PCC104 – Projeto e Análise de Algoritmos

Marco Antonio M. Carvalho

Departamento de Computação
Instituto de Ciências Exatas e Biológicas
Universidade Federal de Ouro Preto

Conteúdo

Fonte

Este material é baseado nos livros

- T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.
- S. Halim. *Competitive Programming*. 3rd Edition, 2013.
- Ian Parberry and William Gasarch. *Problems on Algorithms*. Second Edition, 2002.
- Ian Parberry *Lecture Notes on Algorithm Analysis and Complexity Theory*. Fourth Edition, 2001.

Licença

Algoritmos Gulosos

Ian Parberry e William Gasarch, *Problems on Algorithms*

"*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.

[*Most greedy algorithms are deceivingly simple.*]

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

Algoritmos Gulosos

Justificativa

Algoritmos para problemas de otimização tipicamente realizam uma sequência de passos, com um conjunto de opções a cada passo.

Para muitos destes problemas, utilizar busca completa ou outros paradigmas pode ser lento demais – algoritmos mais "simples" ou mais "espertos" os resolveriam.

Ainda, o problema pode ser tão difícil que abrimos mão de obtermos a solução ótima em trocade uma solução "razoavelmente boa" (subótima).

Definição

Um algoritmo guloso sempre faz a escolha que parece ser correta no momento, ou seja, escolhe sempre o **ótimo local** na esperança que isto o leve até a solução **ótima global**.

No entanto, nem todos os problemas permitem que algoritmos gulosos obtenham soluções ótimas.

Caracterização dos Algoritmos

O processo de construção da solução é iterativo, dividido em passos.

A cada passo, existe um conjunto ou lista de elementos candidatos a fazerem parte da solução.

A cada passo, um destes elementos candidatos é adicionado à solução.

Deve haver uma maneira de verificar se um conjunto específico de elementos produzem uma **solução completa** ou uma **solução parcial**.

Deve haver também uma maneira de verificar se uma dada solução parcial é **viável** ou não.

Uma **função de seleção** determina a cada passo, entre os elementos candidatos, qual é o mais promissor.

A **função de avaliação** determina o valor da solução encontrada, seja ela completa ou parcial.

Caracterização dos Algoritmos

No algoritmo a seguir:

- S: Conjunto de elementos que formam a solução;
- C: Conjunto de elementos candidatos a formar a solução;
- viável: Função que verifica a viabilidade de uma solução parcial ou completa;
- solução: Função que verifica se uma solução é completa.

Versão Genérica

Guloso(C)
Entrada: Conjunto de elementos candidatos C
$S \leftarrow \emptyset$;
enquanto $C \neq \emptyset$ ***e*** *!solução(*S*)* **faça**
  $x \leftarrow$ seleciona(C);
  $C \leftarrow C \setminus x$;
  se *viável(*$S \cup x$*)* **então**
    $S \leftarrow S \cup x$;
  fim
fim
se *solução(*S*)* **então**
  retorna S;
senão
  retorna $\emptyset$*;*
fim

Caracterização dos Problemas

Como determinar se um problema pode ser resolvido via algoritmo guloso?

Sabemos que algoritmos gulosos nem sempre funcionam.

Um problema deve possuir duas características para que os algoritmos gulosos sejam suficientes:

Subestrutura Ótima: A solução ótima do problema original é formada pelas soluções ótimas dos subproblemas;

Propriedade Gulosa: Se fizermos uma escolha que parece a melhor no momento e resolvermos os demais subproblemas da mesma maneira, ainda assim podemos atingir a solução ótima. **Nunca** é necessário reconsiderar as decisões tomadas anteriormente.

Se pudermos determinar estas duas características, teremos bons indícios de que pode funcionar.

Subestrutura Ótima

Um problema possui subestrutura ótima se a solução ótima global contiver as soluções ótimas dos subproblemas (ou ótimos locais parciais).

Esquema:

- Examinamos a solução global ótima;
- Mostramos que uma decisão específica pode ser o primeiro passo para construir a solução;
- Mostramos que esta escolha reduz o problema a uma versão similar, porém, menor;
- Por consequência, restam um ou mais subproblemas a serem resolvidos;
- Mostramos, por exemplo usando contradição, que as soluções para os subproblemas dentro da solução ótima também são ótimas.

Subestrutura Ótima

Implicitamente, este esquema utiliza indução nos subproblemas para provar que fazer escolhas gulosas produz uma solução ótima global.

Um problema que possui subestrutura ótima pode ser dividido sucessivamente, e a combinação das soluções ótimas dos subproblemas corresponde à solução ótima do problema original.

Ou seja, é possível diminuir o problema, e resolvê-lo incrementalmente com ótimos locais, pois eles construirão o ótimo global.

A propriedade de subestrutura ótima é primordial não só para os algoritmos gulosos, mas também para a Programação Dinâmica.

Propriedade Gulosa

Podemos construir a solução ótima global fazendo escolhas (gulosas) pelos ótimos locais.

Em outras palavras, quando fazemos uma escolha em um passo, a fazemos sem nos preocuparmos com os passos seguintes.

De fato, as escolhas atuais podem até se basear indiretamente nas escolhas feitas anteriormente, mas jamais podem se basear nas escolhas futuras.

Ao fazermos escolhas, diminuímos o tamanho do subproblema seguinte. Desta forma, é interessante que o critério guloso seja o mais eficiente possível.

Ótimo vs. Subótimo

Algoritmos gulosos podem ser aplicados a problemas que não possuam subestrutura ótima ou propriedade gulosa, ao custo de não haver garantia de qualidade da solução.

Com efeito, para muitos problemas não se conhecem **algoritmos exatos** (os quais sempre determinam as soluções ótimas globais) eficientes.

Nestas situações, torna-se interessante trocarmos a resposta ótima (aparentemente inalcançável) por uma resposta subótima (e alcançável com certa "facilidade").

Mesmo não havendo qualquer garantia sobre a qualidade da solução, um bom **algoritmo heurístico** guloso ainda pode encontrar as soluções ótimas para instâncias de um problema.

Exemplo – *Coin Change*

Suponha que possuímos uma grande número de moedas com diferentes denominações (25, 10, 5 e 1 centavos).

Dado um valor alvo, desejamos fazer troco utilizando o *menor número* de moedas possível.

Este tipo de problema é chamado de *Coin Change*.

Características – *Coin Change*

O que caracteriza uma instância de um problema do tipo *Coin Change* são a denominação das moedas utilizadas e o valor do troco.

Em algumas versões, não importa apenas a quantidade de moedas do troco, mas também a quantidade de moedas do pagamento, ou seja, deseja-se minimizar o número de moedas em toda a transação (pagamento + troco).

Algoritmo

"Use a moeda de maior denominação, tal que ela não seja maior do que o valor remanescente do troco."

Exemplo

Se as denominações são {25, 10, 5 e 1} centavos e queremos fazer um troco de 42 centavos, então podemos fazer:

- 42-25 = 17
- 17-10 = 7
- 7-5 = 2
- 2-1 = 1
- 1-1 = 0

O total de 5 moedas é a solução ótima!

Análise

O problema descrito anteriormente possui os dois "ingredientes" necessários para aplicarmos algoritmos gulosos com sucesso:

Subestrutura Ótima: As soluções ótimas para os subproblemas do problema original de 42 centavos estão contidas na solução do problema original:

- Para fazer 25 centavos, temos que usar 1 moeda de 25, portanto, a solução ótima;
- Para fazer 17 centavos, temos que usar 10+5+1+1 (portanto, 4 moedas, a solução ótima);
- Para fazer 7 centavos, temos que usar 5+1+1 (3 moedas), portanto, a solução ótima.

Propriedade Gulosa: Dada qualquer quantia V, subtraímos de maneira gulosa o maior valor de denominação que não seja maior que V. Pode ser provado que qualquer outra estratégia não levará à solução ótima.

