

Inteligência Artificial

ACH2016

Aula 3-4: Resolução de problemas por meio de busca

Profa. Karina Valdivia Delgado
EACH-USP

Slides baseados em:

RUSSEL, S.; NORVIG, P. Artificial Intelligence: A modern approach. Third Edition, 2010. Capítulo 3.

Slides do Prof. Edirlei Soares de Lima

Slides da Profa. Leliane Nunes de Barros

Agente que resolve problemas por meio de busca

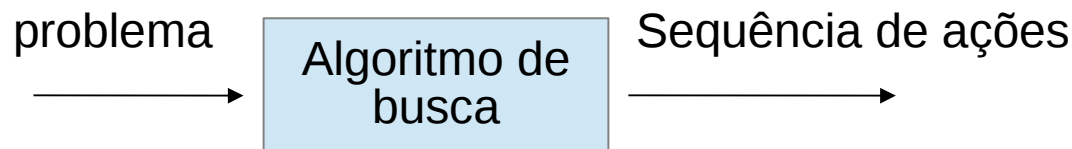
- É um tipo de agente baseado em metas
- Decide o que fazer (possui uma especificação de sua própria meta)
- Conhece os efeitos de suas ações e em que situações elas podem ser executadas
- Usando algoritmos de busca de propósito geral, o agente encontra uma sequência de ações que o leva a um estado do mundo desejado (estado meta)
- Consideram representações atômicas (estados como caixas pretas).

Temas da aula

- Definição de problemas.
- Definição de suas soluções.
- Algoritmos de busca:
 - **Busca sem informação:** algoritmos para os quais não se fornece nenhuma informação sobre o problema a não ser sua definição.
 - **Busca informada:** podem ter sucesso a partir de alguma orientação sobre onde procurar soluções.

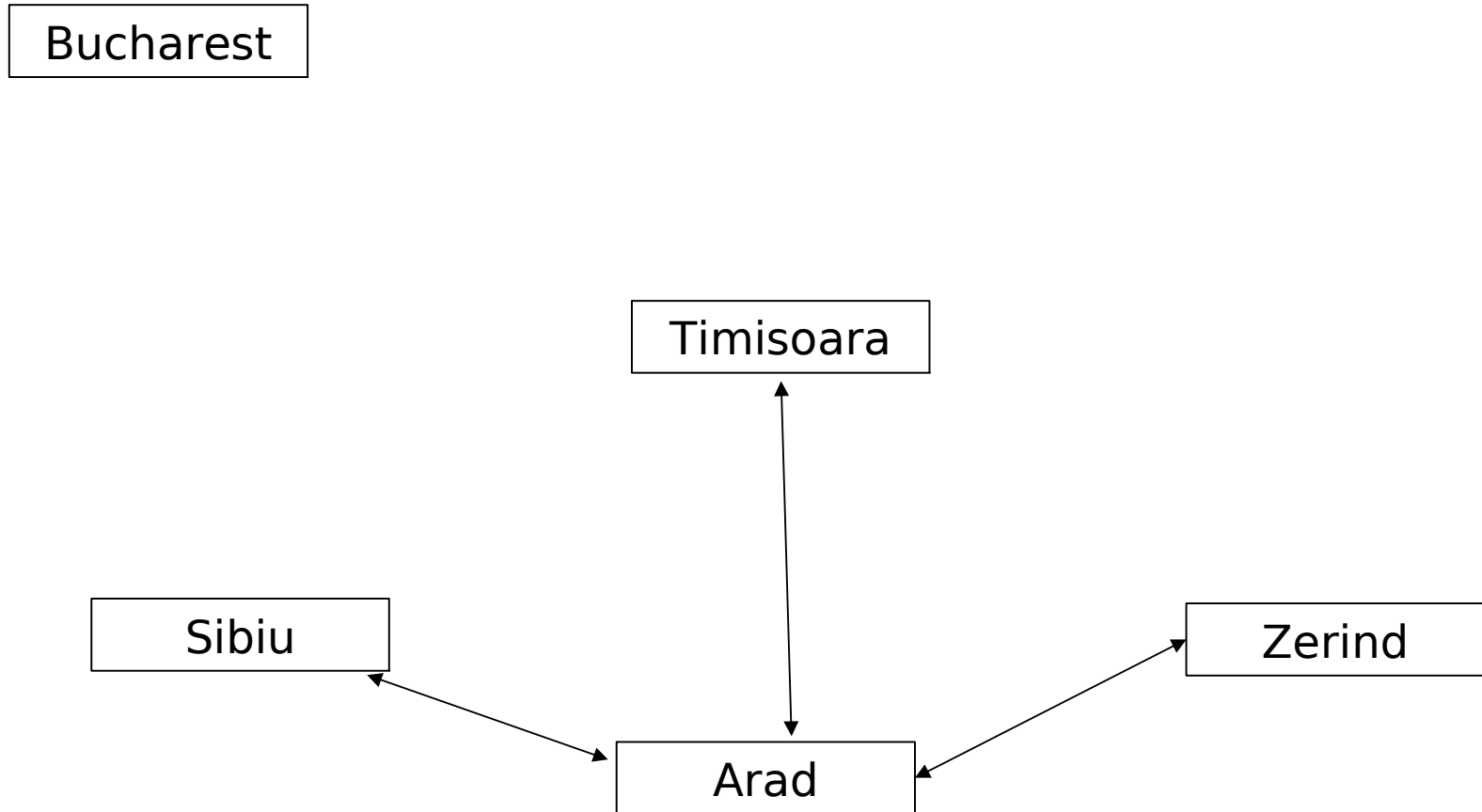
Agente que resolve problemas por meio de busca

- **Formular:** formular o **objetivo** e o **problema** (escolha de um conjunto relevante de estados e ações dado o objetivo).
- **Buscar:** é a tarefa de procurar a sequência de ações que leva do estado atual até um estado objetivo.

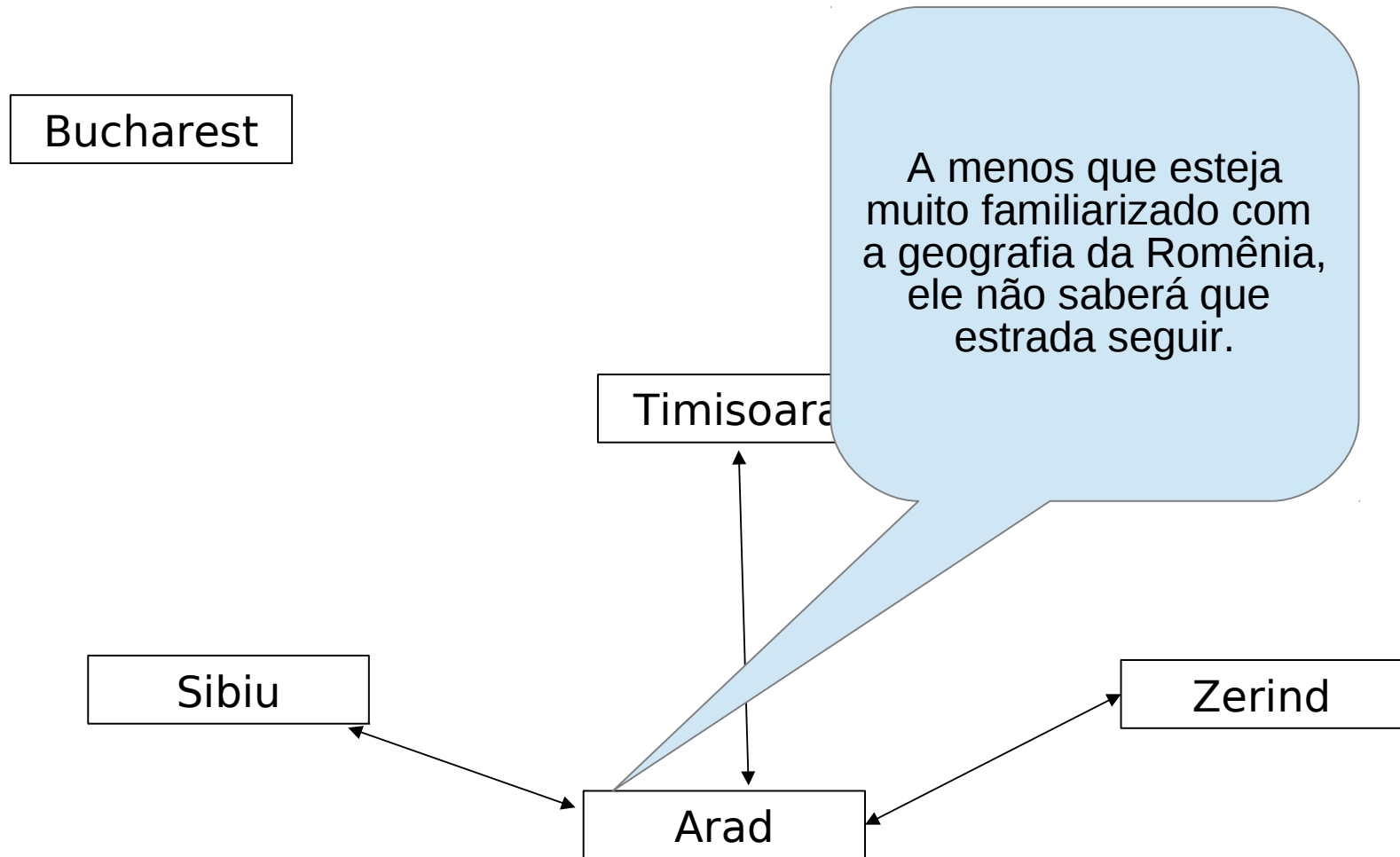


- **Executar:** após que uma solução é encontrada, as ações recomendadas podem ser executadas:
 - O agente pode executar seus planos sem necessidade de percepção.

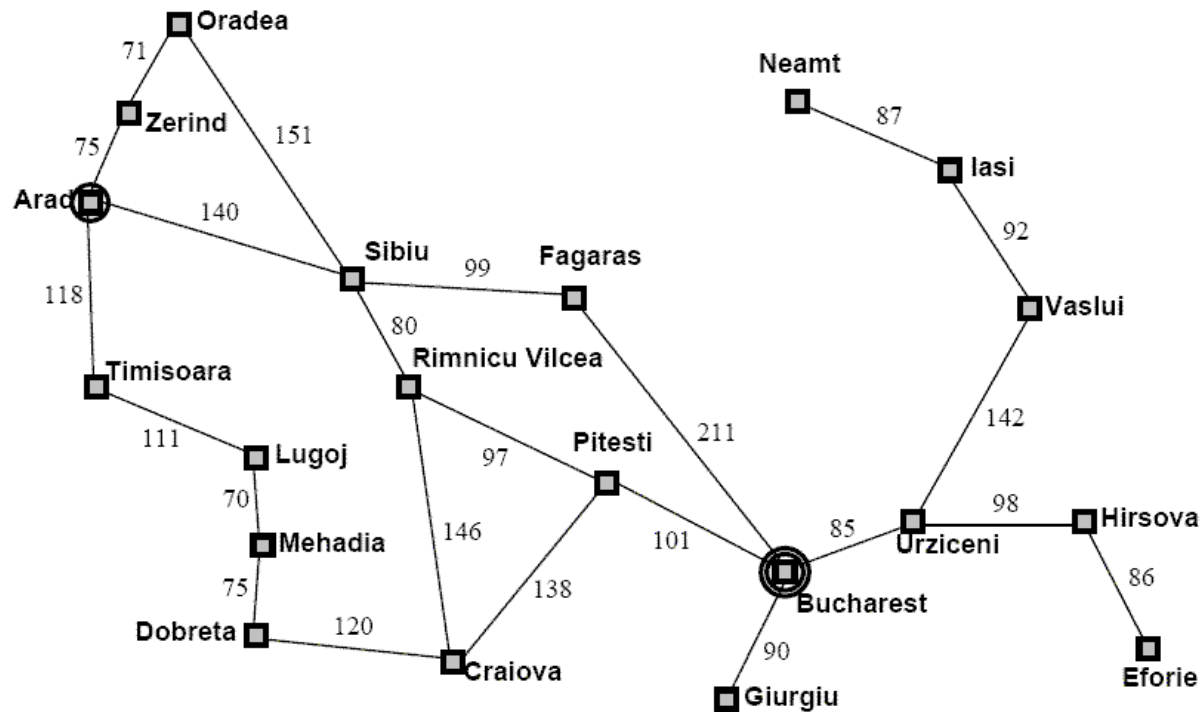
Agente que resolve problemas por meio de busca



Agente que resolve problemas por meio de busca



Agente que resolve problemas por meio de busca



Agente que resolve problemas por meio de busca

Suposições iniciais em relação ao ambiente:

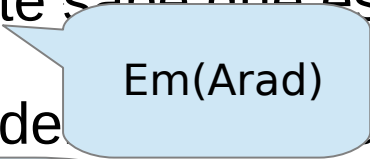
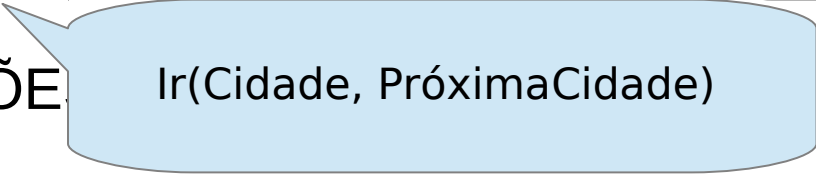
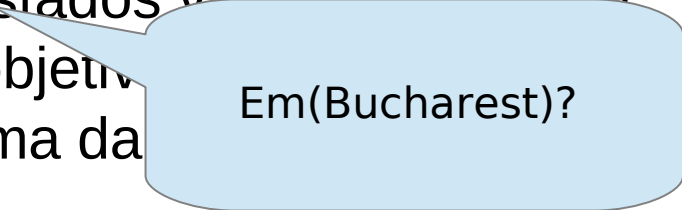
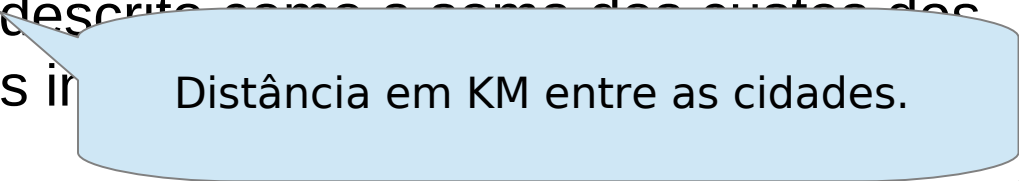
- **Estático:** O Ambiente não pode mudar enquanto o agente está realizando a resolução do problema.
- **Discreto:** o ambiente pode ser representado por um conjunto finito de estados.
- **Observável:** O estado inicial do ambiente precisa ser conhecido previamente.
- **Determinístico:** O próximo estado do agente deve ser determinado pelo estado atual + ação. A execução da ação não pode falhar.

Sobre essas premissas, a **solução** para qualquer problema é uma **sequência de ações** que podem ser executadas sem sensoramento.

Formulação do Problema

- **Estado inicial:** que o próprio agente sabe que está.
- **Ações:** conjunto de ações que podem ser executadas
 - $AÇÕES(s)$
- **Modelo de transição:** uma descrição do que cada ação faz.
 - $RESULTADO(s,a)$
- **Teste de objetivo:** compara estados visitados com um conjunto explícito de estados objetivo ou verifica se os estados visitados satisfazem uma dada propriedade (ex.: cheque-mate do xadrez)
- **Custo do caminho:** que atribui um custo numérico a cada caminho. Pode ser descrito como a soma dos custos dos passos (custo ações individuais em um caminho, $c(s,a,s')$).

Formulação do Problema

- **Estado inicial:** que o próprio agente sabe que está.
- **Ações:** conjunto de ações que pode ser tomadas
 - AÇÃO: Ir(Cidade, PróximaCidade)
- **Modelo de transição:** uma descrição do que cada ação faz.
 - RESULTADO(s,a)
- **Teste de objetivo:** compara estados visitados com um conjunto explícito de estados objetivos. Os estados visitados satisfazem uma dada condição.
- **Custo do caminho:** que atribui um custo numérico a cada caminho. Pode ser descrito como a soma dos custos dos passos (custo ações individuais).

Problema: definições importantes

- **Espaço de estados:** o estado inicial, as ações e o modelo de transição definem o conjunto de estados que podem ser atingidos a partir do estado inicial.
- **Solução:** caminho desde o estado inicial até um estado objetivo.
- **Solução ótima:** tem o menor custo de caminho entre todas as soluções.

Problema: definições importantes

- **Espaço de estados:** o estado inicial, as ações e o modelo de transição definem o conjunto de estados que podem ser atingidos a partir do estado inicial.
- **Solução:** caminho de estados desde o estado inicial até um estado objetivo.
- **Solução ótima:** tem o menor custo de caminho entre todas as soluções.



Mapa da Romênia

Exemplo: Aspirador de Pó

localizações discretas

sujeira discreta

ações determinísticas (limpeza perfeita)

ambiente estático (cômodos permanecem limpos)

Totalmente observável

Exemplo: Aspirador de Pó

- **Estados:**
- **Estado inicial:**
- **Ações:**
- **Modelo de transição:**
- **Teste de objetivo:**
- **Custo do caminho:**

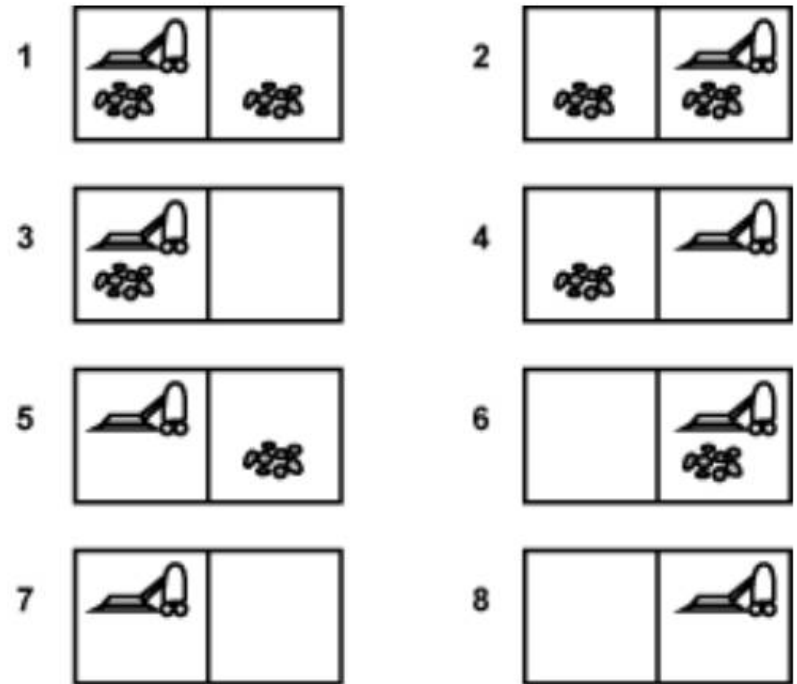
Exemplo: Aspirador de Pó

- **Estados:** 8 estados possíveis.

Um ambiente com n posições tem:

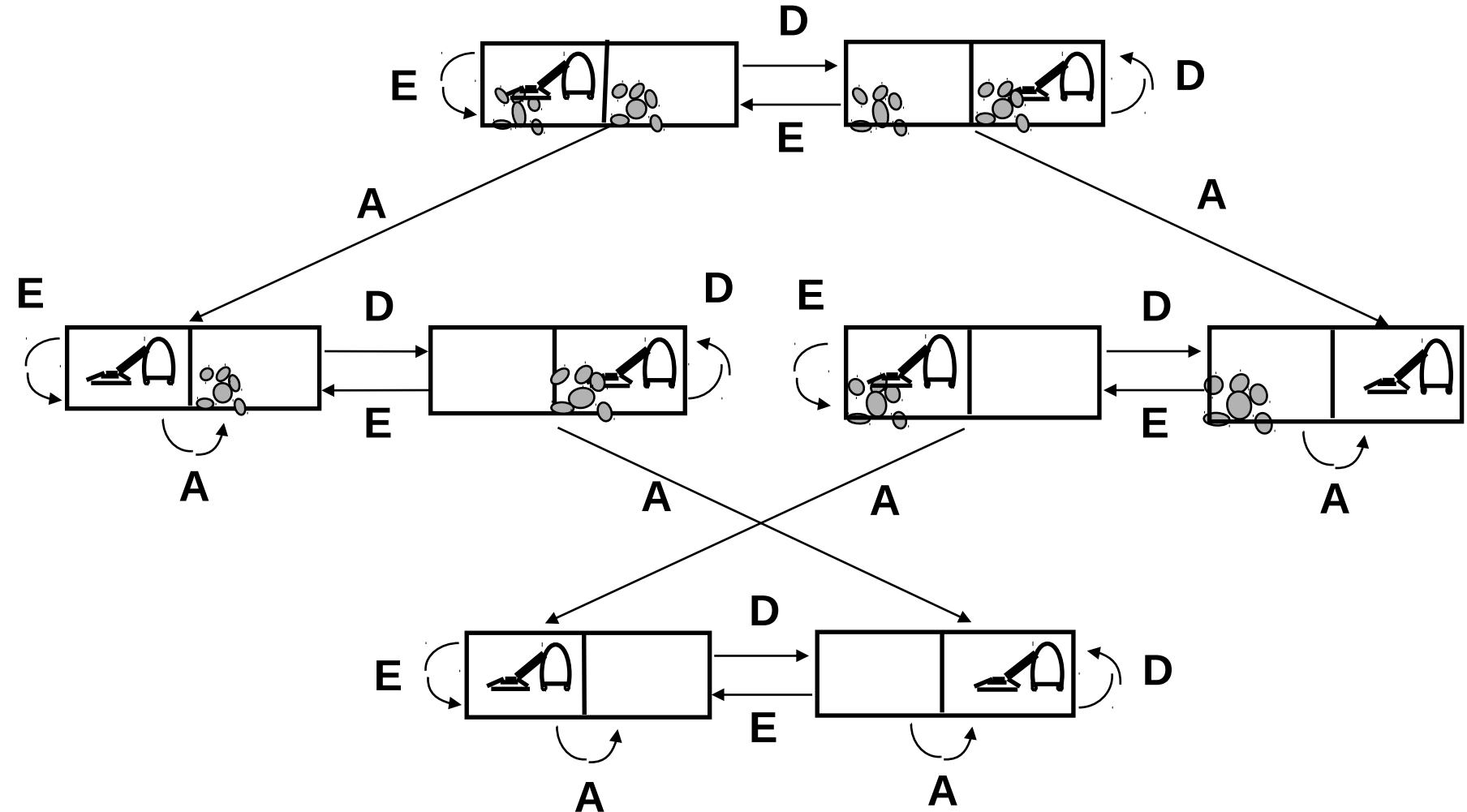
– $n2^n$ estados

- **Estado inicial:** Qualquer estado;
- **Ações:** Mover para direita, mover para esquerda e aspirar
- **Modelo de transição:** As ações têm seus efeitos esperados.
- **Teste de objetivo:** verifica se todas as salas estão limpas.
- **Custo do caminho:** Cada passo tem o custo 1, assim o custo do caminho é definido pelo número de passos



Exemplo: Aspirador de Pó

- Modelo de transição:



Exemplo: 8-Puzzle

- **Estados:** o estado é especificado pela posição de cada uma das oito peças e do quadrado vazio. 181.440 possíveis estados;
- **Estado inicial:** Qualquer estado;
- **Ações:** Mover o quadrado vazio para direita, para esquerda, para cima ou para baixo;
- **Modelo de transição:** dado um estado e uma ação devolve o estado resultante;
- **Teste de objetivo:** verifica se o estado corresponde ao estado objetivo.
- **Custo do caminho:** Cada passo tem o custo 1, assim o custo do caminho é definido pelo número de passos;

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

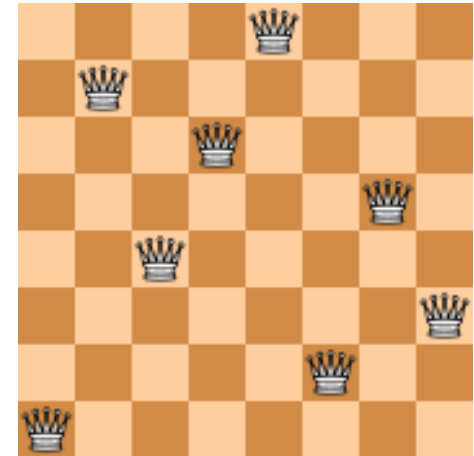
15-puzzle (4x4) – 1.3 trilhões estados possíveis.

