Inteligência Artificial

Resolução de Problemas por Busca

Prof. Fabio Augusto Faria

Material adaptado de Profa. Ana Carolina Lorena e livro "Inteligência Artificial, S. Russell e P. Norving"

1º semestre 2021

Motivação

- Estratégia de Busca é uma das mais **poderosas** abordagens para resolução de problemas em IA

- Explora sistematicamente as **alternativas**

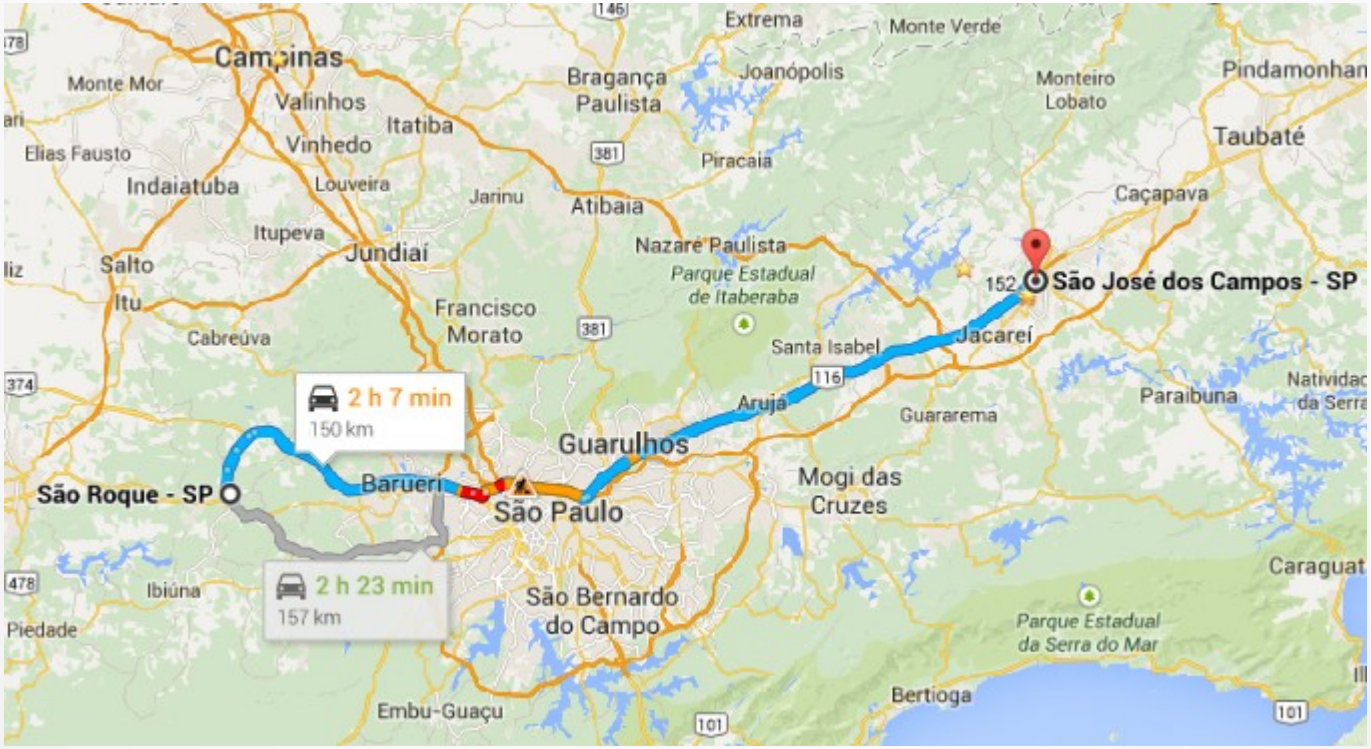
- Encontra a sequência de **passos para uma solução**

UNIFESP
Universidade Federal de São Paulo
1933

Exemplo de problema

- Viajar de SJCampos a São Roque
  - Adotando a menor rota possível

150 km – aprox. 2 horas e 7 minutos
Google maps



UNIFESP
Universidade Federal de São Paulo
1933

Resolução de problemas

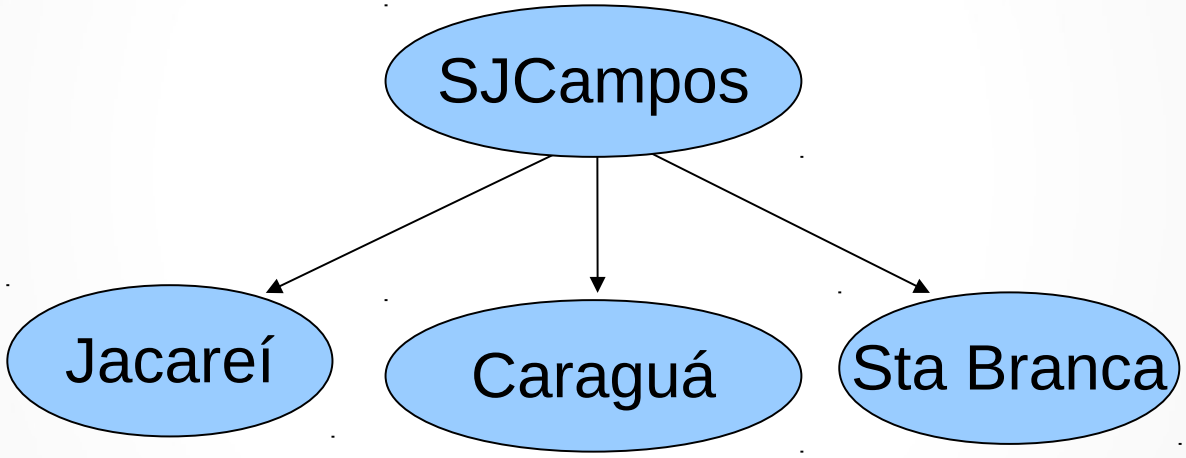
- Primeiro passo: formular objetivos
  - O que deve ser alcançado
    - Ir a São Roque
  - Baseado em situação atual
    - Estamos em São José dos Campos
  - E medida de desempenho
    - Quero seguir a **menor rota** rodoviária possível

- Objetivo pode ser visto como um estado
  - Entre possíveis estados do problema
    - Ex. estado: estar em alguma cidade

UNIFESP
Universidade Federal de São Paulo
1933

Resolução de problemas

- Podem haver estados intermediários no “caminho” da solução
  - Ex.: há três estradas saindo de São José dos Campos



- Descobrir que sequência de ações levará ao objetivo

Que cidades percorrer para chegar a São Roque?

Resolução de problemas

- Segundo passo: formular o problema
  - Processo de decidir que **ações** e **estados** devem ser considerados, dado um **objetivo**
  - No exemplo:
    - **Ações** = direções
    - **Estados** = cidades percorridas
    - **Objetivo** = chegar a SR a partir de SJC usando menor rota

Resolução de problemas

- Terceiro passo: buscar solução
  - Processo de procurar por sequência de ações que alcançam o objetivo
  - Algoritmo de busca
    - Entrada = problema
    - Saída = sequência de ações para chegar à solução

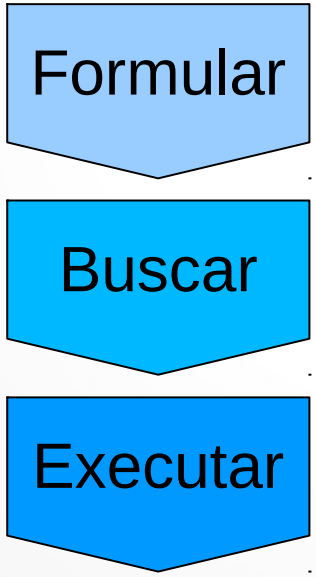
Resolução de problemas

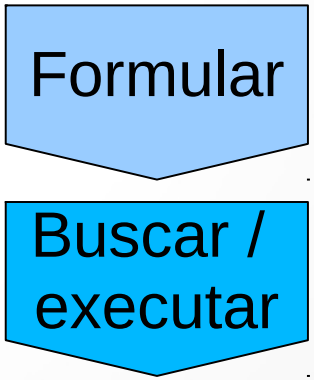
“Uma vez a solução é encontrada, as ações recomendadas podem ser executadas”

- **Execução**
  - No exemplo consiste em seguir a rota de cidades recomendada
  - Execução pode ser intercalada com a busca
    - Ex.: robô percebendo o ambiente e então navegando

