Métodos de Busca

Ádamo L. Santana
adamo@ufpa.br

Agente de Resolução de Problemas

- O Agente Reativo
  - Escolhe suas ações com base apenas nas percepções atuais
  - não pode pensar no futuro, não sabe “aonde vai”

Problemas e Soluções bem Definidos

- Espaço de Estados:
  - conjunto de todos os estados alcançáveis a partir do estado inicial por qualquer sequência de ações.
- Definição do objetivo:
  - propriedade abstrata
    - ex., condição de xeque-mate no Xadrez
  - conjunto de estados finais do mundo
    - ex., estar na cidade-destino
- Solução:
  - caminho (sequência de *ações* ou *operadores*) que leva do estado inicial a um estado final (objetivo).

Solucionando o Problema: formulação, busca e execução

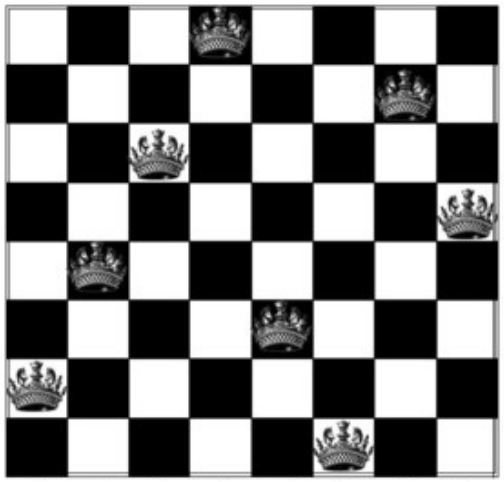
- Formulação do problema e do objetivo:
  - quais são os ***estados*** e as ***ações*** a considerar?
  - qual é (e como representar) o **objetivo**?
- Busca (solução do problema):
  - processo que gera/analisa sequências de ações para alcançar um objetivo
- Execução:
  - Executar (passo a passo) a solução ***completa*** encontrada

Medida de Desempenho na Busca

- Desempenho de um algoritmo de busca:
  - 1. O algoritmo encontrou alguma solução?
  - 2. É uma boa solução?
    - **custo de caminho** (qualidade da solução)
  - 3. É uma solução computacionalmente barata?
    - **custo da busca** (tempo e memória)
- Custo total
  - custo do caminho + custo de busca
- Espaço de estados grande:
  - conflito entre a melhor solução e a solução mais barata

Importância da formulação: 8 rainhas

- Jogo das 8 Rainhas
  - dispor 8 rainhas no tabuleiro de forma que não possam se "atacar"
    - não pode haver mais de uma rainha em uma mesma linha, coluna ou diagonal
  - somente o custo da busca conta
    - não existe custo de caminho

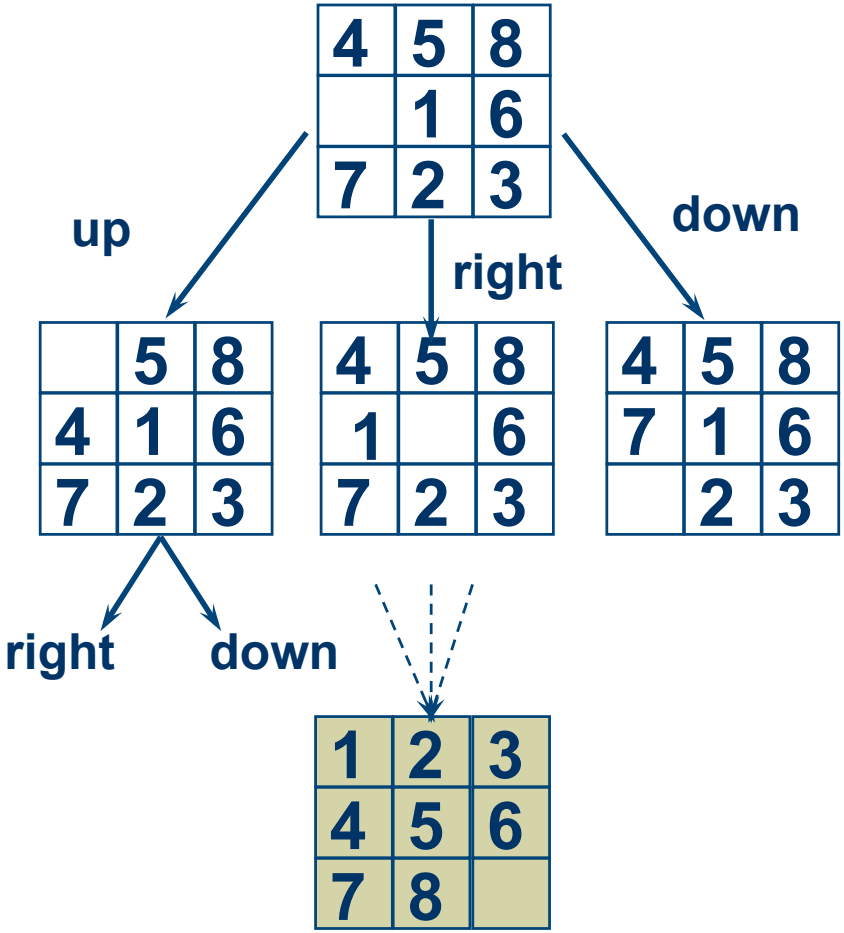


Importância da formulação: 8 rainhas

- Formulação A
  - estados: qualquer disposição com n (n ≤ 8) rainhas
  - operadores: adicionar uma rainha a qualquer quadrado
- Formulação B
  - estados: disposição com 8 rainhas, uma em cada coluna
  - operadores: mover uma rainha atacada para outra casa na mesma coluna

Importância da formulação: 8 números

- Jogo de 8 números:
  - **estados** = cada possível configuração do tabuleiro
  - **estado inicial** = qualquer um dos estados possíveis
  - **teste de término** = ordenado, com branco na posição [3,3]
  - **operadores** = mover branco (esquerda, direita, para cima e para baixo)
  - **custo da solução** = número de passos da solução



Algoritmos de Busca

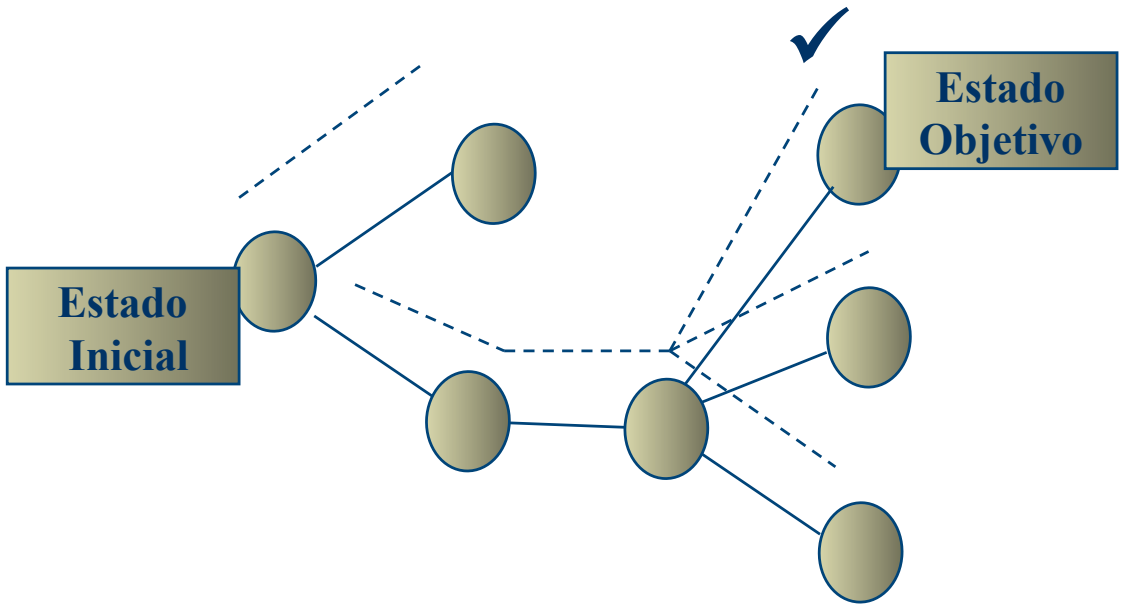
- Métodos de busca:
  - Busca cega: a escolha depende da posição do nó na árvore de busca
  - Busca heurística: a escolha utiliza informações específicas do domínio para ajudar na decisão

Busca Cega

- Técnica de busca
  - Busca em Profundidade
    - A árvore é examinada de cima para baixo
    - Aconselhável nos casos onde os caminhos não são muito longos
  - Busca em Largura
    - A árvore é examinada da esquerda para a direita
    - Aconselhável quando o número de ramos não é muito grande.

