UNINOVE
Universidade Nove de Julho
A UNINOVE É SEMPRE 10

www.belan.pro.br

Material de apoio

Aguardando a entrada de alunos

Peterson Belan

UNINOVE
Universidade Nove de Julho
A UNINOVE É SEMPRE 10

Busca no Espaço de Estados: Conceitos, Algoritmos e Aplicações

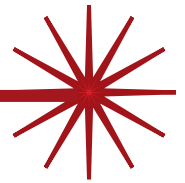
Prof. Dr. Peterson Belan
belan@uni9.pro.br

Busca no espaço de estados

É uma das técnicas mais utilizadas para resolução de problemas em Inteligência Artificial.

A ideia consiste em supor a existência de um **agente** capaz de **executar ações** que modificam o **estado corrente (atual)** de seu mundo.

Assim, dados um **estado inicial** representando a configuração atual do mundo do agente, um **conjunto de ações** que o agente é capaz de executar e uma descrição do **estado final (meta)** que se deseja atingir, a **solução do problema** consiste em uma **sequência de ações** que, quando executada, leva o agente do **estado inicial** até **estado final**.

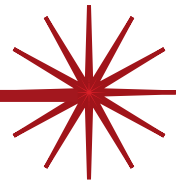


Representação de Problemas de Busca

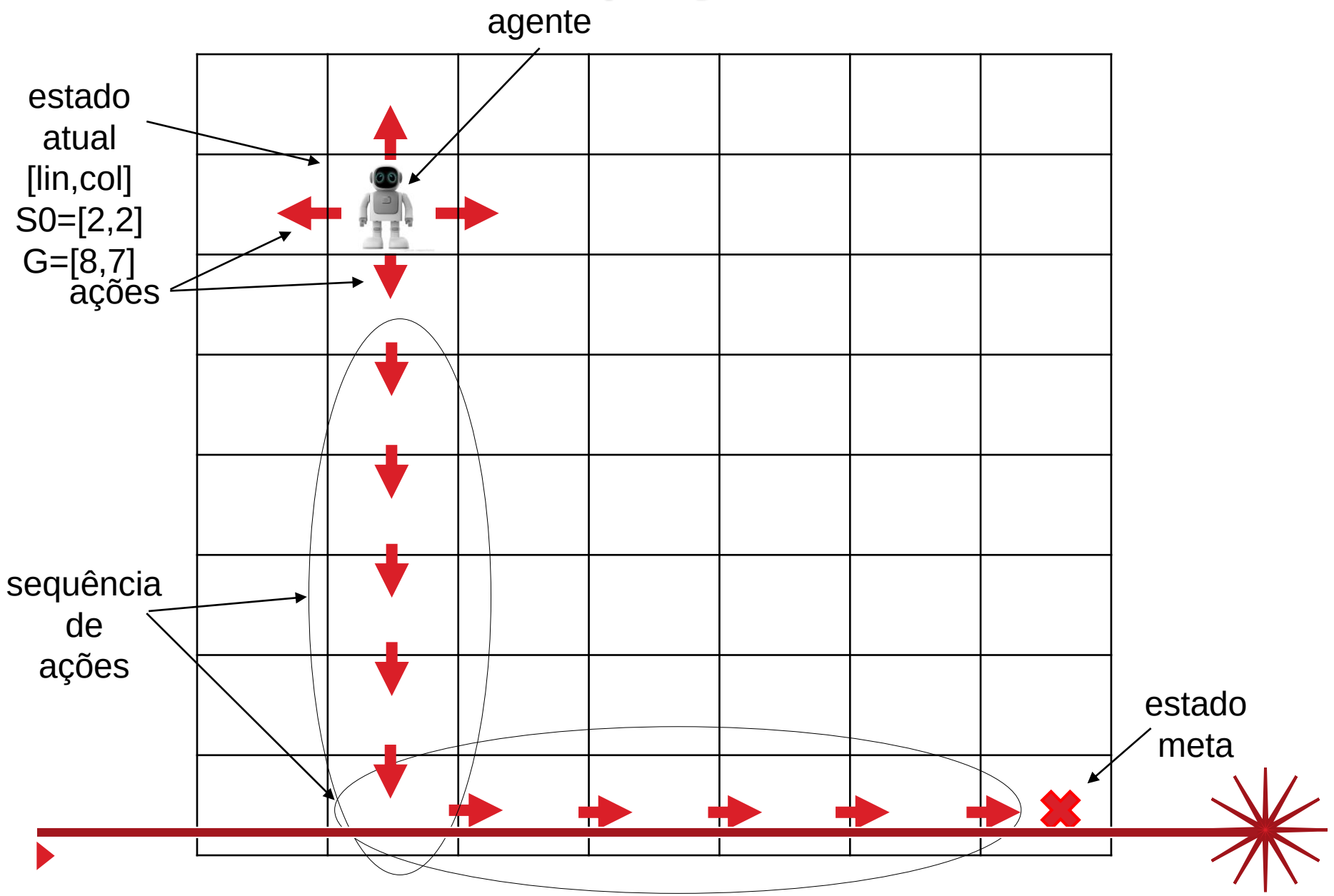
Para representar um problema como uma busca no espaço de estados, em geral, é preciso definir:

- Estados: s (o que representa?)
- Espaço de Estados: S
- Estado Inicial: s_0
- Estado(s) Final(is): G
- Conjunto de Ações: A

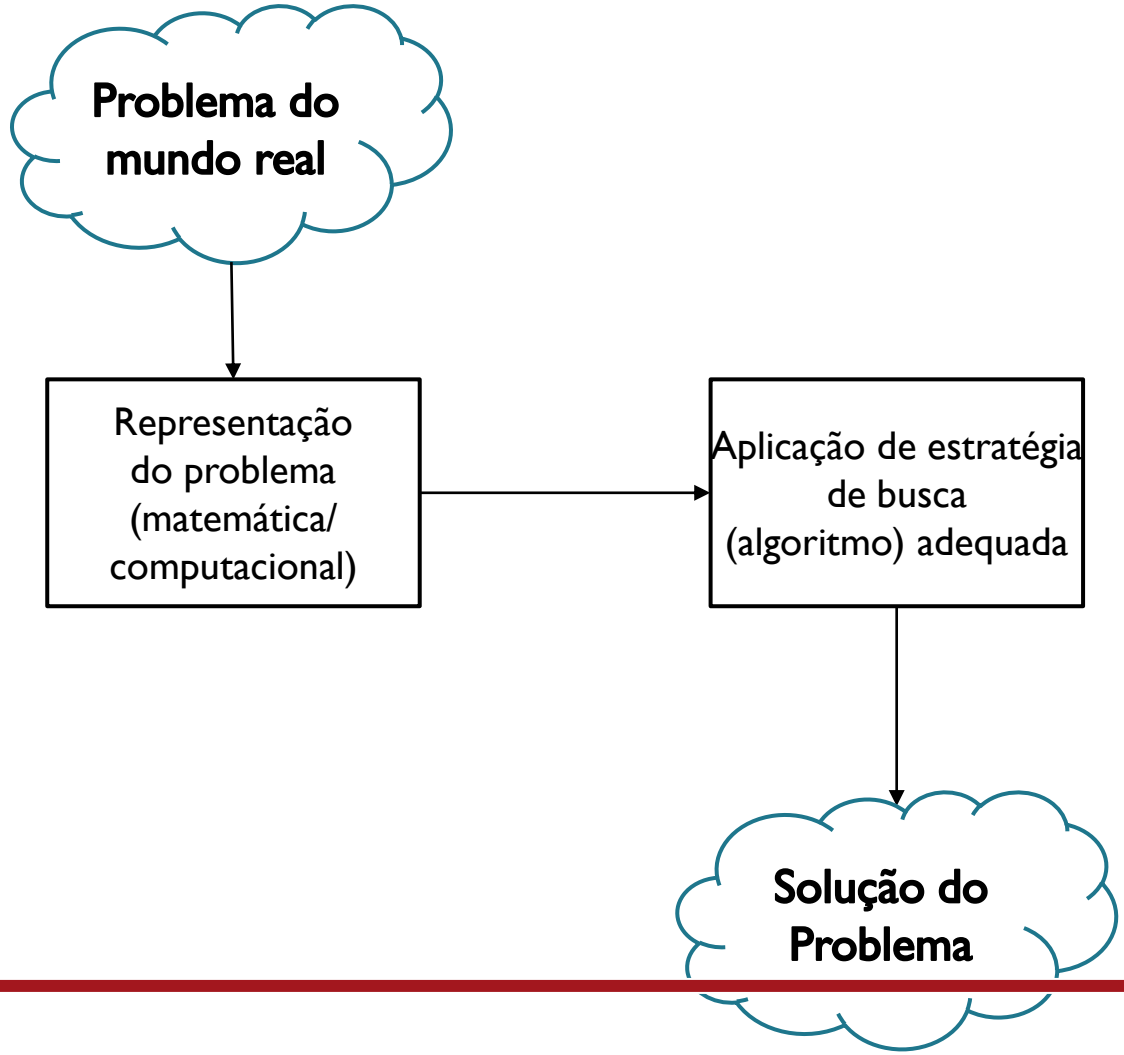
Uma ação $a \in A$ pode ser descrita na forma: **ação(α,α')$\leftarrow\beta$**
onde: α *é o estado atual*,α' *é um estado sucessor e* β é o conjunto de condições *que devem ser obedecidas*



Busca no espaço de estados



Resolução de Problemas por meio de busca no espaço de estados



O mundo do aspirador

Como exemplo, vamos considerar o Mundo do Aspirador [4]. Nesse mundo, o agente ´e um aspirador cuja função ´e limpar as salas de um edifício. Numa versão bastante simplificada, vamos supor que o mundo desse agente seja composto de apenas duas salas, cada uma delas podendo estar limpa ou suja, e que o aspirador seja capaz de executar apenas três ações: *entrarSala1, entrarSala2* e *aspirar*

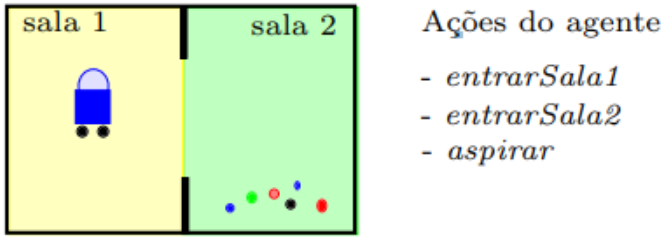


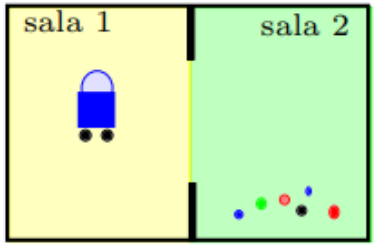
Figura 1. Mundo do Aspirador Simplificado

Russell, S. and Norvig, P. Artificial Intelligence - A Modern Approach, Prentice-Hall, 1995

O mundo do aspirador

Estados são representados por estruturas, onde cada componente denota um atributo do estado representado. Por exemplo, no Mundo do Aspirador, cada estado pode ser representado por uma estrutura da forma [X, Y,Z], onde X ∈ {1, 2} indica a posição do aspirador, Y ∈ {0, 1} indica se a primeira sala está suja e Z ∈ {0, 1} indica se a segunda sala está suja. Dessa forma, o estado em que o aspirador encontra-se na segunda sala e apenas essa sala está suja ´e representado por [2, 0, 1].

O conjunto de estados para o Mundo do Aspirador é



Ações do agente

- *entrarSala1*
- *entrarSala2*
- *aspirar*

Figura 1. Mundo do Aspirador Simplificado

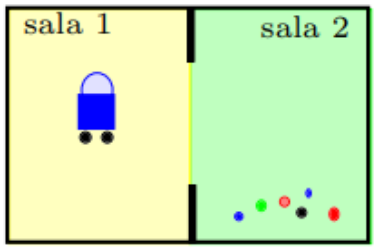
S = {[1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1], [2, 0, 0], [2, 0, 1], [2, 1, 0], [2, 1, 1]}

Russell, S. and Norvig, P. Artificial Intelligence - A Modern Approach, Prentice-Hall, 1995

O mundo do aspirador

As ações são representadas por operadores da forma oper(aspirar, [1, Y,Z], [1, 0,Z]) ← Y = 1 oper(aspirar, [2, Y,Z], [2, Y, 0]) ← Z = 1 (α, s, s') ← β, onde α é uma ação que transforma o estado s no estado s 0 , dado que a condição β esteja satisfeita. Por exemplo, a ação aspirar pode ser representada pelos seguintes operadores:

oper(aspirar, [1, Y,Z], [1, 0,Z]) ← Y = 1
oper(aspirar, [2, Y,Z], [2, Y, 0]) ← Z = 1



Ações do agente

- *entrarSala1*
- *entrarSala2*
- *aspirar*

Figura 1. Mundo do Aspirador Simplificado

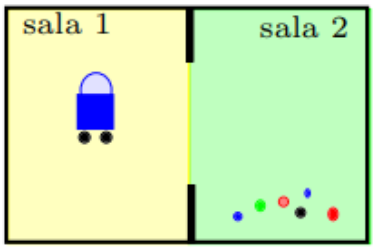
Assim, o conjunto de ações para o Mundo do Aspirador é

A = {oper(entrarSala1, [2, Y,Z], [1, Y,Z]),
oper(entrarSala2, [1, Y,Z], [2, Y,Z]),
oper(aspirar, [1, 1,Z], [1, 0,Z]),
oper(aspirar, [2, Y, 1], [2, Y, 0])}

Russell, S. and Norvig, P. Artificial Intelligence - A Modern Approach, Prentice-Hall, 1995

O mundo do aspirador

Dado um estado s ∈ S, seus estados sucessores são todos aqueles que podem ser atingidos, a partir de s, pela aplicação de um dos operadores do domínio. Por exemplo, expandindo o estado [2, 0, 1], obtemos como estados sucessores [1, 0, 1] e [2, 0, 0]. Esses estados são gerados pela aplicação dos operadores entrarSala1 e aspirar, respectivamente. Note, por exemplo, que o operador entrarSala2 não pode ser usado na expansão do estado [2, 0, 1]; já que, nesse estado, a condição implícita do operador (i.e. Y = 1) não está satisfeita.



Ações do agente

- *entrarSala1*
- *entrarSala2*
- *aspirar*

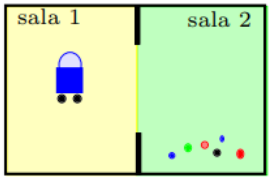
Estados sucessores

Figura 1. Mundo do Aspirador Simplificado

Russell, S. and Norvig, P. Artificial Intelligence - A Modern Approach, Prentice-Hall, 1995

O mundo do aspirador

Exercício 1 Desenhe um grafo representando o espaço de estados para o Mundo do Aspirador. Nesse grafo, cada nó será um estado do mundo e cada arco (rotulado com uma ação) será uma transição entre dois estados. Os arcos devem ser direcionados do estado para seu estado sucessor.



Ações do agente

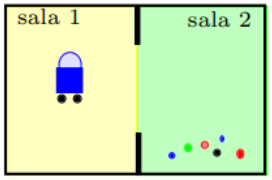
- *entrarSala1*
- *entrarSala2*
- *aspirar*

Figura 1. Mundo do Aspirador Simplificado

Russell, S. and Norvig, P. Artificial Intelligence - A Modern Approach, Prentice-Hall, 1995

O mundo do aspirador

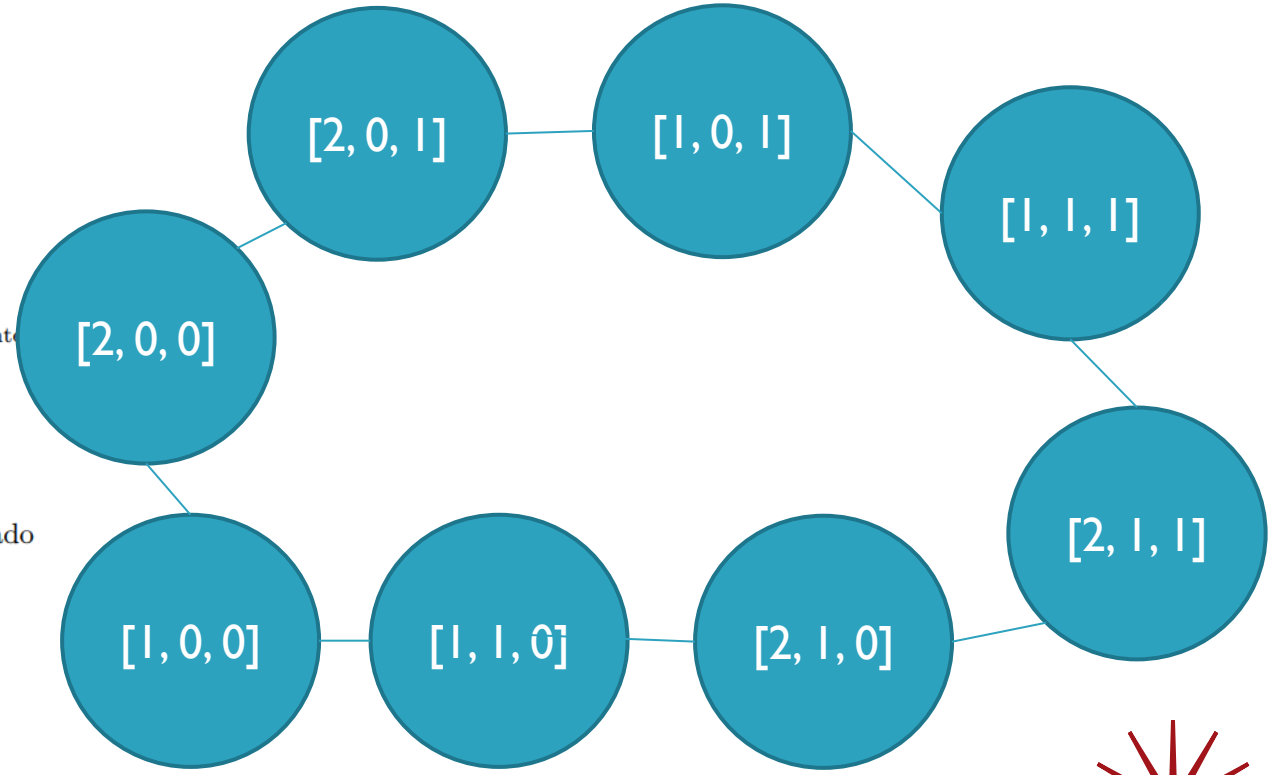
Exercício 1 Desenhe um grafo representando o espaço de estados para o Mundo do Aspirador. Nesse grafo, cada nó será um estado do mundo e cada arco (rotulado com uma ação) será uma transição entre dois estados. Os arcos devem ser direcionados do estado para seu estado sucessor.



