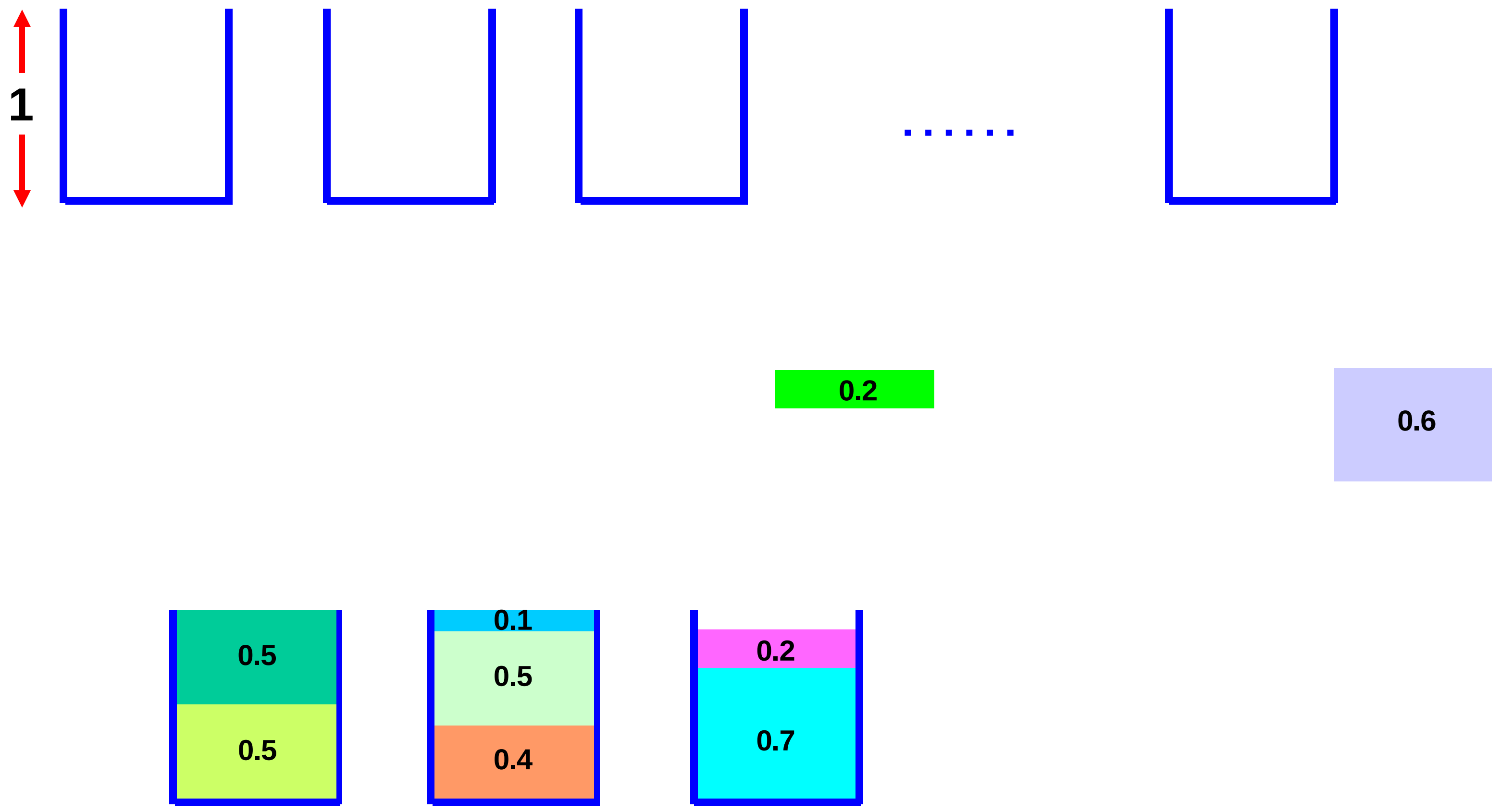
HPD PARA BIN PACKING



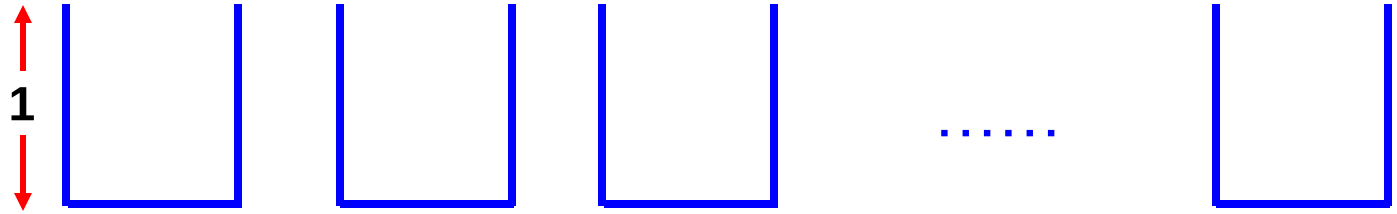
HPD PARA BIN PACKING

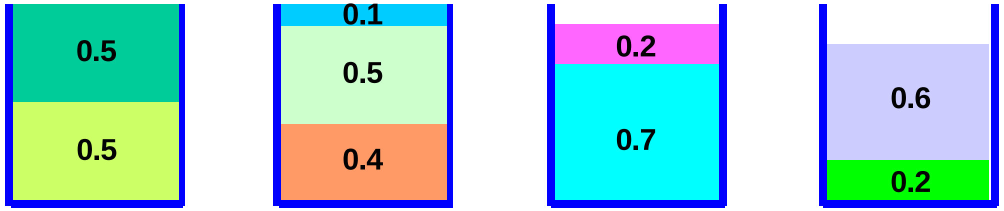


HPD PARA BIN PACKING



HPD PARA BIN PACKING





HEURÍSTICA PD PARA BIN PACKING

- Parte de uma solução parcial *S* vazia
- A cada iteração constrói um bin maximal utilizando PD e insere em *S*
- Repete até que I\S seja vazio

CORRETUDE

- $S = []$ é parcialmente viável
  - Nenhuma caixa violada
- S é parcialmente viável por definição
  - A nova caixa inserida não ultrapassa a capacidade da caixa
- Para quando $I \backslash \mathrm{S} = \emptyset$
  - Portanto $\mathrm{S} = \mathrm{I}$, e S é viável

COMPLEXIDADE

- $T(n) = O(\Delta) \cdot O(Bn)$
- $T(n) = O(n) \cdot O(Bn)$
- $T(n) = O(Bn^2)$

Encontrar o bin com a menor folga

$$max \sum_{i \in I} a_i x_i$$

$$\sum_{i \in I} a_i x_i \leq B$$

$$0 \leq x_i \leq 1$$

$$x_i \in \mathbb{Z}$$

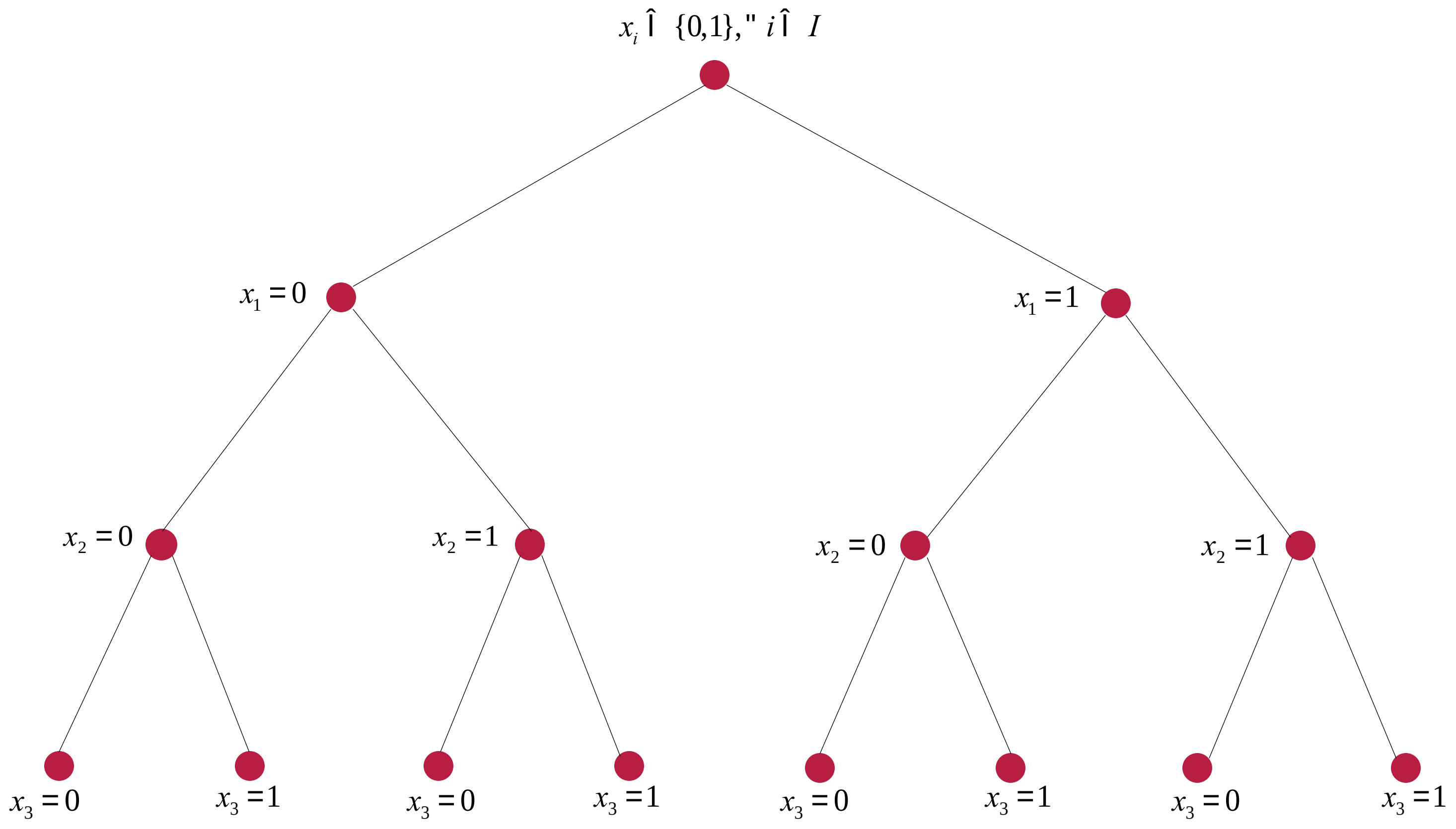
APROXIMAÇÃO

- Dada uma solução com Δ caixas, pelo menos Δ - 1 delas estão mais da metade cheias
- Consequentemente, é 2-aproximativa
  - Vide prova de FF

