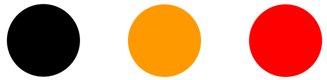


Agentes que Fazem Busca

Prof. Sérgio R. P. da Silva
Profa. Josiane M. P. Ferreira

Texto base:

Stuart Russel e Peter Norving - “Inteligência Artificial”
David Poole, Alan Mackworth e Randy Goebel - “*Computational Intelligence – A logical approach*”
maio/2007

- Agentes inteligentes devem agir de forma que o ambiente passe por um seqüência de estados que maximizam a sua medida de desempenho.

- Esta descrição é muito geral e difícil de traduzir em uma implementação de sucesso para um agente.

- Esta tarefa pode ser simplificada se o agente tiver um <u>**objetivo**</u>.

Tarefas para a resolução de problemas

- Passos:
  - Formulação do OBJETIVO
    - Nível de detalhe das ações
  - Formulação do PROBLEMA
    - Decidir quais ações e estados considerar
  - Busca
    - Dada uma seqüências de ações, qual a melhor?
  - Execução

Formulação → Busca → Execução

Estrutura básica de um agente resolvedor de problemas

```
Função agente-resolução-problemas-simples(percepção) retorna ação
  entrada: percepção, uma percepção
  var estáticas: seq, uma seqüência de ações, inicialmente vazia
            estado, uma descrição do estado atual do mundo
            objetivo, um objetivo, inicialmente nulo
            problema, uma formulação de problema

estado ← ATUALIZAR-ESTADO(estado, percepção)

Se seq está vazia então faça
    objetivo  ← FORMULAR-OBJETIVO (estado)
    problema  ← FORMULAR-PROBLEMA (estado,objetivo)
    seq ← BUSCA(problema)

ação ← PRIMEIRO(seq)
seq ← RESTO(seq)

retornar acão
```

Problemas de representação de conhecimento

- O que existe em um estado?
  - As manchas solares são relevantes para predizer o valor das ações no mercado? A cor da camisa de um jogador de xadrez é relevante na escolha da próxima jogada?
- Em que <u>nível de abstração</u> ou detalhe deve-se descrever o mundo.
  - Um nível muito fino fará "perder a visão floresta pelas das árvores". Um nível muito alto e perderemos detalhes críticos para resolver o problema.
- O <u>número de estados</u> depende da representação e do nível de abstração escolhidos.

Problemas bem definidos e soluções

- Um problema é composto por:
  - **Estado inicial** - o estado em que o agente começa
  - Uma descrição da **ações possíveis** (Função sucessor)
    - Uma forma de gerar potenciais soluções
  - **Teste de objetivo**, que determina se um dado estado é o estado objetivo
  - **Custo de um caminho**, é uma função que atribui uma medida de custo a um determinado caminho.

Espaço de Estados

- O estado inicial e a função sucessor definem implicitamente o **espaço de estados**
- **É o conjunto de todos os estados acessíveis a partir do estado inicial**
- Forma um grafo em que os nós são estados e os arcos as ações
- Um **caminho** no espaço de estados é uma seqüência de estados conectados por uma seqüência de ações

1. **Problemas com um único estado:**
   - Quando é possível calcular exatamente que estados estaremos após uma seqüência de ações.

3. **Problemas com múltiplos estados:**
   - Quando sabemos os efeitos das ações mas temos acesso limitado ao estado que encontra-se o ambiente.

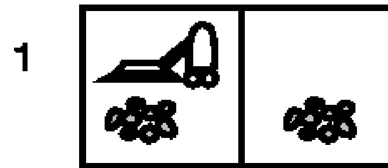
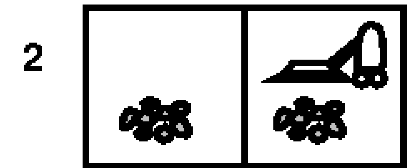
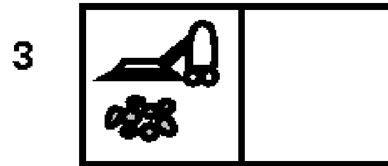
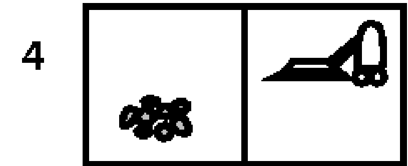
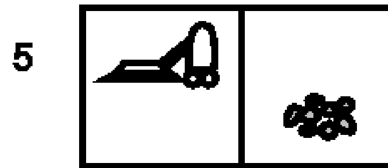
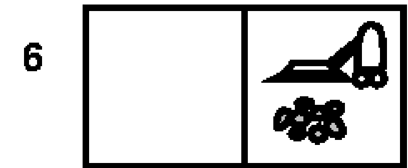
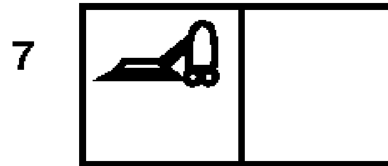
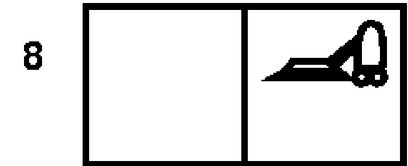
5. **Problemas de contingência:**
   - As ações realizadas pelo agente podem resultar em efeitos não esperados (não determinístico), não há como fazer uma árvore que garanta o resultado.

7. **Problemas de exploração:**
   - Quando o agente não tem informações sobre o efeitos de suas ações, tem apenas do que foi aprendido anteriormente pela exploração do problema.

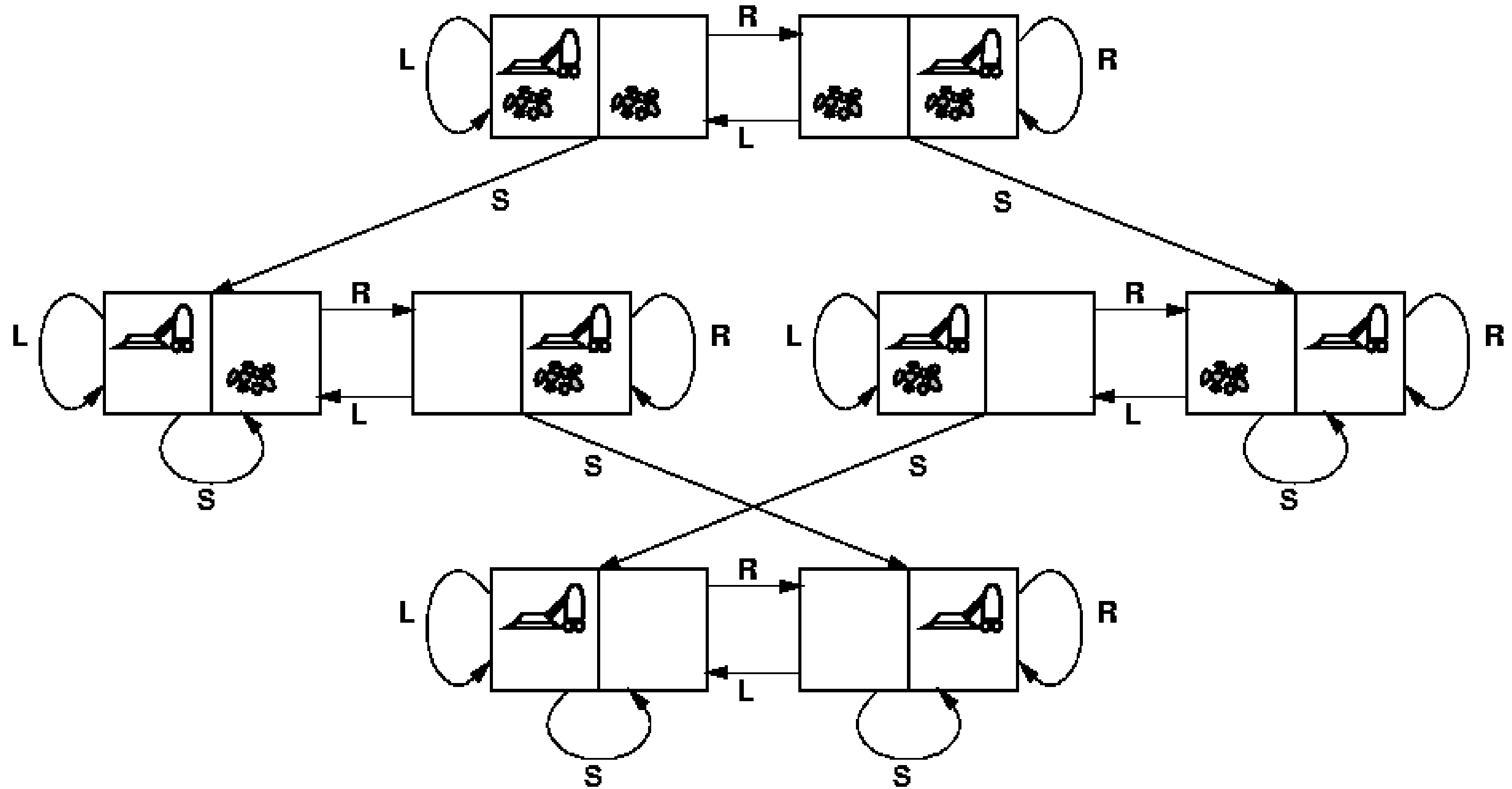
- Problemas de estado único
  - O agente sabe em que estado está.
  - O agente sabe o que cada uma das ações fazem.
  - O agente calculará exatamente o que fazer.
- Problemas de múltiplos estados
  - O agente sabe o que cada uma das ações fazem.
- Problemas de contingência
  - O agente deve usar os sensores enquanto age.
  - A solução é uma árvore de ações ou uma política.
  - Às vezes pode ser viável o entrelaçamento (agir antes de buscar).
- Problemas de exploração
  - O espaço de estados é desconhecido.

