

# Fundamentos de Algoritmos e Estrutura de Dados - Aula 06 - Grafos

Prof. André Gustavo Hochuli

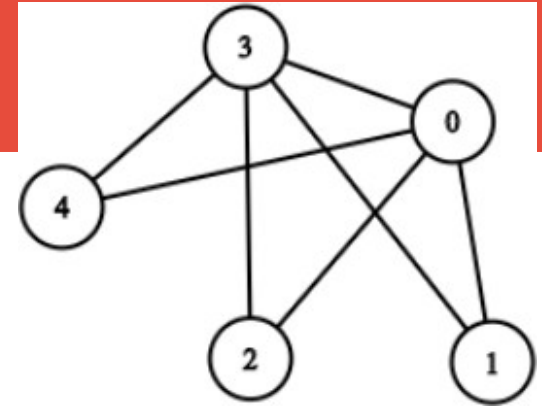
[gustavo.hochuli@pucpr.br](mailto:gustavo.hochuli@pucpr.br)

[aghochuli@ppgia.pucpr.br](mailto:aghochuli@ppgia.pucpr.br)

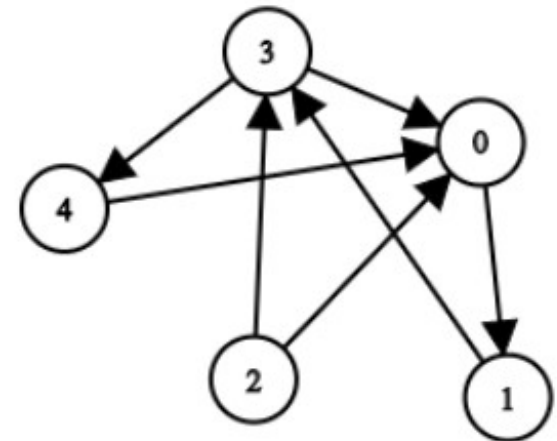
# Plano de Aula

- **Grafos**
  - **Busca Profundidade**
  - **Busca Largura**
  - **Busca A\***
- **Dijkstra**

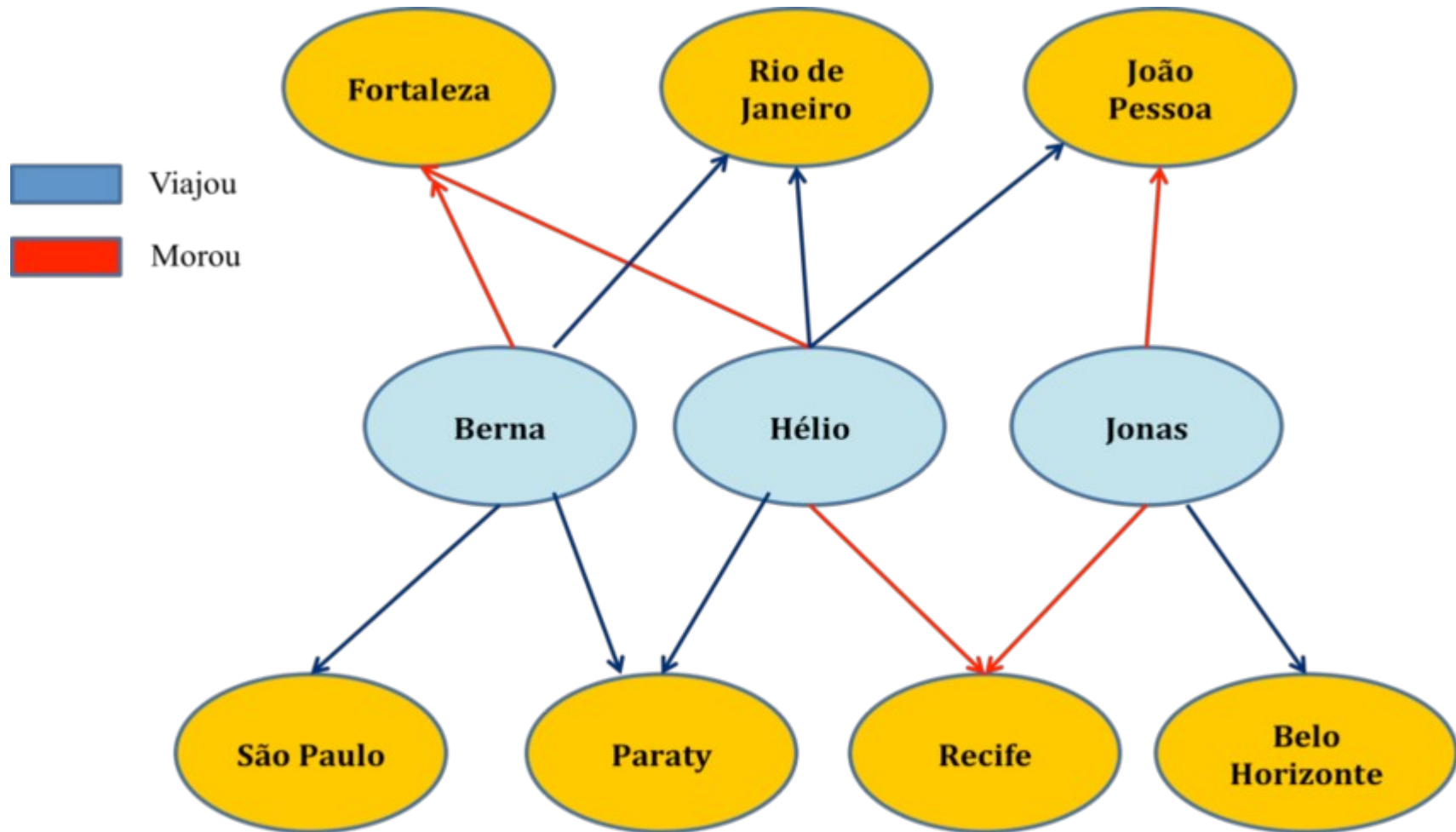
# Grafos



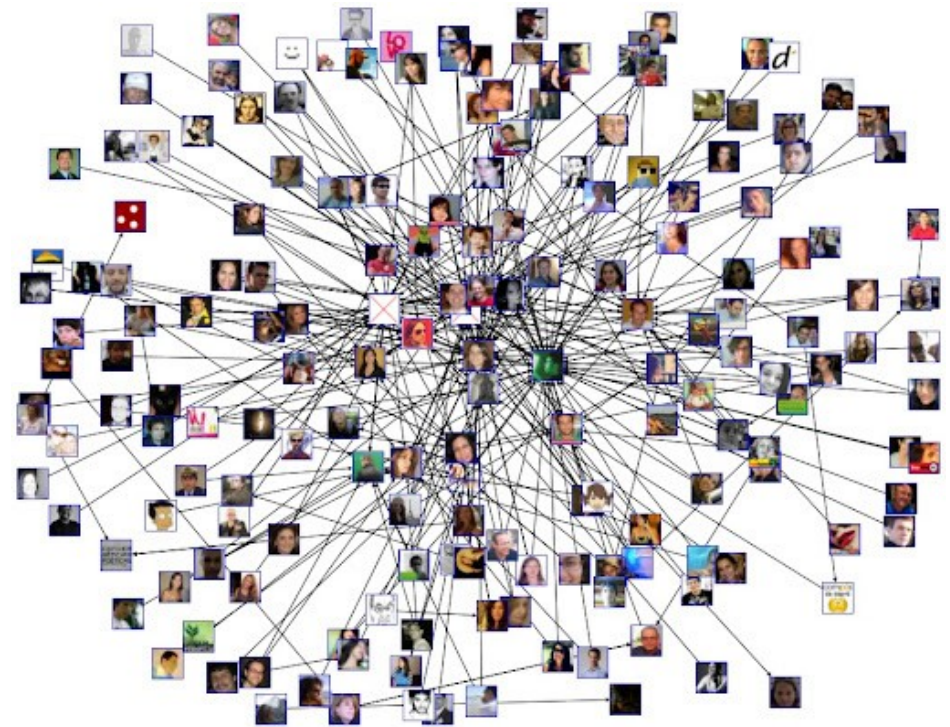
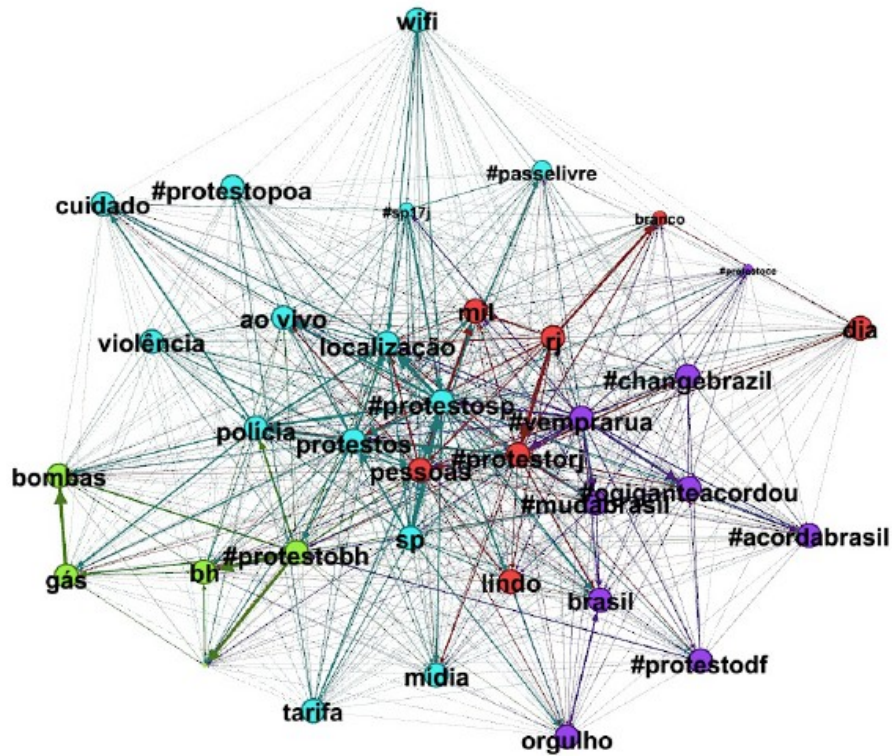
- **Conjunto de Vértices e Arestas**
  - **Direcionado ou Não**
  - **Define graus de relacionamento entre objetos (arestas e vértices)**
- **Utilizado na modelagem de problemas**
  - **Redes Sociais**
    - **Relacionamento entre Empresas, Pessoas, etc**
  - **Roteamento**
    - **Redes de Computadores**
    - **Rotas Rodoviárias, Aéreas, Malha Elétrica....**
  - **Programação Orientada a Objetos (Classes)**



# Grafos (Relacionamentos)



# Grafos (Redes Sociais)



Created with NodeXL (<http://nodexl.codeplex.com/>)