Crítica

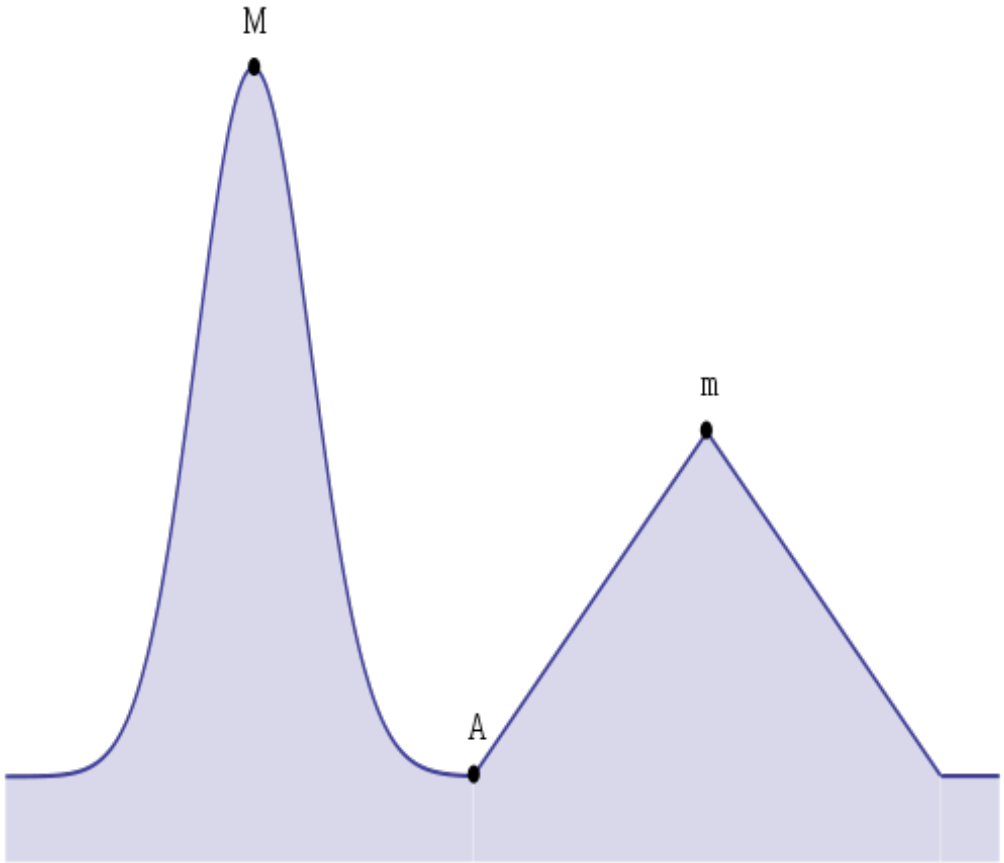
O algoritmo apresentado não funciona para todas as denominações de moedas.

Por exemplo, para $\{1, 3, 4\}$, o algoritmo falha miseravelmente. Tomemos como exemplo a quantia de 6 centavos:

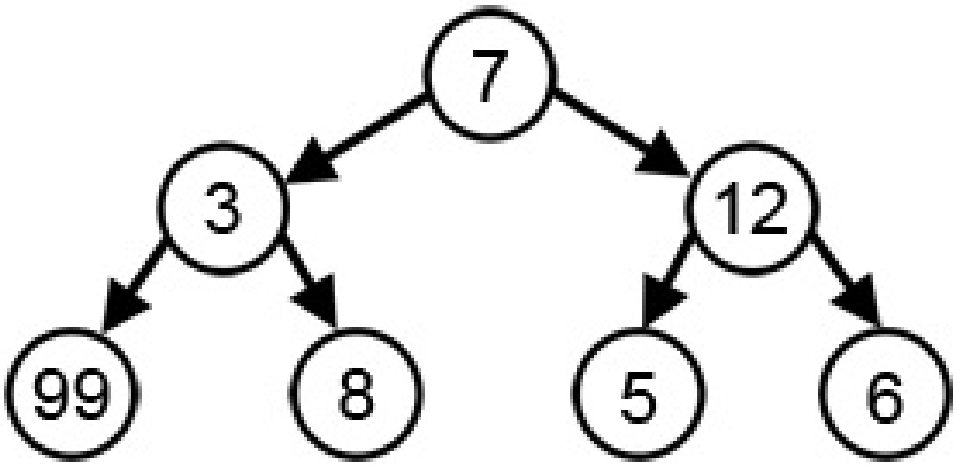
- O algoritmo nos daria a resposta $\{4, 1, 1\}$, com três moedas;
- No entanto, a solução ótima é $\{3, 3\}$.

Este não é o único problema ao qual não se aplicam algoritmos gulosos com sucesso. Com efeito, há vários:

- Problema do Caixeiro Viajante;
- Problema da Mochila 0-1;
- Problemas NP-Difíceis em geral...



Exemplo de falha: Ao buscar o máximo global, um algoritmo guloso encontraria m ao invés de M.



Exemplo de falha: Ao buscar o maior caminho de maior soma, qual solução o algoritmo guloso encontraria?

Elogio

Por outro lado, há vários exemplos de algoritmos gulosos de sucesso:

- O algoritmo de Kruskal para determinação de Árvores Geradoras Mínimas;
- O algoritmo de Djikstra (mesmo embora não seja puramente guloso) para determinação do Menor Caminho de Origem Única em grafos;
- Algoritmo de Huffman para compressão de dados;
- etc.

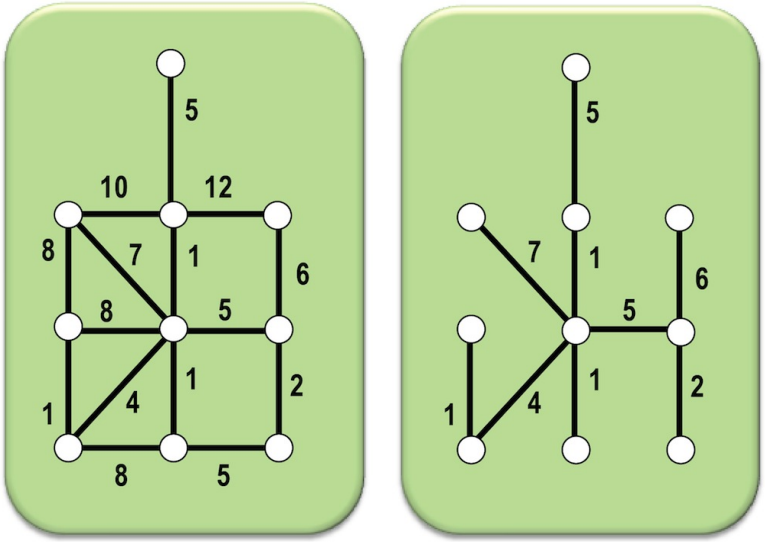
Há também vários problemas que "aceitam" soluções gulosas:

- Problema da mochila fracionária;
- Problema de escalonamento de intervalos;
- etc.

O Problema da Árvore Geradora Mínima

Uma árvore geradora de um grafo ponderado G é um subgrafo conexo e acíclico que possui todos os vértices originais de G e um subconjunto das arestas originais de G.

A árvore geradora de custo mínimo é a árvore geradora de menor custo dentre todas as possíveis em um grafo.



Grafo de exemplo e árvore geradora de custo mínimo.

Algoritmo Guloso 1

"Selecione sucessivamente as arestas de **menor custo** que não gerem ciclos."

Histórico

Este algoritmo foi proposto 1930 pelo matemático tcheco Vojtěch Jarník.

O mesmo algoritmo foi novamente proposto pelo cientista da computação americano Robert C. Prim (* 1921 – † 2009) em 1957 e redescoberto posteriormente pelo holandês Edsger Dijkstra em 1959.

Algoritmo Guloso 1

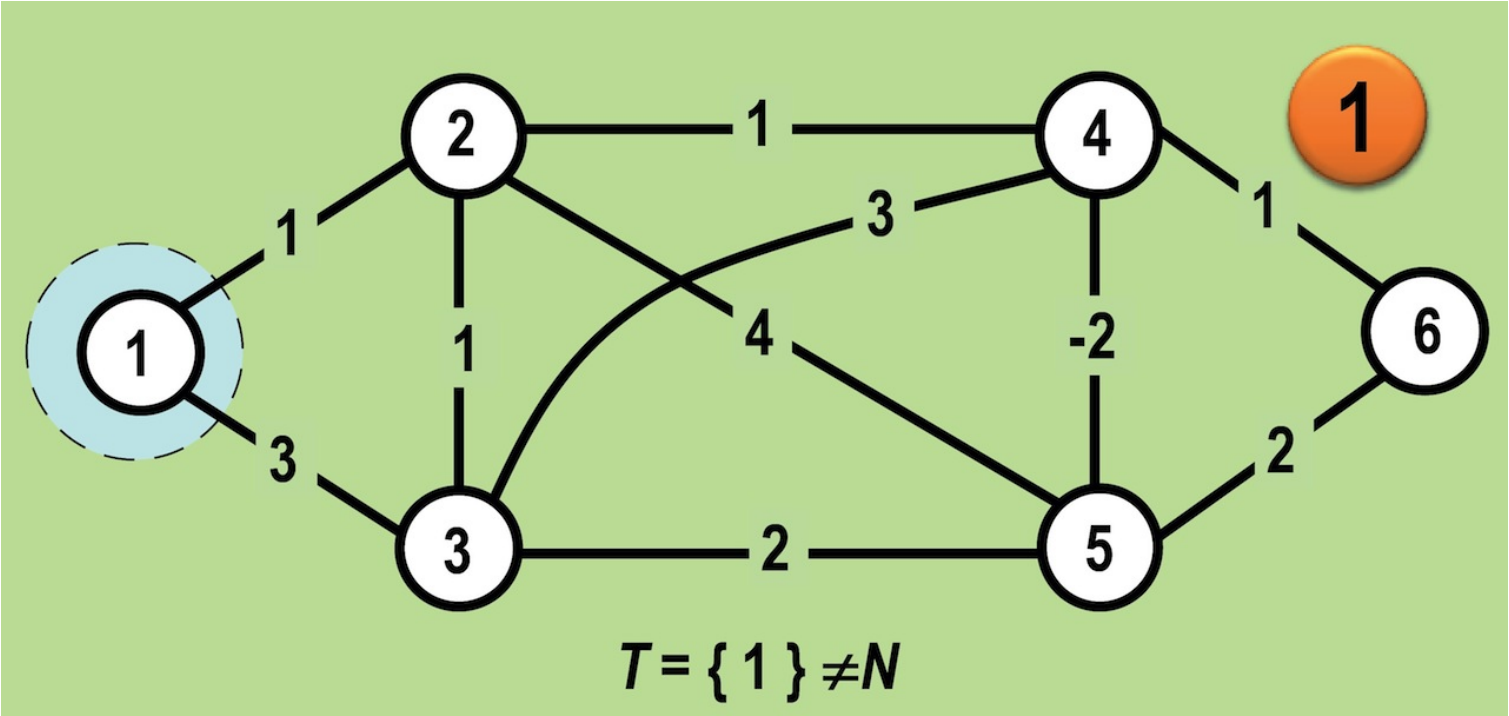
"Selecione sucessivamente as arestas de **menor custo** que não gerem ciclos."

