UFSC – Campus Araranguá
Departamento de Computação
DEC7536 – Projeto e Análise Algoritmos

Algoritmos Gulosos

Prof. Antonio Carlos Sobieranski

2019 / 1

Objetivos da Aula

Objetivo Geral:

- Compreender as principais características dos **Algoritmos Gulosos** no contexto de Projeto de Algoritmos.

Objetivos Específicos:

- Apresentar os conceitos fundamentais e componentes básicos dos algoritmos gulosos.
- Apresentar exemplos de aplicação onde são encontrados resultados ótimos para esta classe de algoritmos, mais especificamente para problemas em grafos e tabela de prefixos.
- Compreender as vantagens e desvantagens das abordagens gulosas.

1. Introdução

- Um Algoritmo Guloso (AG) para um **problema de otimização** sempre faz a escolha que **parece a melhor** para o **momento**, e adiciona a solução **parcial**.
- Objetivo: a **sequência** de **soluções** parciais direcionam a uma **solução** final **ótima**.
- AG's possuem uma **característica** comum:
  - Fazem a **escolha local ótima**, a **cada passo.**

    Ex.: um grafo, escolhe a aresta mais promissora em cada instante.
  - Propriedade da **escolha gulosa**, 1 solução após a outra.
  - Não consideram **planos futuros**, nem reavaliam **soluções passadas**.
  - **Mióticos**, não exploram todas as **possibilidades**.
  - Podem ser **ruins** para várias classes de **problemas**.
  - Se conseguem **fornecer** a **melhor solução**, são escolhidos pela **performance**.

2. Contextualização

- A partir de um conjunto ***C***, determinar ***S*** $\subseteq$ ***C*** tal que:
  - ***S*** satisfaça uma dada propriedade ***P***
  - ***S*** é mínimo ou máximo em relação a algum critério, que satisfaz ***P***
- Um algoritmo guloso para o **problema geral** consiste em um processo **iterativo** em que ***S*** é construído adicionando elementos de ***C***, um a um.

```
function Guloso (C: conjunto): Conjunto;
{ C: conjunto de candidatos }
begin
  S := ∅; { S contem conjunto solucao }
  while (C <> ∅) and not solucao(S) do
    begin
    x := seleciona (C);
    C := C - x;
    if viavel (S + x) then S := S + x;
    end;
  if solucao (S) then return (S) else return ('Nao existe solucao');
end;
```

- Essência do Algoritmo Guloso: **a função de escolha**.

2. Contextualização

- Exemplo: dado um **conjunto** de **moedas** de **1, 2, 5 e 10** centavos.
- Demonstrar um algoritmo guloso para uma máquina que dê o troco com o **menor número possível de moedas** (**solução ótima**)

- Troco de 18 centavos
- Algoritmo parece ser sempre ótimo para alguns sistemas monetários
- e.g.: *USD* e *Euro*

```
function Guloso (C: conjunto): Conjunto;
{ C: conjunto de candidatos }
begin
  S := ∅; { S contem conjunto solucao }
  while (C <> ∅) and not solucao(S) do
    begin
    x := seleciona (C);
    C := C − x;
    if viavel (S + x) then S := S + x;
    end;
  if solucao (S) then return (S) else return ('Nao existe solucao');
end;
```

2. Contextualização

- Vamos **modificar** ligeiramente o problema:
  - Dado um conjunto de moedas de **1, 7 e 10**.
- Algoritmo guloso falha para fornecer uma solução ótima.

- Exemplo:
- Troco de 18 centavos
- Troco de 15 centavos

```
function Guloso (C: conjunto): Conjunto;
{ C: conjunto de candidatos }
begin
  S := ∅; { S contem conjunto solucao }
  while (C <> ∅) and not solucao(S) do
    begin
    x := seleciona (C);
    C := C − x;
    if viavel (S + x) then S := S + x;
    end;
  if solucao (S) then return (S) else return ('Nao existe solucao');
end;
```

Com Programação Dinâmica, a solução é ótima !!!

2. Contextualização

- Algoritmos gulosos são em geral formados por 5 componentes:
  - Conjunto candidato ***C,*** de onde a solução ***S*** será criada;
  - ***f(x)* seleção**: seleciona o **melhor candidato** para ser adicionado a ***S***; (geralmente relacionada com a função objetivo)
  - ***f(x)* viabilidade**: usada para determinar se um **candidato** de ***C*** pode ser usado para **contribuir** com a **solução**; (possível completar o conjunto adicionando mais cadidatos de modo que 1 solução seja obtida ?)
  - ***f(x)* solução**: verifica se ***S*** é uma **solução**;
  - ***f(x)* objetiva**: fornece o **valor** da solução ***S* final** ou **parcial**.

3. Exemplos de Algoritmos Gulosos

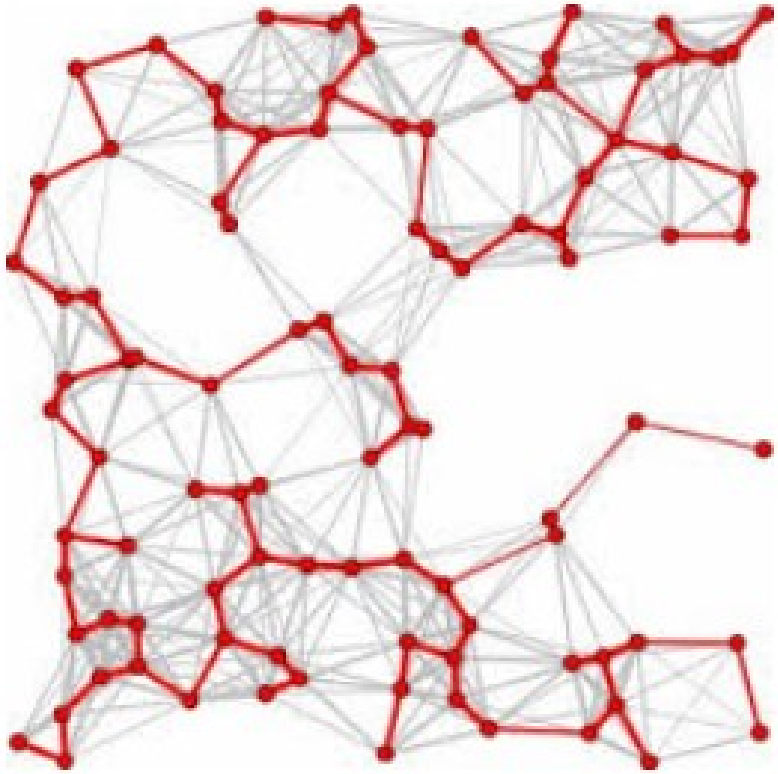
- Vamos analisar algumas estratégias gulosas para o problemas abaixo:
  - **Árvore geradora mínima (AGM)**: algoritmos geralmente eficientes em O(A log V)

    Conecta todos os vértices **sem excesso** e sem **ciclos**
    - Algoritmo guloso genérico AGM
    - PRIM – faz crescer uma árvore até se tornar geradora
    - Kruskal – faz crescer uma floresta até se tornar árvore