Exemplo

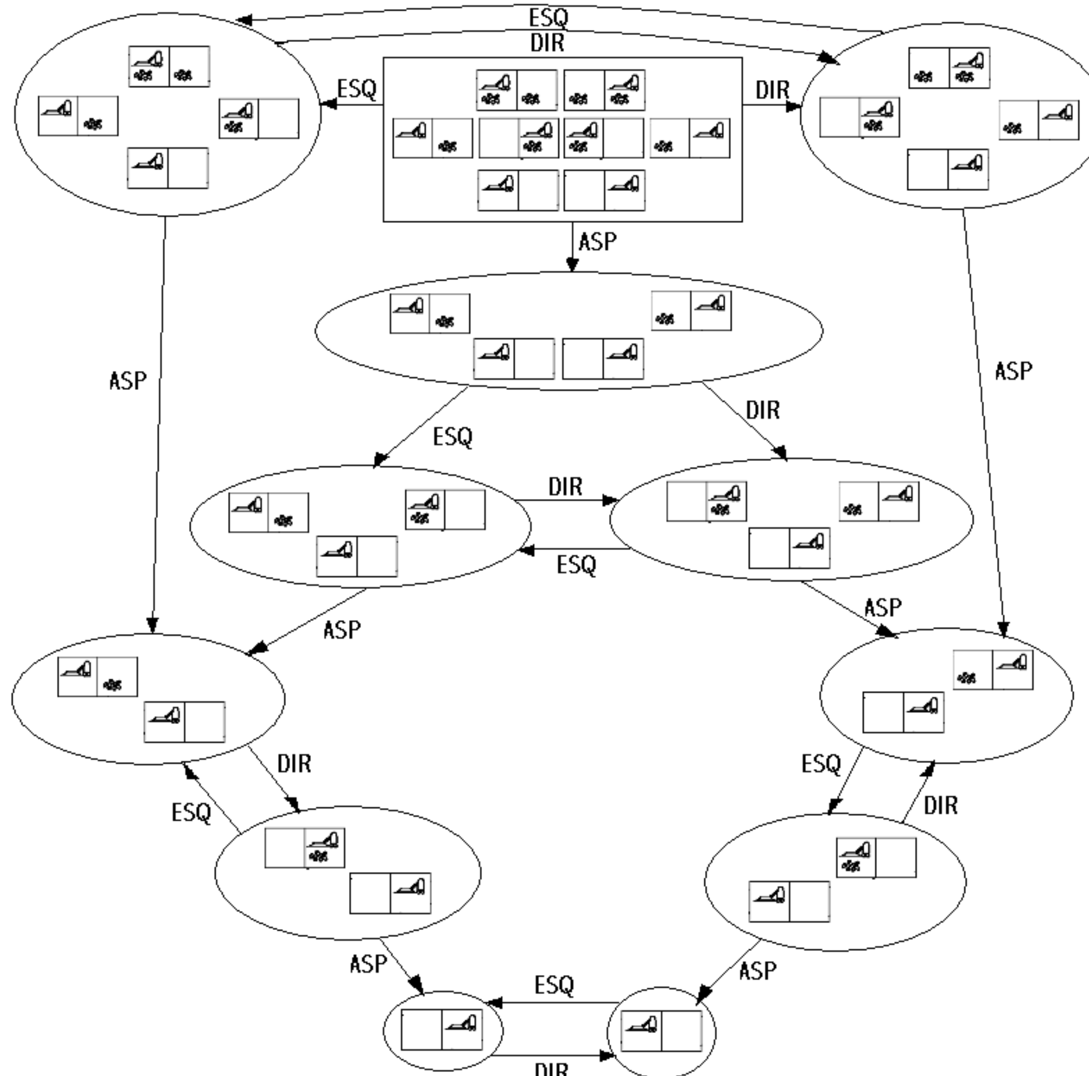
- Estado único
  - Início no estado 5
- Múltiplos estados
  - Início em
    - {1, 2, 3, 4, 5, 6, 7, 8)
  - Ação → Direita
    - {2, 4, 6, 8}
- Contingência
  - Ação → Aspirar
    - Pode sujar um carpete limpo
  - Sensoriamento
    - Local
    - Sujo ou não

1

2

3

4

5

6

7

8


Espaço de estados em problemas de estado único



Espaço de estado em problemas de múltiplos estados

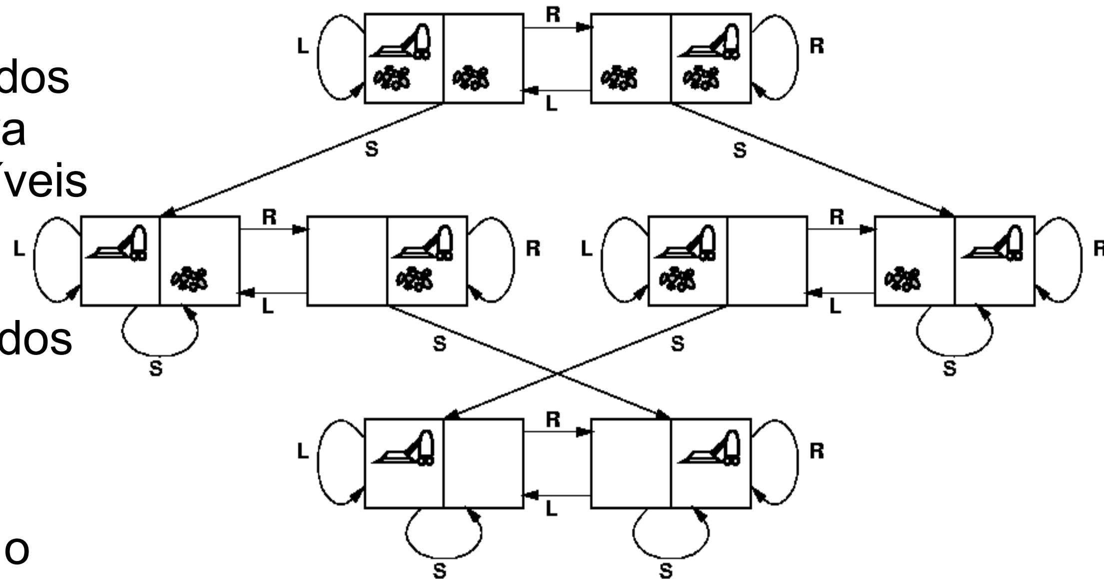


Exemplos de problemas

- Miniproblemas
  - Se destinam a ilustrar ou exercitar diversos métodos de resolução de problemas
  - Podem ter uma descrição concisa e exata
  - Podem ser usados por diversos “buscadores”, para se comparar o desempenho dos algoritmos

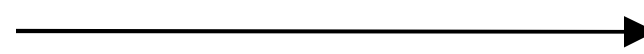
Mundo do aspirador de pó

- Estados possíveis: Os 8 ao lado
- Estado inicial: Um dos 8 estados
- Função sucessor: Gera os estados válidos que resultam da tentativa de executar as três ações possíveis (esq, dir, asp)
- Teste de objetivo: Verifica se todos os quadrados estão limpos
- Custo do caminho: Cada passo custa 1 e o custo do caminho é o número de passos do caminho
- Um ambiente com n posições tem $n2^n$ estados



O problema de quebra-cabeça

2	8	3
1	6	4
7		5

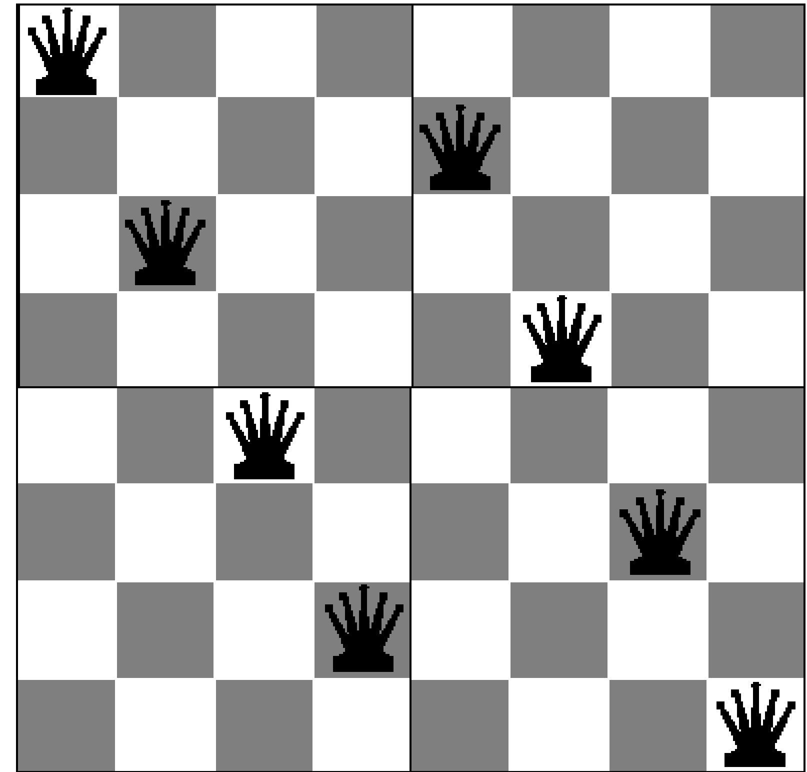


	1	2
3	4	5
6	7	8

- Estados: Matriz 3x3 com a especificação da posição de cada símbolo de 1-8 e o branco
- Estado Inicial: qualquer um estados especificados acima
- Função sucessor: gera os estados válidos resultantes das quatro ações mover o espaço vazio para Dir, Esq, Cima, Baixo
- Teste de objetivo: verifica se o estado corresponde ao estado objetivo
- Custo do caminho: Número de passos do caminho (custo 1 por passo)