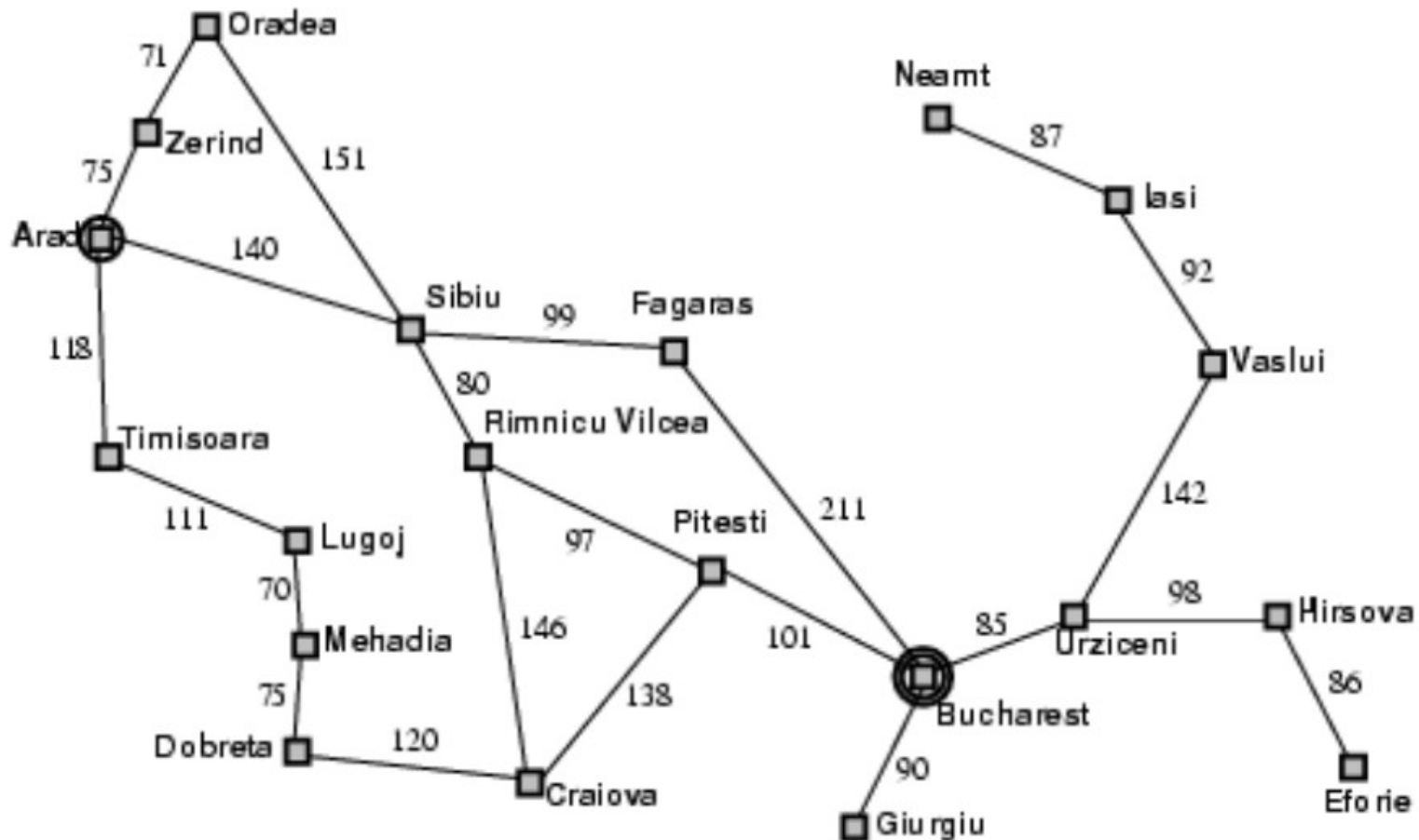


# Grafos (Roteamento)

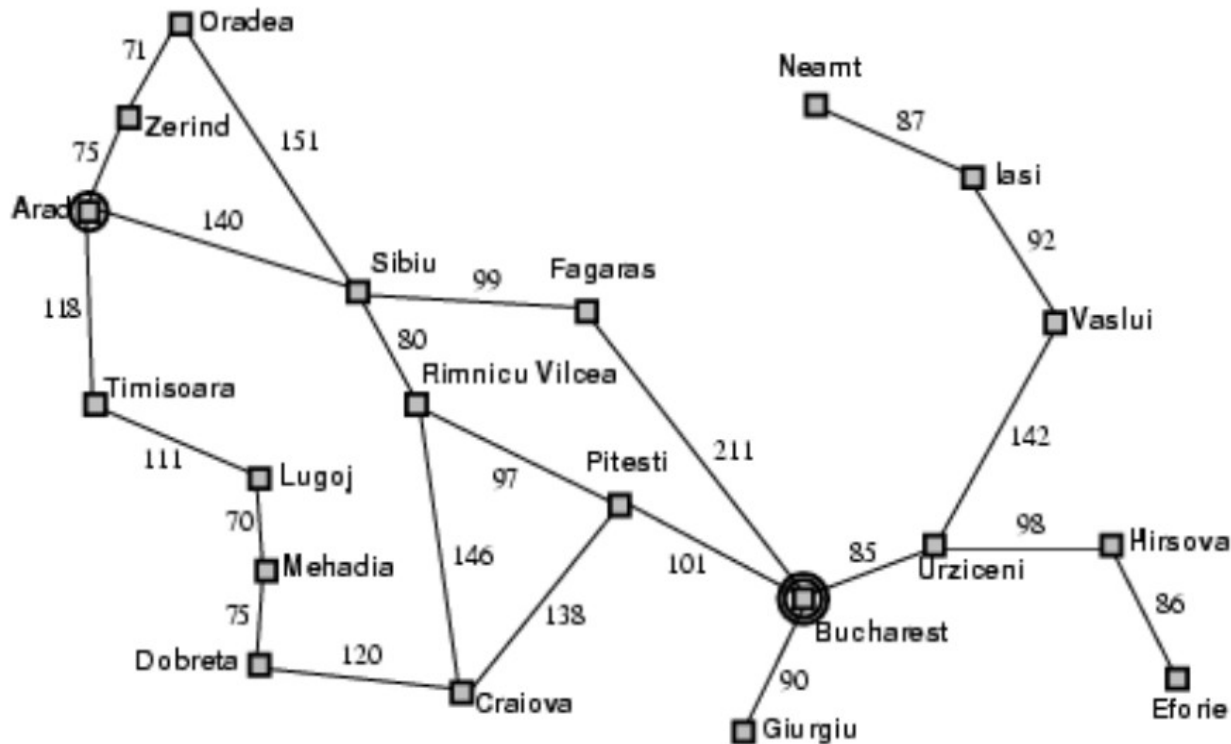


# Busca Cegas (Sem Informação)

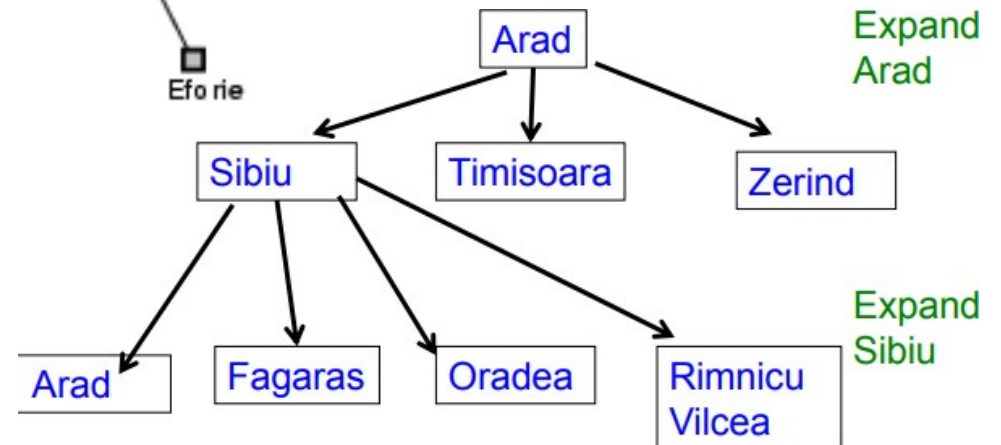
Problema: Arad → Bucharest



# Busca Cegas (Sem Informação)

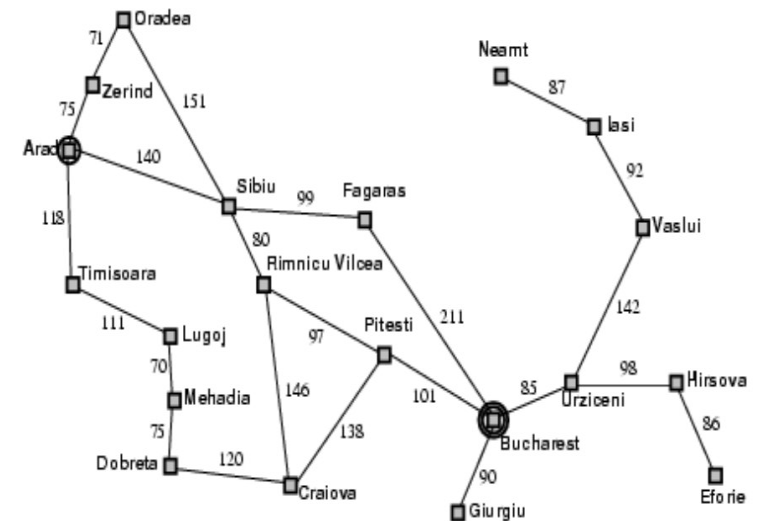
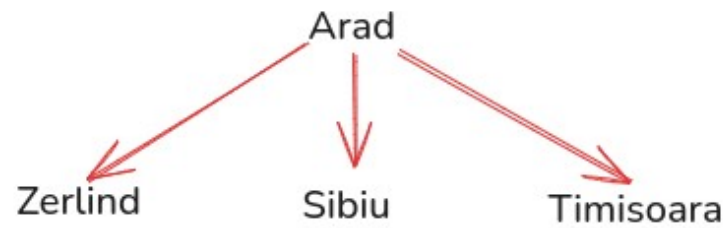


Problema: Qual nodo expandir?

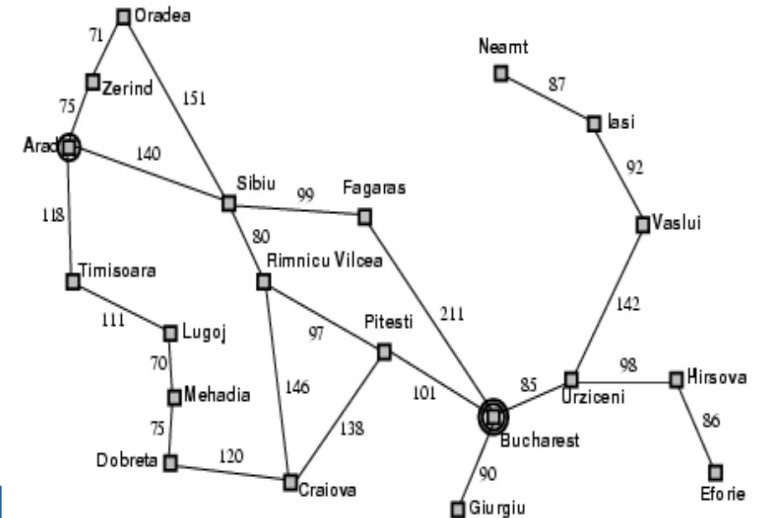
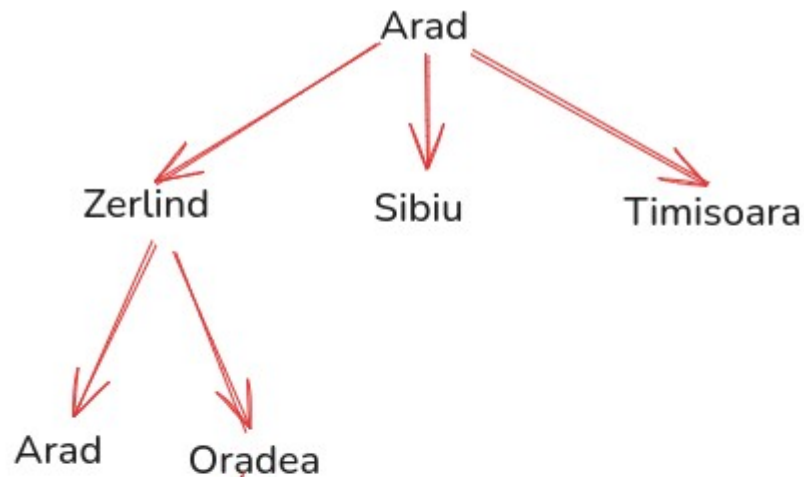




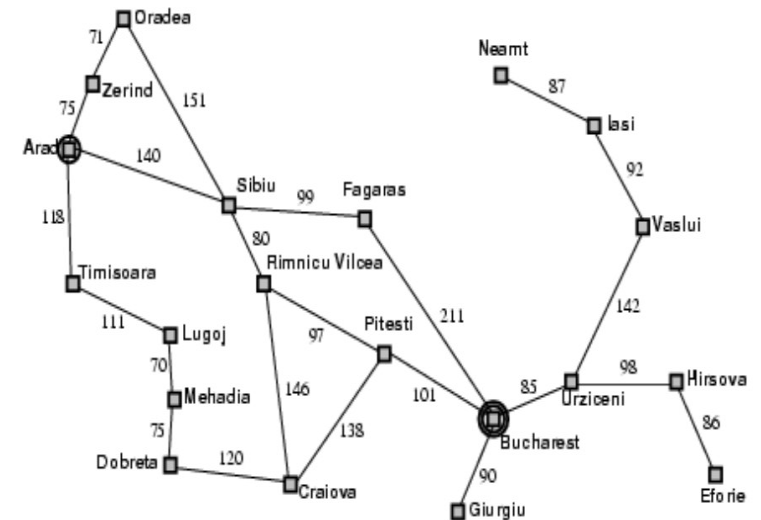
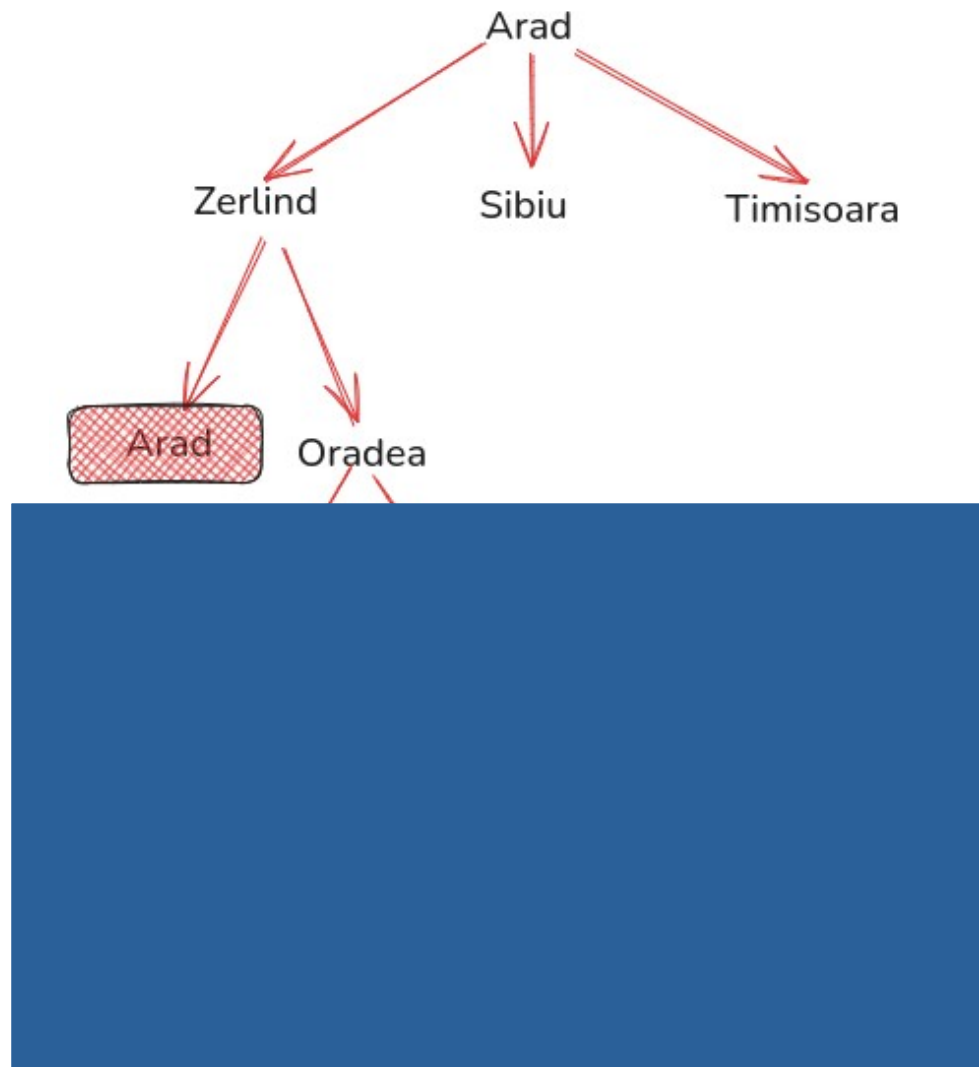
# Busca Cega: Profundidade



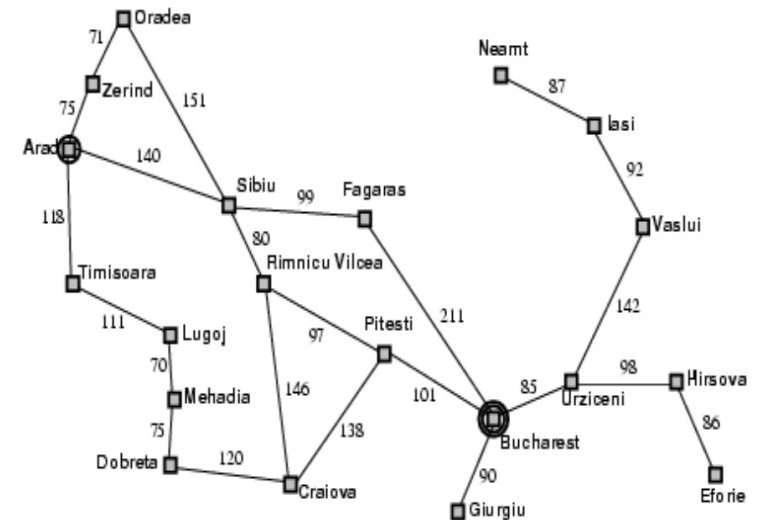
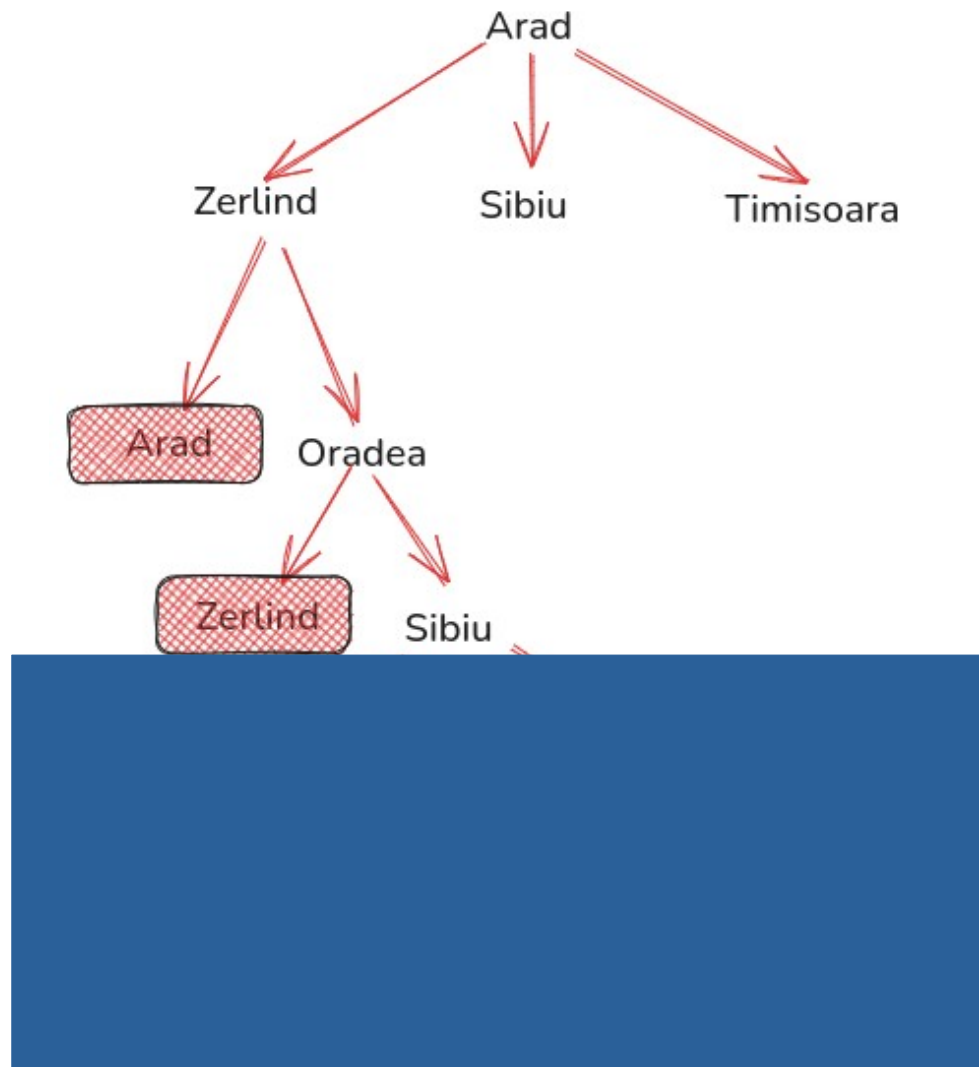
# Busca Cega: Profundidade