Histórico

Este algoritmo foi proposto 1930 pelo matemático tcheco Vojtěch Jarník.

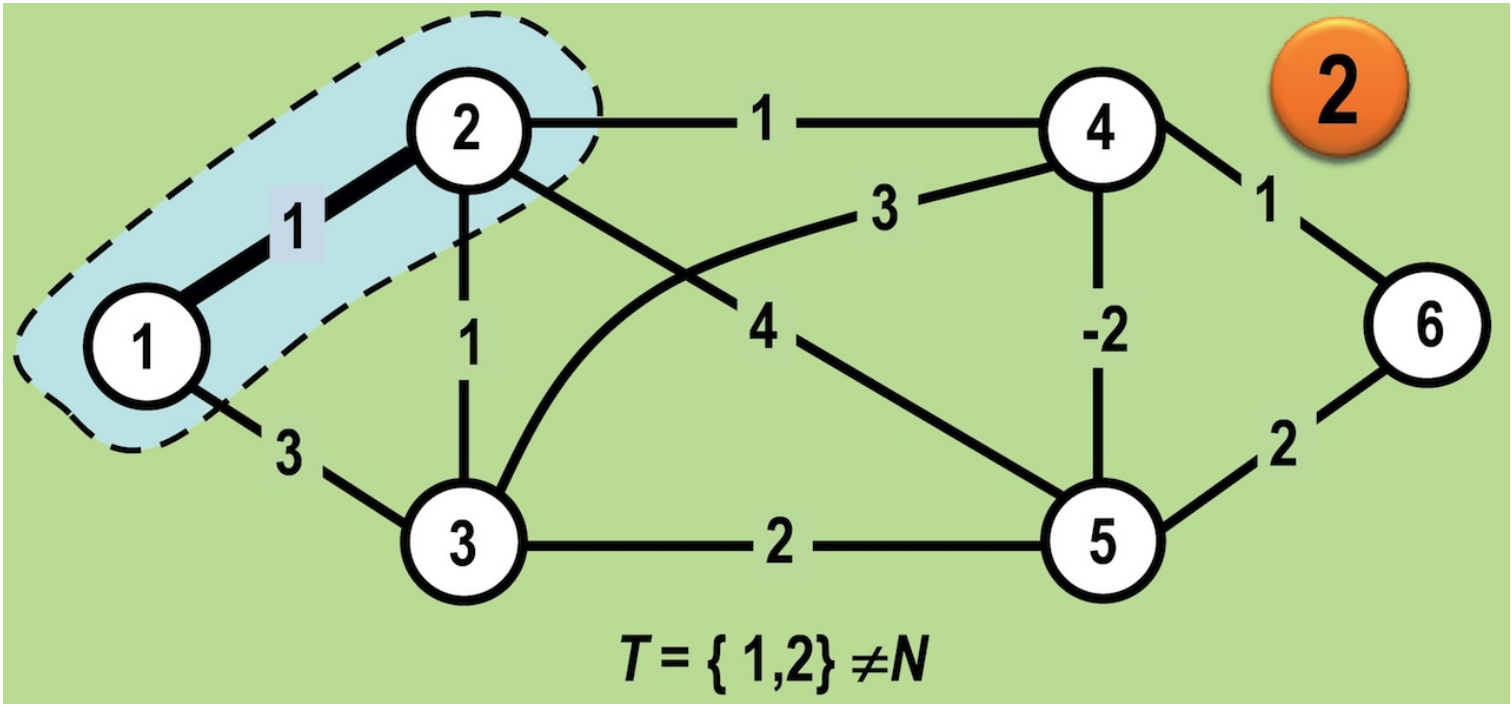
O mesmo algoritmo foi novamente proposto pelo cientista da computação americano Robert C. Prim (* 1921 – † 2009) em 1957 e redescoberto posteriormente pelo holandês Edsger Dijkstra em 1959.

Exemplo



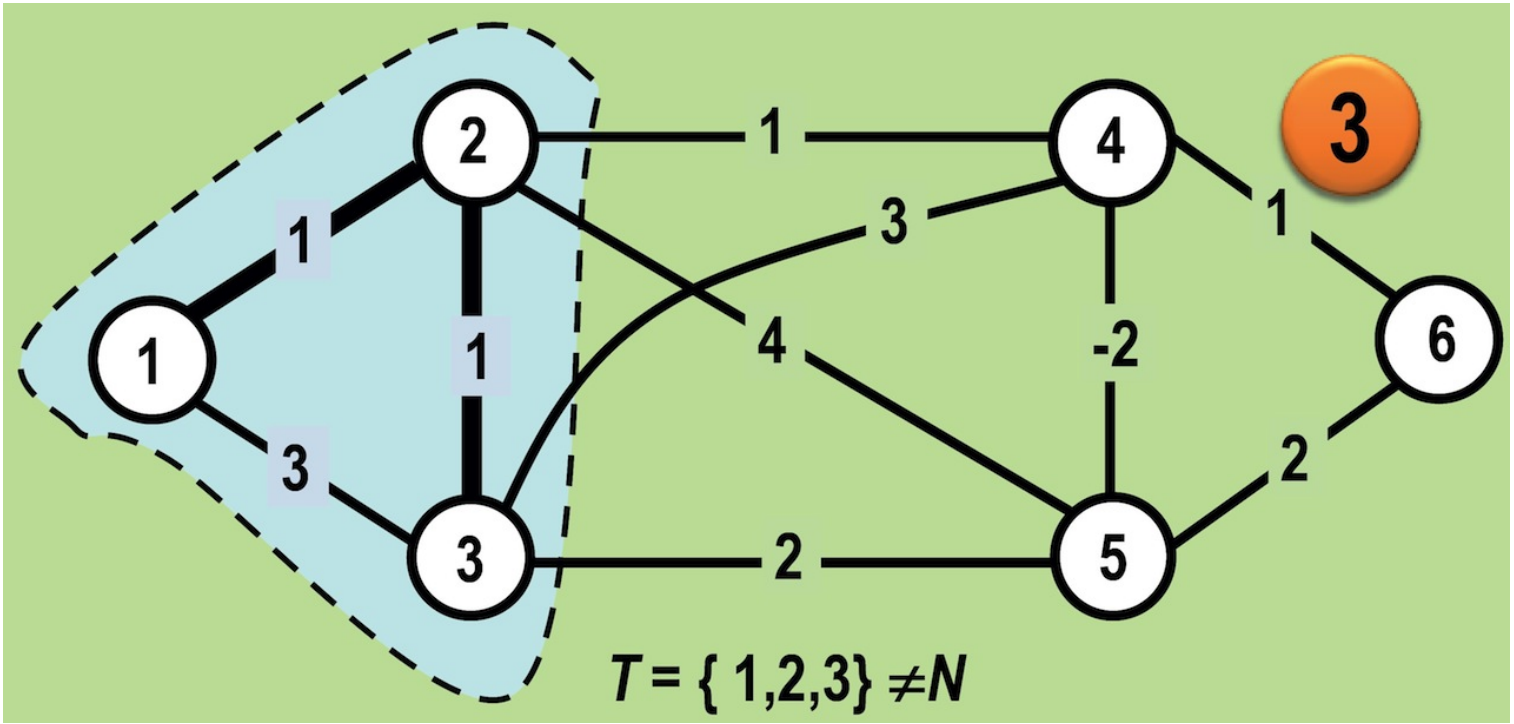
Grafo de exemplo. O vértice 1 é o primeiro a ser escolhido.

