Algoritmos e Estrutura de Dados II (AE23CP-3CP)

Aula #04 - Paradigmas de Projeto de Algoritmos
Método Guloso

Profª Luciene de Oliveira Marin
lucienemarin@utfpr.edu.br

Paradigma de Projeto de Algoritmos

Método Guloso

Considerações

- Polinomial $\times$ Exponencial (não polinomial);
- Tratável $\times$ Intratável
- P $\times$ NP (NP-Completo, NP-Difícil)
- Algoritmos Recursivos $\times$ Não-Recursivos

Exemplo: Sequência de Fibonacci

Para projetar um algoritmo eficiente, é fundamental preocupar-se com a sua complexidade. Como exemplo: considere a **sequência de Fibonacci**

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

A sequência pode ser definida recursivamente:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{if } n > 1 \end{cases}$$

Dado o valor de n, queremos obter o n-ésimo elemento da sequência.
Vamos apresentar dois algoritmos e analisar sua complexidade.

Exemplo: Sequência de Fibonacci - Algoritmo 1: função $fibo1(n)$

Seja a função $fibo1(n)$ que calcula o n-ésimo elemento da sequência de Fibonacci.

Entrada: Valor de n
Saída: O n-ésimo elemento da sequência de Fibonacci

```
function fibo1(n))
    if n = 0 then
        return 0
    else if n = 1 then
        return 1
    else
        return fibo1(n − 1) + fibo1(n − 2)
    end if
end function
```

Experimente rodar este algoritmo para $n = 100$

A complexidade é $O(2^n)$.

(Mesmo se uma operação levasse um picosegundo, 1×10^{-12}s), 2^{100} operações levariam 3×10^{13} anos =30.000.000.000.000 anos.)

Exemplo: Sequência de Fibonacci - Algoritmo 1: função $fibo1(n)$

Seja a função $fibo1(n)$ que calcula o n-ésimo elemento da sequência de Fibonacci.

Entrada: Valor de n
Saída: O n-ésimo elemento da sequência de Fibonacci

```
function fibo1(n))
    if n = 0 then
        return 0
    else if n = 1 then
        return 1
    else
        return fibo1(n − 1) + fibo1(n − 2)
    end if
end function
```

Experimente rodar este algoritmo para $n = 100$
A complexidade é $O(2^n)$.
(Mesmo se uma operação levasse um picosegundo, 1×10^{-12}s), 2^{100} operações levariam 3×10^{13} anos $=$**30.000.000.000.000 anos**.)

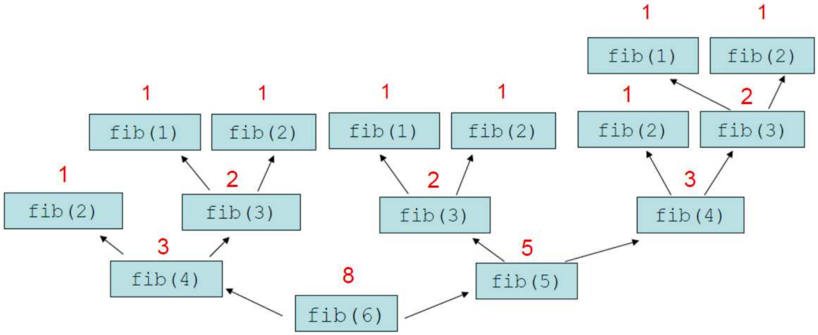
Seja a função $fibo1(n)$ que calcula o n-ésimo elemento da sequência de Fibonacci.

Entrada: Valor de n
Saída: O n-ésimo elemento da sequência de Fibonacci

```
function fibo1(n))
    if n = 0 then
        return 0
    else if n = 1 then
        return 1
    else
        return fibo1(n − 1) + fibo1(n − 2)
    end if
end function
```

Experimente rodar este algoritmo para $n = 100$
A complexidade é $O(2^n)$.
(Mesmo se uma operação levasse um picosegundo, 1×10^{-12}s), 2^{100} operações levariam 3×10^{13} anos =**30.000.000.000.000 anos**.)