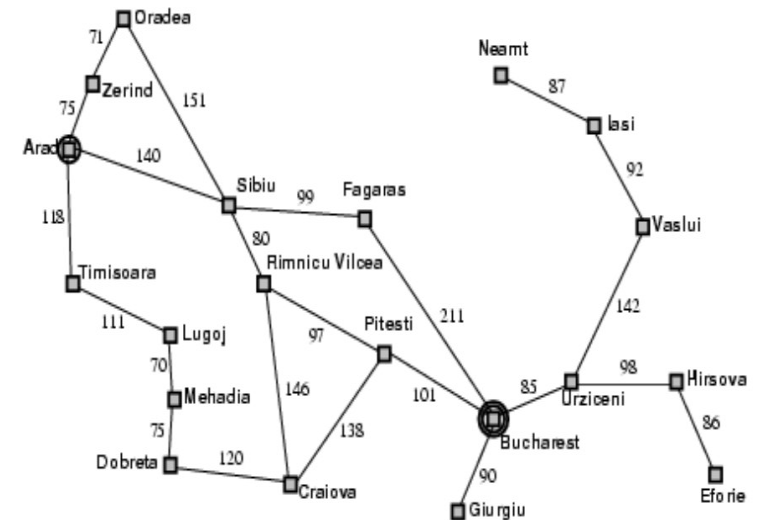
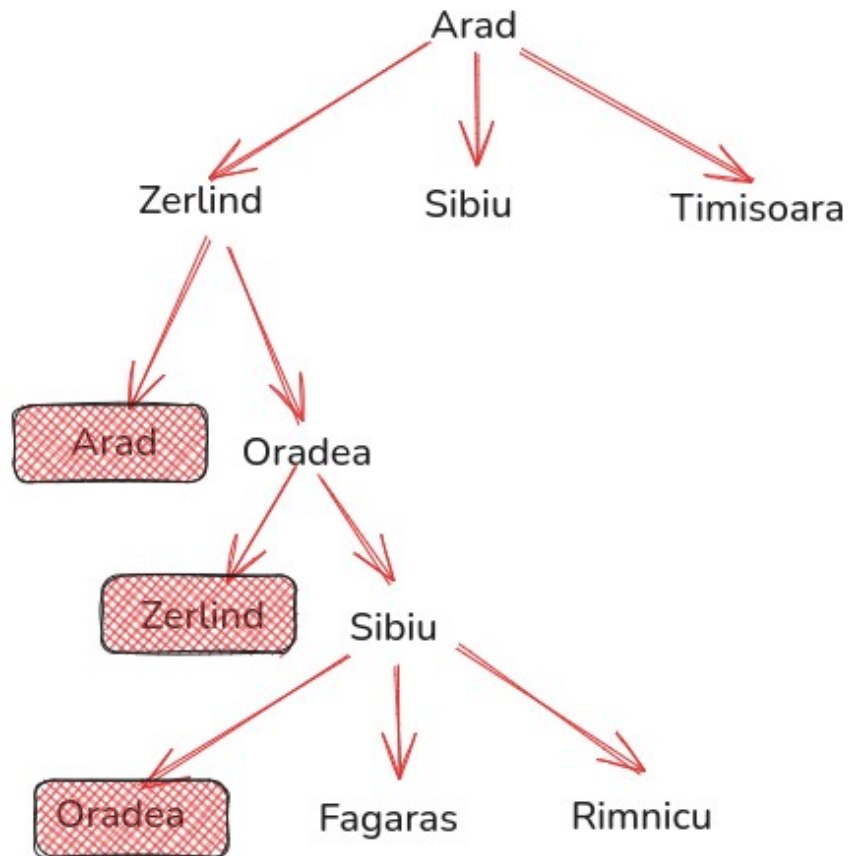
# Busca Cega: Profundidade



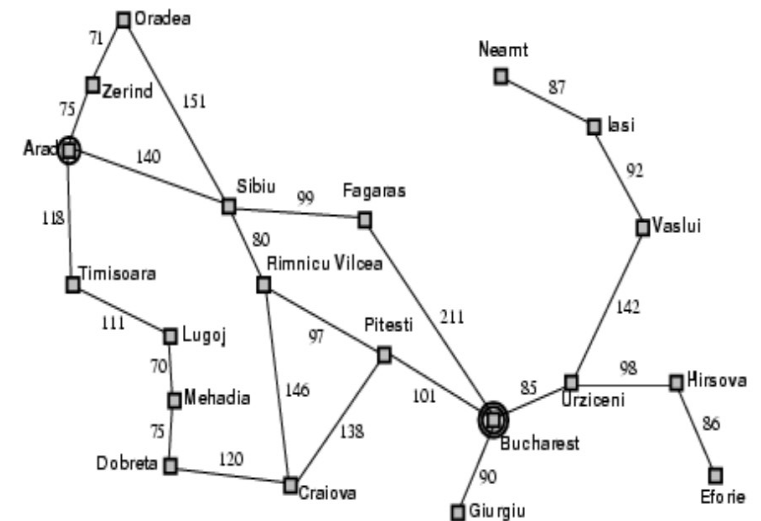
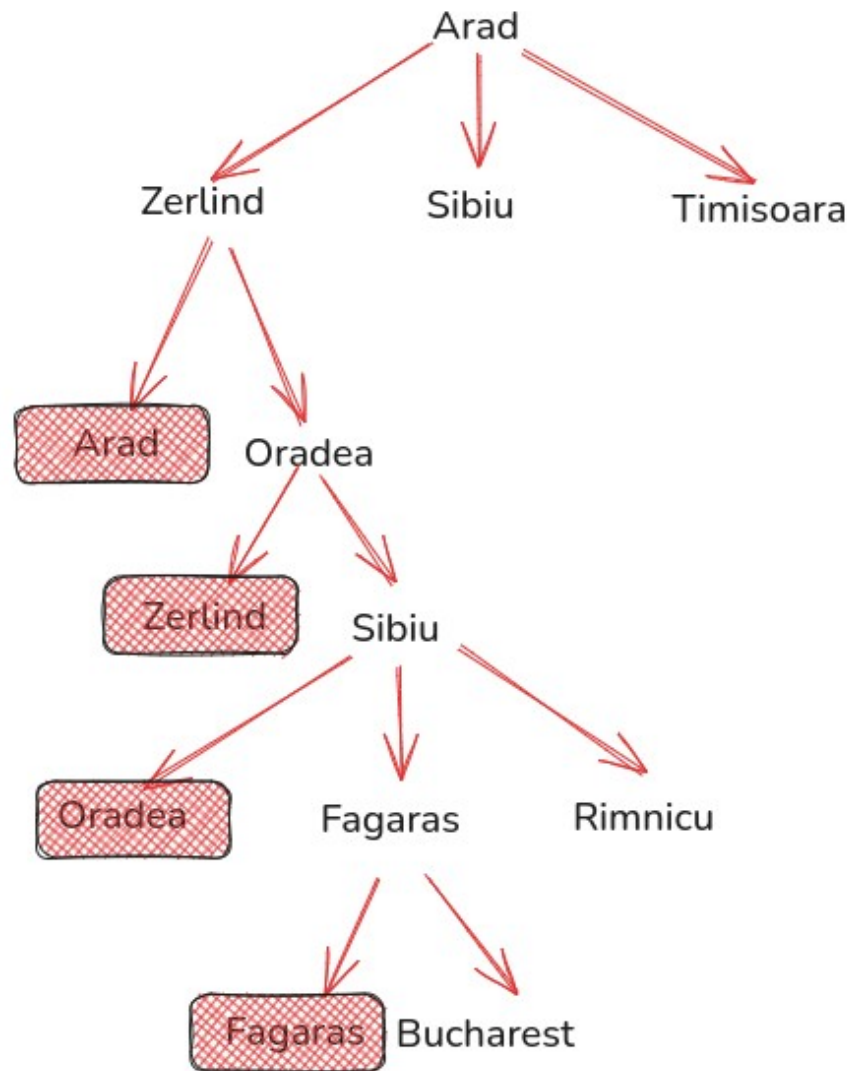
# Busca Cega: Profundidade



# Busca Cega: Profundidade



# Busca Cega: Profundidade



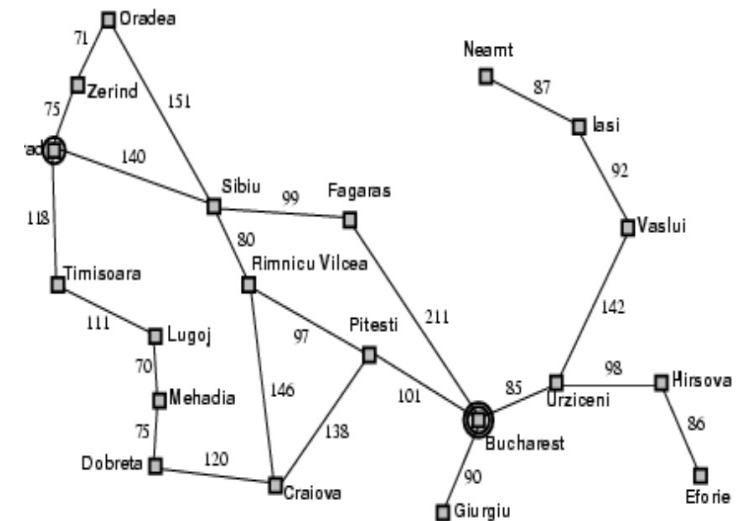
Não Garante a melhor solução!

Ciclos devem ser tratados!

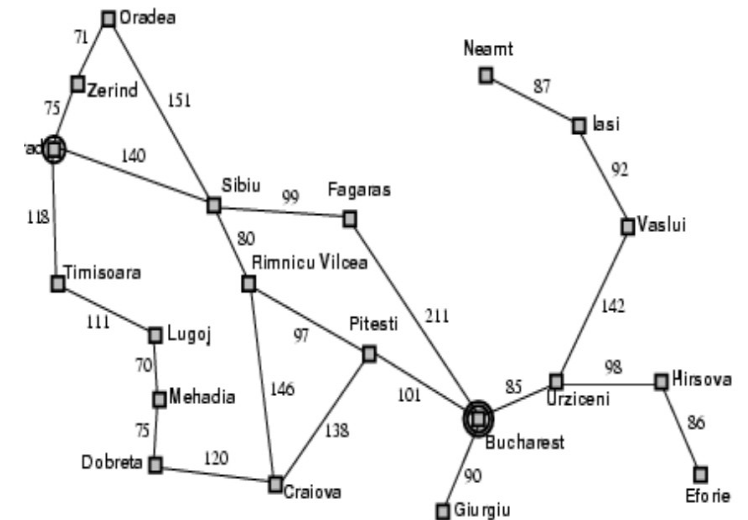
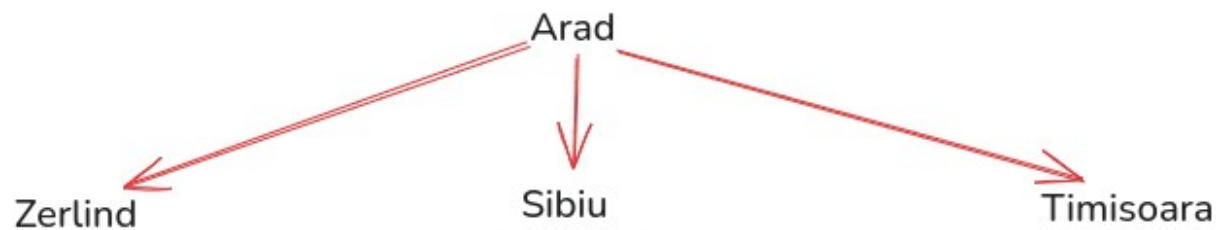


# Busca Cega: Largura

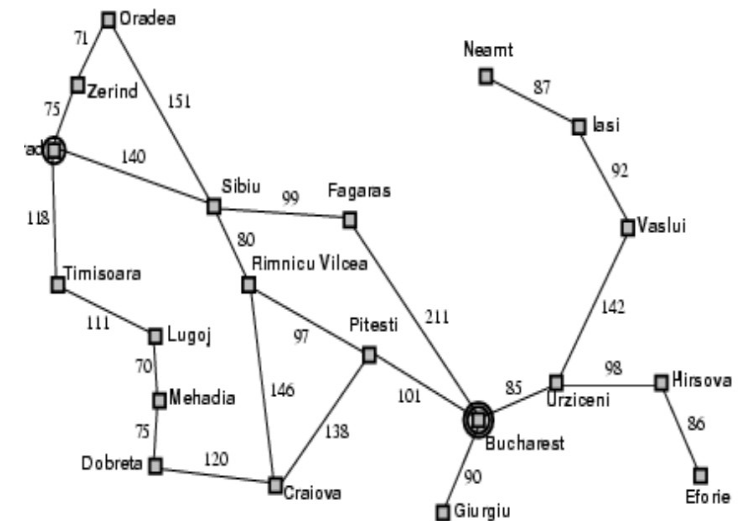
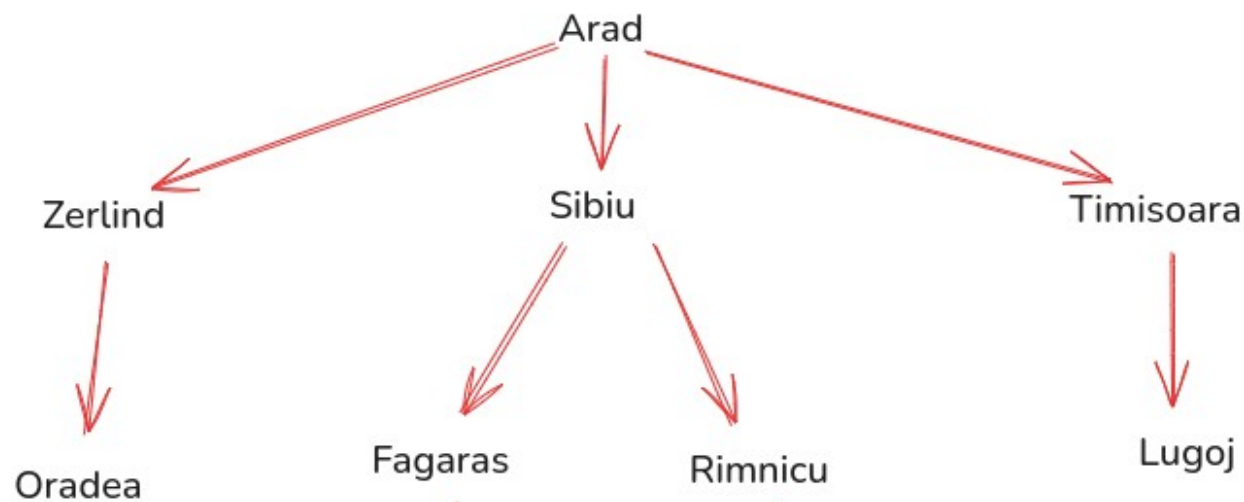
Arad