O problema das 8 rainhas

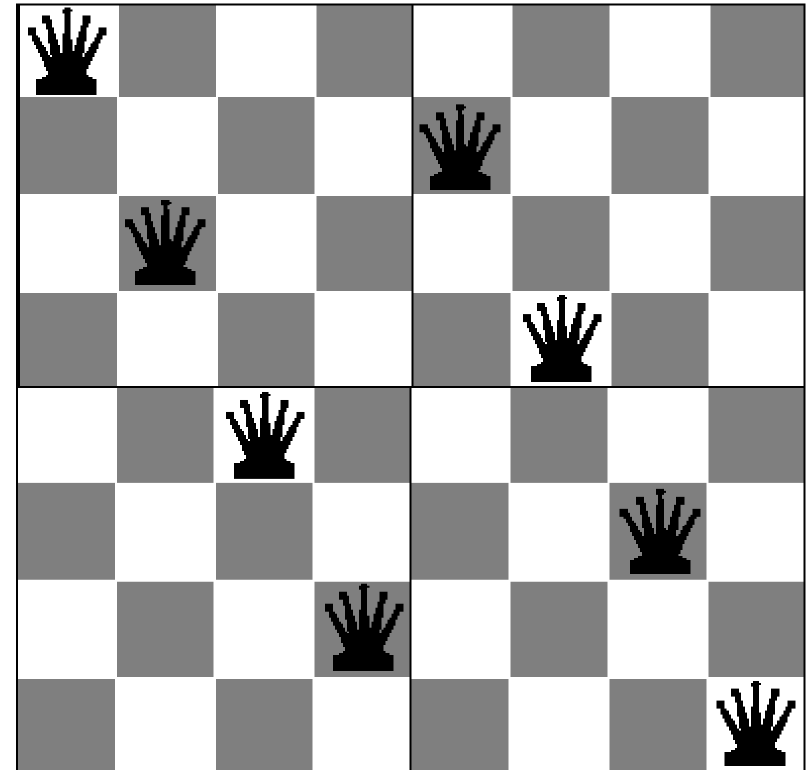
- Estados: Qualquer arranjo de 0 a 8 rainhas no tabuleiro
- Estado inicial: nenhuma rainha no tabuleiro
- Função sucessor: Colocar uma rainha em uma casa
- Teste de Objetivo: 8 rainhas estão no tabuleiro. Nenhuma está sendo atacada.
- Custo do Caminho: Zero
- 3×10^{14} possíveis seqüências a serem investigadas



O problema das 8 rainhas

- Estados: n rainhas (n de 0 a 8) no tabuleiro, uma por coluna nas n colunas mais a esquerda, sem ataques
- Função sucessor: Colocar uma rainha em uma casa cuja coluna seja a primeira à esquerda que não esteja ameaçada por outra rainha
- 2057 possíveis seqüências a serem investigadas

A formulação do problema é FUNDAMENTAL!!!



Cripto-aritmética

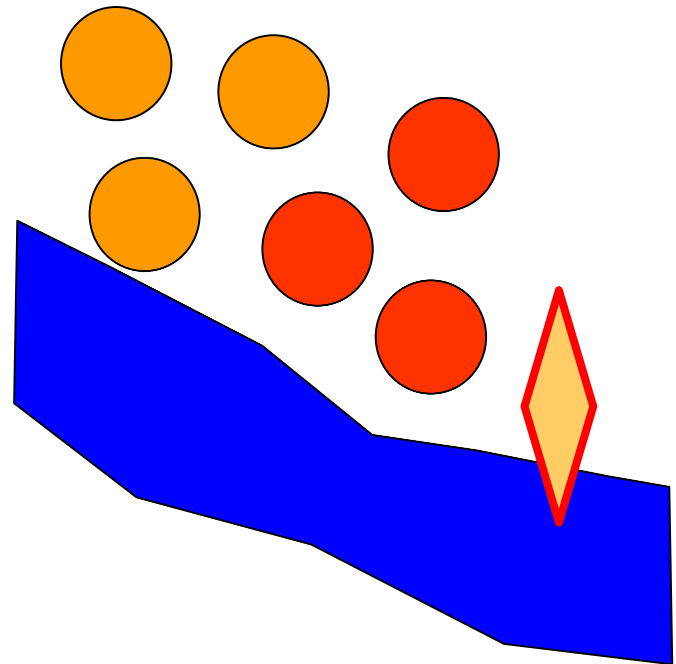
- Estados: Algumas letras trocadas por dígitos
- Estado inicial: nenhum número atribuído a uma letra
- Função sucessor: Trocar todas as ocorrências de uma letra por um dígito que ainda não esteja sendo usado
- Teste de Objetivo: Somente dígitos e a soma está correta
- Custo do Caminho: Zero. Todas as soluções são igualmente válidas

```
FORTY      29786
+ TEN      + 850
+ TEN      + 850
-----      -----
SIXTY      31486
```

```
F = 2, O = 9, R = 7…
```

Missionários e Canibais

- Estados:
  - uma seqüência de três números representando
    - <missionários, canibais, barco>
- Estado inicial: <3,3,1>
- Função sucessor:
  - Levar 1 missionário e 1 canibal,
  - Levar 2 missionários,
  - Levar 2 canibais,
  - Levar 1 missionário,
  - Levar 1 canibal
- Teste de Objetivo:verificar se o estado é igual a <0, 0, 0> (estado objetivo)
- Custo do Caminho: número de passagens pelo rio



Busca [1]

- Geralmente não temos um algoritmo para resolver um problema, mas somente uma especificação do que é uma solução
  - Temos que buscar pela solução
- **Busca** é uma enumeração de um conjunto de potenciais soluções parciais para um problema, de tal forma que se possa verificar se elas são soluções de verdade.
- **Busca** é uma forma de implementar não-determinismo do tipo “não-sei”.
- Até agora vimos como converter um problema semântico de achar uma conseqüência lógica para um problema sintático de busca de derivações que formam uma prova.

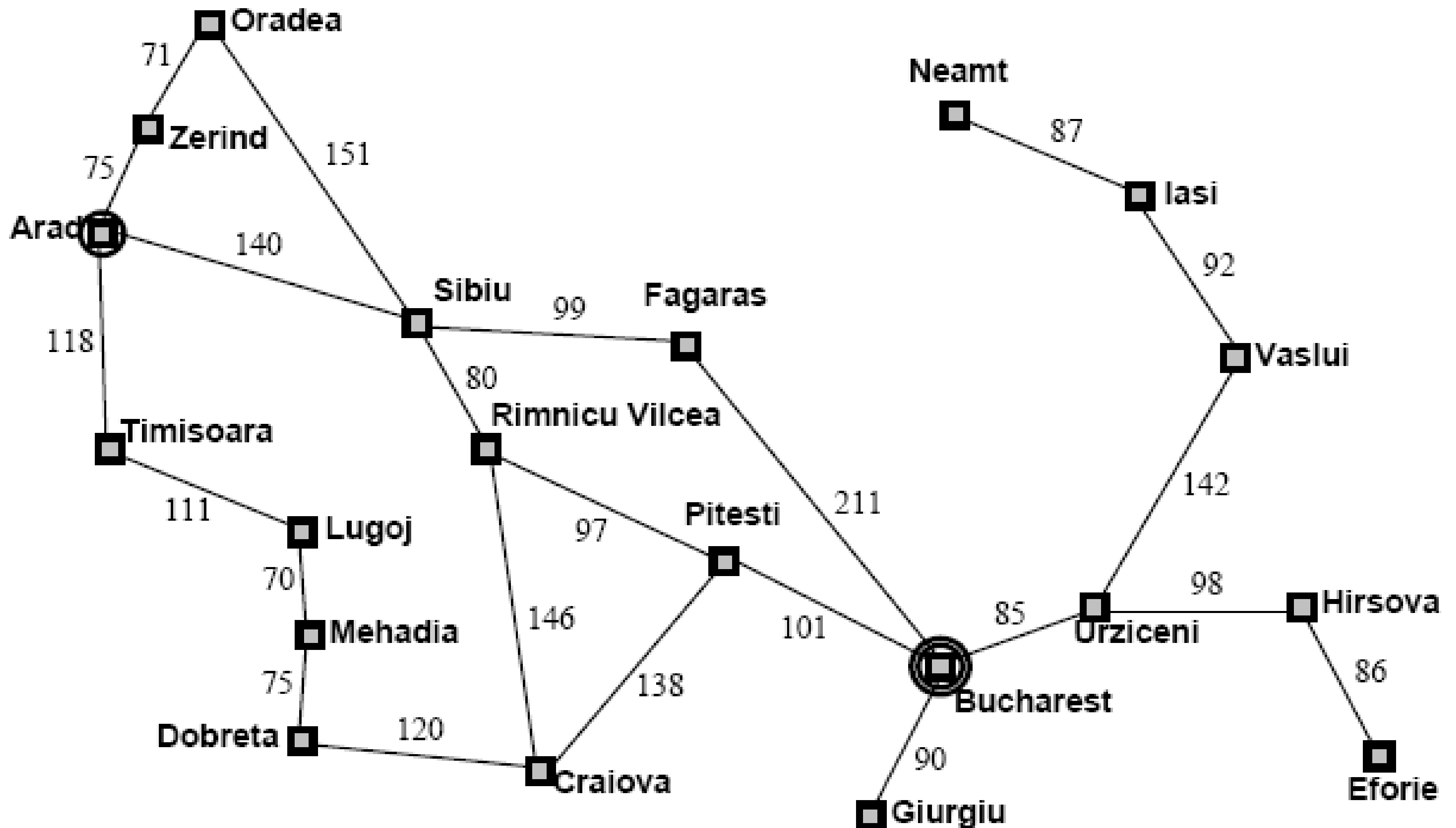
Busca [2]

- Para realizarmos uma busca precisamos de:
  - Uma definição de uma solução potencial.
  - Um método de gerar potenciais soluções, de preferência de uma forma inteligente.
  - Uma forma de verificar se uma solução potencial é de fato uma solução.