24-puzzle (5x5) – 10^{25} estados possíveis.

Exemplo: 8 Rainhas (Incremental)

Cada ação acrescenta uma rainha ao estado

- **Estados:** Qualquer disposição de 0 a 8 rainhas no tabuleiro ($64 \times 63 \times \dots \times 57 = 1.8 \times 10^{14}$ possíveis estados);
- **Estado Inicial:** Nenhuma rainha no tabuleiro;
- **Ações Possíveis:** Colocar uma rainha em qualquer quadrado vazio;
- **Modelo de transição:** devolver uma rainha adicionada em um quadrado específico.
- **Teste de objetivo:** Qualquer estado onde as 8 rainhas estão no tabuleiro e nenhuma esta sendo atacada;
- **Custo:** Não importa nesse caso;

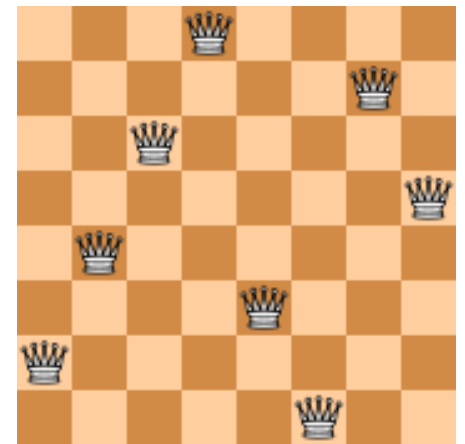


* O jogo possui apenas 92 possíveis soluções (considerando diferentes rotações e reflexões). E apenas 12 soluções únicas.

Exemplo: 8 Rainhas

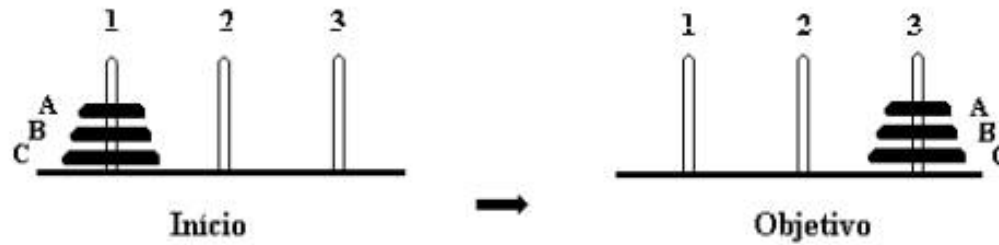
Uma melhor formulação proibiria a colocação de uma rainha em um quadrado que já estiver sob ataque.

- **Estados:** Tabuleiro com n rainhas, uma por coluna, nas n colunas mais à esquerda sem que nenhuma rainha ataque outra (2057 possíveis estados);
- **Ações:** Adicionar uma rainha em qualquer quadrado na coluna vazia mais à esquerda, de tal modo que ela não seja atacada por qualquer outra rainha;



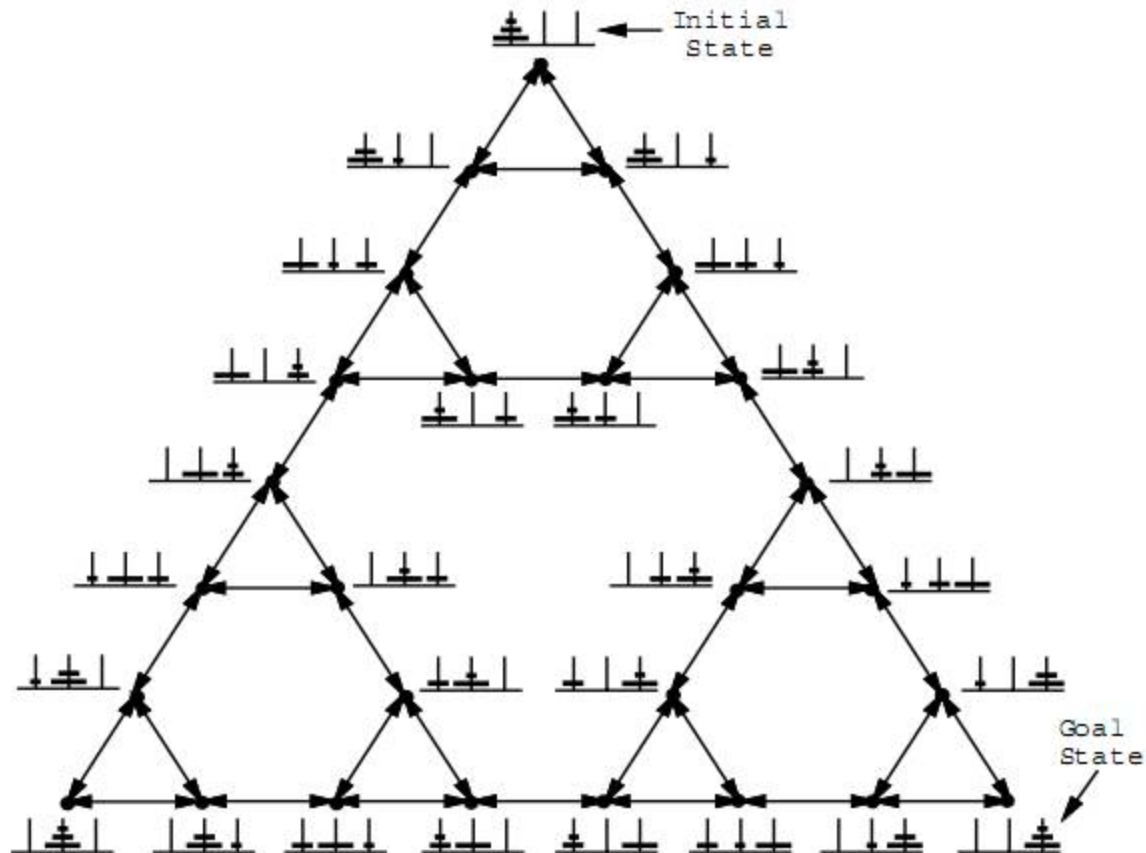
Exercício

- Torre de Hanói



Exercício

- Torre de Hanói: espaço de busca



Aplicações em Problemas Reais

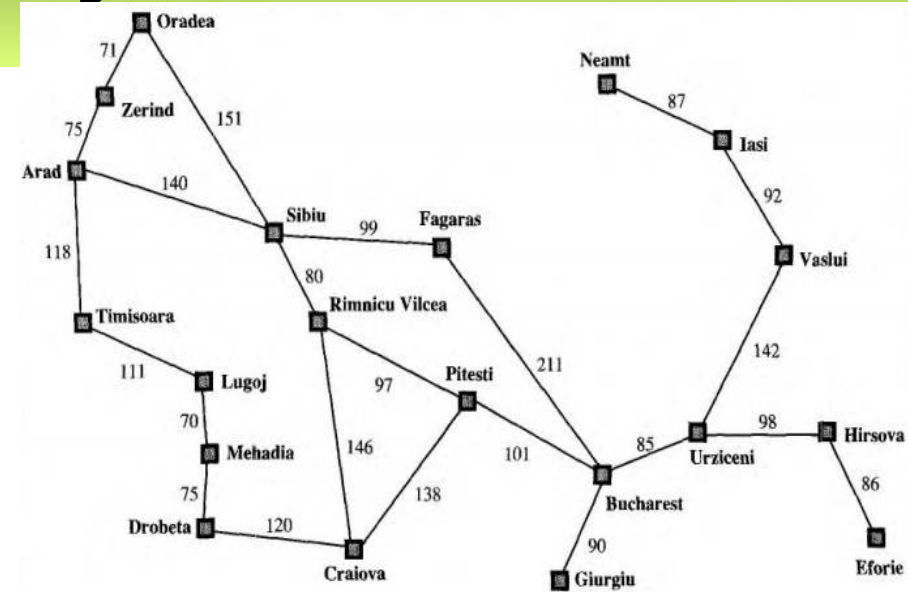
- Planejamento de roteamento:
 - Sistemas de planejamento de viagens;
 - Caixeiro viajante;
 - Rotas em redes de computadores;
- Navegação de robôs;
- Sequência automática de montagem.

Em busca de soluções

- Depois de formular o problema, o estado objetivo deve ser “**buscado**” no espaço de estados.
- A busca no espaço de estados a partir do estado inicial forma uma **árvore de busca** em que a raiz: corresponde ao estado inicial, os arcos são as ações e os nós correspondes aos nós no espaço de estados.
- Processo:
 - Expande-se o estado corrente (no início o estado inicial), gerando um novo conjunto de sucessores;
 - Escolhe-se o próximo estado a expandir seguindo uma **estratégia de busca**;
 - Prossegue-se até chegar ao estado final (solução) ou falhar na busca pela solução;

Em busca de soluções

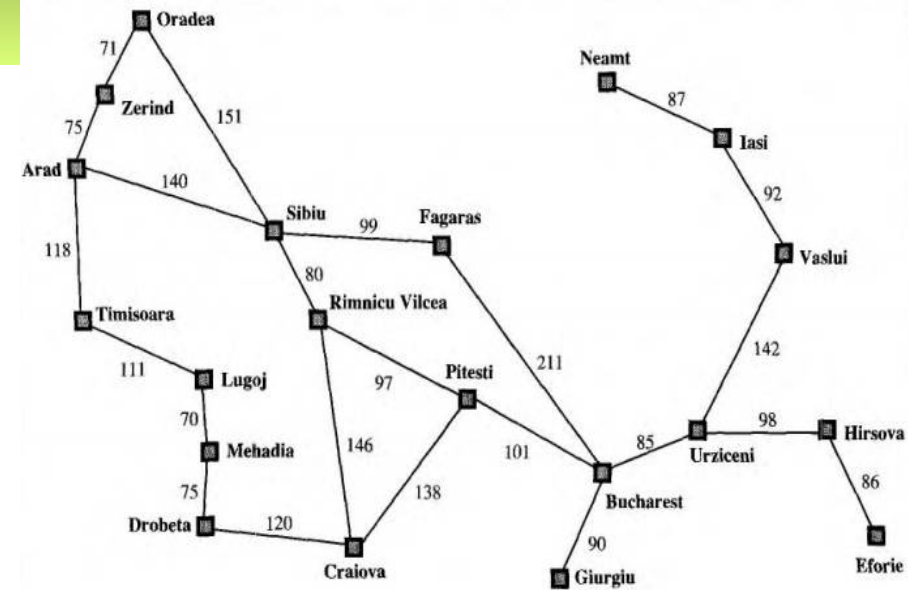
Exemplo: Ir de **Arad** para **Bucharest**



Arad

Em busca de soluções

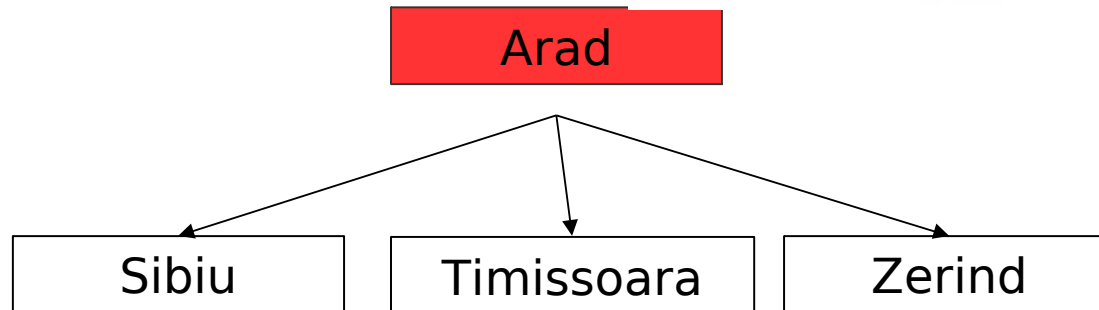
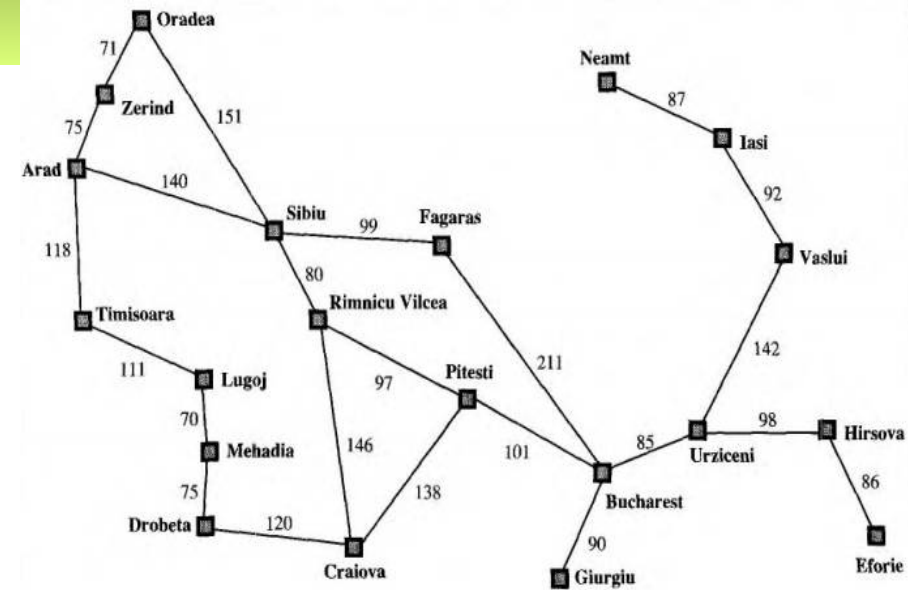
Exemplo: Ir de **Arad** para **Bucharest**



Arad

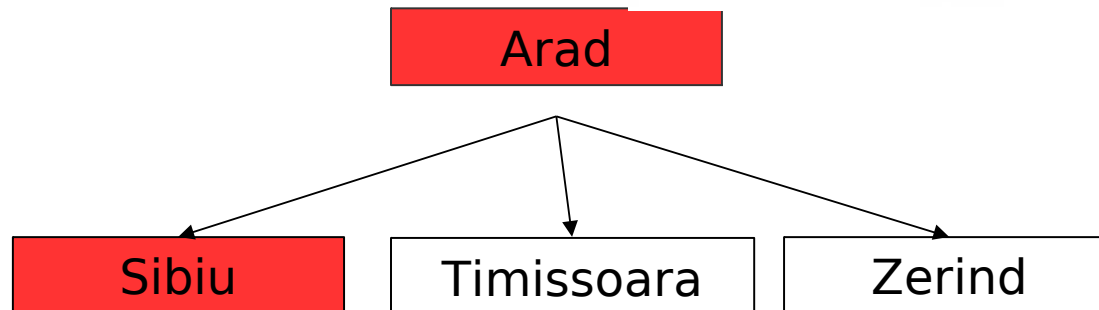
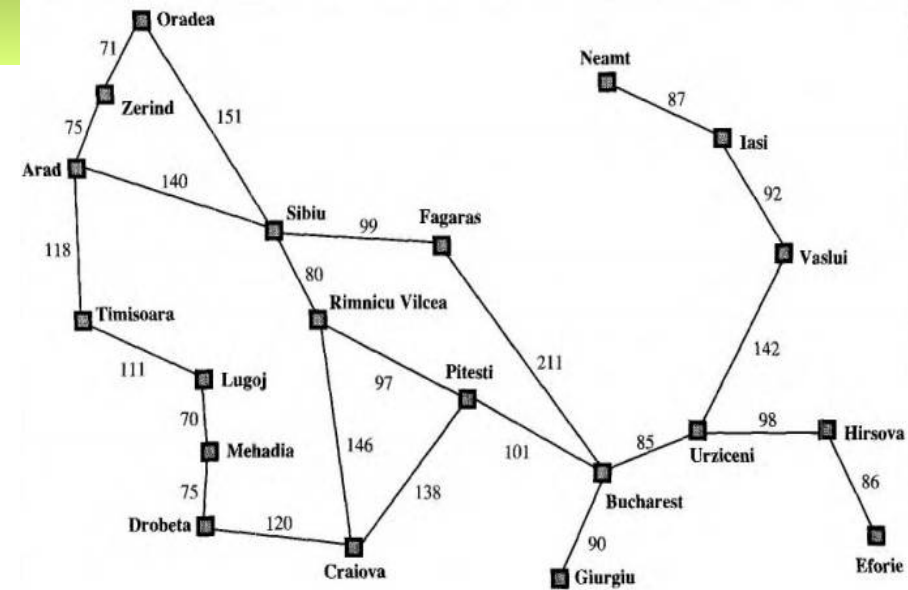
Em busca de soluções

Exemplo: Ir de **Arad** para **Bucharest**



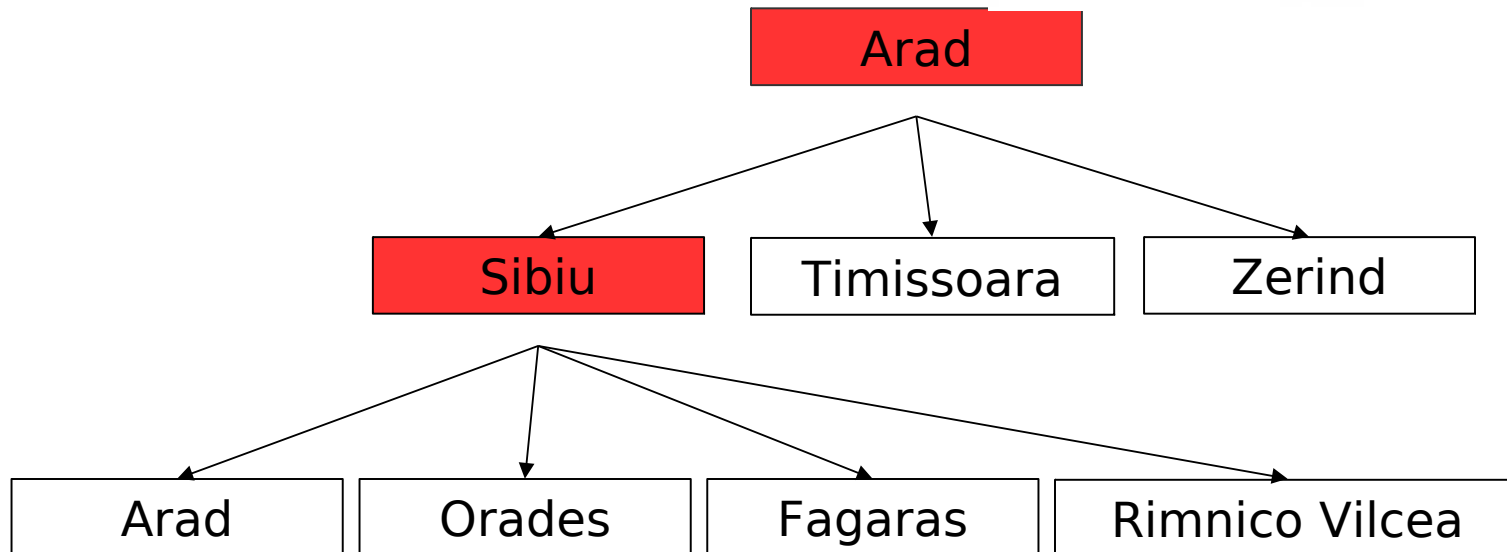
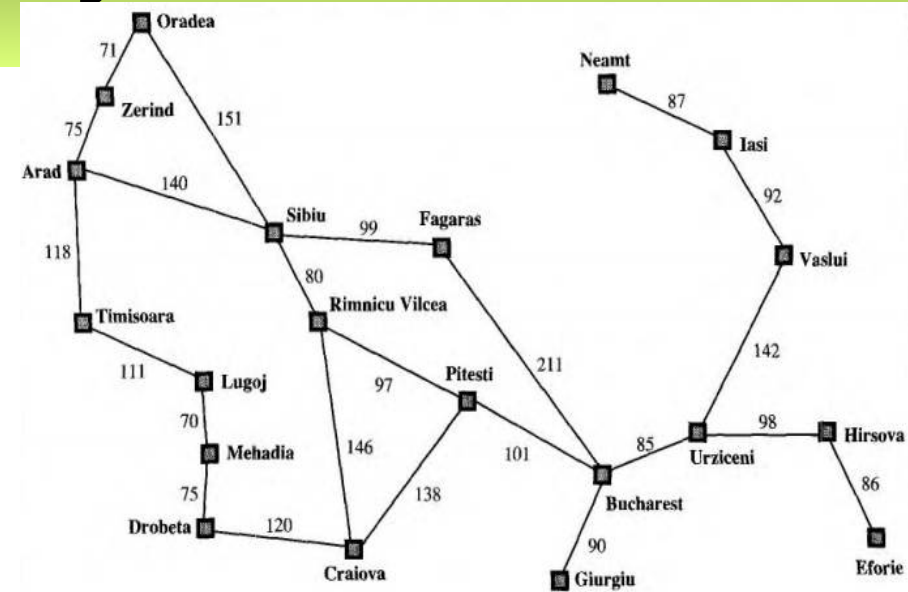
Em busca de soluções