Resolução de problemas

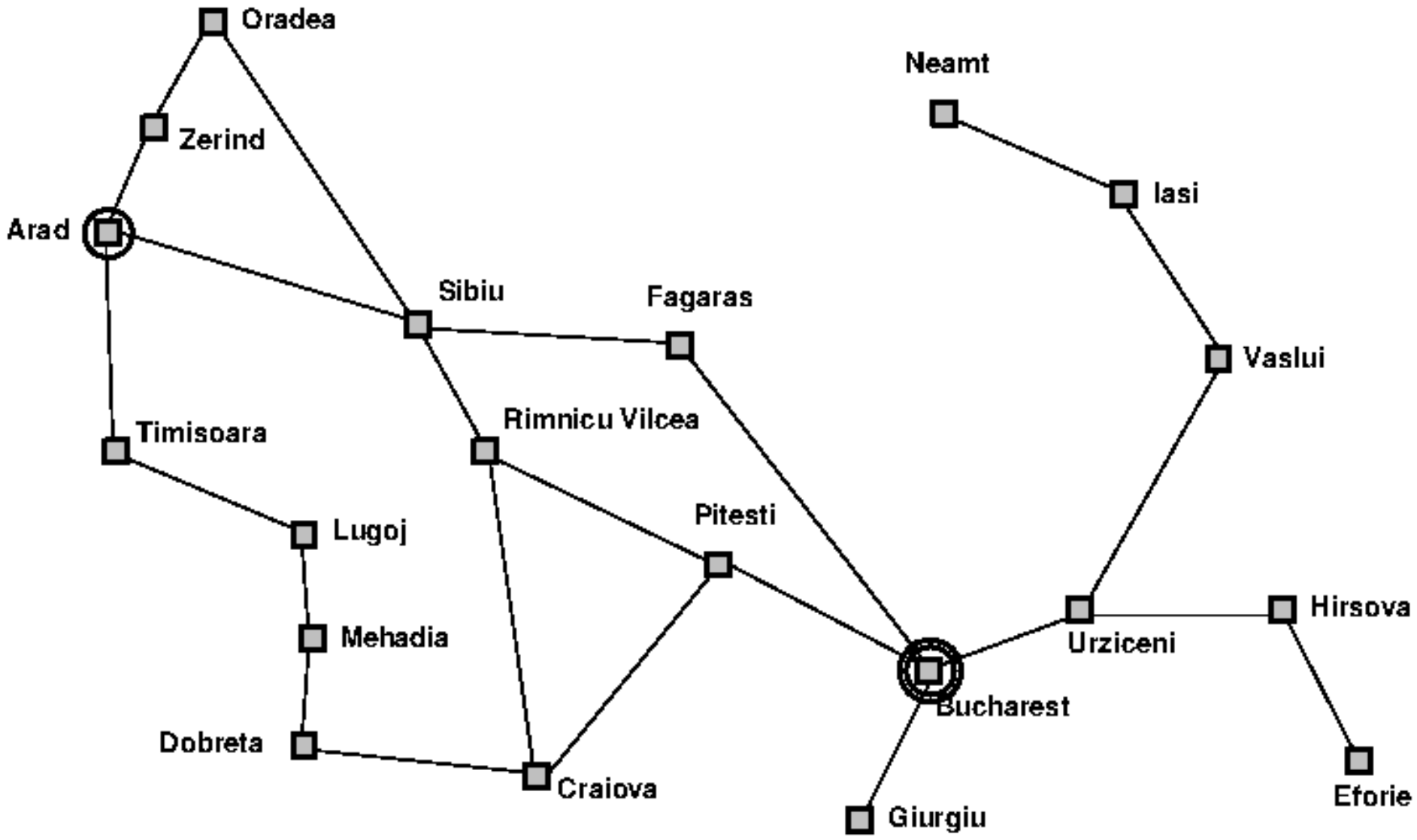
- Três etapas principais:
  - Formulação do problema
  - Busca de soluções
  - Execução da solução





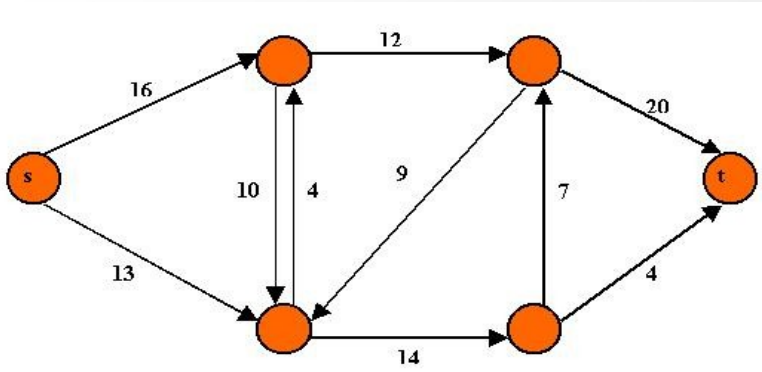
Outro exemplo

- Para o problema da Romênia



Problema

- Definido por cinco componetes:
  - *1)* ***Estado inicial***
    - Ex.: Em(Arad)
  - *2)* ***Ações/operadores possíveis para gerar novo estado***
    - Ex.: Em(Arad) = {Ir(Sibiu), Ir(Timisoara), Ir(Zerind)}
    - Estado inicial e operadores definem espaço de estados
      - Conjunto de todos os estados acessíveis a partir do estado inicial
      - Forma um grafo

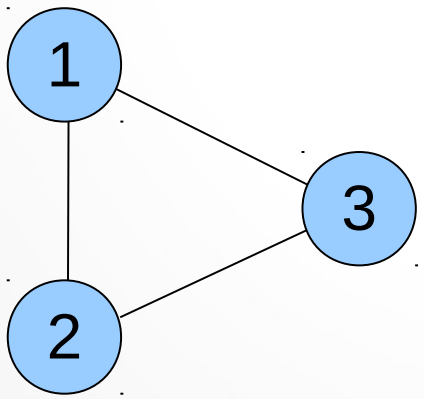


Problema

- Cinco componentes:
  - 3) **Modelo de transição**
    - Descrição do que cada ação faz.
    - Ex. RESULTADO(Em(ARAD),Ir(ZERIND)) = Em(ZERIND);
  - 4) **Teste de objetivo**
    - Determinar se um dado estado é um estado objetivo
    - No ex., o objetivo é o conjunto unitário {Em(Bucareste)}
  - **5) Uma função de custo de caminho**
    - Atribui custo numérico a cada caminho no grafo
    - Caminho é uma sequência de estados conectados
    - Geralmente reflete medida de desempenho
      - No ex., menor caminho em km

Grafo

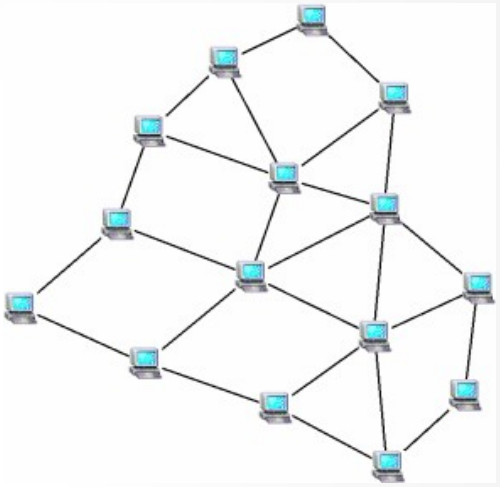
- Representação gráfica das relações entre elementos de dados
  - G = {V,A}
    - Conjunto V de vértices
    - Conjunto A de arestas



V = {1, 2, 3}
A = {(1,2),(1,3),(2,3)}

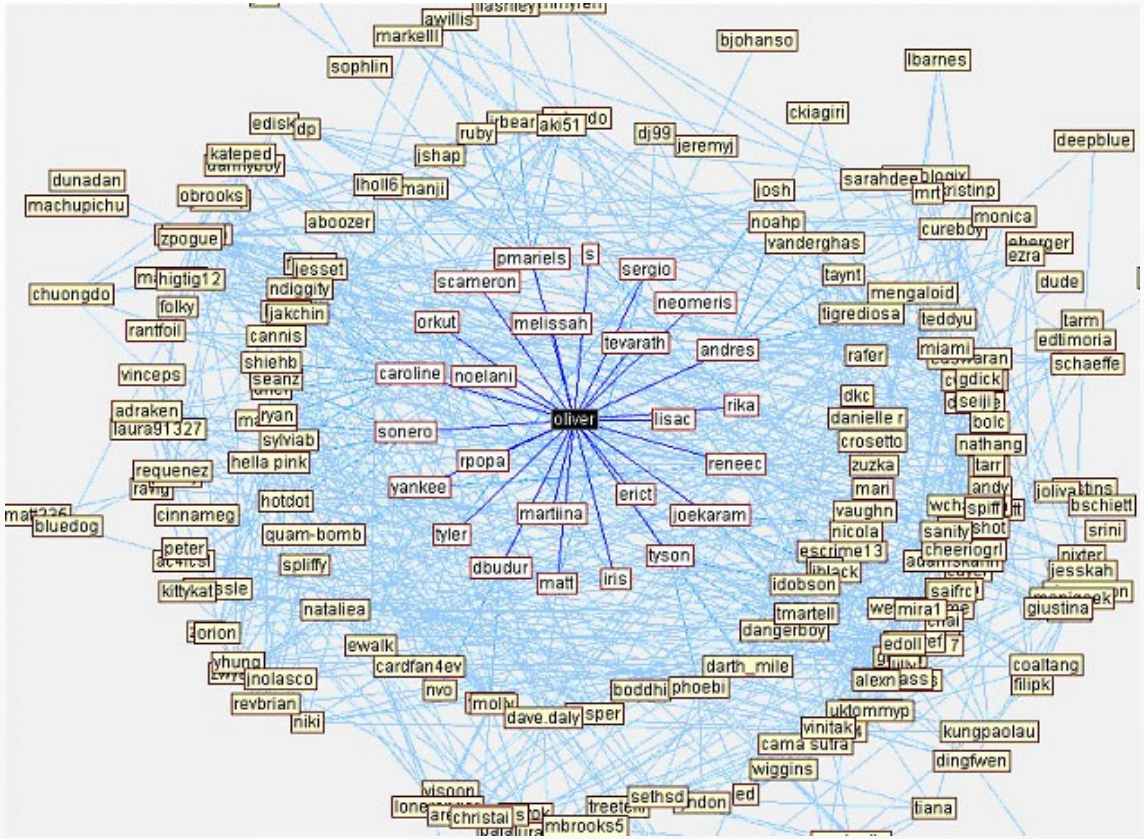
Exemplos

- Rede de computadores
  - Vértices = terminais
  - Arestas = cabos de rede
    - Custo = latência
      - latência é o tempo que uma unidade de informação leva pra transitar pela rede de um ponto a outro



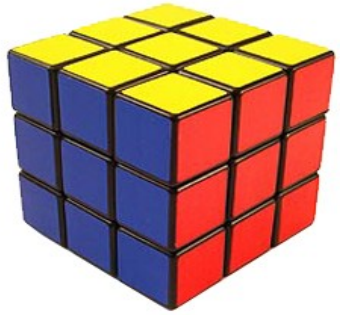
Exemplo

- Rede social
  - Vértices = pessoas
  - Arestas = relação social entre pessoas



Solução

- Uma solução para um problema é um caminho desde o estado inicial até o estado objetivo
  - Solução ótima tem menor custo de caminho entre todas as possíveis soluções



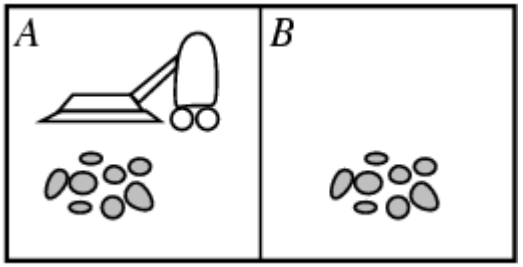
Como formular um problema?