Busca em Grafos

- Existem muitas formas de representar um problema como um grafo, mas duas formas merecem atenção:
  - **Grafo do Espaço de Estados**: no qual um nó representa um estado do mundo e um arco representa uma mudança de um estado para outro no mundo.
    - Ex.: Achar um caminho para um robô.
  - **Grafo do Espaço de Problemas**: no qual um nó representa um problema a ser resolvido e um arco representa decomposições alternativas do problemas.
    - Ex.: Achar uma derivação SLD para uma prova (algoritmo *top-down*)

Um grafo de busca para achar um caminho para um robô na Romênia



Um grafo de busca para uma resolução SLD

$a \leftarrow b \wedge c.$ $\quad a \leftarrow g.$

$a \leftarrow h.$ $\quad b \leftarrow j.$

$b \leftarrow k.$ $\quad d \leftarrow m.$

$d \leftarrow p.$ $\quad f \leftarrow m.$

$f \leftarrow p.$ $\quad g \leftarrow m.$

$g \leftarrow f.$ $\quad k \leftarrow m.$

$h \leftarrow m.$ $\quad p.$

$?a \wedge d$

$yes \leftarrow a \wedge d$

$yes \leftarrow b \wedge c \wedge d$ $\quad yes \leftarrow g \wedge d$ $\quad yes \leftarrow h \wedge d$

$yes \leftarrow j \wedge c \wedge d$ $\quad yes \leftarrow k \wedge c \wedge d$ $\quad yes \leftarrow m \wedge d$ $\quad yes \leftarrow f \wedge d$ $\quad yes \leftarrow m \wedge d$

$yes \leftarrow m \wedge c \wedge d$ $\quad yes \leftarrow m \wedge d$ $\quad yes \leftarrow p \wedge d$

$yes \leftarrow d$

$yes \leftarrow m$ $\quad yes \leftarrow p$

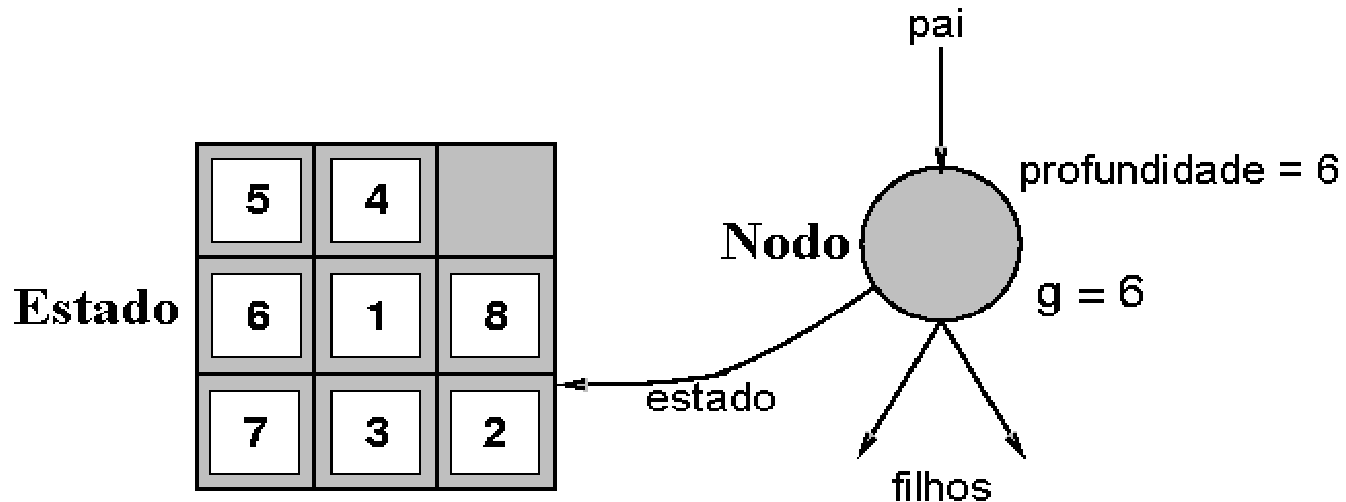
$yes \leftarrow$

Busca em Grafos [1]

- Um **grafo** consiste de um conjunto de **nós** N e um conjunto de pares ordenados de nós A, denominados de **arcos** (ou arestas)
- O nó n_2 é um vizinho do nó n_1 se existe um arco de n_1 para n_2. Isto é, se $\langle n_1, n_2 \rangle \in A$
- Um **caminho** é uma seqüência de nós $\langle n_0, n_1, \ldots, n_k \rangle$ tais que $\langle n_{i-1}, n_i \rangle \in A$
- Dado um conjunto de **nós iniciais** e um conjunto de **nós objetivos**, uma **solução** é um caminho de um nó inicial a um nó objetivo

Estados X Nós

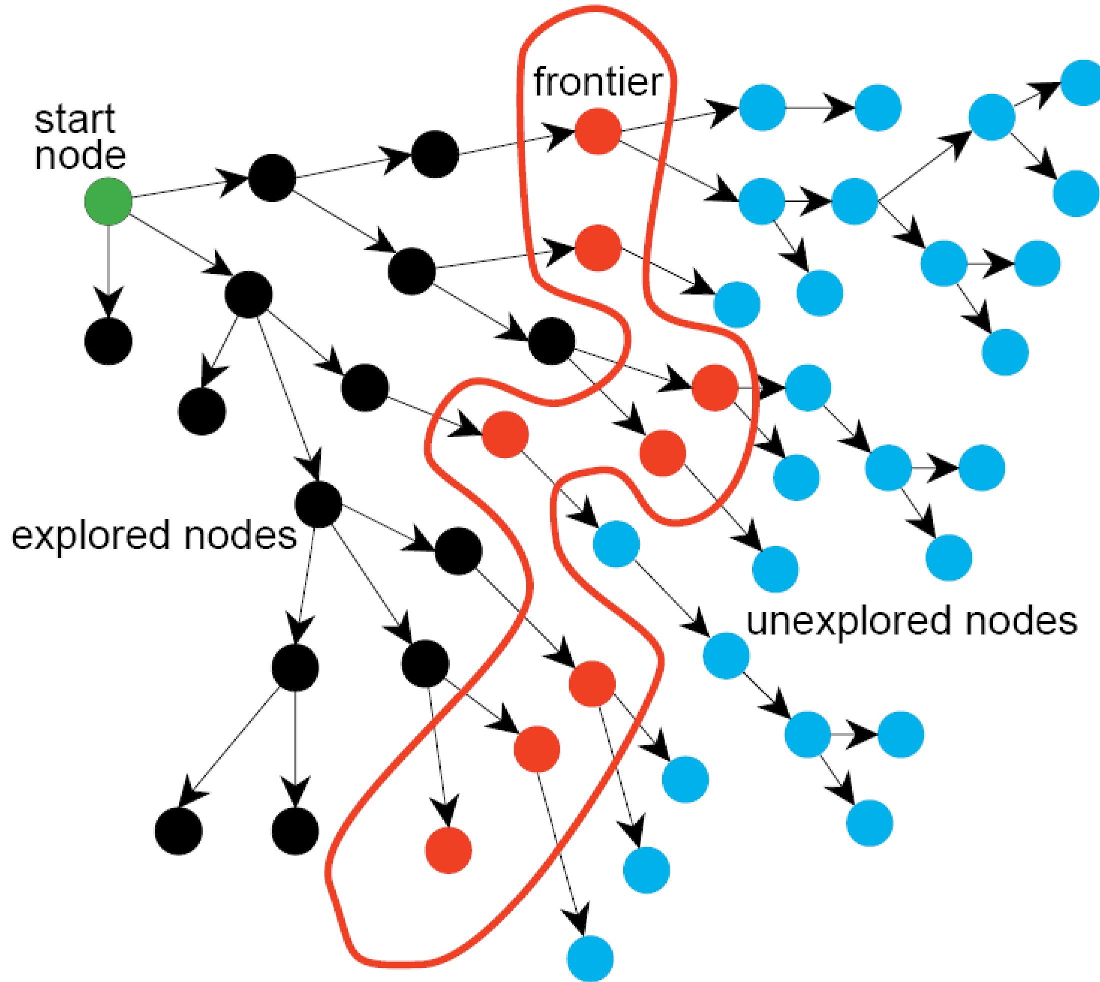
- O estado é uma **representação** de uma configuração física.
- O nó é uma **estrutura de dados**
- Estados não têm pais, filhos, profundidade ou custo;



Busca em Grafos [2]

- Algoritmo de busca genérico:
  - Dado um grafo, um conjunto de nós iniciais e de nós objetivos, explore os caminhos de forma incremental a partir dos nós iniciais.
- Mantenha um **fronteira** de caminhos a partir do nó inicial que já tenha sido explorada.
- Com o avanço da busca, a fronteira expande sobre os nós não explorados até que um nó objetivo é encontrado
- A forma como a fronteira é expandida define a **estratégia de busca**