Exemplo: Ir de **Arad** para **Bucharest**



Em busca de soluções

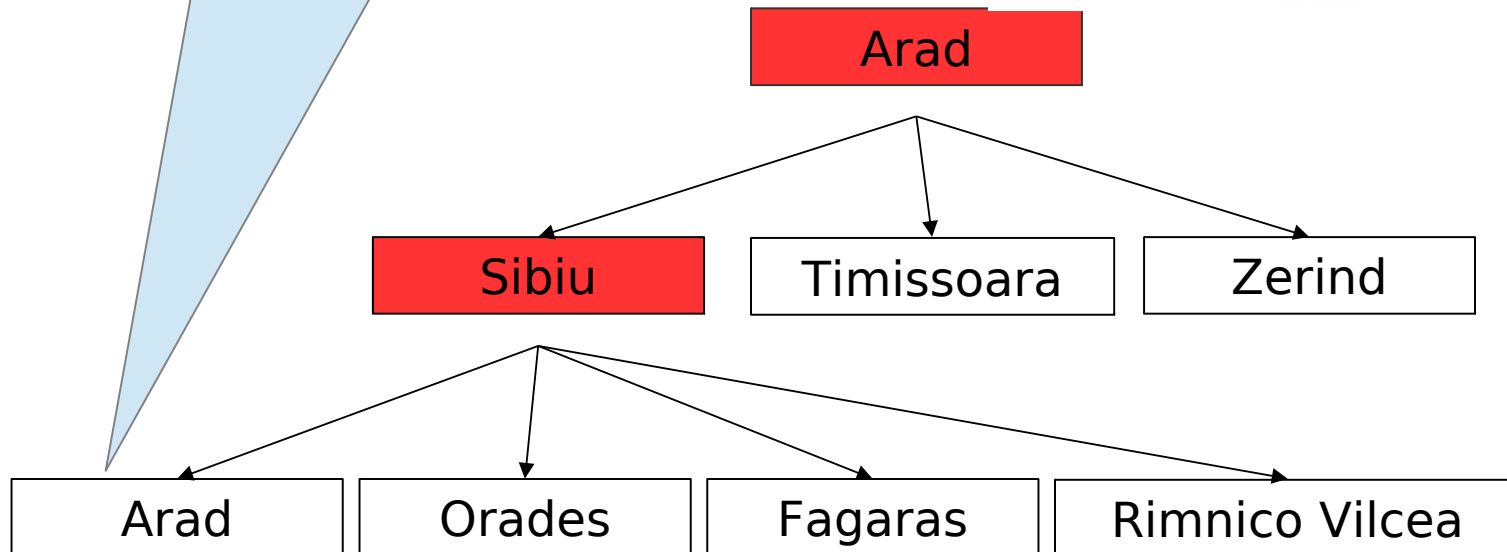
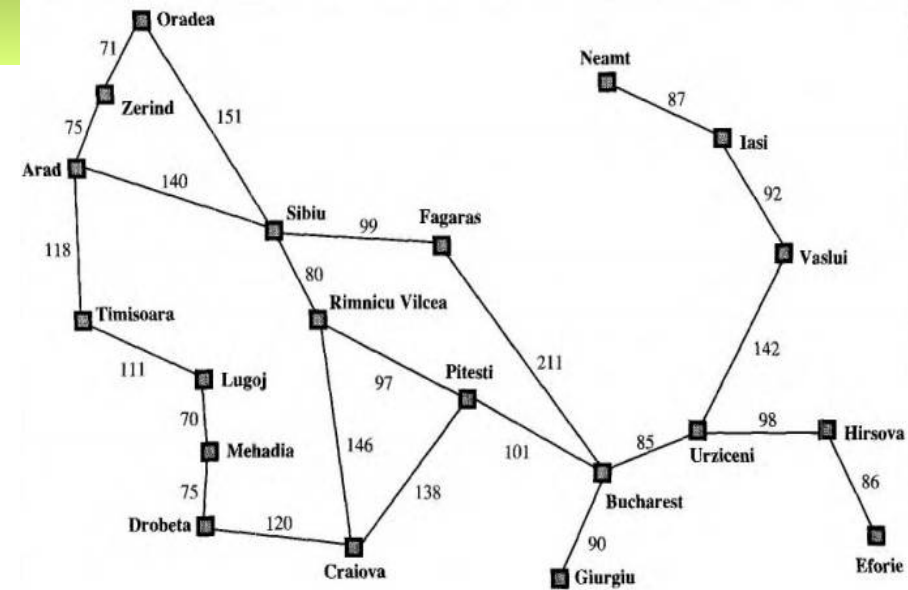
Exemplo: Ir de **Arad** para **Bucharest**



Em busca de soluções

Exemplo: Ir de **Arad** para **Bucharest**

Estado repetido na árvore de busca => que a árvore de busca completa para o problema é infinita

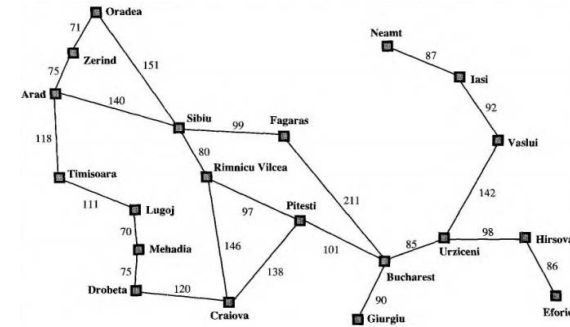


Em busca de soluções

- O espaço de estados é **diferente** da árvore de busca. Ex:

- 20 estados no espaço de estados;
- Número de caminhos infinito;
- Árvore com infinitos nós se consideramos nós repetidos;

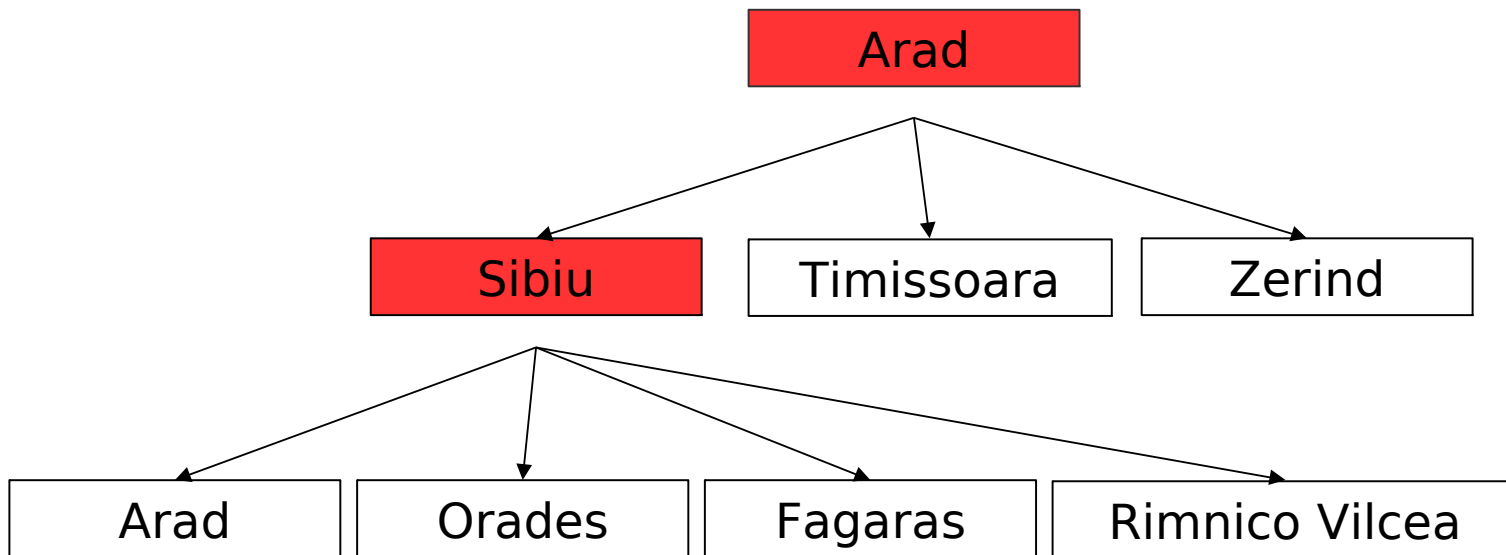
- Felizmente, não é necessário considerar caminhos com laço.



custos de caminho aditivos e custos de passo não negativos => um caminho em laço para qualquer estado nunca será melhor do que o mesmo caminho com o laço removido.

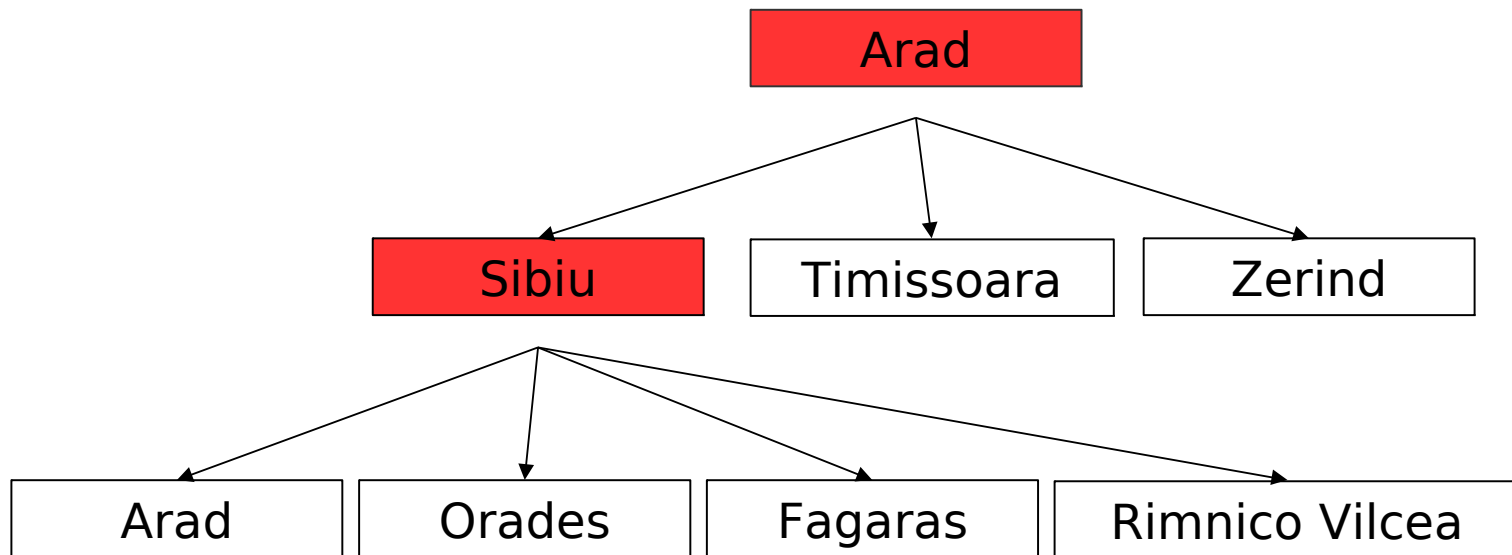
Em busca de soluções

- O conjunto de todos os nós folhas disponíveis para expansão é chamado de **borda**



Em busca de soluções

O conjunto de todos os já expandidos é chamado de **conjunto explorado**



Código Descritivo – Busca

Função Busca(*problema*) **retorna** solução ou falha

inicializar a borda usando o estado inicial do *problema*

inicializar o conjunto explorado como vazio

repita

se borda vazia **então retorna** falha

 escolher um nó folha e remover da borda

se o estado do nó for um estado objetivo **então retorna** solução correspondente

adicionar o nó ao conjunto explorado

 expandir o nó escolhido, adicionando os nós resultantes a borda

apenas se não estiver na borda ou no conjunto explorado

Código Descritivo – Busca

Função Busca(*problema*) **retorna** solução ou falha

inicializar a borda usando *problema* **Para tratar estados repetidos**

inicializar o conjunto explorado como vazio

repita

se borda vazia **então retorna** falha

escolher um nó folha e remover da borda

se o estado do nó for um estado objetivo **então retorna** solução correspondente

adicionar o nó ao conjunto explorado

expandir o nó escolhido, adicionando os nós resultantes a borda

apenas se não estiver na borda ou no conjunto explorado

Código Descritivo – Busca

Função Busca(*problema*) **retorna** solução ou falha

inicializar a borda usando o estado inicial do *problema*

inicializar o conjunto explorado como vazio

repita

a decisão de quem escolher primeiro
define uma **estratégia de busca** diferente

se borda vazia **ent**

escolher um nó folha e remover da borda

se o estado do nó for um estado objetivo **então retorna** solução correspondente

adicionar o nó ao conjunto explorado

expandir o nó escolhido, adicionando os nós resultantes a borda

apenas se não estiver na borda ou no conjunto explorado

Código Descritivo – Busca

Função Busca(*problema*) **retorna** solução ou falha

inicializar a borda usando o estado inicial do *problema*

inicializar o conjunto explorado como vazio

repita

se borda vazia **então retorna** falha

escolher um nó folha e remover da borda

se o estado **Nós anteriormente gerados (os que estão no conjunto explorado ou na borda) podem ser descartados** **então**

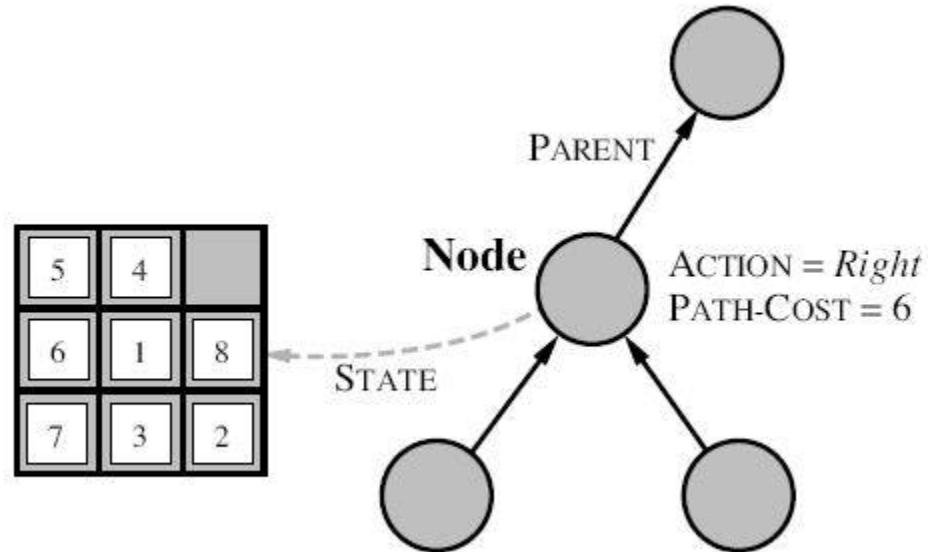
adicionar o nó ao conjunto explorado

expandir o nó escolhido, adicionando os nós resultantes a borda

apenas se não estiver na borda ou no conjunto explorado

Estrutura de dados para árvores de busca

- Um **nó** é formado por:
n.ESTADO
n.PAI
n.AÇÃO: a ação que foi aplicada ao pai para gerar o nó
n.CUSTO-DO-CAMINHO: custo, denotado por $g(n)$, do caminho do estado inicial até o nó.



- **Borda**: pode ser implementado como uma fila.

Três variantes de fila:

- fila FIFO
- fila LIFO (pilha)
- fila de prioridade

- **Conjunto explorado**: pode ser implementado como uma tabela hash.

Medida de Desempenho

Completude:

a estratégia garante encontrar uma solução, se existir uma?

Qualidade da solução:

a estratégia encontra a solução ótima?

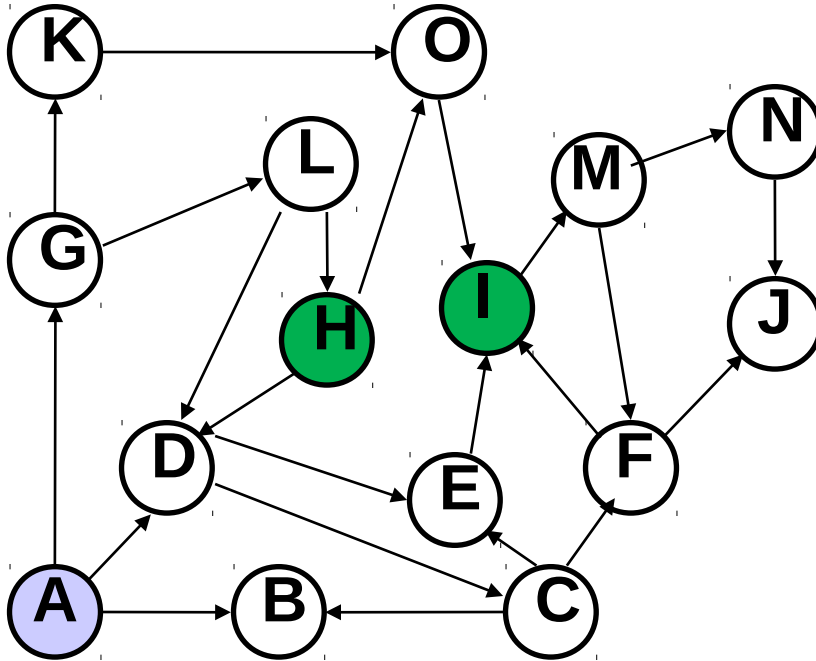
Complexidade de tempo:

quanto tempo gasta para encontrar uma solução?

Complexidade de memória:

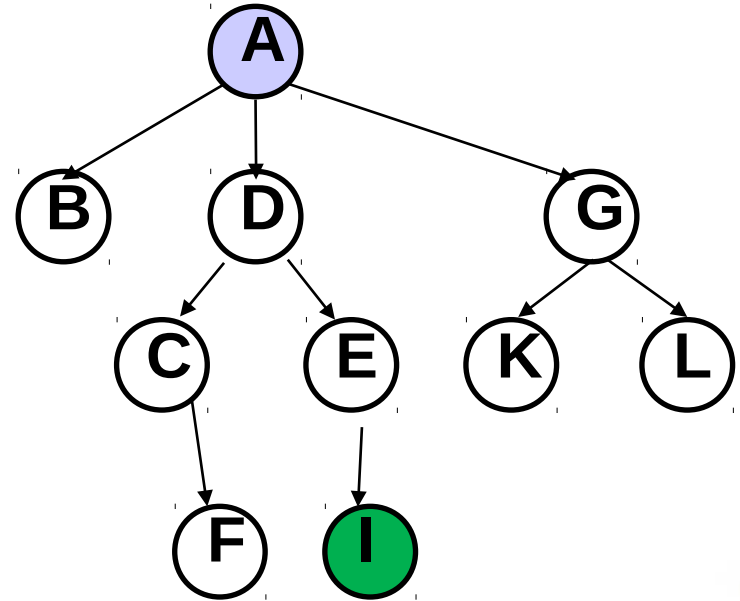
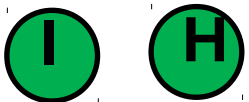
quanto de memória é preciso?

Medida de Desempenho



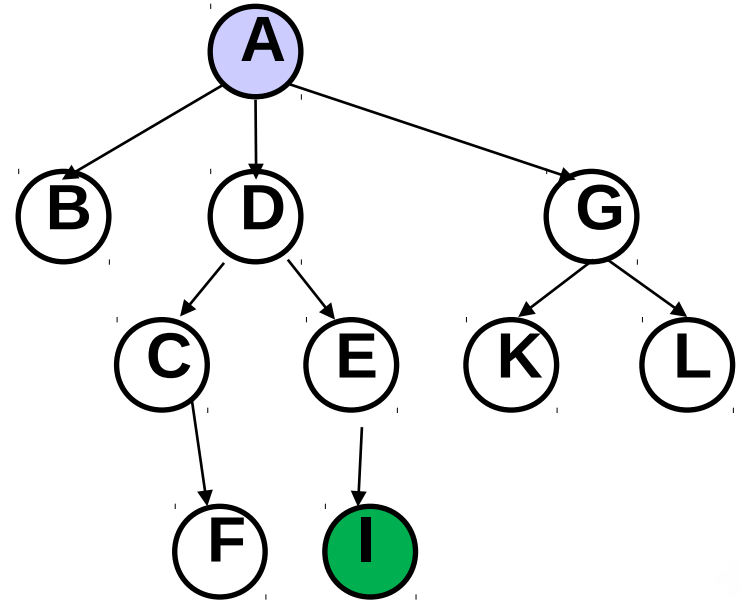
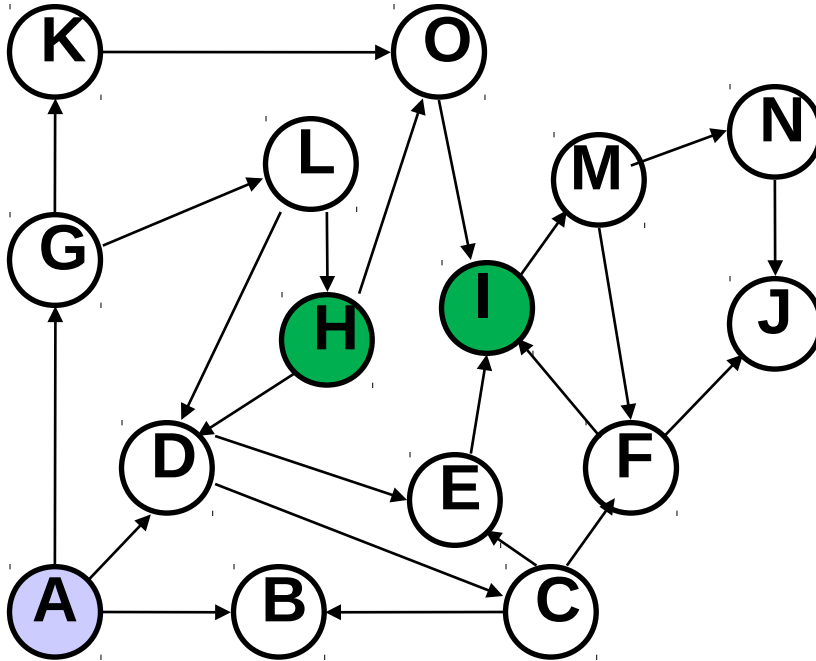
Espaço de busca

A é o estado inicial



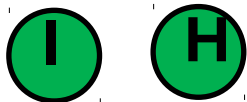
Árvore de busca
gerada sem nós
repetidos

Medida de Desempenho



Espaço de busca

A é o estado inicial



O algoritmo é completo? é ótimo?
Quanto tempo gasta? Quanta
memória gasta?

busca
em nós

Medida de Desempenho

- Complexidade de tempo e espaço
 - Em IA, o grafo do espaço de estados é sempre representado **implicitamente** pelo estado inicial, as ações e o modelo de transição.
 - A complexidade é medida em termos de:
 - Fator de ramificação (número máximo de sucessores de todos os nós): **b**
 - Profundidade do nó objetivo menos profundo: **d**
 - Comprimento máximo de qualquer caminho no espaço de estados: **m**

Métodos de busca

- **Busca sem informação ou cega ou exaustiva ou força bruta ou sistemática :**
 - Não sabe qual o melhor nó da fronteira a ser expandido. Apenas distingue o estado objetivo dos não objetivos.
- **Busca informada ou heurística:**
 - Estima qual o melhor nó da fronteira a ser expandido com base em funções heurísticas. Sabem se um estado não objetivo é “mais promissor”.
- **Busca local:**
 - Operam em um único estado e movem-se para a vizinhança deste estado.

Métodos de busca

- **Algoritmos sem informação:**

- Busca em largura;
- Busca de custo uniforme;
- Busca em profundidade;
- Busca em profundidade iterativa;
- Busca bidirecional

Busca em Largura

- **Estratégia:**

- O nó raiz é expandido, em seguida todos os nós sucessores são expandidos, então todos próximos nós sucessores são expandidos, e assim em diante.

A

Busca em Largura

- **Estratégia:**

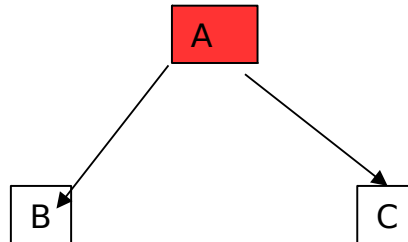
- O nó raiz é expandido, em seguida todos os nós sucessores são expandidos, então todos próximos nós sucessores são expandidos, e assim em diante.

A

Busca em Largura

- **Estratégia:**

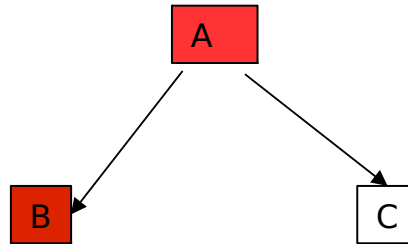
- O nó raiz é expandido, em seguida todos os nós sucessores são expandidos, então todos próximos nós sucessores são expandidos, e assim em diante.