ALGORITMOS DE BRANCH-AND-BOUND

ÁRVORE DE DECISÃO

Exemplo: problemas com |*I*| variáveis binárias



ALGORITMOS DE BRANCH-AND-BOUND

- Algoritmo de enumeração implícita
  - Não necessariamente enumera todos os subproblemas
  - Complexidade depende do número de subproblemas
    - Exponencial no pior caso

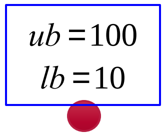
Paradigma mais utilizado para projetar algoritmos exatos para problemas de otimização NP-Difíceis

Assume que um limite inferior e superior para o valor da solução ótima pode ser computado de forma eficiente

Desempenho fortemente relacionado a qualidade dos limites

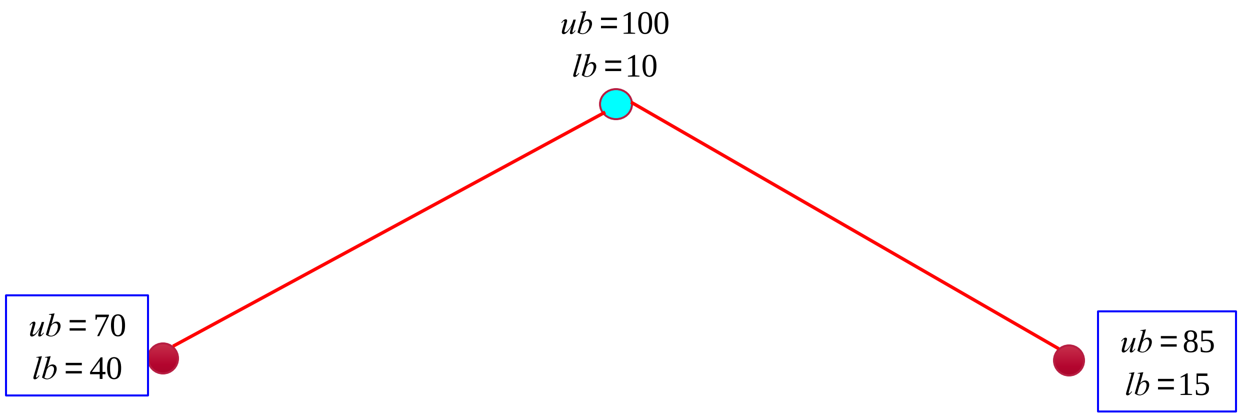
ALGORITMOS DE BRANCH-AND-BOUND

Problemas de maximização