Como escolher estados e ações?

- Abstrair detalhes irrelevantes
  - na formulação do problema (ações e estados)
  - nas funções de custo de caminho e de busca
- Exemplo: dirigir de São José dos Campos a São Roque
  - Não é de interesse:
    - número de passageiros
    - o que toca no rádio (estado)
    - cidades sem acesso rodoviário (estado)
    - o que se come e bebe dentro do carro (ações)
    - número de postos policiais no caminho (custo de caminho)
    - número de operações de adição (custo de busca)
- É tarefa do projetista fazer as boas escolhas

Exemplo problema 1

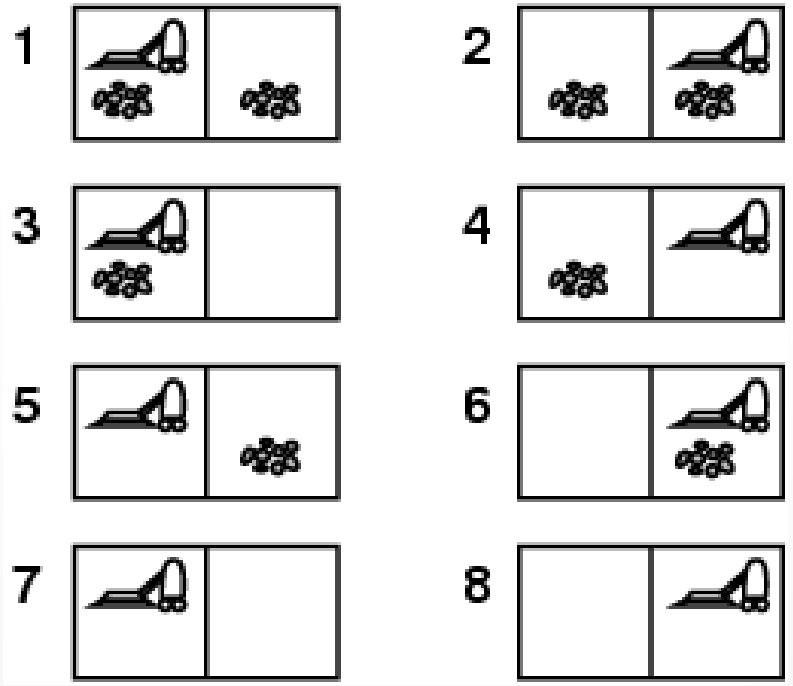
- Aspirador de pó
  - Percorre quadrados e vê se tem sujeira a limpar



  - Operações possíveis:
    - Mover para a direita
    - Mover para a esquerda
    - Aspirar pó
    - Não fazer nada

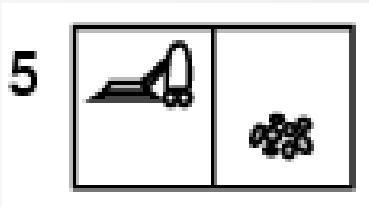
Aspirador de pó

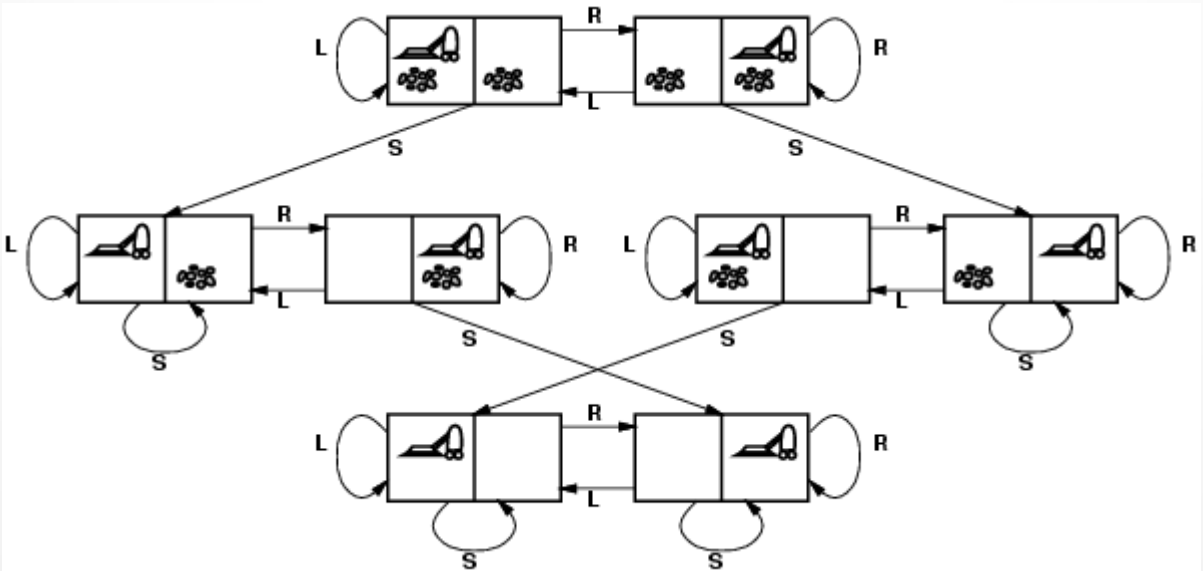
- Formulação do problema:
  - *Estados*:
    - 2 quadrados, contendo ou não sujeira
    - 8 estados possíveis



Aspirador de pó

- Formulação do problema:
  - *Estado inicial*:
    - Qualquer um dos estados possíveis
    - Ex.: estado 5
  - *Ações*: 3 (R, L e S)
  - Modelos de transição: São as setas da fig. 3.3
    - Todas ações têm um efeito esperado exceto as falhas



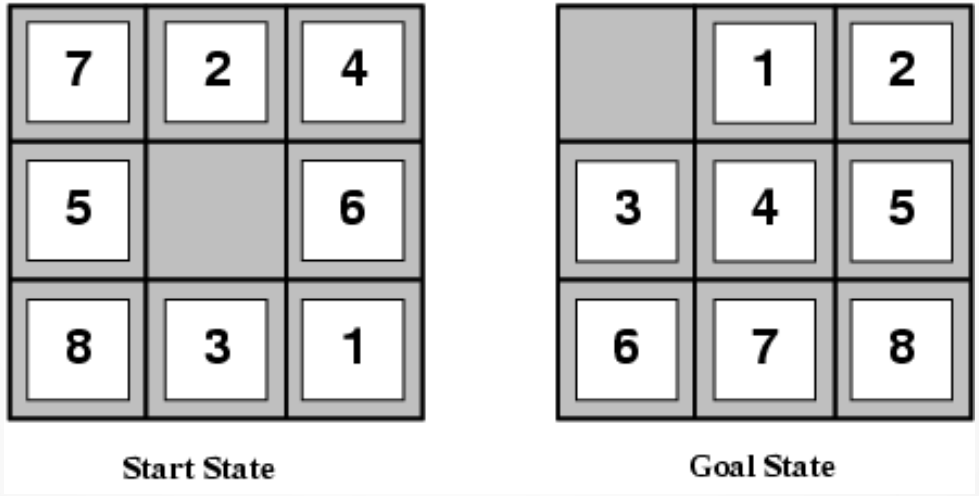


Aspirador de pó

- Formulação do problema:
  - *Teste de objetivo*:
    - Verificar se quadrados estão limpos
  - *Custo de caminho*:
    - Cada passo custa 1
    - Custo do caminho é o número de passos do caminho

Exemplo problema 2

- Quebra-cabeça de 8 peças
  - Tabuleiro 3x3 com 8 peças numeradas e um espaço vazio
  - Uma peça pode deslizar para o espaço
  - Exemplo:



Quebra-cabeça

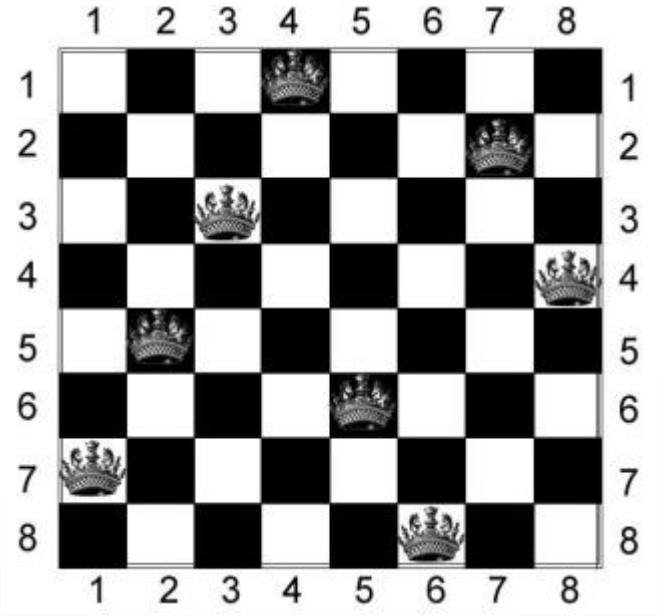
- Formulação do problema: Um estado especifica a posição de cada peça e do espaço vazio nos 9 quadrados.
  - *Estado inicial*:
    - Qualquer estado
  - Ações:
    - Espaço vazio se desloca para esquerda, direita, cima ou baixo
  - Modelo de Transição:
    - Dado um estado e ação, ele devolve estado resultante (move 5 p/ direita)
  - Teste de Objetivo:
    - O estado é o final?
  - Custo do Caminho
    - Cada passo custa 1 e custo do caminho = # de passos no caminho