Ações do agente

- *entrarSala1*
- *entrarSala2*
- *aspirar*

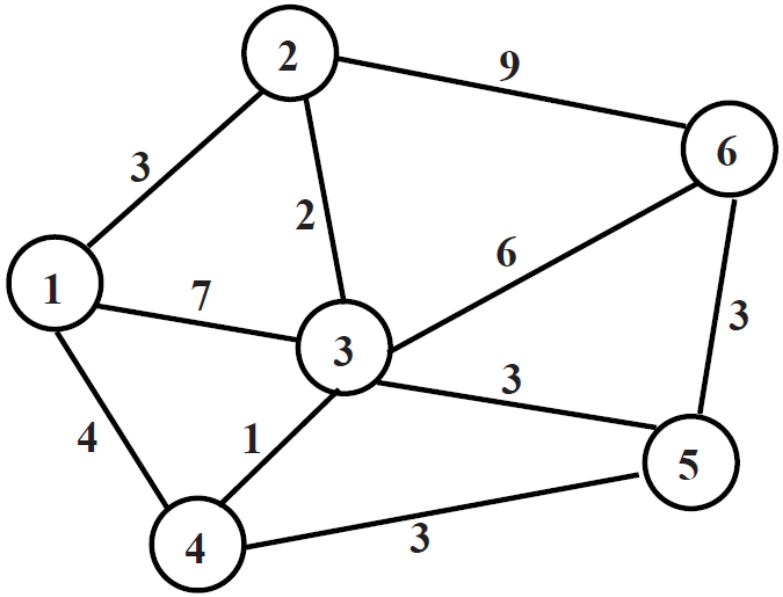
Figura 1. Mundo do Aspirador Simplificado



Russell, S. and Norvig, P. Artificial Intelligence - A Modern Approach, Prentice-Hall, 1995

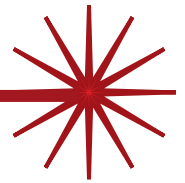
Algoritmos de Busca aplicados ao Problema do Caminho Mínimo - PCM

Definição do PCM: Dado um grafo que representa o problema investigado e o par de vértices que indicam a origem e o destino, o PCM consiste em determinar o caminho de menor custo que liga estes dois vértices.

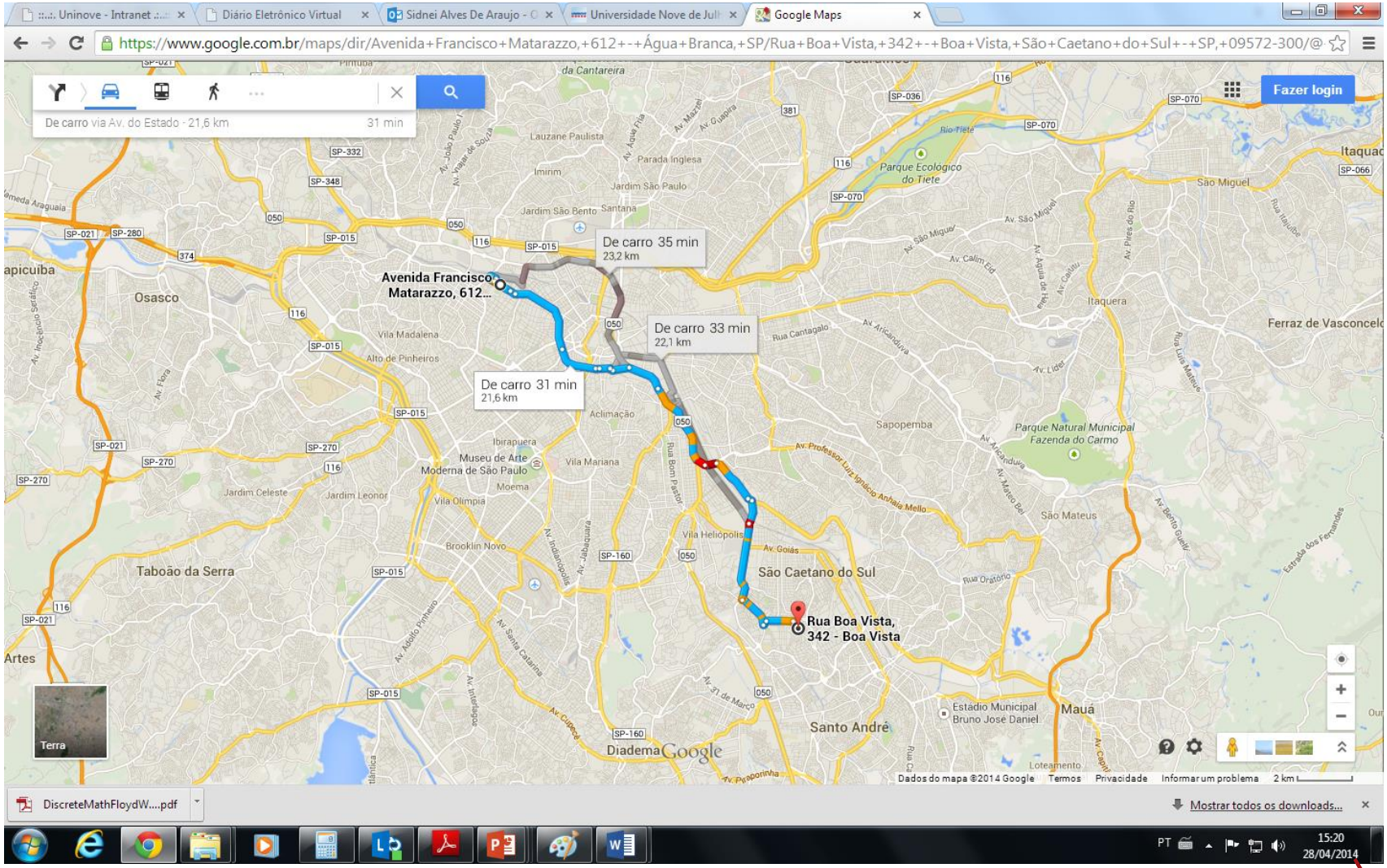


Algoritmos de Busca aplicados ao Problema do Caminho Mínimo - PCM

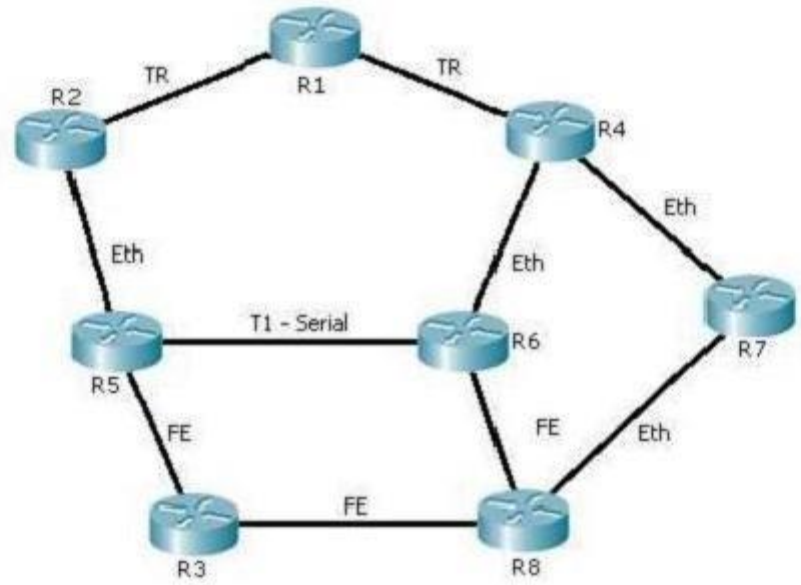
O PCM surge em um número surpreendentemente grande de contextos. Por exemplo, em uma rede de comunicação de dados, na qual os pacotes de dados trafegam a partir de suas origens até seus destinos; na transposição de um rio, onde o caminho mínimo pode levar a economia de milhões de dólares em desapropriações e tempo de projeto, em uma rede de transporte na qual um veículo ou pessoa deve fazer um percurso de um ponto a outro com o menor tempo de viagem, etc .



Exemplo 1


Google Maps
https://www.google.com.br/maps/dir/Avenida+Francisco+Matarazzo,+612+-+Água+Branca,+SP/Rua+Boa+Vista,+342+-+Boa+Vista,+São+Caetano+do+Sul+-+SP,+09572-300/@
De carro via Av. do Estado - 21,6 km
31 min
Fazer login
De carro 35 min
23,2 km
De carro 33 min
22,1 km
De carro 31 min
21,6 km
Avenida Francisco Matarazzo, 612...
Rua Boa Vista, 342 - Boa Vista
Osasco
Taboão da Serra
São Caetano do Sul
Santo André
Mauá
Diadema
Itaquera
Sapopemba
Parque Ecológico do Tietê
Parque Natural Municipal Fazenda do Carmo
Museu de Arte Moderna de São Paulo
Estádio Municipal Bruno José Daniel
Terra
Dados do mapa ©2014 Google
Termos
Privacidade
Informar um problema
2 km
DiscreteMathFloydW....pdf
Mostrar todos os downloads...
PT
15:20
28/04/2014

Exemplo 2 (roteamento em redes)



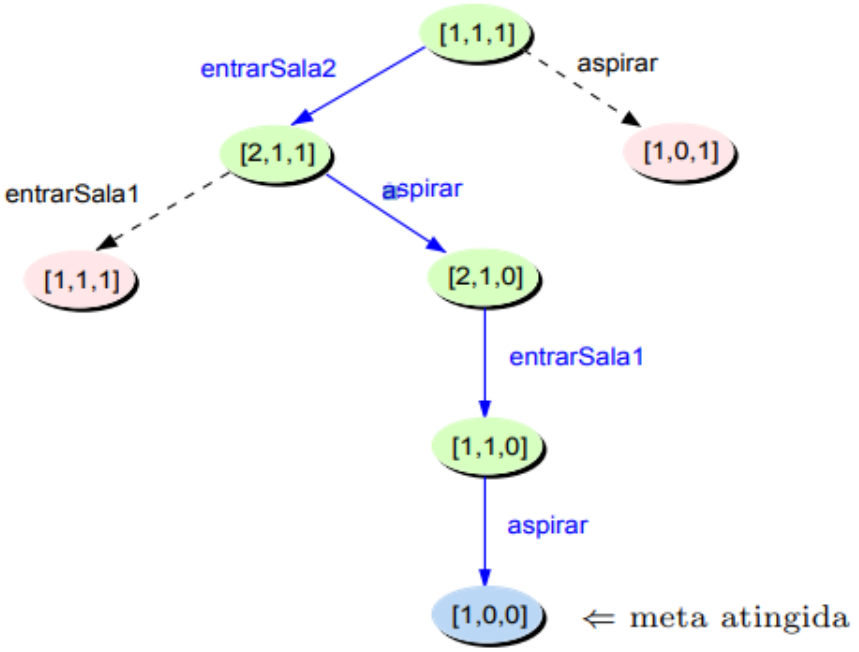
Esse diagrama de rede apresenta oito roteadores (R1 a R8) interligados por tecnologias de rede diferentes. As ligações entre os roteadores foram identificadas com uma sigla referente à tecnologia de rede utilizada. **TR** refere-se à tecnologia Token Ring = 16 Mbps, **Eth** refere-se a Ethernet = 10Mbps, **FE** refere-se a Fast Ethernet = 100Mbps, **T1** refere-se a T1 (serial) = 1,544Mbps

Exemplo 3

Busca não determinística

Passos para construção da árvore:

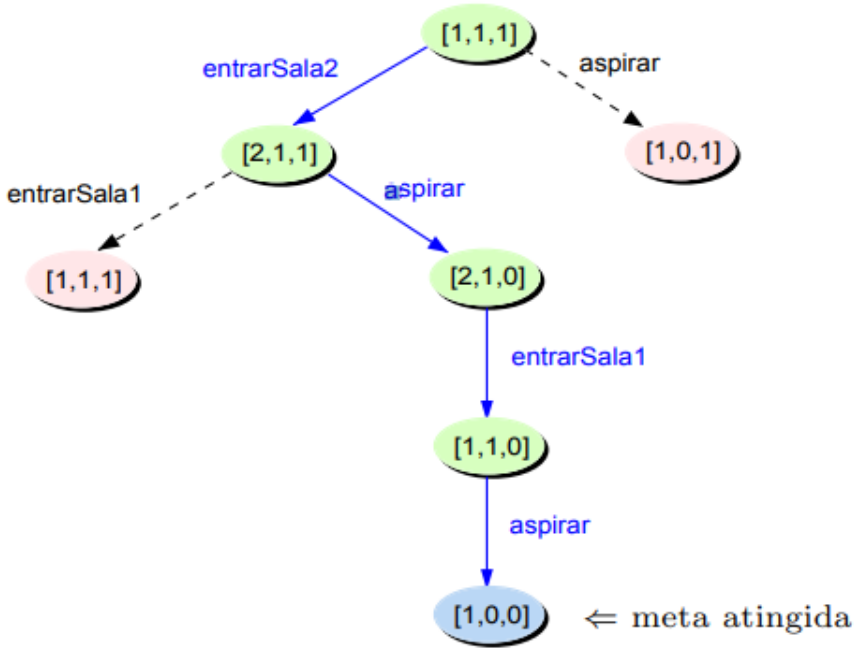
1. Sejam A o conjunto de ações para o Mundo do Aspirador, s0 = [1, 1, 1] e G = {[1, 0, 0], [2, 0, 0]}. O rastreamento da chamada Busca(A, s0, G) pode produzir, por exemplo, a árvore de busca



Busca não determinística

Passos para construção da árvore:

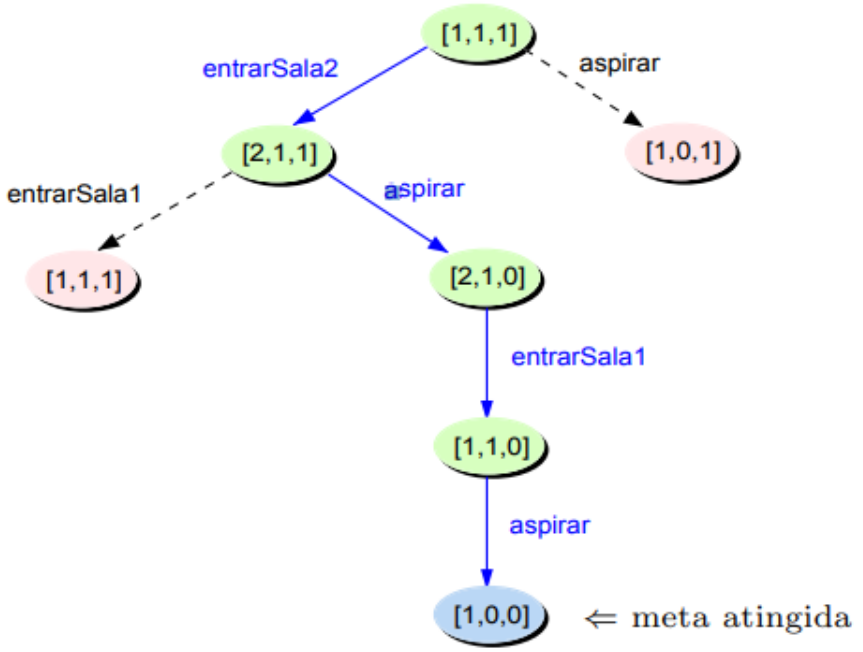
Na primeira iteração do laco no algoritmo Busca, temos Σ = {[1, 1, 1]}. Então, quando a função remove ´e chamada, a única escolha possível é s = [1, 1, 1]. Entretanto, como esse estado não ´e meta (s 6∈ G), seus sucessores ([2, 1, 1] e [1, 0, 1]) são adicionados ao conjunto Σ e a execução prossegue.



Busca não determinística

Passos para construção da árvore:

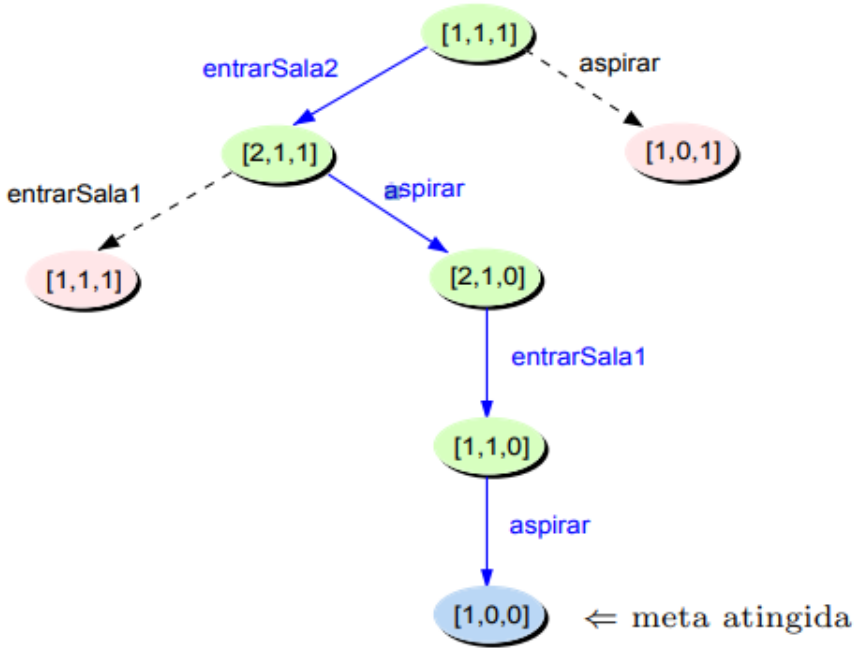
Na primeira iteração do laco no algoritmo Busca, temos Σ = {[1, 1, 1]}. Então, quando a função remove e chamada, a única escolha possível é s = [1, 1, 1]. Entretanto, como esse estado não e meta (s não pertence a G), seus sucessores ([2, 1, 1] e [1, 0, 1]) são adicionados ao conjunto Σ e a execução prossegue.



Busca não determinística

Passos para construção da árvore:

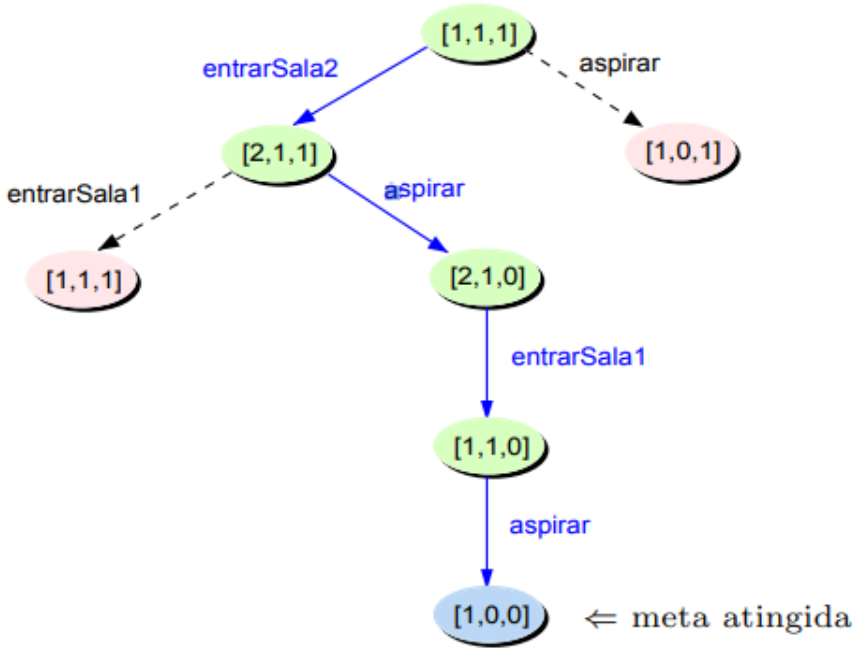
Assim, na segunda iteração do laco, teremos Σ = {[2, 1, 1], [1, 0, 1]}. Como o algoritmo e não determinístico, qualquer um desses estados poderá ser escolhido. Vamos supor que a escolha seja s = [2, 1, 1]. Como esse estado não é meta, na terceira iteração teremos Σ = {[1, 1, 1], [2, 1, 0], [1, 0, 1]}.