Busca em Largura

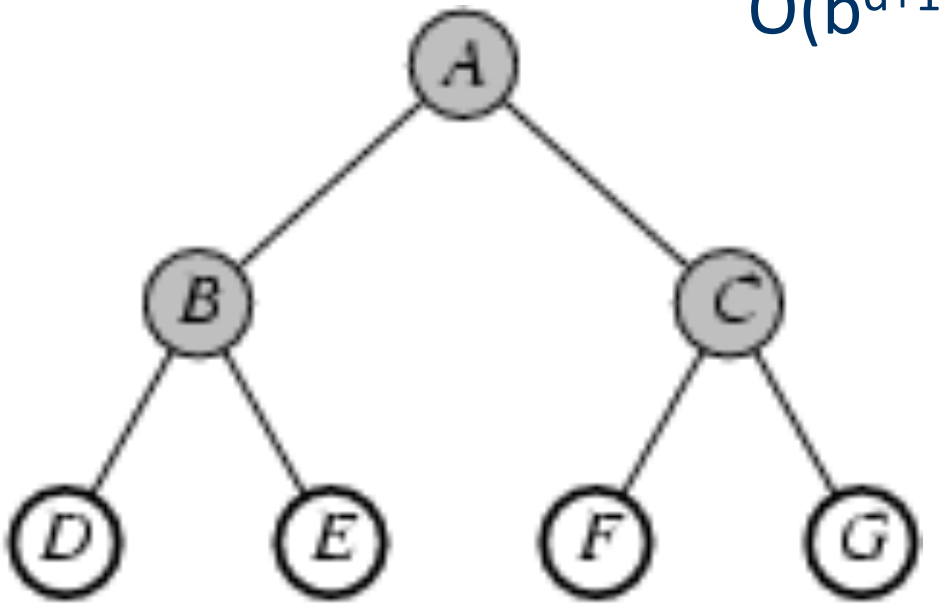
- Busca em Largura envolve a escolha de um caminho e segui-lo até o próximo ponto de decisão ou até o objetivo a ser atingido:



Busca em extensão

- Expandir o nó não-expandido mais perto da raiz.
- *Tempo/Espaço:* $1+b+b^2+b^3+\ldots+b^d + b(b^d-1) =$

$O(b^{d+1}-b)=O(b^{d+1})$



Busca em extensão

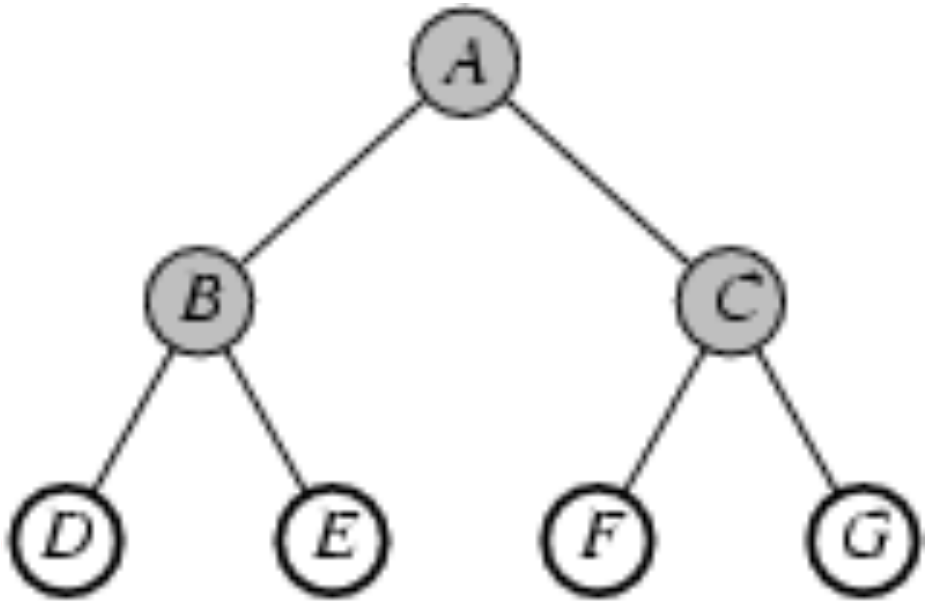
- Estrutura FIFO

  A - ABC

  ABC - ABCDE

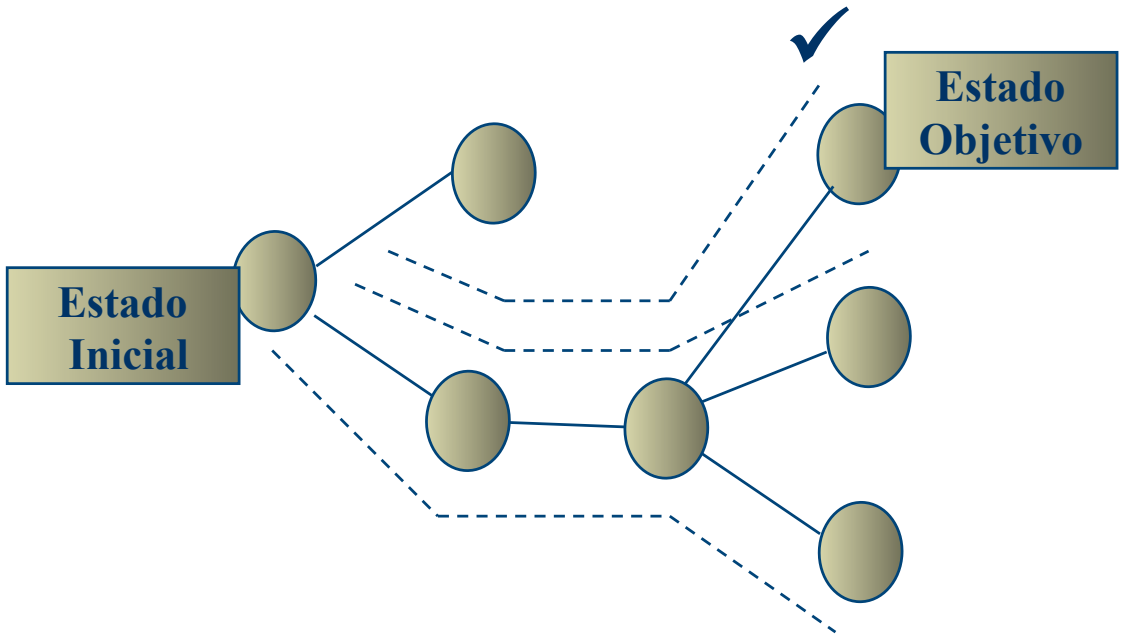
  ABCDE – ABCDEFG

  ...



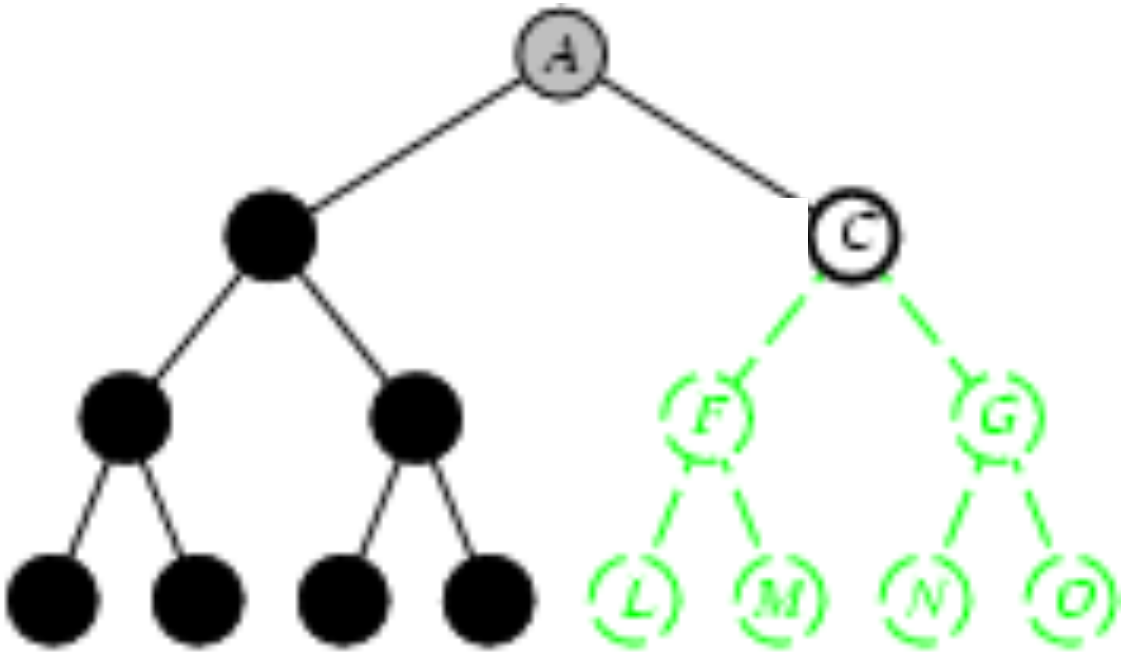
Busca em Profundidade

- Busca em Profundidade envolve buscar o final de um caminho antes de tentar um caminho alternativo:



Busca em Profundidade

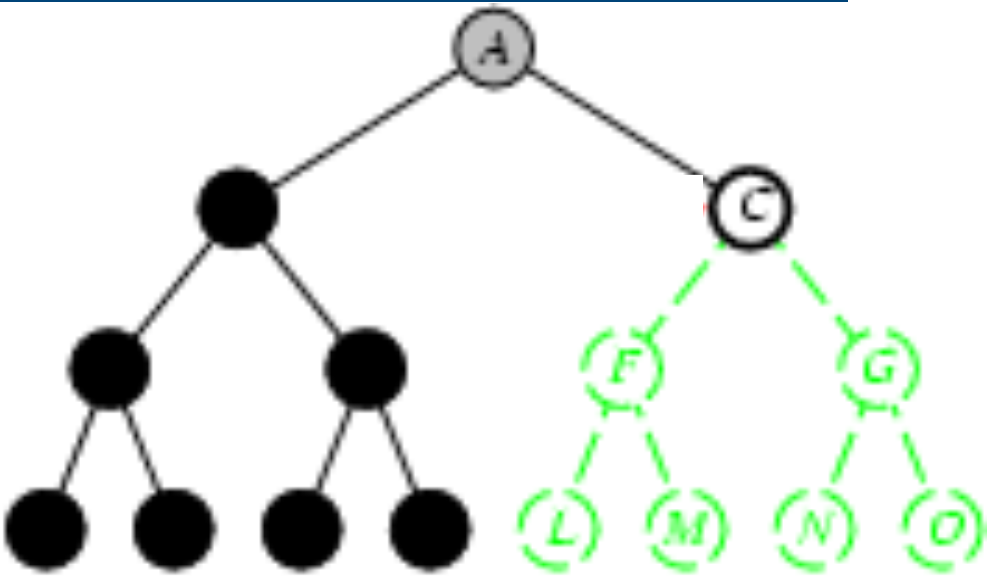
- Expande o nó não-expandido mais profundo.
- Tempo: $O(b^m)$ | Espaço: $O(bm)$



Busca em Profundidade

- Estrutura LIFO

A – BCA
BCA – DEBCA
DEBCA – FGDEBCA
HIDEBCA – HIDEBCA
HIDEBCA – HIDEBCA
HIDEBCA – JKHIDEBCA
JKHIDEBCA – JKHIDEBCA
JKHIDEBCA – JKHIDEBCA
JKHIDEBCA – FGJKHIDEBCA
...