Resolução de problemas pela busca em grafos

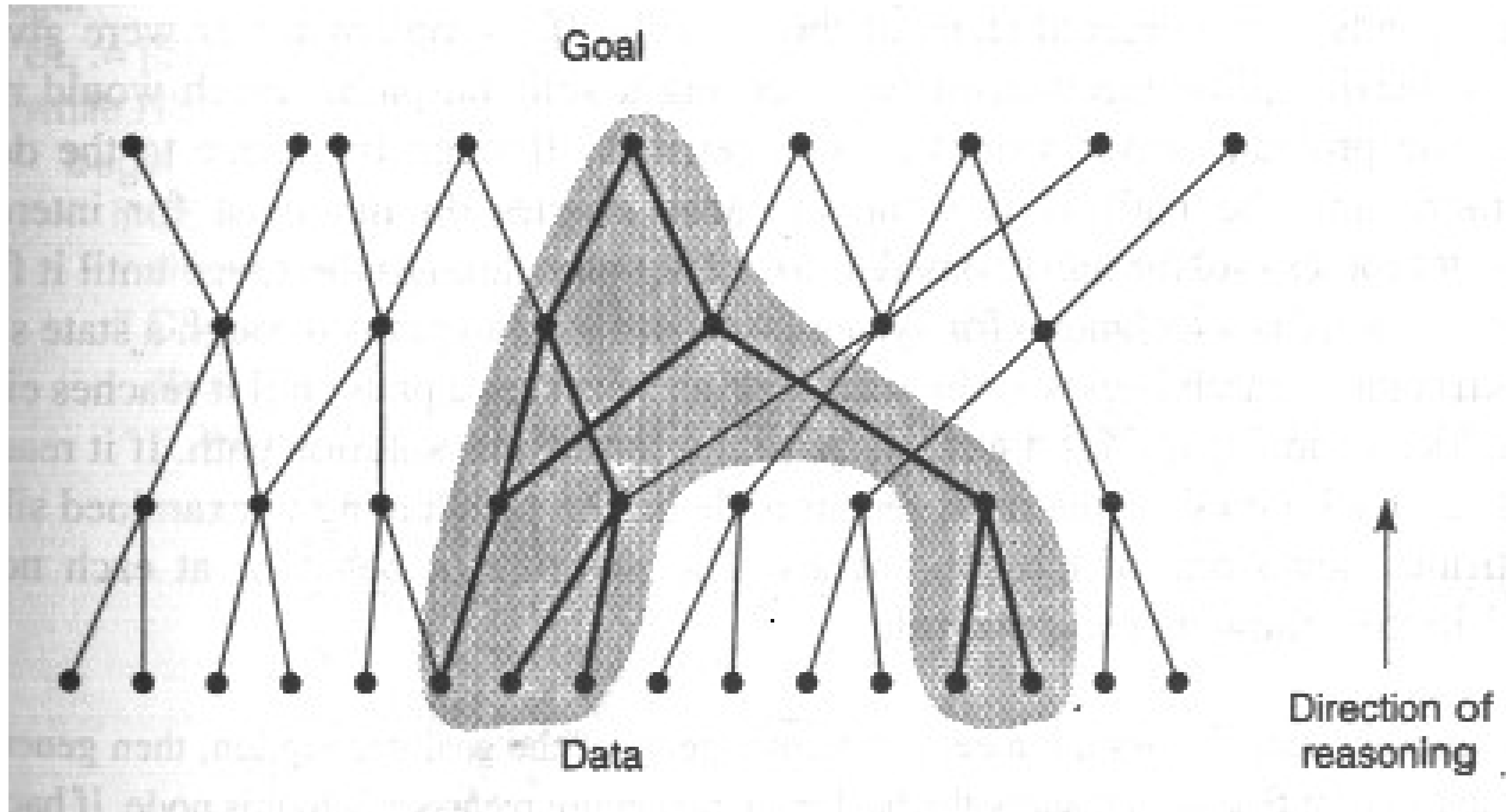


Estratégias de busca

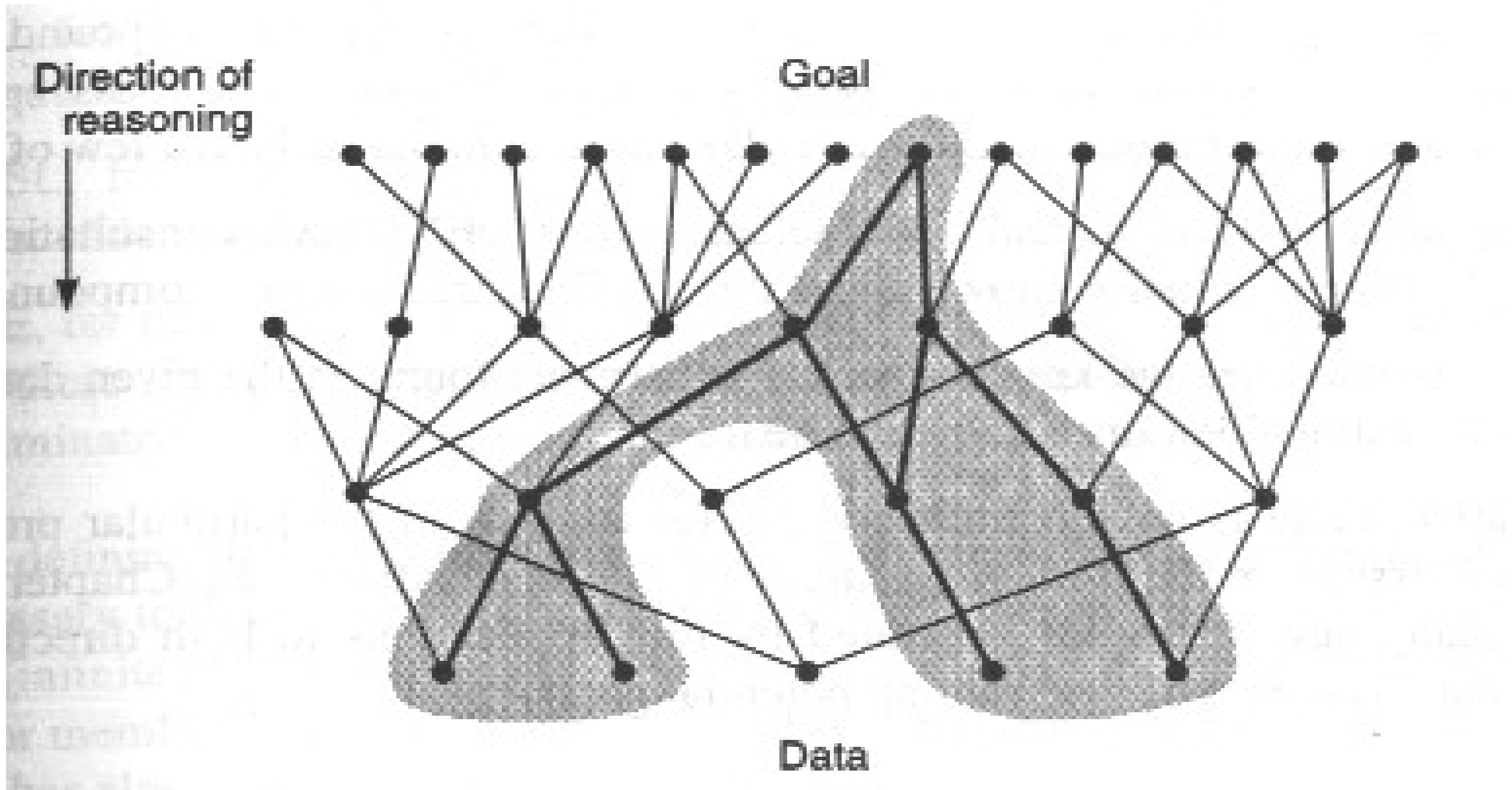
- A função de exploração (ou função sucessor) cria novos nós usando os operadores do problema para criar os estados correspondentes.

- Direção da expansão:
  1. Do estado inicial para um estado final (*busca dirigida por dados* ou *encadeamento para frente*).
  2. De um estado final para o estado inicial (*busca baseada em objetivo* ou *encadeamento para trás*).
  3. Busca bi-direcional (é realizada nas duas direções ao mesmo tempo).

Encadeamento para frente

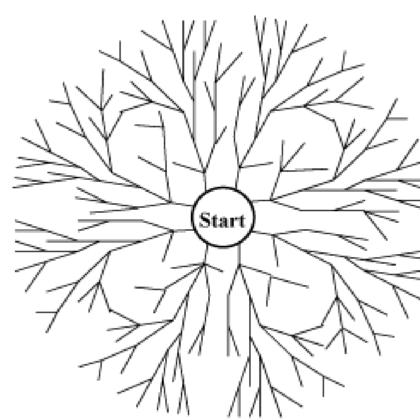


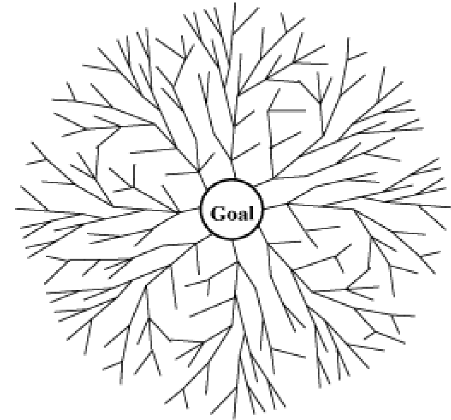
Encadeamento para trás



Busca Bidirecional

- Busca em duas direções:
  - Para frente, a partir do nó inicial, e
  - Para trás, a partir do nó final (objetivo)
- A busca pára quando os dois processos geram um mesmo estado intermediário.
- É possível utilizar *estratégias* diferentes em cada direção da busca.