Busca não determinística

Passos para construção da árvore:

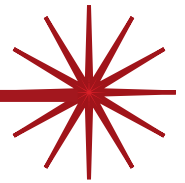
O algoritmo escolhe s = [2, 1, 0] e adiciona seu único sucessor [1, 1, 0] a Σ. Dessa forma, iniciamos a quarta iteração do laço com Σ = {[1, 1, 1], [1, 1, 0], [1, 0, 1]}. Supondo que nessa iteração a escolha seja s = [1, 1, 0], Σ passa a ser o conjunto {[1, 1, 1], [1, 0, 0], [1, 0, 1]}. Finalmente, se na quinta iteração a escolha for s = [1, 0, 0], como esse ´e um estado meta, a busca termina e o caminho [entrarSala2, aspirar, entrarSala1, aspirar] É devolvida como solução do problema



Algoritmos (estratégias) de Busca

Há diversos algoritmos de busca. Entre eles, Largura e em Profundidade, denominados algoritmos de busca não informada por não levarem em conta a qualidade da solução encontrada, e os algoritmos de busca informada apresentados abaixo:

- Busca pelo Menor Custo
- Algoritmo de Dijkstra
- Busca pela Melhor Estimativa (Greedy Search)
- Busca A*



Algoritmos de busca Informada

✓Menor custo

- Leva em conta a função de custo **f(s)=g(s)**, onde **g(s)** é o custo acumulado do caminho que vai do o estado inicial até o estado corrente;
- É lento, mas **garante encontrar solução ótima**

✓Dijkstra

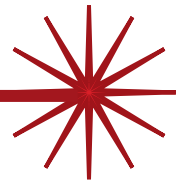
- Leva em conta a função de custo **f(s)=g(s)**;
- Otimiza o algoritmo "menor custo" por meio de **podas**;
- **Garante encontrar solução ótima**

✓Melhor estimativa (busca gulosa)

- Leva em conta a função de custo **f(s)=h(s)**, onde **h(s)** é uma informação heurística (em nossos exercícios, estimativa de distância até o nó objetivo)
- É rápido, mas **não garante encontrar solução ótima**

✓A*

- Leva em conta a função de custo **f(s)=g(s)+h(s)**
- Combina menor custo com melhor estimativa
- **Garante encontrar solução ótima**

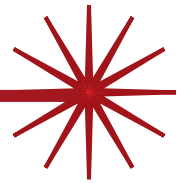


Menor Custo

Geralmente, os algoritmos de busca não especificam explicitamente o conjunto de estados de um problema. Isso acontece porque esses estados podem ser gerados sob demanda, à medida em que forem sendo encontrados durante a busca.

O algoritmo de busca pelo menor custo combina os comportamentos da busca em largura e da busca em profundidade. Nesse tipo de busca, cada vez que um estado é expandido, um custo é associado a cada um de seus sucessores. O algoritmo progride expandindo sempre o estado de menor custo, até que um estado meta seja encontrado.

O algoritmo de busca pelo menor custo é obtido através do uso de uma fila de prioridades ascendente para guardar os estados a serem expandidos.



Menor Custo

Em muitos problemas, as ações podem ter custos associados, dependendo da dificuldade que o agente tem em executá-las. Para levar essa informação em conta durante a busca, precisamos adicionar mais um campo na especificação dos operadores: oper(α, s, s' , g(α)) ← β, onde g(α) e o custo da ação α, que transforma o estado s num estado s 0 , dado que a condição β esteja satisfeita

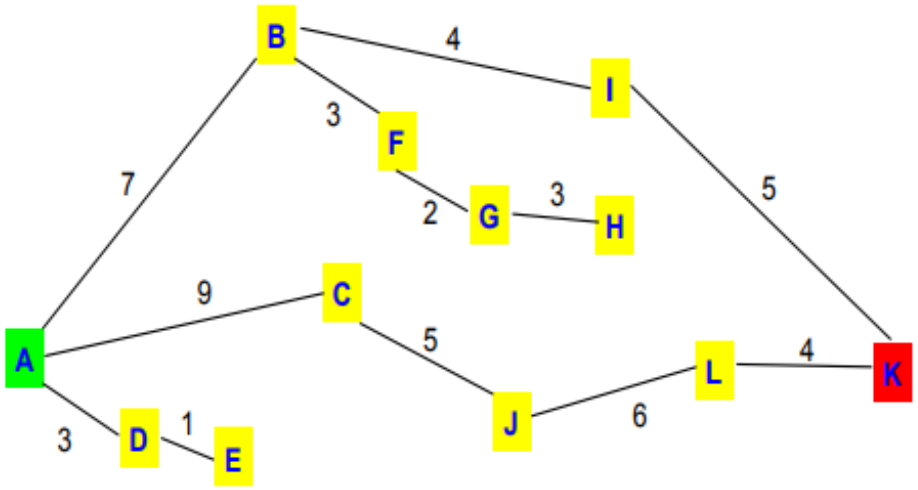


Figura 5. Mapa de vias entre cidades

Menor Custo

Como exemplo, vamos considerar o Problema das Rotas. Esse problema consiste em, dado um mapa de vias (veja a Figura 5), uma cidade de origem e uma cidade de destino, encontrar uma rota de distancia mínima entre essas duas cidades. Nesse domínio, o agente é capaz de viajar de uma cidade à outra e, portanto, suas ações podem ser especificadas conforme segue

via(a, b, 7)
via(a, c, 9)
via(a, d, 3)
via(b, f, 3)
via(b,i, 4)
via(c, j, 5)
via(d, e, 1)
via(f, g, 2)
via(g, h, 3)
via(i, k, 5)
via(j, l, 6)
via(k, l, 4)
oper(vai(A, B), A, B, C) ← via(A, B, C)
oper(vai(A, B), A, B, C) ← via(B, A, C)

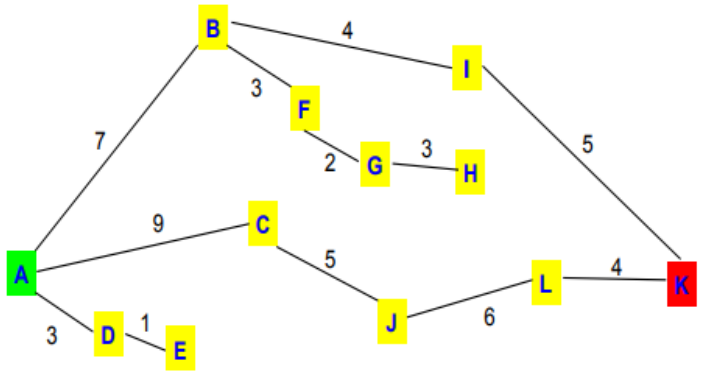


Figura 5. Mapa de vias entre cidades

Menor Custo

via(a, b, 7)
via(a, c, 9)
via(a, d, 3)
via(b, f, 3)
via(b,i, 4)
via(c, j, 5)
via(d, e, 1)
via(f, g, 2)
via(g, h, 3)
via(i, k, 5)
via(j, l, 6)
via(k, l, 4)
oper(vai(A, B), A, B, C) ← via(A, B, C)
oper(vai(A, B), A, B, C) ← via(B, A, C)

Custo de caminho. Quando as ações têm custo, o custo de um caminho $[a_1, a_2, \ldots, a_n]$ é $\sum_{i=1}^{n} g(a_i)$, onde $g(a_i)$ é o custo da ação a_i. Por exemplo, de acordo com a Figura 5, o custo do caminho $[vai(a, d), vai(d, e)]$ é $3 + 1 = 4$.

Numa árvore de busca, todo estado s está associado a um caminho que leva do estado inicial s_0 até s. Então, podemos definir o *custo de um estado numa árvore de busca* como sendo o custo do caminho que leva até ele.

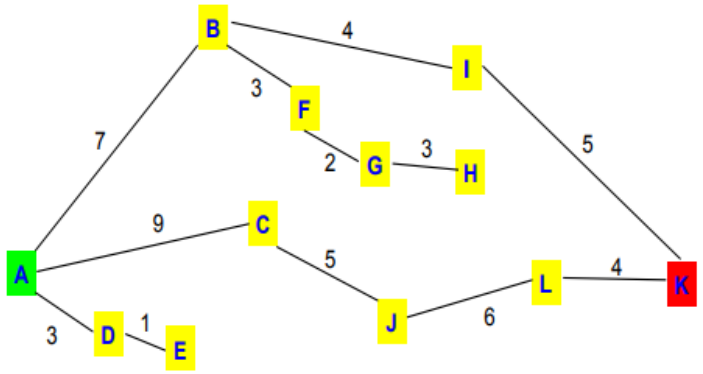
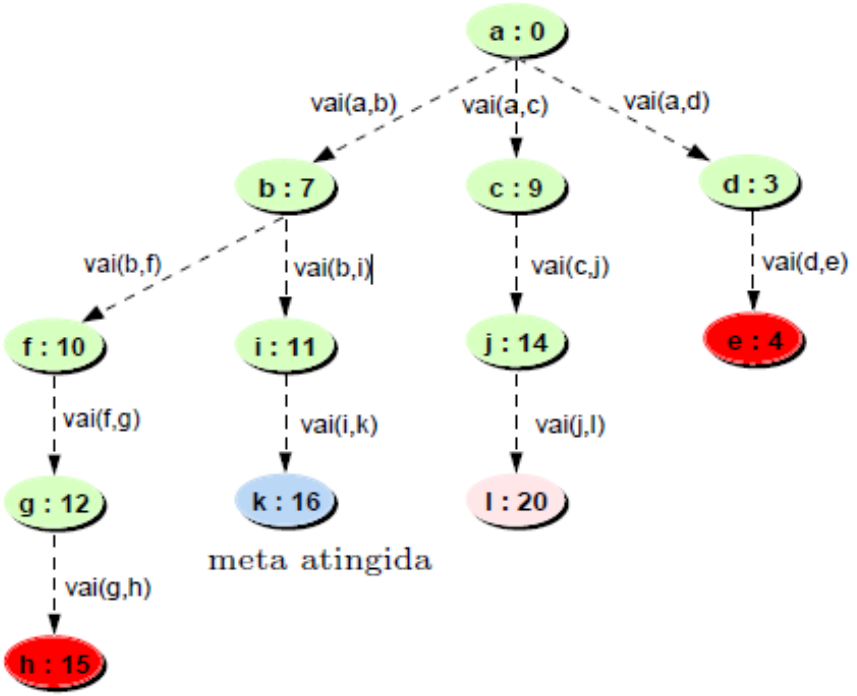


Figura 5. Mapa de vias entre cidades

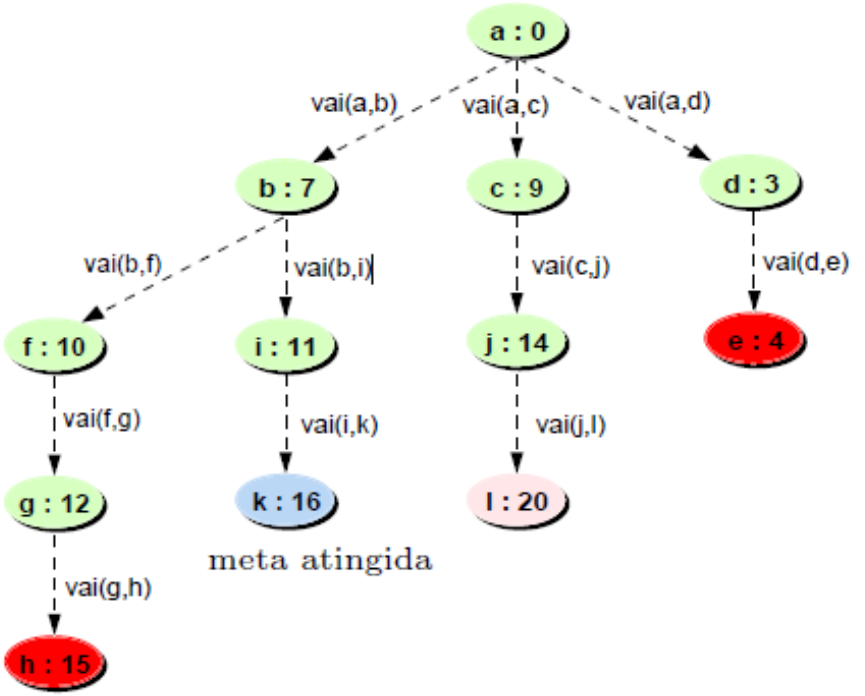
Menor Custo

Sejam A o conjunto de ações para o Problema das Rotas, s0 = a e G = f[k]. O rastreamento de BuscaMenorCusto(A; s0; G) produz a árvore de busca da, onde os estados estão numerados na ordem em que são expandidos durante a busca e cada um deles tem um custo associado.



Menor Custo

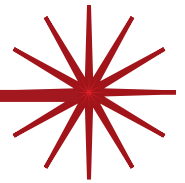
Observe que o algoritmo pula de uma ramo para outro da árvore, misturando busca em largura com busca em profundidade. Observe também que, quando o estado i : 11 é expandido, o estado meta k : 16 é gerado. Entretanto, em vez de parar a busca, o algoritmo prossegue expandindo os estados g : 12, j : 14 e h : 15, que têm custos menores. Isso é necessário porque, eventualmente, a expansão de um desses estados poderia gerar um estado k com custo menor que 16.



Menor Custo

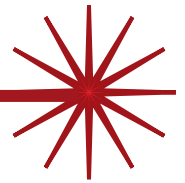
Passos para construção da árvore:

1. Coloque o estado inicial (s_0) na raiz da árvore;
2. Gere os sucessores do estado inicial e coloque-os no nível 1, com seus respectivos custos [f(s)=g(s)];
3. A partir daí, gere os sucessores do estado ***s*** de menor valor associado (usando o conjunto de ações ***A***), independentemente do seu nível ou ramificação. A busca termina quando o estado ***s*** selecionado para gerar sucessores é estado final, ou seja, quando $s \in G$.



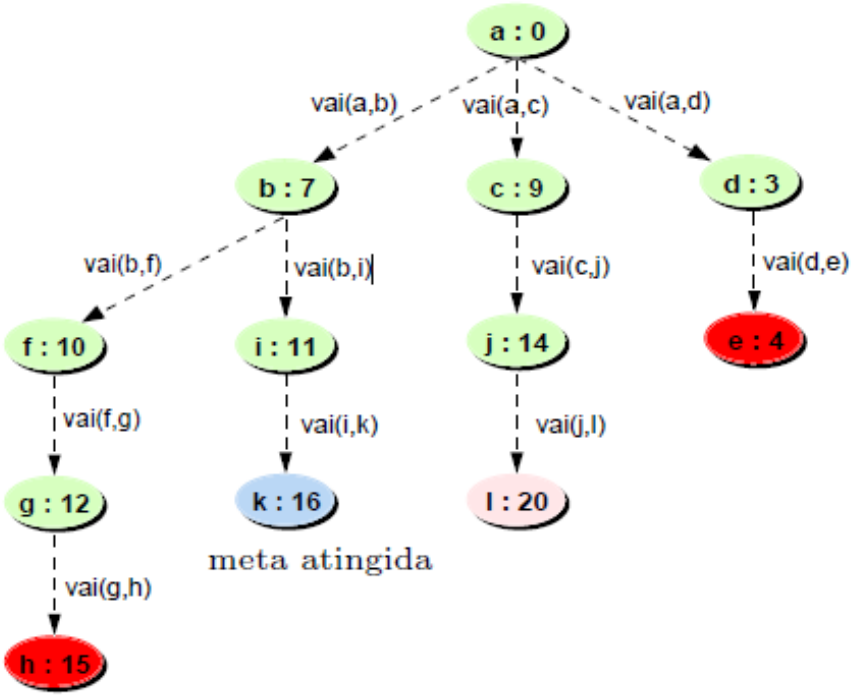
Menor Custo

```
BuscaMenorCusto(A, s0, G)
1  Γ ← ∅
2  Σ ← {s0}
3  enquanto Σ ≠ ∅ faça
4    s ← removePrimeiro(Σ)
5    se s ∈ G então devolva caminho(s)
6    Γ ← Γ ∪ {s}
7    insereEmOrdem(sucessoresG(s, A) − Γ, Σ)
8  devolva fracasso
```



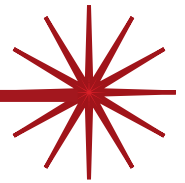
Menor Custo: exercício

Desenhe a árvore de busca produzida, para o Problema das Rotas, quando o algoritmo BuscaMenorCusto é chamado com s0 = k e G = [a].