Busca em Largura

- **Estratégia:**

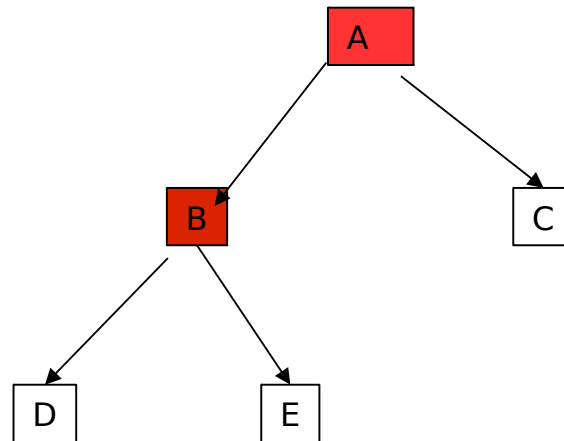
- O nó raiz é expandido, em seguida todos os nós sucessores são expandidos, então todos próximos nós sucessores são expandidos, e assim em diante.



Busca em Largura

- **Estratégia:**

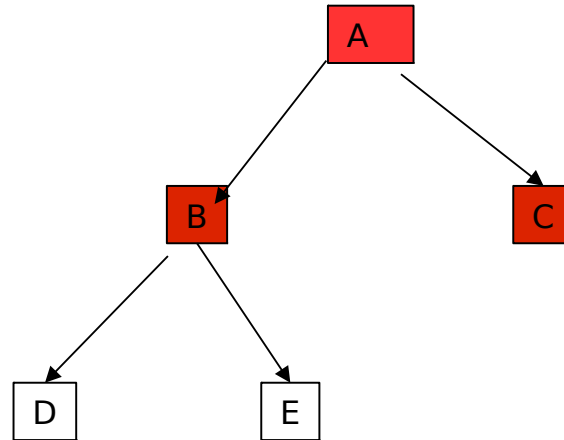
- O nó raiz é expandido, em seguida todos os nós sucessores são expandidos, então todos próximos nós sucessores são expandidos, e assim em diante.



Busca em Largura

- **Estratégia:**

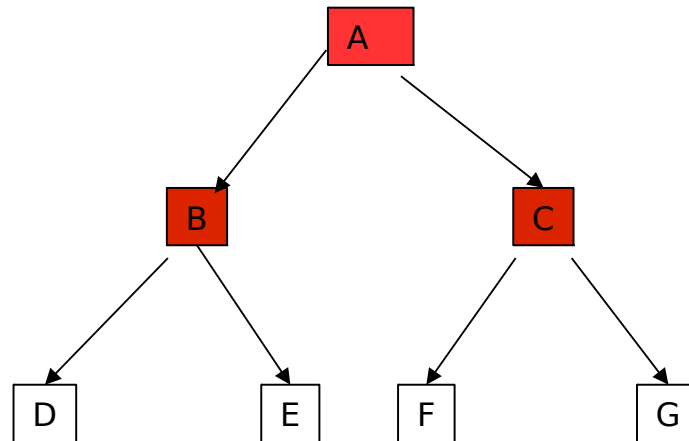
- O nó raiz é expandido, em seguida todos os nós sucessores são expandidos, então todos próximos nós sucessores são expandidos, e assim em diante.



Busca em Largura

- **Estratégia:**

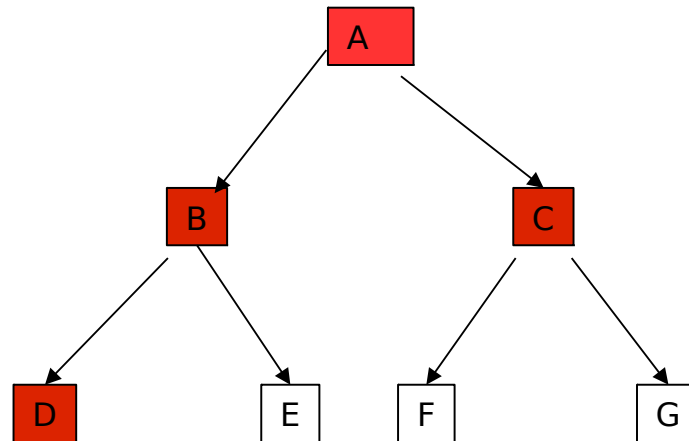
- O nó raiz é expandido, em seguida todos os nós sucessores são expandidos, então todos próximos nós sucessores são expandidos, e assim em diante.



Busca em Largura

- **Estratégia:**

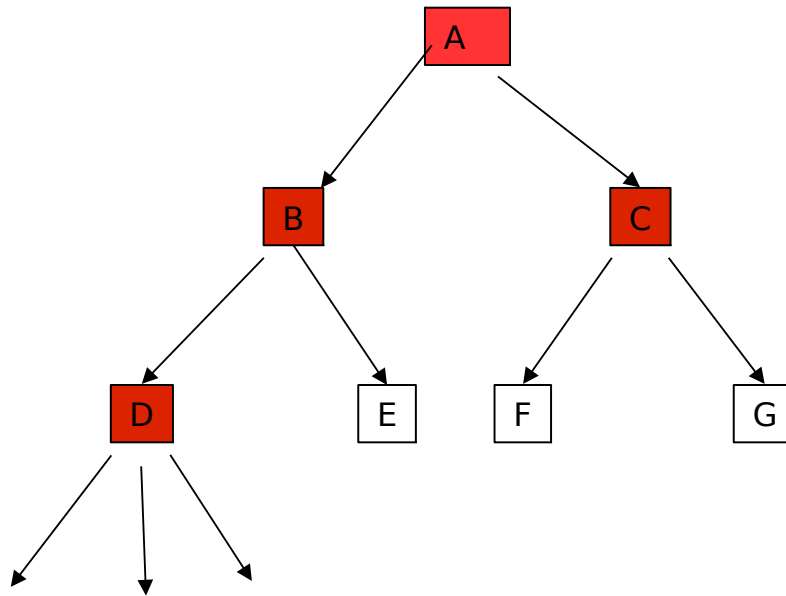
- O nó raiz é expandido, em seguida todos os nós sucessores são expandidos, então todos próximos nós sucessores são expandidos, e assim em diante.



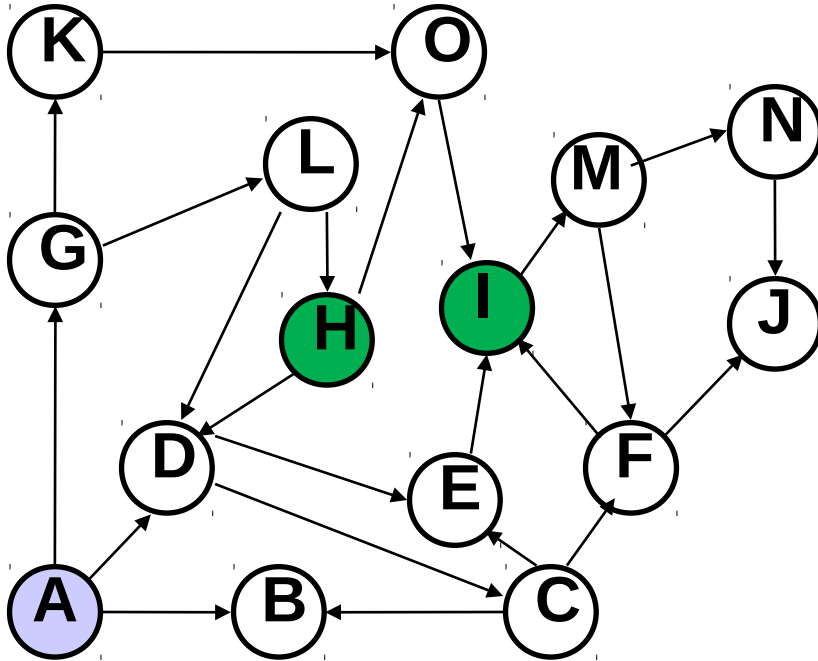
Busca em Largura

- **Estratégia:**

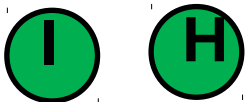
- O nó raiz é expandido, em seguida todos os nós sucessores são expandidos, então todos próximos nós sucessores são expandidos, e assim em diante.



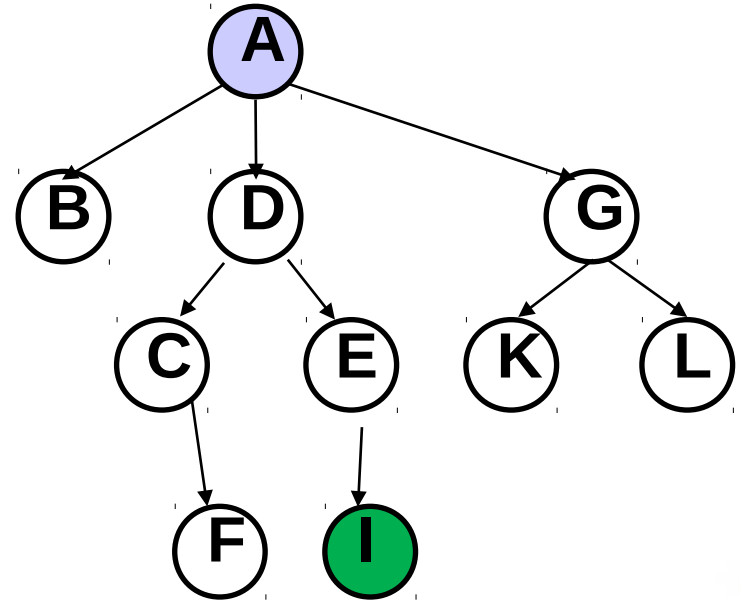
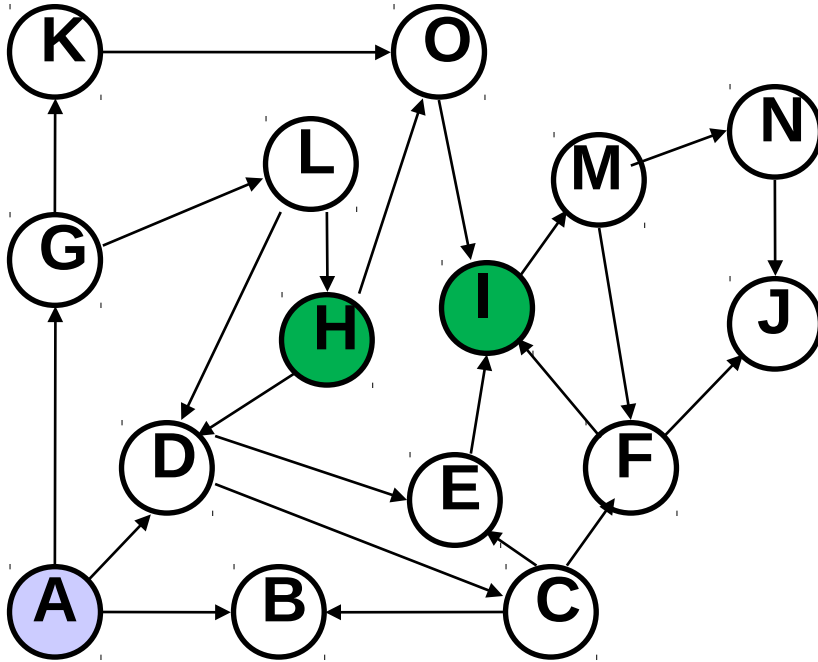
Busca em Largura



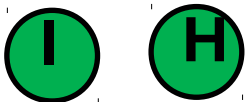
A é o estado inicial



Busca em Largura

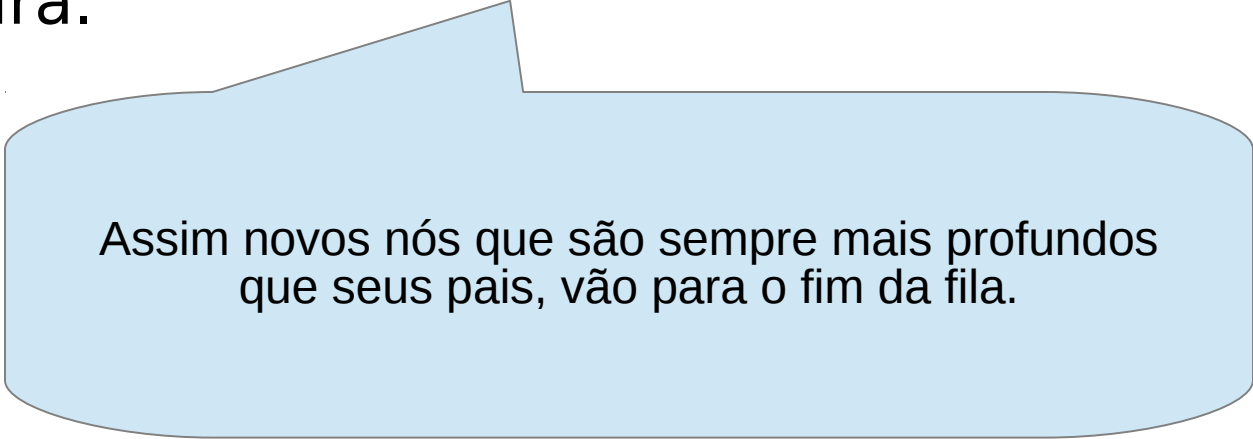


A é o estado inicial



Busca em Largura

Pode ser implementado com base no pseudocódigo da função “Busca” apresentado anteriormente. Utiliza uma estrutura de **fila FIFO** para armazenar os nós da fronteira.



Assim novos nós que são sempre mais profundos que seus pais, vão para o fim da fila.

Busca em Largura

Função Busca-Largura(*Problema*) **retorna** solução ou falha

nó ← um nó com custoCaminho=0 e estado igual a problema.ESTADO-INICIAL

Se problema.TESTE-OBJETIVO(nó.ESTADO) **retorna** SOLUÇÃO(nó)

borda ← INSIRA(nó,borda)

explorado ← conjunto vazio

repita

se VAZIA(borda) **então retorna** falha

nó ← POP (borda)

explorado ← INSIRA(nó.ESTADO, explorado)

para cada ação em problema.AÇÔES(nó.ESTADO) **faça**

filho ← NO-FILHO(problema, nó, ação)

se (filho.ESTADO) não está em explorado ou borda **então**

se problema.TESTE-OBJETIVO(filho.ESTADO) **então retorna** SOLUÇÃO(filho)

borda ← INSIRA (filho, borda)

Busca em Largura

Função Busca-Largura(*Problema*) **retorna** solução ou falha

nó ← um nó com custoCaminho=0 e estado igual a problema.ESTADO-INICIAL

Se problema.TESTE-OBJETIVO(nó.ESTADO) **retorna** SOLUÇÃO(nó)

borda ← INSIRA(nó,borda)

explorado ← conjunto vazio

repita

se VAZIA(borda) **então ret**

nó ← POP (borda)

explorado ← INSIRA(nó.ESTADO, explorado)

para cada ação em problema.AÇÕES(nó.ESTADO) **faça**

filho ← NO-FILHO(problema, nó, ação)

se (filho.ESTADO) não está em explorado ou borda **então**

se problema.TESTE-OBJETIVO(filho.ESTADO) **então retorna** SOLUÇÃO(filho)

borda ← INSIRA (filho, borda)

borda é uma fila FIFO com nó como elemento único

Busca em Largura

Função Busca-Largura(*Problema*) **retorna** solução ou falha

nó ← um nó com custoCaminho=0 e estado igual a problema.ESTADO-INICIAL

Se problema.TESTE-OBJETIVO(nó.ESTADO) **retorna** SOLUÇÃO(nó)

borda ← INSIRA(nó,borda)

explorado ← conjunto vazio

repita

se VAZIA(borda) **então retorna** falha

nó ← POP (borda)

explorado ← INSIRA(nó.ESTADO,explorado)

para cada ação em problema.AÇÕES

filho ← NO-FILHO(problema, nó, ação)

se (filho.ESTADO) não está em explorado ou borda **então**

se problema.TESTE-OBJETIVO(filho.ESTADO) **então retorna** SOLUÇÃO(filho)

borda ← INSIRA (filho, borda)

Escolhe o nó mais raso na borda

Busca em Largura

Função Busca-Largura(*Problema*) **retorna** solução ou falha

nó ← um nó com custoCaminho=0 e estado igual a problema.ESTADO-INICIAL

Se problema.TESTE-OBJETIVO(nó.ESTADO) **retorna** SOLUÇÃO(nó)

borda ← INSIRA(nó,borda)

explorado ← conjunto vazio

repita

se VAZIA(borda) **então retorna** falha

nó ← POP (borda)

explorado ← INSIRA(nó.ESTADO, explorado)

para cada ação em problema.AÇÕES(nó.ESTADO) **faça**

filho ← NO-FILHO(problema, nó, ação)

se (filho.ESTADO) não está em explorado ou borda **então**

se problema.TESTE-OBJETIVO(filho.ESTADO) **então retorna** SOLUÇÃO(filho)

borda ← INSIRA (filho, borda)

Função que gera um nó sucessor

Propriedades da busca em largura

Suponha que temos uma árvore em que cada estado tenha **b** sucessores e que **d** é a profundidade do nó meta menos profundo.

Completa?

Solução ótima?

Tempo?

Espaço?

Propriedades da busca em largura

Suponha que temos uma árvore em que cada estado tenha **b** sucessores e que **d** é a profundidade do nó meta menos profundo.

- **Completa?**

Sim. Garante encontrar a solução se existir uma (e se b for finito)

- **Solução ótima?**

Sempre encontra a primeira solução mais rasa. Devolve a solução ótima se todos os passos tiverem o mesmo custo.

- **Tempo?**

Número total de nós gerados = $1 + b + b^2 + b^3 + \dots + b^d = O(b^d)$

- **Espaço?** $O(b^d)$

Guarda todos os nós na memória

Propriedades da busca em largura

Suponha que temos uma árvore em que cada estado tenha **b** sucessores e que **d** é a profundidade do nó meta menos profundo.

•Com

Sim. Garante (se b for finito)

São gerados b nós no primeiro nível

se b for

•Solução ótima?

Sempre encontra a primeira solução encontrada é a mais rasa. Devolve a solução ótima se todos os passos tiverem o mesmo custo.

•Tempo?

Número total de nós gerados = $1 + b + b^2 + b^3 + \dots + b^d = O(b^d)$

•Espaço? $O(b^d)$

Guarda todos os nós na memória

Propriedades da busca em largura

Suponha que temos uma árvore em que cada estado tenha **b** sucessores e que **d** é a profundidade do nó meta menos profundo.

- **C**

Sin
fini

Se a solução está na profundidade d e é o último nó gerado em aquele nível, temos b^d nós.

a (e se b for

- **Solução ótima?**

Sempre encontra a primeira solução. Devolve a solução ótima se todos os passos tiverem o mesmo custo.

- **Tempo?**

Número total de nós gerados = $1 + b + b^2 + b^3 + \dots + b^d = O(b^d)$

- **Espaço?** $O(b^d)$

Guarda todos os nós na memória

Propriedades da busca em largura

Suponha que temos uma árvore em que cada estado tenha **b** sucessores e que **d** é a profundidade do nó meta menos profundo.

- **Completa?**

Sim. Garante encontrar a solução se existir uma (e se b for finito)

- **Solução ótima?**

Sempre encontra a primeira solução mais rasa. Devolve a solução ótima se todos os passos tiverem o mesmo custo.

- **Tempo?**

Número total de nós gerados = $1 + b + b^2 + b^3 + \dots + b^d = O(b^d)$

- **Espaço?** $O(b^d)$

Guarda todos os nós na memória



É assustador!!!

Propriedades da busca em largura

** Considerando $b = 10$, geração de um milhão de nós por segundo e cada nó ocupando 1KB de memória.*

Profundidade (d)	Nós	Tempo	Memória
2	110	0.11 ms	107 KB
4	11110	11 ms	10.6 MB
6	10^6	1.1 seg	1 GB
8	10^8	2 min	103 GB
10	10^{10}	3 horas	10 TB
12	10^{12}	13 dias	1 PB

Busca de Custo Uniforme

- **Estratégia:**

- Expande sempre o nó de menor custo de caminho (menor $g(n)$).
- Feito através do armazenamento da borda como uma fila de prioridade.
- A borda deve permitir testes eficientes de pertença em conjunto. Por isso deve combinar recursos de fila de prioridade e de uma tabela hash.
- Se o custo de todos os passos for o mesmo, o algoritmo acaba sendo o mesmo que a busca em largura.

Busca de Custo Uniforme

- **Estratégia:**
 - Expande sempre o nó de menor custo de caminho.

A

Busca de Custo Uniforme

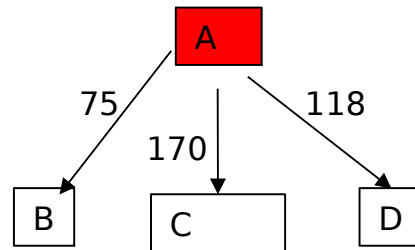
- **Estratégia:**
 - Expande sempre o nó de menor custo de caminho.



Busca de Custo Uniforme

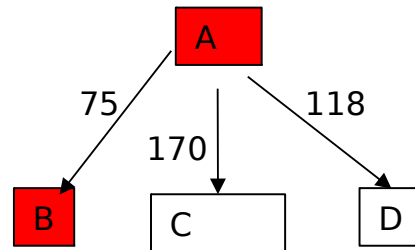
- **Estratégia:**

- Expande sempre o nó de menor custo de caminho.



Busca de Custo Uniforme

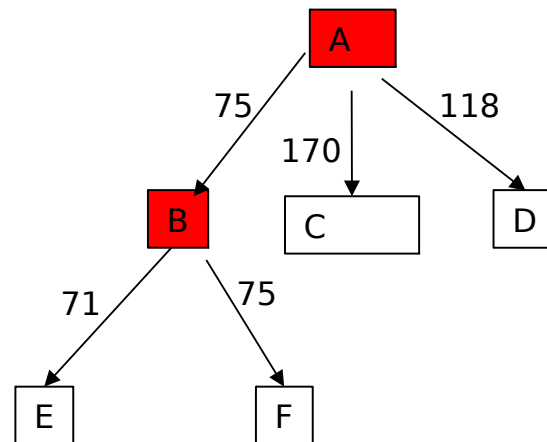
- **Estratégia:**
 - Expande sempre o nó de menor custo de caminho.



Busca de Custo Uniforme

- **Estratégia:**

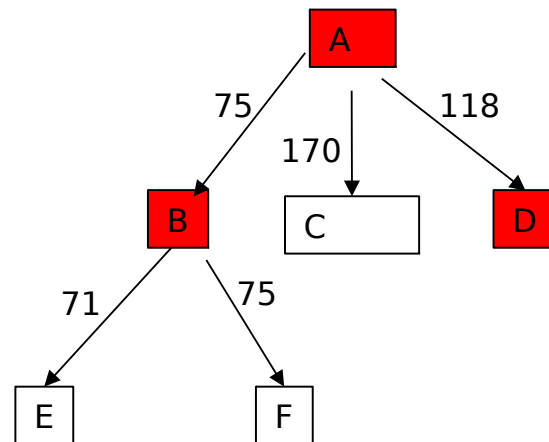
- Expande sempre o nó de menor custo de caminho.