  - **Codificação e tabela de prefixos**
    - Algoritmo de Huffmann

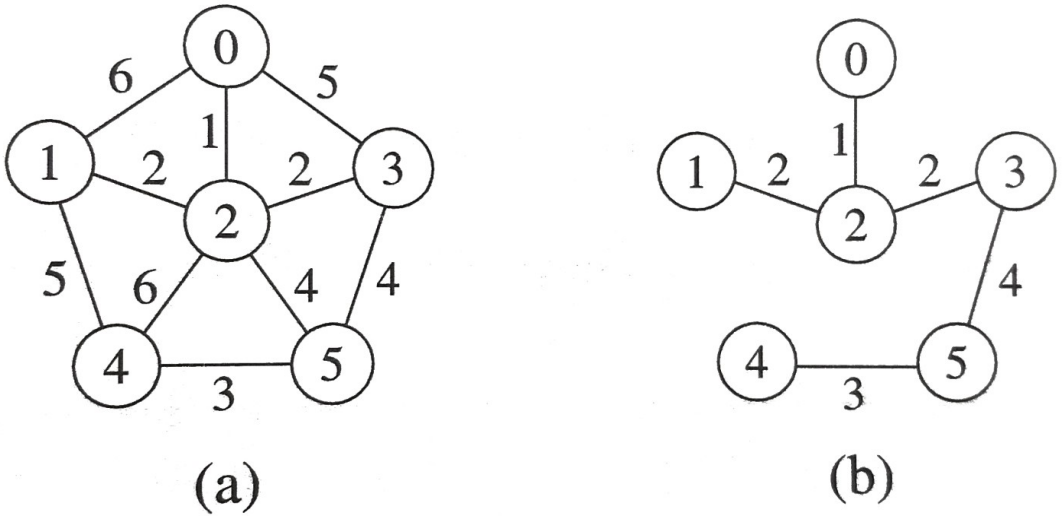
3.1. Árvore Geradora Mínima – Problema Geral

- Problema da Árvore Geradora Mínima
  - Dado um grafo ***G(V, A)*** não-dirigido com pesos, com $w_{i,j}$ denotando o peso da aresta entre ***i*** e ***j***.



3.1. Árvore Geradora Mínima – Problema Geral

- Problema da Árvore Geradora Mínima
  - Árvore geradora é um sub-grafo $G'(V, A' \subseteq A)$ de G, tal que $\forall v_i, v_j \in V$ existe um caminho de v_i, v_j, e **G'** é acíclico.
  - Árvore geradora mínima: é uma árvore geradora G' com **peso mínimo** e total de arestas de comprimento $|V| - 1$.



Remover qualquer aresta de G' torna o grafo desconexo, adicionar qualquer aresta torna um ciclo

3.1. Árvore Geradora Mínima – Problema Geral

- Árvore Geradora Mínima

Algoritmo GenéricoAGM

Entrada: um conjunto C de arestas A conectando os vértices v_i e v_j

Saída: uma árvore geradora de peso mínimo

$S = \varnothing$

while S não constitui uma árvore geradora mínima **do**

    (u,v) = Seleciona(C)

    if aresta (u,v) é segura para S **then**

        $S = S \cup \{(u,v)\}$

return S

3.2. Algoritmo de PRIM

- Algoritmo de PRIM para árvore geradora mínima
  - Pode ser **derivado** ao *AGMGenérico* anterior

Algoritmo GenéricoAGM

Entrada: um conjunto C de arestas A conectando os vértices v_i e v_j

Saída: uma árvore geradora de peso mínimo

$S = \varnothing$

while S não constitui uma árvore geradora mínima **do**

  (u,v) = Seleciona(C)

  if aresta (u,v) é segura para S **then**

    $S = S \cup \{(u,v)\}$

return S

  - Sempre seleciona a aresta de menor peso da árvore corrente para o resto do grafo → a aresta mais barata que cruza um corte.

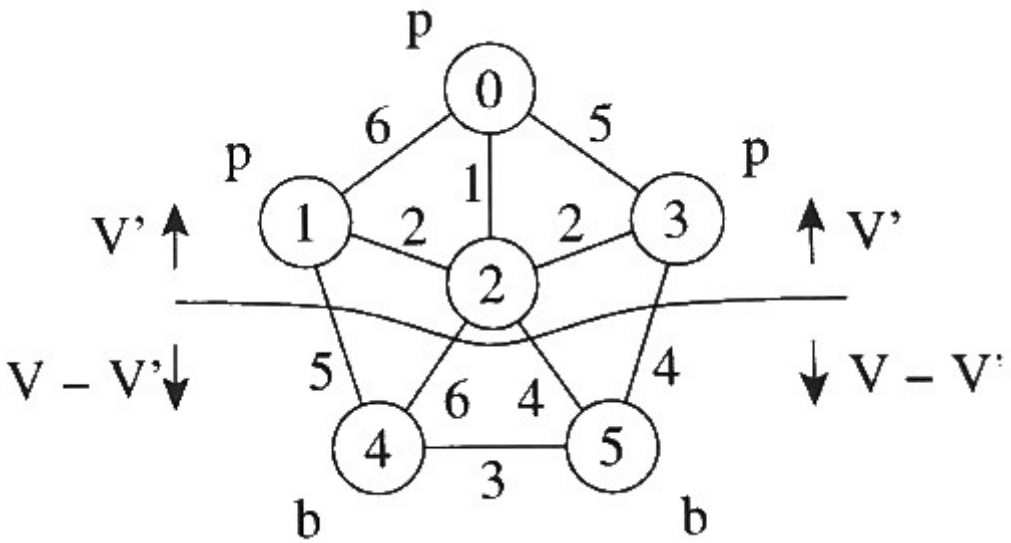
3.2. Algoritmo de PRIM

- Algoritmo de PRIM para árvore geradora mínima
- Conceito de **aresta segura**:

Teorema: Seja ***G=(V, A)*** um grafo conexo, não direcionado e com pesos ***p*** sobre as arestas ***A***.