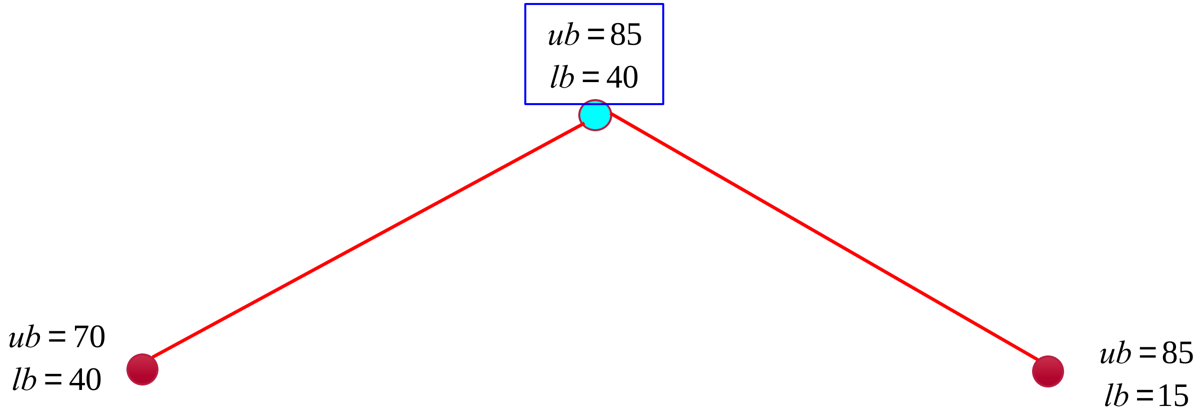
ALGORITMOS DE BRANCH-AND-BOUND

Problemas de maximização



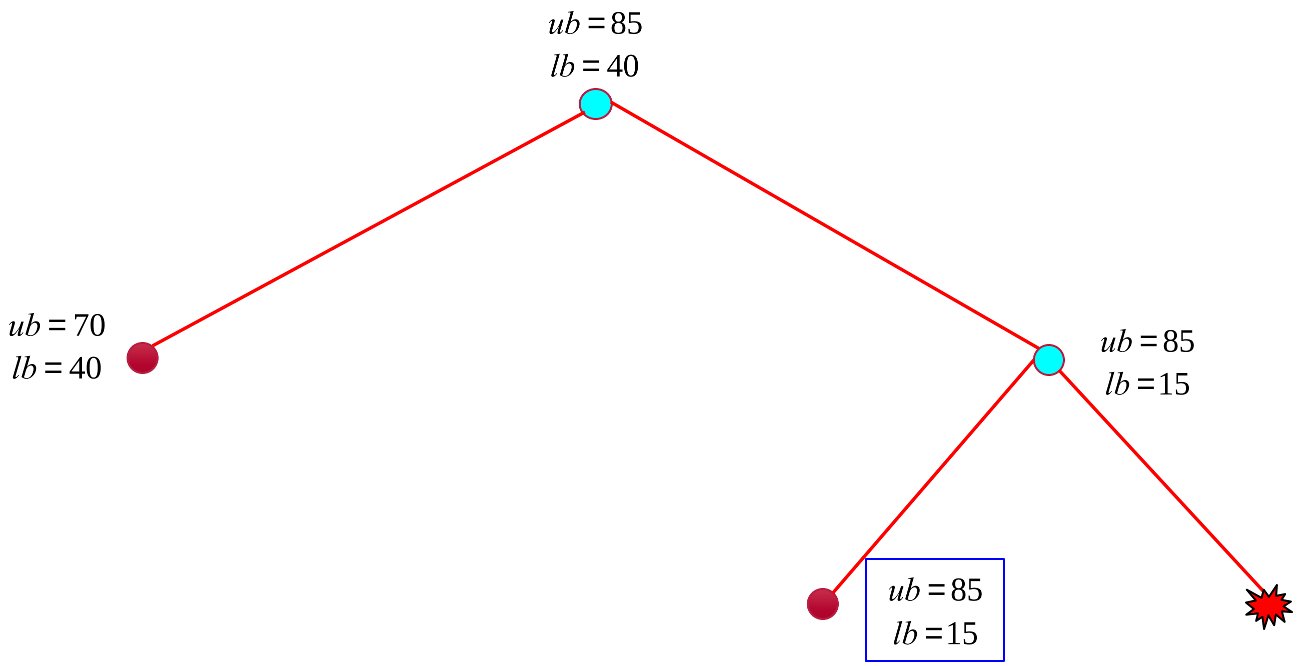
ALGORITMOS DE BRANCH-AND-BOUND

Problemas de maximização



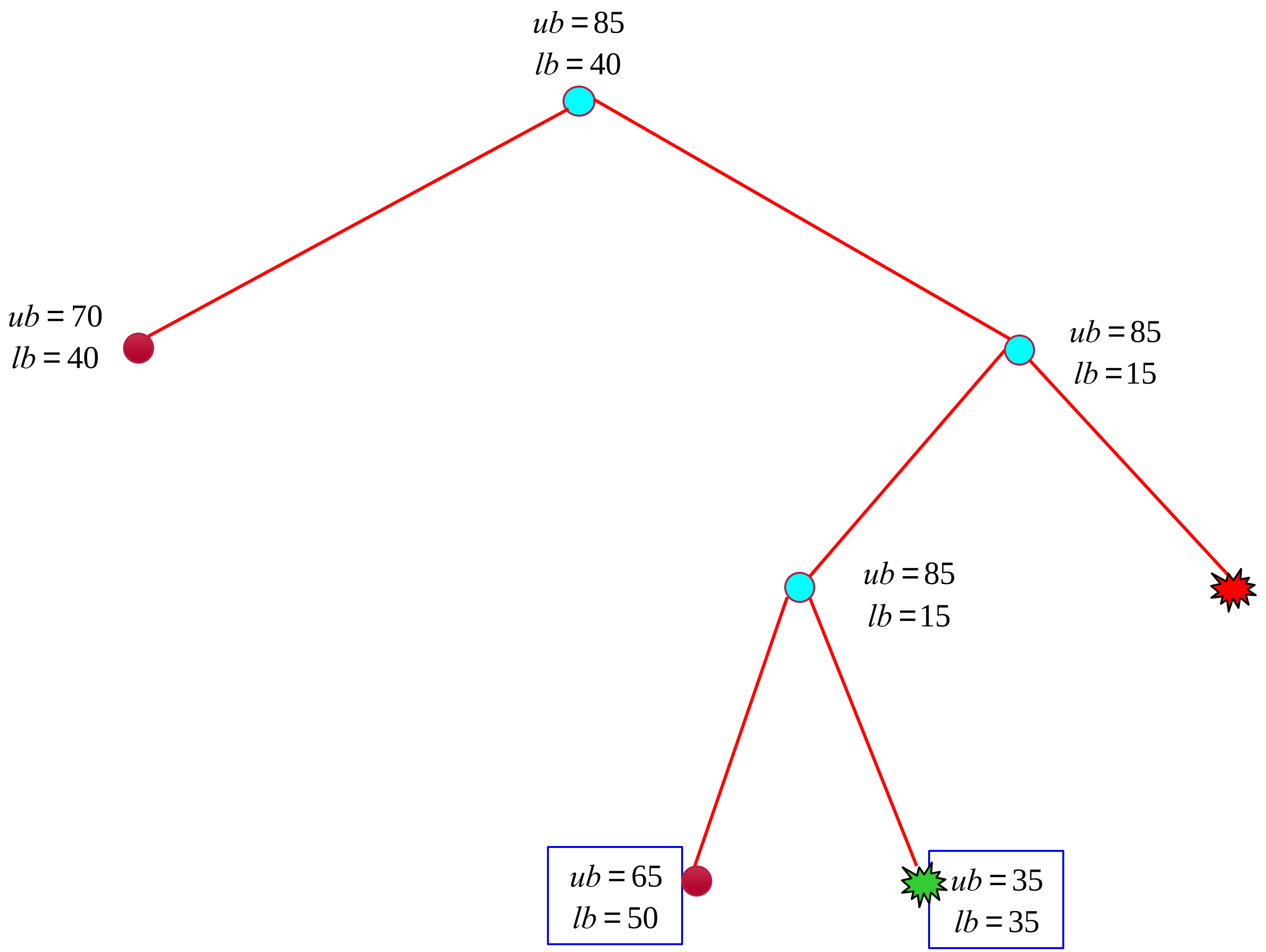
ALGORITMOS DE BRANCH-AND-BOUND

Problemas de maximização



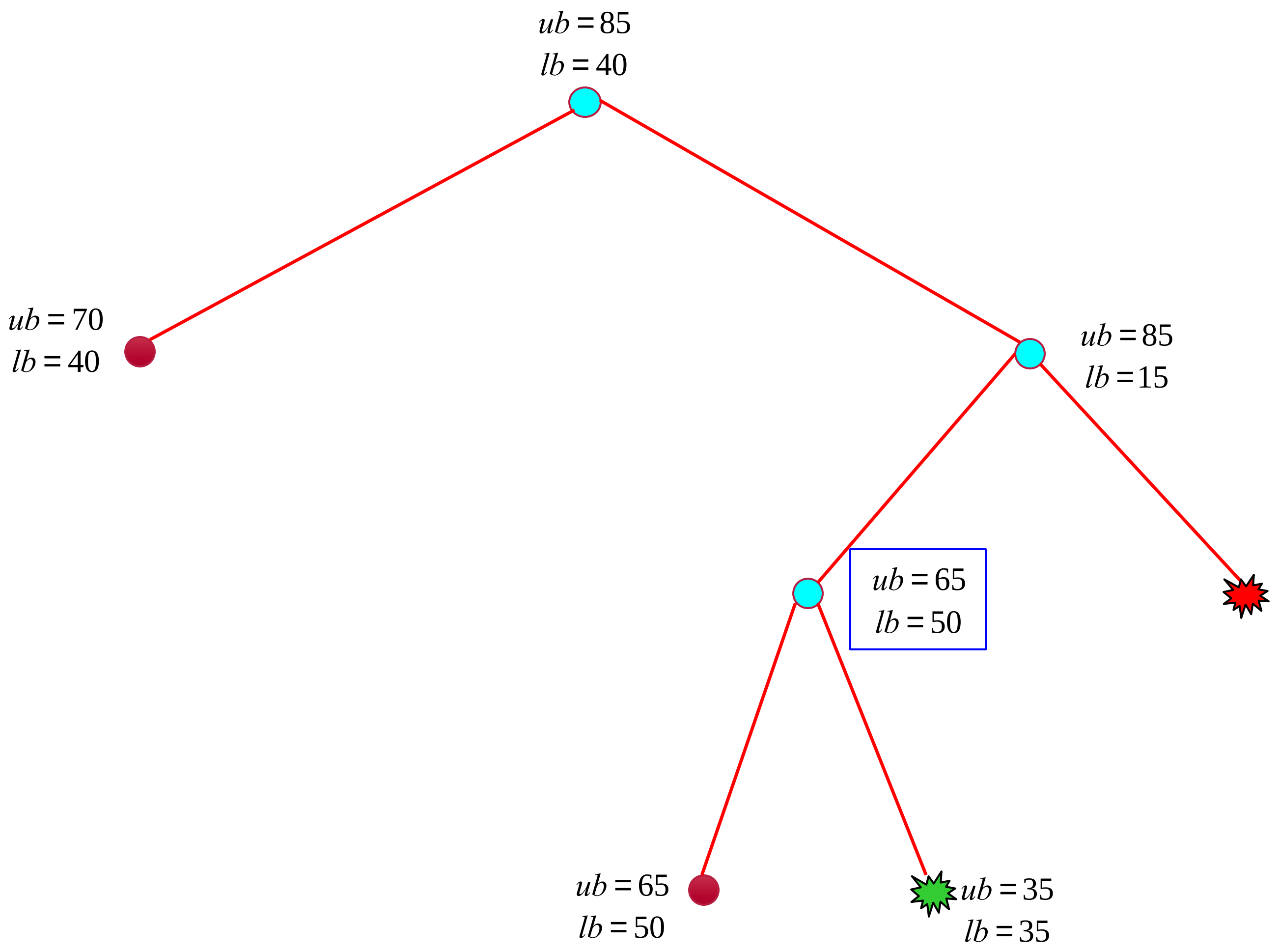
ALGORITMOS DE BRANCH-AND-BOUND

Problemas de maximização



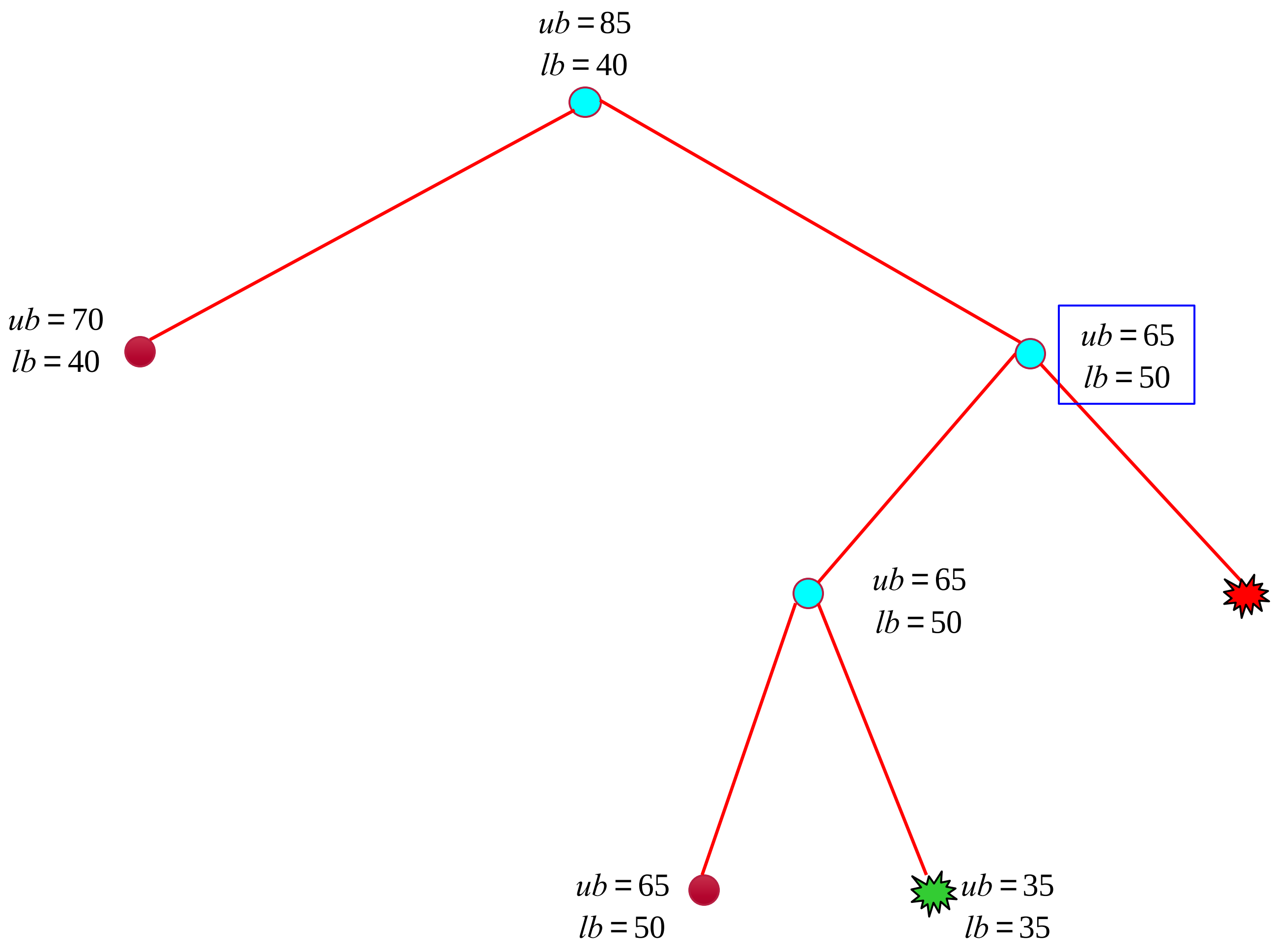
ALGORITMOS DE BRANCH-AND-BOUND

Problemas de maximização



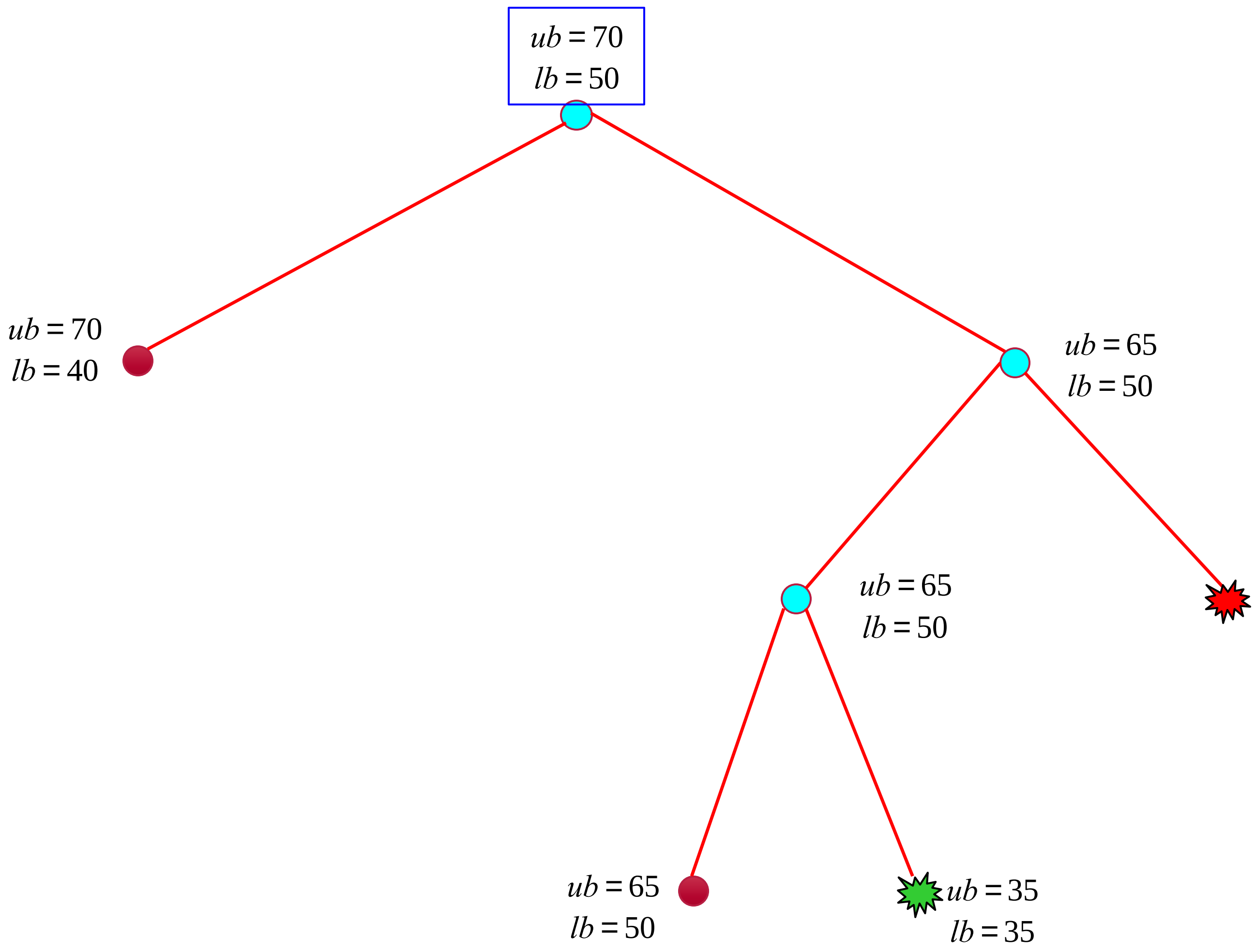
ALGORITMOS DE BRANCH-AND-BOUND

Problemas de maximização



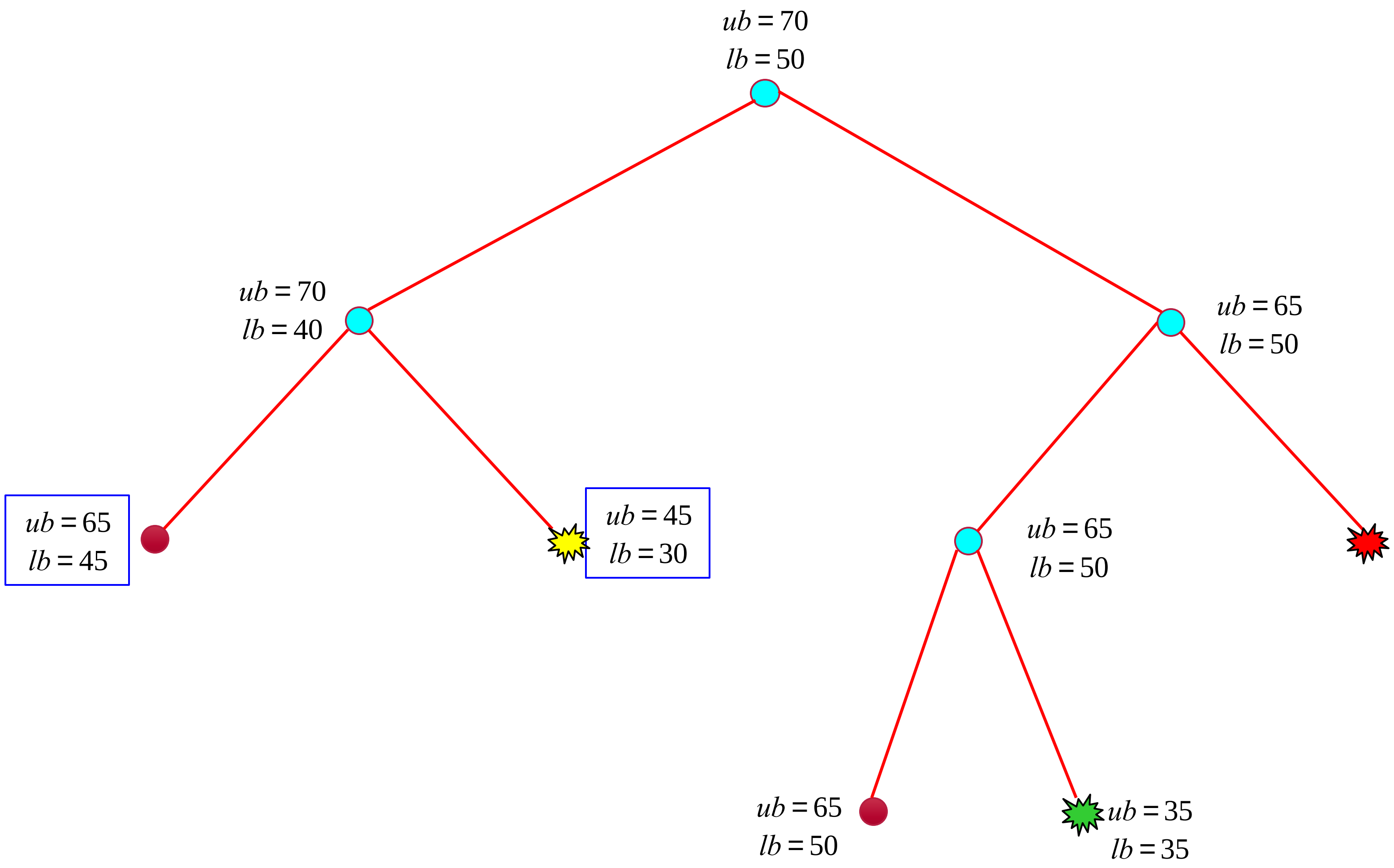
ALGORITMOS DE BRANCH-AND-BOUND

Problemas de maximização



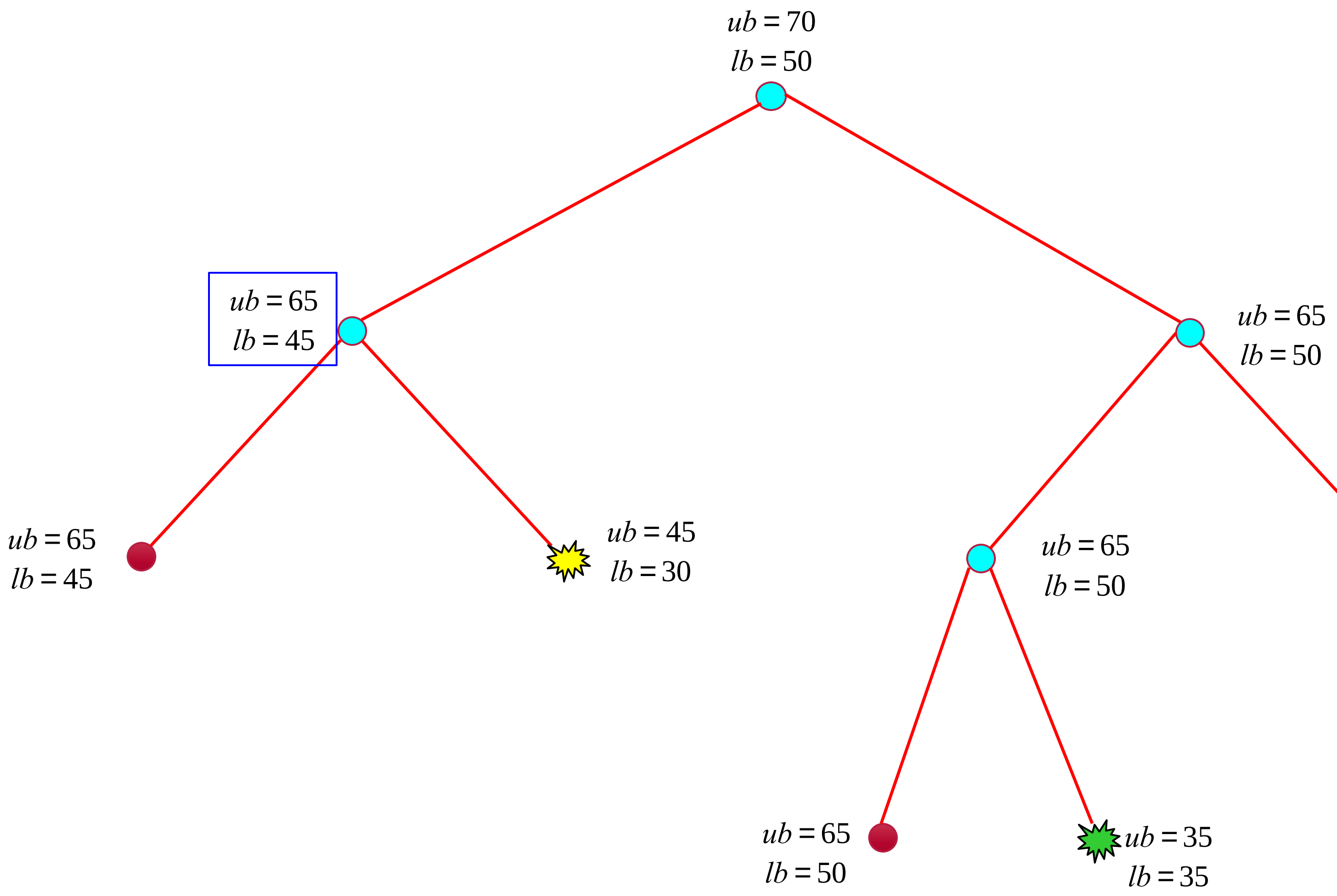