Quebra-cabeça

- Problema dos quebra-cabeças deslizantes
  - NP-completo
    - Ainda não existem algoritmos determinísticos que encontram a solução em tempo polinomiais
      - 8 peças: 181440 estados possíveis
      - 15 peças: 1,3 trilhão de estados
      - 24 peças: 10^{25} estados

Exemplo problema 3

- Problema das 8 rainhas
  - Posicionar 8 rainhas em um tabuleiro de xadrez de forma que nenhuma rainha ataque outra

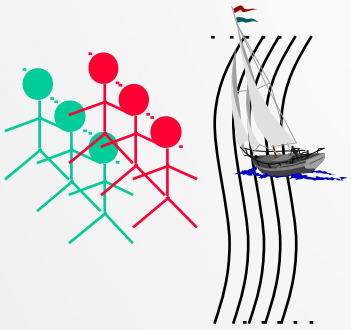


8 rainhas

- Formulação incremental: adiciona rainhas ao tabuleiro.
  - *Estados*: Qualquer disposição de 0 a 8 rainhas no tabuleiro. Há $3x10^{14}$ possíveis estados.
  - *Estado inicial*:
    - Nenhuma rainha no tabuleiro
  - *Ações*:
    - Colocar uma rainha qualquer em um espaço vazio
  - *Teste de objetivo*:
    - 8 rainhas no tabuleiro e nenhuma é atacada
  - *Custo*:
    - Não é de interesse, pois apenas o estado final é importante

Outros problemas

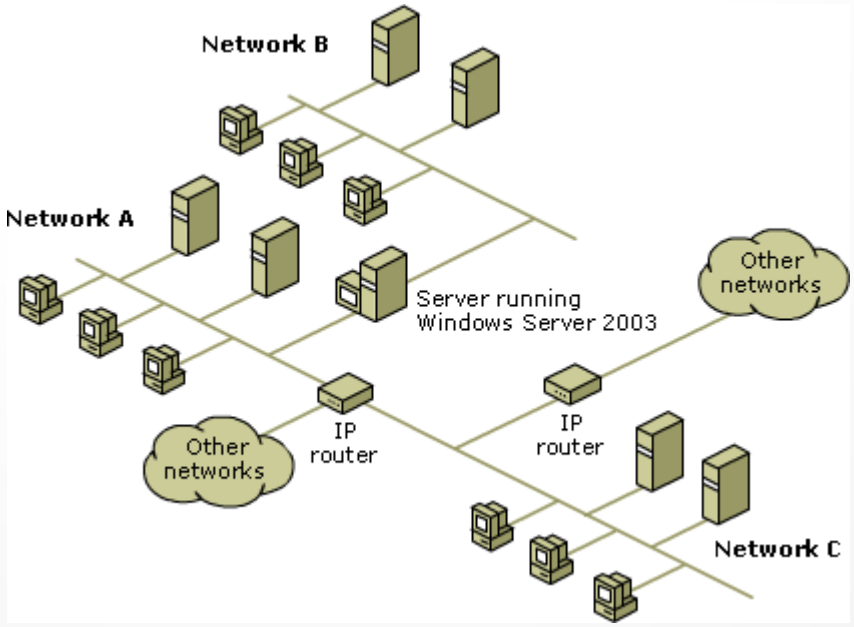
- Torre de Hanoi
- Palavras cruzadas
- Canibais e missionários



Três missionários e três canibais vão atravessar de uma margem para a outra de um rio, usando um barco onde só cabem duas pessoas de cada vez. Os missionários têm que ser cautelosos para que os canibais nunca os excedam em número em nenhuma das margens do rio. Como poderão passar todos para a outra margem, usando apenas aquele barco?

Problemas reais

- Problema de roteamento
  - Exemplos:
    - Roteamento de pacotes em rede de computadores
    - Planejamento de operações militares
    - Sistema de planejamento de viagens aéreas

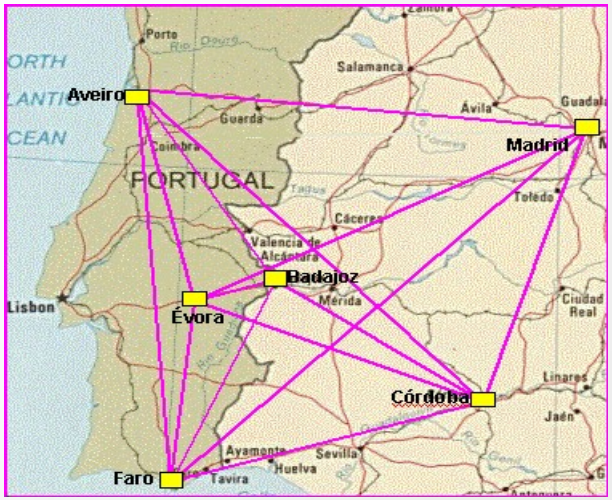


Ex.: viagens aéreas

- Formulação do problema:
  - *Estados*: cada estado é uma posição (aeroporto) e hora atual.
  - *Estado inicial*:
    - Especificado pelo problema
  - *Ações*:
    - Retorna estado após tomar algum vôo programado
  - *Modelo de Transição*:
    - Estado resultante terá o destino e horário de chegada
  - *Teste de objetivo*:
    - Estar no destino após um tempo especificado
  - *Custo de caminho*:
    - Pode ser tempo da viagem, por exemplo

Problemas reais

- Problema do caixeiro-viajante
  - Visitar um conjunto de cidades uma única vez usando o percurso mais curto
  - Exemplo:
    - Planejar viagens
    - Planejar movimentos de máquinas
      - Perfuração de placas de circuito
      - Industriais



Problemas reais

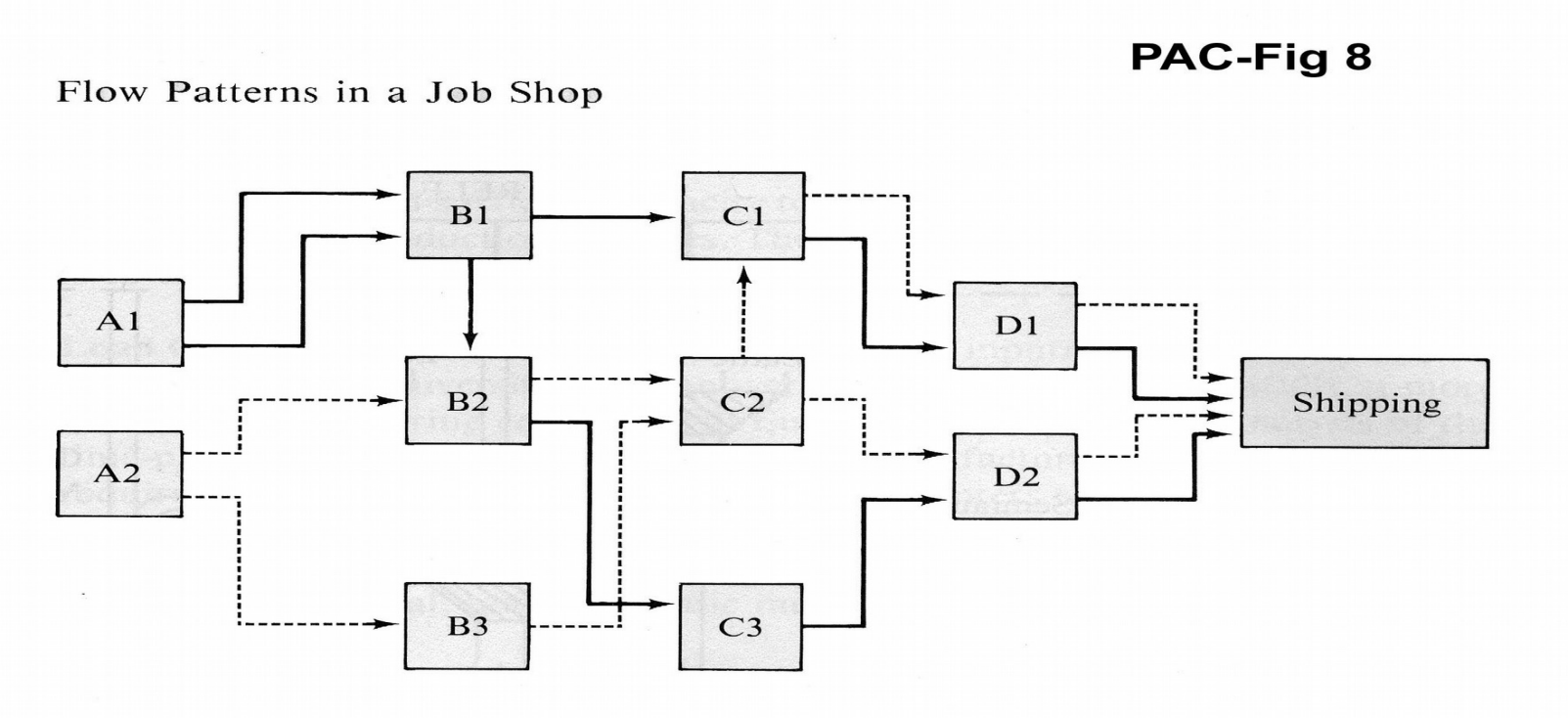
- Layout de VLSI
  - Posicionamento de componentes e conexões em chip minimizando
    - Área
    - Retardos de circuitos
    - Capacitâncias de fuga
  - E maximizando
    - Rendimento industrial