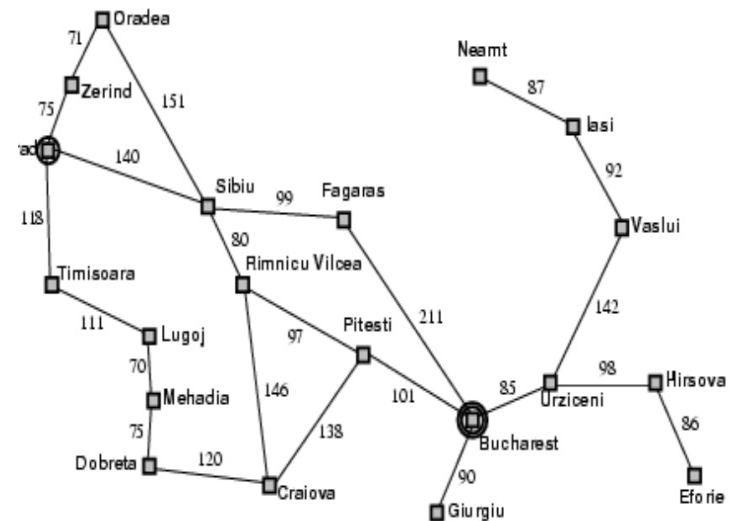
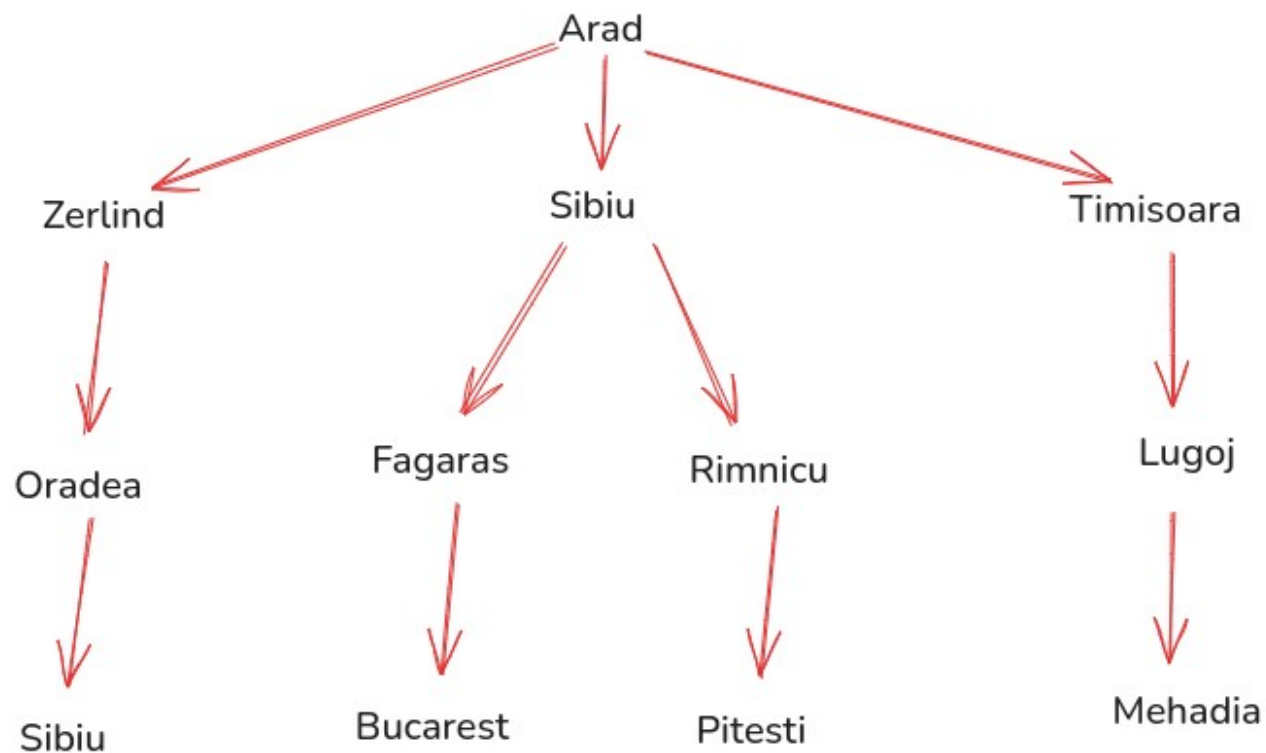
# Busca Cega: Largura



# Busca Cega: Largura



# Busca Cega: Largura



Garante a melhor solução!

Ciclos devem ser tratados!

# Busca Cega: Largura

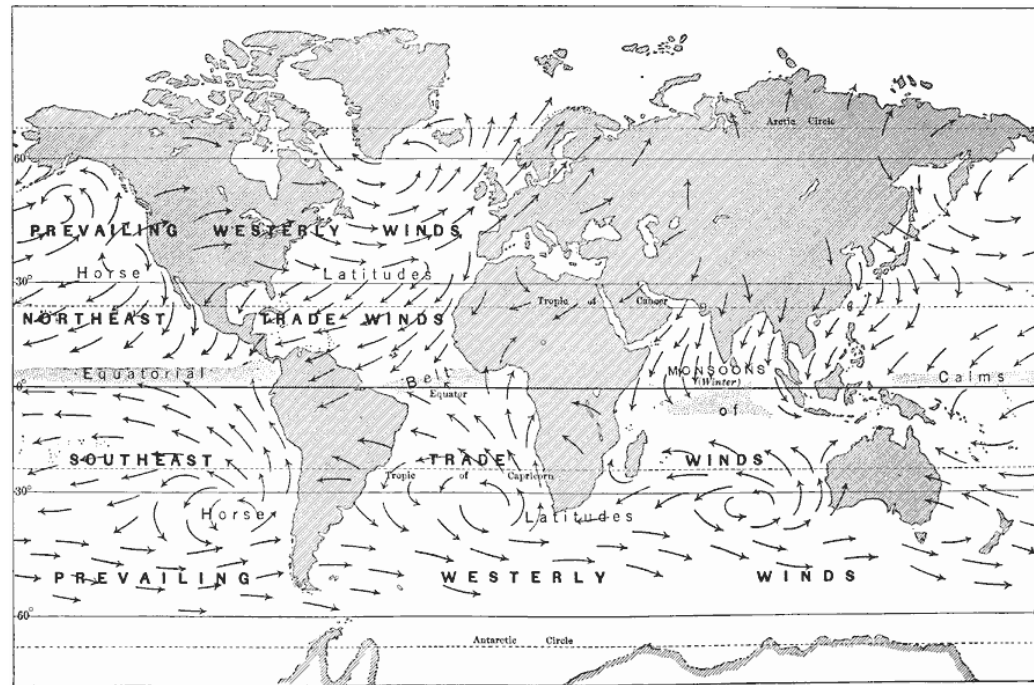
- Bons resultados para arvores com pouca profundidade
- Caso contrário, o custo computacional é alto
- Exemplo
  - Ramificação  $b=10$ , 1 M nodos/seg , 1 KB por node

profundidade	nós	tempo	memória
2	1100	0,11 seg	1 MB
4	111.100	11 seg	106 MB
6	$10^7$	19 min	10 GB
8	$10^9$	31 horas	1 TeraB
10	$10^{11}$	129 dias	101 TeraB
12	$10^{13}$	35 anos	10 PentaB
14	$10^{15}$	3.523 anos	1 exaB

$1+b^1+b^2+(b^3-b) = 1+10+100+(1000-10) = 1101$

# Busca Heurística (Informada)

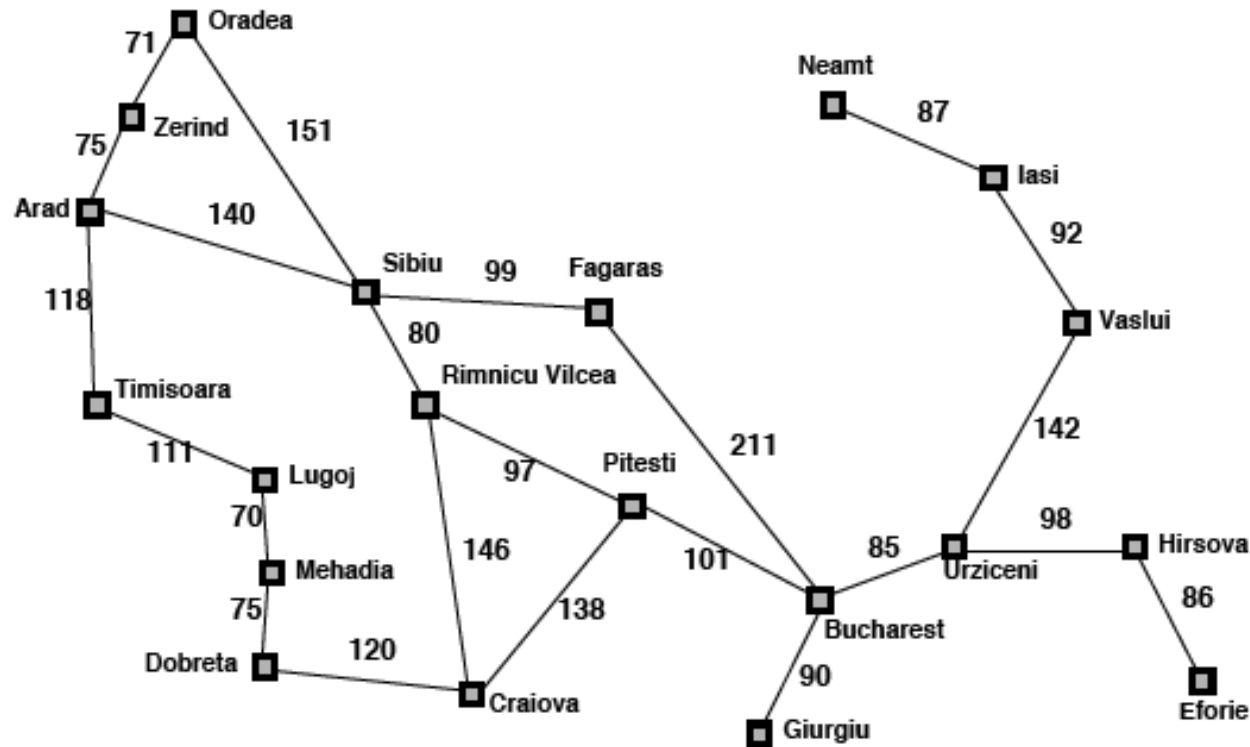
- Utiliza uma função heurística para determinar a próxima expansão
- Problema: Procurar um barco no oceano
  - Cega: M2 por M2
  - Heurística: Corrente Marítima, Ventos, ...





# Busca Heurística

## Romania with step costs in km

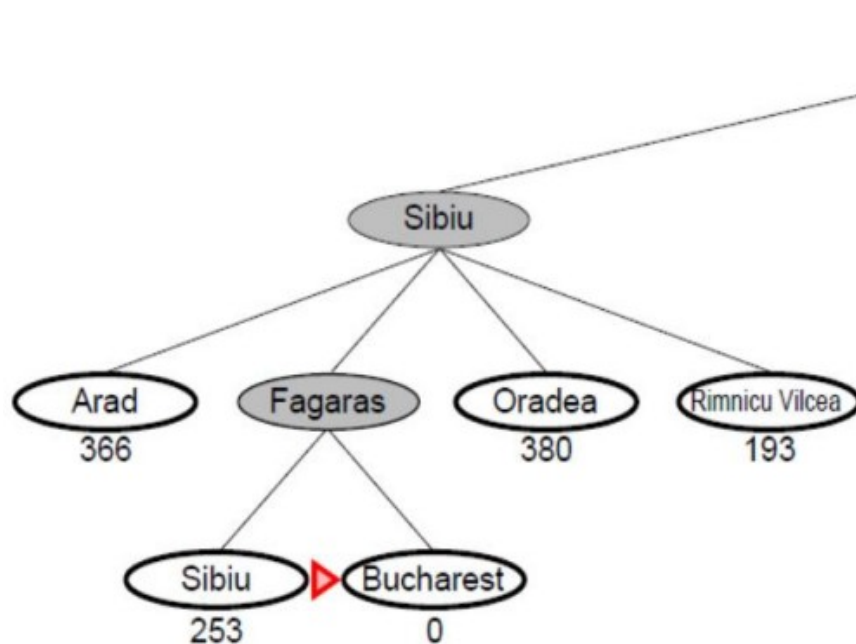


Straight-line distance  
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

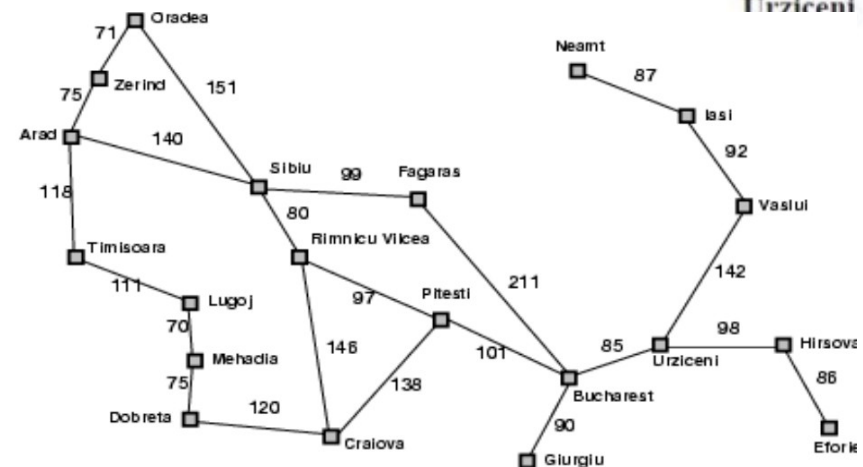
# Busca Heurística: “Ambiciosa” (Greedy Search)

- $f(n) = h(n)$
- $h(n)$ : Custo de  $n$  até folha ( $n \rightarrow$  destino)



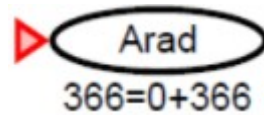
*straight-line distances  
to Bucharest*