Algoritmo Geral para Busca em Grafos

entrada:

um grafo,

um conjunto de nós inicias,

um procedimento booleano *objetivo(n) que testa se n* é um nó objetivo.

fronteira := {*s* : *s* é um nó inicial};

enquanto *fronteira não for vazia*:

selecione e **remova** o caminho $<n_0, . . . , n_k>$ da *fronteira;*

se *objetivo*(n_k)

retorne $<n_0, . . . , n_k>$;

para cada vizinho *n de* n_k

adicione $<n_0, . . . , n_k, n>$ a *fronteira;*

fim_enquanto.

Algoritmo de Busca em Grafos: pressuposições

- Assumimos que depois que o algoritmo de busca retornou uma resposta, ele pode ser questionado por mais respostas e o procedimento continuará.
- O valor que é selecionado da fronteira em cada estágio faz parte da definição da estratégia de busca.
- Os *vizinhos* definem o grafo.
- *é_objetivo* define o que é uma solução.

Algoritmo de Busca em Grafos em Prolog [1]

```
% Busca por um nó objetivo a partir do nó F0 sem
   devolver o caminho

busca(F0) :-
   selecione(No, F0, F1),
   é_objetivo(No).

busca(F0) :-
   selecione(No, F0, F1),
   vizinhos(No, NN),
   adicione _a_fronteira(NN, F1, F2),
   busca(F2).
```

Algoritmo de Busca em Grafos em Prolog [2]

```
% Busca por um caminho até nó objetivo a partir do nó F
% no(nó, caminho, custo_caminho)

busca_c(F0,[No|P]) :-
   selecione(no(No,P,PC),F0,F1),
   é_objetivo(No).

busca_c(F0,Path) :-
   selecione(no(No,P,PC),F0,F1),
   vizinhos(No,NN),
   adicione_caminhos(NN,no(No,P,PC),NF),
   adicione_a_fronteira(NF,F1,F2),
   busca_c(F2,Path).
```

Critérios de avaliação das estratégias de busca [1]

- Completude (completeza):
  - A estratégia **sempre** encontra uma solução quando existe alguma?
- Custo do tempo:
  - Quanto **tempo** gasta para encontrar uma solução? Normalmente medida em termos do número de nós expandidos.
- Custo de memória:
  - Quanta **memória** é necessária para realizar a busca? Normalmente medida pelo tamanho máximo que a lista de nós abertos (fronteira) assume durante a busca.
- Qualidade/otimicidade:
  - A estratégia encontra **a melhor solução** quando existem soluções diferentes?
    - Menor custo de caminho

Critérios de avaliação das estratégias de busca [2]

- O **fatorde ramificação** de um nó é o seu número de vizinhos.
- As complexidades de tempo e espaço são medidas em termos de:
  - *b* : Fator de ramificação máximo da árvore de busca.
  - *d* : Profundidade da solução de menor custo.
  - *m* : Profundidade máxima do espaço de estados (pode ser ∞).

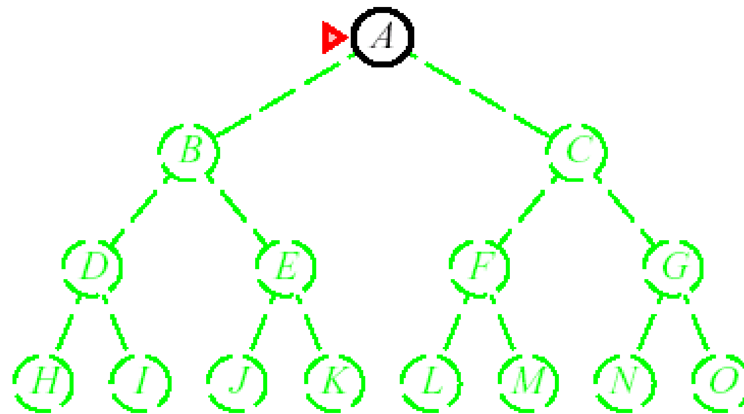
Busca Desinformada

- Busca em profundidade
- Busca em largura
- Busca pelo custo mínimo – Busca Gulosa

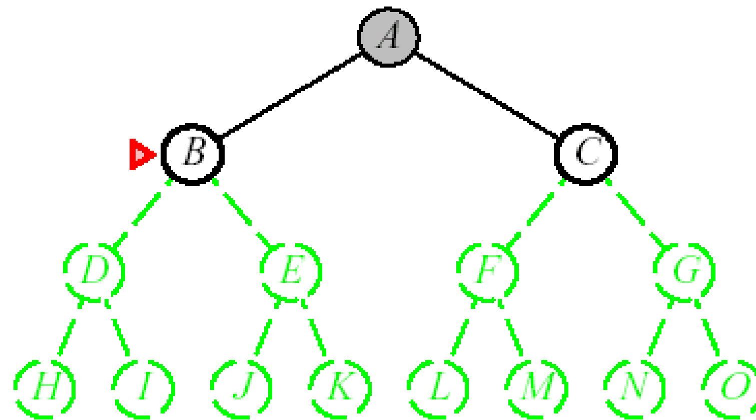
Busca em Profundidade

- A **busca em profundidade** trata a fronteira como uma pilha.
- Ela sempre seleciona um dos últimos elementos adicionados a fronteira.
- Se a fronteira é $[p_1, p_2, . . .]$
  - p_1 é selecionado. Os caminhos que estendem p_1 são adicionados na frente da pilha (antes de p_2).
  - p_2 somente é selecionado quando todos os caminhos partindo de p_1 já foram explorados.