Problemas reais

- Navegação de robôs
  - Generalização do problema de roteamento
  - Lidar também com erros em sensores e controles

Problemas reais

- Problemas de alocação (*Scheduling*)
  - Salas de aula
  - Máquinas industriais (*job shop*)

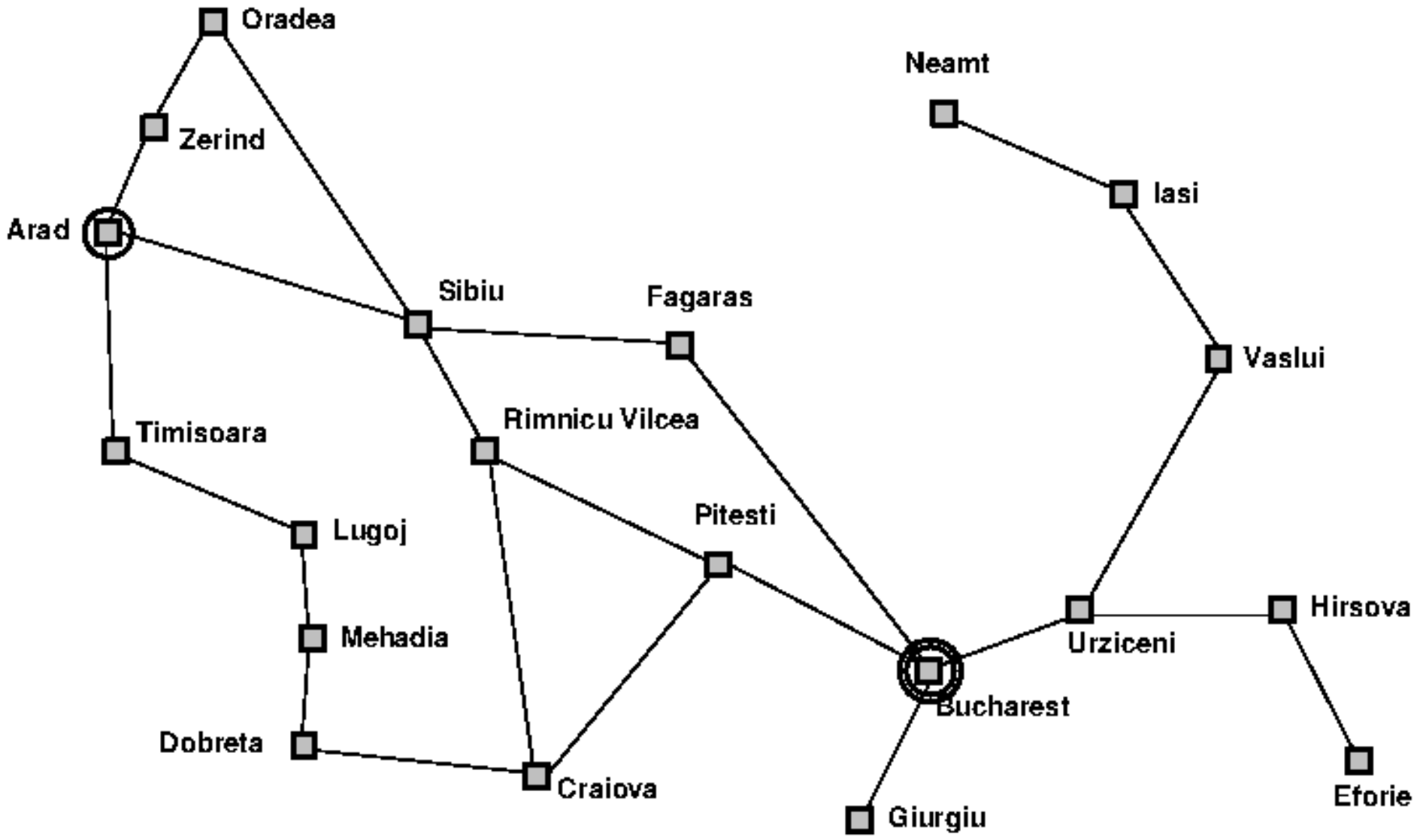


http://www.mcts.com/images/cgi/PAC-FIG8.JPG

Busca por soluções

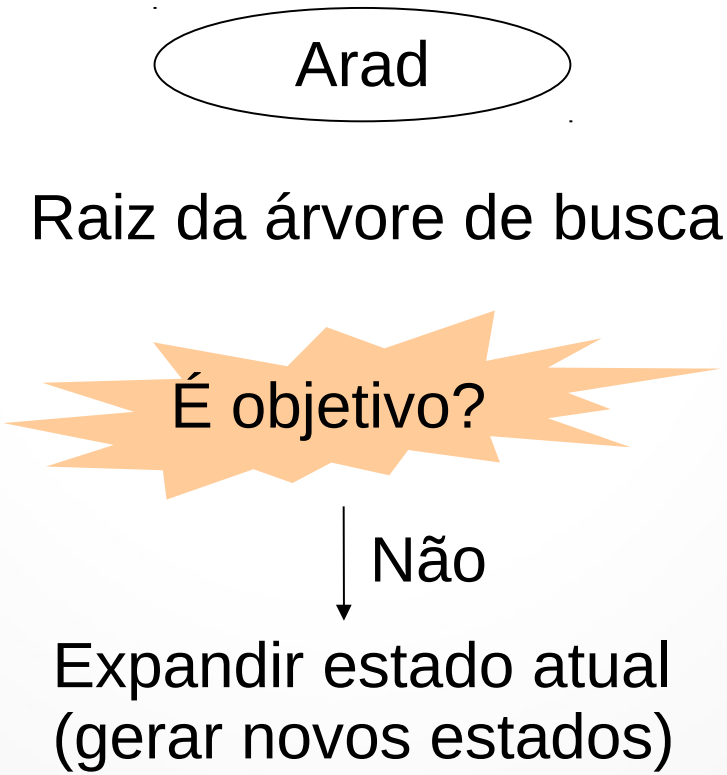
- Resolução dos problemas após formulação
  - Por meio de busca no espaço de estados
    - Sequencia de ações possíveis formam **árvore de busca**:
      - Com estado inicial na raiz
      - Ramos são as ações
      - Nós são os estados no espaço de estados do problema.

Exemplo


Oradea
Neamt
Zerind
Iasi
Arad
Sibiu
Fagaras
Vaslui
Timisoara
Rimnicu Vilcea
Pitesti
Lugoj
Hirsova
Mehadia
Urziceni
Bucharest
Dobreta
Craiova
Eforie
Giurgiu