Problemas de Busca

- Mais de um nó objetivo.
- Mais de um nó inicial.
- Nestas situações:
  - Encontrar qualquer caminho de um nó inicial para um nó objetivo
  - Encontrar o melhor caminho

Busca em Profundidade X Busca em Largura

- Busca em Largura e Busca em Profundidade não precisam ser realizados em uma ordem específica
- Memória utilizada pelas duas técnicas:
  - BP: precisa armazenar todos os nós filhos não visitados de cada nó entre o nó atual e o nó inicial
  - BL: antes de examinar nó a uma profundidade *d*, é necessário examinar e armazenar todos os nós a uma produndidade *d-1*
  - BP utiliza menos memória que BL

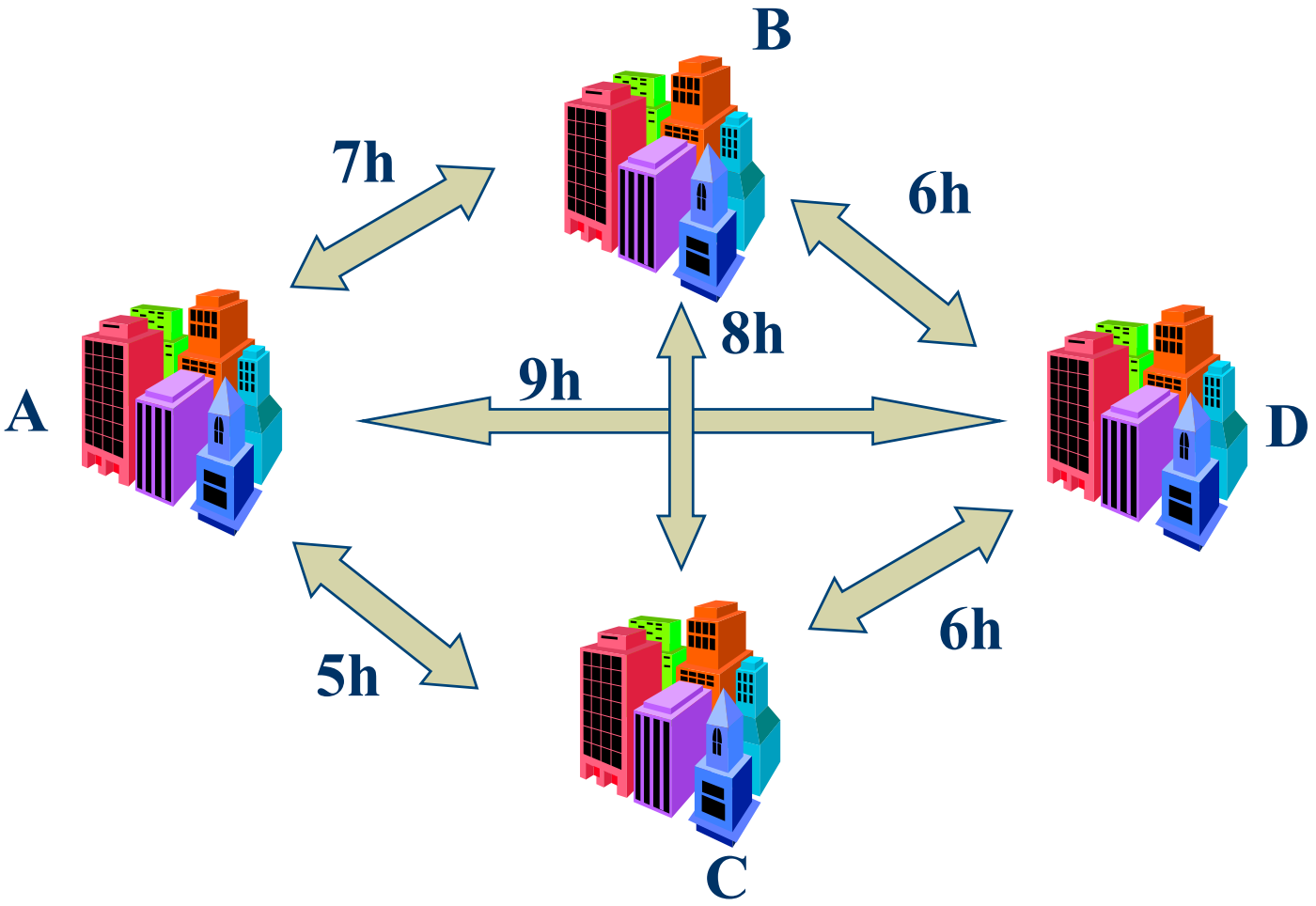
Busca em Profundidade X Busca em Largura

- Quanto ao tempo
  - BP é geralmente mais rápida
  - Métodos de busca cega não examinam a árvore de forma ótima, o que poderia minimizar o tempo gasto para resolver o problema

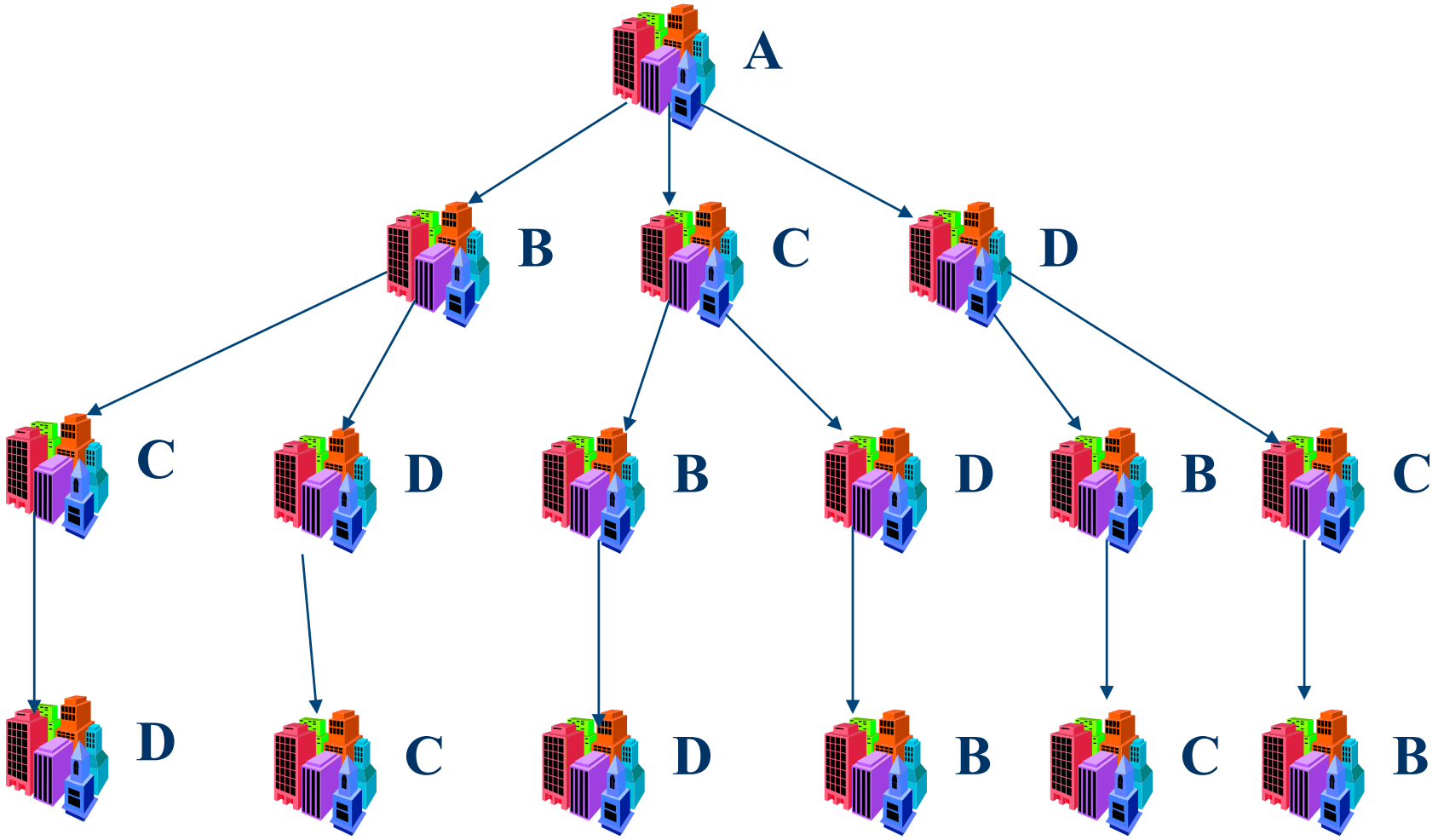
Busca Heurística

- Com o aumento da árvore e do número de possíveis caminhos, o tempo de busca aumenta
- Ex: Problema do caixeiro viajante: um caixeiro viajante deve visitar N cidade em sua área de vendas
  - O caixeiro começa de uma base, visita cada cidade uma única vez e retorna à base.
  - A cada viajem deve ser associado um custo
    - O caixeiro viajante deve percorrer a rota mais curta

O problema do Caixeiro Viajante



O problema do Caixeiro Viajante representado como uma árvore de busca



Explosão Combinatória

- Com 4 cidades, tem-se 6 caminhos possíveis
- Com dez cidade, temos 362880 caminhos possíveis
- Como prevenir ou pelo menos limitar isto?