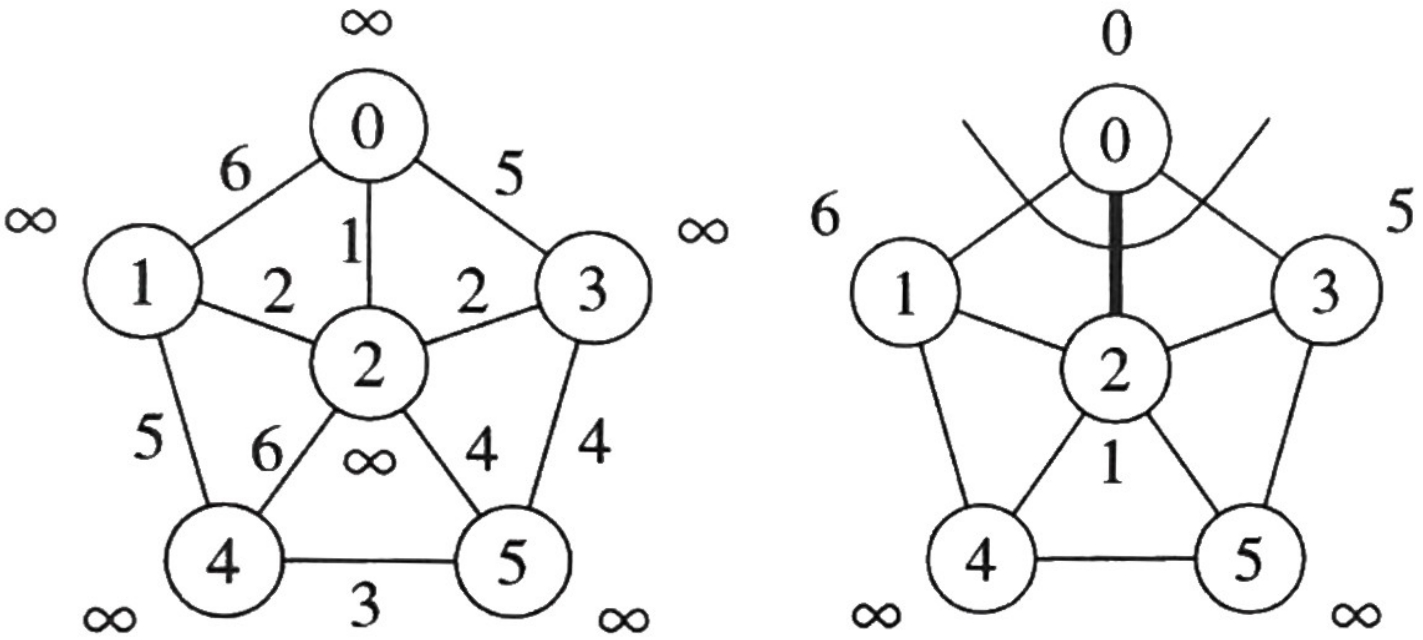
Seja ***S*** um subconjunto de ***V*** que está incluído em alguma árvore geradora mínima para ***G***, e seja ***(V', V–V')*** um corte qualquer que respeita ***S***, e seja ***(u,v)*** uma aresta leve cruzando ***(V', V–V')***. Logo, a aresta ***(u,v)*** é uma aresta segura para ***S***.