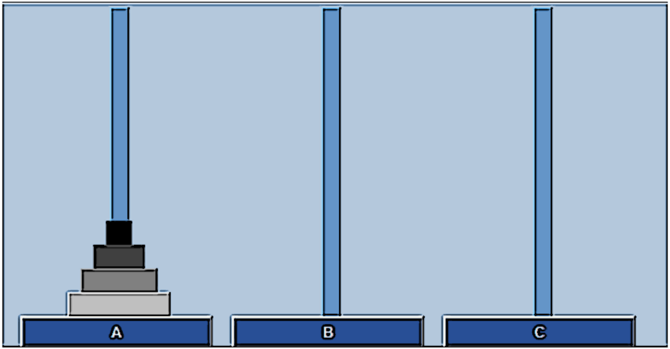
Exemplo: pilha de recursão para fibo1(6)



Exemplo: Sequência de Fibonacci - Algoritmo 2: função *fibo*2(n)

```
function fibo2(n))
    if n = 0 then
        return 0
    else if n = 1 then
        return 1
    else
        penultimo ← 0
        ultimo ← 1
        for i ← 2 to n do
            atual ← penultimo + ultimo
            penultimo ← ultimo
            ultimo ← atual
        end for
        return atual
    end if
end function
```

A complexidade agora passou de $O(2^n)$ para $O(n)$
Você sabe que dá para fazer em $O(log\ n)$?


A
B
C

Exemplo: Torres de Hanoi

Algoritmo 1 $Hanoi(N, Orig, Dest, Temp, contador)$

```
if N = 1 then
    mover o menor disco do pino Orig para o pino Dest;
    incrementar o contador
else
    Hanoi(N − 1, Orig, Temp, Dest, contador);
    mover o N-ésimo menor disco do pino Orig para o pino Dest;
    incrementar o contador
    Hanoi(N − 1, Temp, Dest, Orig, contador);
end if
```

Exemplo: Torres de Hanoi

Complexidade do algoritmo $Hanoi(N, Orig, Dest, Temp, contador)$:

O número mínimo de "movimentos" para conseguir transferir todos os discos da primeira estaca à terceira é 2^{n-1}, sendo n o número de discos. Logo: $O(2^n)$

- Para solucionar um Hanói de 4 discos, são necessários 15 movimentos
- Para solucionar um Hanói de 7 discos, são necessários 127 movimentos
- Para solucionar um Hanói de 15 discos, são necessários 32.767 movimentos
- Para solucionar um Hanói de 64 discos, como diz a lenda, são necessários 18.446.744.073.709.551.615 movimentos.

Problema do Caixeiro Viajante PCV (Traveling Sallesman Problem - TSP)

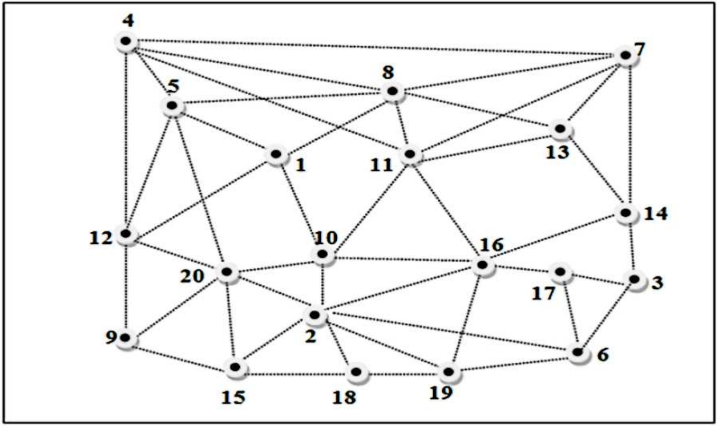


Figura 1 - Instância do PCV para 20 cidades

Problema do Caixeiro Viajante PCV

Resolvendo o PCV pelo método da busca exaustiva:

Espaço de busca para o PCV é um conjunto de permutações das n cidades.

- Cada permutação das n cidades caracteriza-se como uma lista ordenada que define a seqüência das cidades a serem visitadas. A solução ótima é uma permutação que corresponda a uma tour (ou passeio) de caminho mínimo.
- Cada *tour* pode ser representada de $2n$ maneiras diferentes (para um modelo simétrico).
- Considerando-se que há $n!$ formas de permutar n números, o tamanho do espaço de busca é então $|S| = \dfrac{n!}{(2n)} = \dfrac{(n-1)!}{2}$
- Logo, complexidade $O(n!)$

Para se avaliar a taxa de crescimento desta expressão, para um PCV de:

- 10 cidades há 181.000 caminhos possíveis
- 20 cidades há aproximadamente cerca de 10.000.000.000.000.000 caminhos possíveis[a]

[a] Michalewics & Fogel. 2004. Modern Heuristics. How to solve it.