ALGORITMOS DE BRANCH-AND-BOUND

Problemas de maximização



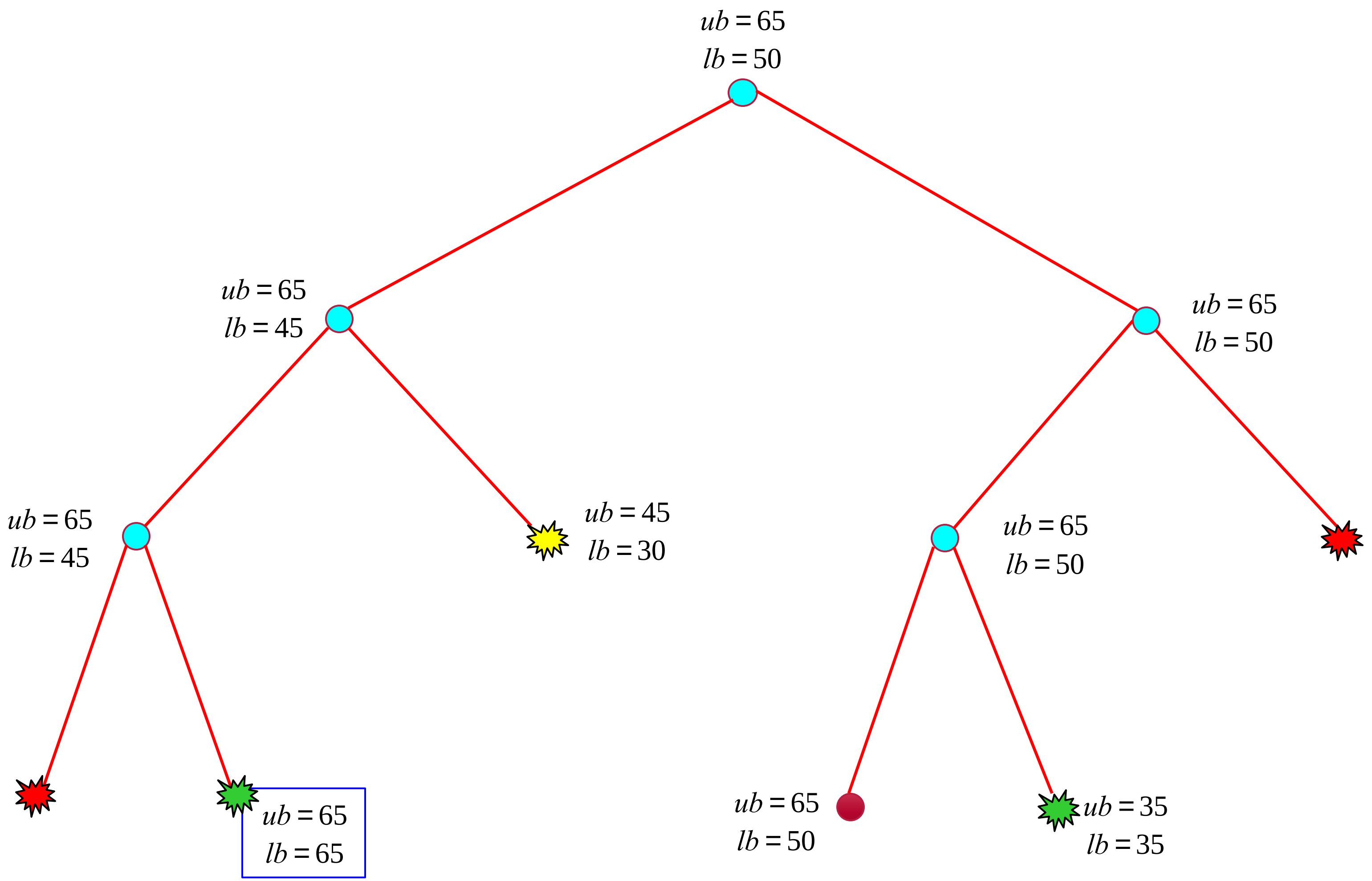
ALGORITMOS DE BRANCH-AND-BOUND

Problemas de maximização



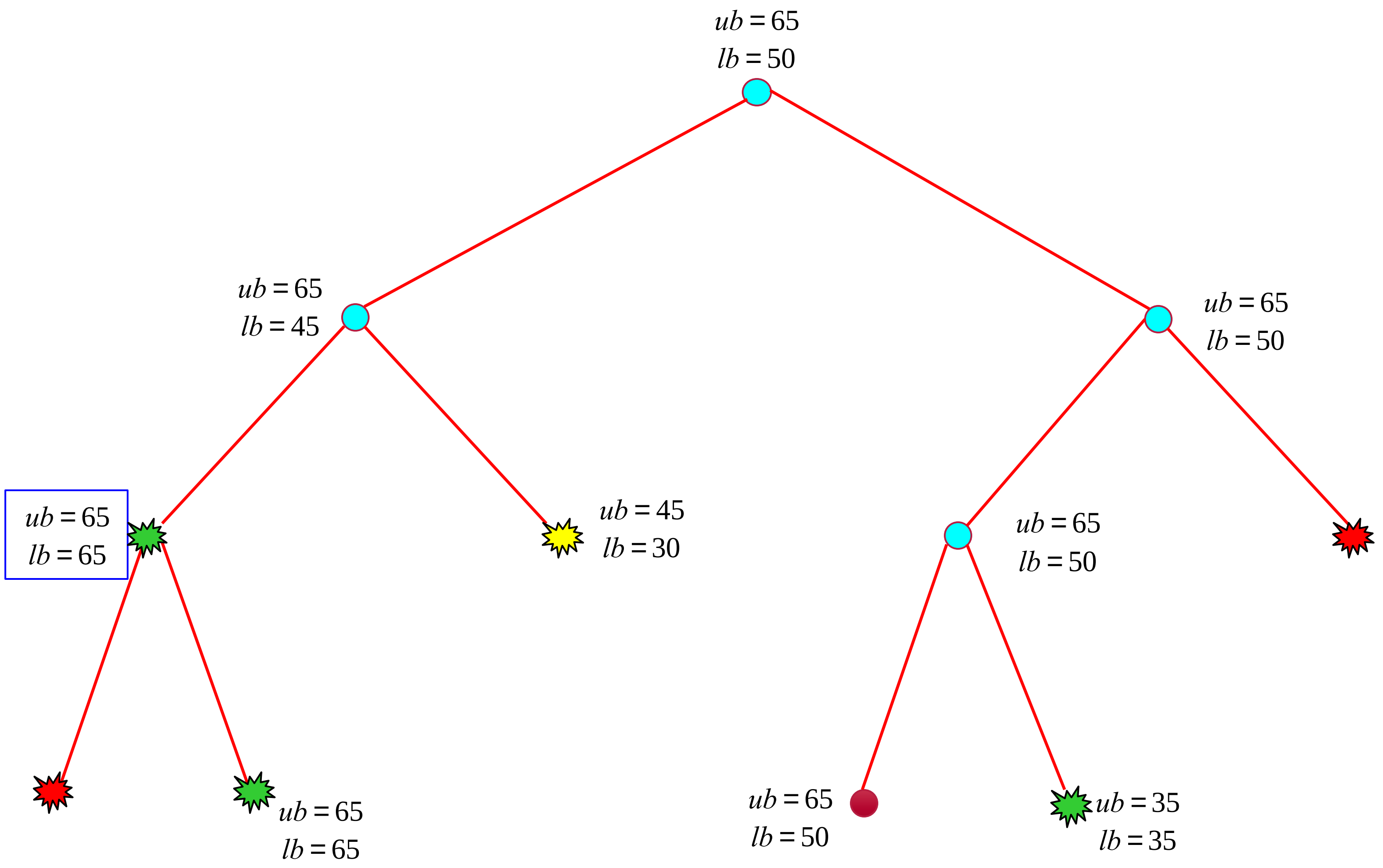
ALGORITMOS DE BRANCH-AND-BOUND

Problemas de maximização



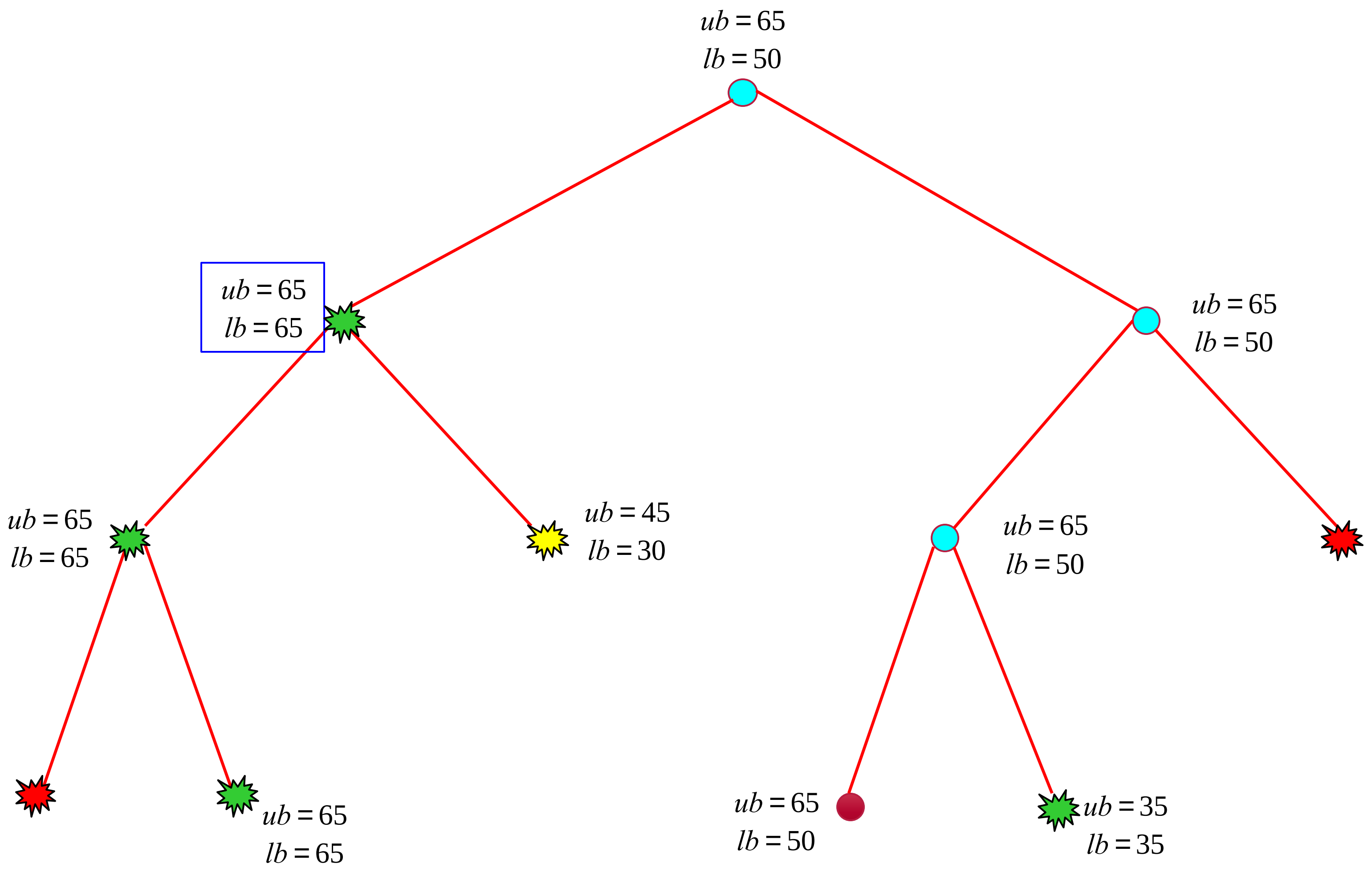
ALGORITMOS DE BRANCH-AND-BOUND

Problemas de maximização



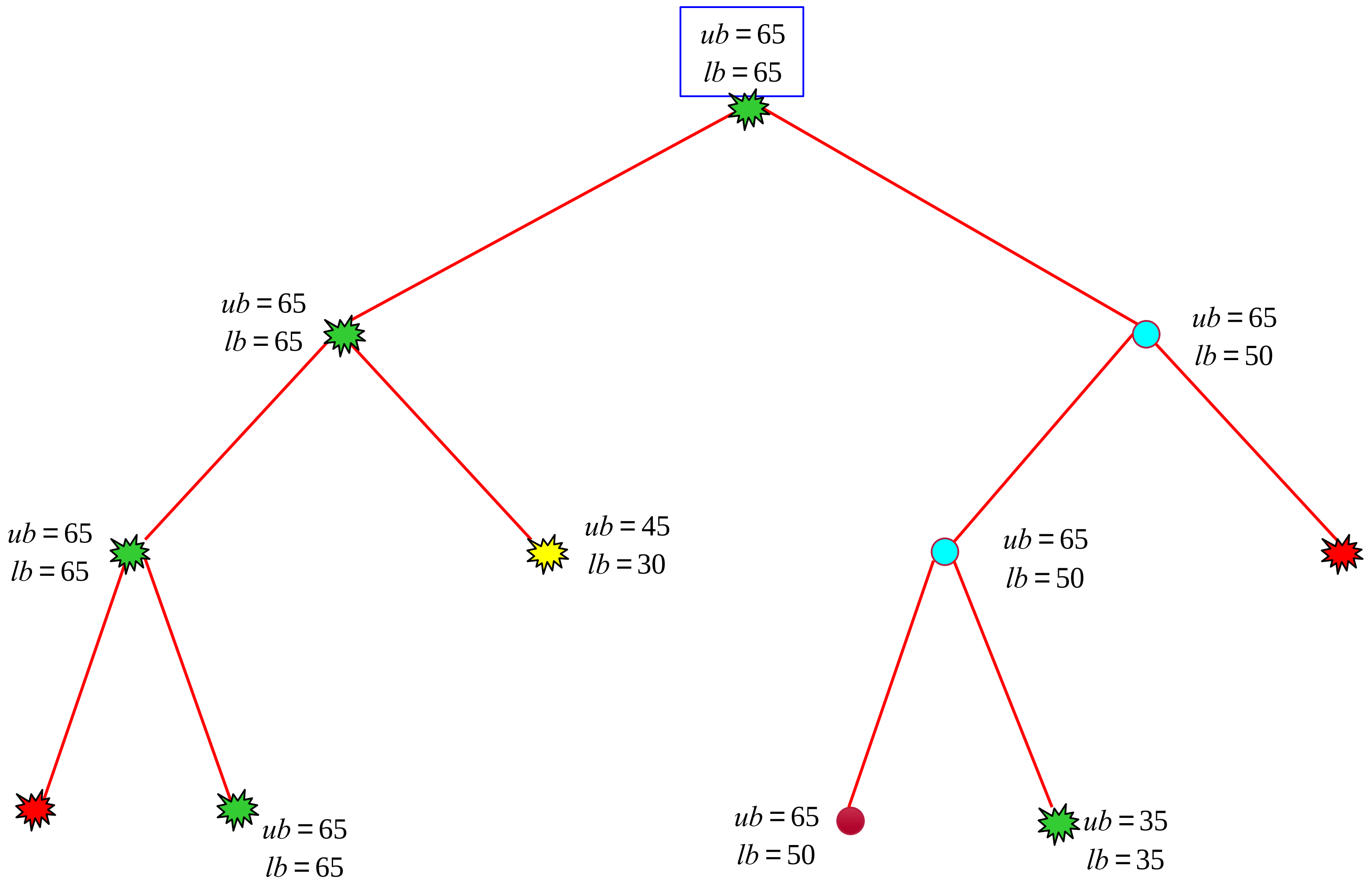
ALGORITMOS DE BRANCH-AND-BOUND

Problemas de maximização



ALGORITMOS DE BRANCH-AND-BOUND

Problemas de maximização



São muito usados para avaliar o desempenho de heurísticas

Maximização

$$gap_{H,i}\left(S^{H(i)}\right) = \frac{UB - valor(S^{H(i)})}{UB}$$

Minimização