Aresta leve => Propriedade Gulosa



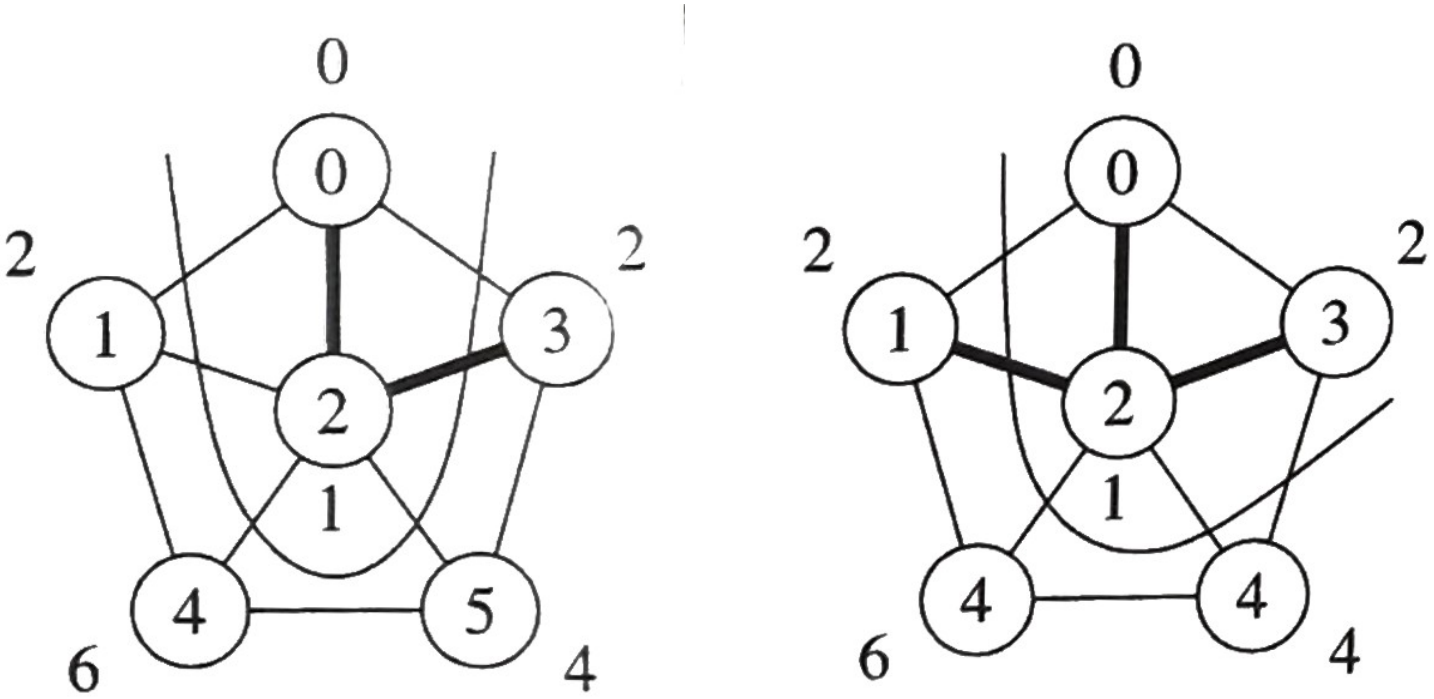
3.2. Algoritmo de PRIM

- Algoritmo de PRIM para árvore geradora mínima



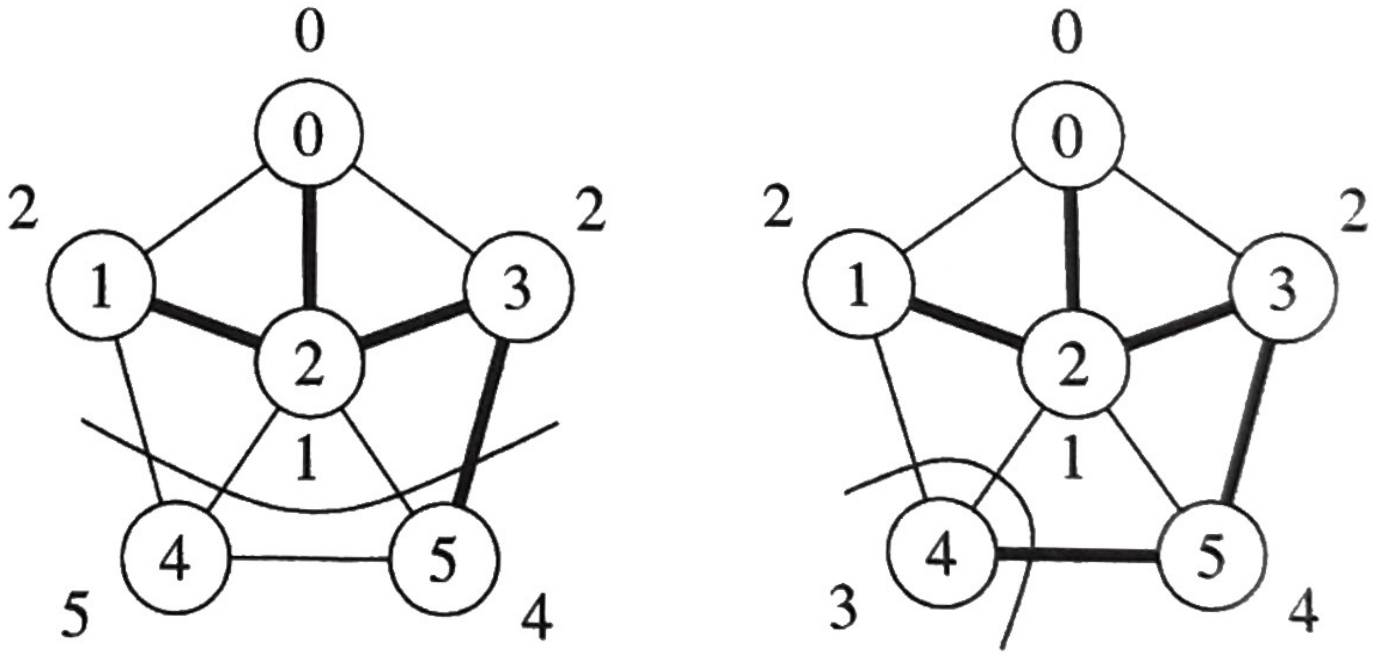
3.2. Algoritmo de PRIM

- Algoritmo de PRIM para árvore geradora mínima
  - Algoritmo de PRIM



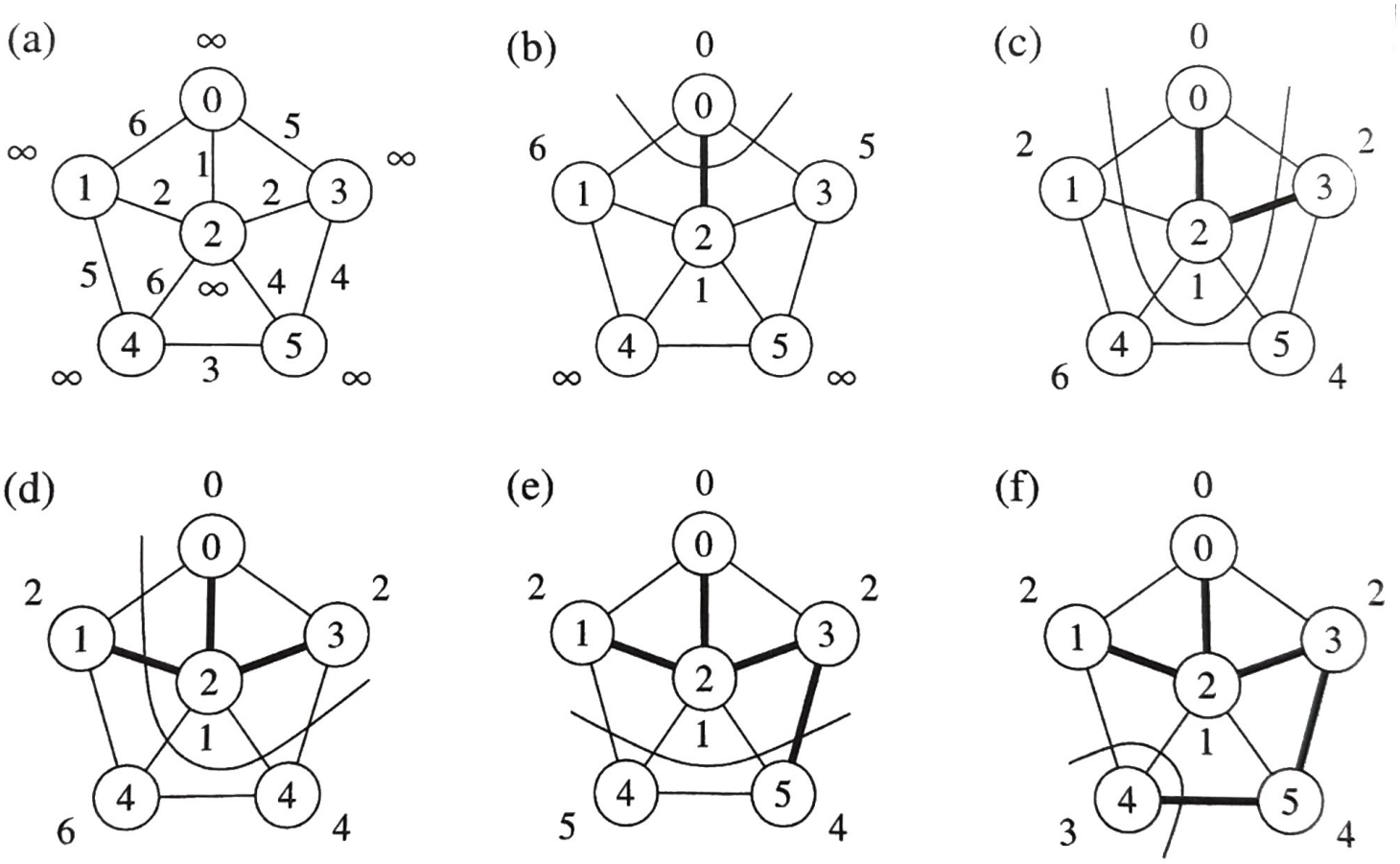
3.2. Algoritmo de PRIM

- Algoritmo de PRIM para árvore geradora mínima
  - Algoritmo de PRIM



3.2. Algoritmo de PRIM

- Algoritmo de PRIM para árvore geradora mínima
  - Algoritmo de PRIM



3.3. Algoritmo de Kruskal

- Algoritmo de **Kruskal** para árvore geradora mínima
  - Pode ser derivado do algoritmo **AGMGenérico**, com outro conceito de aresta segura.

Algoritmo GenéricoAGM

Entrada: um conjunto C de arestas A conectando os vértices v_i e v_j

Saída: uma árvore geradora de peso mínimo

$S = \varnothing$

while S não constitui uma árvore geradora mínima **do**

    (u,v) = Seleciona(C)