Heurística

- É preciso limitar de alguma forma o espaço de busca, e assim tornar o processo de busca mais rápido e eficiente;
- Humanos utilizariam “macetes”
- Em IC são chamados de heurísticas
- Estas heurísticas ajudam a limitar a busca.

Algoritmos Heurísticos

- Escolher primeiro as opções mais promissoras
  - Em algumas situações é possível obter medidas que determinam uma ordenação razoável
- Exemplos de algoritmos heurísticos:
  - Branch and Bound
  - Hill Climbing
  - A* (Busca menor custo)
  - ...

Algoritmo Branch and Bound

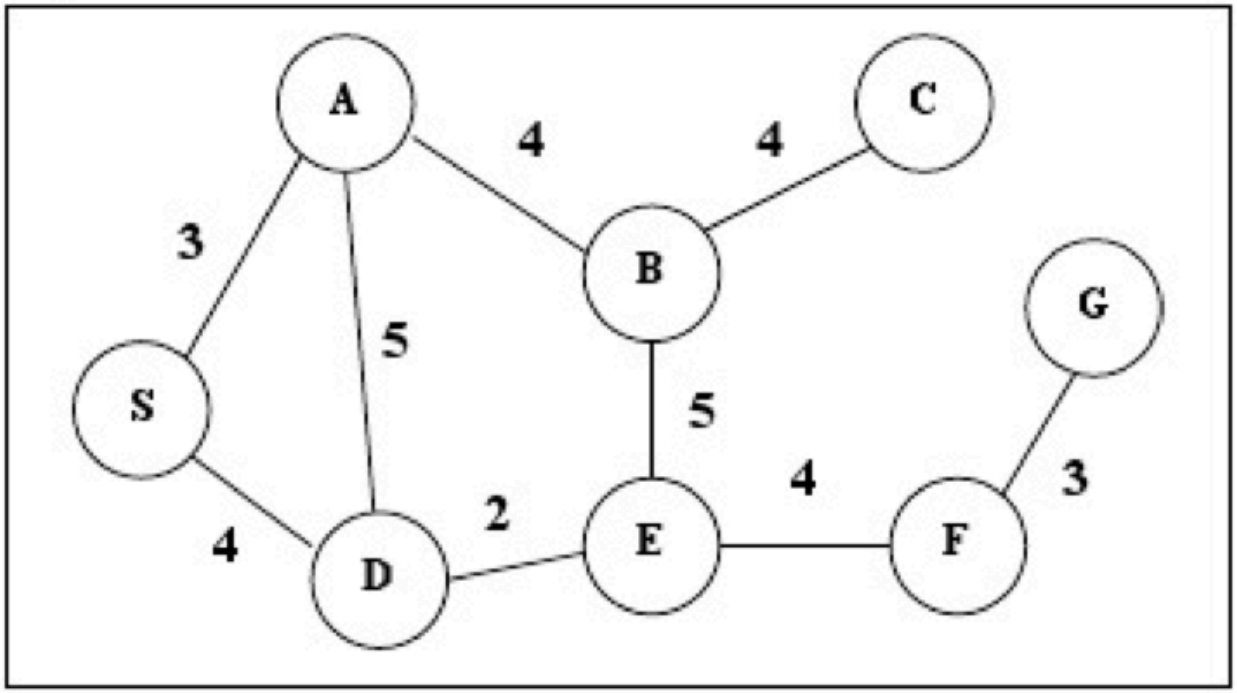
- Este método considera que a cada passo são estabelecidos um *bound*(limite) de quais *branches*(ramos) serão investigados
- Caminhos mais curtos são sempre avaliados primeiro (e expandidos) primeiro, deste modo espera-se que a solução encontrada seja a ideal
- Aplicando o *branch and bound* ao problema do caixeiro viajante, poderiam ser considerados somente aqueles caminhos que levam a uma nova cidade, e que tenham o tempo mais curto entre as cidades

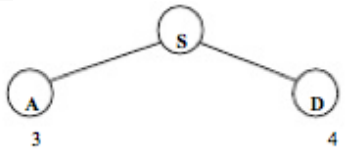
Algoritmo Branch and Bound

- Em sua forma algorítmica, os nós são particionados (branch) e suas soluções (caminhos) podados segundo um limite inferior (bound) de solução
- Menores caminhos são avaliados a cada passo
- A escolha do próximo nó a avaliar é feita a partir do conjunto de nós gerados que não tenham sido eliminados, nem foram ramificados
- Restrições do problema são avaliadas posteriormente às ramificações

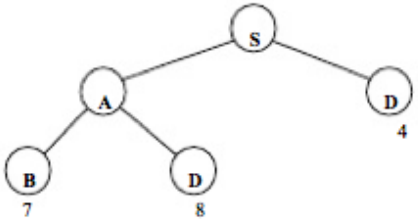
Algoritmo Branch and Bound

- Ex:

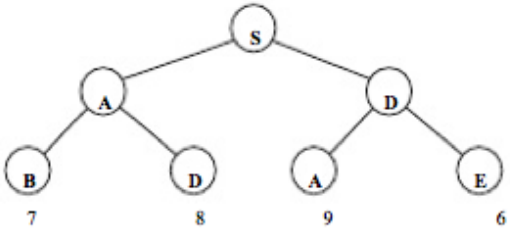




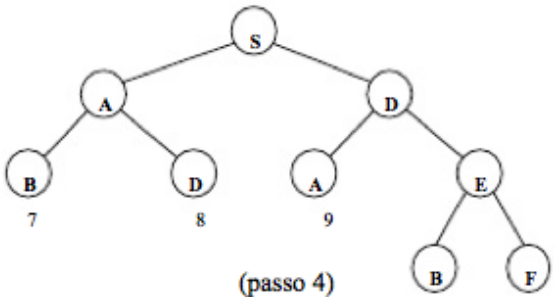
(passo 1)



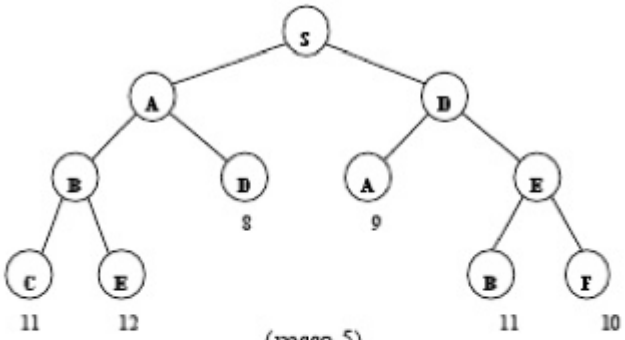
(passo 2)



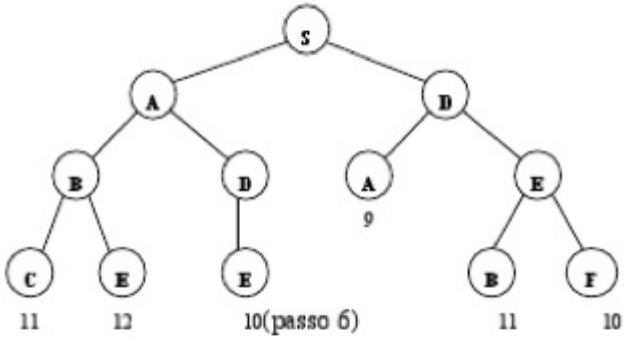
(passo 3)



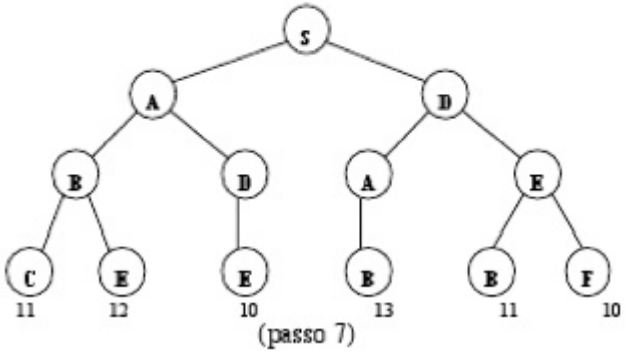
(passo 4)