Métodos de Projetos de Algoritmos

- Método Guloso
- Divisão-e-Conquista
- Programação Dinâmica
- *Backtracking*
- *Branch-and-bound*

Método Guloso

Idéias Básicas da Estratégia Gulosa

Idéia básica da estratégia gulosa:

Construir por etapas uma resposta ótima. Em cada passo:

- selecionar um elemento da entrada (o melhor localmente);
- decidir se ele é viável - vindo a fazer parte da resposta - ou não.

Após uma seqüência de decisões, uma solução para o problema é alcançada.

Nessa seqüência de decisões, nenhum elemento é examinado mais de uma vez: ou ele fará parte da saída, ou será descartado.

Exemplo 1: Cálculo do Troco

- **Descrição:** Seja $E = \{e_1, e_2, ..., e_n\}$, $e_1 > e_2 > ... > e_n$, um conjunto de n denominações de moedas (ou cédulas), e M um valor positivo que representa o troco.
- **Problema:** Fornecer o montante M com o menor número de moedas.
- **Seqüência de decisões:** Escolher r_1, depois r_2, . . .

Exemplo 1: Cálculo do Troco

Descrição: Seja $E = \{100, 50, 10, 5, 1\}$, e M um valor positivo que representa o troco.
Estratégia Gulosa:

- No passo i, escolher $r_i = j$, tal que $e_j \leq M$ e $e_{j-1} > M$ e subtrair e_j de M para o próximo passo.
- É possível provar que a estratégia gulosa funciona neste caso.
- Ex.: troco de R$450,00 $\rightarrow$ 5 moedas

- A mesma estratégia funciona para $E = \{300, 250, 100, 1\}$? 52 moedas
- Qual a melhor estratégia?

Exemplo 1: Cálculo do Troco

Descrição: Seja $E = \{100, 50, 10, 5, 1\}$, e M um valor positivo que representa o troco.

Estratégia Gulosa:

- No passo i, escolher $r_i = j$, tal que $e_j \leq M$ e $e_{j-1} > M$ e subtrair e_j de M para o próximo passo.
- É possível provar que a estratégia gulosa funciona neste caso.
- Ex.: troco de R$450,00 $\rightarrow$ 5 moedas
- A mesma estratégia funciona para $E = \{300, 250, 100, 1\}$?
  52 moedas
- Qual a melhor estratégia?

Método guloso

Exemplo 1: Cálculo do Troco

Descrição: Seja $E = \{100, 50, 10, 5, 1\}$, e M um valor positivo que representa o troco.

Estratégia Gulosa:

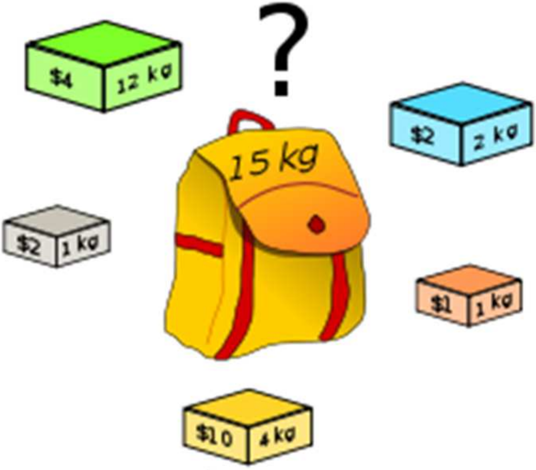
- No passo i, escolher $r_i = j$, tal que $e_j \leq M$ e $e_{j-1} > M$ e subtrair e_j de M para o próximo passo.
- É possível provar que a estratégia gulosa funciona neste caso.
- Ex.: troco de R\$450,00 $\rightarrow$ 5 moedas
- A mesma estratégia funciona para $E = \{300, 250, 100, 1\}$? 52 moedas
- Qual a melhor estratégia?