    if aresta (u,v) é segura para S **then**

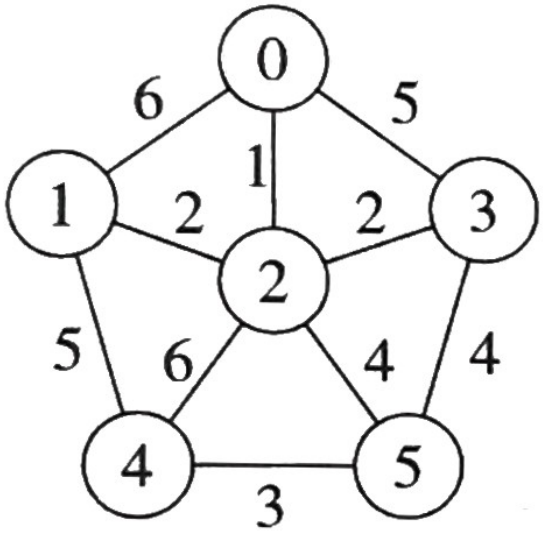
        $S = S \cup \{(u,v)\}$

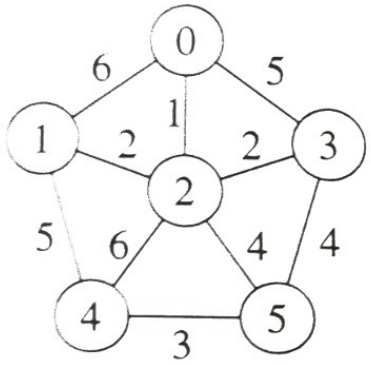
return S

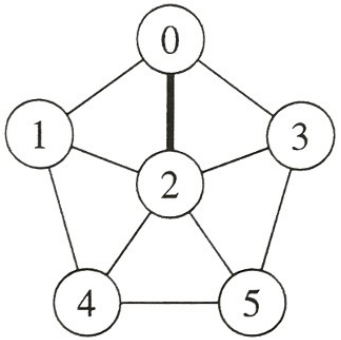
  - Estratégia simples:
    - ***S*** é uma floresta. O algoritmo inicia com uma floresta de **|V|** árvores de **1** vértice: em **|V|** passos, une 2 árvores até que exista apenas 1 árvore na floresta.
    - Adiciona uma **aresta de menor peso** a ***S*** que **não** forma um **ciclo**.

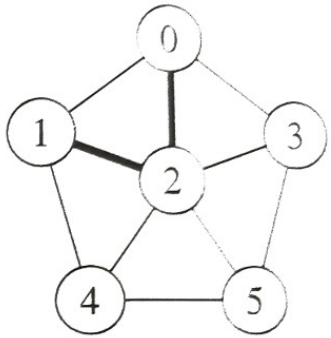
3.3. Algoritmo de Kruskal

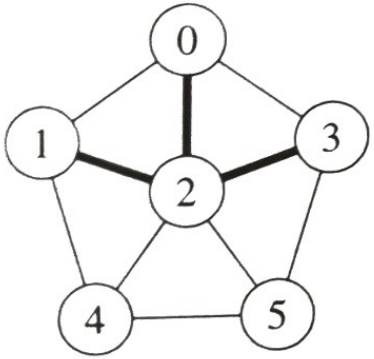
- Algoritmo de Kruskal para árvore geradora mínima