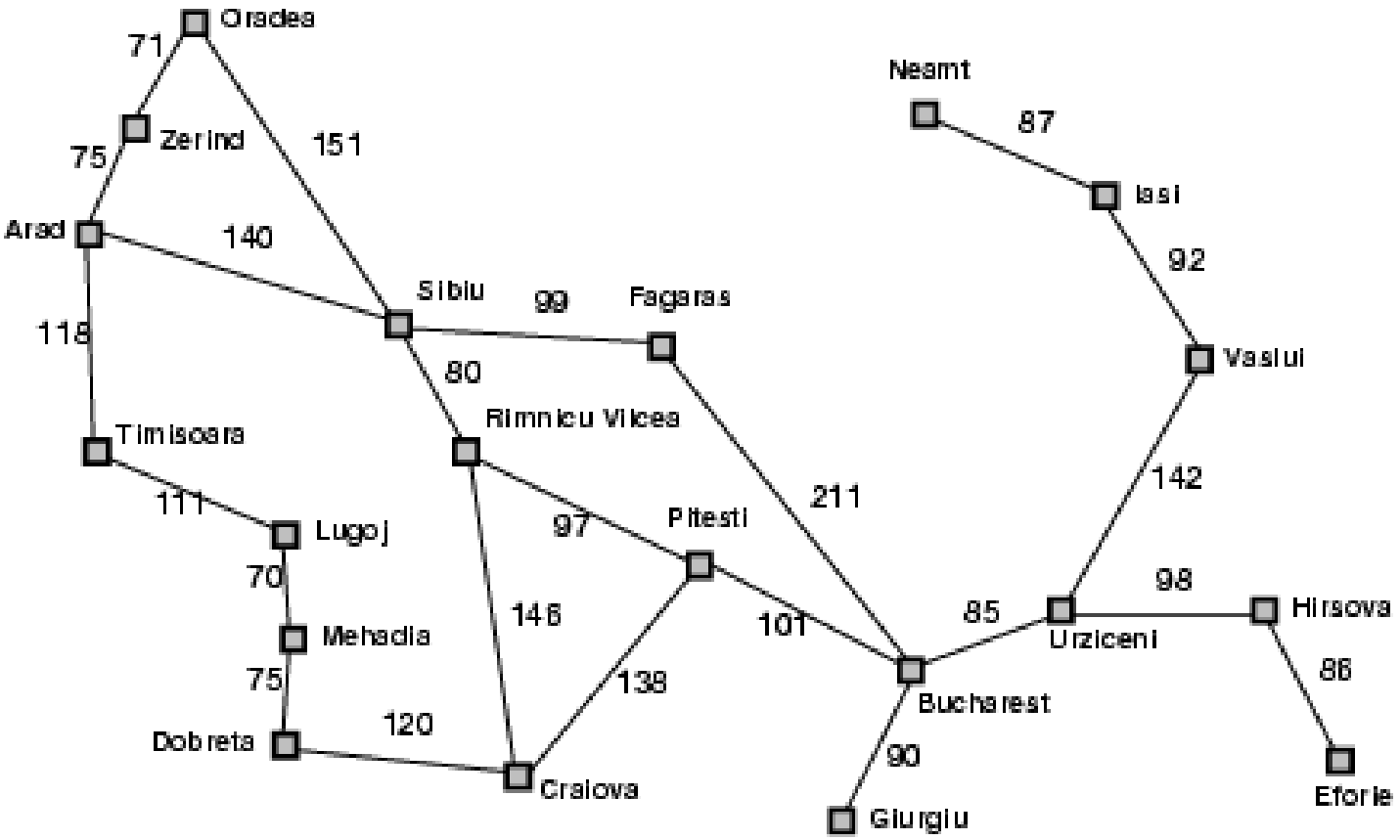


Melhor Estimativa

- Função de avaliação $f(n) = h(n)$ (heurística)
  - –Estimativa do custo de *n* até o *objetivo*
- Exemplo da Romênia. $h_{DLR}(n)$ = distância em linha reta de n até Bucareste
- Busca gulosa expande o nó que parece mais próximo ao objetivo



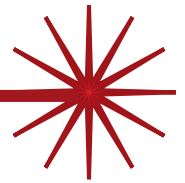
Melhor Estimativa



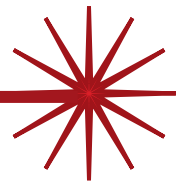
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Melhor Estimativa

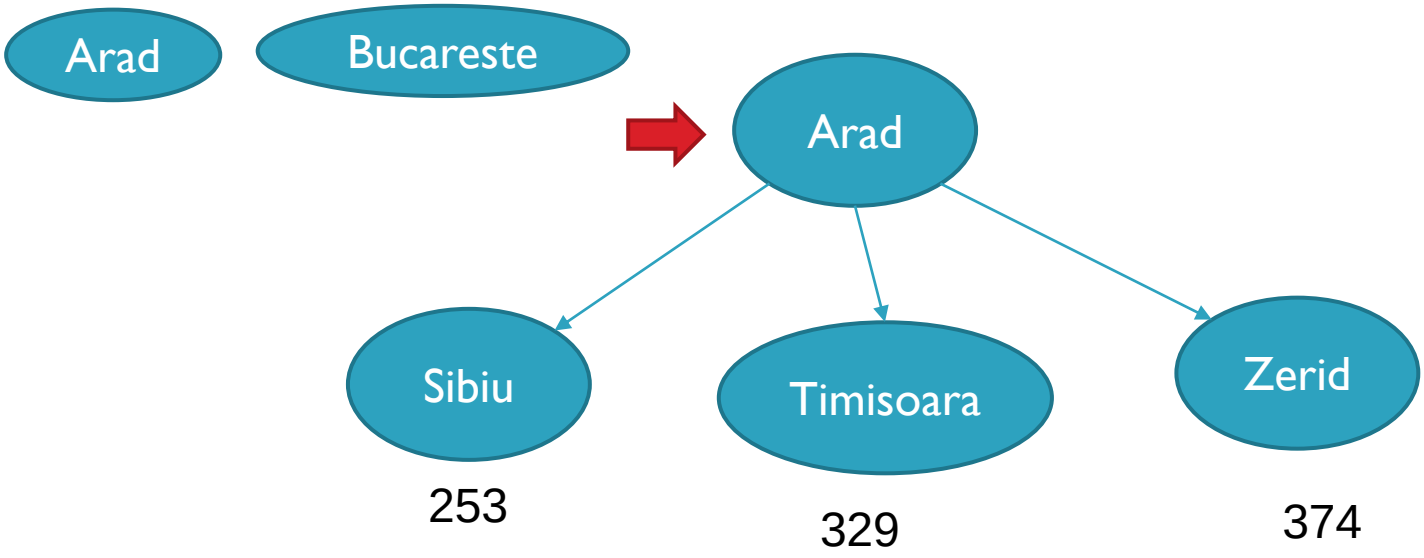
- Função de avaliação $f(n) = h(n)$ (heurística)
  - –Estimativa do custo de *n* até o *objetivo*
- Exemplo da Romênia. $h_{DLR}(n)$ = distância em linha reta de n até Bucareste
- Busca gulosa expande o nó que parece mais próximo ao objetivo