Algoritmo Guloso - Cálculo do Troco

Algoritmo 2 "Dá o troco" para n unidades usando o menor número possível de moedas

function $\mathrm{TROCO}(n)$
    const $C \leftarrow \{100, 25, 10, 5, 1\}$
    $S \leftarrow \emptyset$
    $s \leftarrow 0$
    while $s \neq n$ **do**
        $x \leftarrow$ o maior item em C tal que $s + x \leq n$
        if este item não existe **then**
            return "Não foi encontrada uma solução!"
        end if
        $S \leftarrow S \cup$ uma moeda de valor x
        $s \leftarrow s + x$
    end while
    return S
end function

O Problema da Mochila

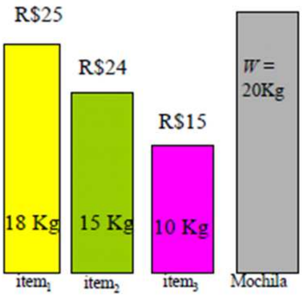


Exemplo 2: Problema da Mochila

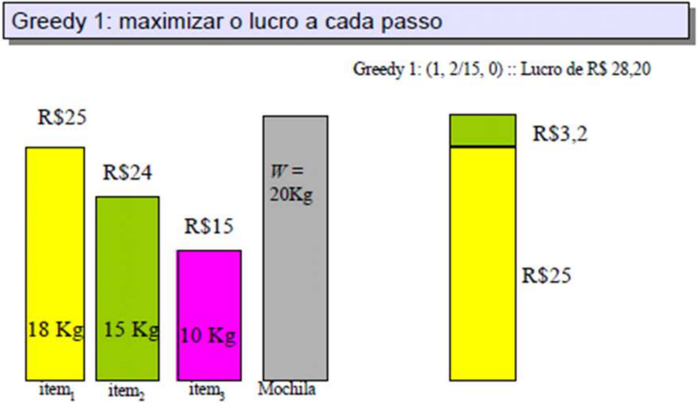
- **Descrição:** Temos n objetos com pesos $p_1, p_2, ..., p_n$ e lucros $l_1, l_2, ..., l_n$. Temos ainda uma mochila de capacidade M. Se uma fração $x_i (0 \leq x_i \leq 1)$ do objeto i for colocada na mochila, resulta em um lucro $x_i \times l_i$.
- **Problema:** Maximizar o lucro que pode ser levado na mochila.
- **Sequencia de decisões:** Escolher primeiro objeto, escolher segundo objeto, etc.

O Problema da Mochila Fracionária

S = {($item_1$, 18, 25), ($item_2$, 15, 24), ($item_3$, 10, 15)} e W= 20

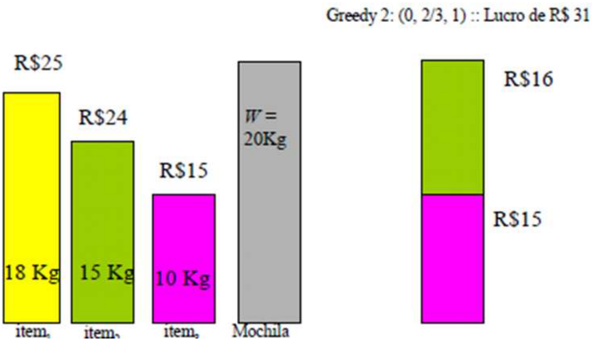


O Problema da Mochila Fracionária



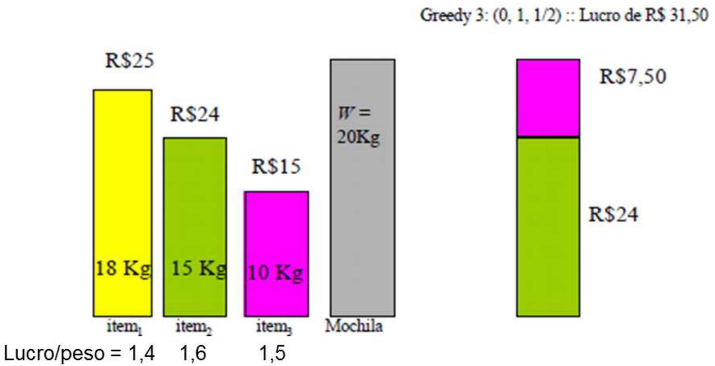
O Problema da Mochila Fracionária

Greedy 2: conservar a capacidade



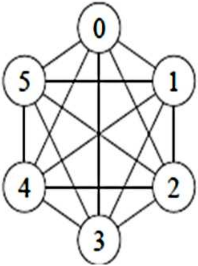
O Problema da Mochila Fracionária

Greedy 3: maximiza lucro por unidade de peso



Problema do Caixeiro Viajante

Seja uma instância do PCV com 6 cidades e suas respectivas distâncias entre elas:



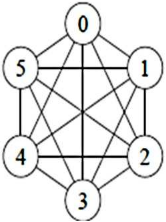
	1	2	3	4	5
0	3	10	11	7	25
1		8	12	9	26
2			9	4	20
3				5	15
4					18

Problema do Caixeiro Viajante

Caminho ótimo para esta instância: 0 1 2 5 3 4 0 (comprimento 58).