Exemplo



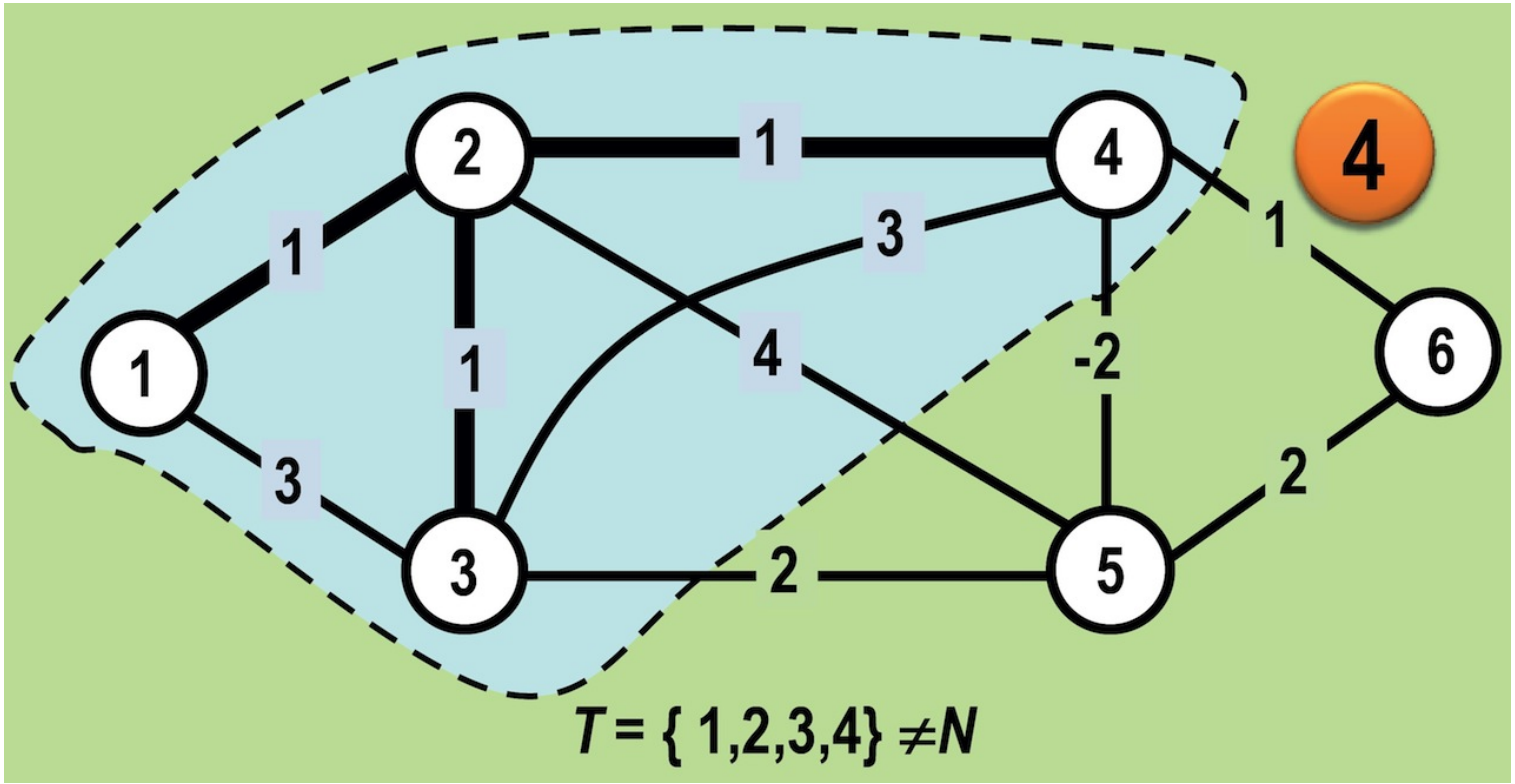
Inserção do vértice 2 e da aresta $\{1, 2\}$.
A região em azul indica os vértices escolhidos.

Exemplo



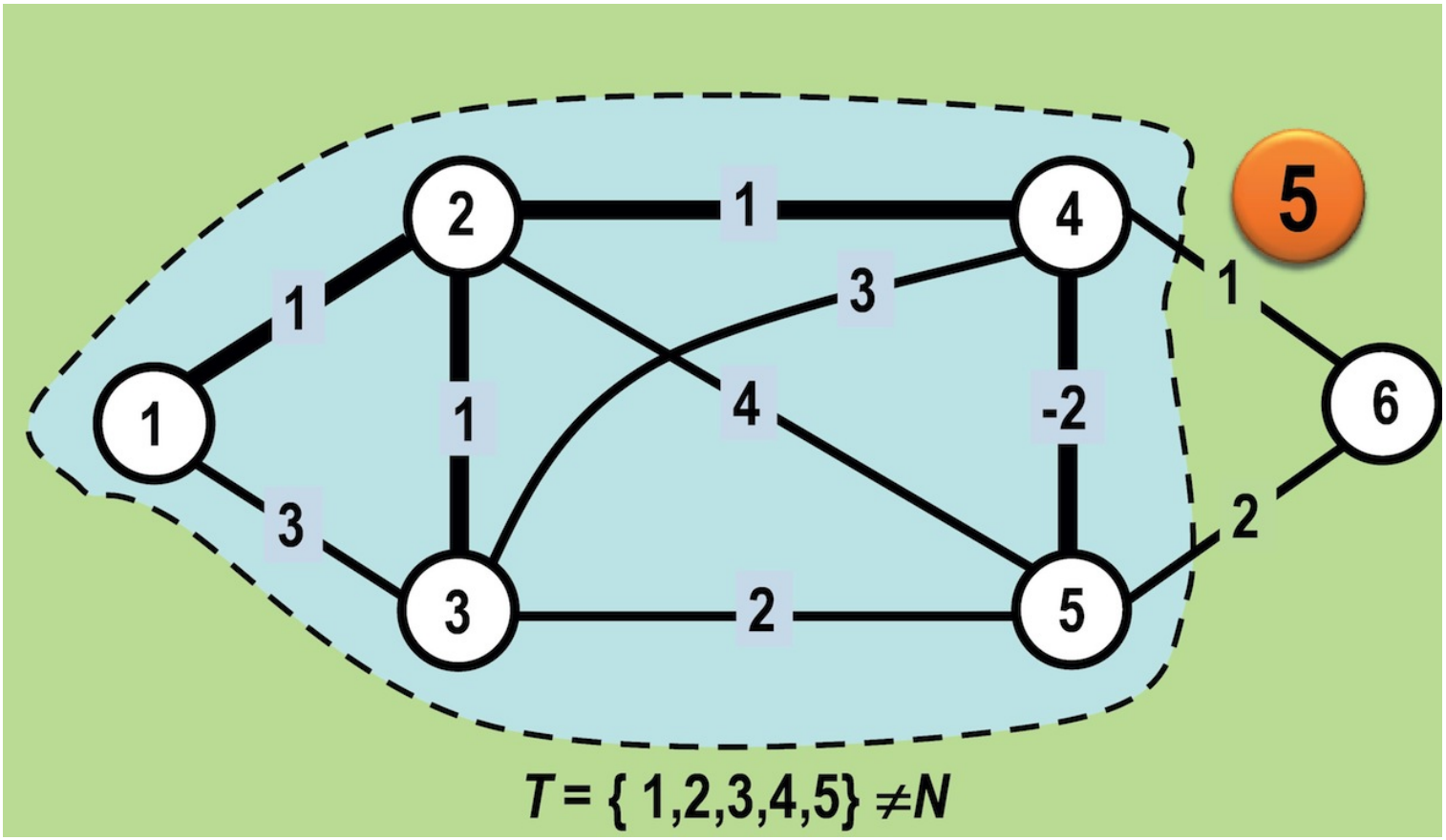
Inserção do vértice 3 e da aresta {2, 3}.

Exemplo



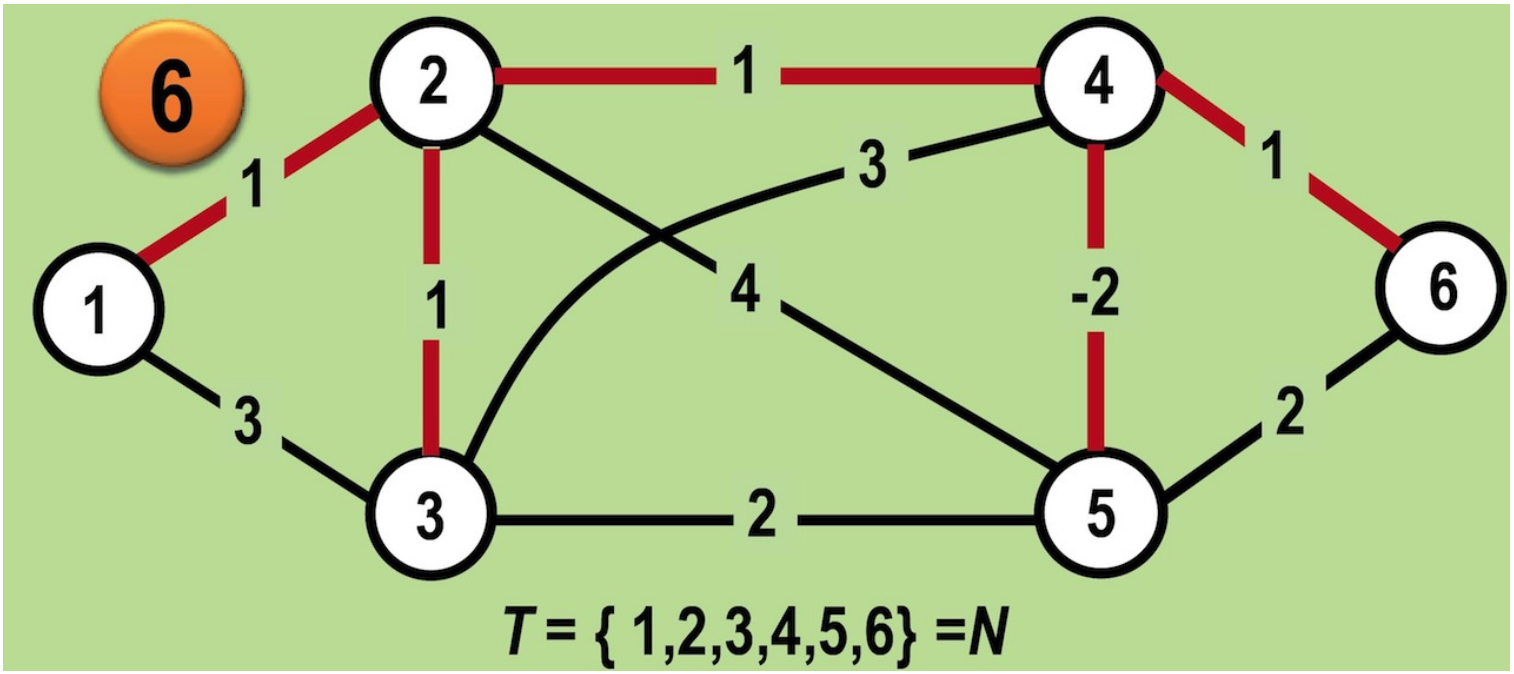
Inserção do vértice 4 e da aresta $\{2, 4\}$.