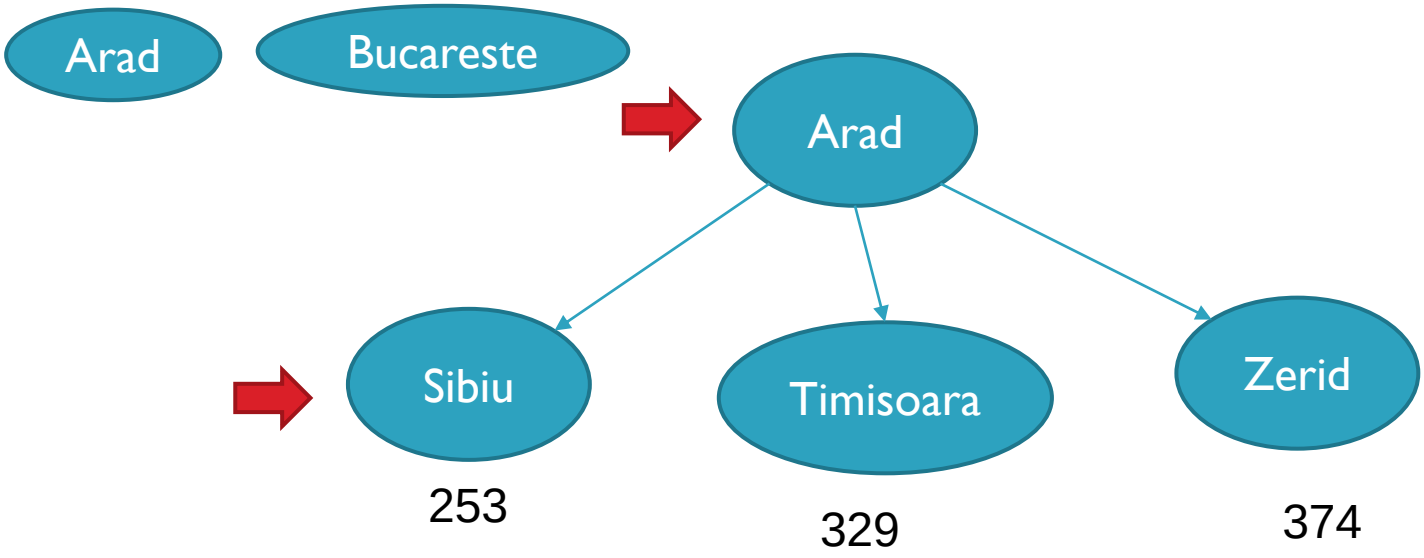
Melhor Estimativa



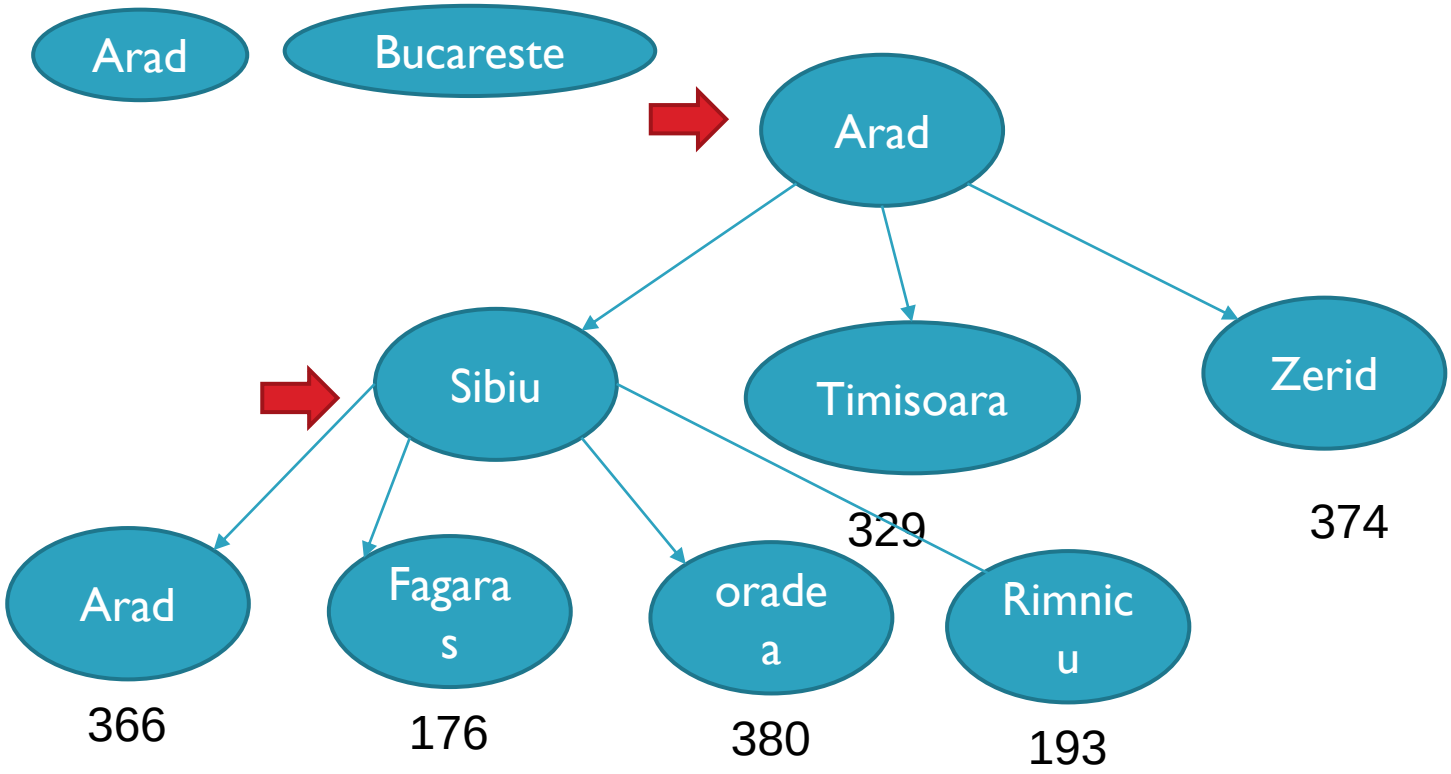
Melhor Estimativa



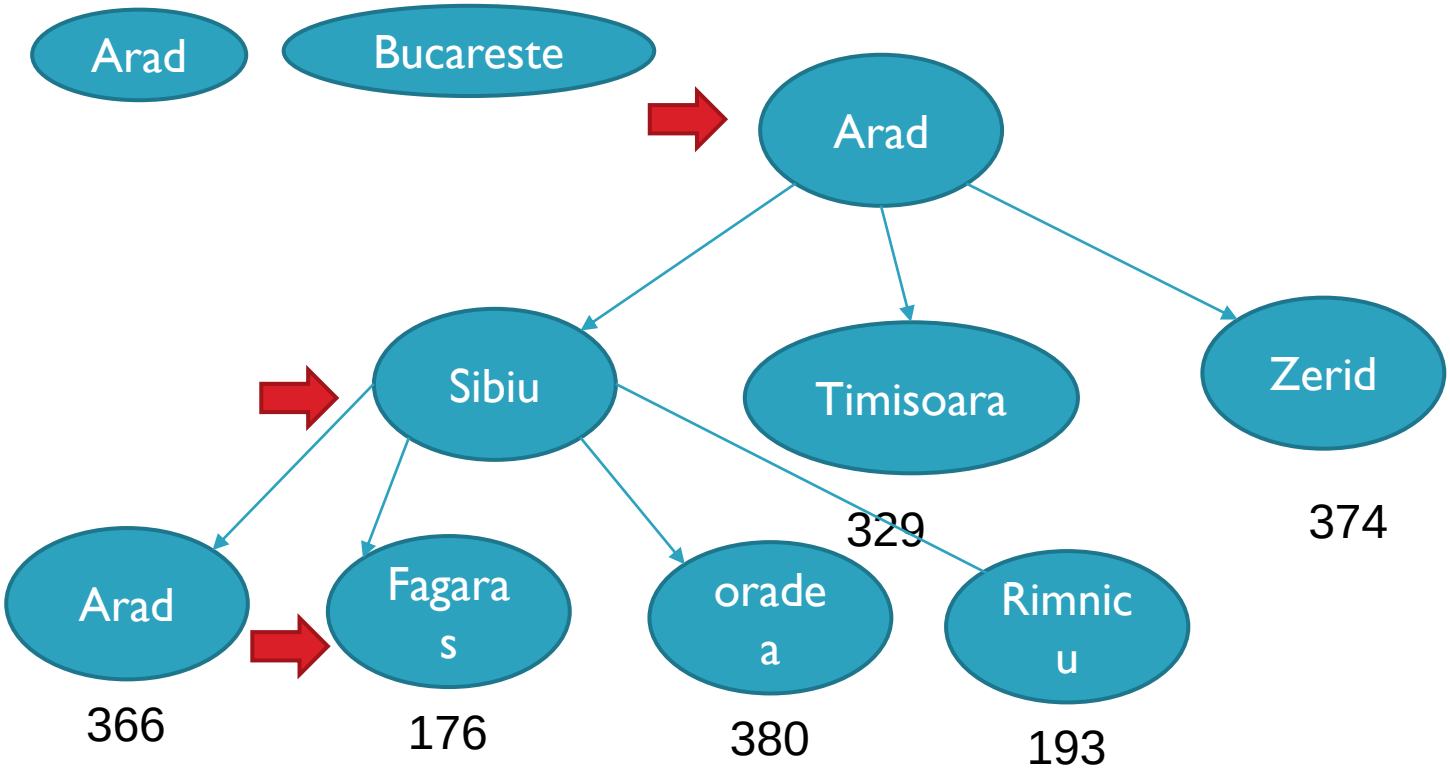
Melhor Estimativa



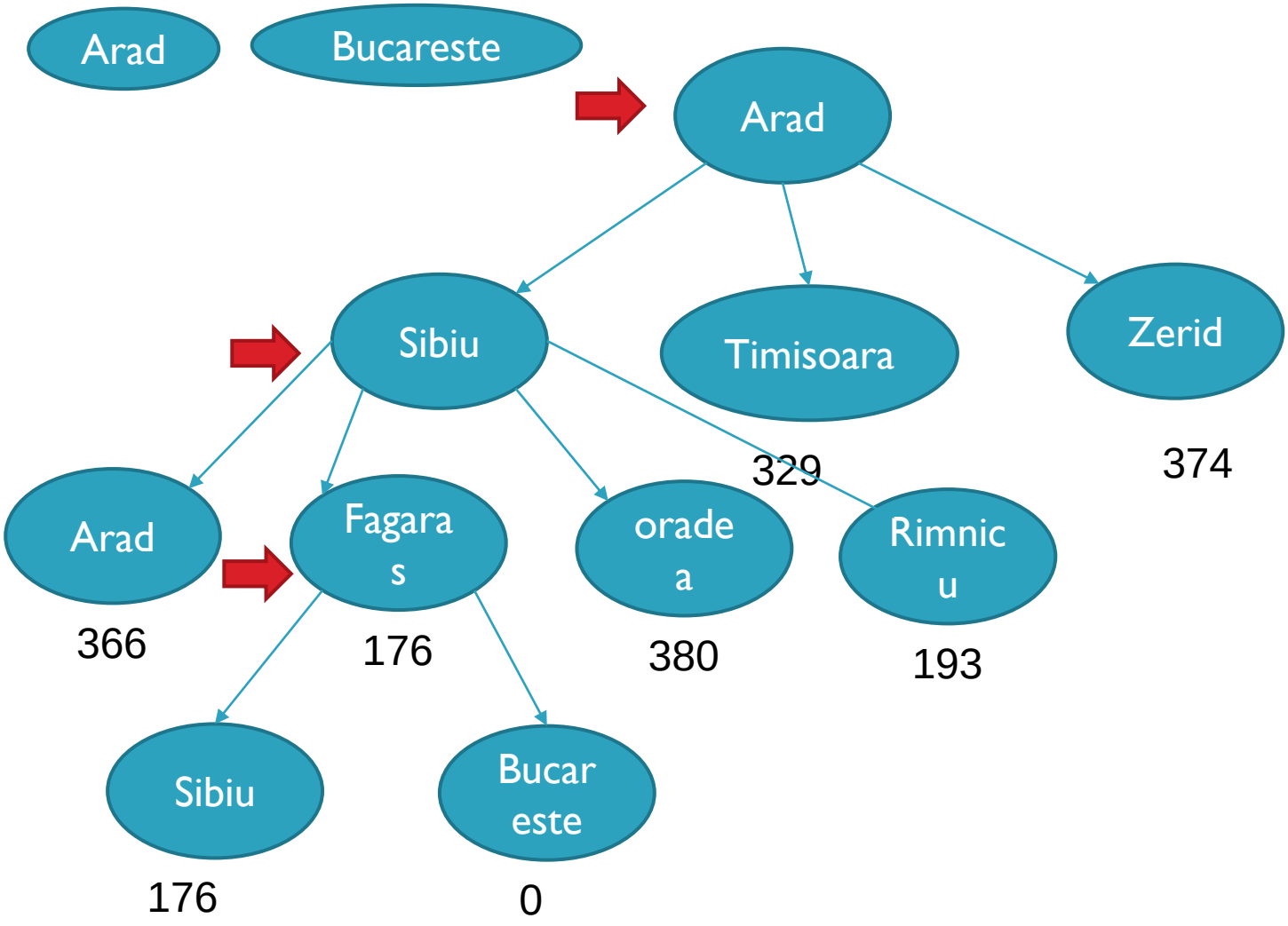
Melhor Estimativa



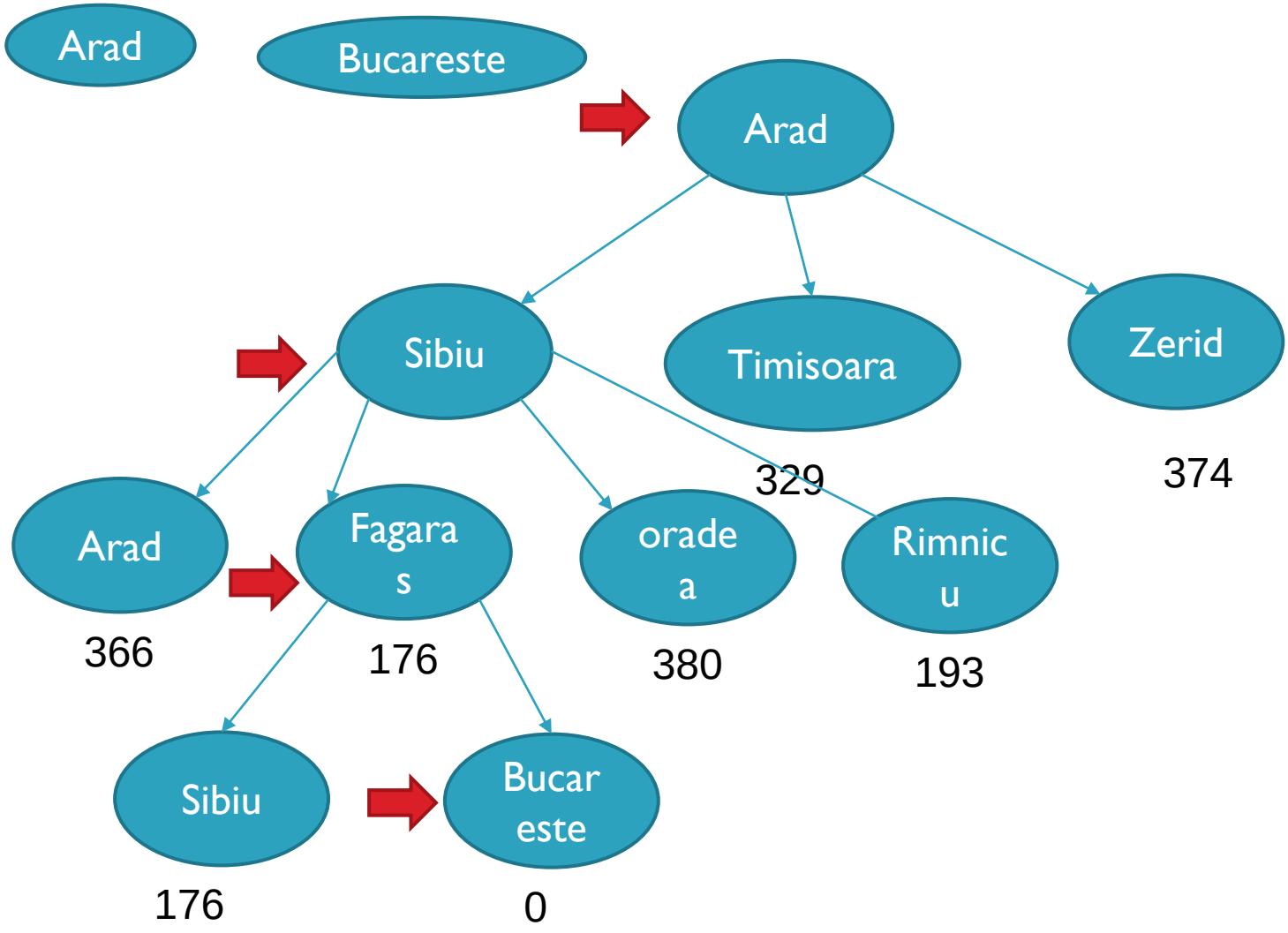
Melhor Estimativa



Melhor Estimativa

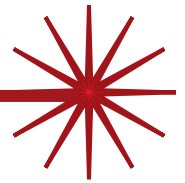


Melhor Estimativa



Melhor Estimativa

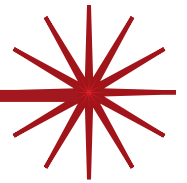
```
BuscaMelhorEstimativa(A, s_0, G)
1  Γ ← ∅
2  Σ ← {s_0}
3  enquanto Σ ≠ ∅ faça
4    s ← removePrimeiro(Σ)
5    se s ∈ G então devolva caminho(s)
6    Γ ← Γ ∪ {s}
7    insereEmOrdem(sucessoresH(s, A) − Γ, Σ)
8  devolva fracasso
```



Melhor Estimativa

Passos para construção da árvore:

1. Coloque o estado inicial (s_0) na raiz da árvore;
2. Gere os sucessores do estado inicial e coloque-os no nível 1, com seus respectivos custos [f(s)=h(s)];
3. A partir daí, gere os sucessores do estado ***s*** de menor valor associado (usando o conjunto de ações ***A***), independentemente do seu nível ou ramificação. A busca termina quando o estado ***s*** selecionado para gerar sucessores é estado final, ou seja, quando ***s*** $\in$ ***G***.



Melhor Estimativa

Completa?

–Não – pode ficar presa em loops.

–Exemplo: Iasi -> Fagaras

•Ótima?

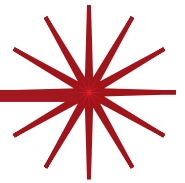
–Não

•Complexidade de Tempo?

–$O(b^m)$, mas uma boa heurística pode reduzir bastante.

•Espaço?

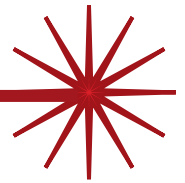
–$O(b^m)$ - Semelhante a busca em profundidade



A*

Conforme vimos, a busca pelo menor custo minimiza o custo do caminho percorrido ate o estado corrente, enquanto a busca pela melhor estimativa tenta minimizar o custo estimado do caminho a ser percorrido do estado corrente ate um estado meta. Para garantir a solução de custo mínimo, a busca pelo menor custo acaba tendo que examinar uma grande quantidade de estados no espaço de estados do problema, podendo ser muito insciente.

Por outro lado, a busca pela melhor estimativa reduz o espaço de busca consideravelmente; porem, não consegue garantir uma solução de custo mínimo. Felizmente, e possível combinar essas duas estratégias de busca num mesmo algoritmo, denominado A*



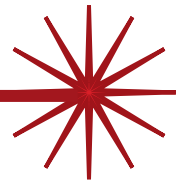
A*

Tipo mais conhecido de busca pela melhor escolha

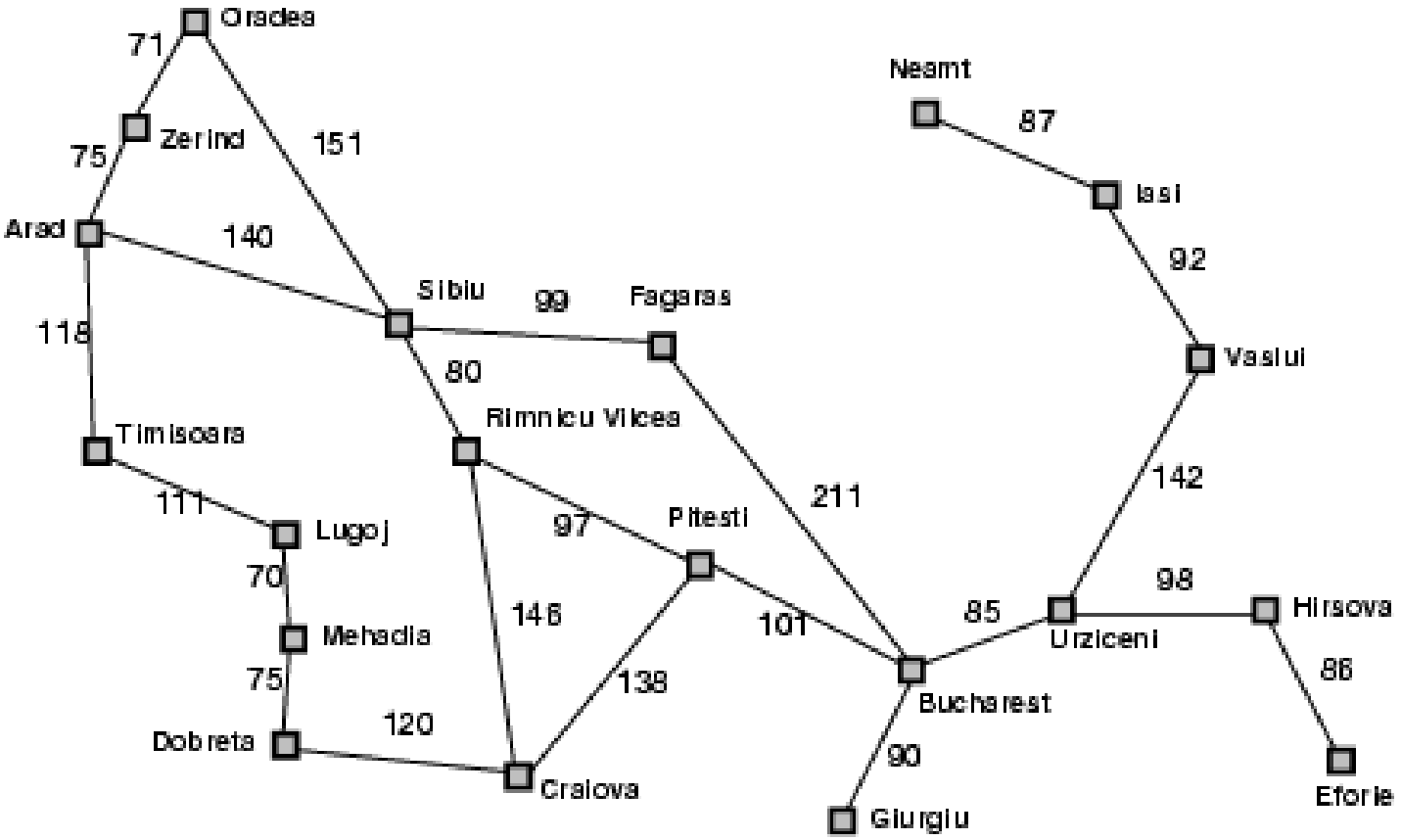
- Função de avaliação
  - $f(n) = g(n) + h(n)$

Onde

- $g(n)$ = Custo até atingir n
- $h(n)$ = Estimativa de custo do n para o objetivo
- $f(n)$ = Custo total do caminho de n para o objetivo