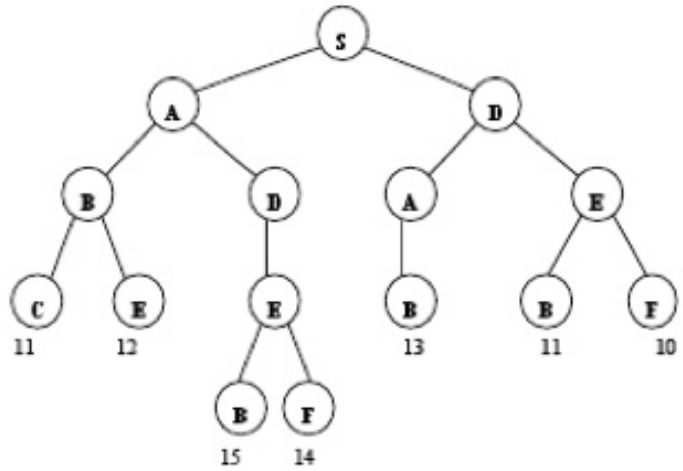
(passo 5)



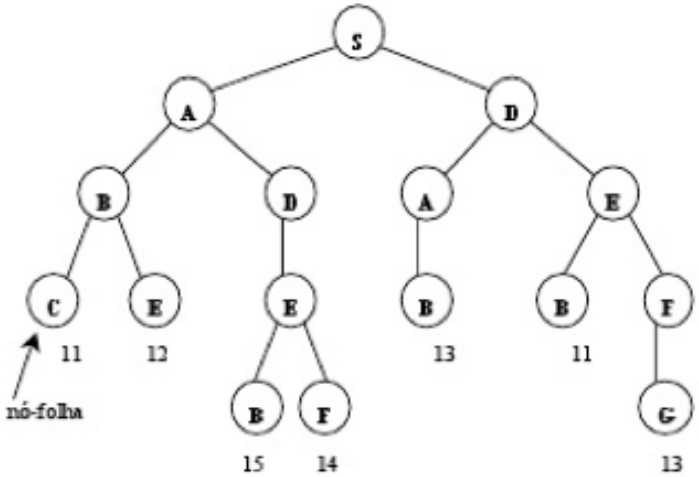
10(passo 6)



(passo 7)



(passo 8)



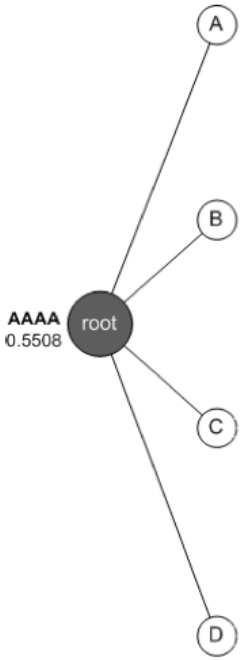
(passo 9)

Algoritmo Branch and Bound

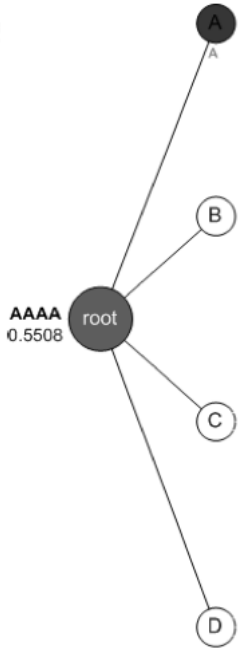
- Ex:

		Operation 1	2	3	4
Team member	A	0.9	0.8	0.9	0.85
	B	0.7	0.6	0.8	0.7
	C	0.85	0.7	0.85	0.8
	D	0.75	0.7	0.75	0.7

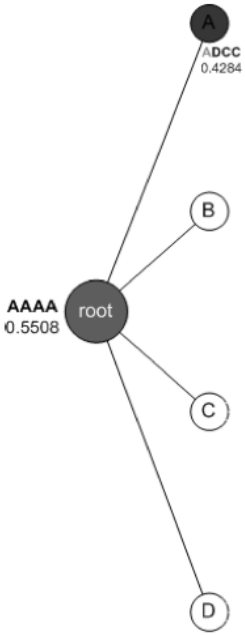
P(sucesso)>45%



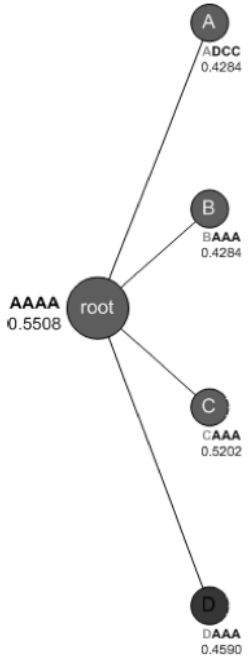
(passo 1)



(passo 2)



(passo 3)

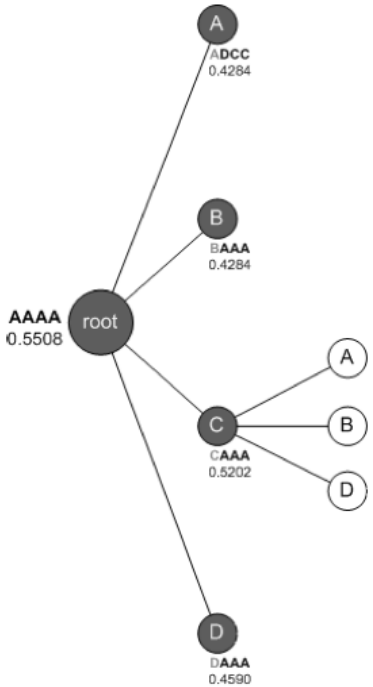


(passo 4)

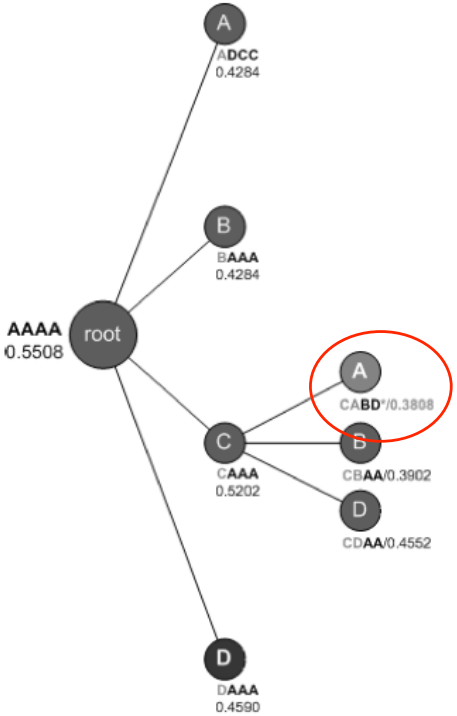
Algoritmo Branch and Bound

- Ex:

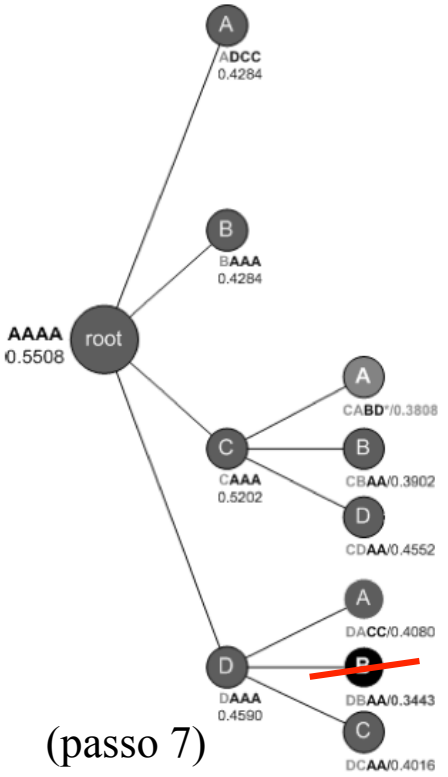
Team member	Operation 1	2	3	4
A	0.9	0.8	0.9	0.85
B	0.7	0.6	0.8	0.7
C	0.85	0.7	0.85	0.8
D	0.75	0.7	0.75	0.7



(passo 5)



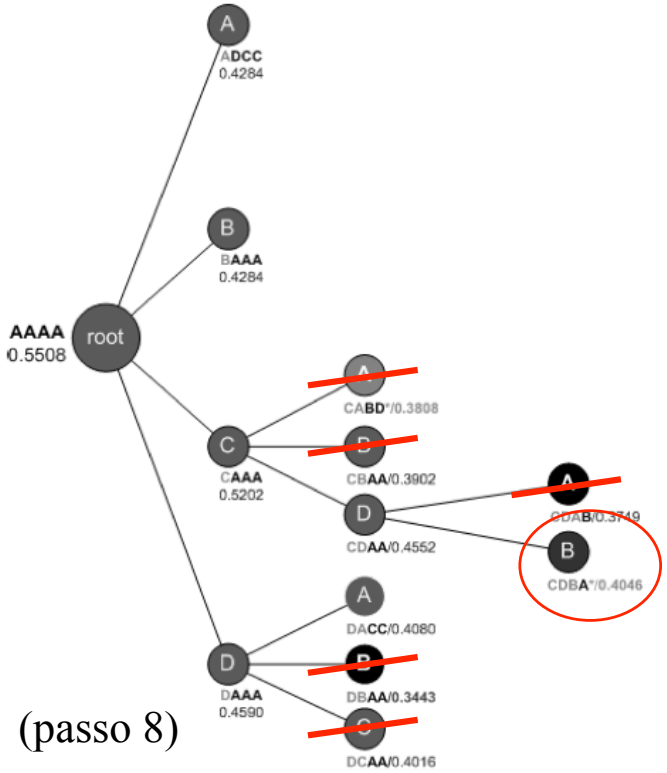
(passo 6)