Exemplo



Inserção do vértice 5 e da aresta $\{4, 5\}$.

Exemplo



Inserção do vértice 6 e da aresta $\{4, 6\}$.
A árvore geradora mínima foi determinada.

Algoritmo Guloso 2

"Inclua vértices na árvore sucessivamente, selecionando sempre as arestas de **menor custo** que não formem ciclos."

Histórico

Este algoritmo foi proposto em 1956 por Joseph Bernard Kruskal Jr. (* 1928 – † 2010), estatístico, matemático, cientista da computação e psicometrista americano.

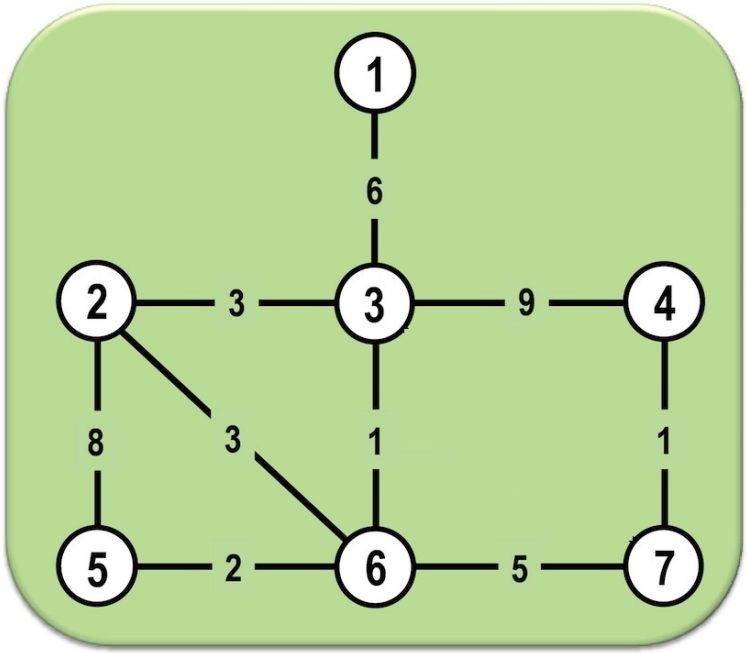
Algoritmo Guloso 2

"Inclua vértices na árvore sucessivamente, selecionando sempre as arestas de **menor custo** que não formem ciclos."

Histórico

Este algoritmo foi proposto em 1956 por Joseph Bernard Kruskal Jr. (* 1928 – † 2010), estatístico, matemático, cientista da computação e psicometrista americano.

Exemplo

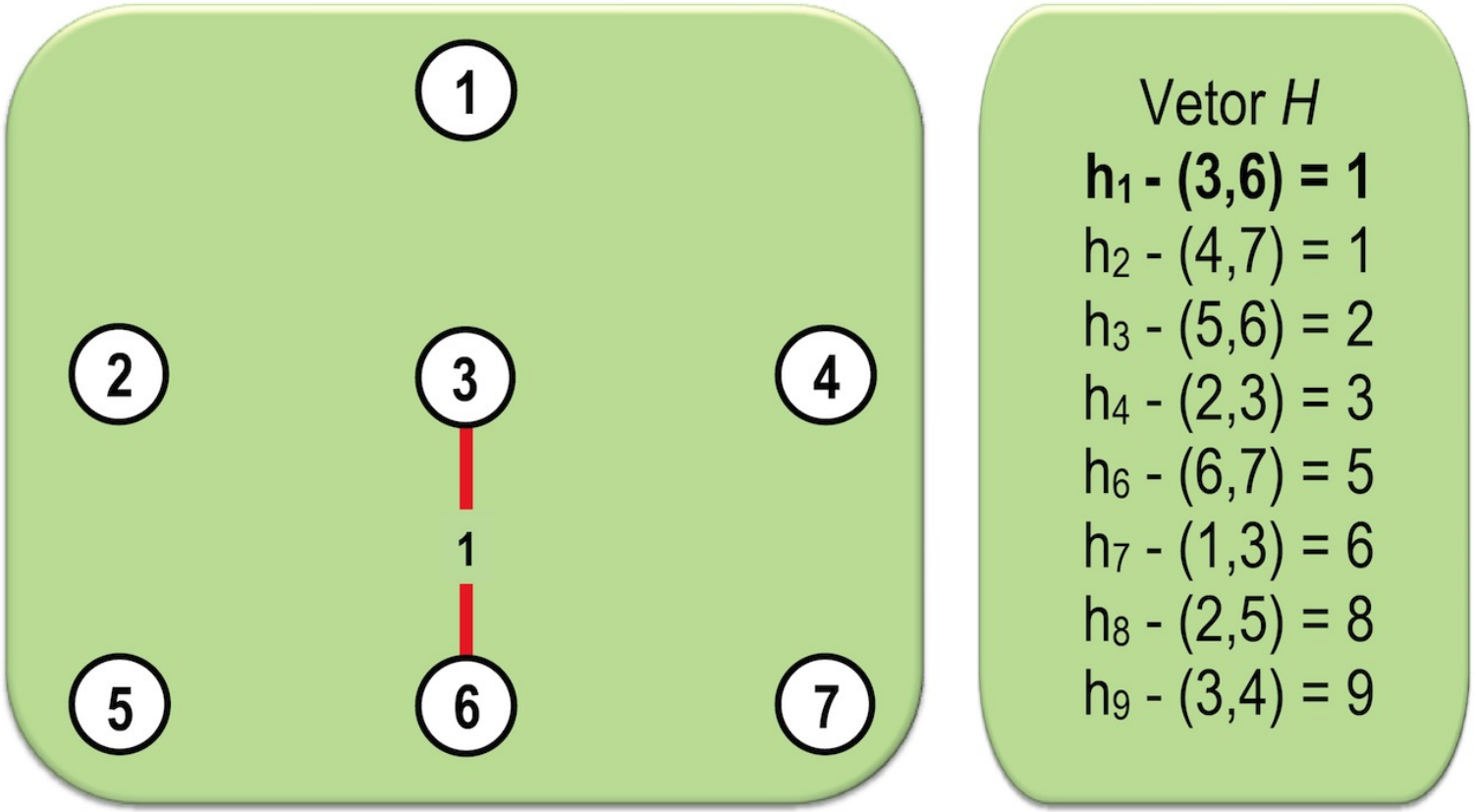


Vetor H

h_1 - (3,6) = 1; h_2 - (4,7) = 1; h_3 - (5,6) = 2;
h_4 - (2,3) = 3; h_5 - (2,6) = 3; h_6 - (6,7) = 5
h_7 - (1,3) = 6; h_8 - (2,5) = 8; h_9 - (3,4) = 9

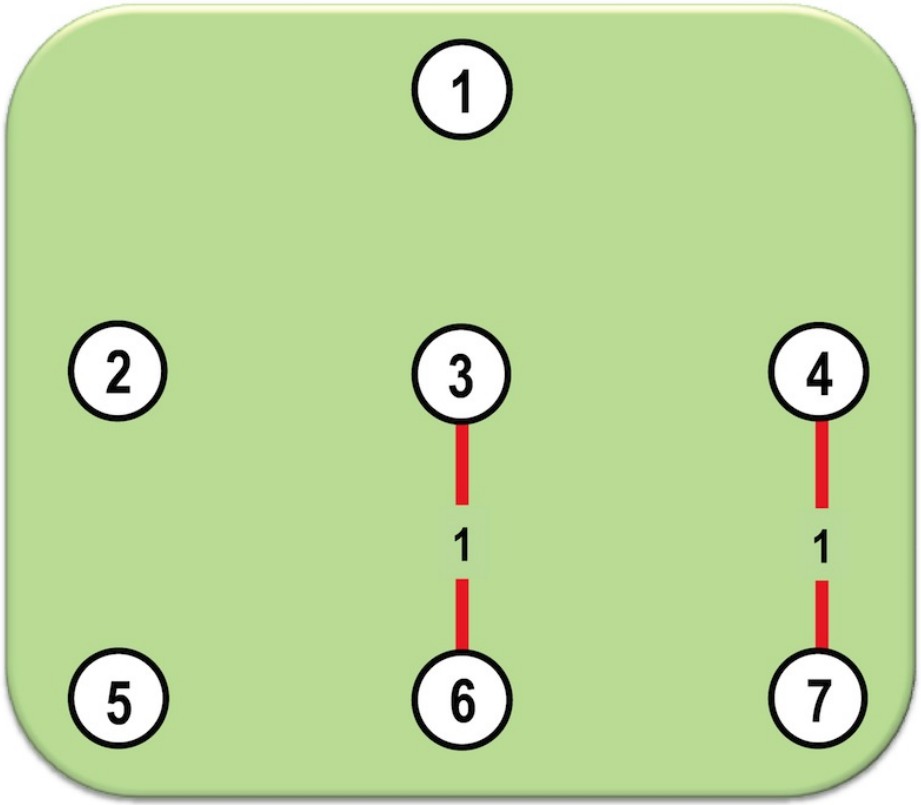
Grafo de exemplo e vetor H desordenado.