Busca de Custo Uniforme

- **Estratégia:**

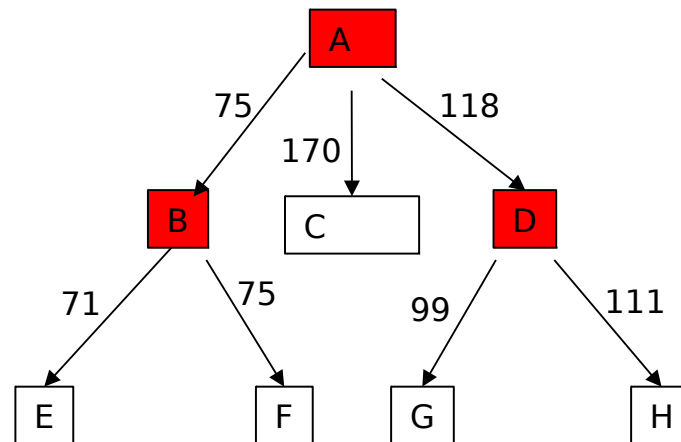
- Expande sempre o nó de menor custo de caminho.



Busca de Custo Uniforme

- **Estratégia:**

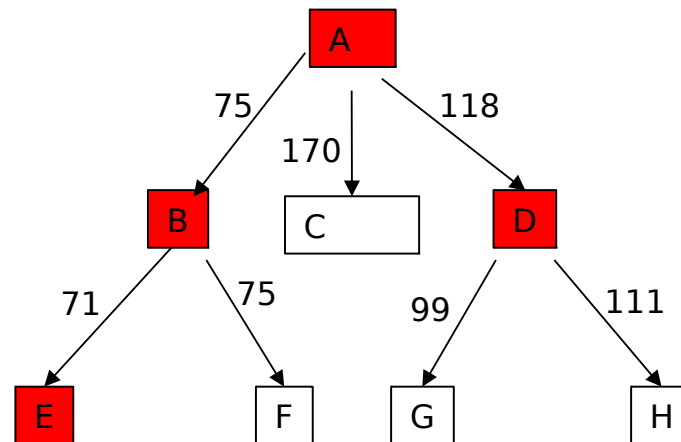
- Expande sempre o nó de menor custo de caminho.



Busca de Custo Uniforme

- **Estratégia:**

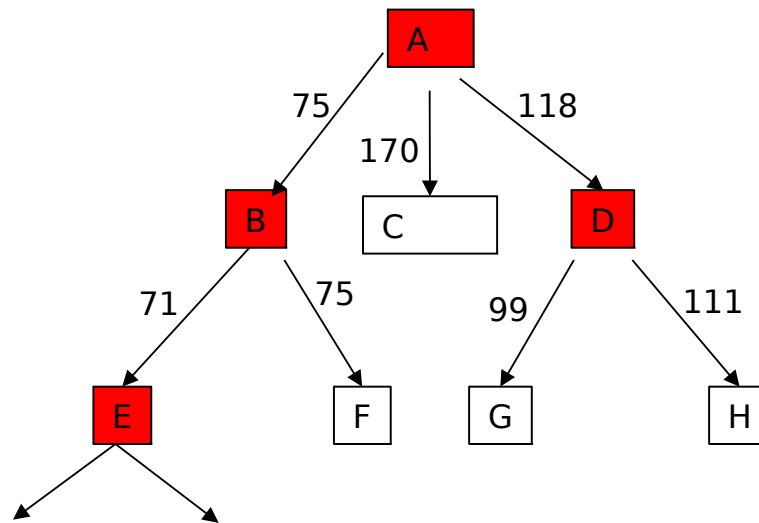
- Expande sempre o nó de menor custo de caminho.



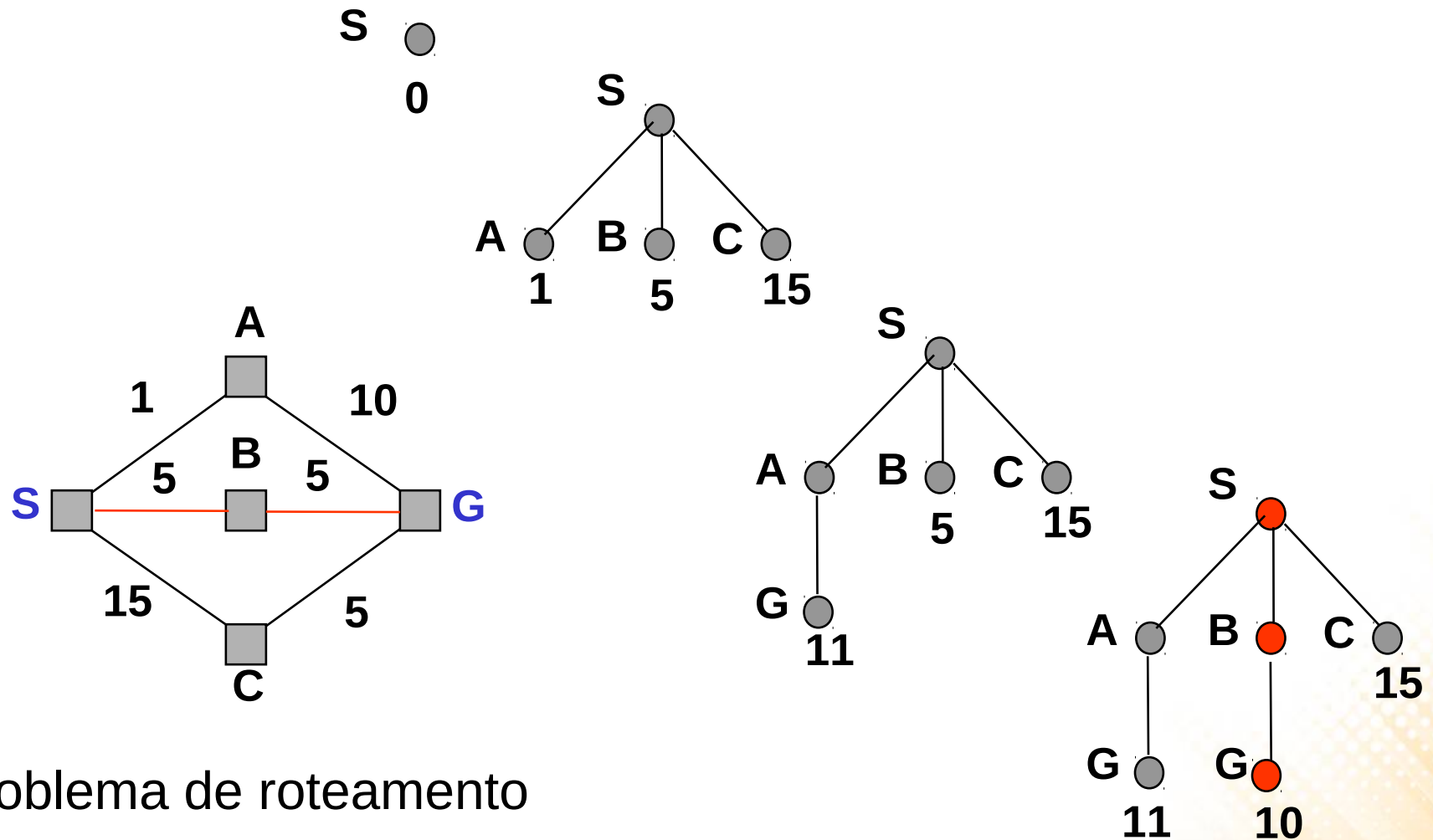
Busca de Custo Uniforme

- **Estratégia:**

- Expande sempre o nó de menor custo de caminho.

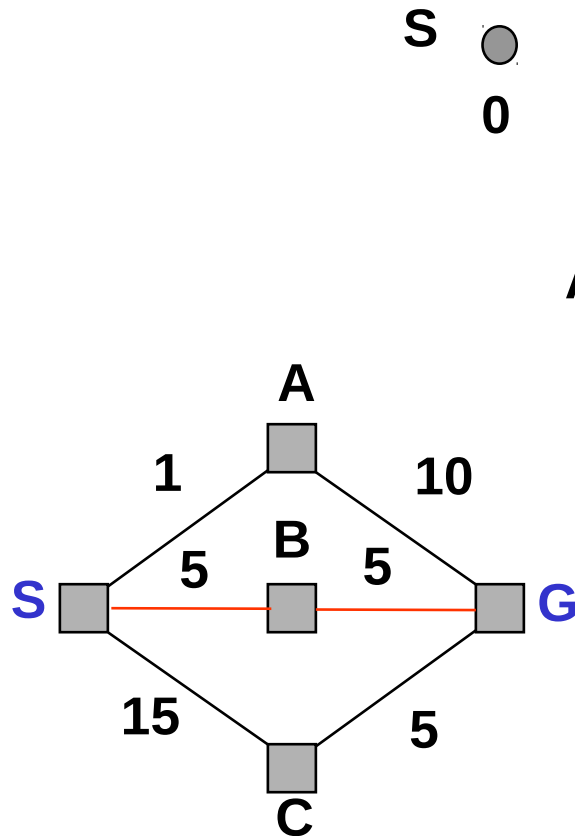


Busca de Custo Uniforme

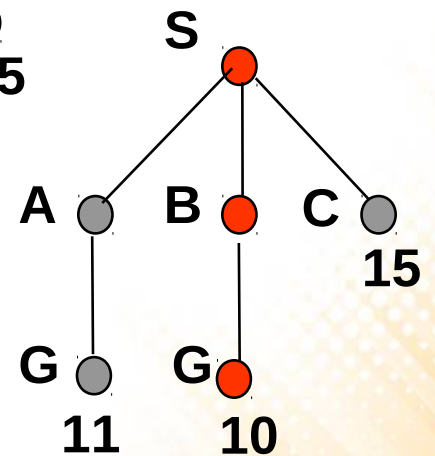
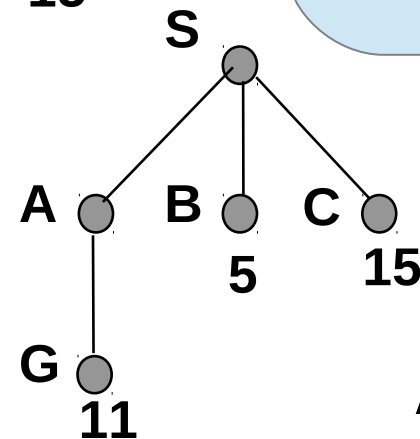
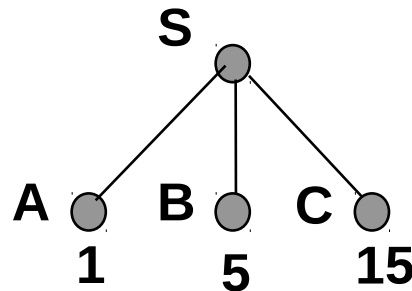
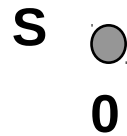


Busca de Custo Uniforme

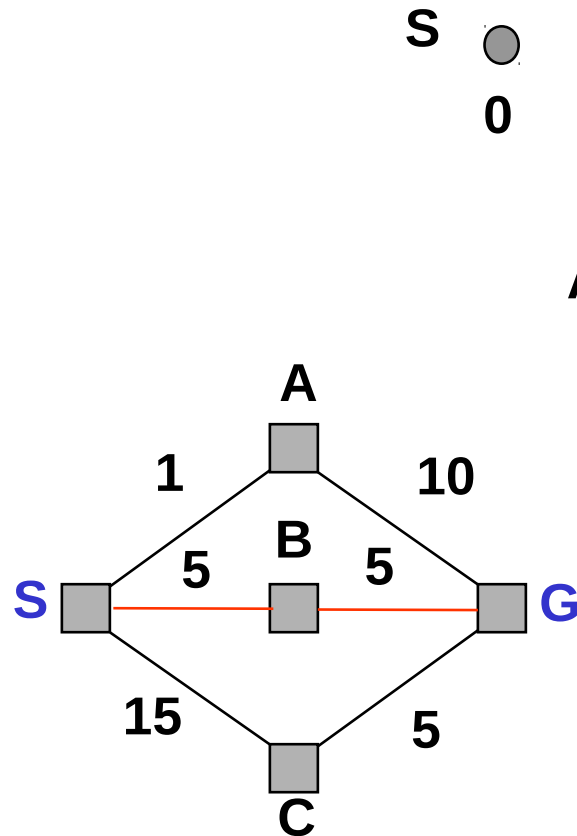
O teste de meta é feito quando um nó é escolhido da borda



Problema de roteamento

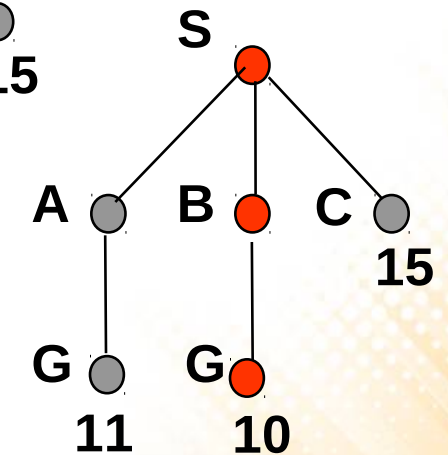
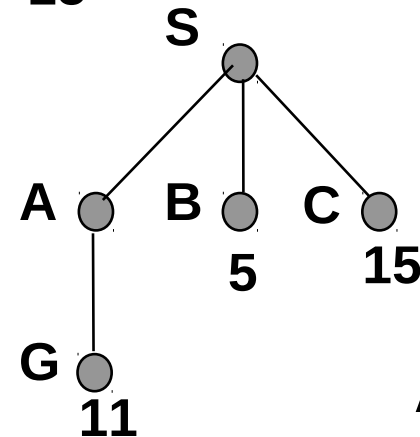
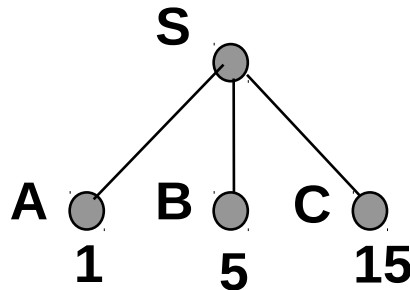


Busca de Custo Uniforme



Problema de roteamento

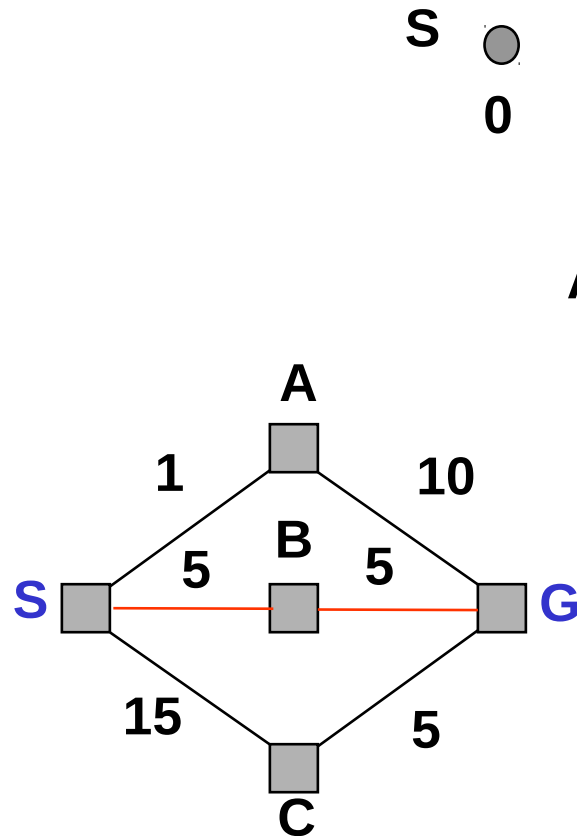
S 0



Na fila de prioridade
temos:

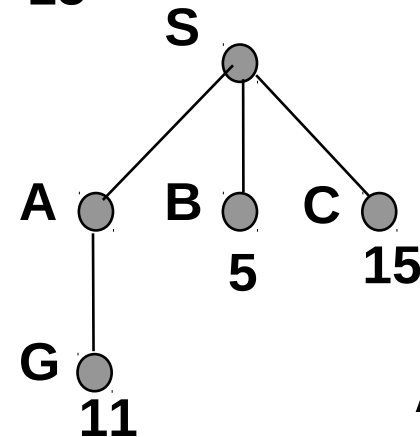
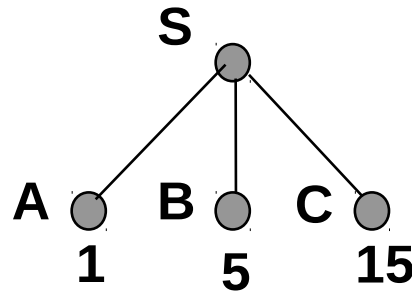
B(5) G(11) C(15)

Busca de Custo Uniforme

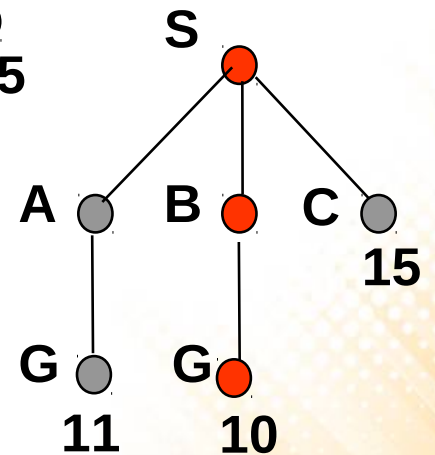


Problema de roteamento

S 0

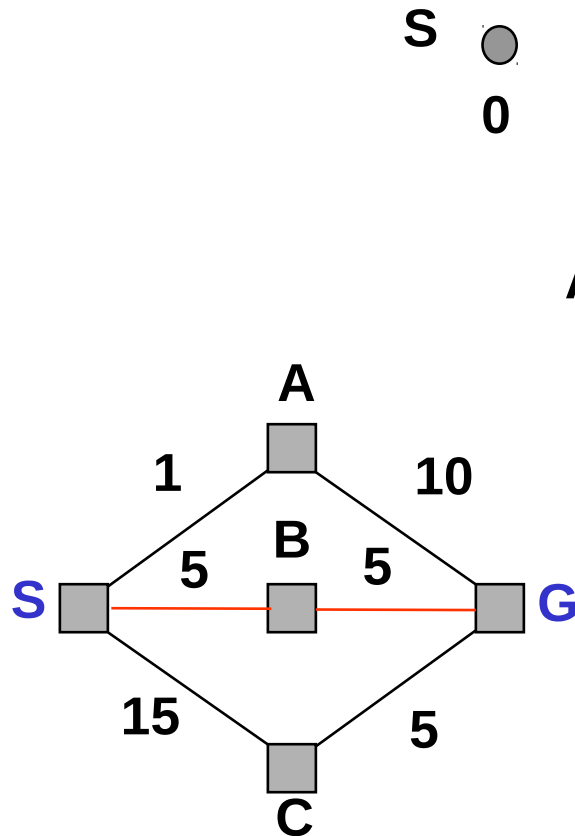


O nó escolhido da fila é o nó B e ele não é a meta

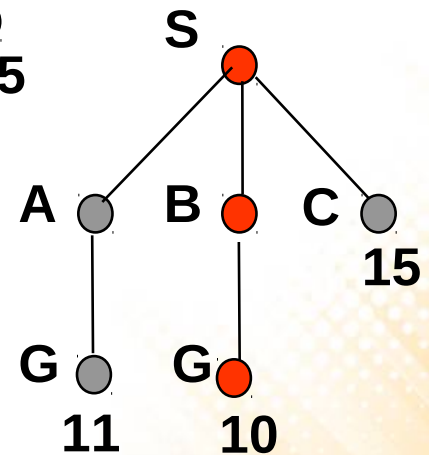
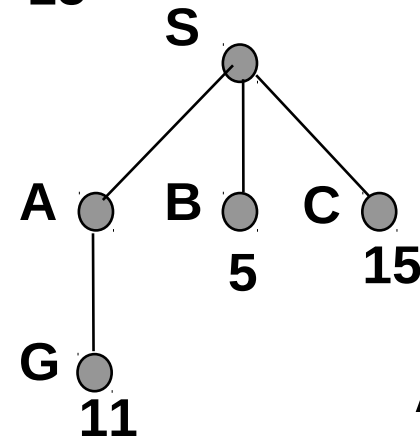
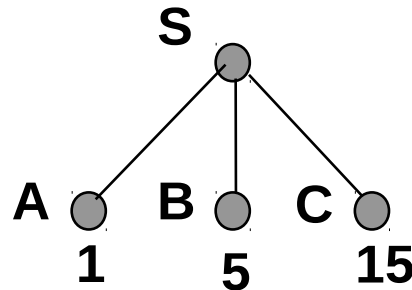
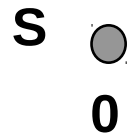


Busca de Custo Uniforme

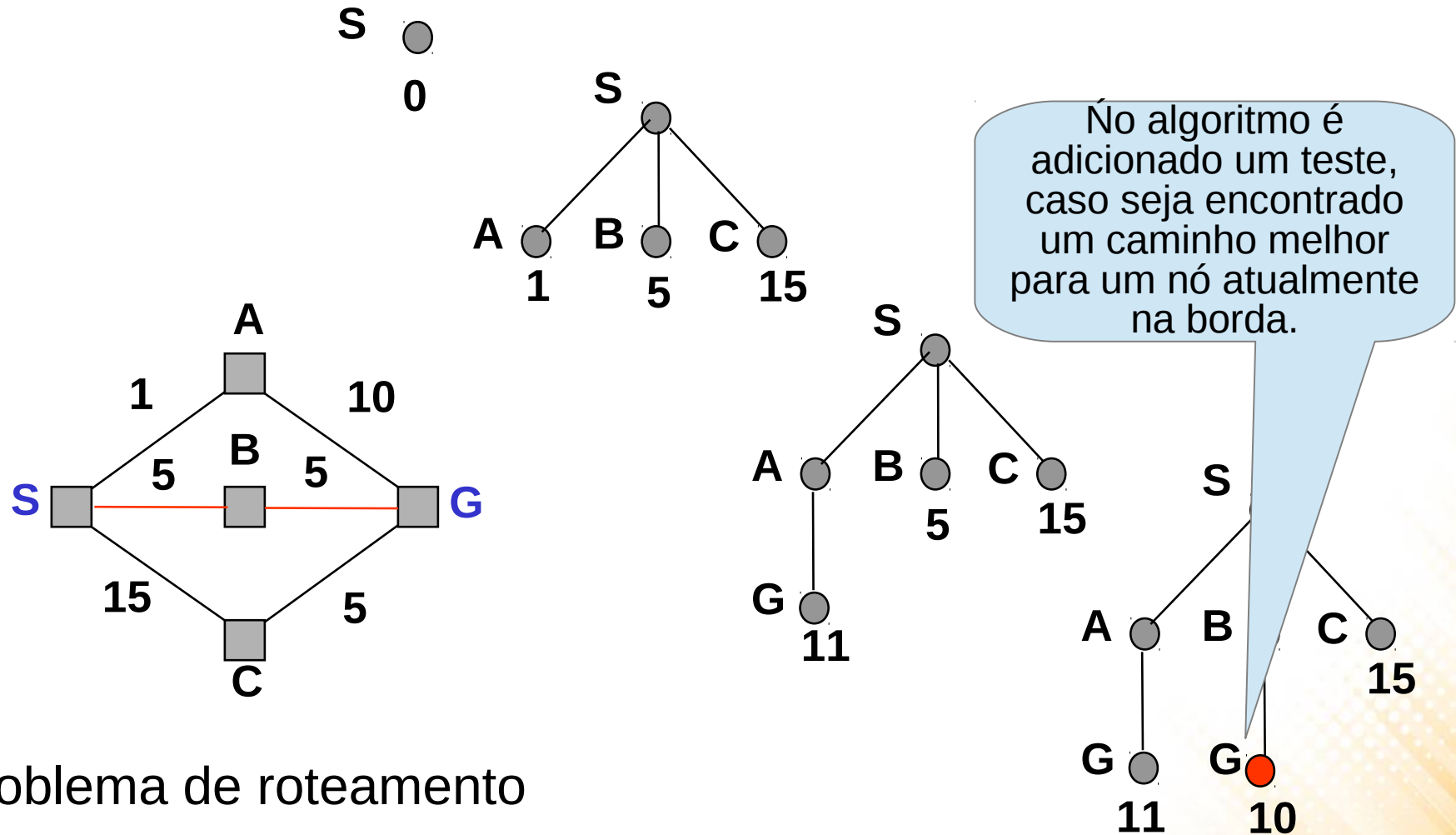
Geramos os sucessores de B e os colocamos na borda, no nosso exemplo temos que colocar G, mas agora com custo 10.