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
	199
	374



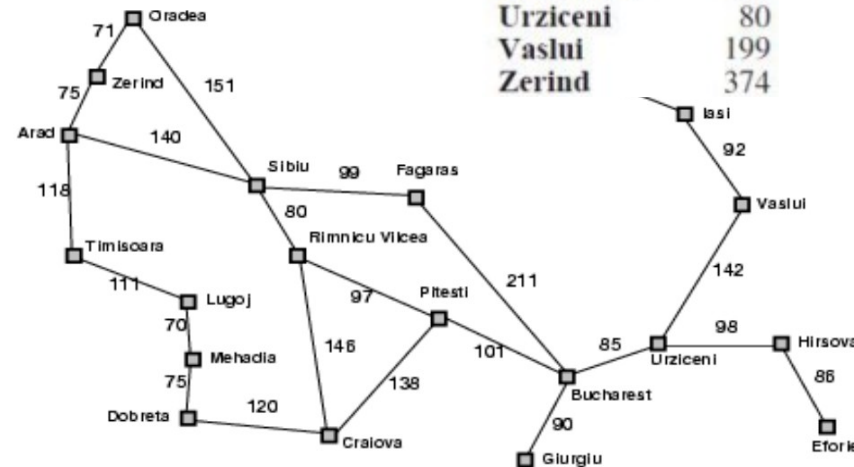
# Busca Heurística: A\*

- $f(n) = g(n) + h(n)$

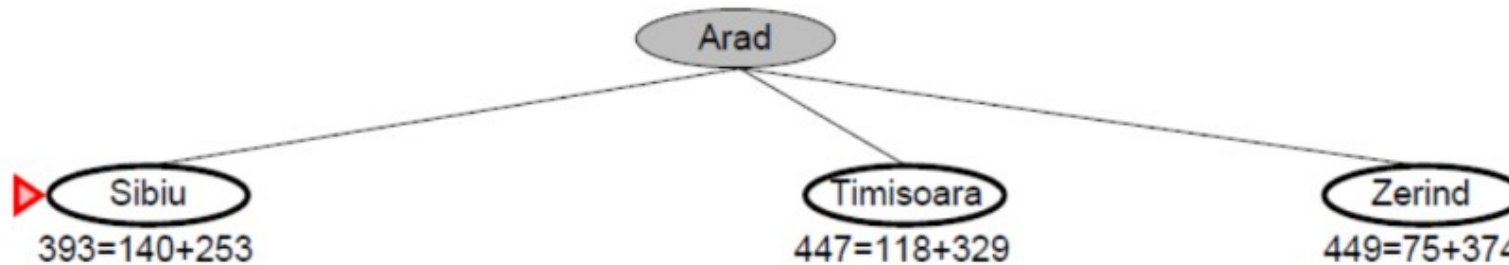


*straight-line distances  
to Bucharest*

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

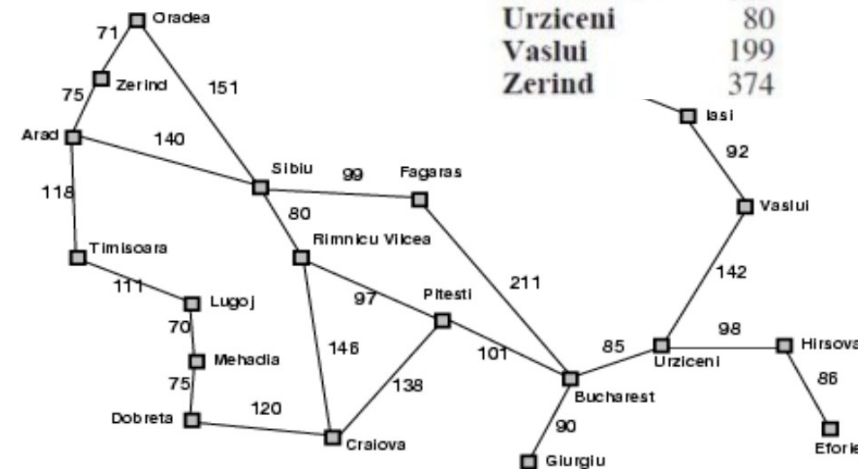


# Busca Heurística: A\*



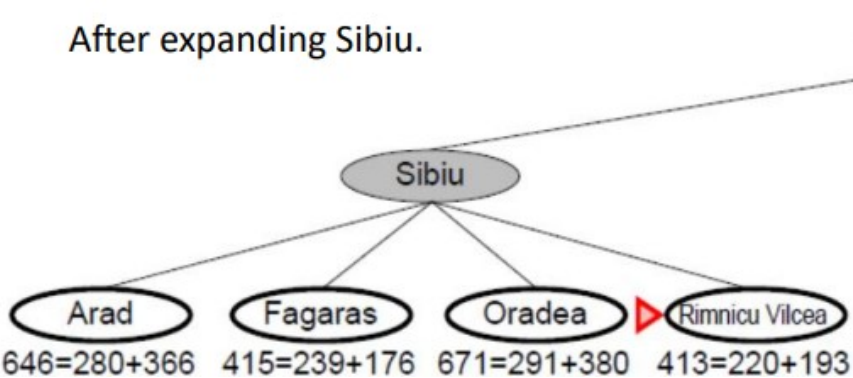
*straight-line distances  
to Bucharest*

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



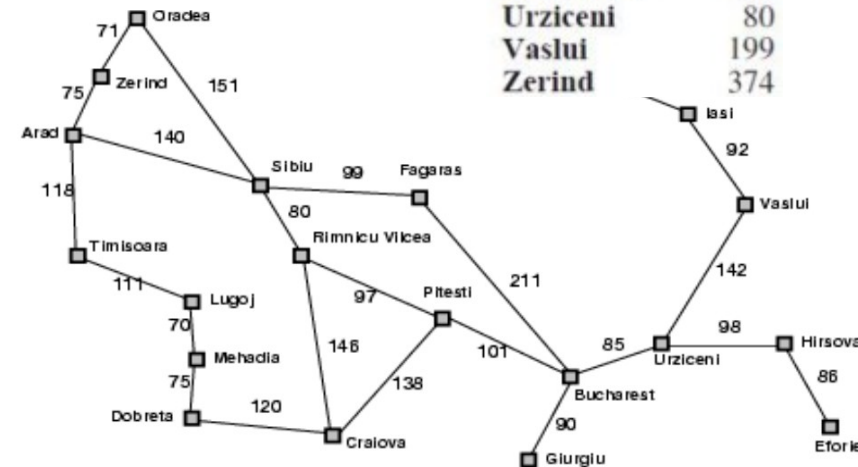
# Busca Heurística: A\*

After expanding Sibiu.



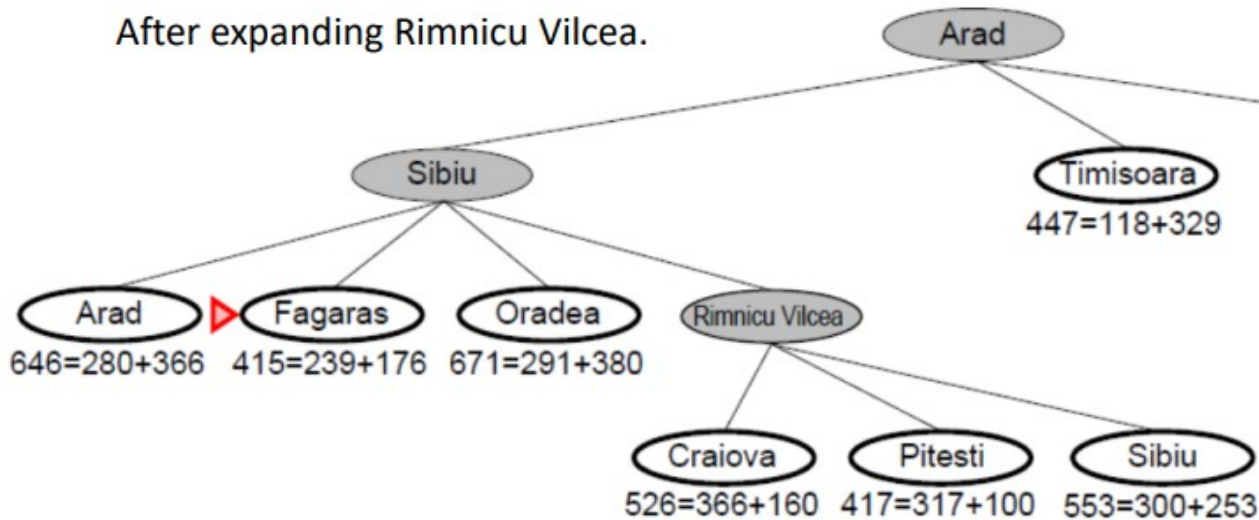
*straight-line distances  
to Bucharest*

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



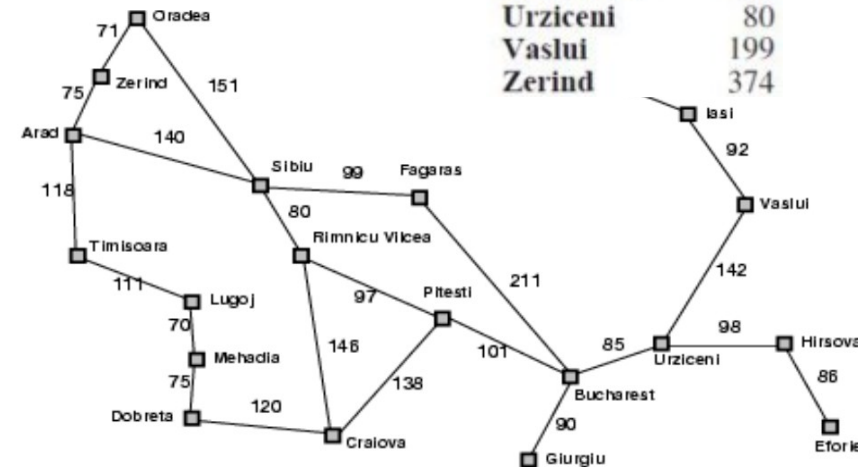
# Busca Heurística: A\*

After expanding Rimnicu Vilcea.



straight-line distances  
to Bucharest

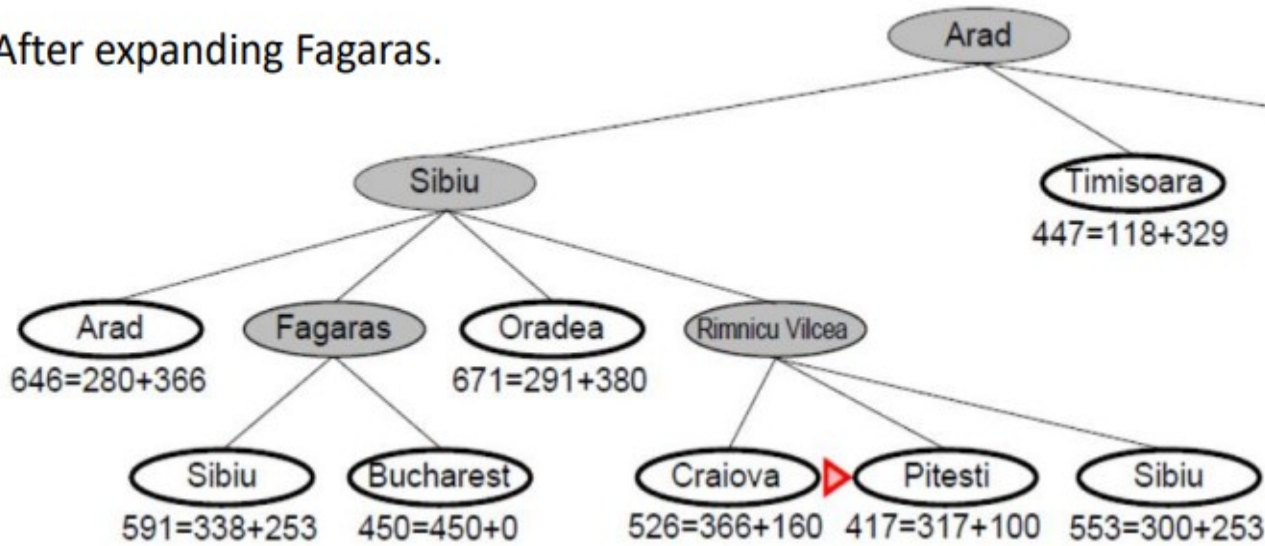
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374





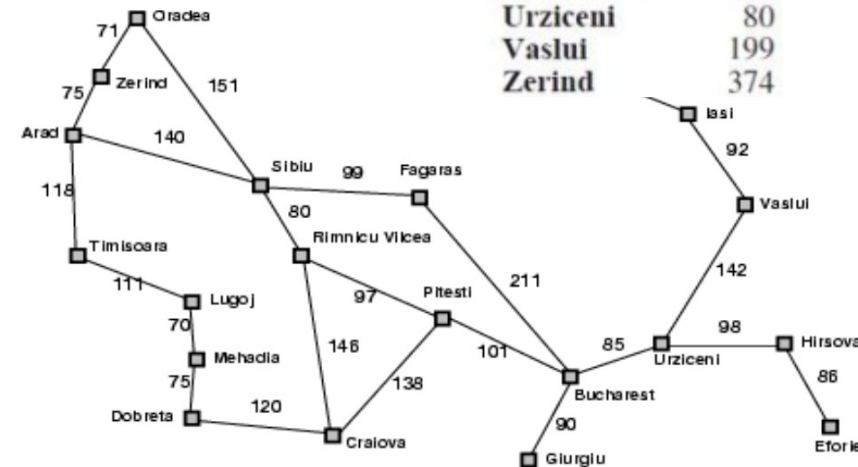
# Busca Heurística: A\*

After expanding Fagaras.



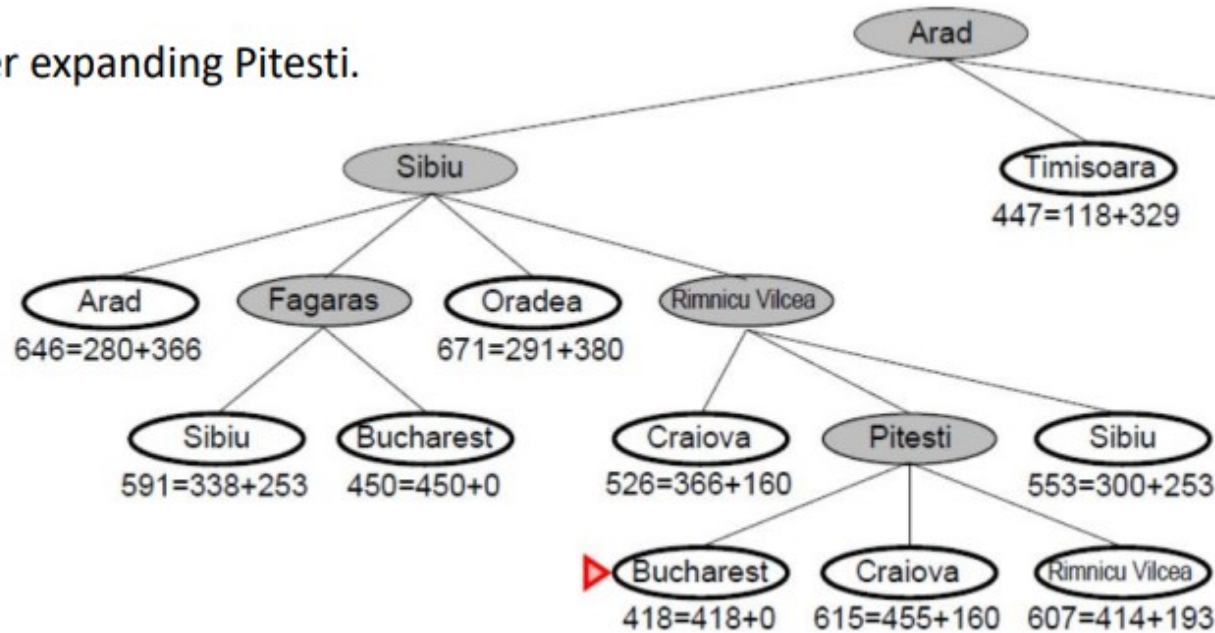
straight-line distances  
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



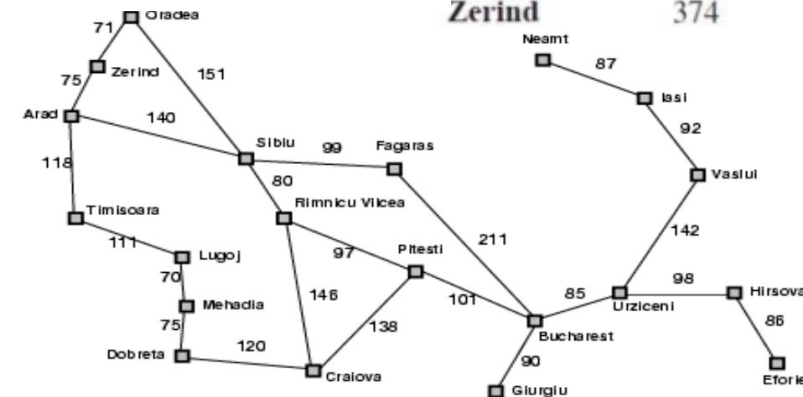
# Busca Heurística: A\*

After expanding Pitesti.