Exemplo



Inserção da primeira aresta em T.

Exemplo



Vetor H

h_1 - (3,6) = 1

h_2 - (4,7) = 1

h_3 - (5,6) = 2

h_4 - (2,3) = 3

h_5 - (2,6) = 3

h_6 - (6,7) = 5

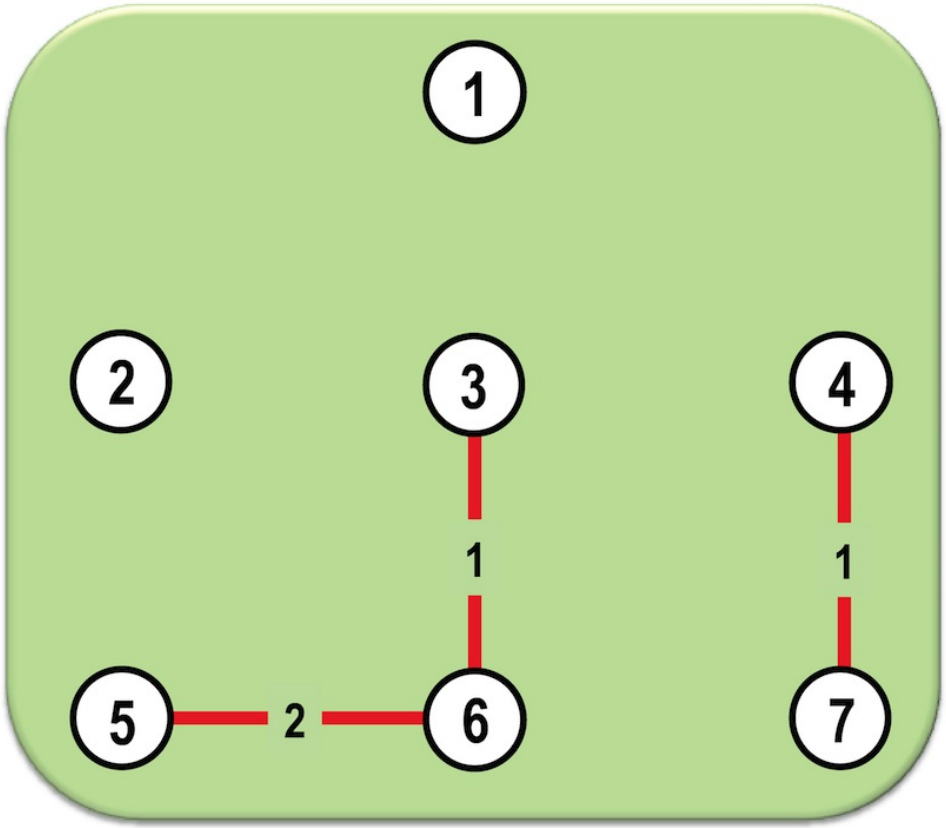
h_7 - (1,3) = 6

h_8 - (2,5) = 8

h_9 - (3,4) = 9

Inserção da segunda aresta em T.

Exemplo

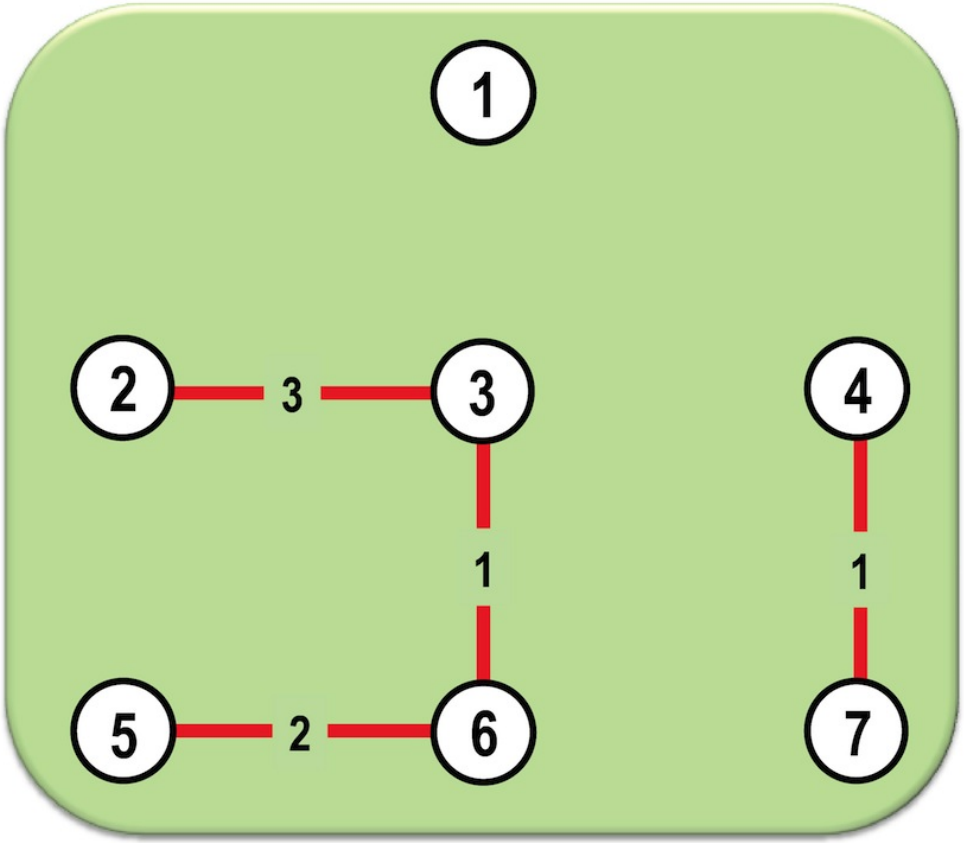


Vetor H

h_1 - (3,6) = 1
h_2 - (4,7) = 1
h_3 - (5,6) = 2
h_4 - (2,3) = 3
h_5 - (2,6) = 3
h_6 - (6,7) = 5
h_7 - (1,3) = 6
h_8 - (2,5) = 8
h_9 - (3,4) = 9

Inserção da terceira aresta em T.

Exemplo

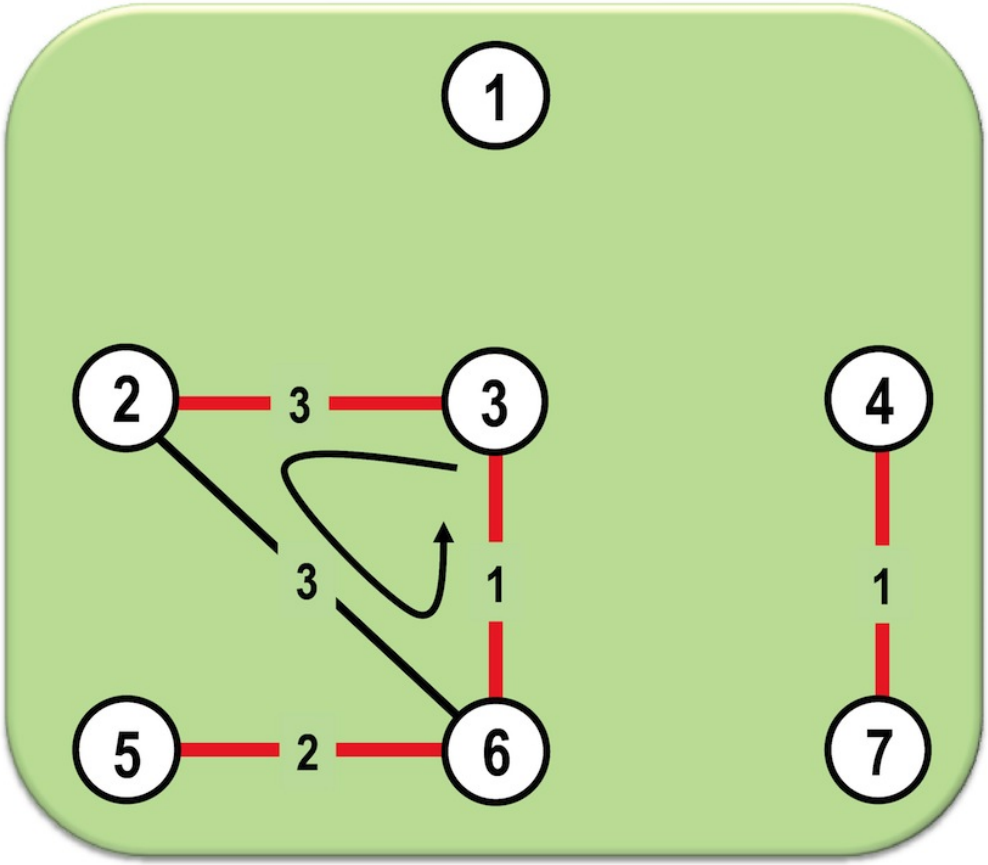


Vetor H

h_1 - (3,6) = 1
h_2 - (4,7) = 1
h_3 - (5,6) = 2
h_4 - (2,3) = 3
h_5 - (2,6) = 3
h_6 - (6,7) = 5
h_7 - (1,3) = 6
h_8 - (2,5) = 8
h_9 - (3,4) = 9

Inserção da quarta aresta em T.