Problema de roteamento

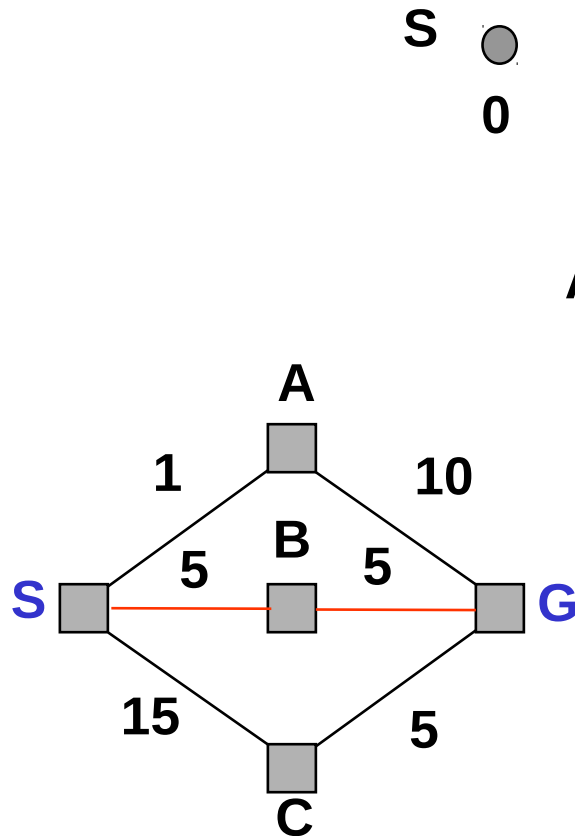


Busca de Custo Uniforme

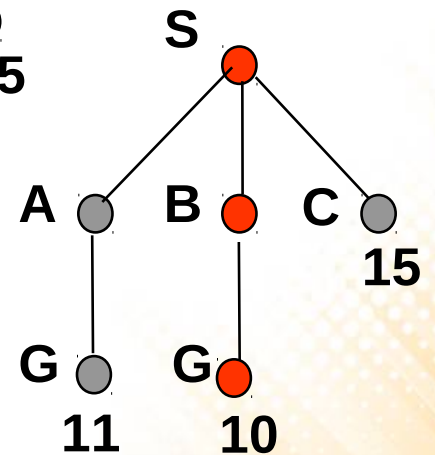
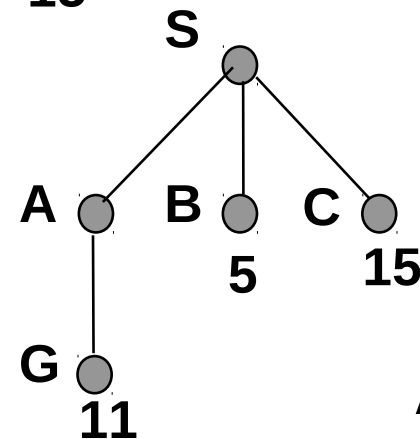
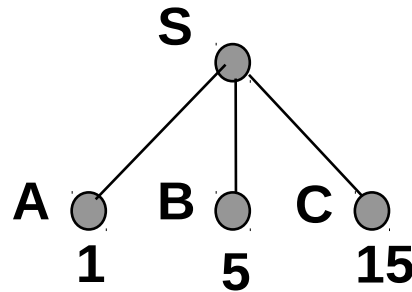


Busca de Custo Uniforme

Na fila de prioridade
temos:
G(10) C(15)



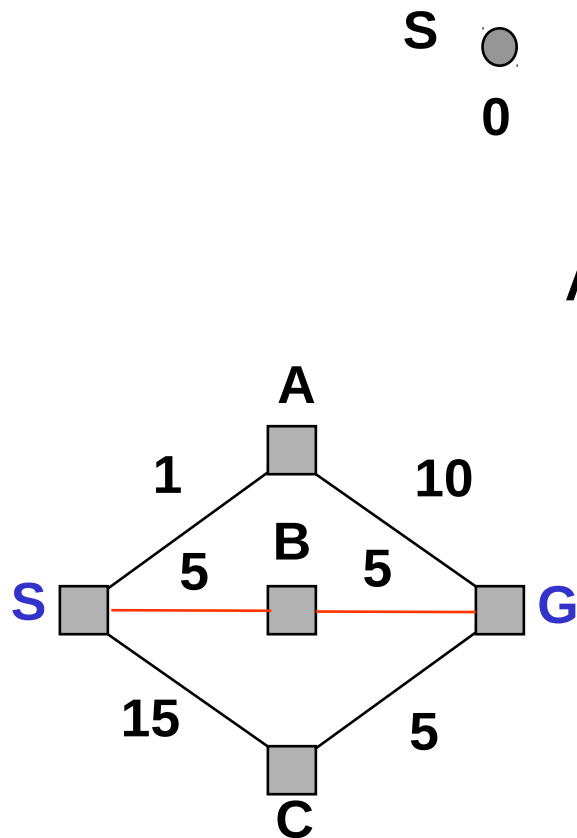
S 0



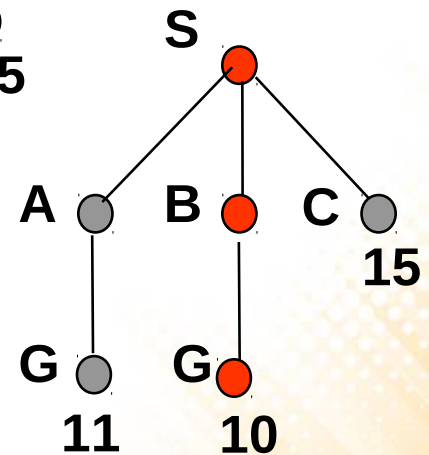
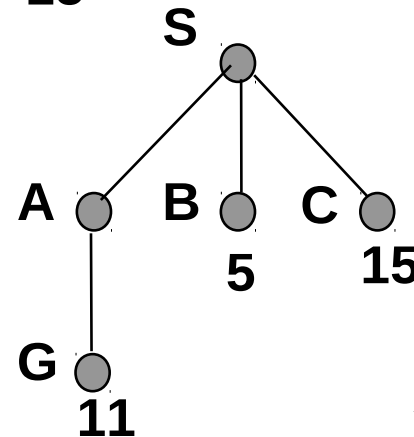
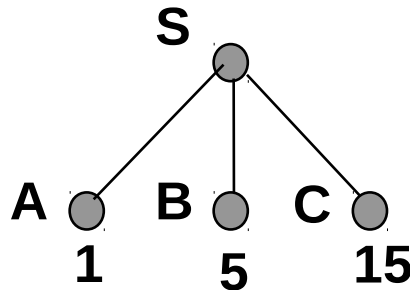
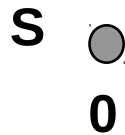
Problema de roteamento

Busca de Custo Uniforme

Escolhemos G da fila de prioridades, que já é um estado meta.



Problema de roteamento



Propriedades da busca de Custo Uniforme

Suponha que temos uma árvore em que cada estado tenha **b** sucessores, **C** é o custo ótimo e **alpha** o custo por passo.

Completa?

Solução ótima?

Tempo?

Espaço?

Propriedades da busca de Custo Uniforme

Suponha que temos uma árvore em que cada estado tenha **b** sucessores, **C** é o custo ótimo e **alpha** o custo por passo.

Completa? Sim, se b for finito e o custo de todos os passos for positivo.

Solução ótima? Sim

Tempo? $O(b^{1+(C/\alpha)})$

Espaço? $O(b^{1+(C/\alpha)})$

Propriedades da busca de Custo Uniforme

Suponha que temos uma árvore em que cada estado tenha **b** sucessores, **C** é o custo ótimo e **alpha** o custo por passo.

Completa? Sim, se b for finito e o custo de todos os passos for positivo.

Solução ótima? Sim

É o número de passos, isto é, essa divisão é igual a d.

Tempo?

$$O(b^{1+(C/\alpha)})$$

Espaço?

$$O(b^{1+(C/\alpha)})$$

Busca de Custo Uniforme

- Quando todos os custos de passo são iguais, a busca de custo uniforme é similar à busca em largura.
 - Exceto que a busca em largura para logo que gerar o objetivo, enquanto a busca de custo uniforme não.

Busca em Profundidade

- **Estratégia:**

- Expande o nó mais profundo na borda atual da árvore de busca.

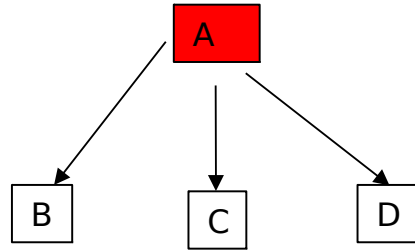
A

Busca em Profundidade

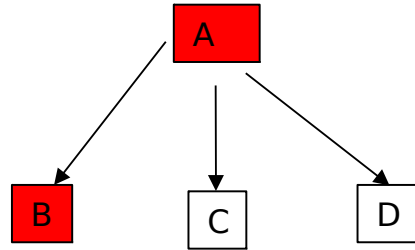
A

A small red square with a black border, containing the capital letter 'A' in black. It is positioned in the center of the slide.

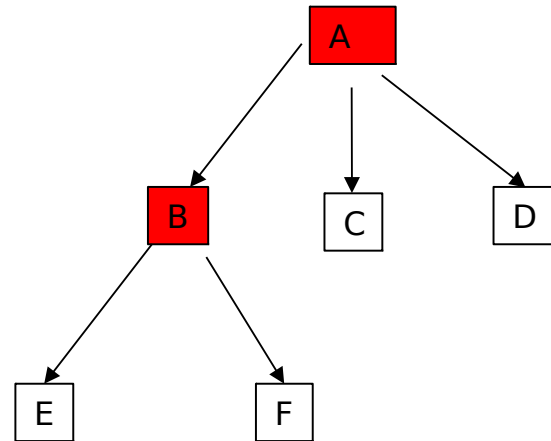
Busca em Profundidade



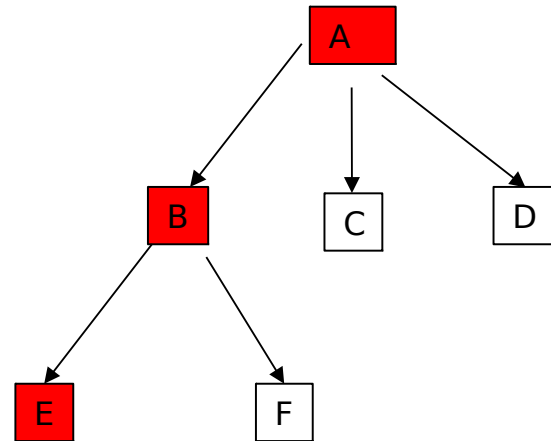
Busca em Profundidade



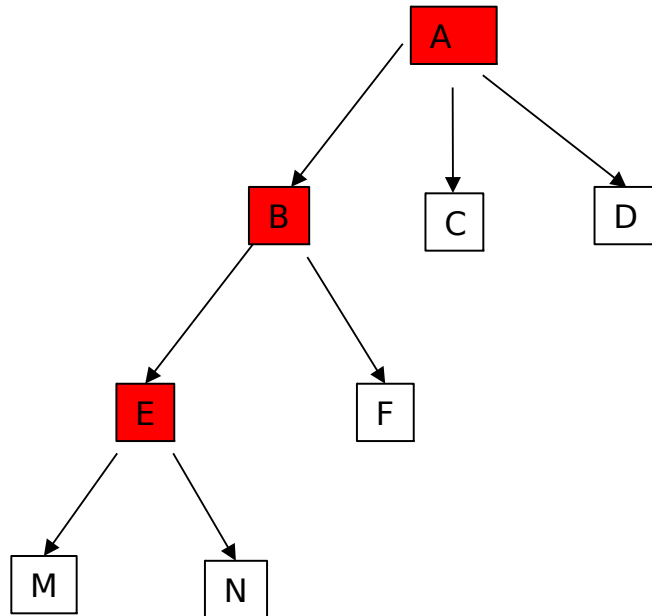
Busca em Profundidade



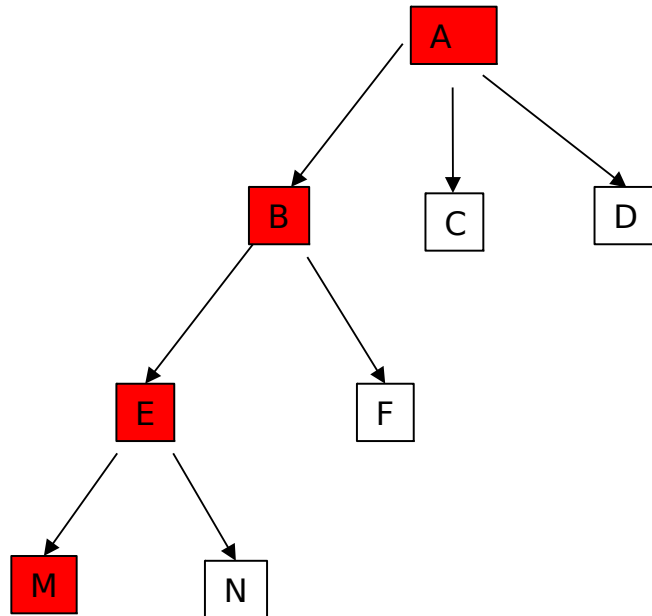
Busca em Profundidade



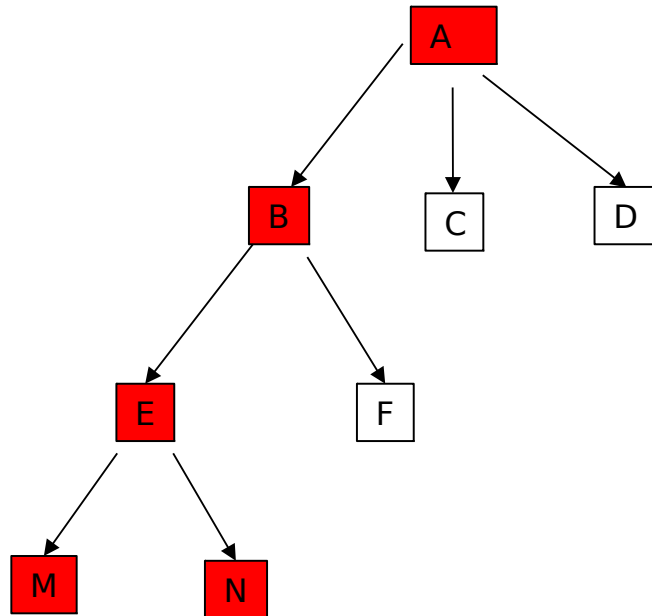
Busca em Profundidade



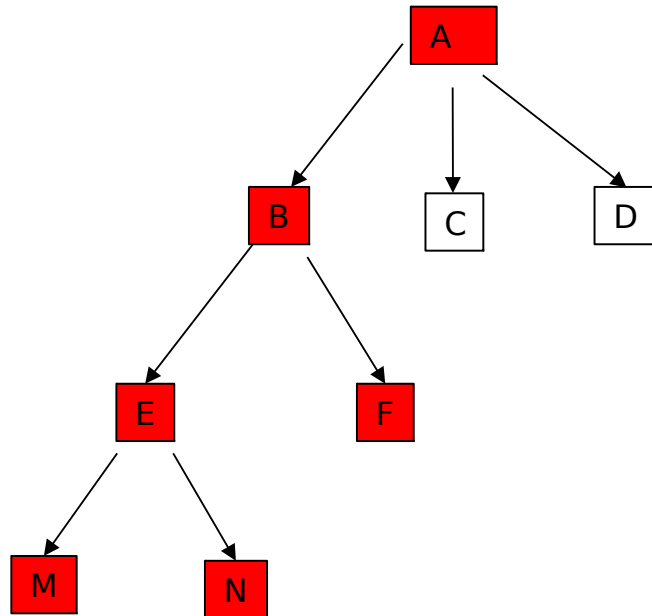
Busca em Profundidade



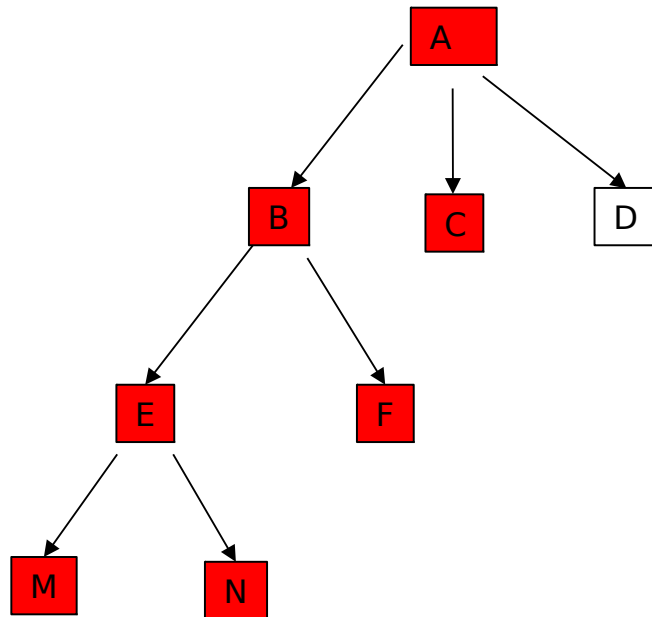
Busca em Profundidade



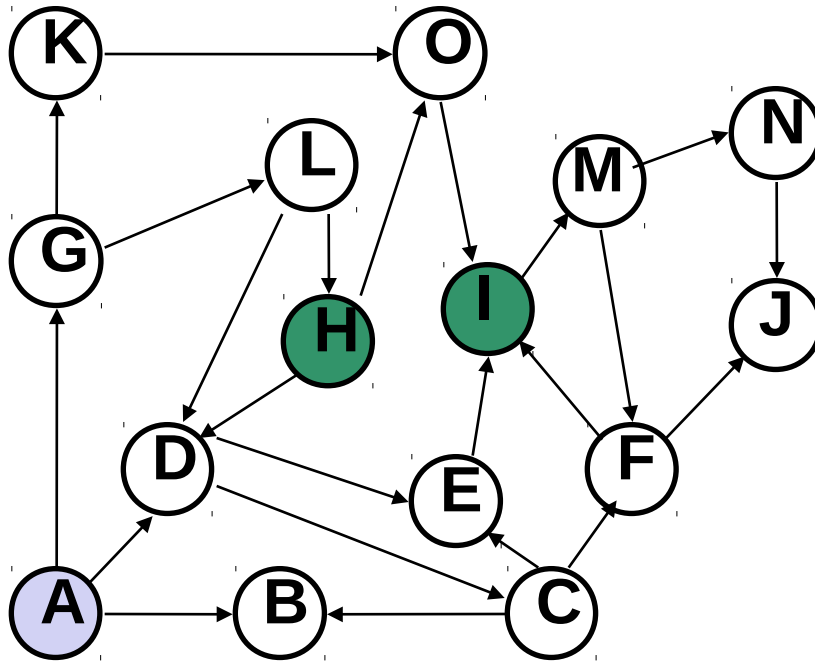
Busca em Profundidade



Busca em Profundidade



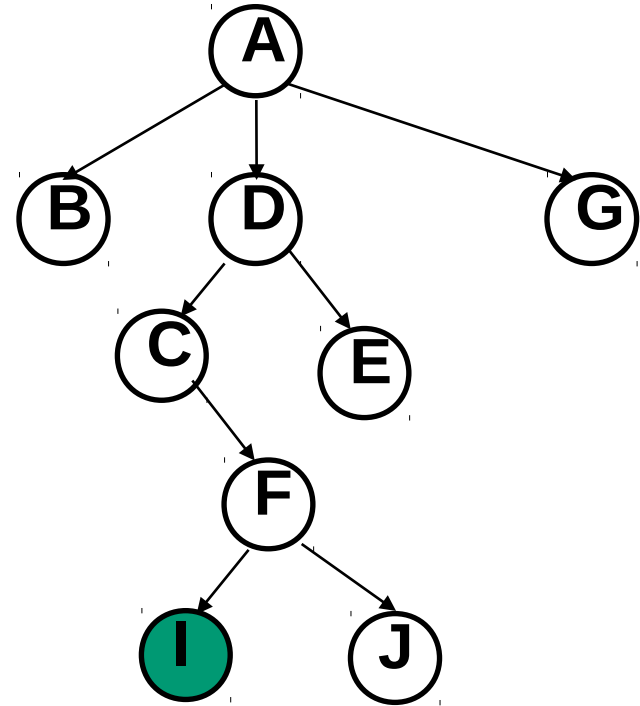
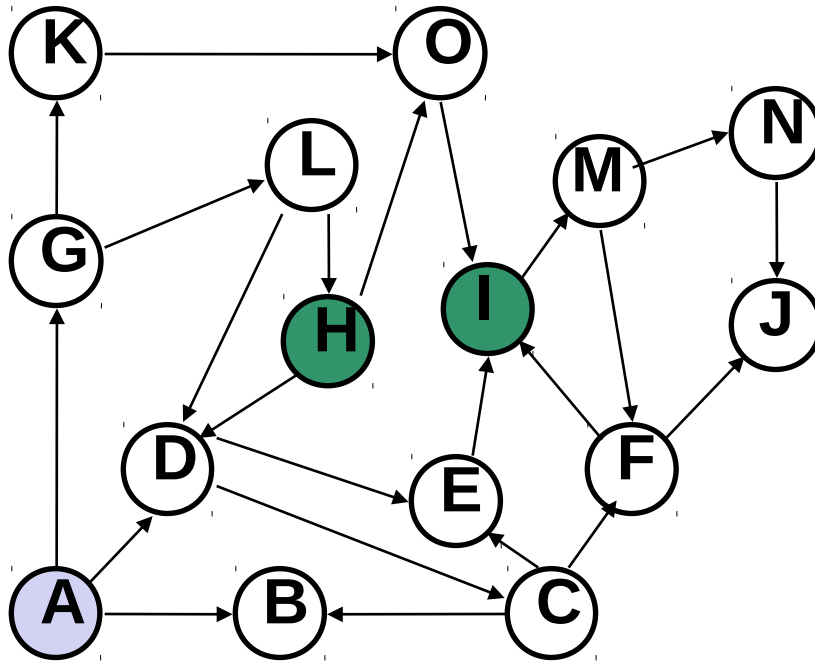
Busca em Profundidade



A é o estado inicial

I **H** são estados meta

Busca em Profundidade



A é o estado inicial

I **H** são estados meta

Propriedades da busca em Profundidade

- Pode ser implementado com base no pseudocódigo da função “Busca” apresentado anteriormente. Utiliza uma estrutura de fila LIFO (pilha) para armazenar os nós da fronteira.
- Pode também ser implementado de forma recursiva.

Propriedades da busca em Profundidade

Completa?

Tempo?

Espaço?

Solução ótima?