A*

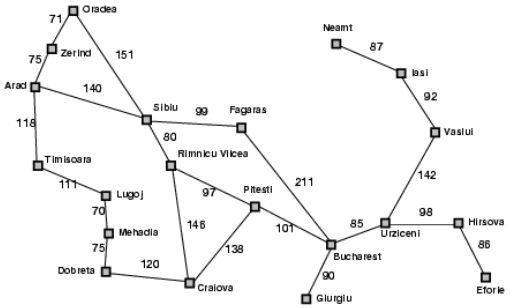


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

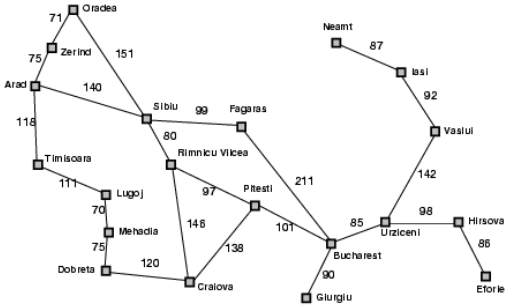
A*

Arad

Bucareste

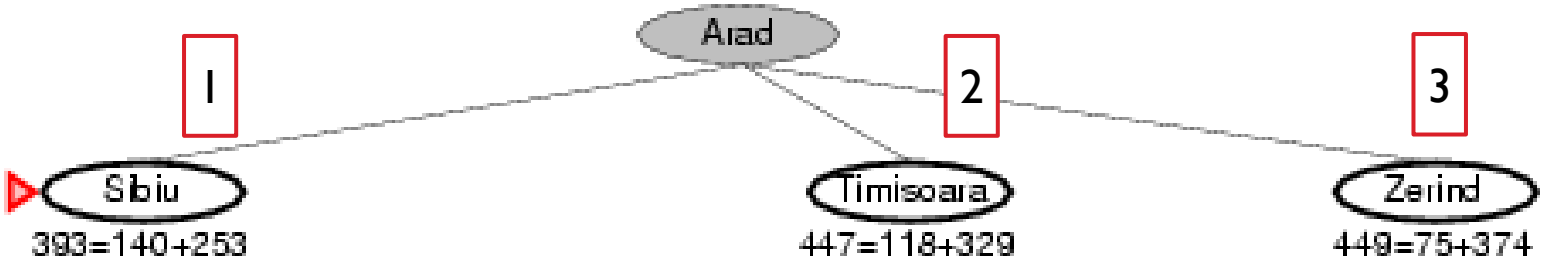


A*

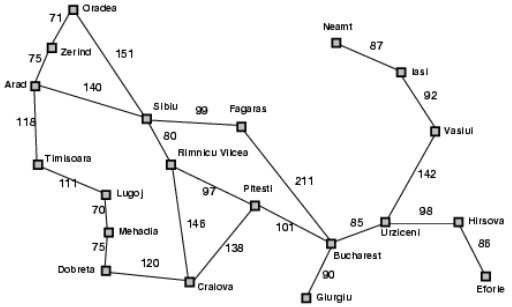


Arad

Bucareste

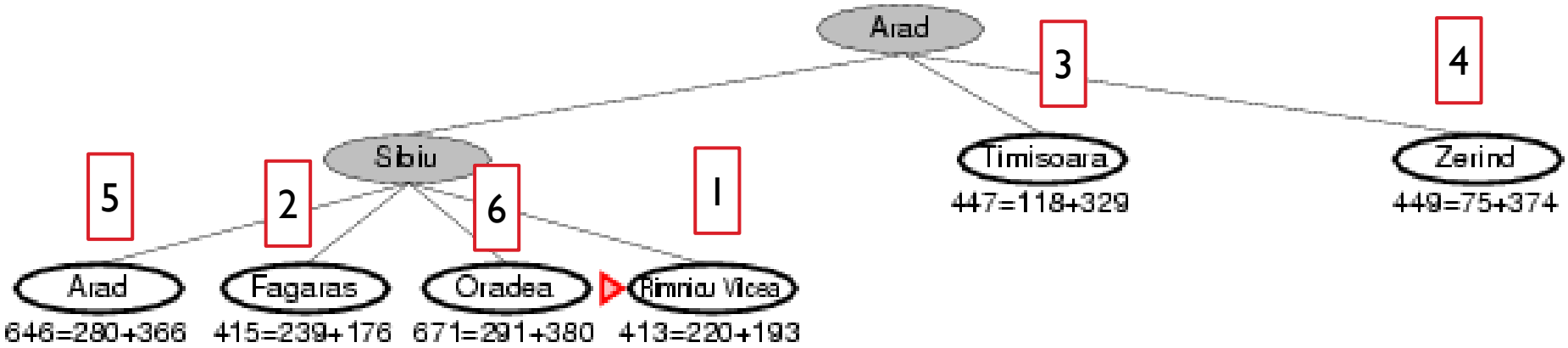


A*

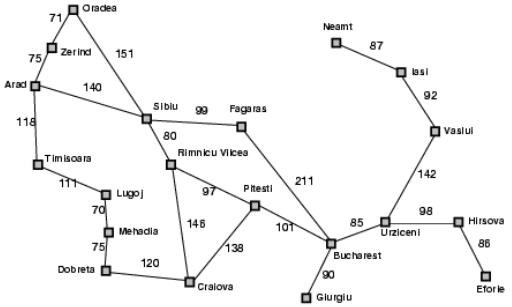


Arad

Bucareste

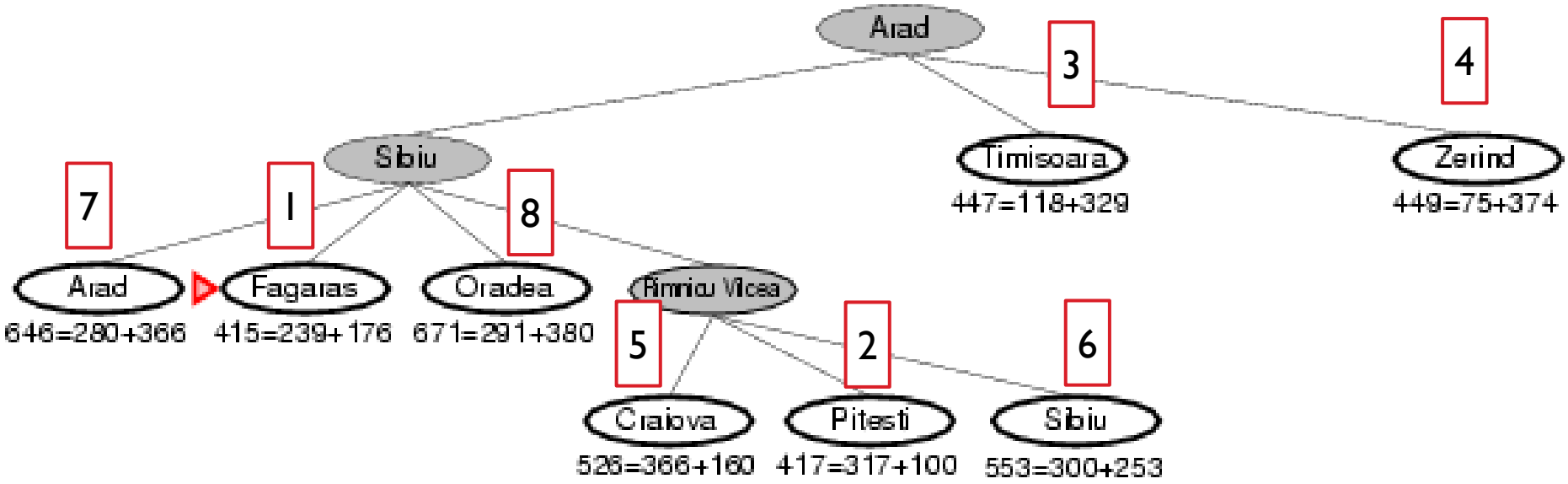


A*

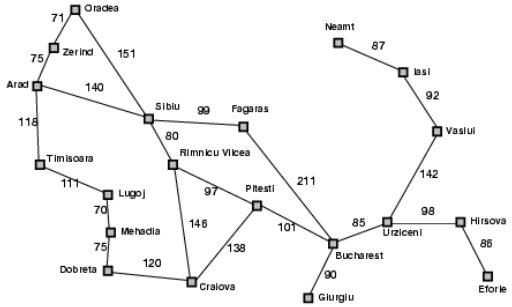


Arad

Bucareste

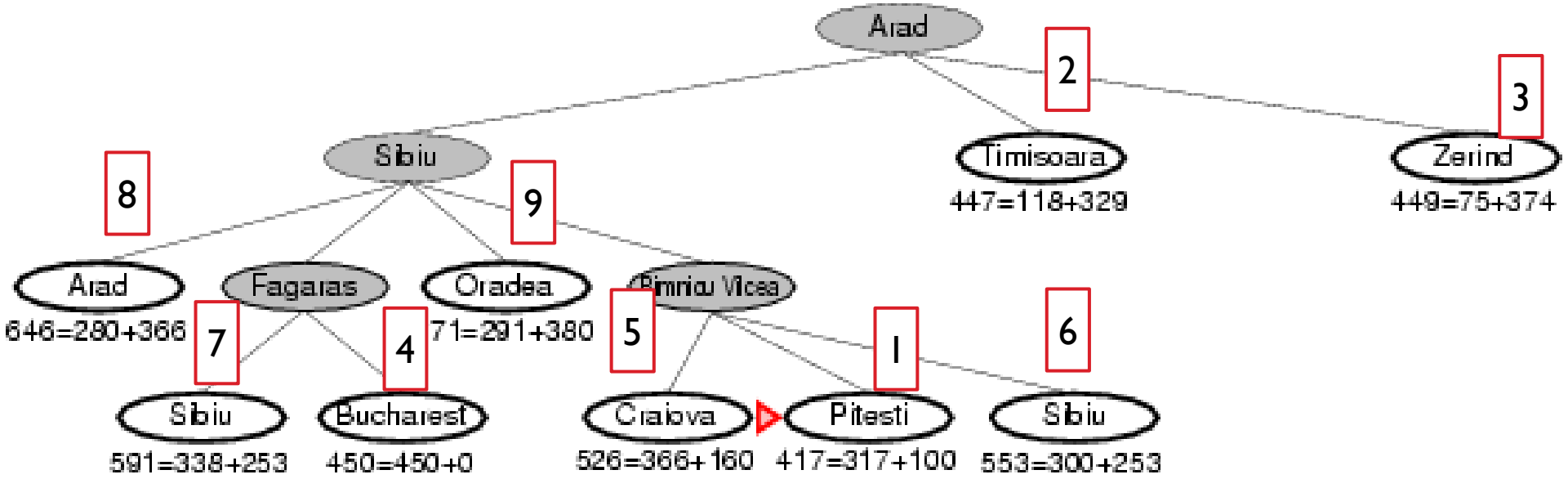


A*



Arad

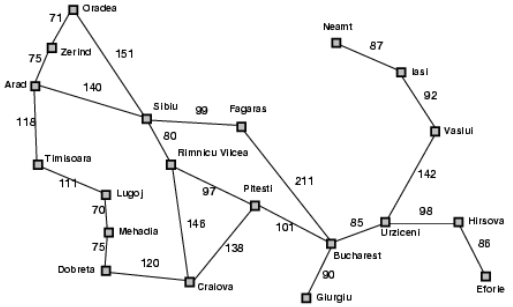
Bucareste

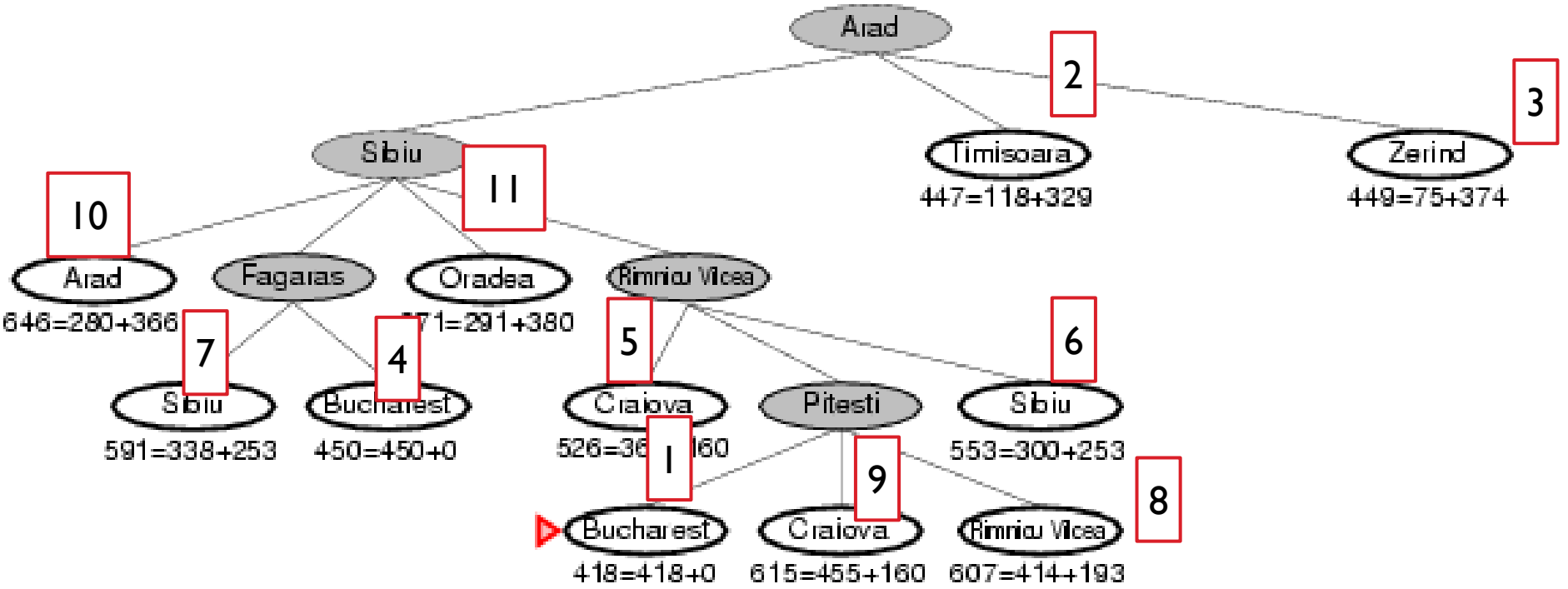


A*

Arad

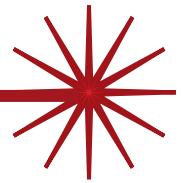
Bucareste





A*

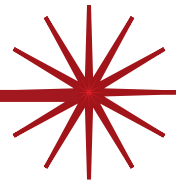
```
BUSCA A*(A, s0, G)
1  Γ ← ∅
2  Σ ← {s0}
3  enquanto Σ ≠ ∅ faça
4    s ← removePrimeiro(Σ)
5    se s ∈ G então devolva caminho(s)
6    Γ ← Γ ∪ {s}
7    insereEmOrdem(sucessoresF(s, A) − Γ, Σ)
8  devolva fracasso
```



A*

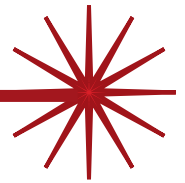
Passos para construção da árvore:

1. Coloque o estado inicial (s_0) na raiz da árvore;
2. Gere os sucessores do estado inicial e coloque-os no nível 1, com seus respectivos custos [f(s)=g(s)+h(s)];
3. A partir daí, gere os sucessores do estado ***s*** de menor valor associado (usando o conjunto de ações ***A***), independentemente do seu nível ou ramificação. A busca termina quando o estado ***s*** selecionado para gerar sucessores é estado final, ou seja, quando ***s*** $\in$ ***G***.



Dijkstra

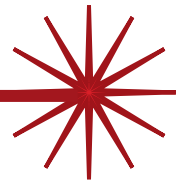
O algoritmo calcula o caminho mais curto levando em consideração o peso das arestas entre um nó e todos os demais nós do grafo.



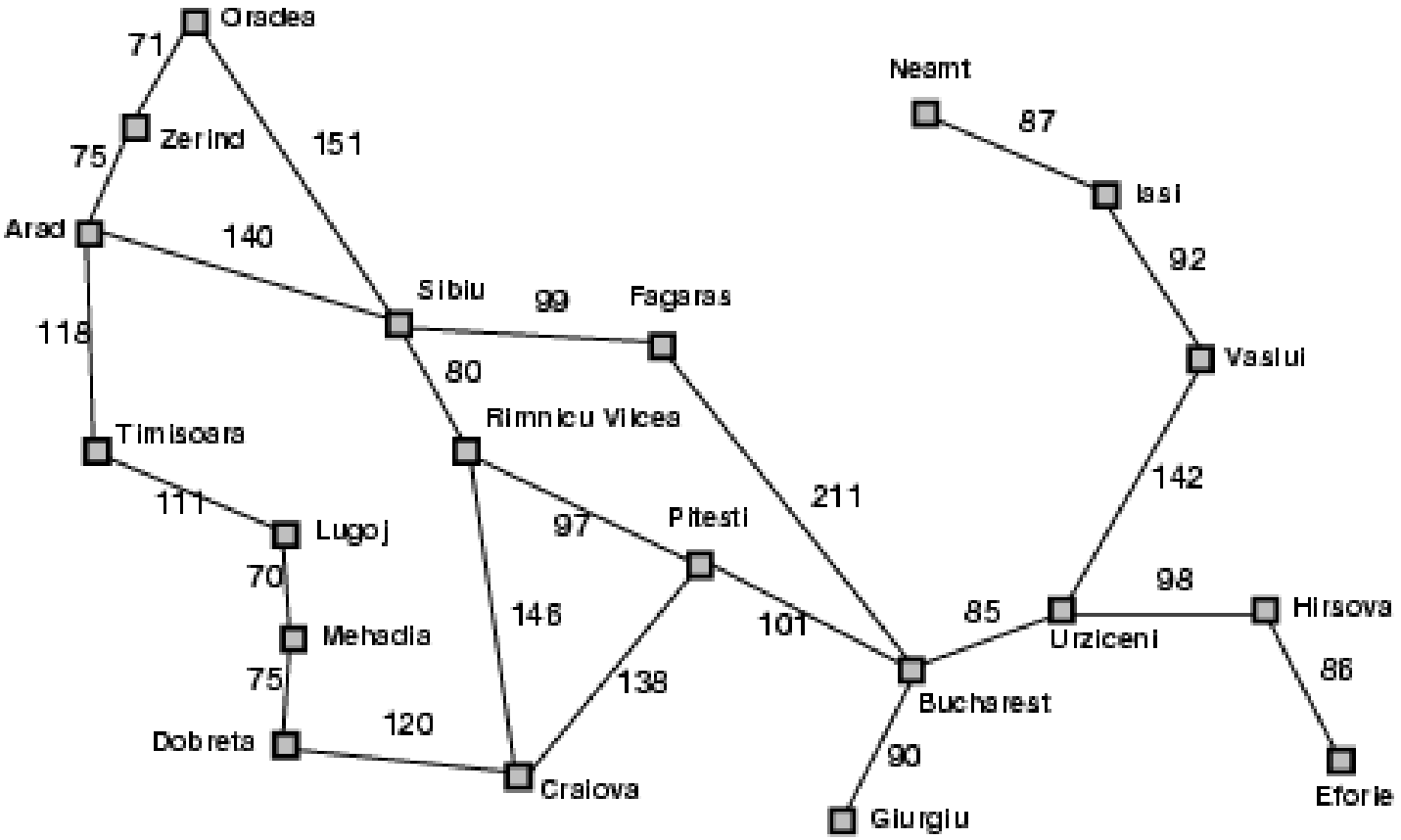
Dijkstra

Passos para construção da árvore:

1. Coloque o estado inicial (s_0) na raiz da árvore;
2. Gere os sucessores do estado inicial e coloque-os no nível 1, com seus respectivos custos [f(s)=g(s)];
3. A partir daí, gere os sucessores do estado ***s*** de menor valor associado (usando o conjunto de ações ***A***), independentemente do seu nível ou ramificação, observando as podas de nós repetidos. A busca termina quando o estado ***s*** selecionado para gerar sucessores é estado final, ou seja, quando $s \in G$.



Dijkstra



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Dijkstra

Dijkstra(A, s_0, G)

1 $\Gamma \leftarrow \varnothing$;

2 $\Sigma \leftarrow \{s_0\}$

3 enquanto $\Sigma \neq 0$ faça

4 $s \leftarrow$ removePrimeiro(Σ)

5 se $s \in G$ então devolva caminho(s)

6 $\Gamma \leftarrow \Gamma \cup \{ s \}$

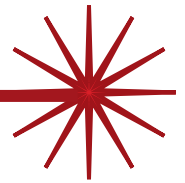
7 $U \leftarrow$ gere_sucessoresG(s, α)

8 Para cada $u \in U \wedge u \notin \Gamma$

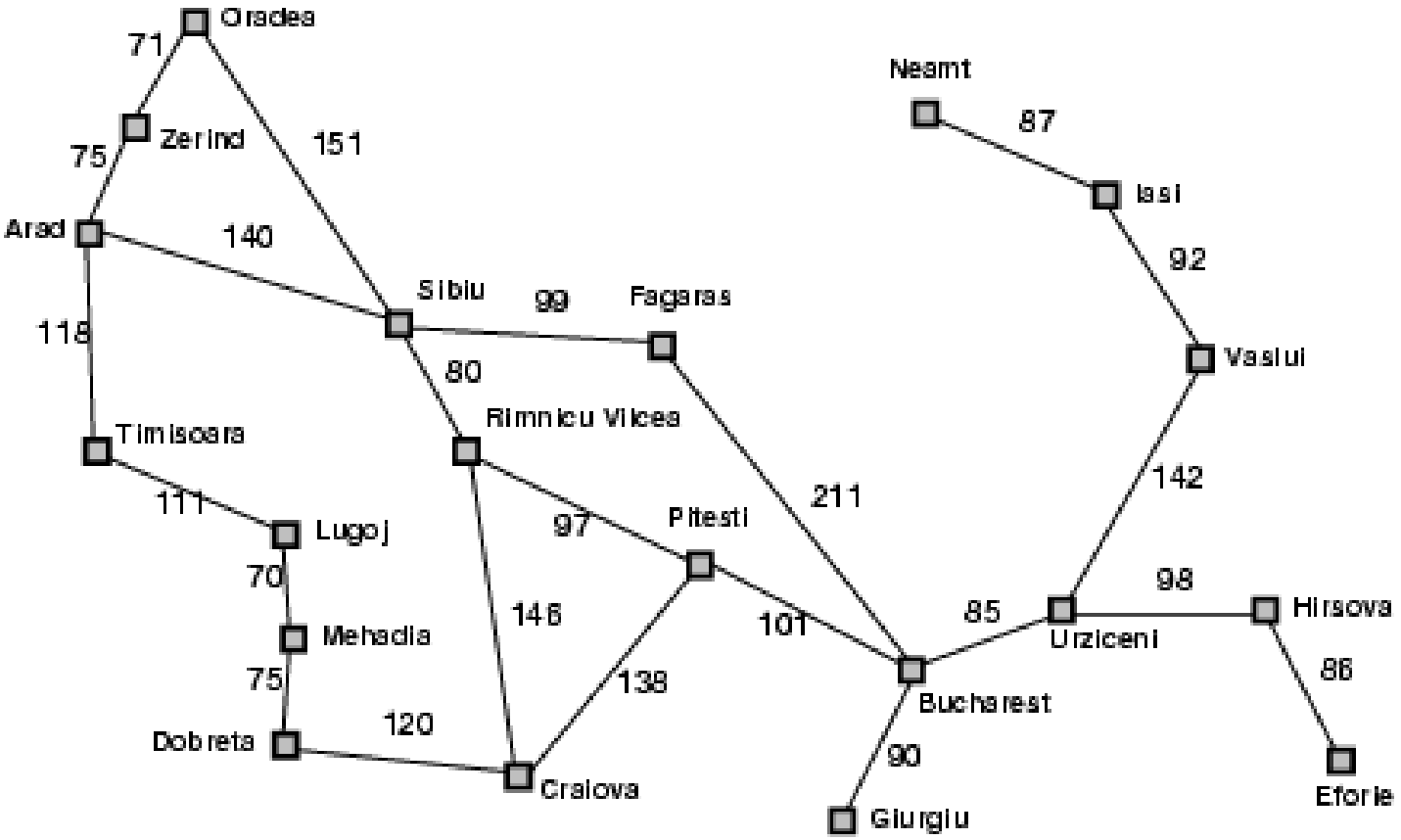
9 se $u \notin \Sigma$ então insereEmOrdem(u,Σ)

10 senão insereEmOrdem(u,Σ) se, e somente se, u é menor que sua réplica em Σ

11 devolva fracasso

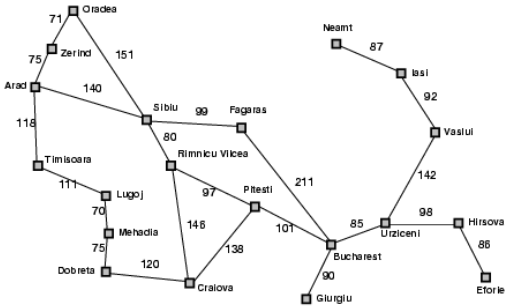


Dijkstra



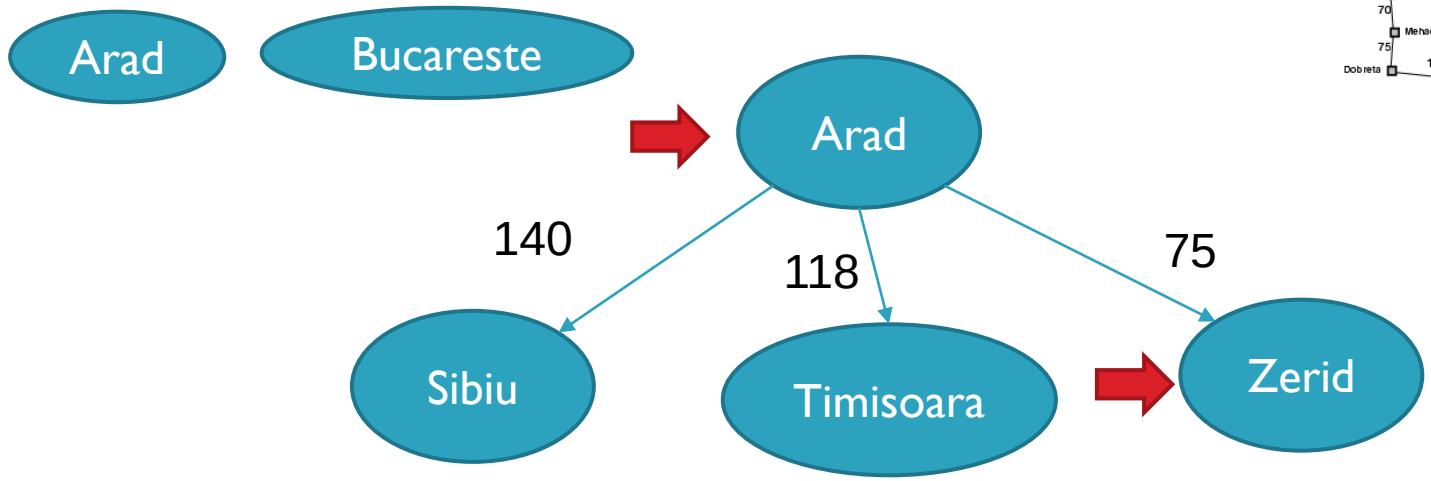
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Dijkstra