$$gap_{H,i}\left(S^{H(i)}\right) = \frac{valor\left(S^{H(i)}\right) - LB}{LB}$$

HEURÍSTICAS DE BRANCH-AND-BOUND

Algoritmos de Branch-and-bound são muito eficientes para resolver o problema da *Programação Linear Inteira*

HEURÍSTICA DE BRANCH-AND-BOUND

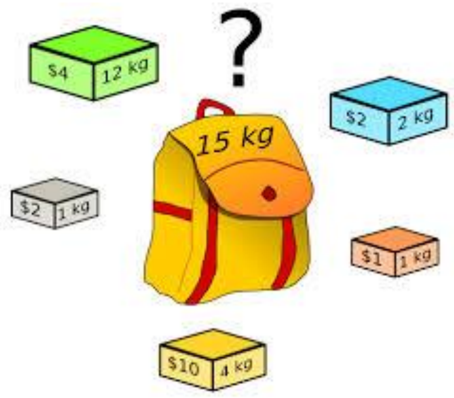
- Programação Linear Inteira (PLI)

$$\begin{array}{ll} \text{maximize} & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} & A\mathbf{x} \le \mathbf{b}, \\ & \mathbf{x} \ge \mathbf{0}, \\ \text{and} & \mathbf{x} \in \mathbb{Z}^n, \end{array}$$

$$\begin{aligned} \max\ & y \\ -x + y &\le 1 \\ 3x + 2y &\le 12 \\ 2x + 3y &\le 12 \\ x, y &\ge 0 \\ x, y &\in \mathbb{Z} \end{aligned}$$

- PLI é NP-Difícil

- É relativamente simples transformar problemas de otimização combinatória em PLI



$$\max \sum_{i \in I} c_i x_i \quad s.t.$$

$$\sum_{i \in I} a_i x_i \leq B$$

$$0 \leq x_i \leq 1$$

$$x_i \in \mathbb{Z}$$

HEURÍSTICA DE BRANCH-AND-BOUND

- Programação Linear ~~Inteira~~ (PL)

PLI

$$
\begin{array}{ll}
\text{maximize} & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\
\text{subject to} & A\mathbf{x} \leq \mathbf{b}, \\
 & \mathbf{x} \geq \mathbf{0}, \\
\text{and} & \mathbf{x} \in \mathbb{Z}^n,
\end{array}
$$

PL

$$
\begin{array}{ll}
\text{maximize} & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\
\text{subject to} & A\mathbf{x} \leq \mathbf{b}, \\
 & \mathbf{x} \geq \mathbf{0}, \\
\text{and} & \mathbf{x} \in \mathbb{R}^n,
\end{array}
$$

- PL é Polinomial!