Propriedades da busca em Profundidade

Completa? Não. Falha em espaços com profundidade infinita, ou com loops. Se modificado para evitar estados repetidos é *completo em espaços finitos*.

Tempo? $O(b^m)$: ruim se m é muito maior que d

Espaço? Se não precisamos do conjunto de explorados para tratar nós repetidos é $O(bm)$, i.e., linear com m .

Solução ótima? Não. Devolve a primeira solução encontrada

Busca em Profundidade

- **Não recomendado** para grandes árvores de busca ou quando a **profundidade máxima é desconhecida**.
- O algoritmo pode fazer uma busca muito longa mesmo quando a resposta do problema está localizada a poucos nós da raiz da árvore.

Busca em profundidade iterativa

- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.

A

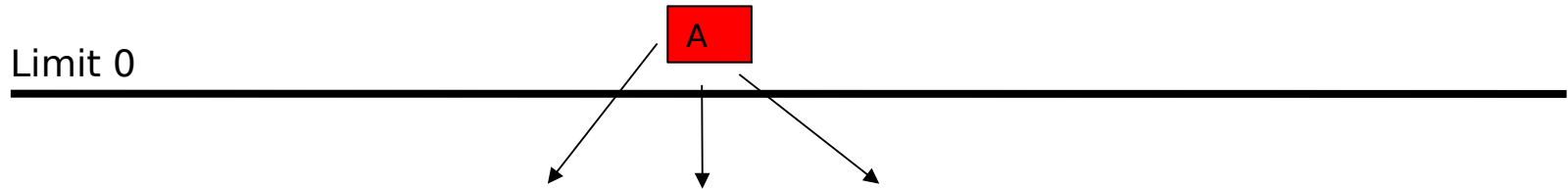
Busca em profundidade iterativa

- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.



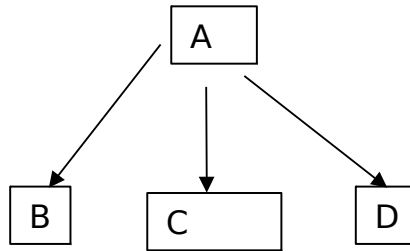
Busca em profundidade iterativa

- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.



Busca em profundidade iterativa

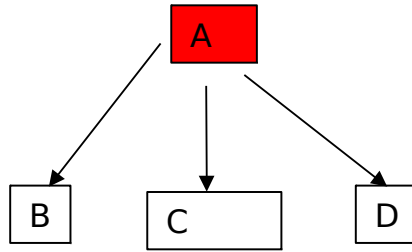
- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.



Limit 1

Busca em profundidade iterativa

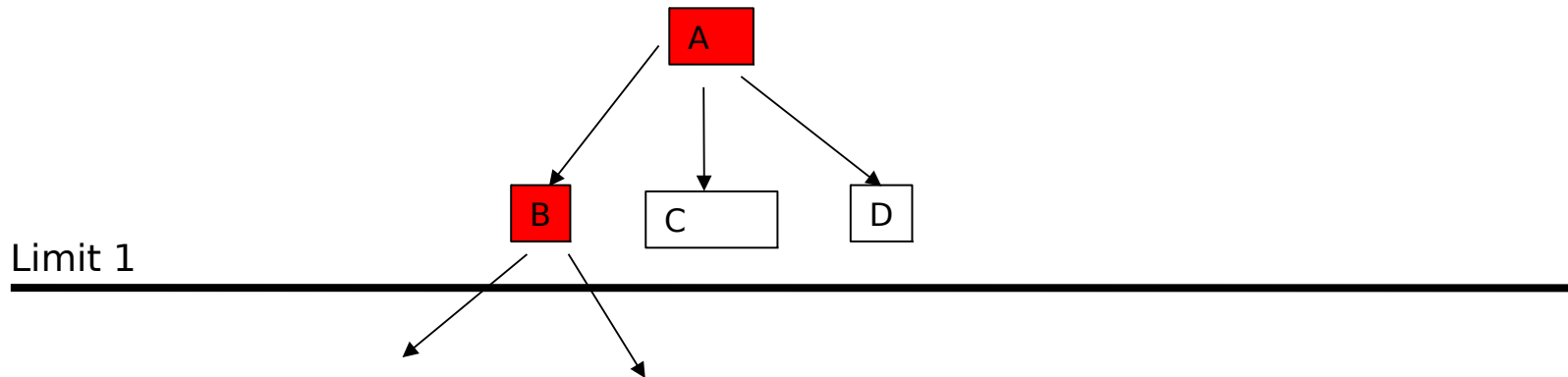
- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.



Limit 1

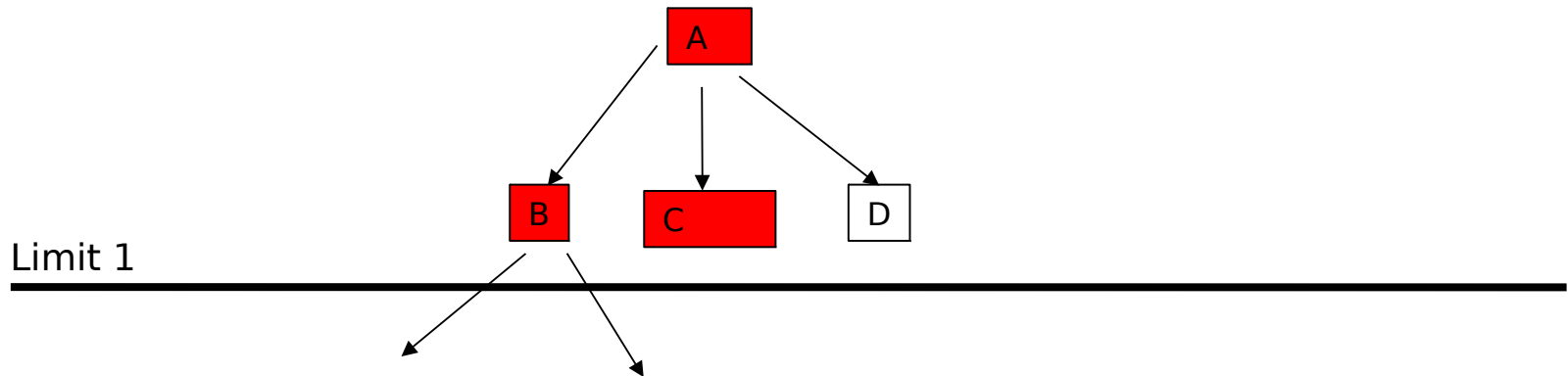
Busca em profundidade iterativa

- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.



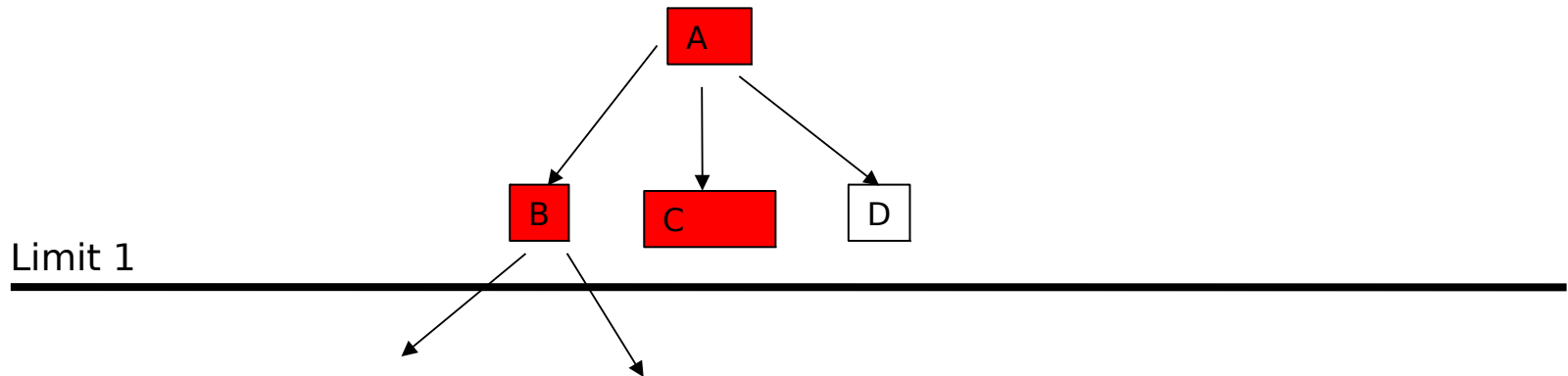
Busca em profundidade iterativa

- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.



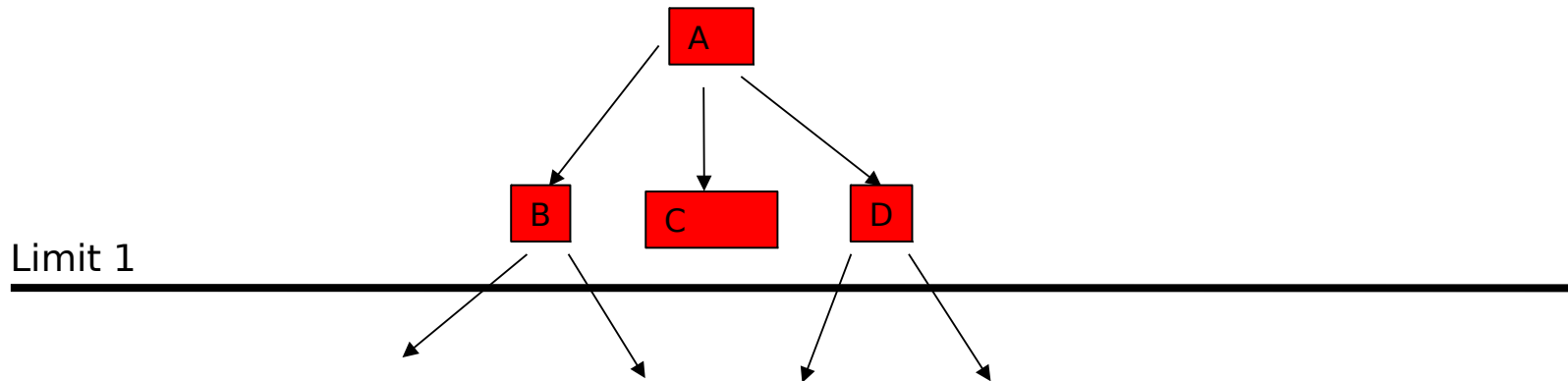
Busca em profundidade iterativa

- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.



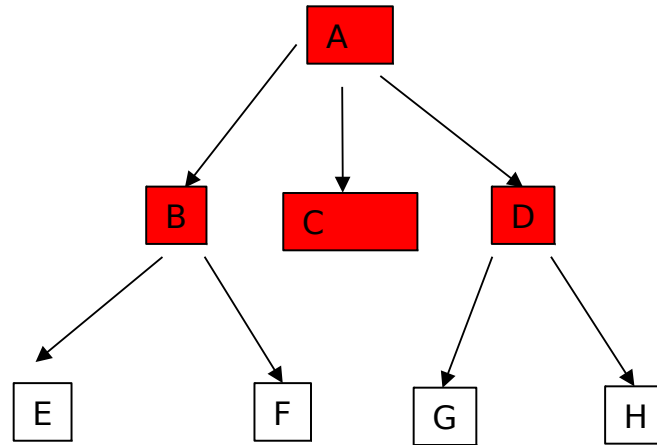
Busca em profundidade iterativa

- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.



Busca em profundidade iterativa

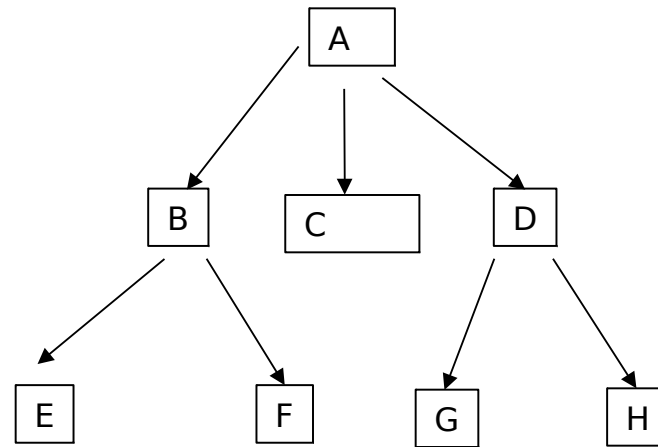
- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.



Limit 2

Busca em profundidade iterativa

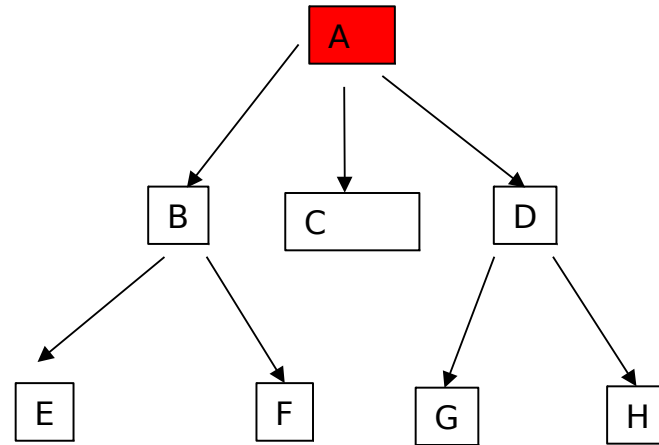
- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.



Limit 2

Busca em profundidade iterativa

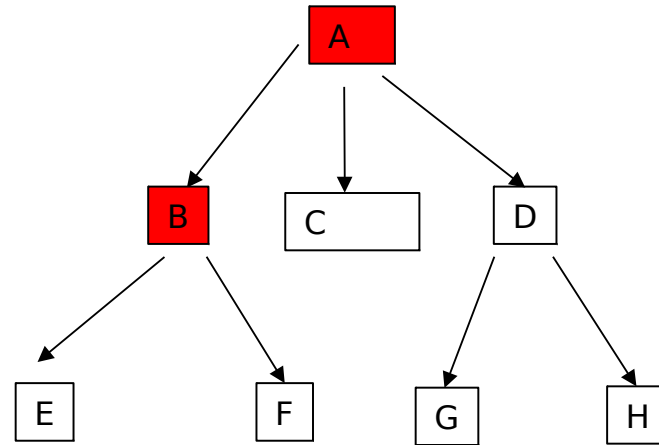
- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.



Limit 2

Busca em profundidade iterativa

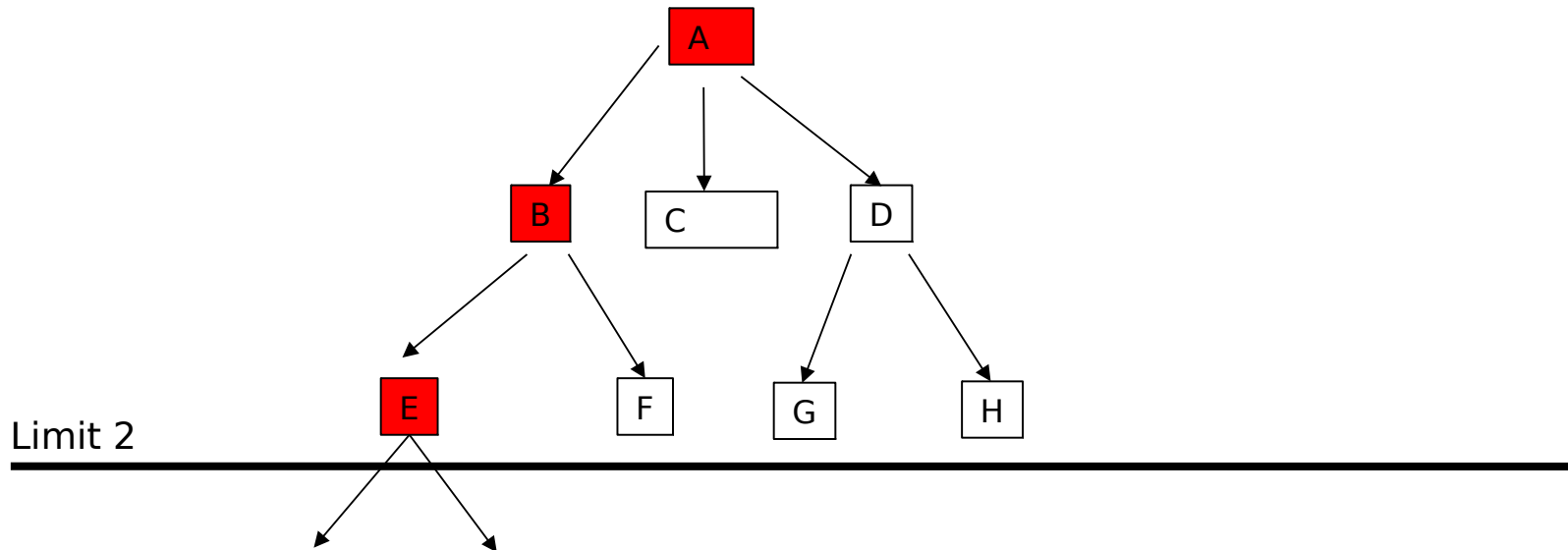
- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.



Limit 2

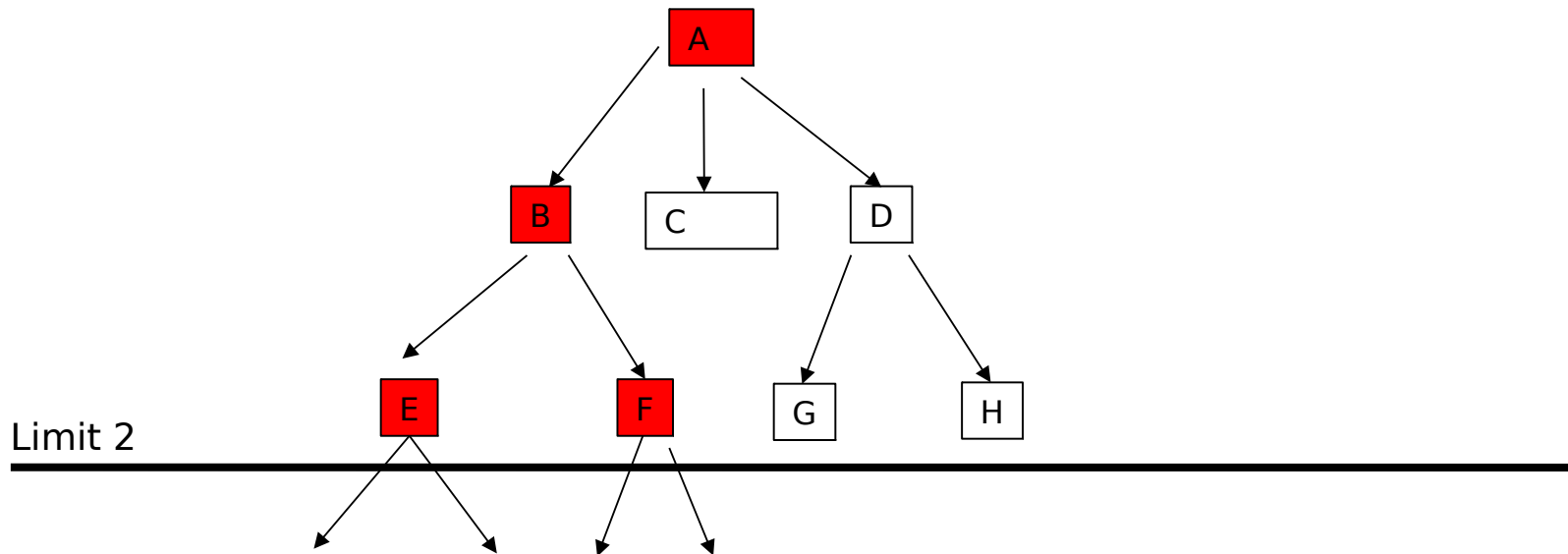
Busca em profundidade iterativa

- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.



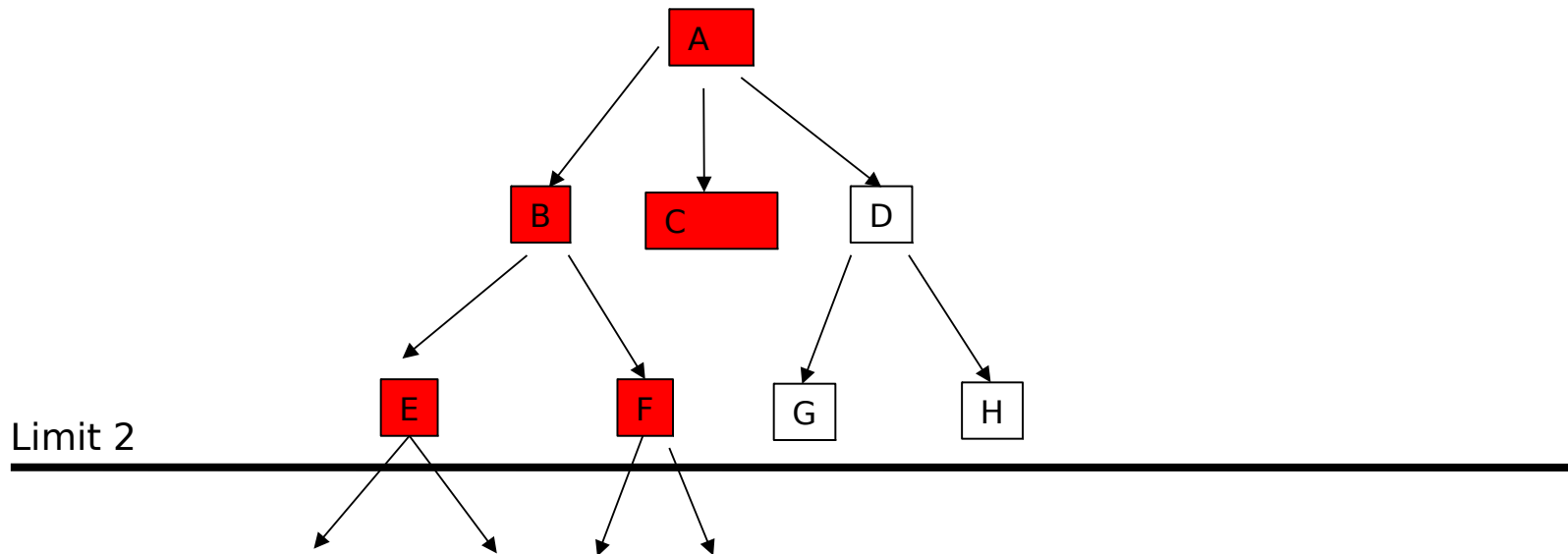
Busca em profundidade iterativa

- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.