Exemplo

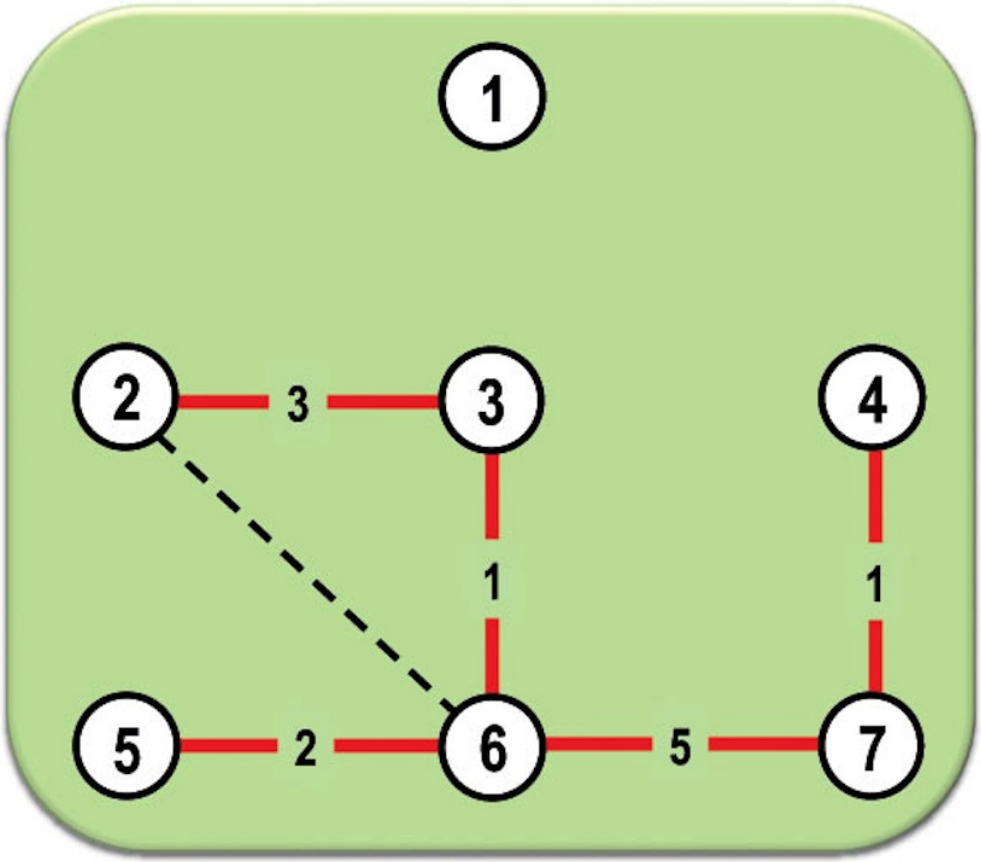


Vetor H

h_1 - (3,6) = 1
h_2 - (4,7) = 1
h_3 - (5,6) = 2
h_4 - (2,3) = 3
h_5 - (2,6) = 3
h_6 - (6,7) = 5
h_7 - (1,3) = 6
h_8 - (2,5) = 8
h_9 - (3,4) = 9

Tentativa de inserção da quinta aresta em T.

Exemplo



Vetor H

h_1 - (3,6) = 1
h_2 - (4,7) = 1
h_3 - (5,6) = 2
h_4 - (2,3) = 3
h_5 - (2,6) = 3
h_6 - (6,7) = 5
h_7 - (1,3) = 6
h_8 - (2,5) = 8
h_9 - (3,4) = 9

Inserção da quinta aresta em T.

O Problema da Mochila

Dadas uma mochila de capacidade W e uma lista de n itens distintos e únicos (enumerados de 1 a n), cada um com um peso $w_1, w_2, \ldots, w_n$ e um valor $v_1, v_2, \ldots, v_n$, maximizar o valor carregado na mochila, respeitando sua capacidade.

Versão 1 – O Problema da Mochila Fracionária

Nesta versão do problema, é possível fracionar os itens, caso seja necessário, para preencher a mochila.

Versão 2 – O Problema da Mochila 0-1

Nesta versão do problema, não é possível fracionar os itens.

Algoritmos Gulosos

Algoritmo Guloso 1

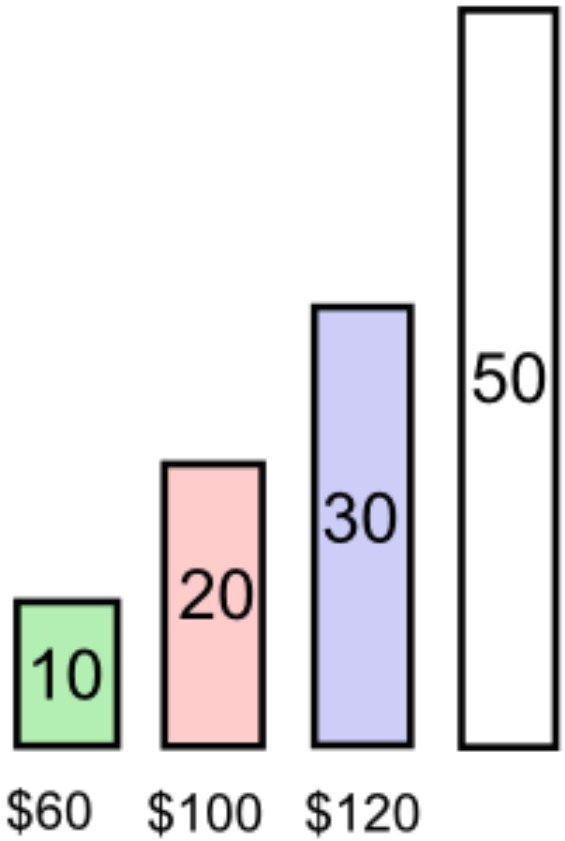
"Selecione sucessivamente o item de **maior valor** que caiba na mochila."

Algoritmo Guloso 2

"Selecione sucessivamente o item de **menor peso** que caiba na mochila."

Algoritmo Guloso 3

"Selecione sucessivamente o item de **maior valor por unidade de peso** que caiba na mochila."

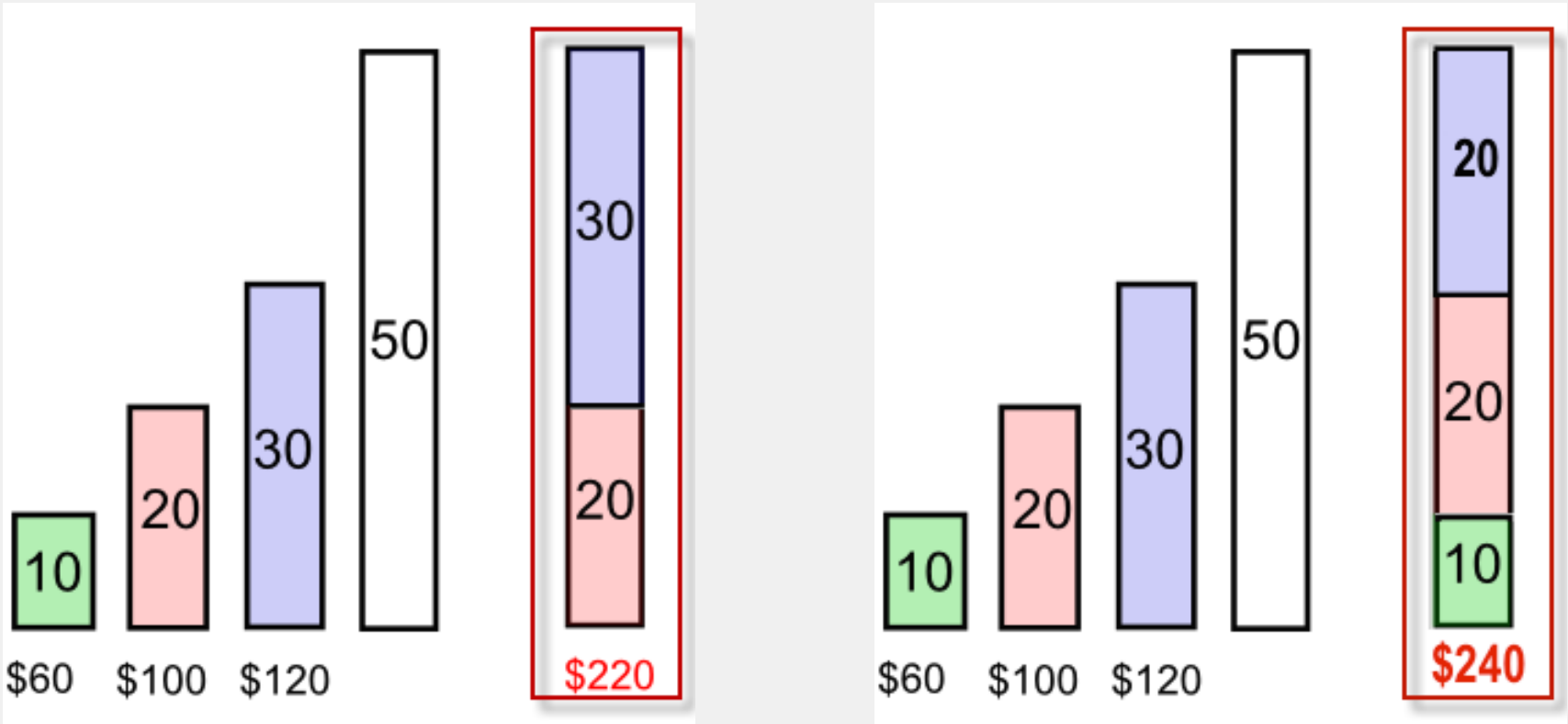


Instância do problema da mochila: 3 itens e mochila de capacidade 50.