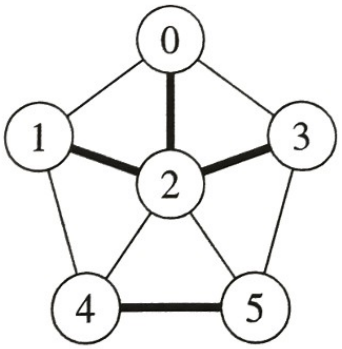


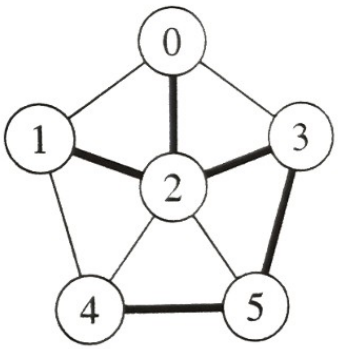






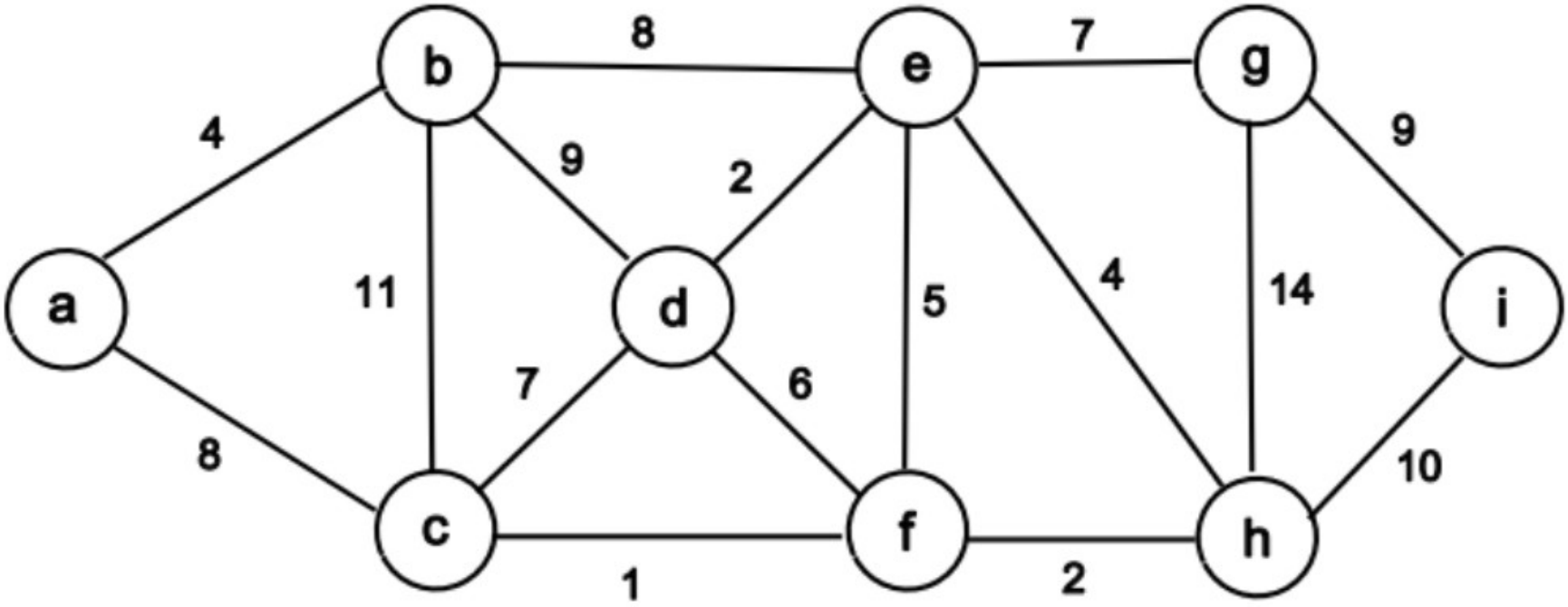






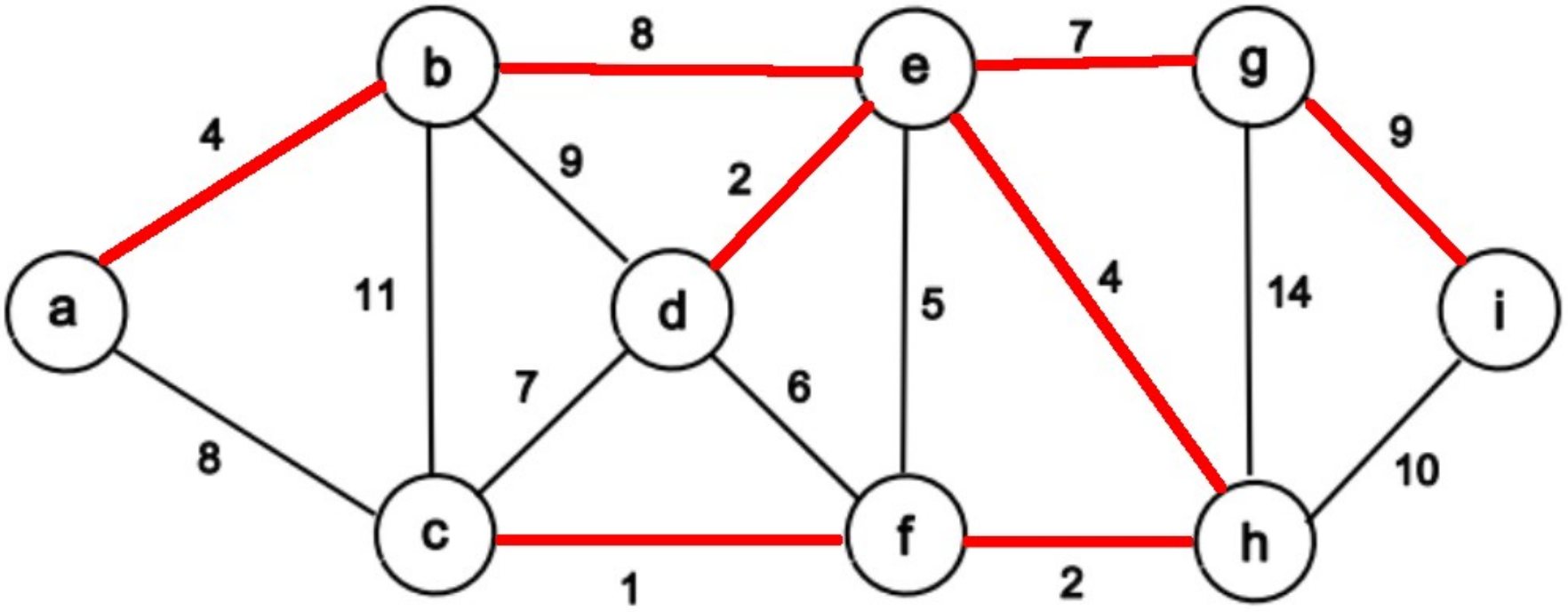
3.3. Algoritmo de Kruskal

- Algoritmo de Kruskal para árvore geradora mínima



3.3. Algoritmo de Kruskal

- Algoritmo de Kruskal para árvore geradora mínima



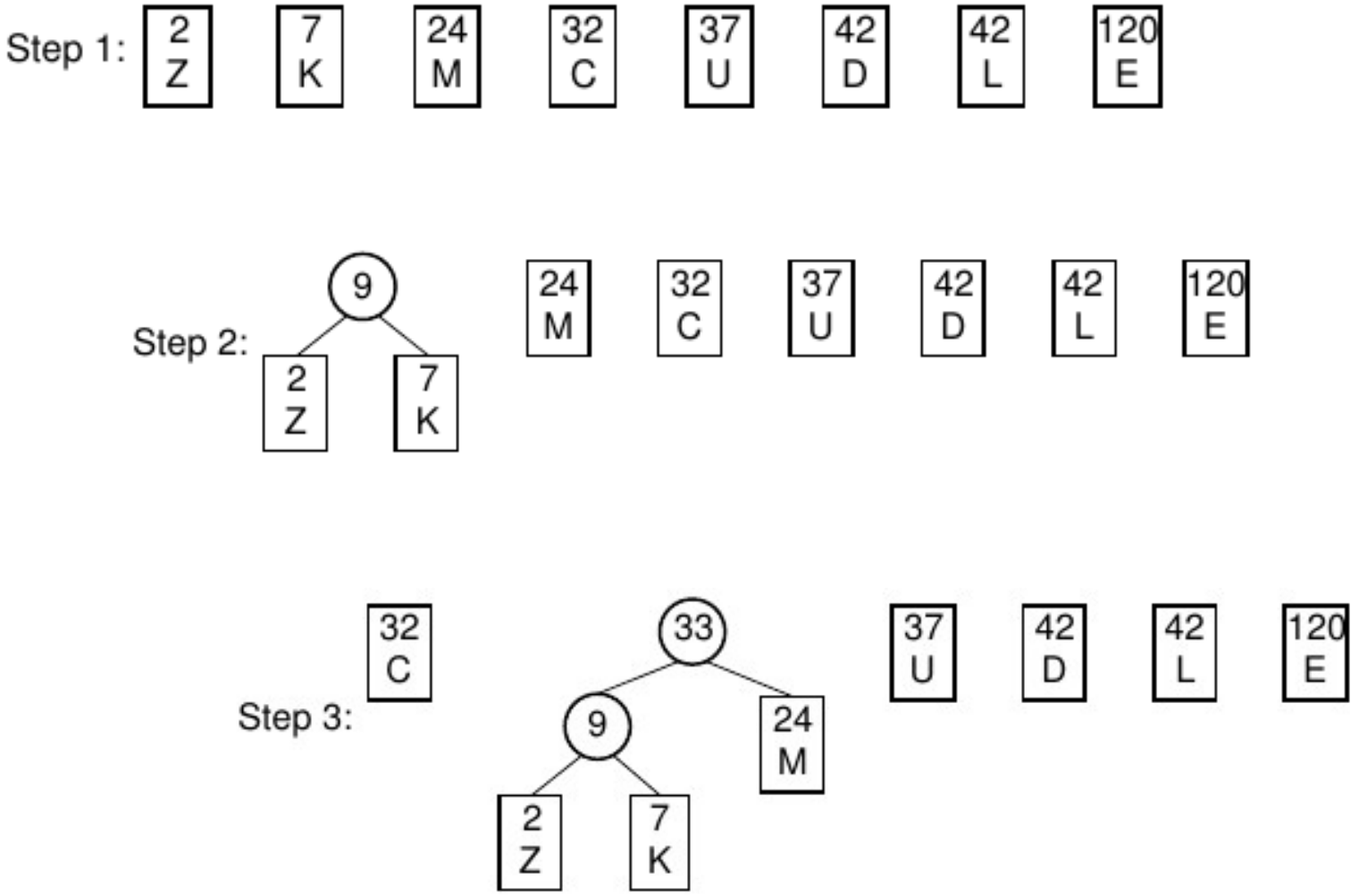
3.4. Codificação de Huffman

- Codificação de Huffman → **método guloso** para produzir códigos de prefixo
  - David Huffman, 1952, MIT
  - Método para criar prefixos, utilizado para compressão de dados sem perda.
  - **Objetivo** é codificar mensagens em um alfabeto de ***n*-caracteres** de modo que a mensagem codificada seja a menor possível (comprimento)
  - **Premissa básica gulosa**: agrupar 2 letras menos frequentes (***x*** e ***y***) como folhas em uma árvore, agrupando-as em um código composto (***xy***), e iniciar recursão.