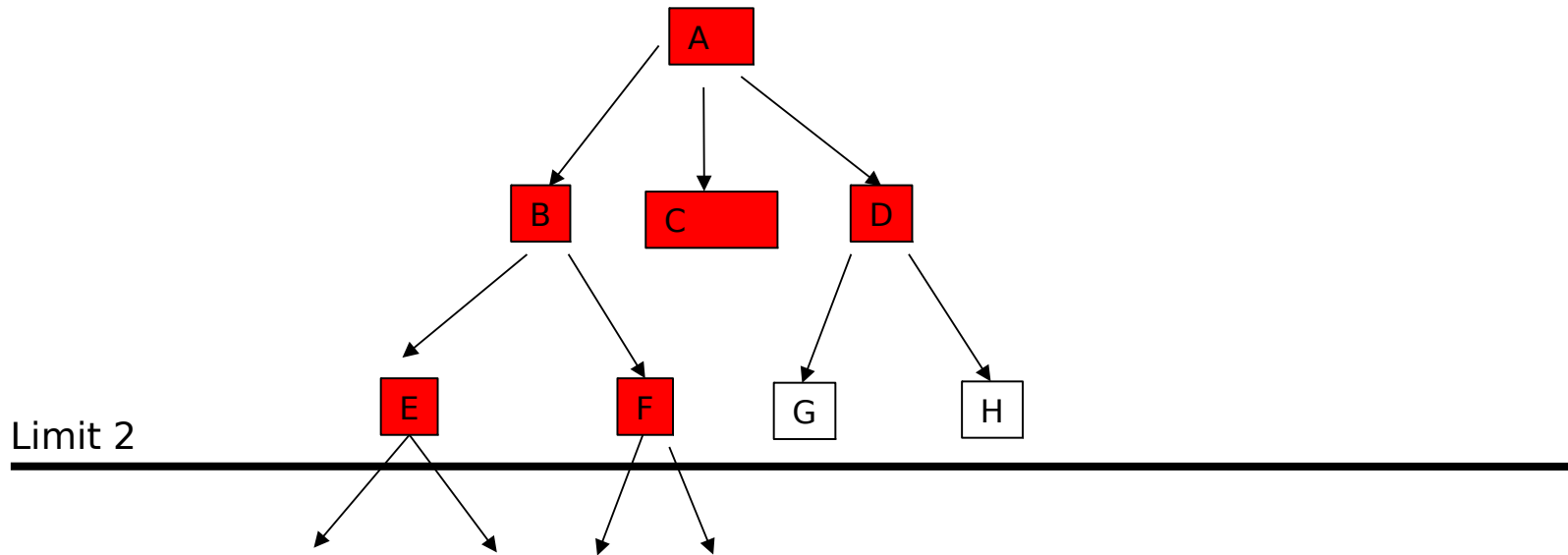
Busca em profundidade iterativa

- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.



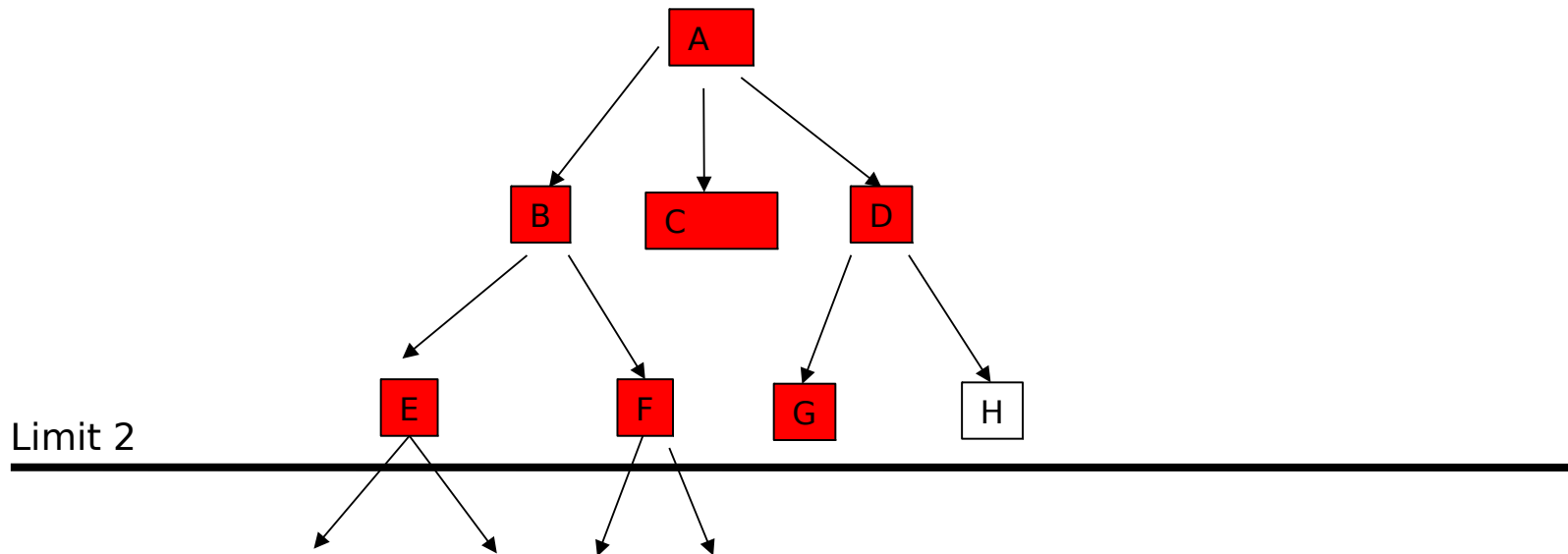
Busca em profundidade iterativa

- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.



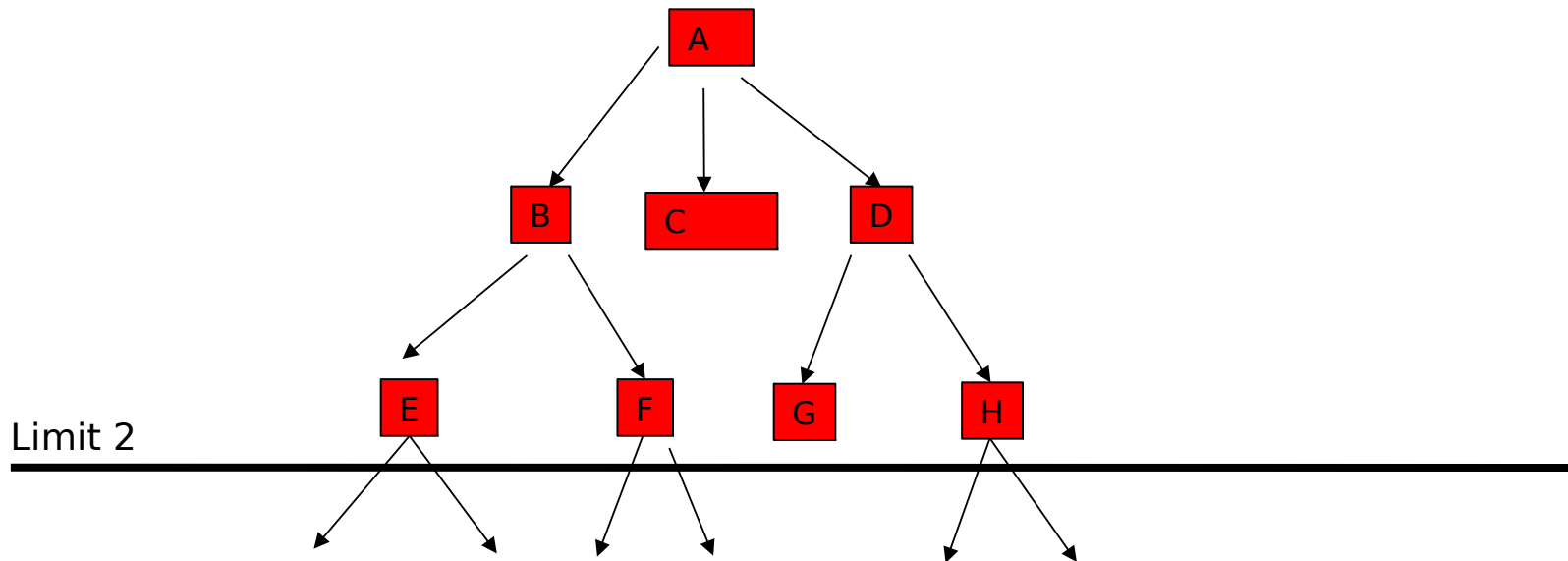
Busca em profundidade iterativa

- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.



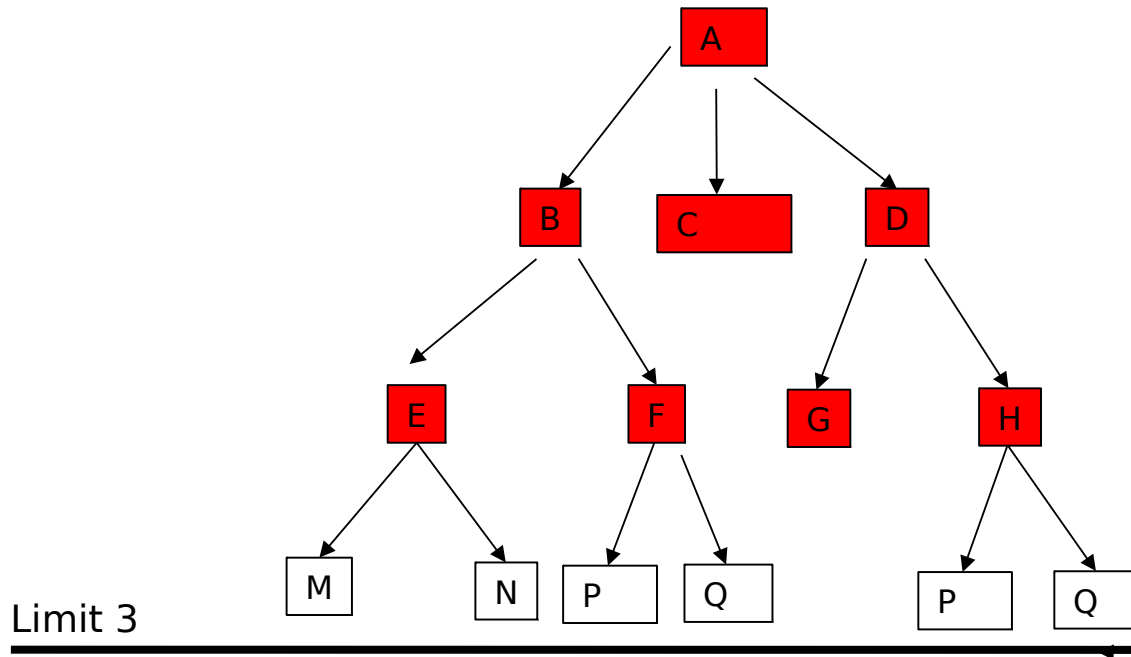
Busca em profundidade iterativa

- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.



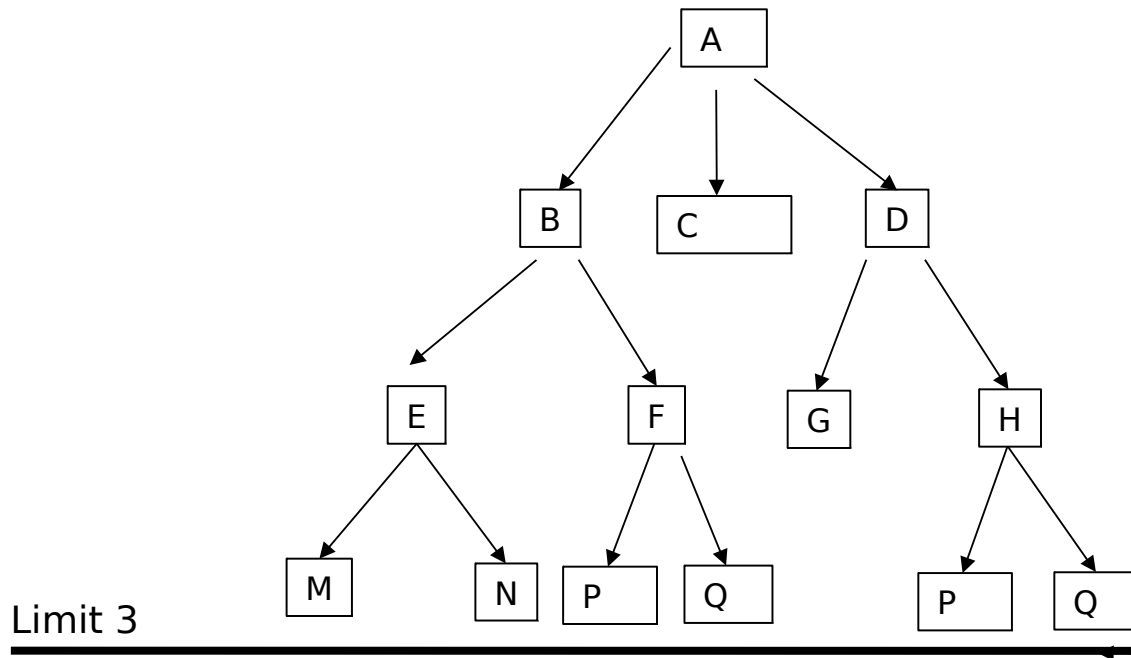
Busca em profundidade iterativa

- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.



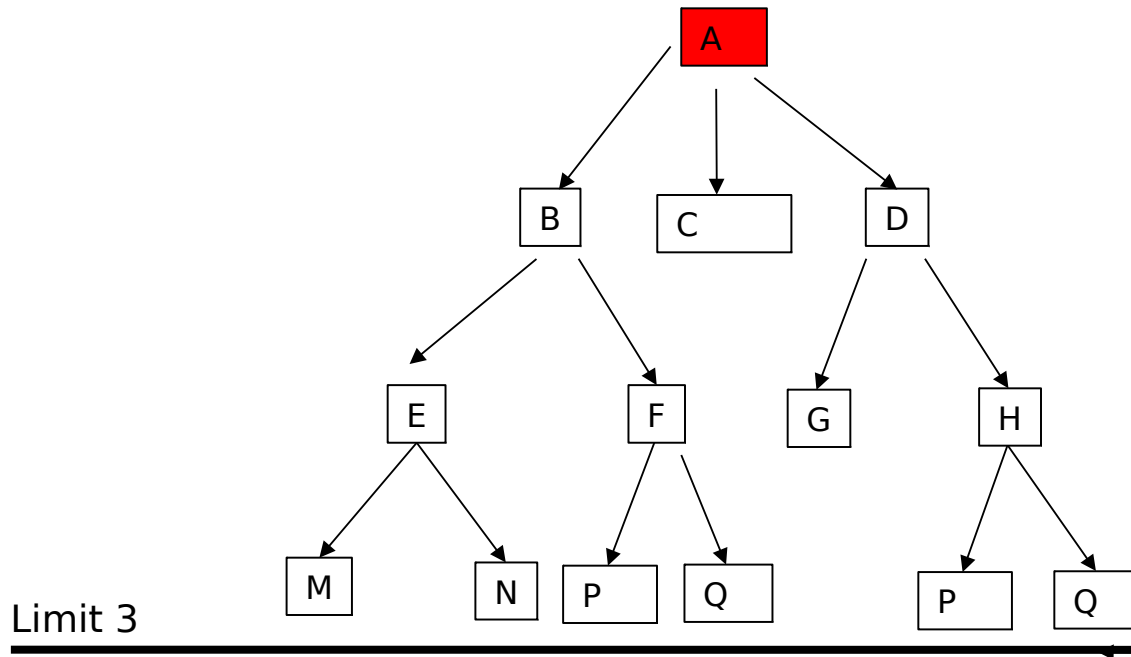
Busca em profundidade iterativa

- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.

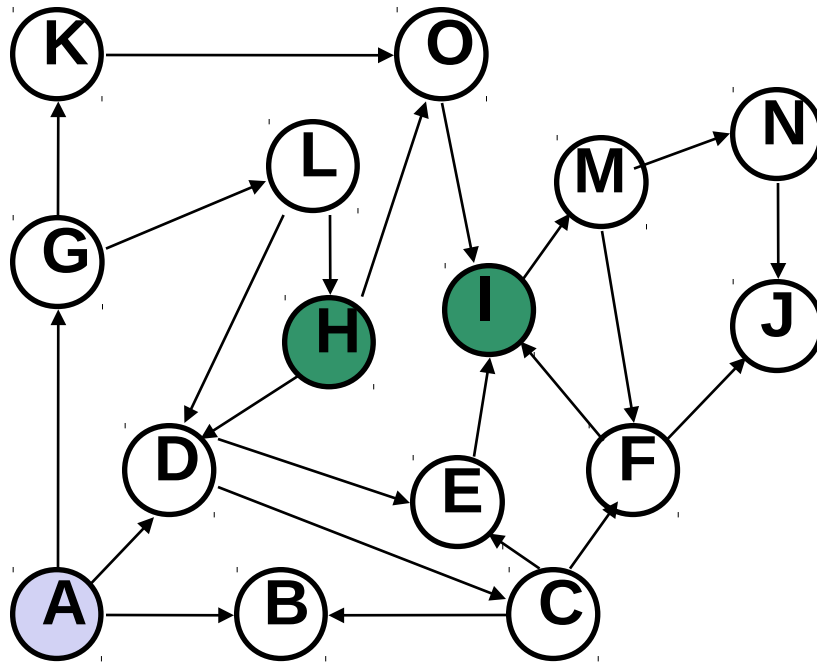


Busca em profundidade iterativa

- **Estratégia:** Consiste em uma busca em profundidade onde o limite de profundidade é incrementado gradualmente.



Busca em profundidade iterativa



?

A é o estado inicial

I **H** são estados meta

Busca em profundidade iterativa

- Combina os benefícios da busca em largura com os benefícios da busca em profundidade.
- Evita o problema de caminhos muito longos ou infinitos.
- A repetição da expansão de estados não é tão ruim, pois a maior parte dos estados está nos níveis mais baixos.

Busca em profundidade iterativa

Completa? Sim, se b for finito.

Tempo? $db + (d-1)b^2 + \dots + 1b^d = O(b^d)$

Espaço? $O(bd)$

Solução ótima? Sim, se todos os passos tiverem o mesmo custo.

Busca em profundidade iterativa

Os nós no nível inferior (profundidade d) são gerados uma vez

Completa? Sim, se b for finito.

Tempo? $db + (d-1)b^2 + \dots + 1b^d = O(b^d)$

Espaço? $O(bd)$

Solução ótima? Sim, se todos os passos tiverem o mesmo custo.

Busca em profundidade iterativa

O nós do penúltimo nível inferior (profundidade $d-1$) são gerados duas vezes

Completa? Sim, se b é finito.

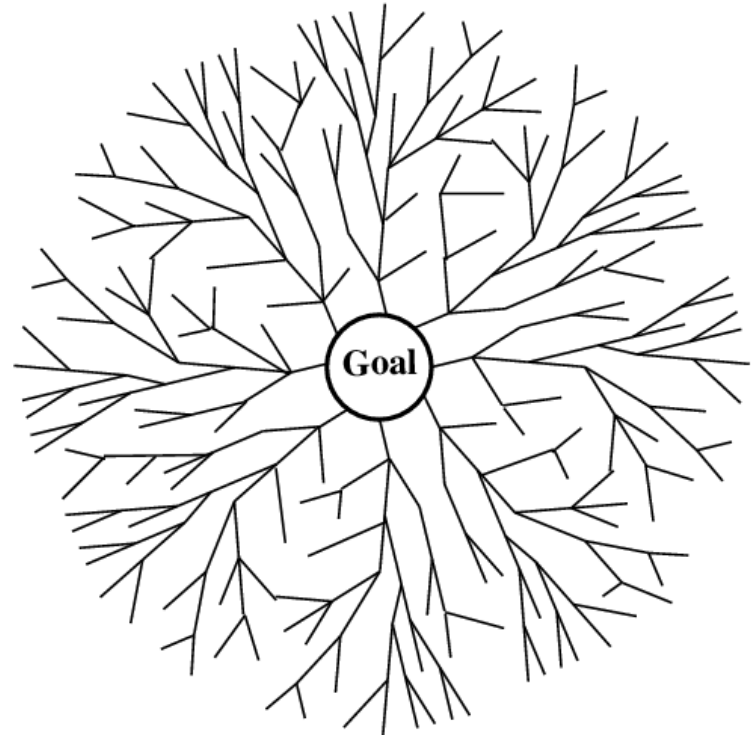
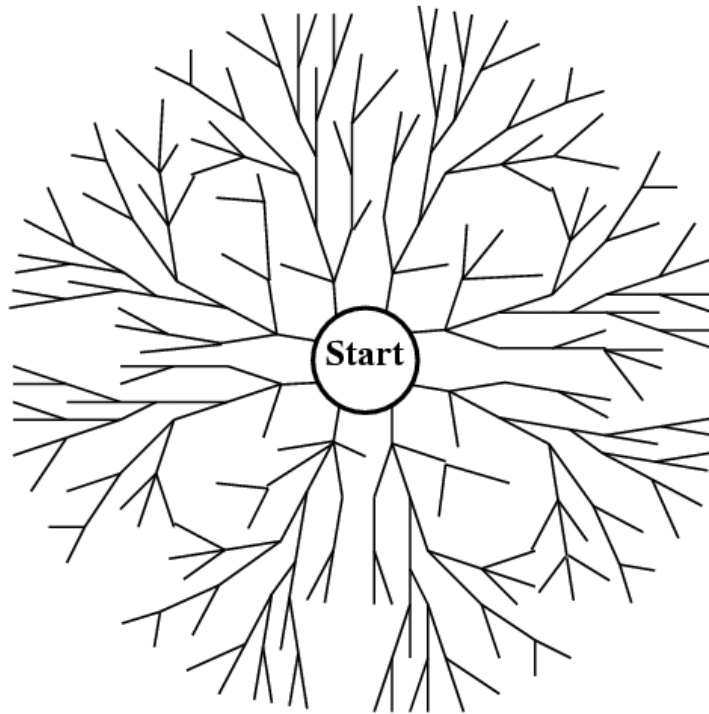
Tempo? $db + (d-1)b^2 + \dots + 1b^d = O(b^d)$

Espaço? $O(bd)$

Solução ótima? Sim, se todos os passos tiverem o mesmo custo.

Busca Bidirecional

- A busca se inicia ao mesmo tempo a partir do estado inicial e do estado final.



Busca Bidirecional

- útil se o estado inicial e o estado meta são conhecidos
 - explora duas árvores de busca simultaneamente:
 - uma com o estado inicial como raiz e
 - outra com o estado meta,
- expandindo nós em ambas as direções com a esperança de se encontrarem no meio

Propriedades da busca bidirecional

Completa? Sim, se b for finito e se ambas as direções usarem busca em largura.

Tempo? $O(b^{d/2})$

Espaço? $O(b^{d/2})$ (guarda todos os nós na memória)

Solução ótima? Sim, se o custo de todos os passos for idêntico e se ambas as direções usarem busca em largura.

Comparação dos Métodos de Busca Cega

Critério	Largura	Uniforme	Profundidade	Profundidade iterativa	Bidirecional
Completo?	Sim ¹	Sim ^{1, 2}	Não	Sim ¹	Sim ^{1, 4}
Ótimo?	Sim ³	Sim	Não	Sim ³	Sim ^{3, 4}
Tempo	$O(b^{d+1})$	$O(b^{1+(C/\alpha)})$	$O(b^m)$	$O(b^d)$	$O(b^{d/2})$

Espaço

b = fator de folhas por nó.

d = profundidade da solução mais profunda.

m = profundidade máxima da árvore.

¹ completo se b for finito.

² completo se o custo de todos os passos for positivo.

³ ótimo se o custo de todos os passos for idêntico.

⁴ se ambas as direções usarem busca em largura.

Comparação dos Métodos de Busca Cega

Critério	Largura	Uniforme	Profundidade	Profundidade iterativa	Bidirecional
Completo?	Sim ¹	Sim ^{1, 2}	Não	Sim ¹	Sim ^{1, 4}
Ótimo?	Sim ³	Sim	Não	Sim ³	Sim ^{3, 4}
Tempo	$O(b^{d+1})$	$O(b^{1+(C/\alpha)})$	$O(b^m)$	$O(b^d)$	$O(b^{d/2})$
Espaço	$O(b^{d+1})$	$O(b^{1+(C/\alpha)})$	$O(bm)$	$O(bd)$	$O(b^{d/2})$