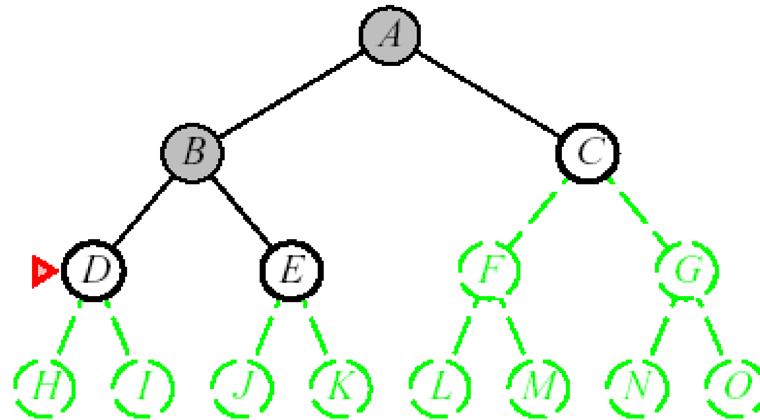
Exemplo – onde M é o nó objetivo



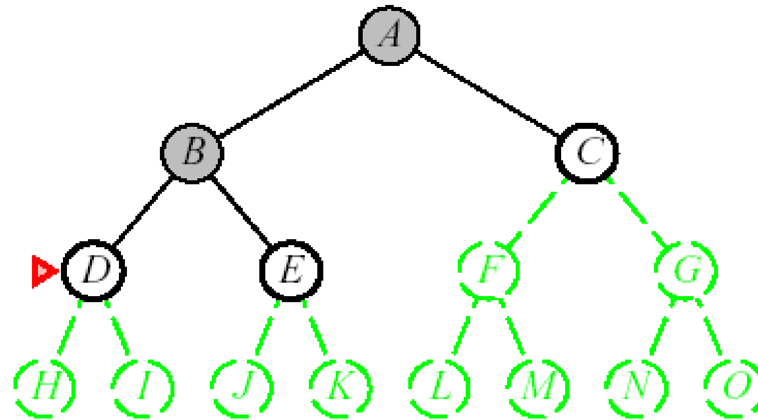
Exemplo – onde M é o nó objetivo



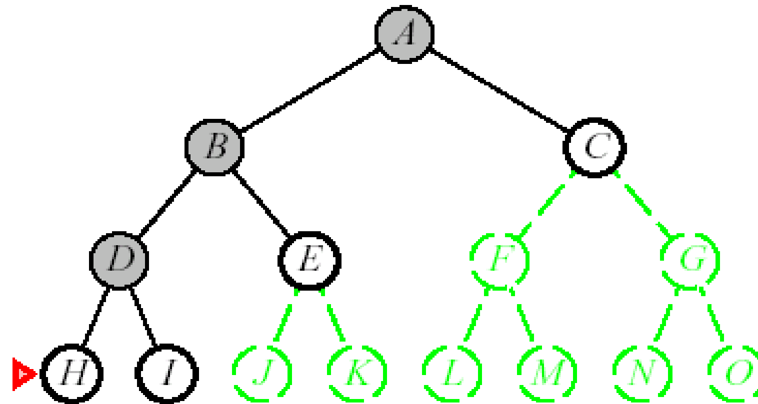
Exemplo – onde M é o nó objetivo



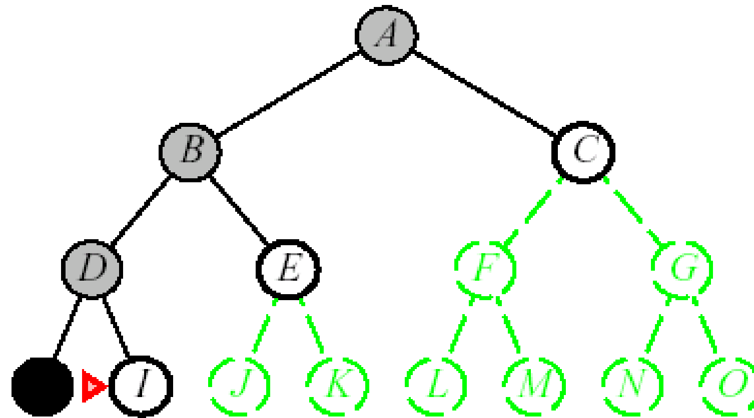
Exemplo – onde M é o nó objetivo



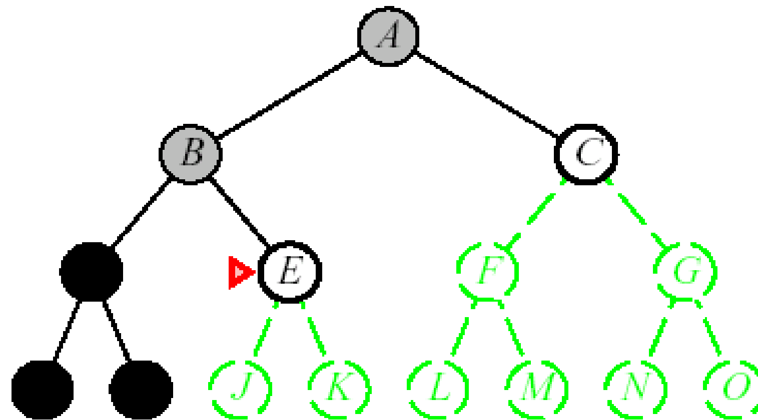
Exemplo – onde M é o nó objetivo



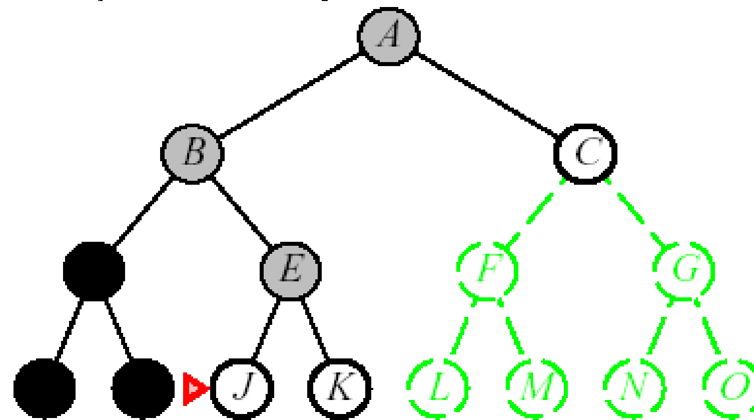
Exemplo – onde M é o nó objetivo



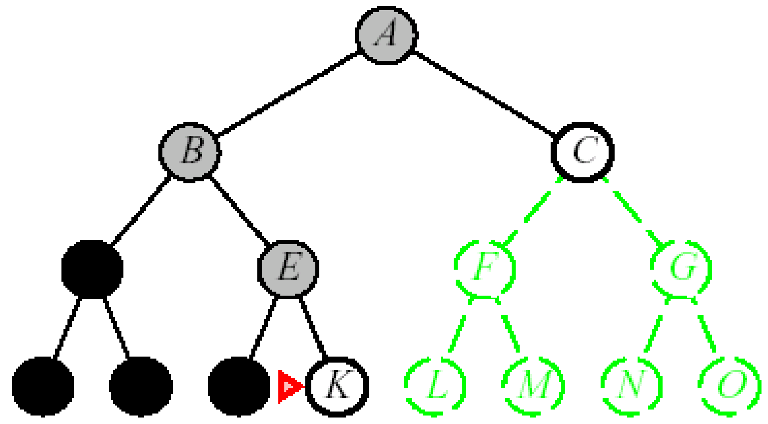
Exemplo – onde M é o nó objetivo



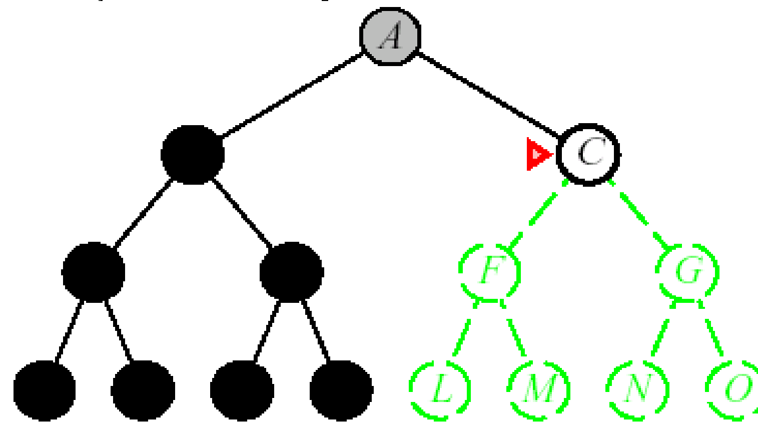
Exemplo – onde M é o nó objetivo



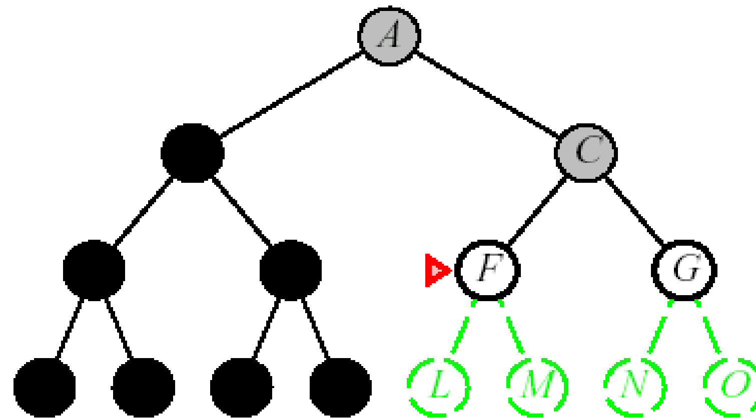
Exemplo – onde M é o nó objetivo



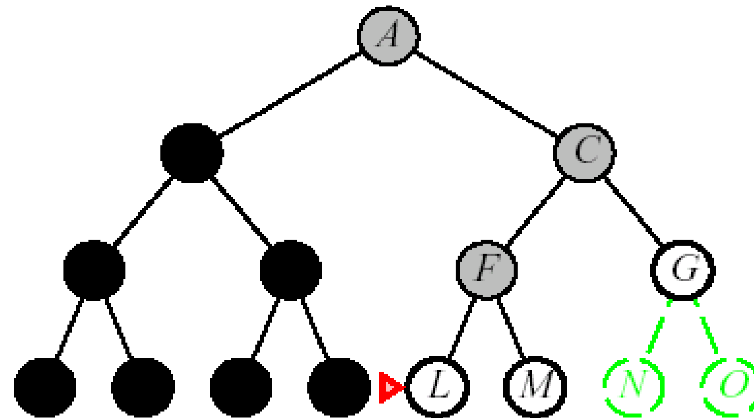
Exemplo – onde M é o nó objetivo



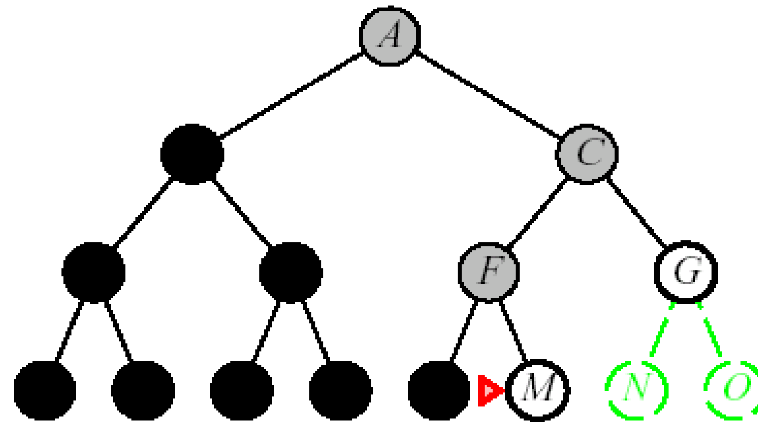
Exemplo – onde M é o nó objetivo



Exemplo – onde M é o nó objetivo



Exemplo – onde M é o nó objetivo



Busca em Profundidade em Prolog

```
% Usa o algoritmo genérico, mas trata a fronteira como uma
% pilha, selecionando o primeiro elemento da pilha

selecione(No, [No|Fronteira], Fronteira).

adicione_a_fronteira(Vizinhos, Fronteira1, Fronteira2) :-
    append(Vizinhos, Fronteira1, Fronteira2).
```

Complexidade da busca em profundidade

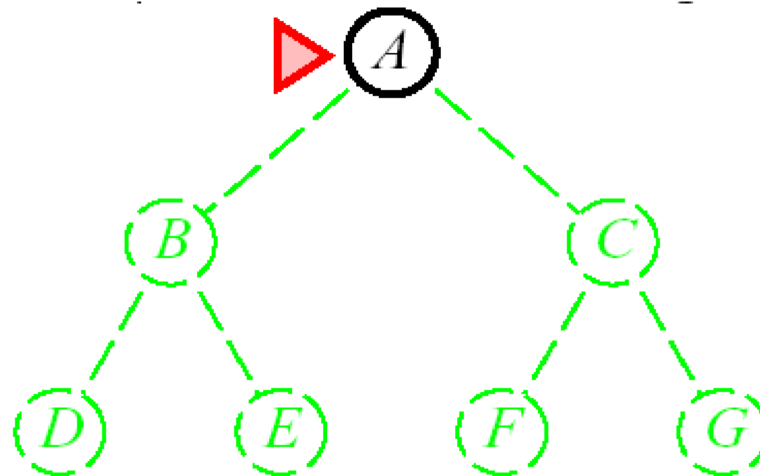
- A busca em profundidade não dá garantias de parada em grafos infinitos ou grafos com ciclos.
- A complexidade de espaço é linear no tamanho do caminho que está sendo explorado
- A busca não é restringida pelo objetivo.