Para a heurística do vizinho mais próximo:

- Se iniciarmos pelo vértice 0, o vértice mais próximo é o 1 com distância 3. A partir do 1, o mais próximo é o 2, a partir do 2 o mais próximo é o 4, a partir do 4 o mais próximo é o 3, a partir do 3 restam o 5 e o 0.
- O comprimento do caminho 0 1 2 4 3 5 0 é 60.



	1	2	3	4	5
0	3	10	11	7	25
1		8	12	9	26
2			9	4	20
3				5	15
4					18

Problema da Árvore Geradora Mínima

- Uma árvore geradora (ou de espalhamento) de um grafo G é um subgrafo acíclico (i. e., sem ciclos) que contém todos os vértices (nodos) do grafo.
- Se o grafo não é conexo, cada componente terá uma árvore geradora.
- Se as arestas contém custos e, além do mínimo de arestas queremos as arestas de custo mínimo temos um **Árvore Geradora Mínima**.
- É um problema de agrupamento encontrado em redes, por exemplo.

Método guloso

Problema da Árvore Geradora Mínima

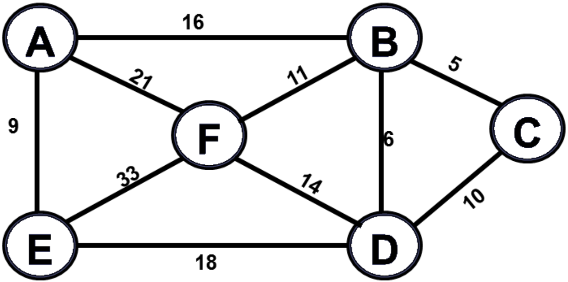
Uma estratégia sugerida pelo método guloso para determinar uma árvore geradora de custo mínimo é a seguinte:

- As arestas serão consideradas em ordem não decrescente de seus custos, e vamos construir uma floresta de árvores.
- A cada passo, seleciona-se uma aresta de custo mínimo dentre as ainda não examinadas.
- Se sua inclusão criar um ciclo, ela é descartada; caso contrário, ela é incluída (talvez ligando duas árvores da floresta).

Exemplo. Árvore geradora mínima de grafo não dirigido

Consideremos um grafo não dirigido $G =< V, E >$ com:

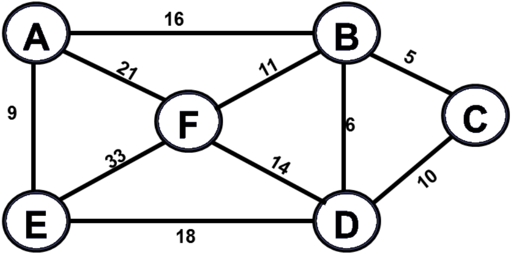
- 6 vértices: $V = \{a, b, c, d, e, f\}$;
- 10 arestas: $E = \{(a, b), (a, e), (a, f), (b, c), (b, d), (b, f), (c, d), (d, e), (d, f), (e, f)\}$;



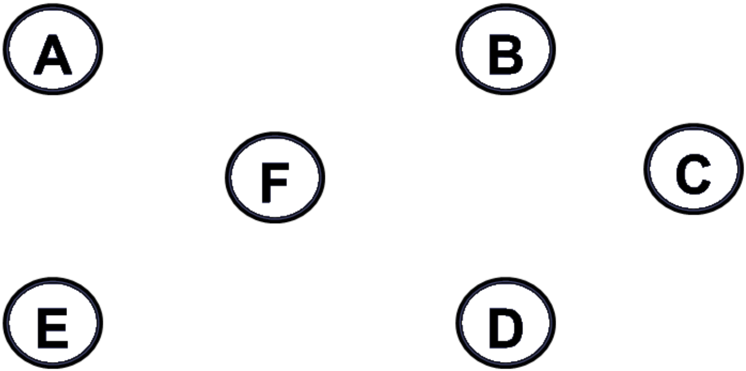
$E = \{(a,b),(a,e),(a,f),(b,c),(b,d),(b,f),(c,d),(d,e),$
$(d,f),(e,f)\};$