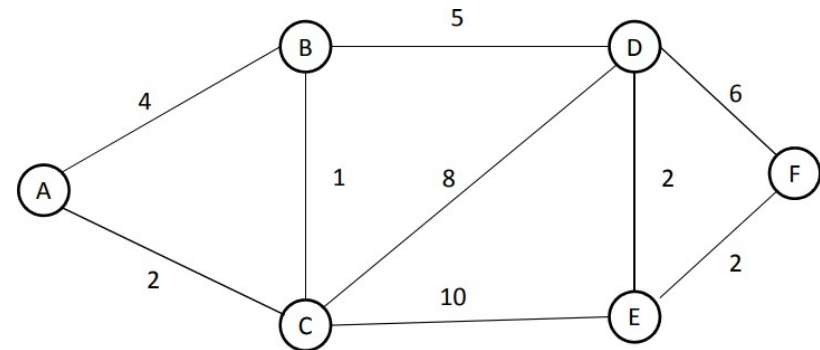
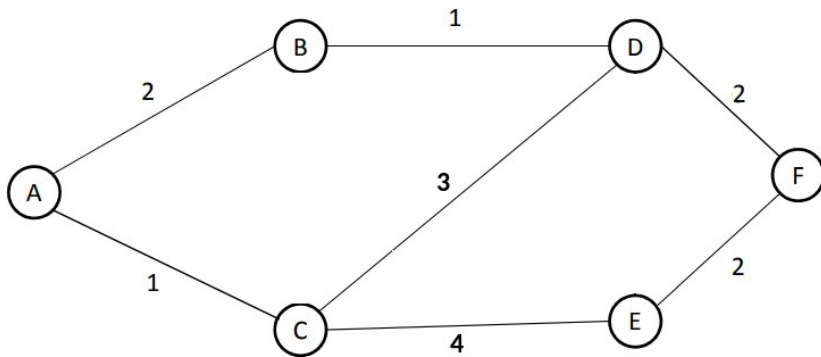
straight-line distances  
o Bucharest

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



# Algoritmo de Dijkstra

- Edsger Dijkstra - 1959
- Caminho mais curto em grafo
- De A para F

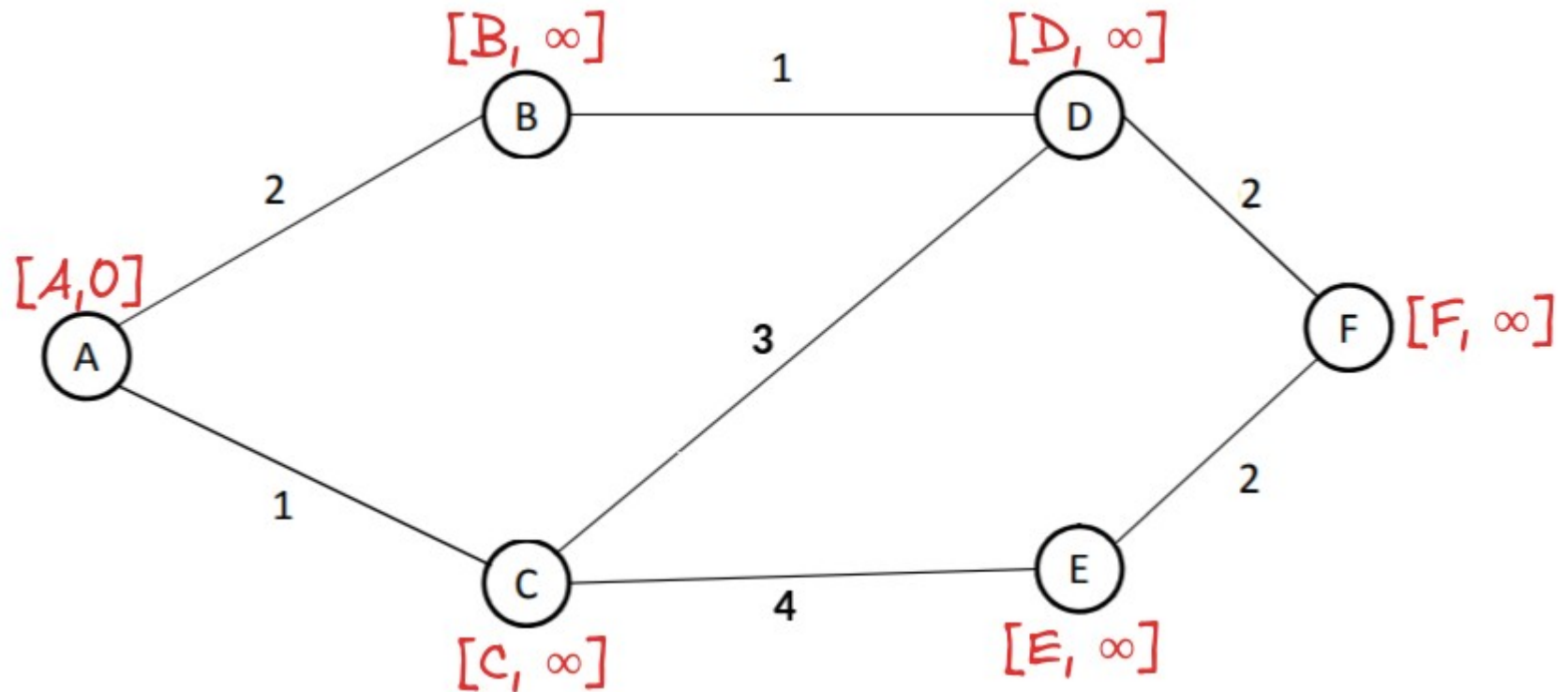


# Algoritmo de Dijkstra

Inicialização:

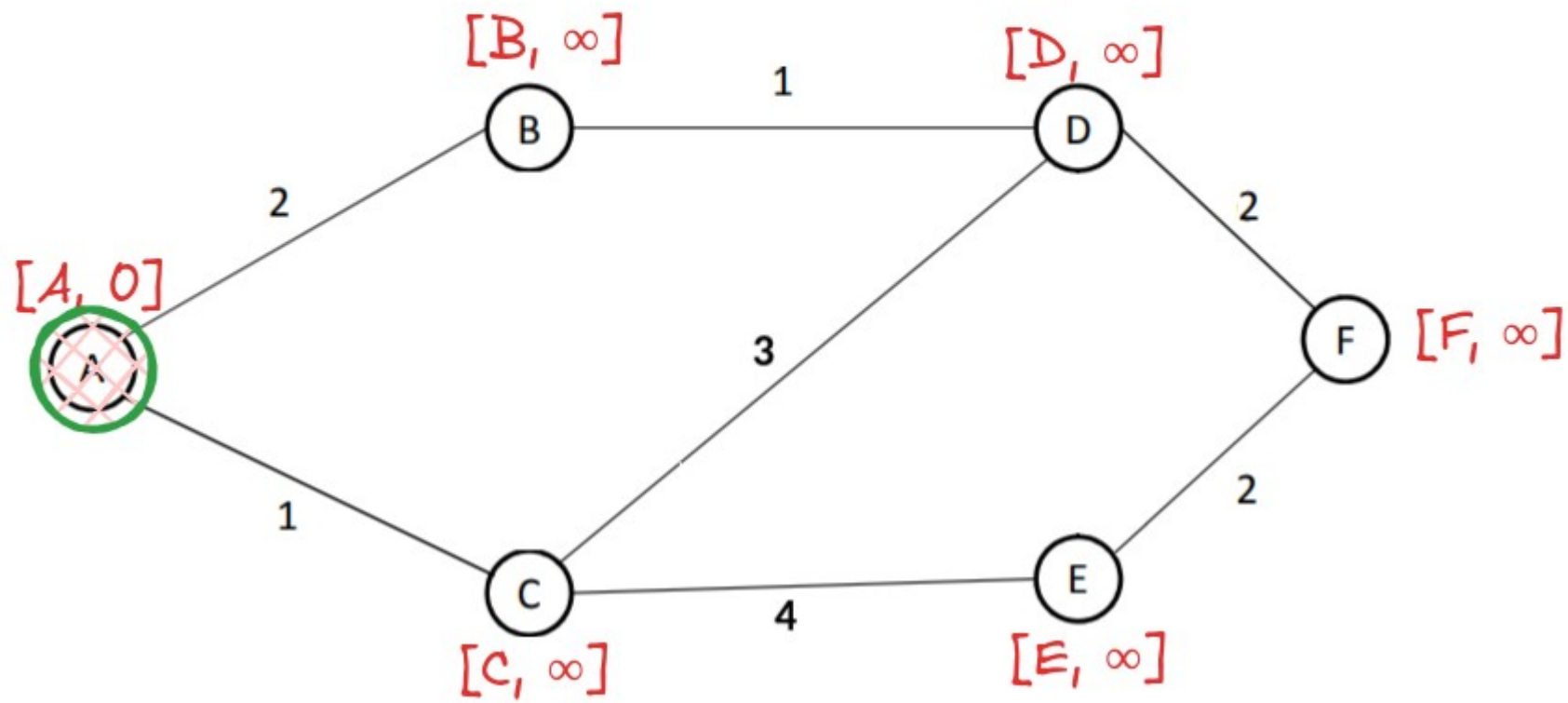
Distância do nó de origem = 0;

Demais nós =  $\infty$ ;



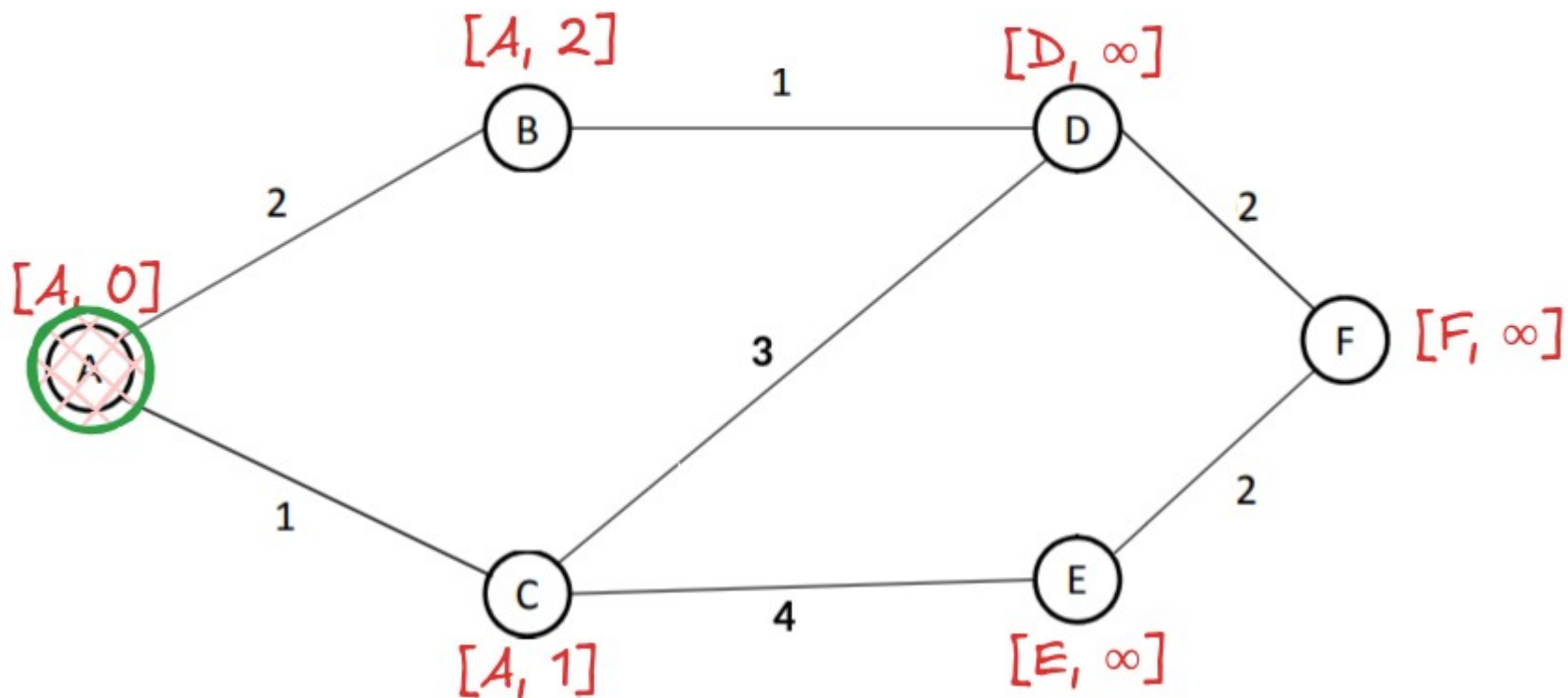
# Algoritmo de Dijkstra

- ✗ 1. Selecione o nó não visitado com menor distância.
- 2. Para cada vizinho, calcule a nova distância via o nó atual.  
Atualize a distância do vizinho se a nova for menor.
- 3. Repita até todos os nós serem visitados



# Algoritmo de Dijkstra

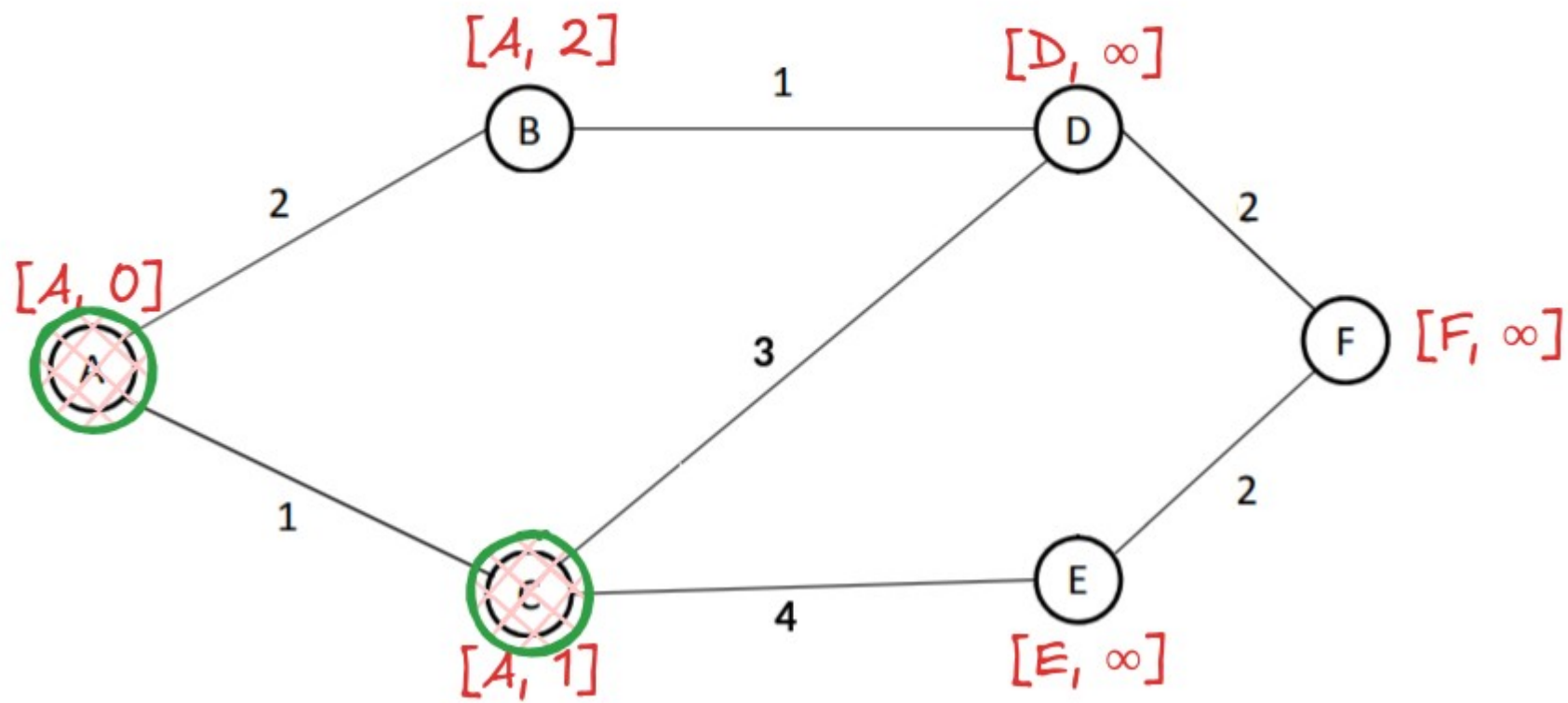
1. Selecione o nó não visitado com menor distância.
- X 2. Para cada vizinho, calcule a nova distância via o nó atual.  
Atualize a distância do vizinho se a nova for menor.
3. Repita até todos os nós serem visitados