1952 – A Method for the Construction of Minimum-Redundancy Codes

3.4. Codificação de Huffman

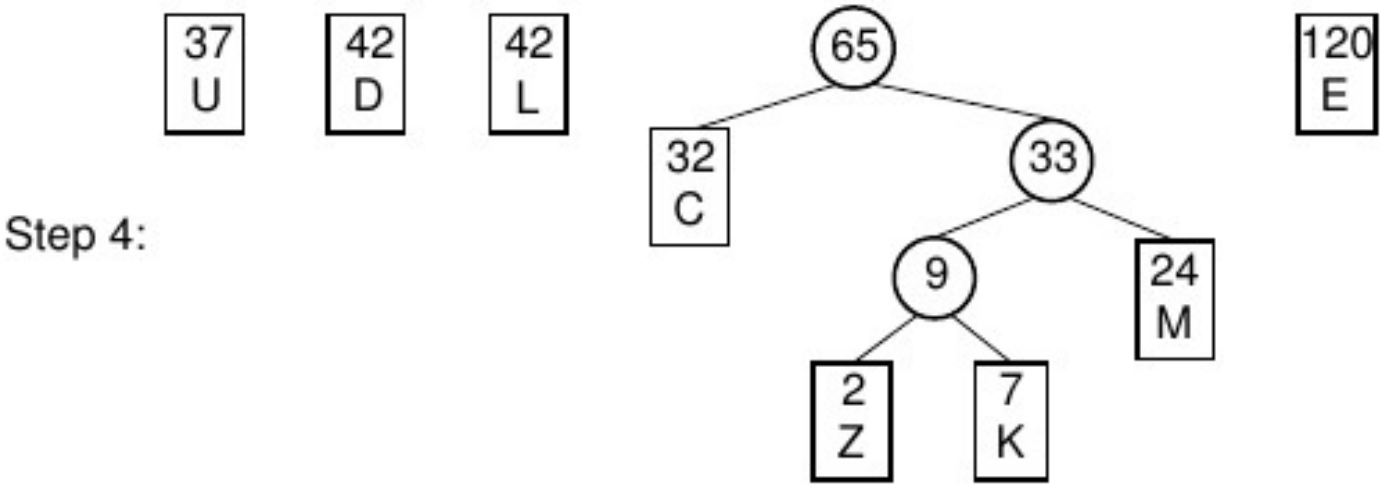
- Exemplo:



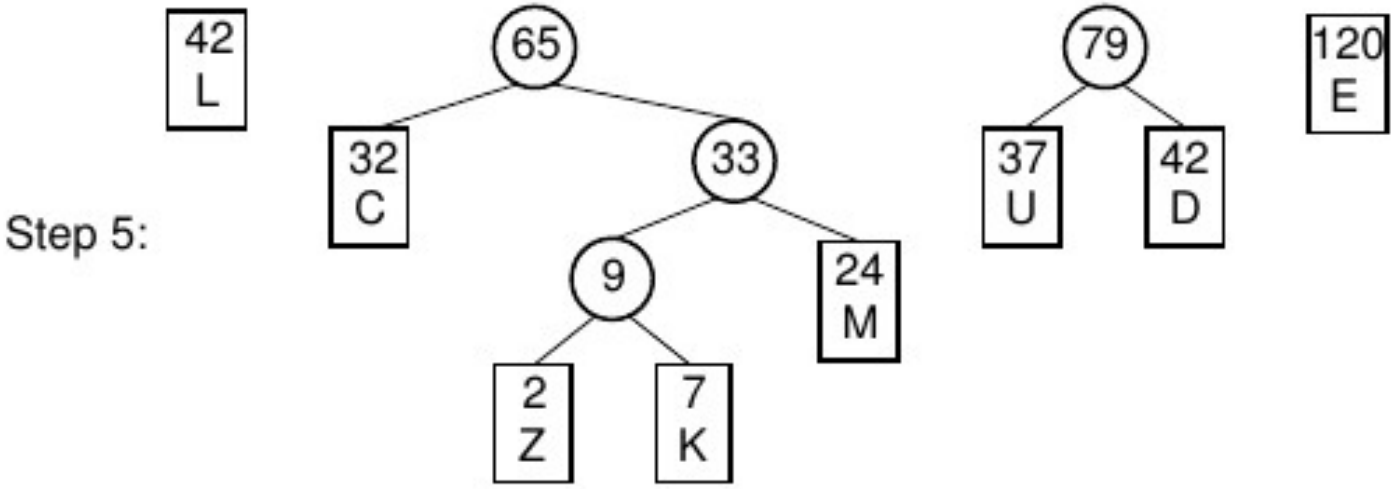
3.4. Codificação de Huffman

- Exemplo:

Step 4:

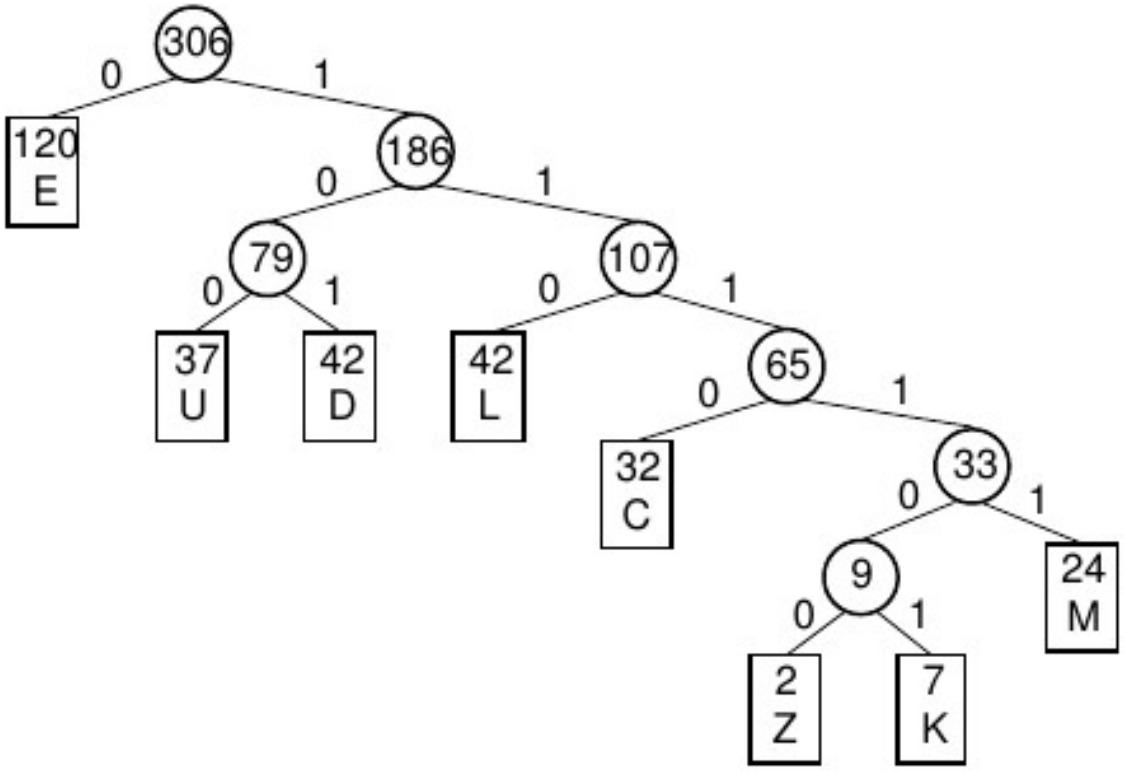


Step 5:



3.4. Codificação de Huffman

- Exemplo:



- Saída do Algoritmo: Tabela de símbolos, 0 à esquerda, 1 à direita.
- Símbolos obtidos pela varredura da árvore.