Custos (ou pesos):

$custo(a,b) = 16,\ custo(a,e) = 9,\ custo(a,f) = 21,$
$custo(b,c) = 5,\ custo(b,d) = 6,\ custo(b,f) = 11,$
$custo(c,d) = 10,\ custo(d,e) = 18,\ custo(d,f) = 14,$
$custo(e,f) = 33.$

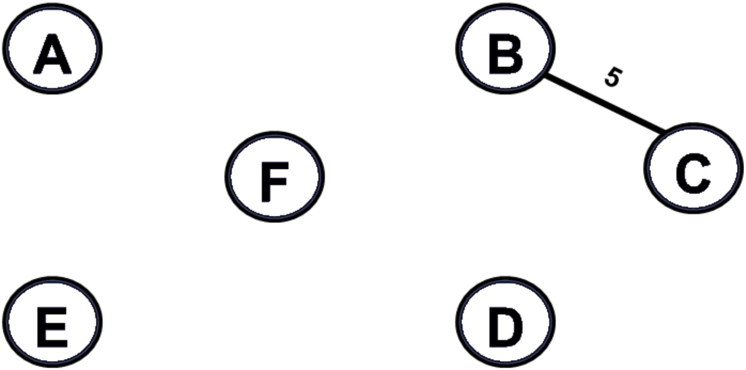


Problema da Árvore Geradora Mínima

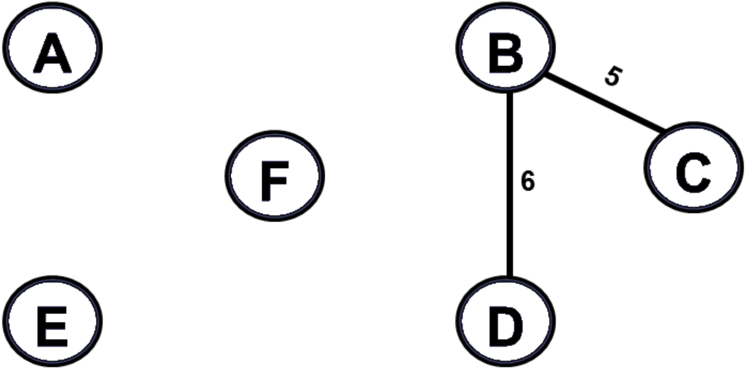


$$C = \{A, E, B, C, D, F\}$$

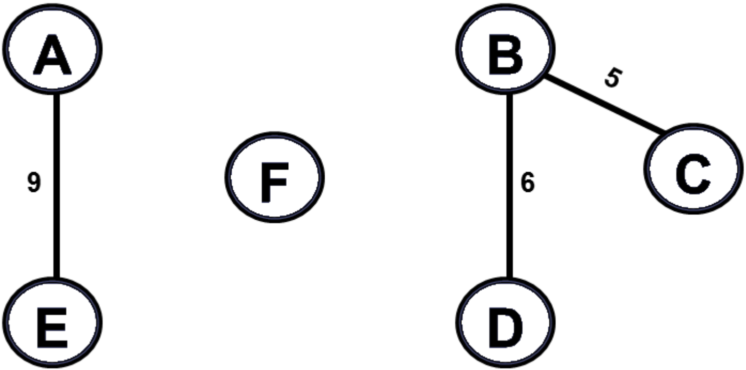
Problema da Árvore Geradora Mínima



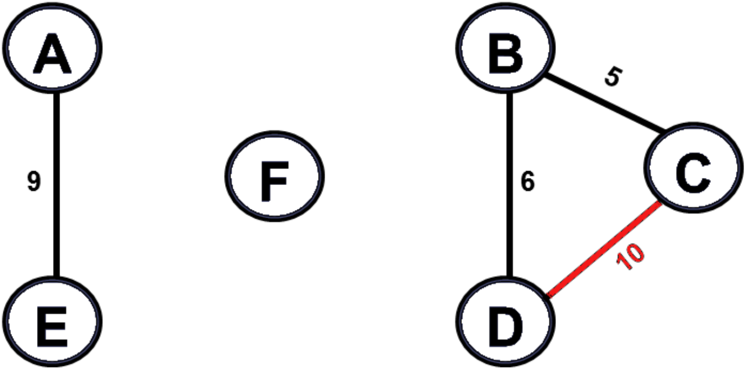
$$C = \{A, E, B, C, D, F\}$$



$$C = \{A, E, B, C, D, F\}$$

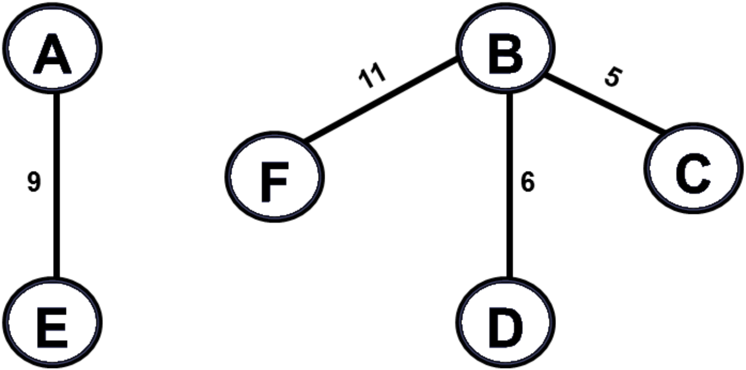


$C = \{A, E, B, C, D, F\}$

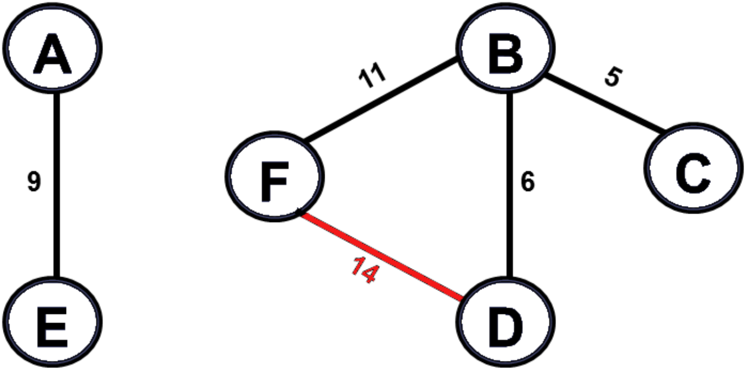


$C = \{A, E, B, C, D, F\}$

Problema da Árvore Geradora Mínima