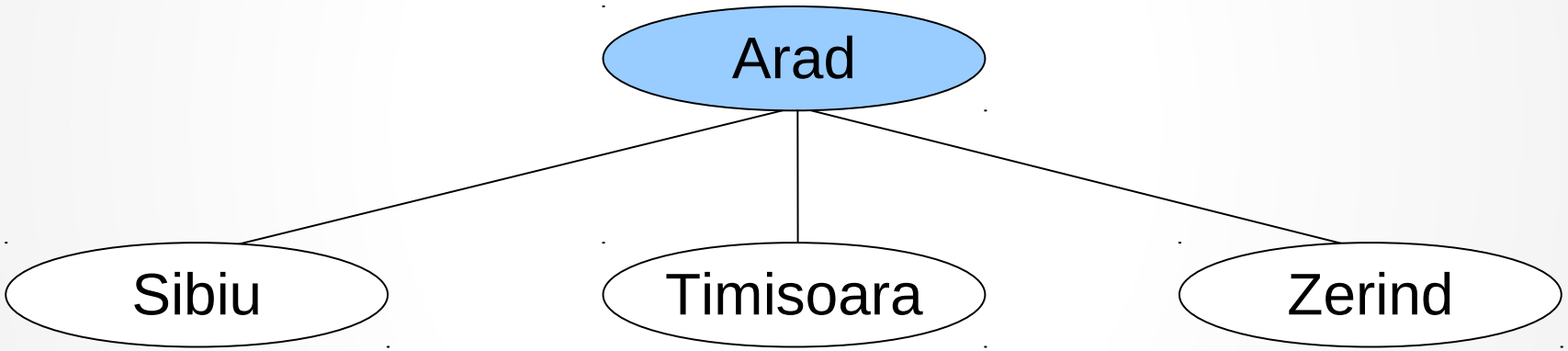
Exemplo

- (a) Estado inicial



Exemplo

- (b) Expandindo Arad

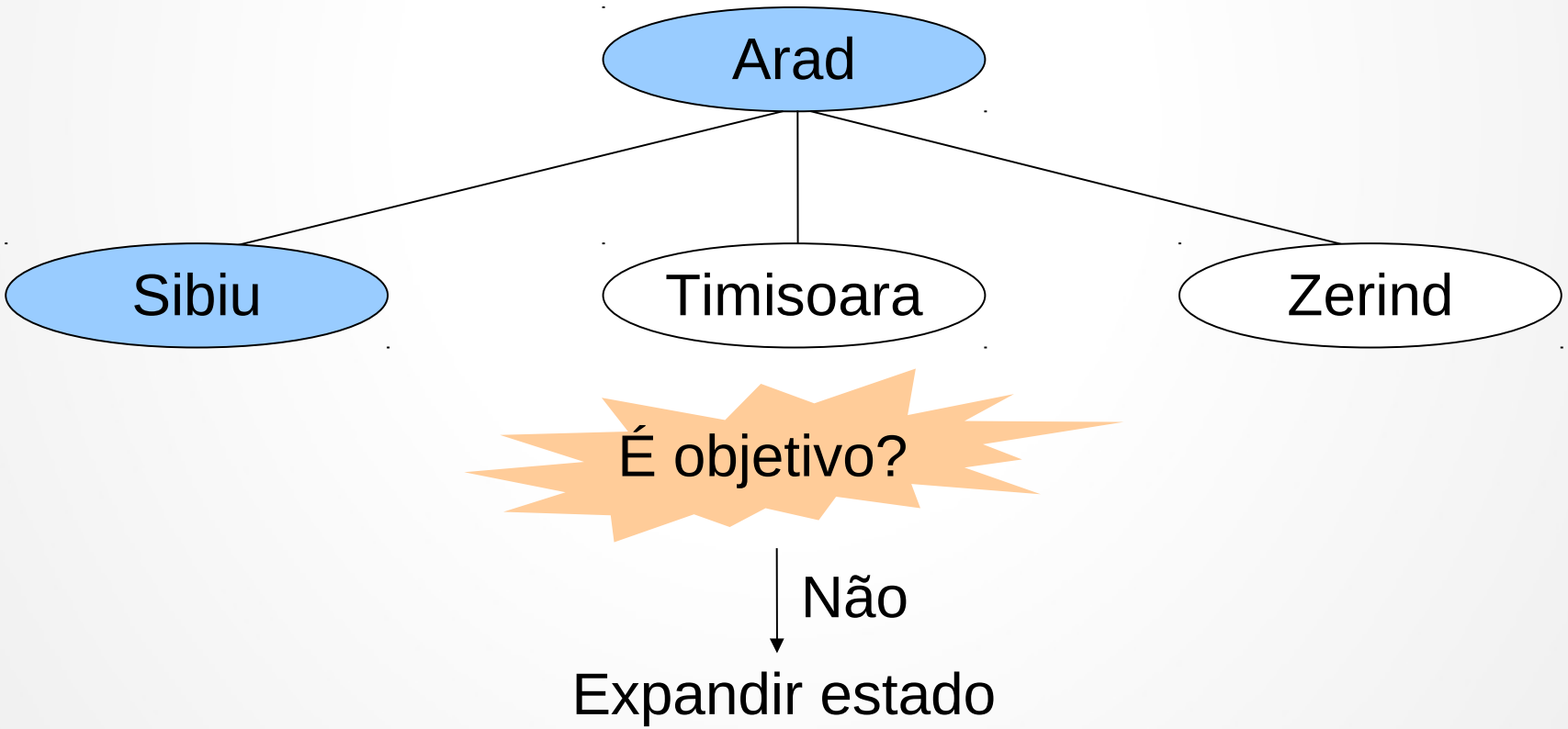


Qual escolher para futuras considerações?

Essência da busca: seguir uma opção e deixar as outras reservadas para mais tarde, no caso da primeira não levar a uma solução

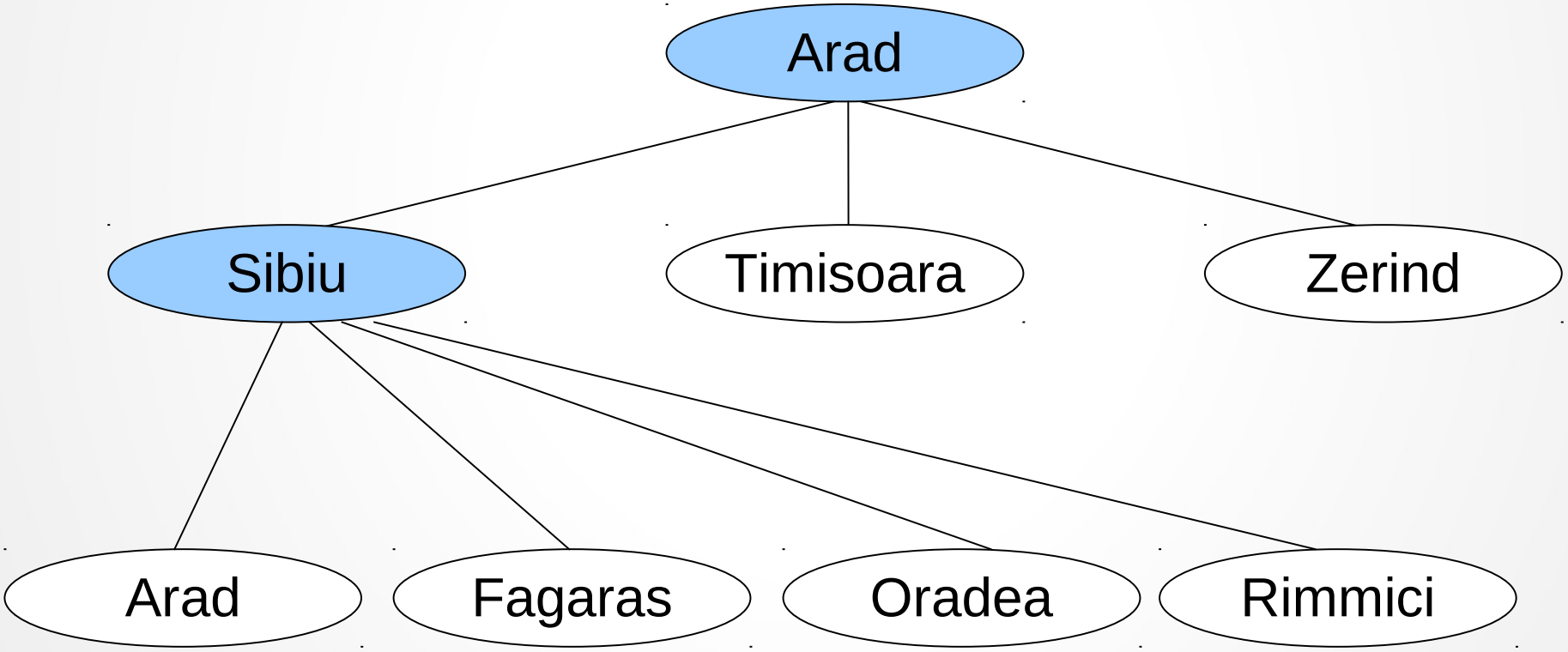
Exemplo

- (c) Supor escolha de Sibiu



Exemplo

- (c) Expandindo Sibiu



Pode escolher entre quaisquer estados ainda não visitados

Busca

- Continua-se a escolher, testar e expandir até
  - Encontrar solução ou
  - Não existirem mais estados a serem visitados
- A estratégia de escolha de estado a visitar e expandir é determinada pela **estratégia de busca**

Algoritmo

- Algoritmo geral de busca em árvore:

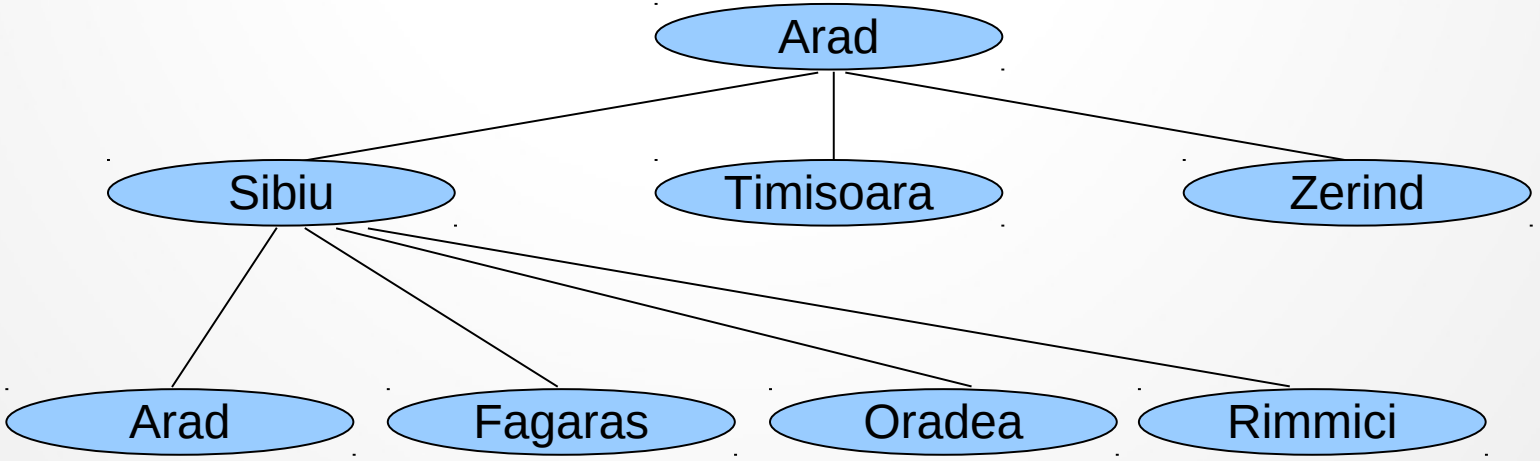
```
função BUSCA-EM-ÁRVORE(problema, estratégia) retorna solução
ou falha
    iniciar a árvore de busca com o estado inicial
    repita
        se não existe nenhum candidato para expansão então
            retornar falha
        escolher nó para expansão de acordo com estratégia
        se o nó contém um estado objetivo então
            retornar a solução
        senão
            expandir o nó
            adicionar os nós resultantes à árvore de busca
```

Árvore de busca

- Os nós da árvore podem guardar mais informação do que apenas o estado: estrutura de dados com pelo menos 5 componentes:
  1. O estado correspondente
  2. O seu nó pai
  3. O operador aplicado para gerar o nó (a partir do pai)
  4. A profundidade do nó
  5. O custo do nó (desde a raiz)
  6. Os nós-filhos