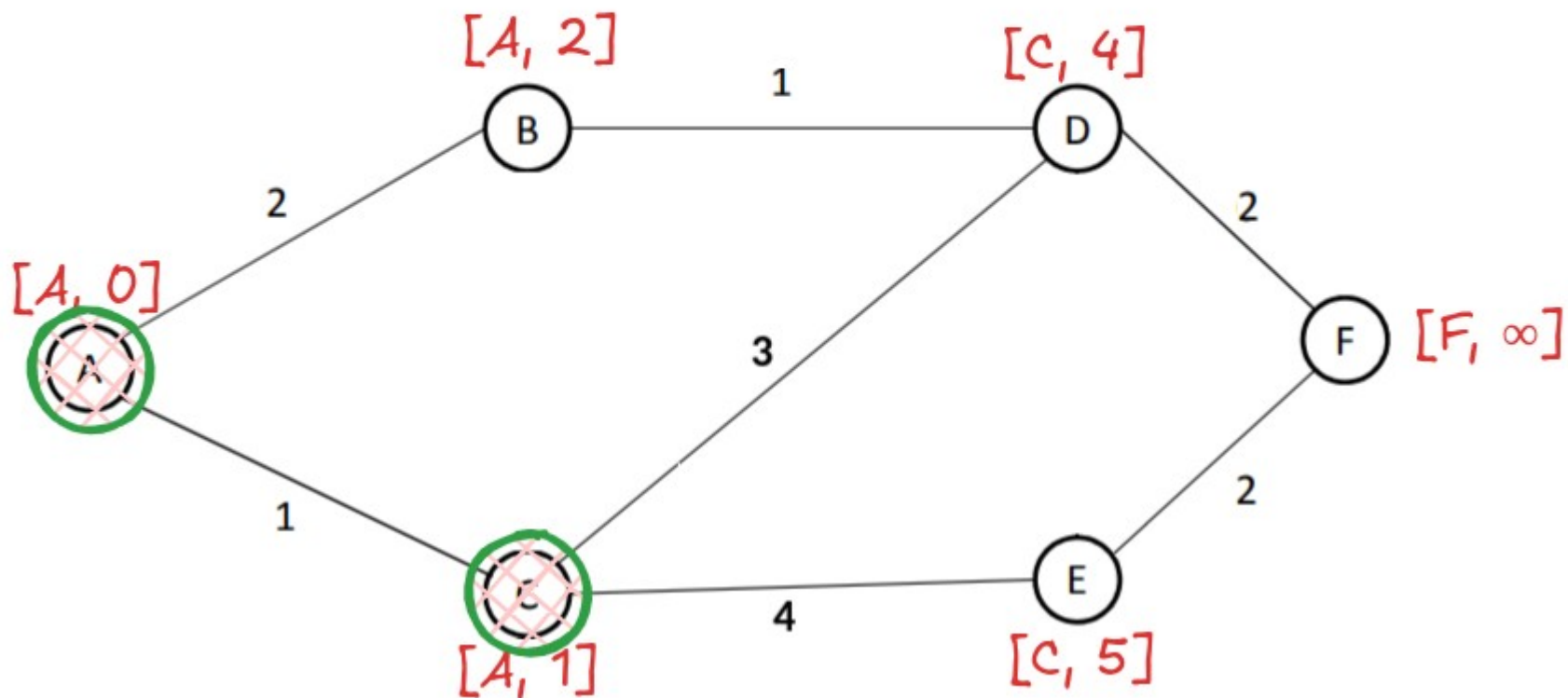
# Algoritmo de Dijkstra

- X 1. Selecione o nó não visitado com menor distância.
- 2. Para cada vizinho, calcule a nova distância via o nó atual.  
Atualize a distância do vizinho se a nova for menor.
- 3. Repita até todos os nós serem visitados



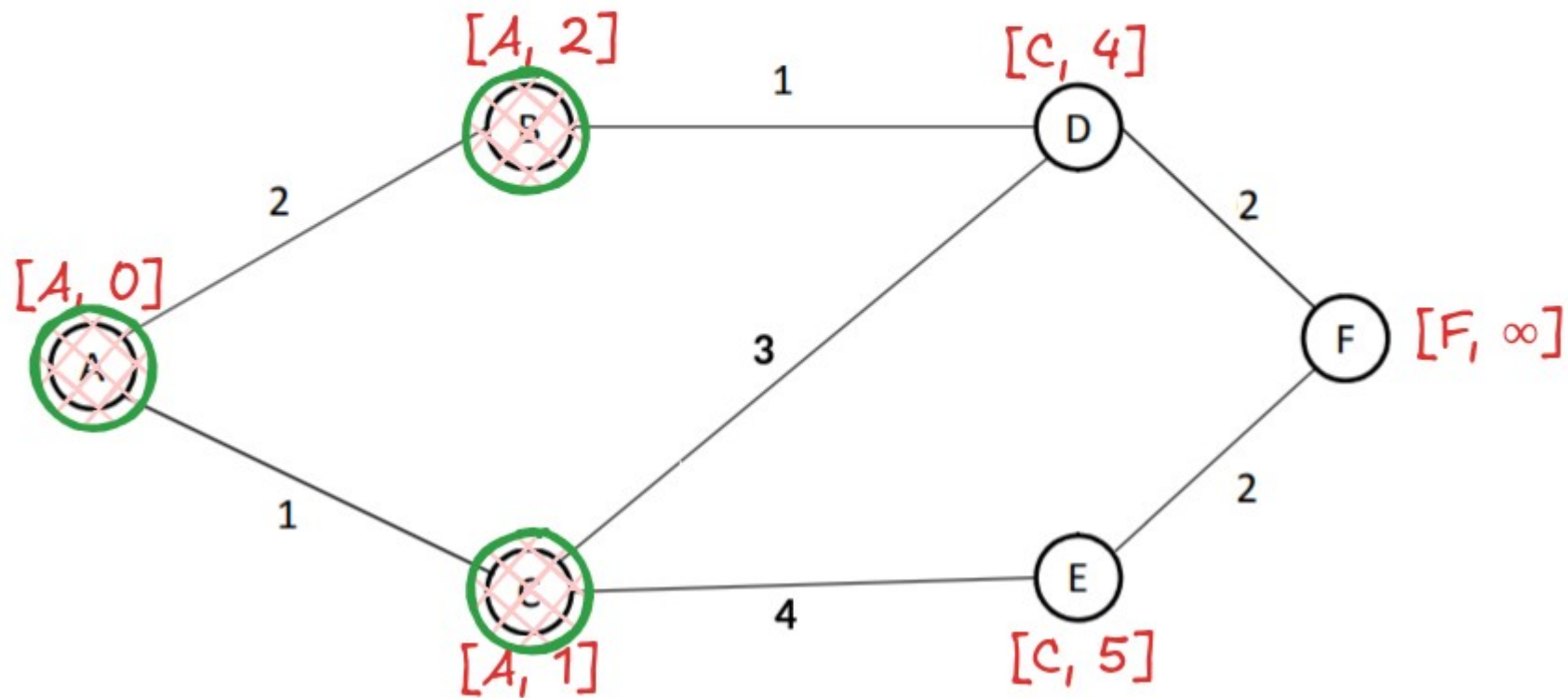
# Algoritmo de Dijkstra

1. Selecione o nó não visitado com menor distância.
- X 2. Para cada vizinho, calcule a nova distância via o nó atual.  
Atualize a distância do vizinho se a nova for menor.
3. Repita até todos os nós serem visitados



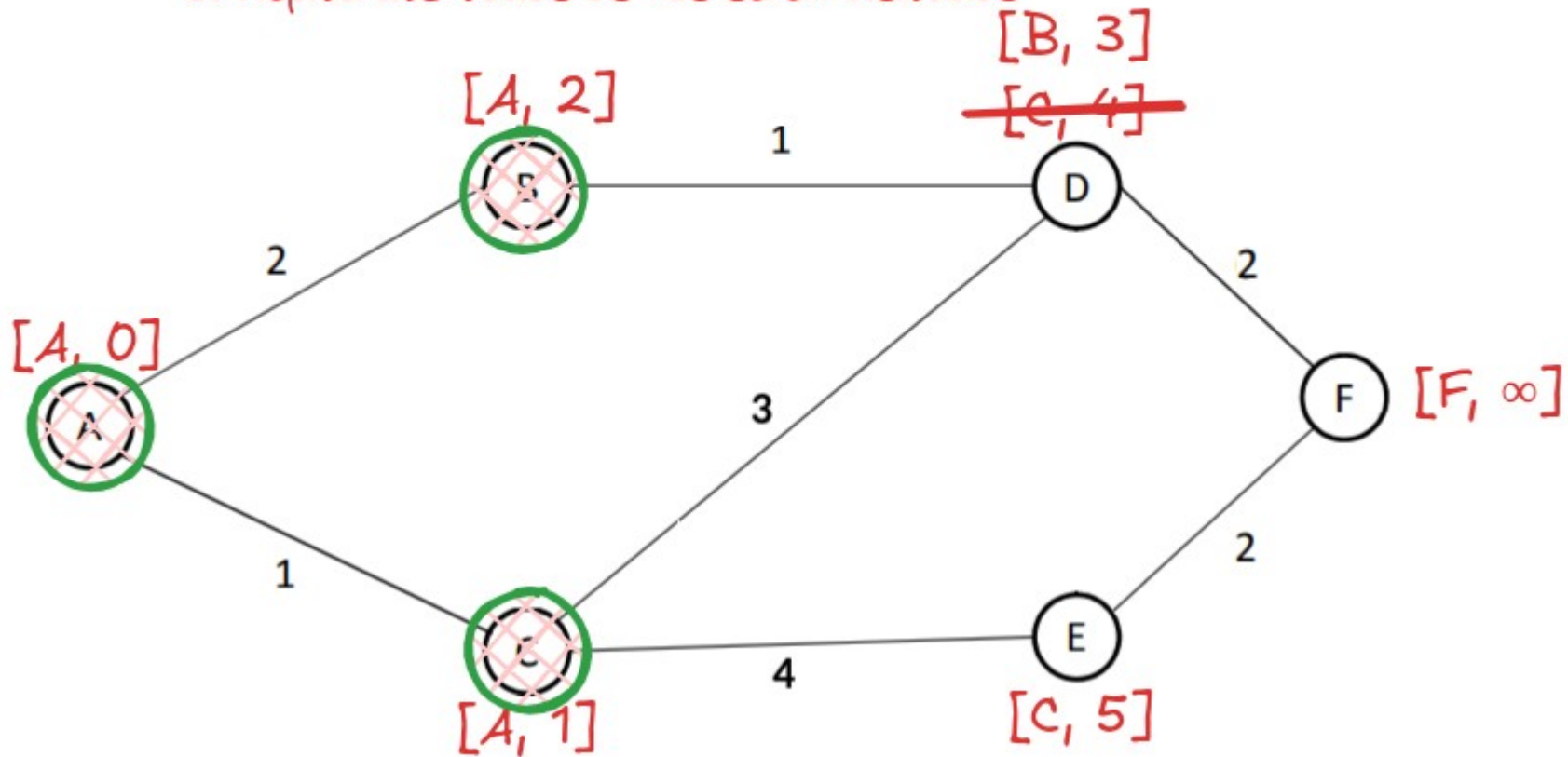
# Algoritmo de Dijkstra

- X 1. Selecione o nó não visitado com menor distância.
- 2. Para cada vizinho, calcule a nova distância via o nó atual.  
Atualize a distância do vizinho se a nova for menor.
- 3. Repita até todos os nós serem visitados



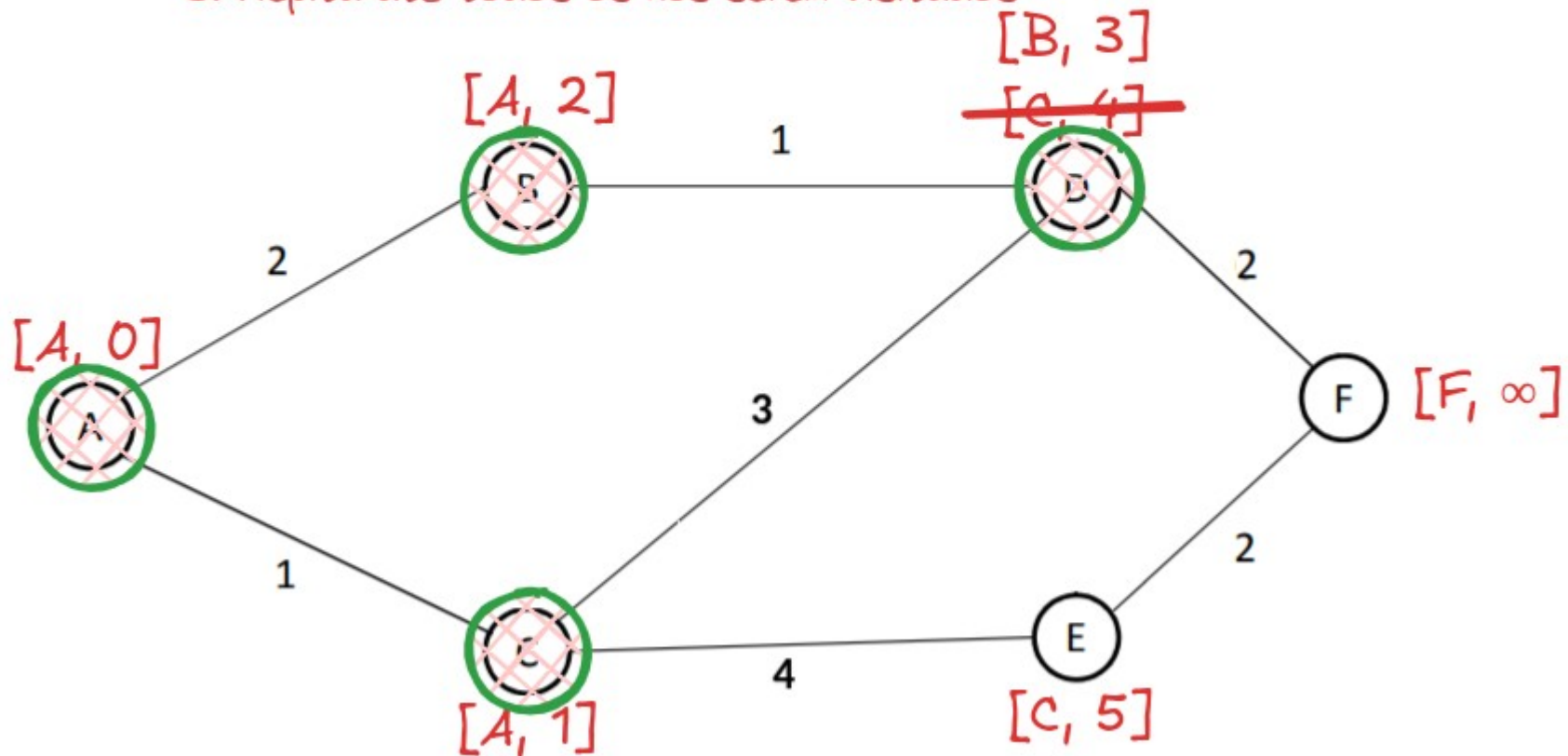
# Algoritmo de Dijkstra

1. Selecione o nó não visitado com menor distância.
- X 2. Para cada vizinho, calcule a nova distância via o nó atual.  
→ Atualize a distância do vizinho se a nova for menor.
3. Repita até todos os nós serem visitados