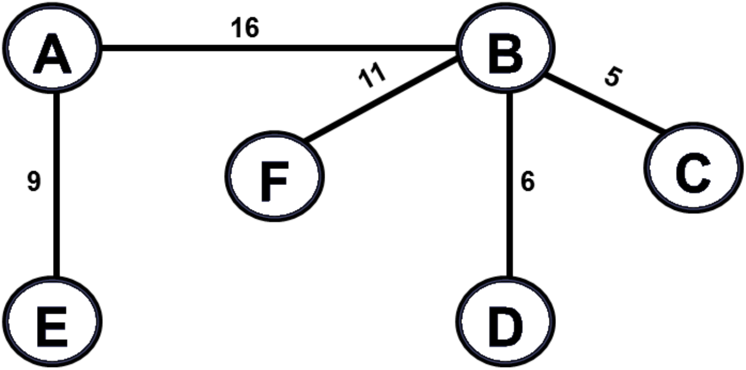


$$C = \{A, E, B, C, D, F\}$$



$$C = \{A, E, B, C, D, F\}$$

Problema da Árvore Geradora Mínima



$$C = \{A, E, B, C, D, F\}$$

Considerações Finais

Decisões tomadas de forma isolada em cada passo

- Estratégia de pegar o melhor no momento
- Solução ótima local
- Quando o algoritmo termina, espera-se que a solução ótima tenha sido encontrada

Conclusão

- Alguns problemas são resolvidos de forma ótima.
- Outros apresentam soluções bem *pobres*.

Referências

- Cormen et. al. *Algoritmos - Teoria e Prática*.
- Nivio Ziviani. *Projeto de Algoritmos com Implementações em Pascal e C*.
- Toscani e Veloso. *Complexidade de Algoritmos*.