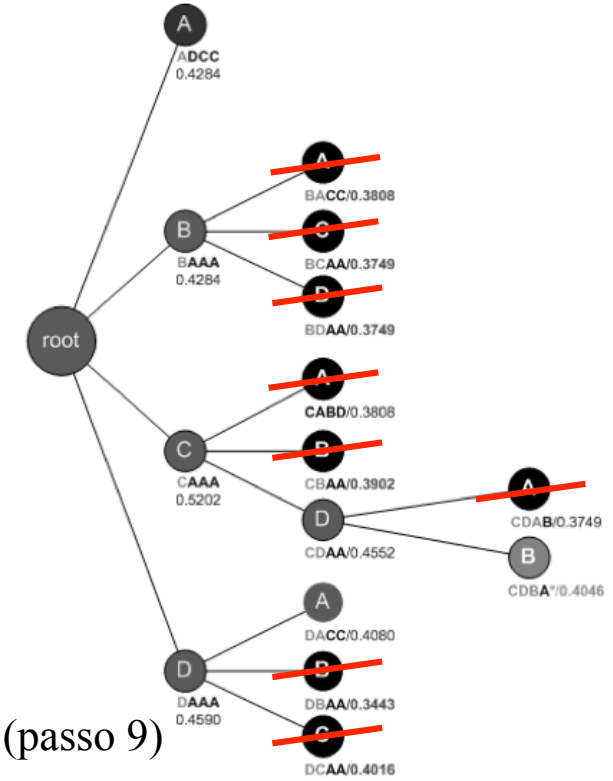
(passo 7)

Algoritmo Branch and Bound

		Operation			
		1	2	3	4
Team member	A	0.9	0.8	0.9	0.85
	B	0.7	0.6	0.8	0.7
	C	0.85	0.7	0.85	0.8
	D	0.75	0.7	0.75	0.7

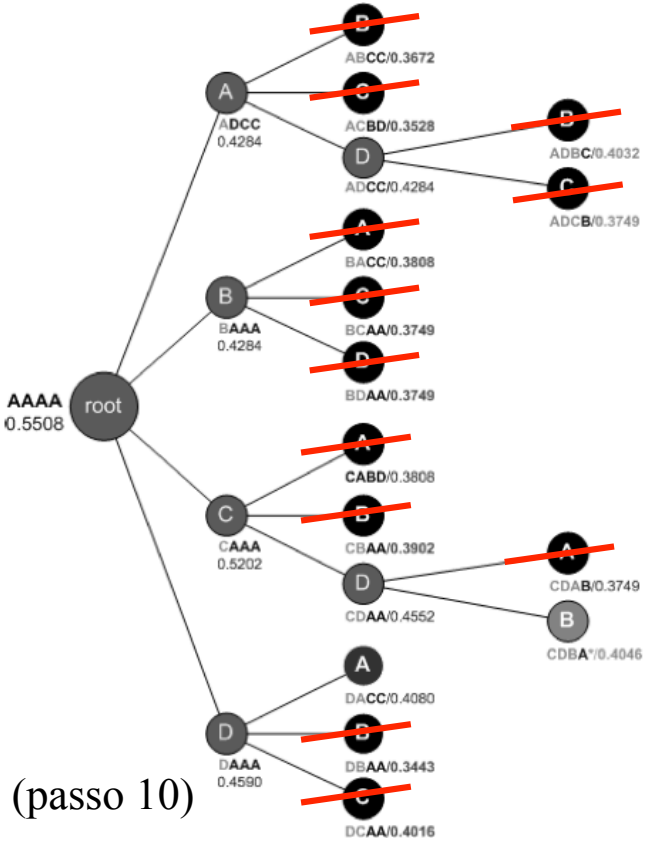


(passo 8)

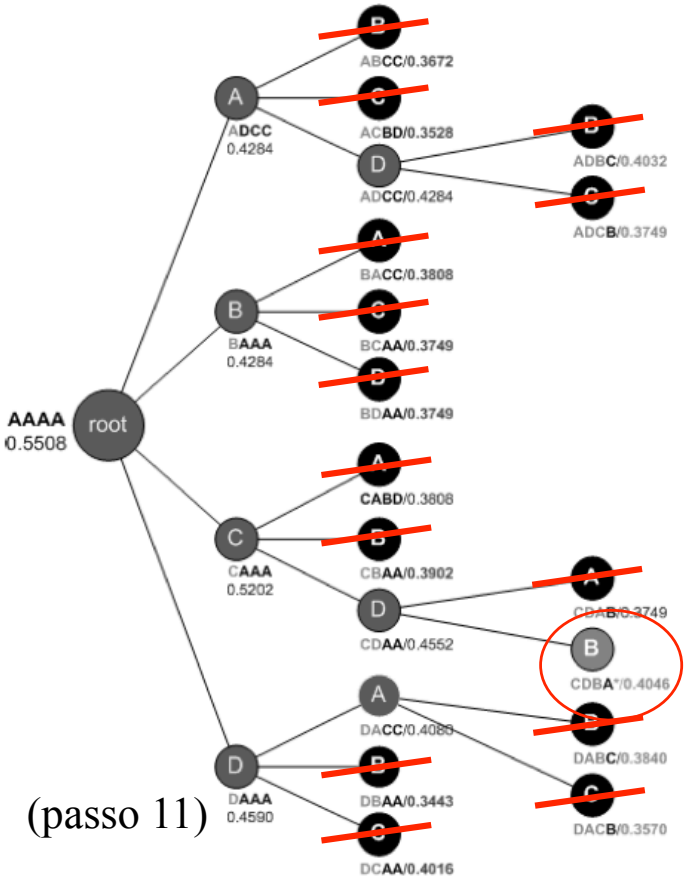


(passo 9)

Algoritmo Branch and Bound



(passo 10)



(passo 11)

0.4046<0.45
Nenhuma solução é válida

Problemas com a Busca Heurística

- E o caso onde o menor caminho da primeira cidade leva a uma cidade com caminhos muito longos?
  - Neste caso, poder-se-ia voltar atrás e tomar o segundo menor caminho, etc.
  - *o processo de “olhar para frente e voltar atrás” pode levar tempo
- O tempo gasto na avaliação da função heurística para selecionar um nó para a expansão deve ser recuperado por uma redução correspondente no tamanho do espaço de busca explorado.

Busca A*

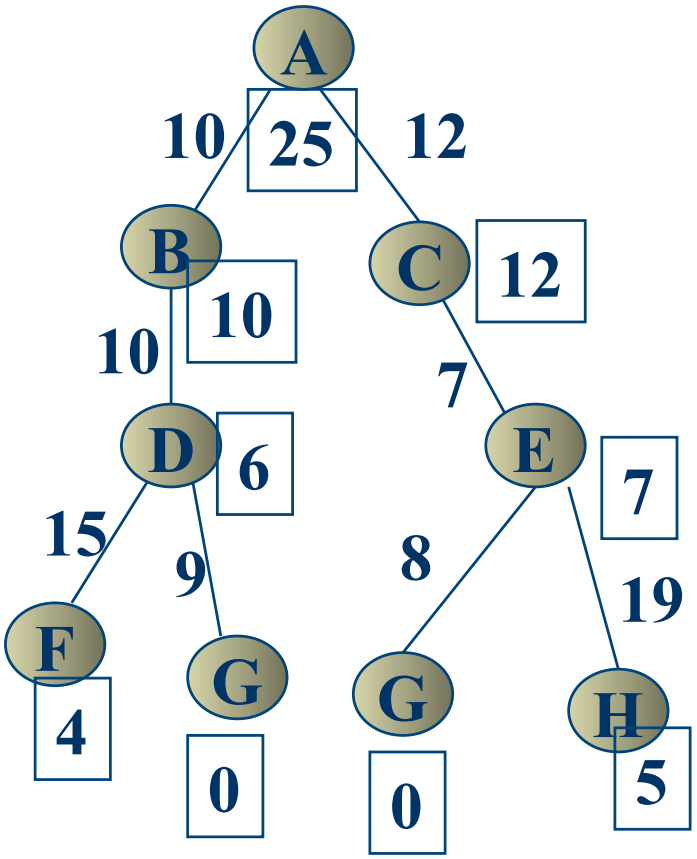
- Qualidade da busca depende do quão melhor for a estimativa
- Avalia os nós a serem escolhidos combinando ***g(n)***, o custo para alcançar cada nó, e ***h(n)***, o custo para ir de um nó *n* até o **nó objetivo:**

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

- *h(n)* é uma estimativa do custo do caminho de um nó *n* até o nó objetivo, esta deve ser simples de ser computada
- Desta forma, *f(n)* representa o custo estimado da solução de custo mais baixo passando por *n.*