Busca em largura

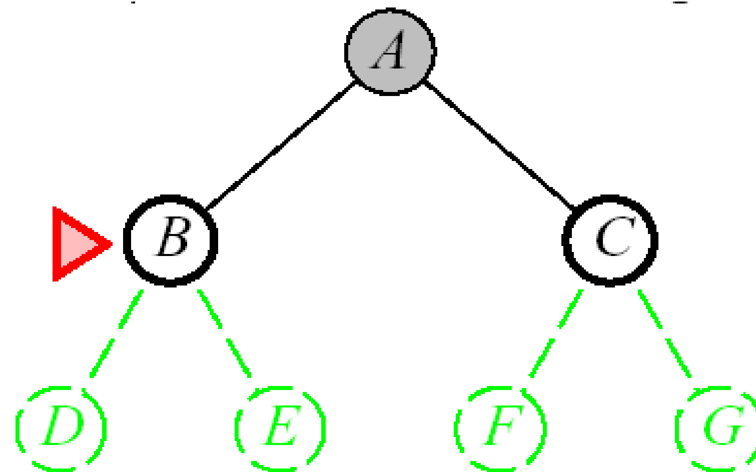
- **Busca em largura** trata a fronteira como um fila.
- Ela sempre seleciona um dos elementos mais antigos adicionados a fronteira.
- Se a fronteira é [p_1, p_2, . . . , p_r]:
  - p_1 é selecionado. Seus vizinhos são adicionados no final da fila, depois de p_r .
  - p_2 é selecionado depois.

```
% Usa o algoritmo genérico, mas trata a fronteira como
% uma fila,selecionando o primeiro elemento da fila

selecione(No, [No|Fronteira], Fronteira).

adicione_a_fronteira(Vizinhos, Fronteira1, Fronteira2) :-
    append(Fronteira1, Vizinhos, Fronteira2).
```

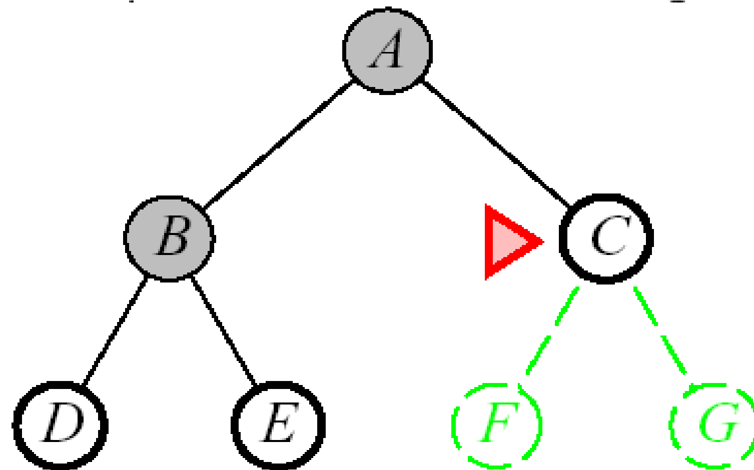
Exemplo – onde D é o nó objetivo



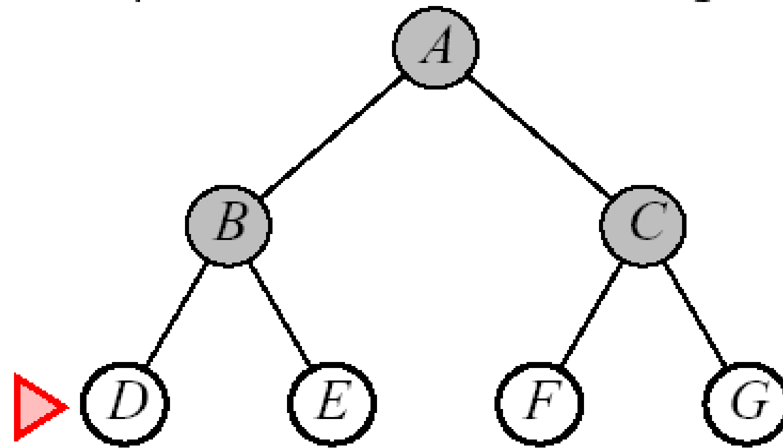
Exemplo – onde D é o nó objetivo



Exemplo – onde D é o nó objetivo



Exemplo – onde D é o nó objetivo



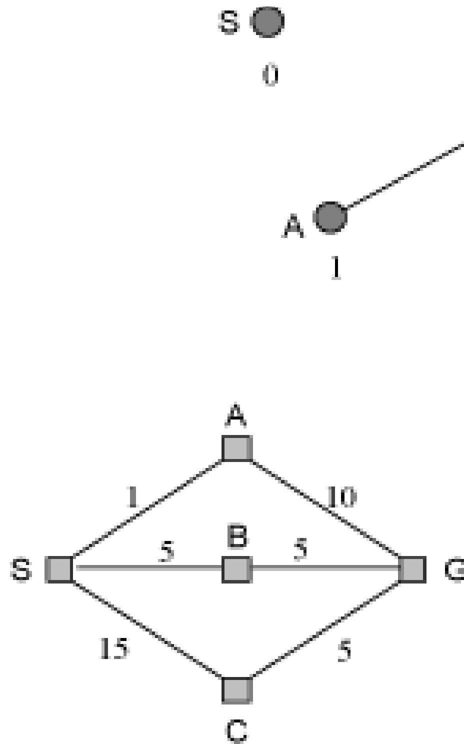
Complexidade da Busca em largura

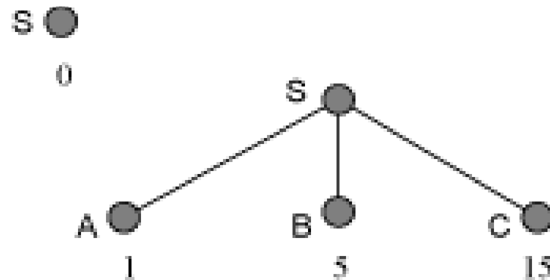
- Se o fator de ramificação de todos os nós é finito, a busca em largura garante achar uma solução se ela existir.
- É garantido achar o caminho com menor número de arcos.
- A complexidade de tempo é exponencial no tamanho do caminho: b^n, onde b é o fator de ramificação, e n é o tamanho do caminho.
- A complexidade de espaço é exponencial no tamanho do caminho: b^n.
- A busca não é restringida pelo objetivo.

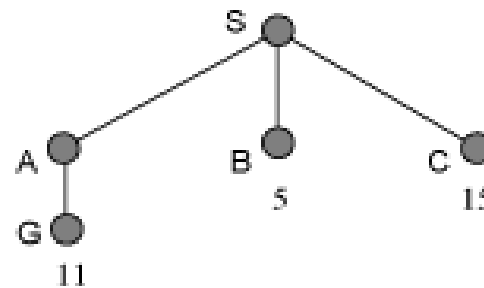
Busca pelo Menor Custo Primeiro

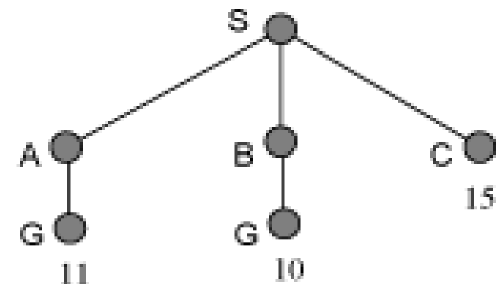
- Algumas vezes existem custos associados com os arcos. O custo de um caminho é a somatória dos custos de seus arcos.
  - *custo(<n0, . . . , nk>)* : $\sum_{i=1}^{k} |\langle n_{i-1}, n_i \rangle|$
- Em cada estágio, a busca pelo menor custo primeiro seleciona o caminho na fronteira com o menor custo.
- A fronteira é uma fila de prioridade ordenada pelo custo do caminho.
- Ela acha o caminho de menor custo até um nó objetivo.
- Quando os custos dos arcos são iguais ➔ busca em largura.

Busca pelo Menor Custo Primeiro









(a) (b)

```
% Usa o algoritmo genérico, mas trata a fronteira como
% uma fila de prioridade ordenada pelo custo,
% selecionando o primeiro elemento da fila

selecione(No, [No|Fronteira], Fronteira).

adicione_a_fronteira(Vizinhos, Fronteira1, Fronteira2) :-
   append(Fronteira1, Vizinhos, Fronteira2),
   ordene_por_g(Fronteira2, Fronteira3).

% É melhor implementar a inserção na fila como uma
  inserção em um heap

busca_menor_custo(Fronteira3).
```

Exercício:

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

Exercício: Resolva o problema dos Missionários e Canibais

- Represente os estados da forma: <M,C,B>
- Função sucessor:
  - Levar 1 missionário e 1 canibal,
  - Levar 2 missionários,
  - Levar 2 canibais,
  - Levar 1 missionário,
  - Levar 1 canibal
- Resolva por busca em largura e por busca em profundidade