Árvore de Busca

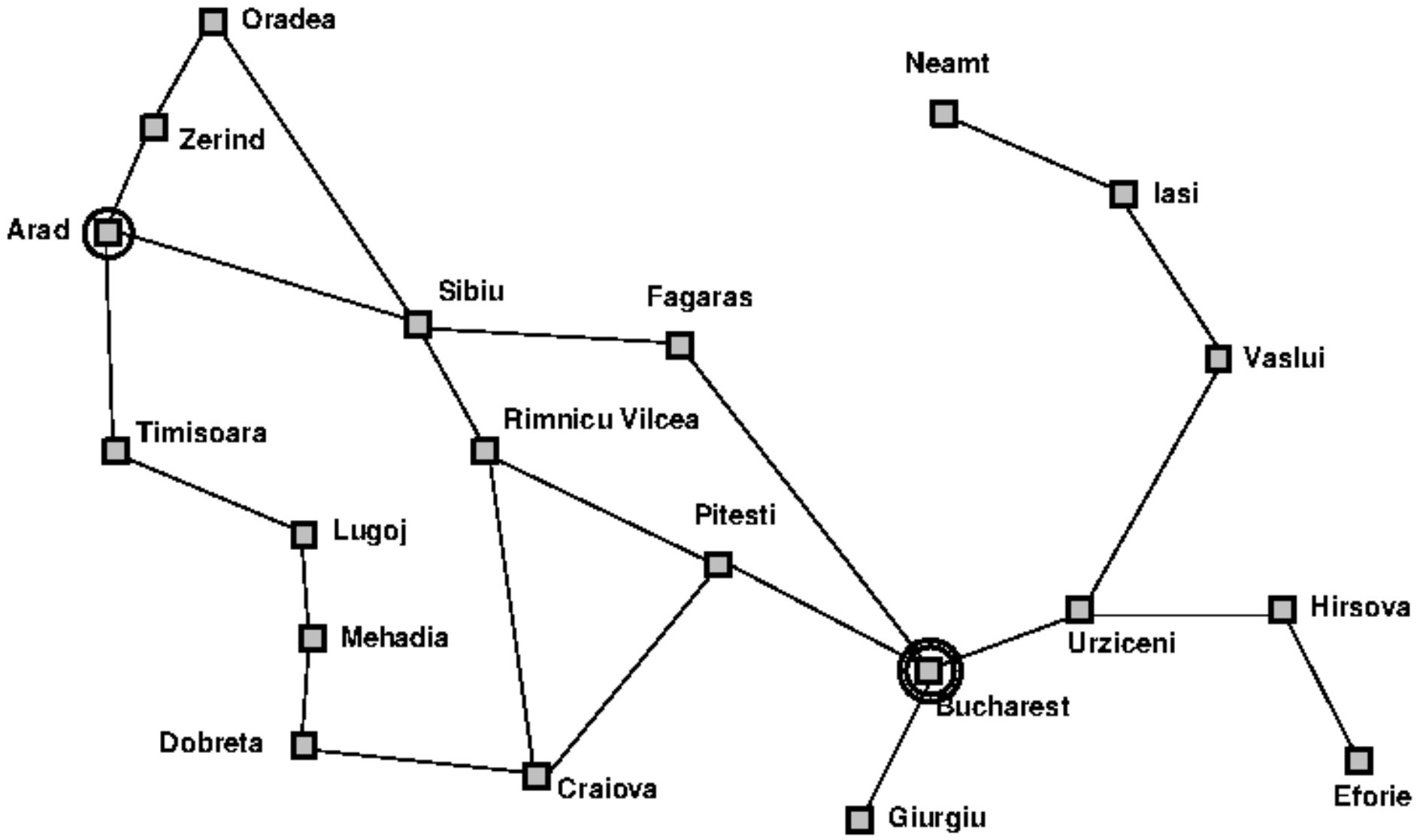
- Exemplo:
  - Nó Sibiu
    - Pai = Arad
    - Operador = ir de Arad a Sibiu
    - Profundidade = 1
    - Custo = x km
    - Filhos = {Arad, Fagaras, Oradea, Rimmici}



Exercício 1

- Considere o problema de encontrar uma rota de Arad a Bucareste (mapa no próximo slide). Responda:
  - Que rota você pegaria e por quê?
  - Descreva o raciocínio você usou para encontrar essa rota.

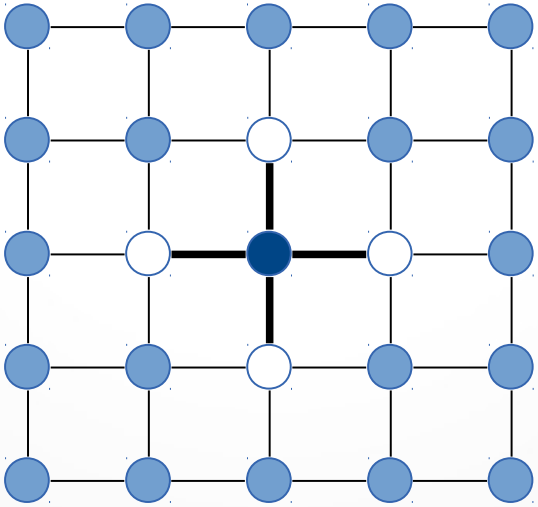
Exercício 1


Oradea
Neamt
Zerind
Iasi
Arad
Sibiu
Fagaras
Vaslui
Timisoara
Rimnicu Vilcea
Pitesti
Lugoj
Hirsova
Mehadia
Urziceni
Bucharest
Dobreta
Craiova
Eforie
Giurgiu

Exercício 2

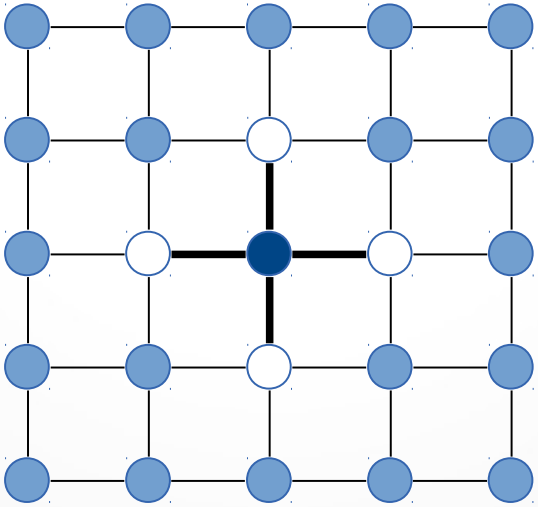
Problema da grade retangular: buscar uma rota entre dois pontos de uma grade retangular

- Muito usado em jogos
- Na grade, cada estado tem quatro sucessores (norte, sul, leste, oeste)



Exercício 2

Problema da grade retangular: formule esse problema considerando os quatro aspectos que devem ser especificados para a aplicação de algoritmos de busca



Inteligência Artificial

Busca sem informação - Estratégias

Prof. Fabio Augusto Faria

Material adaptado de Profa. Ana Carolina Lorena e livro "Inteligência Artificial, S. Russell e P. Norving"

1º semestre 2021

Busca

- Passos básicos:

  1- Escolhe estado,

  2- Testa se é objetivo

  3- Se não for, expande (gera sucessores) e retorna a 1

  até

  - Encontrar solução ou
  - Não existirem mais estados a serem visitados

- Escolha de estado a visitar e expandir é determinada pela estratégia de busca

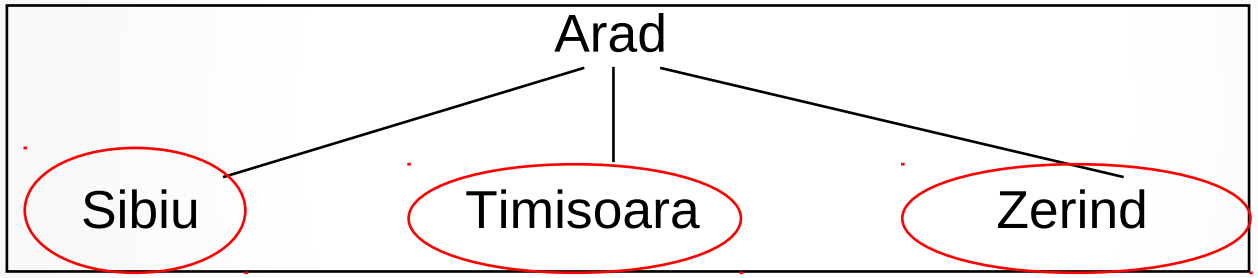
Fronteira ou Borda

Fronteira **do espaço de estados**

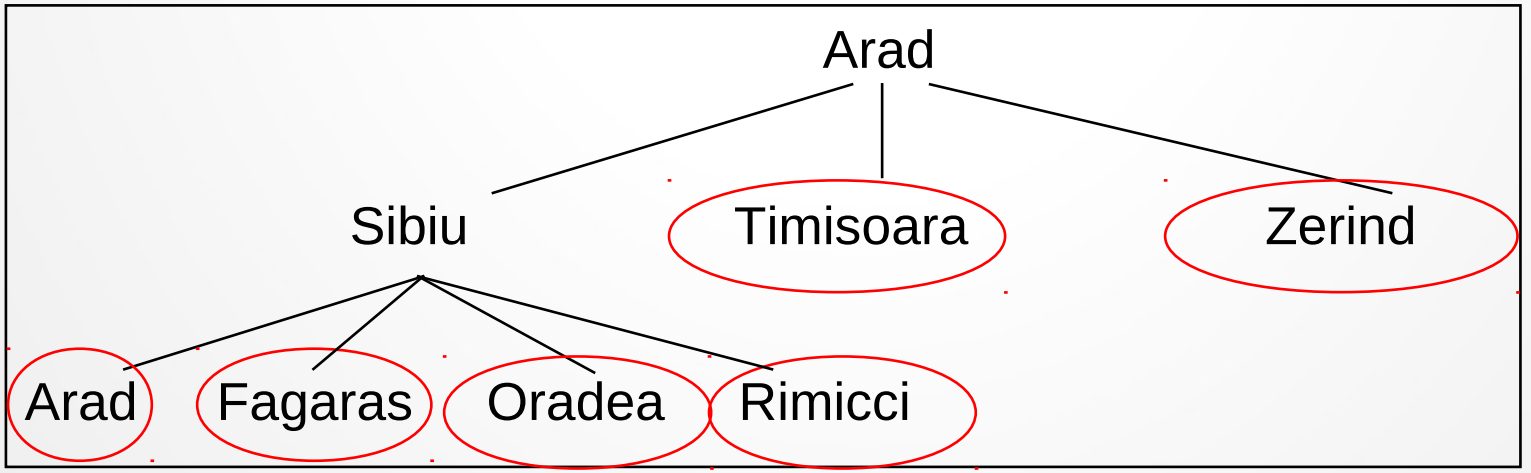
nós (estados) disponíveis para serem expandidos no momento

estado inicial =>

fronteira



fronteira



fronteira

Fronteira

Algoritmo genérico de busca usando fronteira:

(começa com a fronteira contendo o estado inicial do problema)

1. Selecionar (e remover) o primeiro nó (estado) da *fronteira* do espaço de estados;
   - se a fronteira está vazia, o algoritmo termina com falha.
2. Testar se o nó é um estado final (objetivo):
   - se “sim”, então retornar nó - a busca termina com sucesso.
3. Gerar um novo conjunto de estados pela aplicação dos operadores ao nó selecionado;
4. Inserir os nós gerados na *fronteira*, de acordo com a estratégia de busca usada, e voltar para o passo (1).