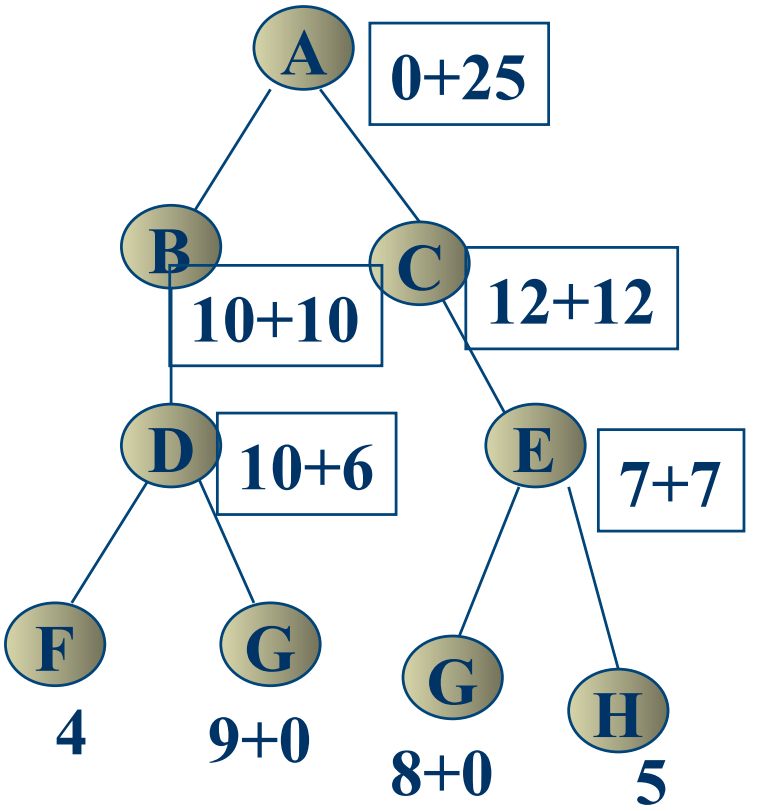
Exemplo de Busca A*

- Considere as distâncias em linha reta entre as cidades e a cidade G, representadas dentro dos quadros de cada nó
- Considere ainda o problema de sair da cidade A e ir até a cidade G, considerando o melhor caminho



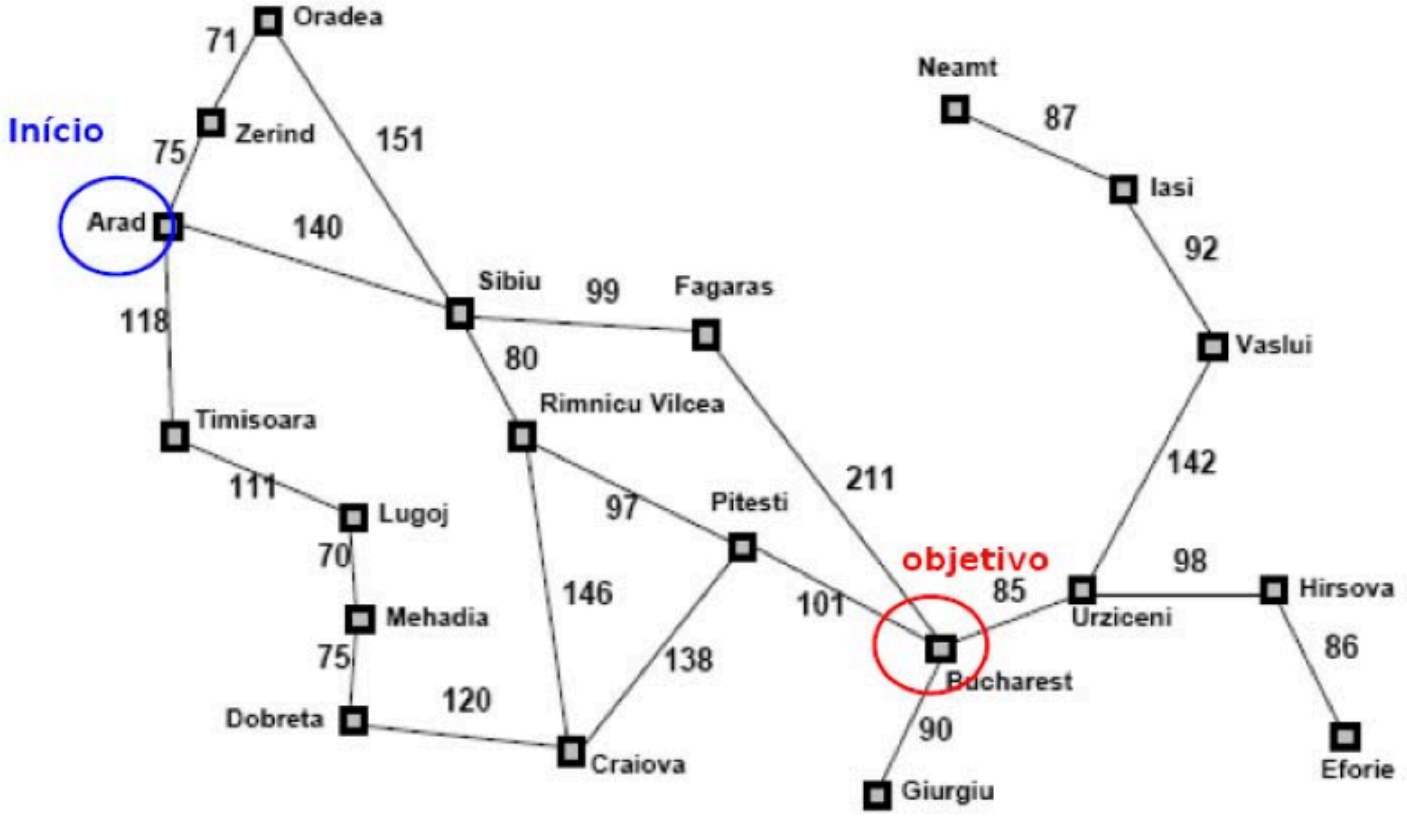
Exemplo de Busca A*

- Considere as distâncias em linha reta entre as cidades e a cidade G, representadas dentro dos quadros de cada nó
- Considere ainda o problema de sair da cidade A e ir até a cidade G, considerando o melhor caminho