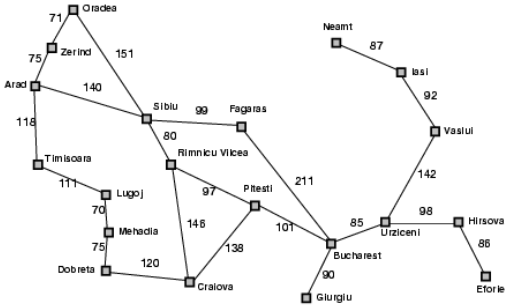


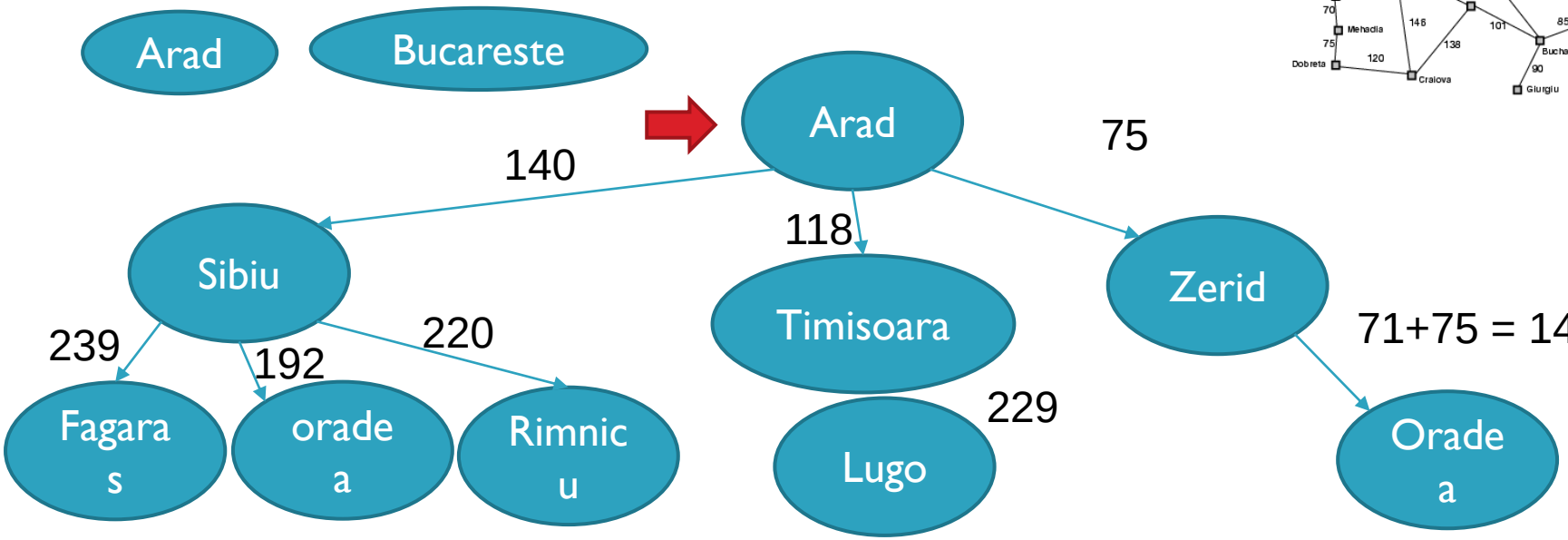
Arad

Bucareste

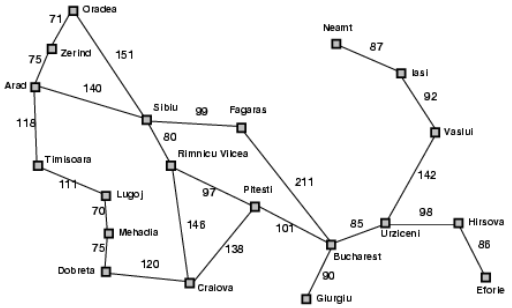


Dijkstra


Oradea
71
Neamt
87
Zerind
75
151
Iasi
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
111
Lugoj
97
Pitesti
211
70
98
Hirsova
Mehadia
146
85
Urziceni
101
75
138
Bucharest
86
Dobreta
120
Craiova
90
Giurgiu
Eforie


Arad
Bucareste
Arad
140
75
118
Sibiu
Timisoara
Zerid
239
192
220
229
71+75 = 146
Fagara
s
orade
a
Rimnic
u
Lugo
Orade
a

Dijkstra



Arad

Bucareste

Arad

140

75

118

Sibiu

Timisoara

Zerid

239

291

220

71+75 = 146

Fagara s

orade a

Rimnic u

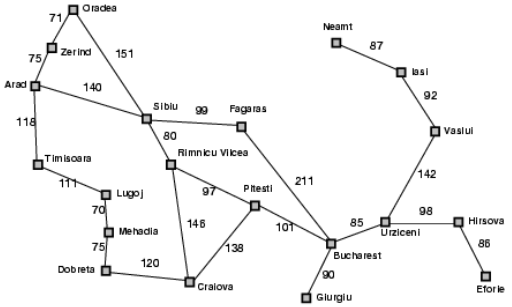
lugo

229

Orade a

Mesmo nó, devemos podar o nó com maior custo

Dijkstra



Arad

Bucareste

Arad

140

75

118

Sibiu

Timisoara

Zerid

239

291

220

71+75 = 146

Fagara s

Orade a

Rimnic u

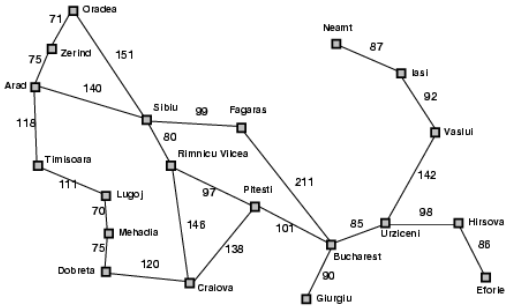
lugo

229

Orade a

Mesmo nó, devemos podar o nó com maior custo

Dijkstra



Arad

Bucareste

Arad

140

75

118

Sibiu

Timisoara

Zerid

239

220

71+75 = 146

Fagara
s

Rimnic
u

lugo

Orade
a

229

299

297

450

366

319

bucare
ste

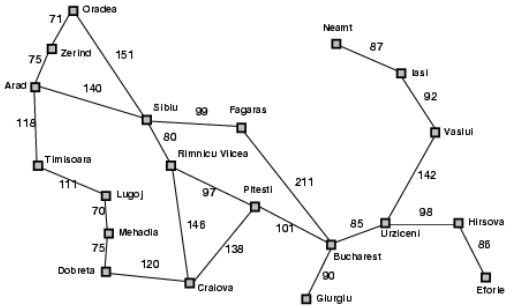
Pitesti

craiov
a

mehad
ia

Sibiu

Dijkstra



Arad

Bucareste

Arad

140

75

Sibiu

118

Timisoara

Zerid

239

220

71+75 = 146

Fagara
s

Rimnic
u

229

lugo

Orade
a

366

319

299

297

450

bucare
ste

Pitesti

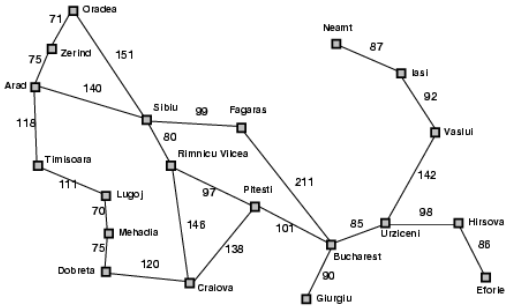
craiov
a

mehad
ia

Sibiu

Mesmo nó, devemos podar o nó com maior custo

Dijkstra



Arad

Bucareste

Arad

140

75

118

Sibiu

Timisoara

Zerid

239

220

71+75 = 146

Fagara
s

Rimnic
u

229

lugo

Orade
a

297

366

319

299

450

bucare
ste

Pitesti

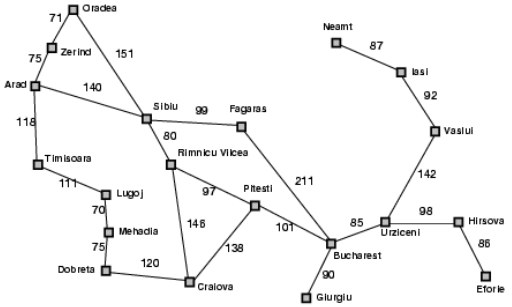
craiov
a

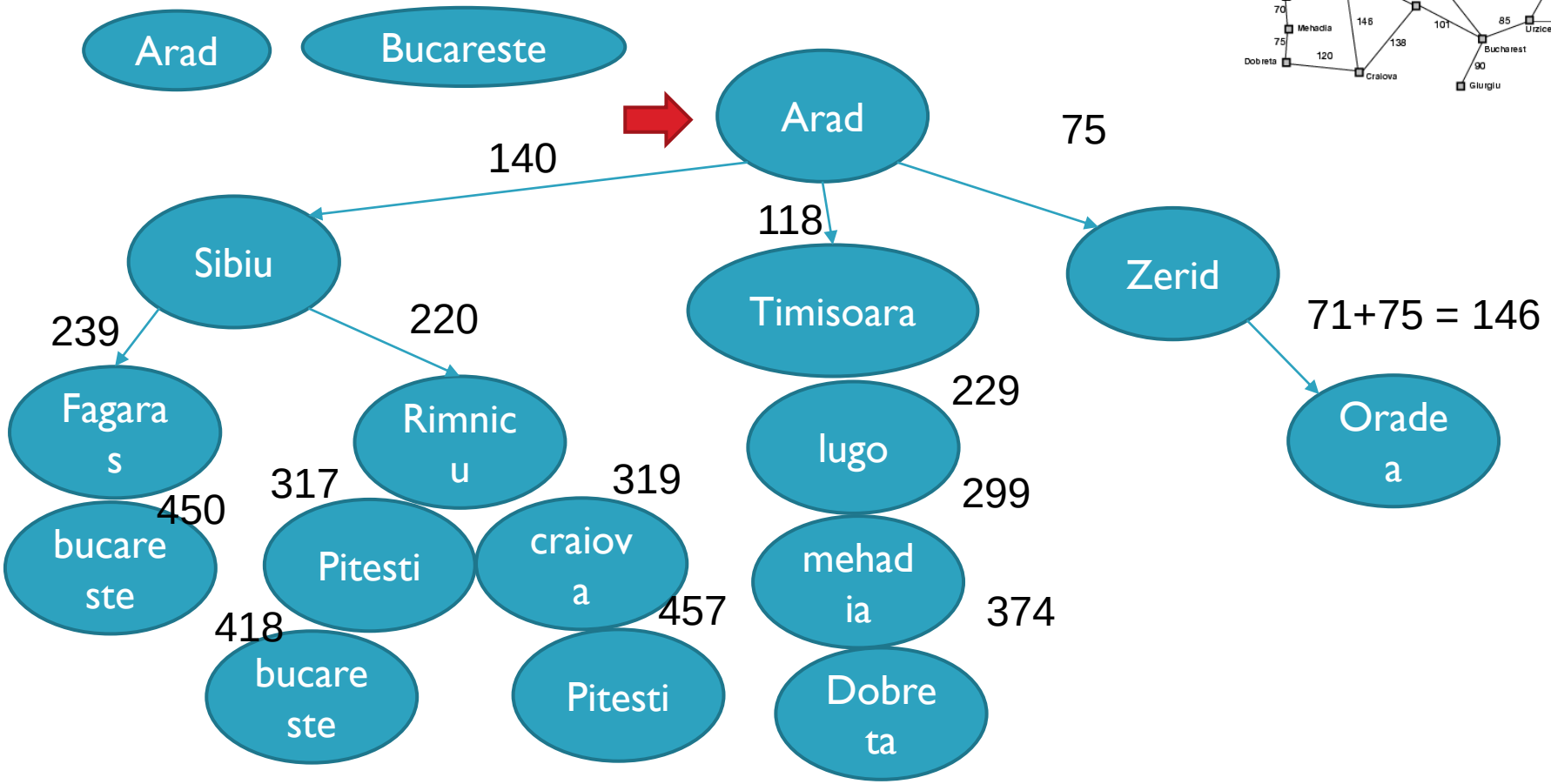
mehad
ia

Sibiu

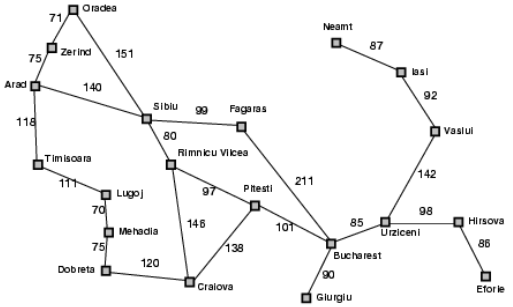
Mesmo nó, devemos podar o nó com maior custo

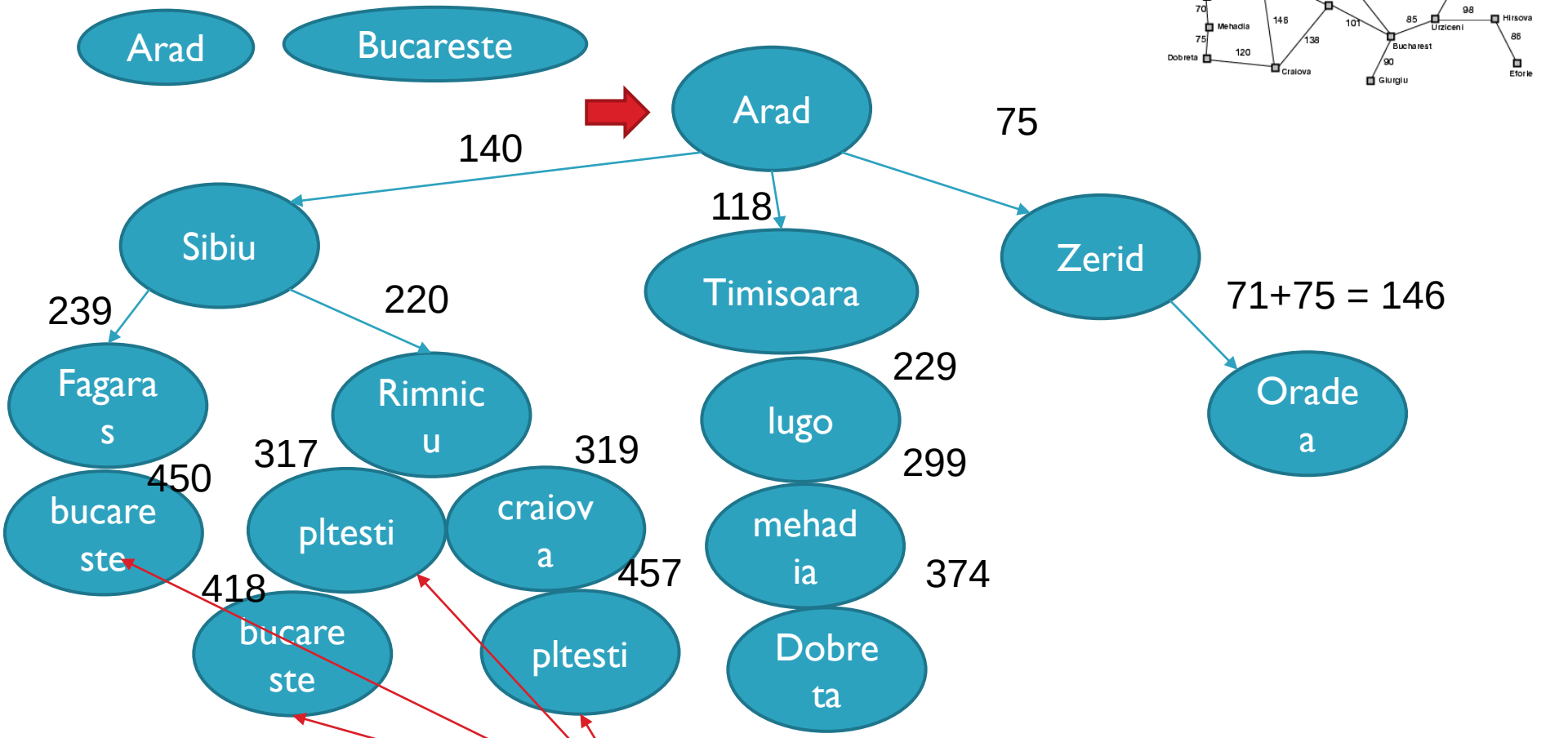
Dijkstra


Oradea
71
Neamt
87
Zerind
75
151
Iasi
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
111
Lugoj
97
Pitesti
211
70
98
Hirsova
Mehadia
146
85
Urziceni
101
75
138
Bucharest
86
Dobreta
120
90
Craiova
Giurgiu
Eforie


Arad
Bucareste
Arad
140
75
118
Sibiu
Timisoara
Zerid
239
220
71+75 = 146
Fagara
s
Rimnic
u
lugo
Orade
a
229
317
319
299
450
bucare
ste
Pitesti
craiov
a
mehad
ia
457
374
418
bucare
ste
Pitesti
Dobre
ta