4. Vantagens e Desvantagens da Estratégia Gulosa

- Algoritmos gulosos apresentam uma série de **vantagens** algorítmicas:
  - **Simplicidade**: frequentemente fáceis de descrever;
  - **Eficiência**: frequentemente podem ser implementados mais eficientemente;
  - Se solucionam o problema (ótimo), são escolhidos por serem **rápidos**.
- Algoritmos gulosos apresentam **limitações**:
  - Projetar uma abordagem gulosa pode ser fácil. Projetar a abordagem **correta** pode ser difícil;
  - Pode não somente falhar, como trazer a pior solução possível (PCV);
  - Escolhas muito precoces, impedem de procurar ótimos globais;
  - Algoritmos gulosos nem sempre atingem soluções ótimas, mas são escolhidos quando.
- Algoritmos gulosos são excelentes para vários problemas em grafos.

4. Vantagens e Desvantagens da Estratégia Gulosa

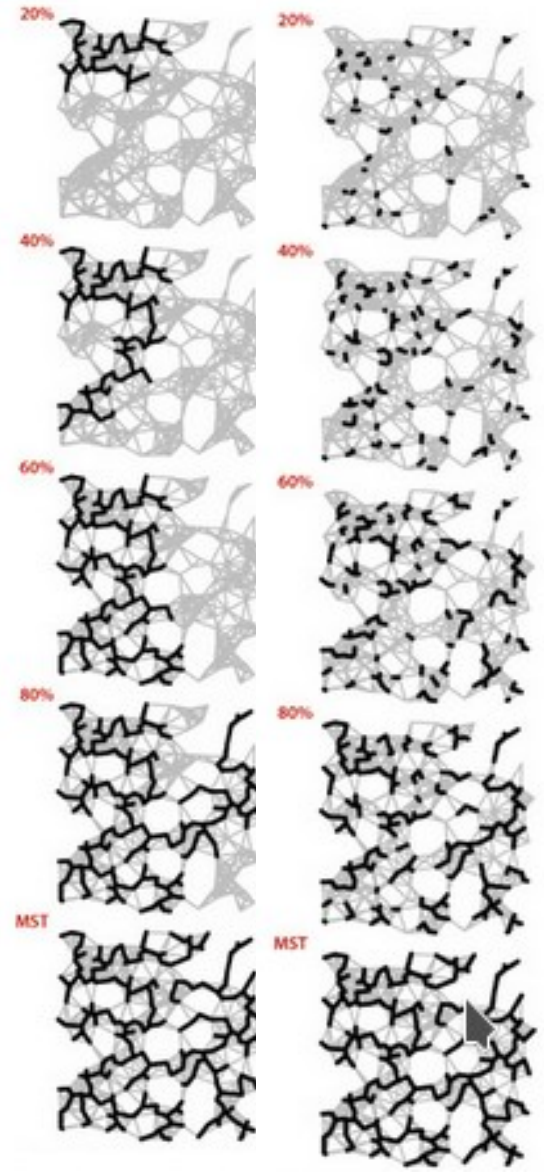
Casos onde a estratégia gulosa funciona ?

- Quando o problema apresentar a **propriedade da escolha gulosa**, ou seja, depender de soluções ótimas locais.
- **2 provas:** AG permanece à frente, transformação gradual.

Algoritmos Gulosos

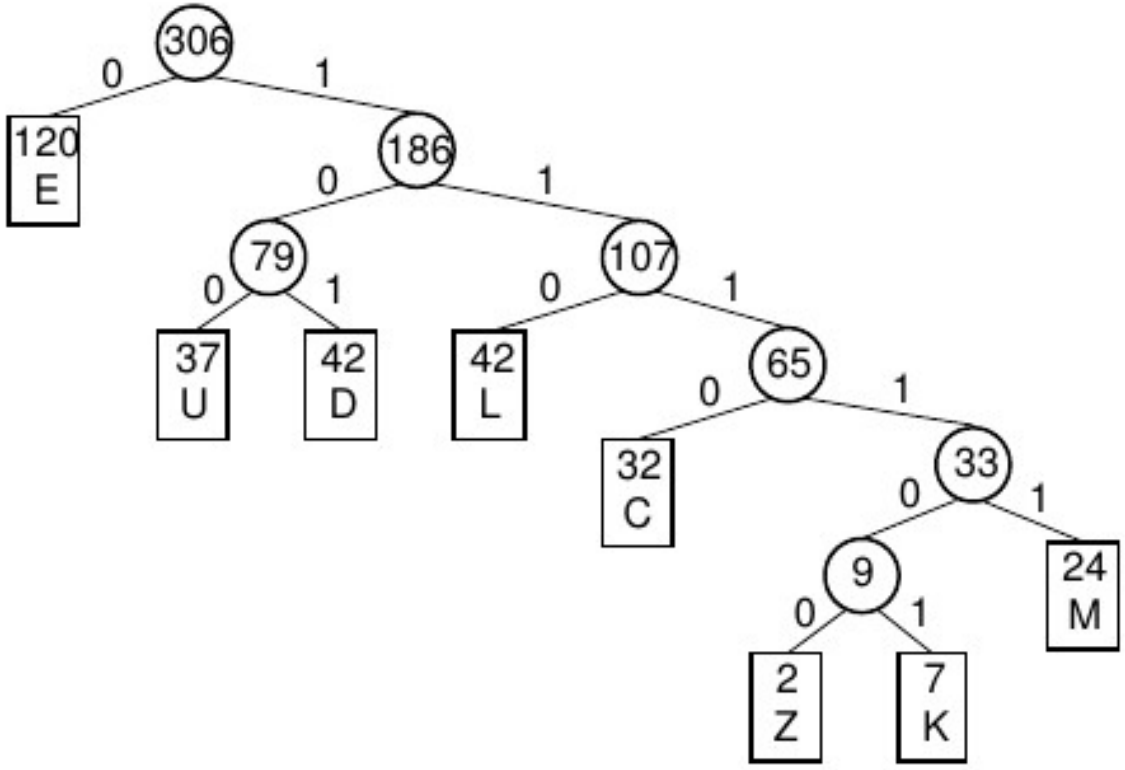
Síntese da Aula

- Conceitos de Algoritmo Gulosos.
- PRIM : faz crescer uma árvore até que ela se torne geradora
- Kruskal : faz crescer uma floresta até que ela se torne árvore.
- Ambos tem a característica Gulosa: a cada iteração "**abocanham**" a aresta que parece mais promissora.
- Huffmann: genial pela sua simplicidade.



Início do trabalho a ser entregue

- Codificação por tabela de prefixos – Huffman



Referências Bibliográficas

- **Cormen**, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford. **Introduction to Algorithms (3rd ed.)**. Massachusetts Institute of Technology. pp. 253–280. ISBN 978-0-262-03384-8. 2009.
- **Ziviani,** Nivio. **Projeto de Algoritmos com Implementações em Pascal e C**, Segunda Edição. Editora Thomson, 2004.