# Algoritmo de Dijkstra

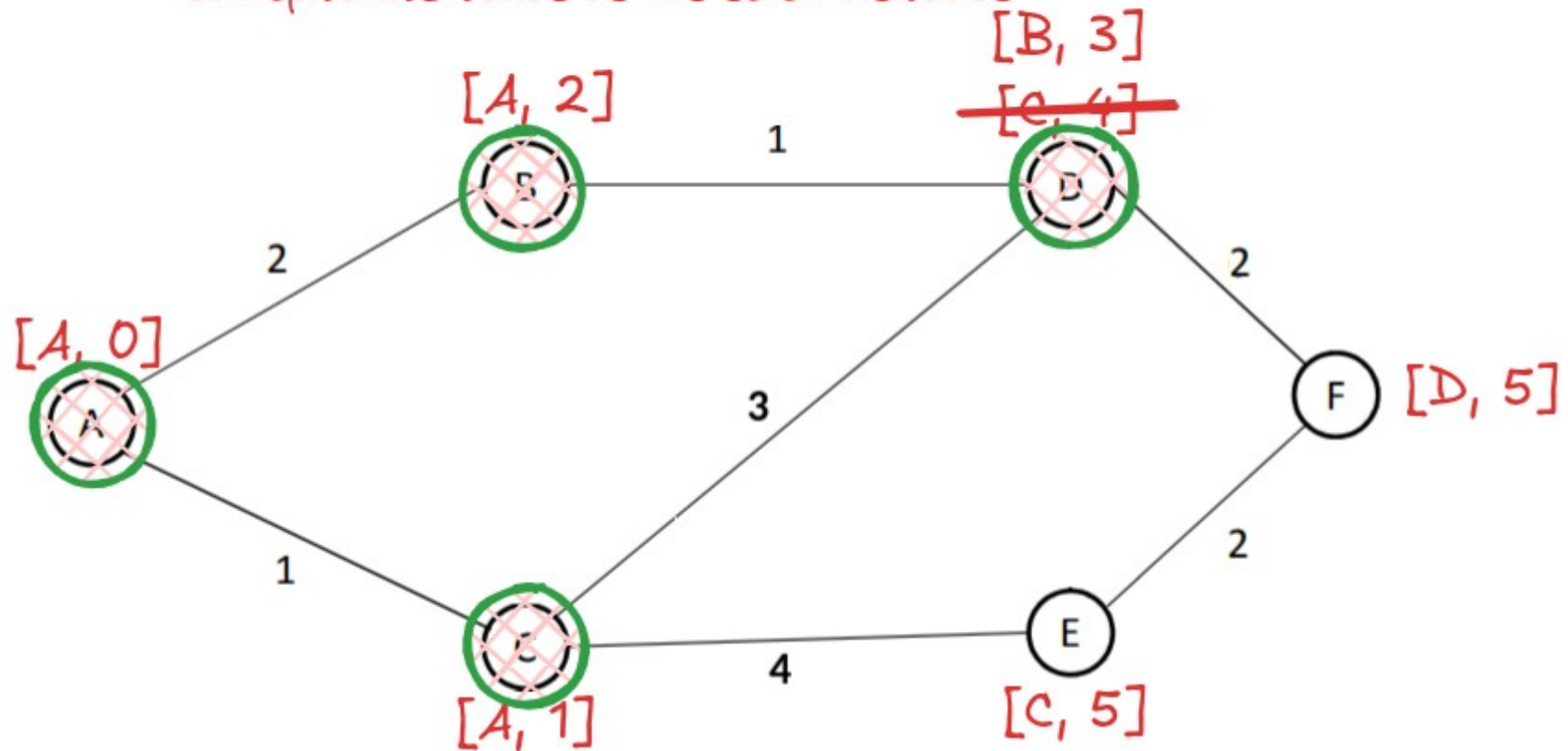
- X 1. Selecione o nó não visitado com menor distância.
- 2. Para cada vizinho, calcule a nova distância via o nó atual. Atualize a distância do vizinho se a nova for menor.
- 3. Repita até todos os nós serem visitados





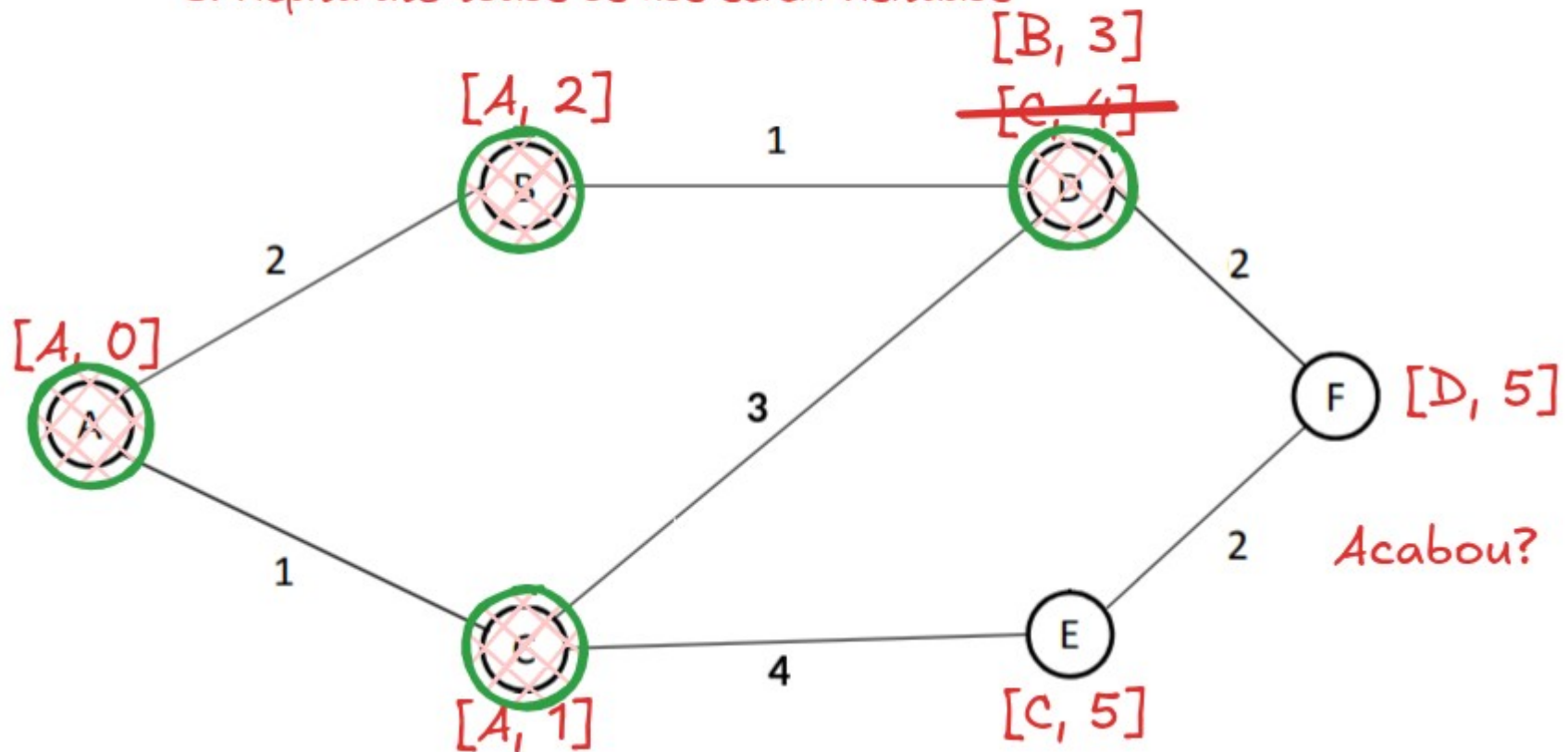
# Algoritmo de Dijkstra

1. Selecione o nó não visitado com menor distância.
- X 2. Para cada vizinho, calcule a nova distância via o nó atual.  
Atualize a distância do vizinho se a nova for menor.
3. Repita até todos os nós serem visitados



# Algoritmo de Dijkstra

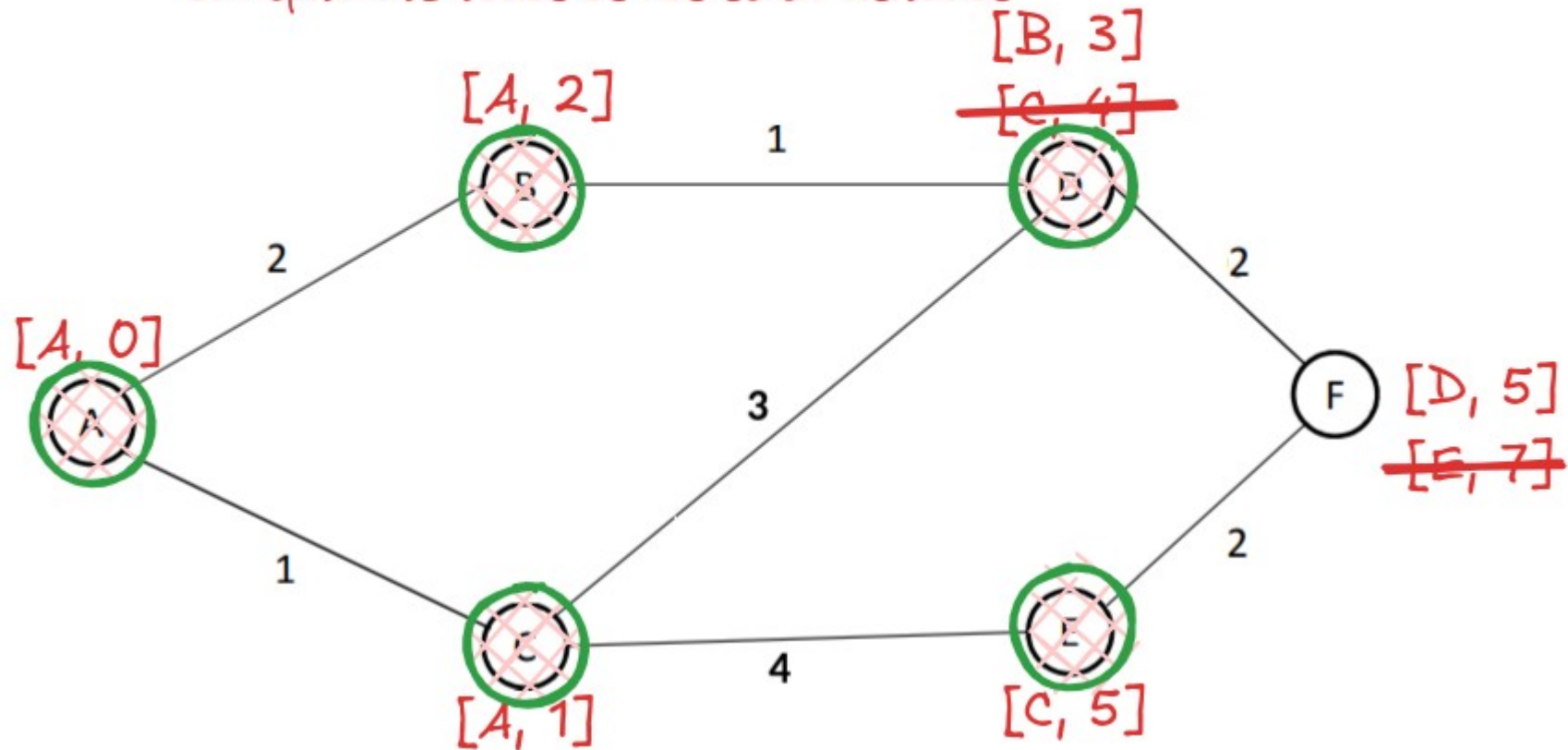
1. Selecione o nó não visitado com menor distância.
- X 2. Para cada vizinho, calcule a nova distância via o nó atual.  
Atualize a distância do vizinho se a nova for menor.
3. Repita até todos os nós serem visitados



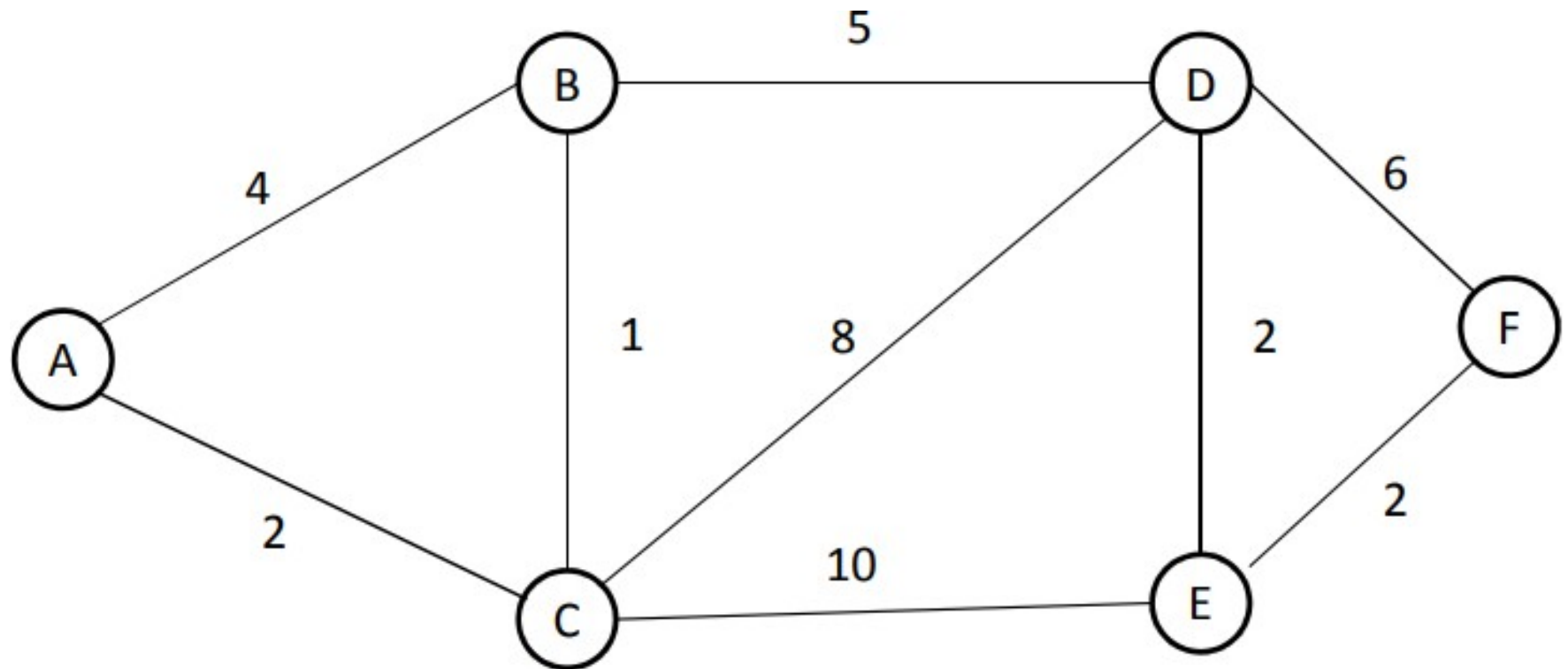


# Algoritmo de Dijkstra

- X 1. Selecione o nó não visitado com menor distância.
- X 2. Para cada vizinho, calcule a nova distância via o nó atual.  
Atualize a distância do vizinho se a nova for menor.
- 3. Repita até todos os nós serem visitados



# Algoritmo de Dijkstra



# Codificando Grafos

- [LINK DEEPNOTE](#)

# Trabalho

- Implementação do algoritmo de Dijkstra
- Monte um grafo que represente um problema real
- Aplique o algoritmo de Dijkstra
- Utilize a biblioteca *pyvis* para ilustrar o grafo
- Entrega via AVA
- **VERIFIQUE OS DETALHES NO AVA!!**