HEURÍSTICA DE BRANCH-AND-BOUND

- Transforma o problema em um problema de *Programação Linear Inteira* (PLI)
- Executa o *Branch-and-bound* para PLI por uma quantidade polinomial de *nós*

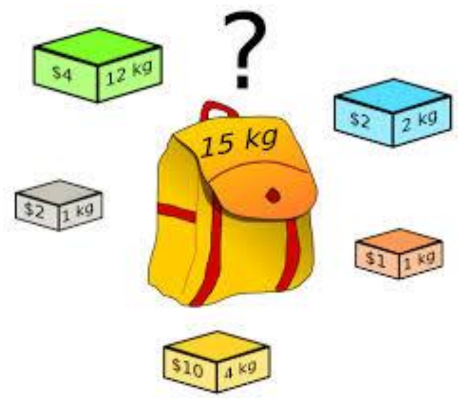
HEURÍSTICA DE BRANCH-AND-BOUND

Para problemas de maximização

- Limite inferior
  - Heurística polinomial para PLI
- Limite superior
  - Solução ótima da relaxação linear do PLI
  - Pode ser computada em $O(n^3)$

CORRETUDE

- Se a formulação estiver correta,
  - Por definição, a solução do PLI é viável



$$\max \sum_{i \in I} c_i x_i \quad s.t.$$

$$\sum_{i \in I} a_i x_i \leq B$$

$$0 \leq x_i \leq 1$$

$$x_i \in \mathbb{Z}$$

COMPLEXIDADE

- Se,
  - A formulação é polinomial
  - Os algoritmos que computam o LB e UB são polinomiais
  - E o número de nós avaliados é polinomial
- Então,
  - A complexidade da heurística é polinomial

APROXIMAÇÃO

- A princípio, nenhuma
  - Pois o número de nós avaliados pode ser arbitrariamente pequeno

HEURÍSTICAS CONSTRUTIVAS

Thiago Noronha (tfn@dcc.ufmg.br)

PROBLEMA DA MOCHILA QUADRÁTICA (QKS)

PROBLEMA DA MOCHILA QUADRÁTICA (QKS)

$$I = \{1, 2, \ldots, |I|\}$$
$$q_{ij} \in N$$
$$w_i \in N$$
$$c \in N$$

$$\max \sum_{i \in I} \sum_{j \in I} q_{ij} x_i x_j \quad s.t.$$
$$\sum_{i \in I} w_i x_i \leq c$$
$$x_i \in \{0,1\}$$

PROBLEMA DA MOCHILA QUADRÁTICA (QKS)

- Exemplo da matriz de benefício

$$\begin{array}{c|cccc|}
 & 1 & 2 & 3 & 4 \\
\hline
1 & 1 & 3 & 0 & 5 \\
2 & 3 & 1 & 4 & 2 \\
3 & 0 & 4 & 1 & 0 \\
4 & 5 & 2 & 0 & 1
\end{array}$$

PROBLEMA DA MOCHILA QUADRÁTICA (QKS)

- Exemplo da matriz de benefício

- $S = \{2, 4\}$

$$
\begin{array}{c|cccc|}
 & 1 & 2 & 3 & 4 \\
\hline
1 & 1 & 3 & 0 & 5 \\
2 & 3 & 1 & 4 & 2 \\
3 & 0 & 4 & 1 & 0 \\
4 & 5 & 2 & 0 & 1
\end{array}
$$

PROBLEMA DA MOCHILA QUADRÁTICA (QKS)

- Exemplo da matriz de benefício
- $S = \{2, 4\}$
- Custo = 1 + 2 + 1 + 2 = 6

$$\begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ \hline 1 & 1 & 3 & 0 & 5 \\ 2 & 3 & \boxed{1} & 4 & \boxed{2} \\ 3 & 0 & 4 & 1 & 0 \\ 4 & 5 & \boxed{2} & 0 & \boxed{1} \end{array}$$

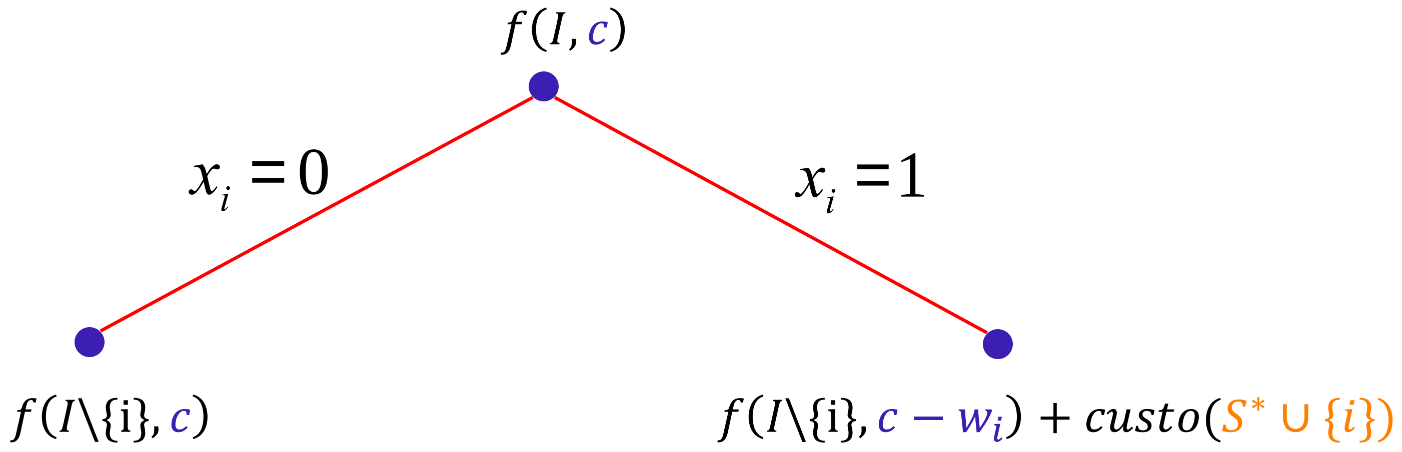
PRINCÍPIO DE PROJETO

Utilizar uma PD incompleta,

ou seja,

não considerar todos os subproblemas

HEURÍSTICA PD PARA QKS



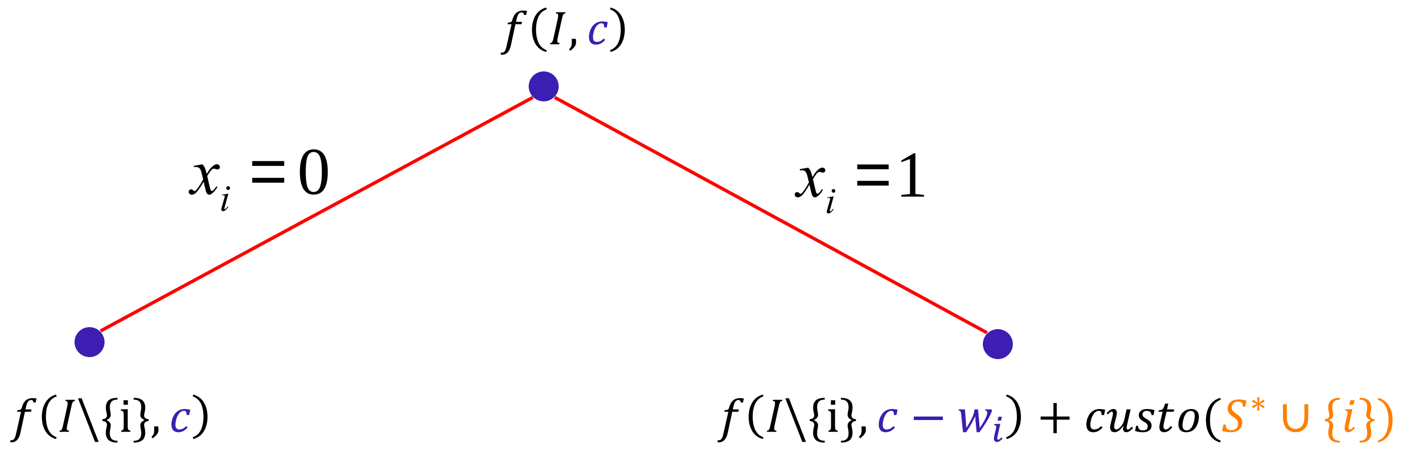
CORRETUDE

Da Heurística de PD

Seja $S(I, c)$ a solução de um subproblema $f(I, c)$

- $S(\{\ \}, c)$ é viável para qualquer c
- Assumindo $S(I \backslash \{i\}, c)$ e $S(I \backslash \{i\}, c - w_i)$ viáveis
  - $S(I, c)$= $\mathrm{S}(I \backslash \{i\}, c)$ é viável
  - $S(I, c)$= $\mathrm{S}(I \backslash \{i\}, c - w_i) \cup \{i\}$ também é viável

COMPLEXIDADE DE PD-QKS



$O(c \cdot |I|^2)$

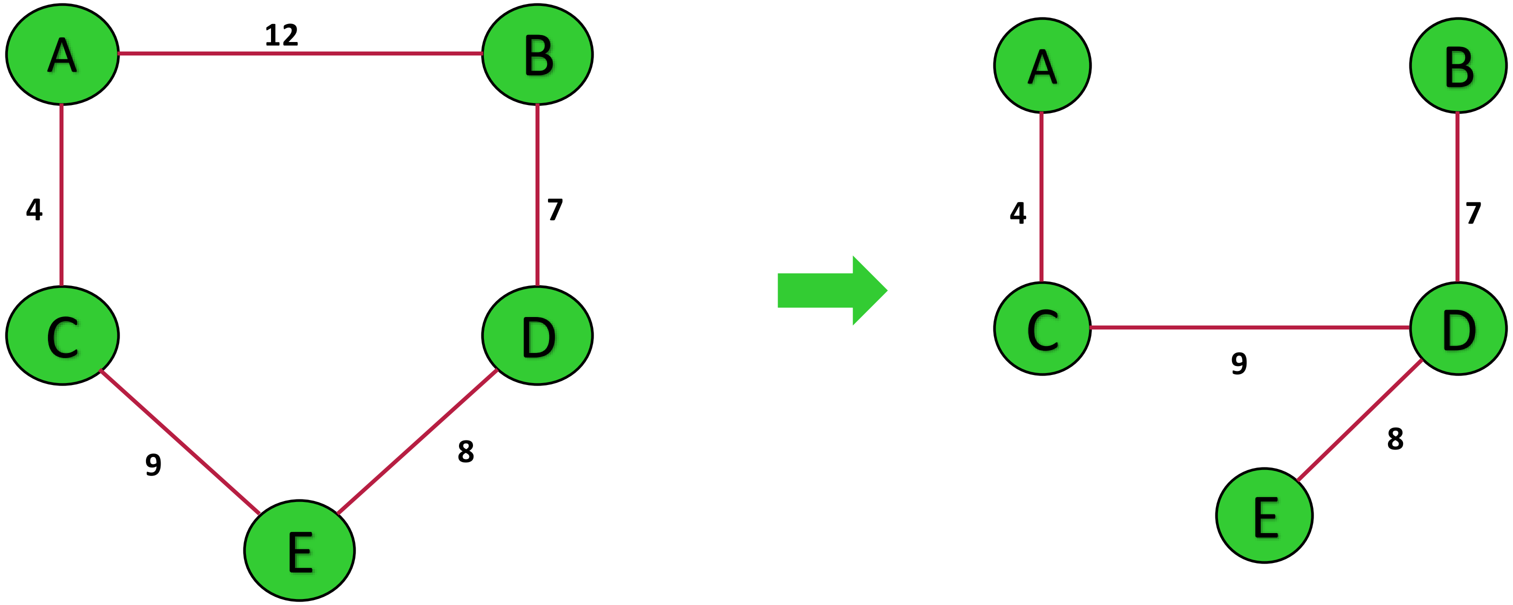
HEURÍSTICA DA ÁRVORE GERADORA MÍNIMA DUPLA

Redução ”heurística”