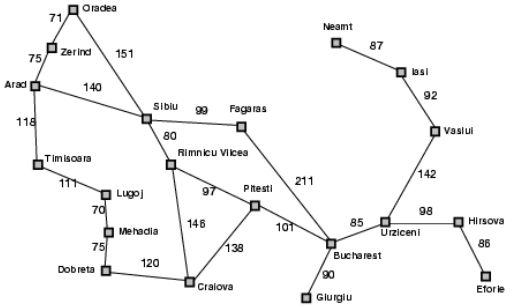
Dijkstra





Mesmo nó, devemos podar o nó com maior custo

Dijkstra



Arad

Bucareste

Arad

140

75

118

Sibiu

Timisoara

Zerid

239

220

71+75 = 146

Fagara s

Rimnic u

lugo

Orade a

229

317

319

299

450

pltesti

craiov a

mehad ia

418

457

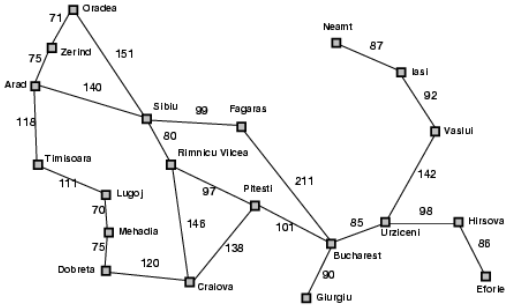
374

bucare ste

Dobre ta

Mesmo nó, devemos podar o nó com maior custo

Dijkstra



Arad

Bucareste

Arad

140

75

118

Sibiu

Timisoara

Zerid

239

220

71+75 = 146

Fagara s

Rimnic u

lugo

Orade a

229

317

319

299

pltesti

craiov a

mehad ia

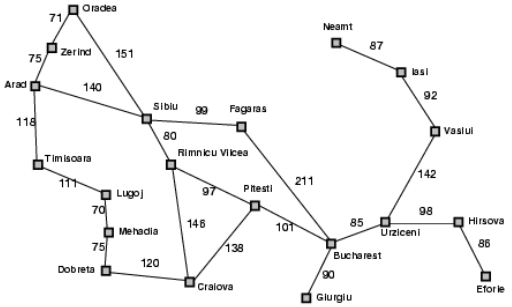
374

418

bucare ste

Dobre ta

Dijkstra



Arad

Bucareste

Arad

140

75

118

Sibiu

Timisoara

Zerid

239

220

71+75 = 146

Fagara
s

Rimnic
u

lugo

Orade
a

229

317

319

299

pltesti

craiov
a

mehad
ia

374

418

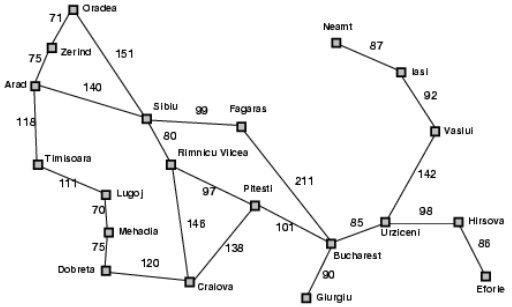
bucare
ste

Dobre
ta

494

craiov
a

Dijkstra



Arad

Bucareste

Arad

140

75

118

Sibiu

Timisoara

Zerid

239

220

71+75 = 146

Fagara
s

Rimnic
u

lugo

Orade
a

229

317

319

299

pltesti

craiov
a

mehad
ia

374

418

bucare
ste

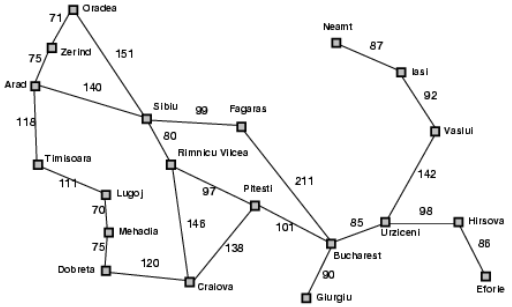
Dobre
ta

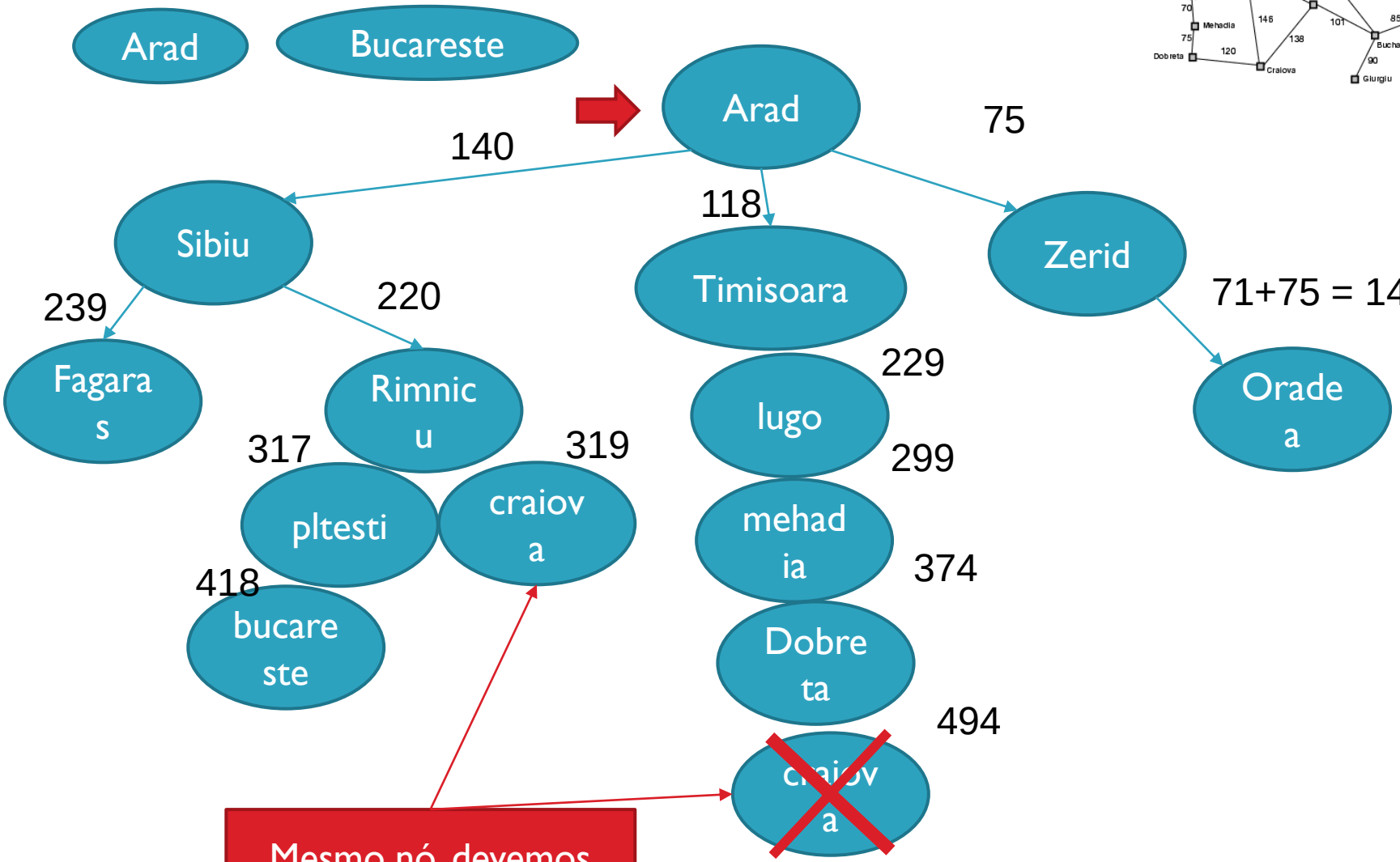
494

craiov
a

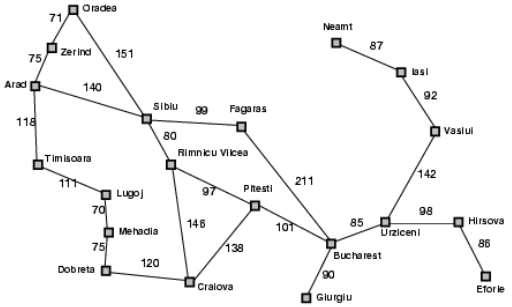
Mesmo nó, devemos podar o nó com maior custo

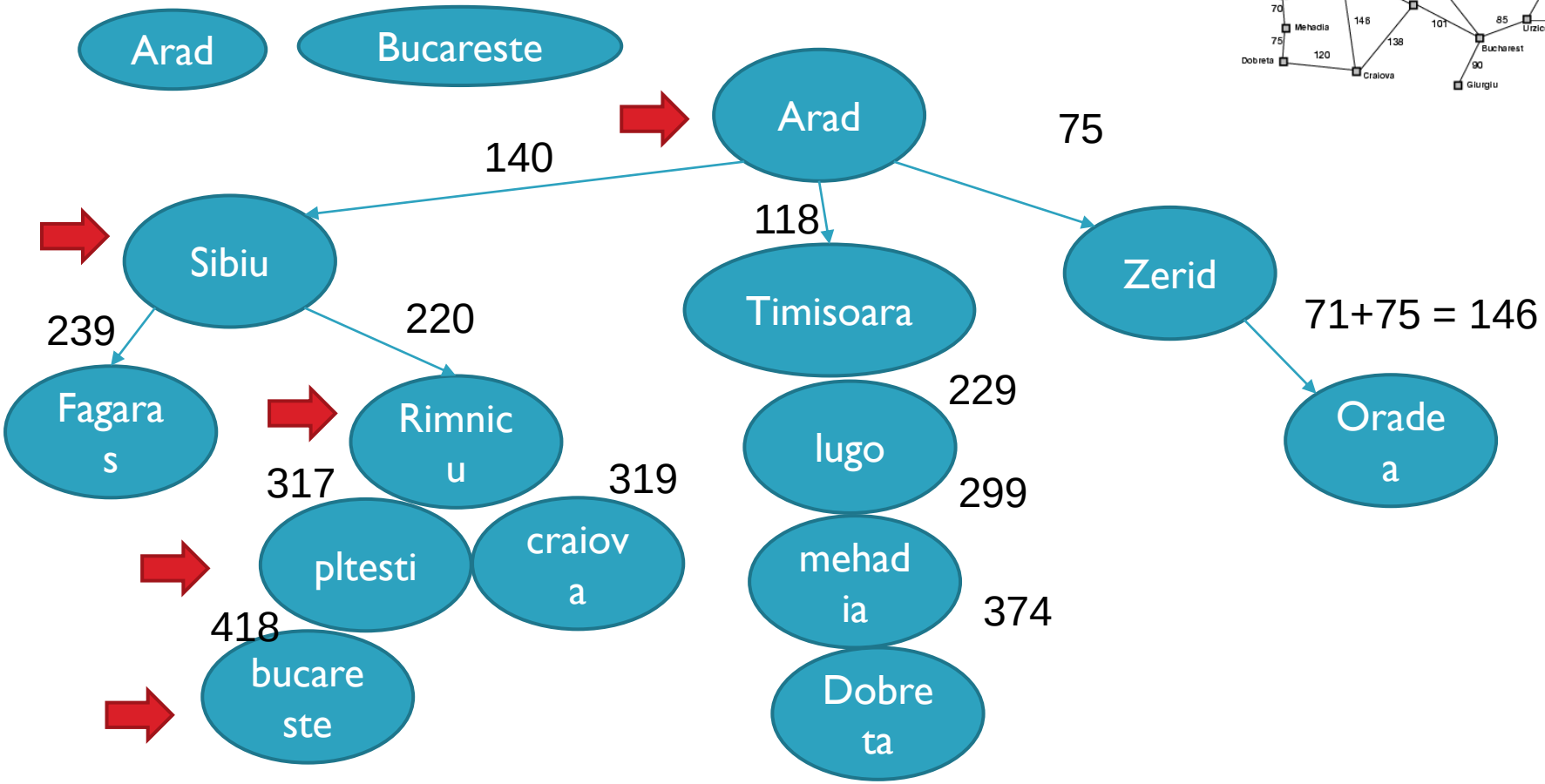
Dijkstra


Oradea
71
Zerind
75
151
Arad
140
Sibiu
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Dobreta
120
90
Craiova
Giurgiu
Eforie
Neamt
87
Iasi
92
Vaslui
142


Arad
Bucareste
Arad
140
75
118
Sibiu
Timisoara
Zerid
239
220
71+75 = 146
Fagara
s
Rimnic
u
lugo
Orade
a
229
317
319
299
pltesti
craiov
a
mehad
ia
374
418
bucare
ste
Dobre
ta
494
craiov
a
Mesmo nó, devemos podar o nó com maior custo

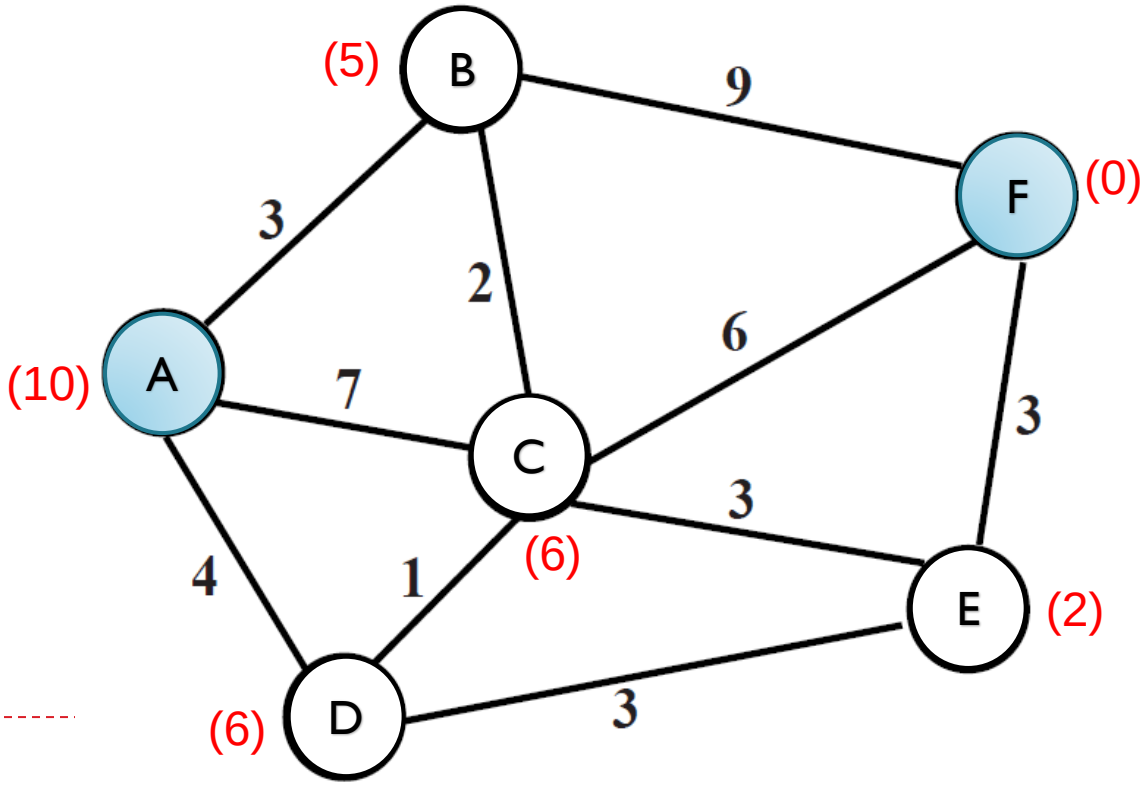
Dijkstra


Oradea
71
Neamt
87
Zerind
75
151
Iasi
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
111
Lugoj
97
Pitesti
211
70
98
Hirsova
Mehadia
146
85
Urziceni
101
75
138
Bucharest
86
Dobreta
120
Craiova
90
Giurgiu
Eforie

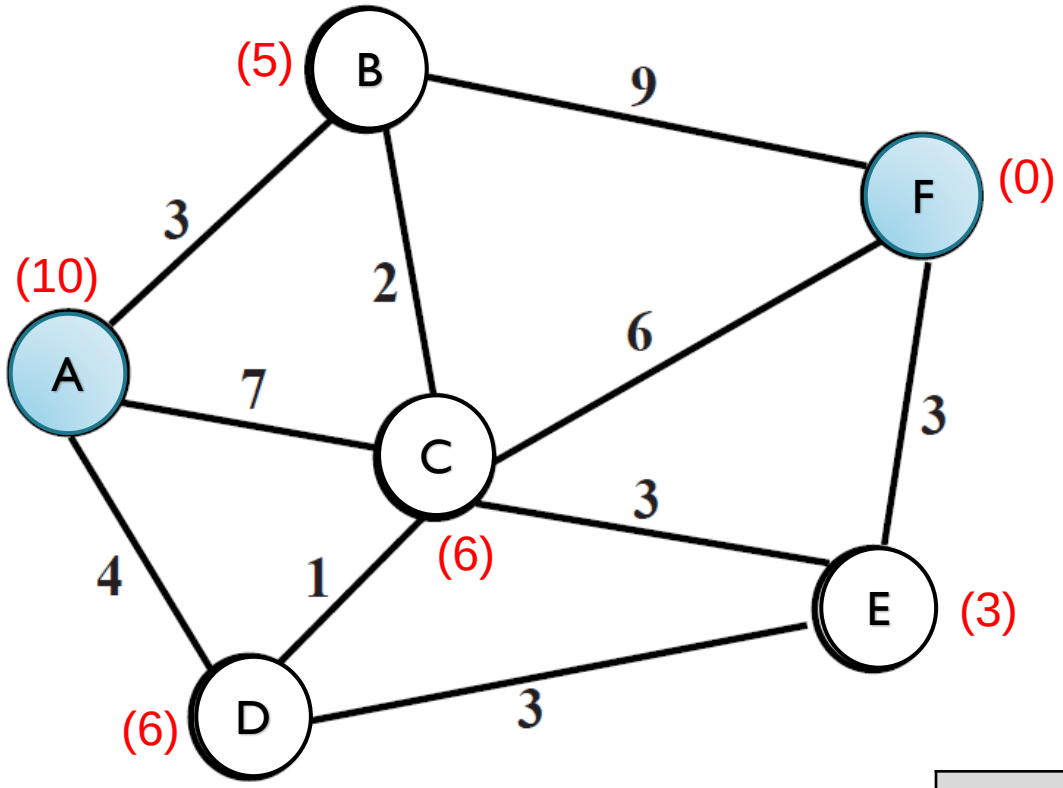

Arad
Bucareste
Arad
140
75
118
Sibiu
Timisoara
Zerid
239
220
71+75 = 146
Fagara
s
Rimnic
u
229
lugo
Orade
a
317
319
299
pltesti
craiov
a
mehad
ia
374
418
bucare
ste
Dobre
ta

Exercício 1

Considere o **Problema de Caminho Mínimo** no qual se deseja encontrar uma rota que ligue o nó ***A*** ao nó ***F***. No grafo abaixo cada arco indica a distância em km entre os nós (custo) e entre parênteses estão indicadas as estimativas de distâncias até o nó objetivo. Formalize (represente) o problema e encontre rotas utilizando as seguintes estratégias de busca: Menor Custo, Dijkstra, Melhor Estimativa e A*.



Exercício 1 – Representação Probl.



Matriz de Adjacências/Distâncias

	A	B	C	D	E	F
A	∞	3	7	4	∞	∞
B	3	∞	2	∞	∞	9
C	7	2	∞	1	3	6
D	4	∞	1	∞	3	∞
E	∞	∞	3	3	∞	3
F	∞	9	6	∞	3	∞

Matriz heurística

h(A)	h(B)	h(C)	h(D)	h(E)	h(F)
10	5	6	6	3	0

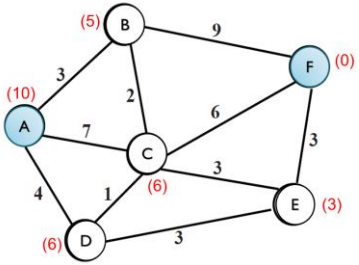
S = {A,B,C,D,E,F}

s_0=A

G={F}

A = {vai(α,α')←aresta(α,α')}

Exercício 1 – Representação Probl.



Exercício 2

Considere o **Problema de Caminho Mínimo** no qual se deseja encontrar uma rota que ligue o roteador ***A*** ao roteador ***I***. No grafo abaixo cada arco indica a distância em km entre os roteadores (custo) e entre parênteses estão indicadas as estimativas de distâncias até o nó objetivo. Formalize o problema e encontre rotas utilizando as seguintes estratégias de busca: Largura, Profundidade, Menor Custo, Dijkstra, Melhor Estimativa e A*.