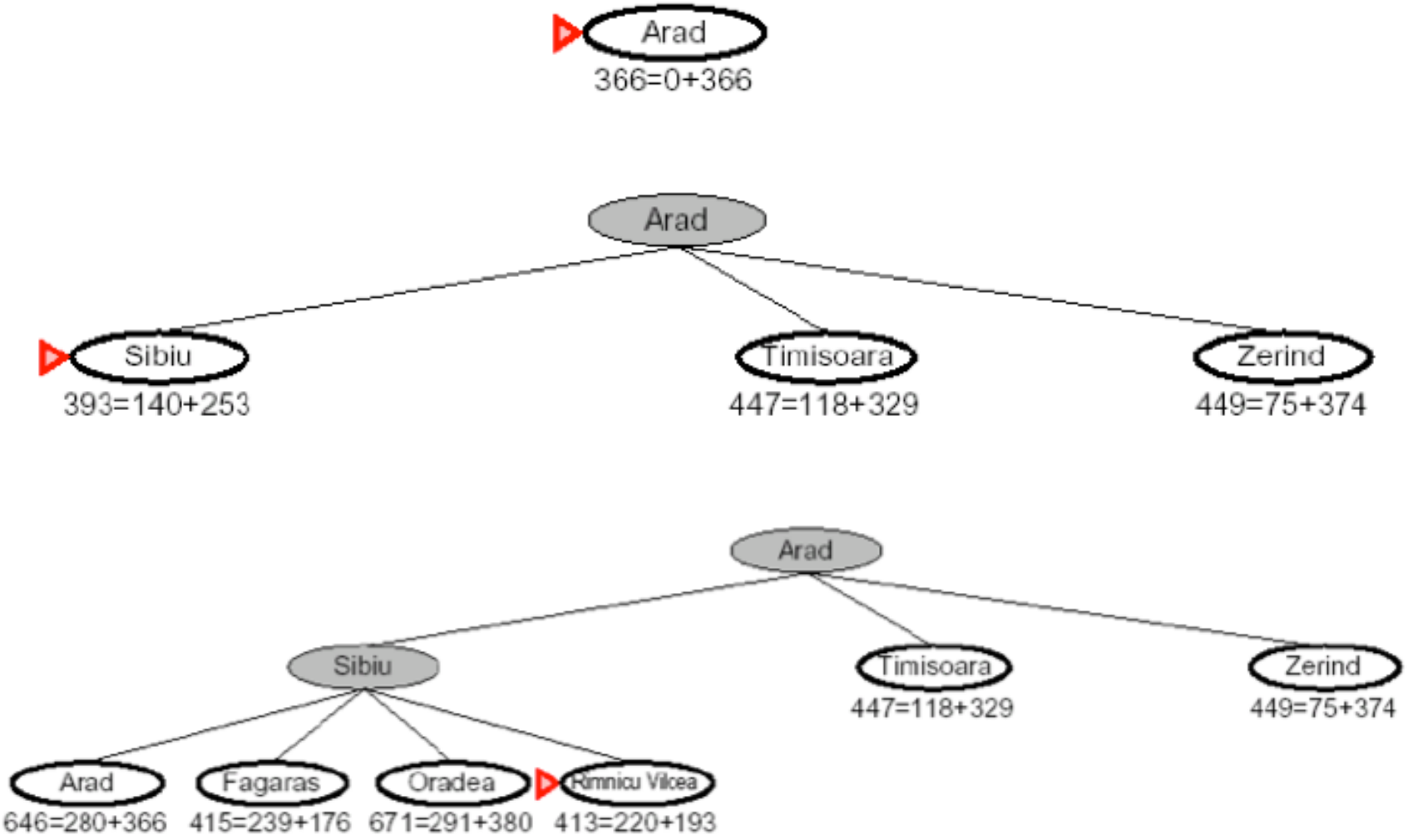
Busca A*

- Ex: Chegar de *Arad* a *Bucharest*

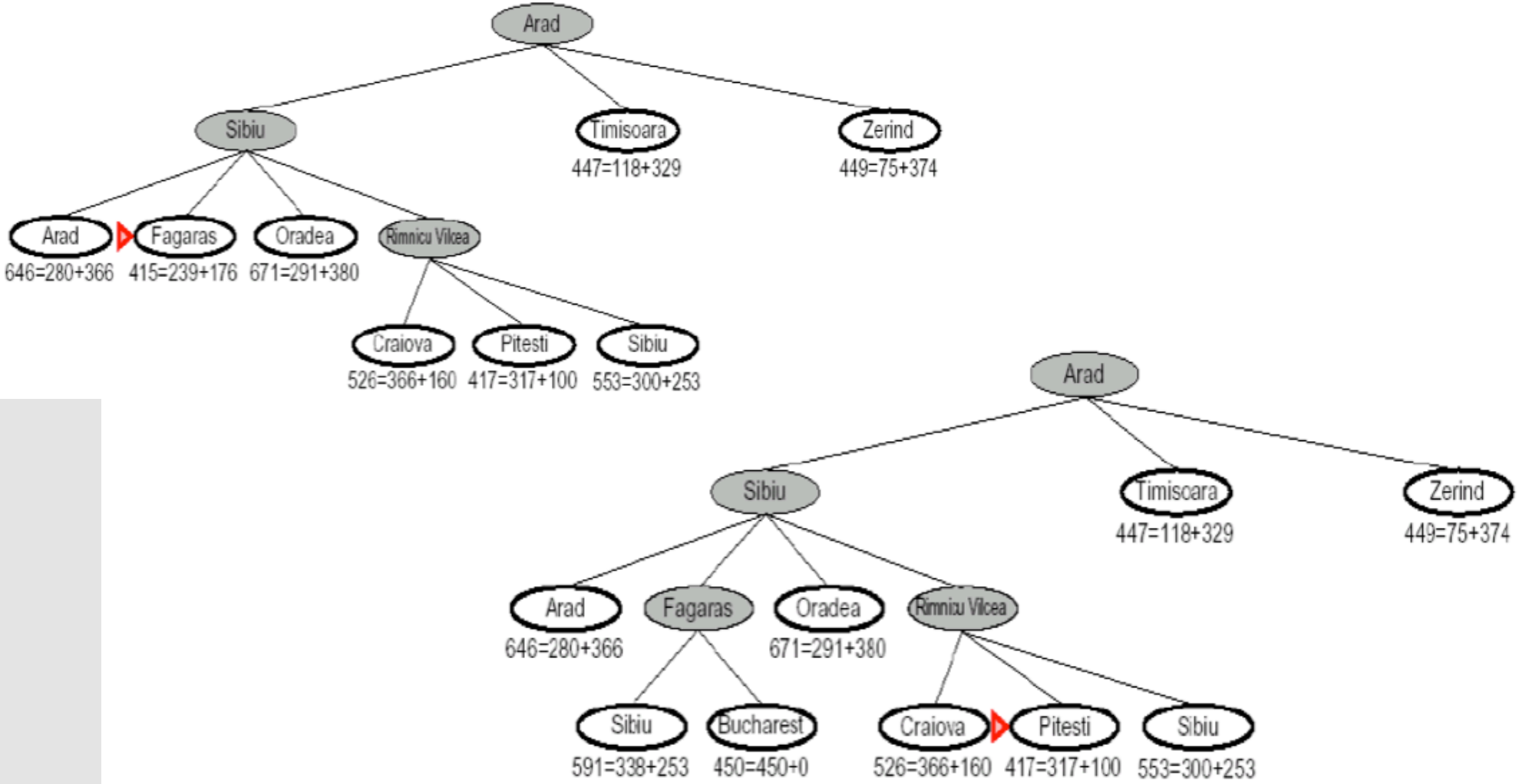


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

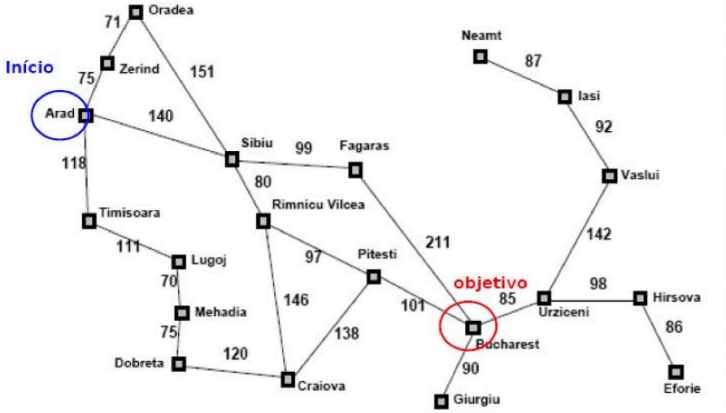
Busca A*


Arad
366=0+366
Arad
Sibiu
393=140+253
Timisoara
447=118+329
Zerind
449=75+374
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
413=220+193

Busca A*


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253
Arad
Sibiu
Timiscara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimniu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

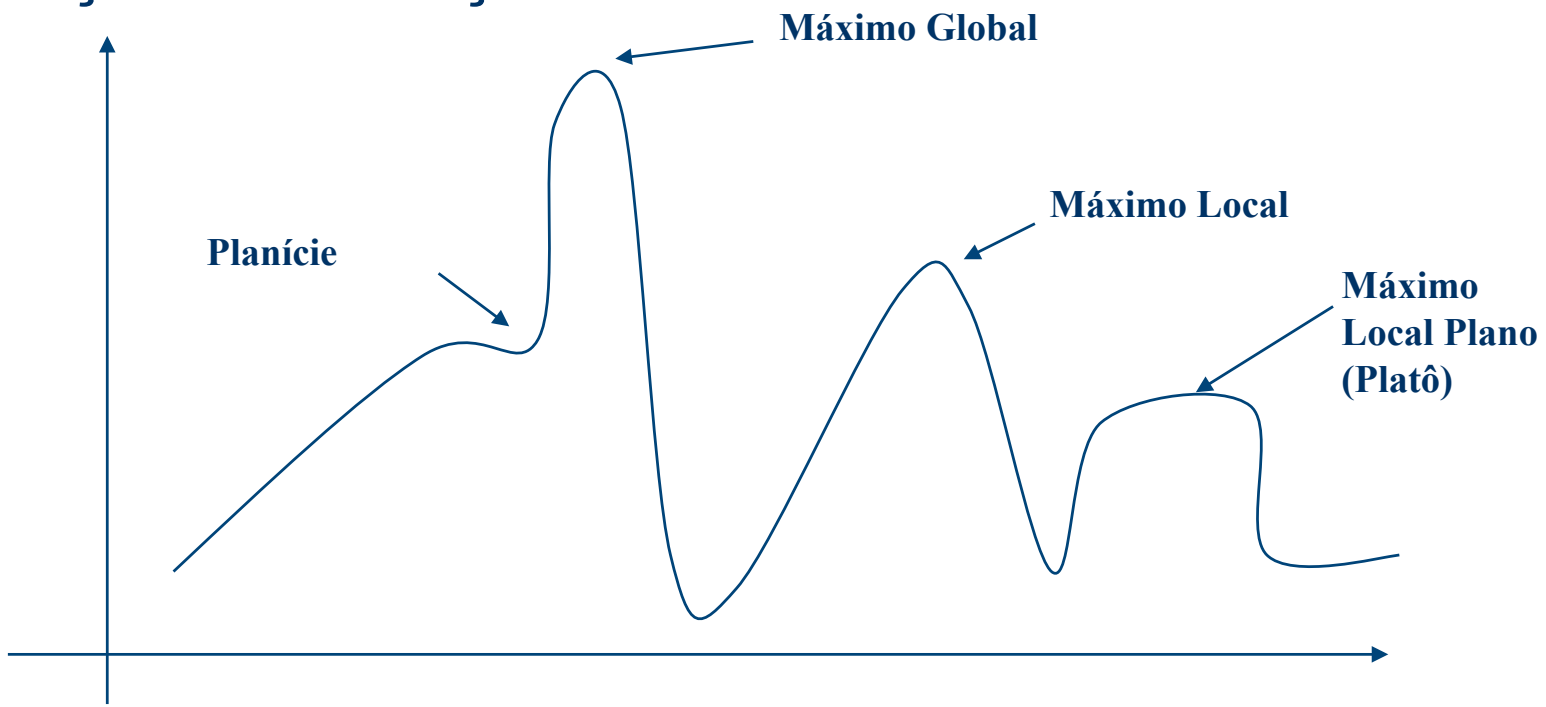
Busca A*


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


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

Subida de Encosta (Hill Climbing)

- Investigam pontos adjacentes do espaço de busca e movem-se na direção indicada pela melhora no valor da função de avaliação.



Subida de Encosta (Hill Climbing)

- Escolhe uma solução inicial aleatoriamente e altera um elemento desta iterativamente, segundo o menor valor, na busca de encontrar o estado final
- É um método de busca local, a cada momento o algoritmo considera somente os estados imediatamente acessíveis a partir do estado atual
- Problemas com Hill Climbing
  - Máximo local: existe um pico mais elevado
  - Planície: todos os pontos vizinhos levam ao mesmo valor
  - Platô: representa um máximo local plano

- Dúvidas???