Exercício

Determine a solução obtida por cada um dos 3 algoritmos apresentados.

Soluções Ótimas



Problema da Problema da Mochila 0-1 e Mochila Fracionária.

```
1  MochilaFracionaria(w[1,...,n], v[1,...,n], W)
   Entrada: pesos w, valores v, capacidade W
2  para i = 1 até n faça
3  |   x[i] ← 0;
4  fim
5  enquanto peso < W faça
6  |   i ← índice do melhor objeto;
7  |   se peso+w[i] ≤ W então
8  |   |   x[i] ← 1;
9  |   |   peso←peso+w[i];
10 |   senão
11 |   |   x[i] ← (W−peso)/w[i];
12 |   |   peso← W;
13 |   fim
14 fim
15 retorna x;
```

Análise - Problema da Mochila Fracionária

Subestrutura Ótima: As soluções ótimas para os subproblemas do problema original de 50 unidades de peso estão contidas na solução do problema original:

- Para uma mochila de peso 10, selecionamos o item verde, com peso 10 e valor \$60, portanto, a solução ótima (as alternativas seriam 1/2 objeto rosa com peso 10 e valor \$50, ou 1/3 do objeto azul, com peso 10 e valor \$40);
- Para uma mochila de peso 30, selecionamos o item verde e o rosa, com peso 30 e valor \$160, novamente, a solução ótima (a alternativa seria selecionar o item azul, com peso 30 e valor \$120).

Propriedade Gulosa: Dado qualquer peso W, subtraímos de maneira gulosa a maior fração da melhor relação valor/peso que não seja maior que W e que não implique em selecionar mais do que 100% de um item. Mesmo informalmente, fica claro que esta estratégia sempre levará à solução ótima.

Análise - Problema da Mochila 0-1

Subestrutura Ótima: As soluções ótimas para os subproblemas do problema original de 50 unidades de peso estão contidas na solução do problema original?

- Para uma mochila de peso 10, selecionamos o item verde, com peso 10 e valor \$60, portanto, a solução ótima (não há alternativas);
- Para uma mochila de peso 30, selecionamos o item verde e o rosa, com peso 30 e valor \$160, novamente, a solução ótima (a alternativa seria selecionar o item azul, com peso 30 e valor \$120);
- Para uma mochila de peso 50, selecionamos os itens rosa e verde, com peso 30 e valor \$160, o que não é a solução ótima (itens rosa e azul, com peso 50 e valor \$220).

Load Balancing

Pode ser difícil derivar uma estratégia gulosa. Com efeito, encontrar algoritmos gulosos é uma arte, assim como encontrar bons algoritmos de busca completa exige criatividade.

Load Balancing ou Balanceamento de Carga é um tipo de algoritmo guloso bem conhecido.

Uma dica que surge deste exemplo: se não houver nenhuma estratégia gulosa óbvia, tente ordenar os dados ou inserir dados artificiais e ver se uma estratégia gulosa emerge.

Load Balancing

Dados:

- $1 \leq C \leq 5$ tubos que podem armazenar 0, 1 ou 2 amostras;
- $1 \leq S \leq 2C$ amostras; e
- Uma lista M dos volumes das S amostras.

O objetivo é determinar em qual tubo cada amostra deve ser armazenada, de maneira a minimizar o "desequilíbrio" entre os tubos.

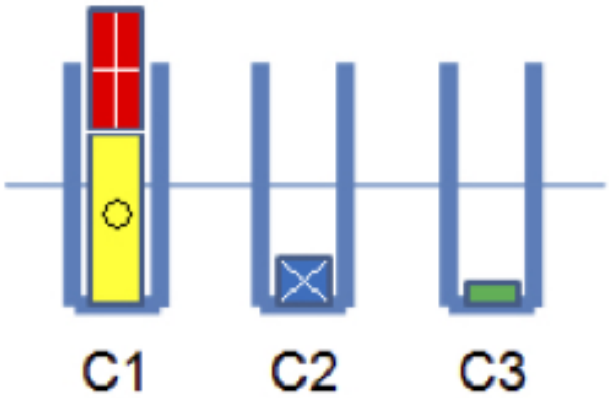
Load Balancing

Para a ilustração a seguir, considere:

- $A = \sum_{j=1}^{S} M_j/C$ é a média da massa total em cada um dos C tubos;
- **IMBALANCE** $= \sum_{i=1}^{C} |X_i - A|$ é a soma das diferenças entre a massa em cada tubo e A, em que X_i é a massa total no tubo i.

A = 5

C = 3, **S** = 4, **M** = {5, 1, 2, 7}
Average mass / chamber
A = (5 + 1 + 2 + 7) / 3 = 5



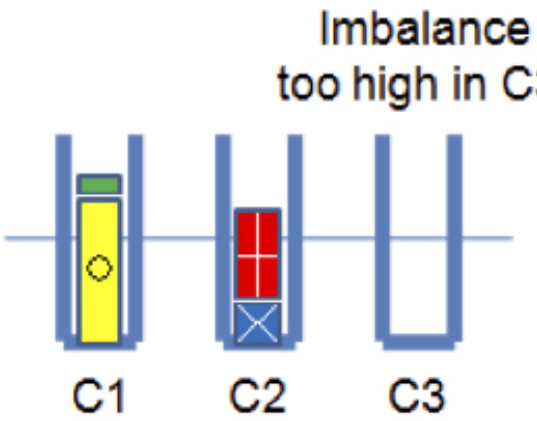
A = 5

IMBALANCE =
|(7+5)-5| + |2-5| + |1-5| =
7 + 3 + 4 = **14**

Load Balancing

Ao analisarmos este problema para projetarmos um algoritmo, fazemos algumas observações:

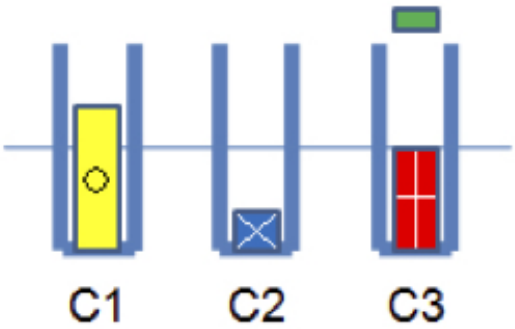
1. Dado que $C \leq 5$ e $S \leq 10$, é viável a aplicação de um algoritmo de busca completa;
2. Se há um tubo vazio, é preferível mover uma amostra de um tubo com duas amostras para o tubo vazio – caso contrário, o tubo vazio aumentaria o desequilíbrio;
3. Se $S > C$, então $S - C$ amostras deverão ser inseridas em tubos não vazios – princípio da casa dos pombos!



A = 5

IMBALANCE =
|(7+1)-5| + |(2+5)-5| + |0-5| =
3 + 2 + 5 = **10**

In this example, we already assign 3 specimens to 3 chambers, the 4th specimen must be paired with one of these 3 specimens already in their own chambers. Here, it is added to chamber 3.



A = 5

IMBALANCE =
|7-5| + |2-5| + |(5+1)-5| =
2 + 3 + 1 = **6**

Load Balancing

O princípio é que a solução deste problema pode ser simplificada pela ordenação e adição de elementos: se $S < 2C$, adicione $2C - S$ amostras *dummy*, com massa zero.

Por exemplo, $C = 3$, $S = 4$, $M = \{5, 1, 2, 7\}$ é transformado em $C = 3$, $S = 6$, $M = \{5, 1, 2, 7, 0, 0\}$.

Depois, ordenamos as amostras de acordo com a massa: $\{5, 1, 2, 7, 0, 0\} \rightarrow \{0, 0, 1, 2, 5, 7\}$.

Com isto, o critério guloso se torna aparente:

- Emparelhar as amostras com massas M_1 e M_{2C}, colocando-os no tubo 1;
- Emparelhar as amostras com massas M_2 e M_{2C-1}, colocando-os no tubo 2...

Dúvidas?