Redução ”heurística”

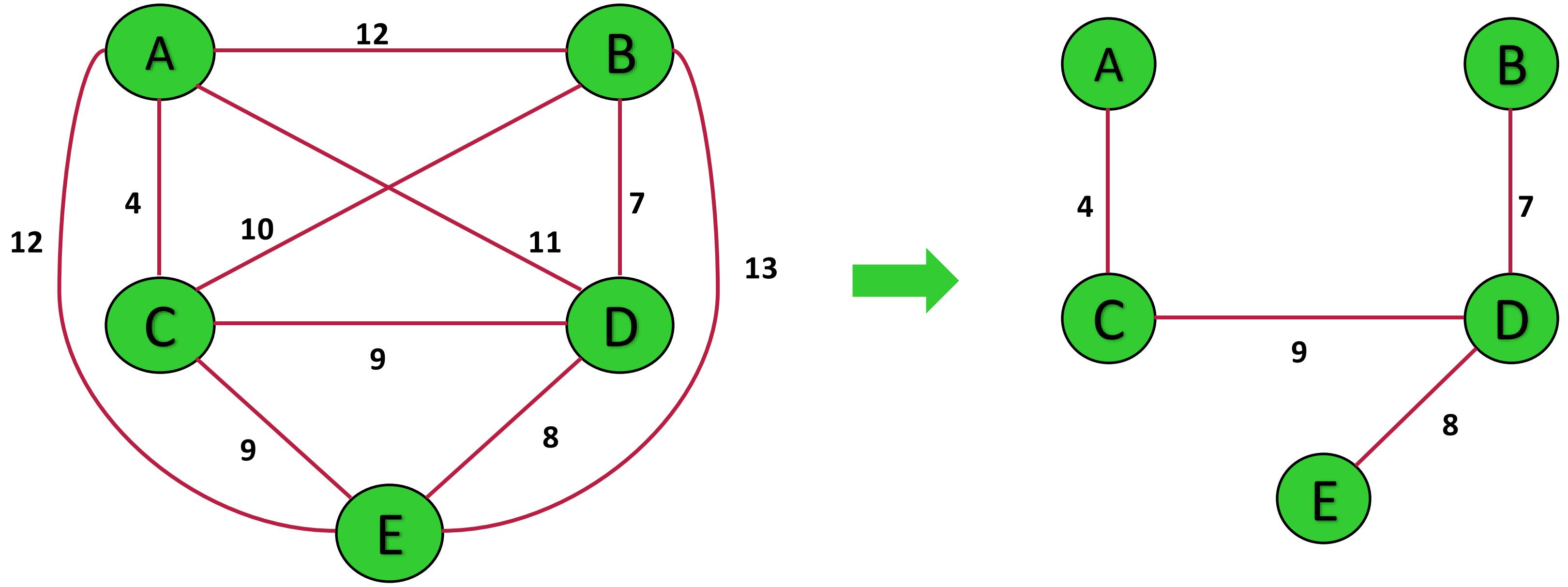
Parte da solução ótima de uma problema semelhante e mais simples

HEURÍSTICA DA ÁRVORE GERADORA MÍNIMA DUPLA



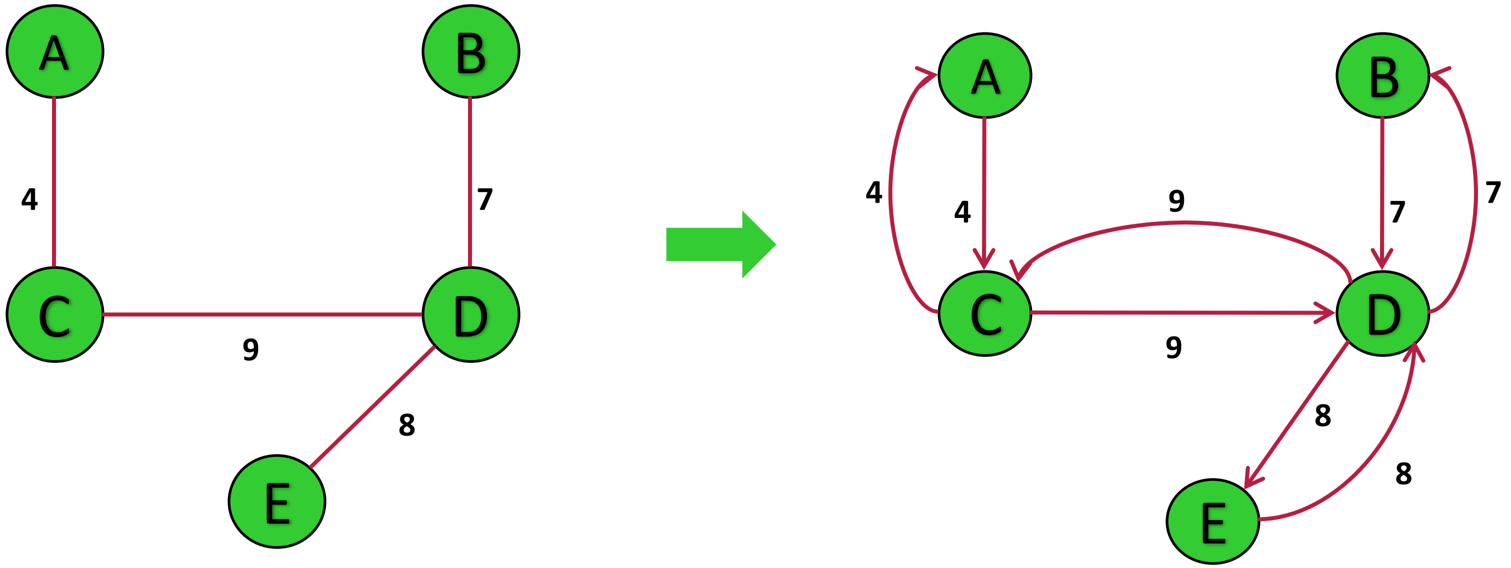
HEURÍSTICA DA ÁRVORE GERADORA MÍNIMA DUPLA

- Calcular a Árvore Geradora Mínima (AGM) do grafo



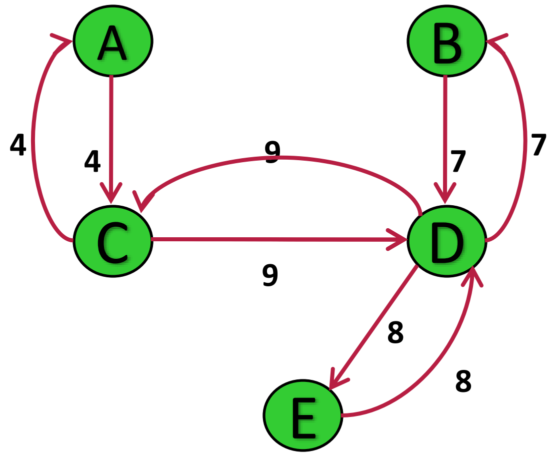
HEURÍSTICA DA ÁRVORE GERADORA MÍNIMA DUPLA

- Duplicar as arestas da AGM



HEURÍSTICA DA ÁRVORE GERADORA MÍNIMA DUPLA

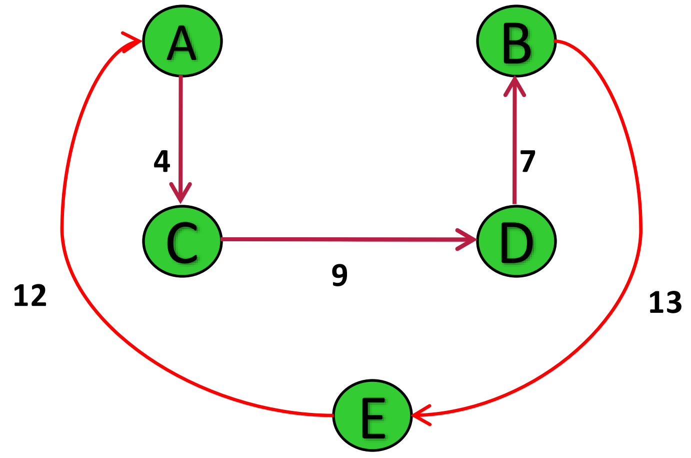
- Calcular um ciclo euleriano no grafo resultante, executando uma busca em profundidade



(A, C, D, B, D, E, D, C, A)

HEURÍSTICA DA ÁRVORE GERADORA MÍNIMA DUPLA

- Percorrer a lista de vértices visitados,
  - eliminando os vértices repetidos, e formando um ciclo hamiltoniano



(A, C, D, B, D, E, D, C, A) ➡ (A, C, D, B, E, A)

CORRETUDE

- Corretude para grafos completos
  - AGM contém todos os vértices do grafo
  - AGM dupla admite um ciclo euleriano
    - todos os vértices tem grau par
  - Qualquer vértice repetido pode ser eliminado
    - Existe uma aresta entre o seu antecessor e o sucessor
  - Todos os vértices repetidos são eliminados
    - O ciclo resultante contém todos os vértices sem repetição
  - Consequentemente, a solução resultante é viável

- $T(n) = O(m \log n) + O(n) + O(n) + O(n)$
- $T(n) = O(m \log n)$

APROXIMAÇÃO

- 2-aproximativa para o problema de TSP Euclidiano
  - Sejam
    - H^* o custo ciclo hamiltoniano ótimo,
    - T^* o custo da árvore geradora mínima
    - H o custo da solução retornada pela heurística
  - Temos que
    - $T^* \leq H^*$ e
    - $H \leq 2.T^*$,
  - Sendo assim
    - $H \leq 2.H^*$