Estratégias de busca

- Busca sem informação ou cega
  - Informação sobre os estados são somente aquelas fornecidas na definição do problema
    - Somente geram sucessores e distinguem se um estado é objetivo ou não

- Estratégias que sabem se um estado é mais promissor que outro são estratégias de busca com informação ou busca heurística

- Diferentes estratégias se distinguem pela ordem em que nós são expandidos

Algoritmo

Algoritmo mais detalhado:

Função Inserir: controla a ordem de inserção de nós na fronteira do espaço de estados (de acordo com estratégia de busca)

função Busca-Genérica (*problema*, Função-Insere)

(retorna uma solução ou falha)
fronteira ← Inserir (Nó (Estado-Inicial [*problema*]))
loop do
se *fronteira* está vazia **então retorna** falha
nó ← Remove-Primeiro (*fronteira*)
se Teste-Término [*problema*] aplicado a Estado [*nó*] tiver sucesso
então retorna *nó*
fronteira ← Inserir(*fronteira*, Expandir[*problema*, *nó*])
end

Estratégias de busca sem informação

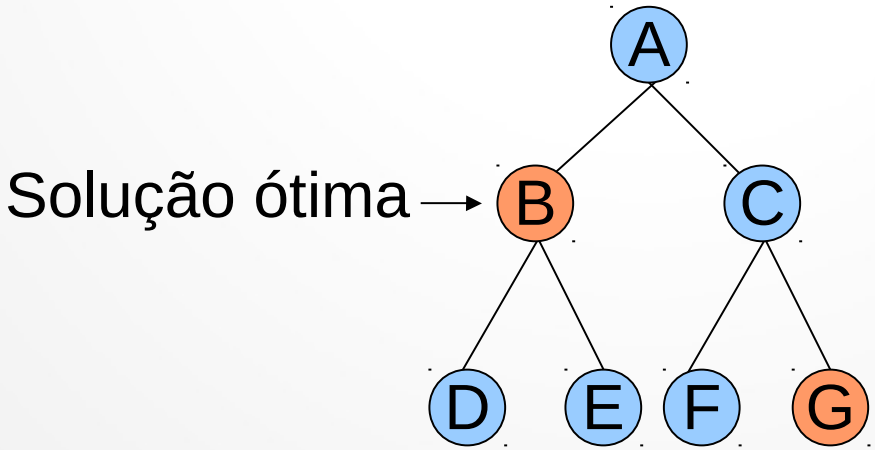
- Estratégias para determinar a ordem de ramificação dos nós:
  1. Busca em largura ou extensão
  2. Busca de custo uniforme
  3. Busca em profundidade
  4. Busca com aprofundamento iterativo
- Direção da ramificação:
  1. Do estado inicial para um estado final
  2. De um estado final para o estado inicial
  3. Busca bi-direcional

Desempenho 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)

Considerações sobre desempenho

- Completeza
  - O algoritmo garante encontrar solução quando ela existe?
- Otimização
  - A estratégia encontra a solução ótima?
    - Para passos com igual custo, é aquela em menor profundidade na árvore de busca



Considerações sobre desempenho

- Complexidade de tempo
  - Quanto tempo o algoritmo leva para encontrar uma solução?
- Complexidade de espaço
  - Quanta memória é necessária para executar a busca?

Análises em função de:

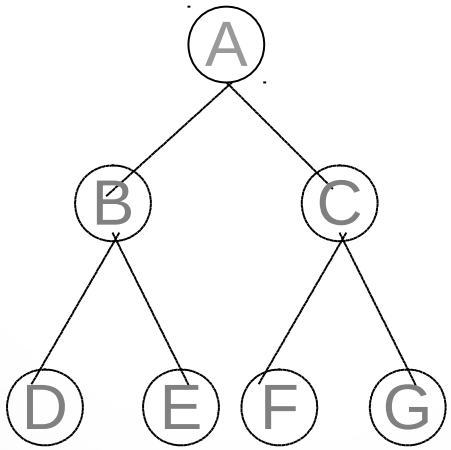
b: fator de ramificação da árvore
d: profundidade da solução mais rasa
m: profundidade máxima da árvore de busca

Busca em largura

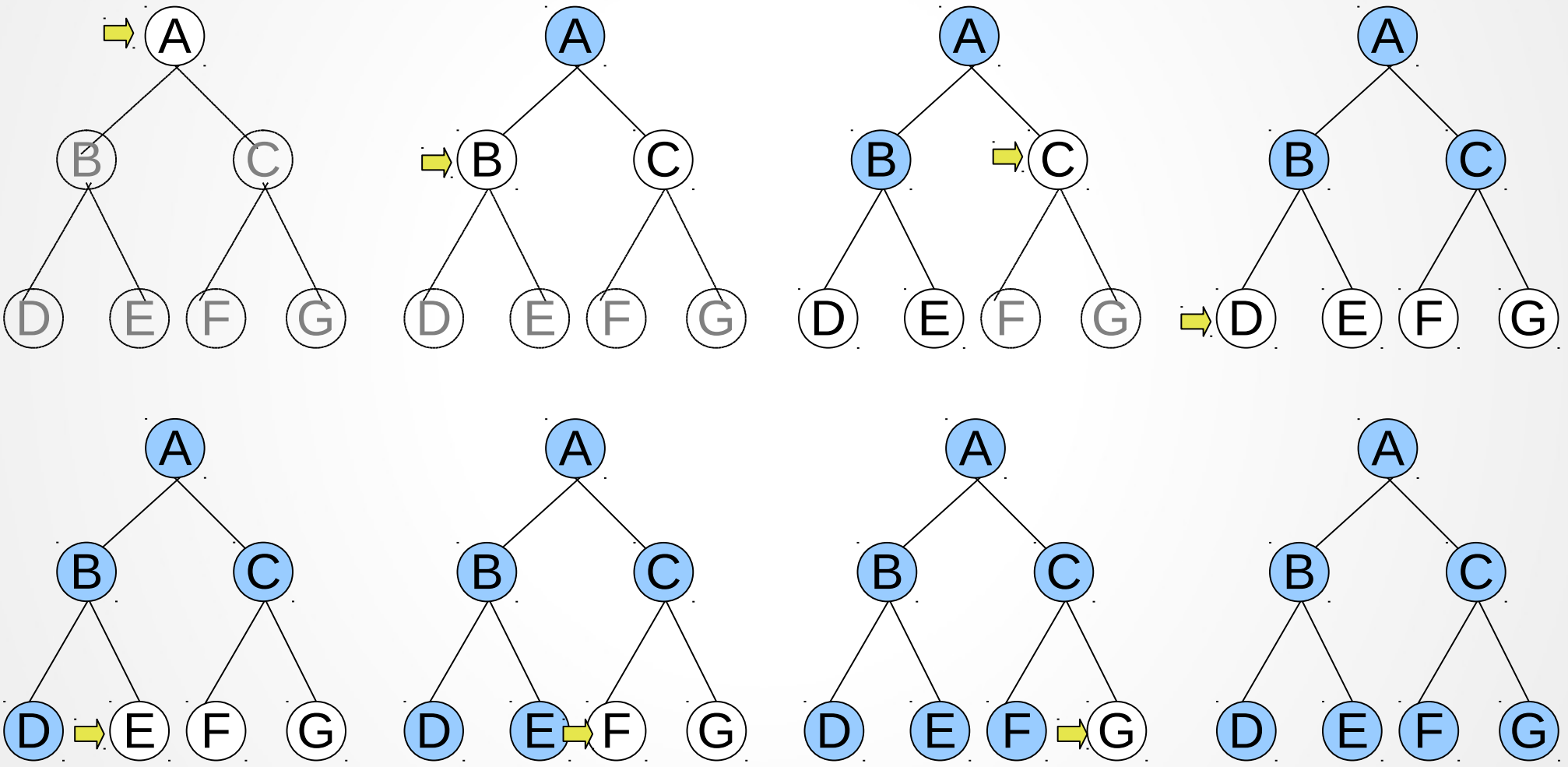
- Nó raiz é expandido primeiro, depois todos os seus sucessores, depois os sucessores deles e assim por diante
  - Busca em extensão
  - Todos os nós em um nível da árvore de busca são expandidos antes dos nós do nível seguinte

Busca em largura

- Ordem de expansão dos nós:
  1. Nó raiz
  2. Todos os nós de profundidade 1
  3. Todos os nós de profundidade 2, etc…



Busca em largura

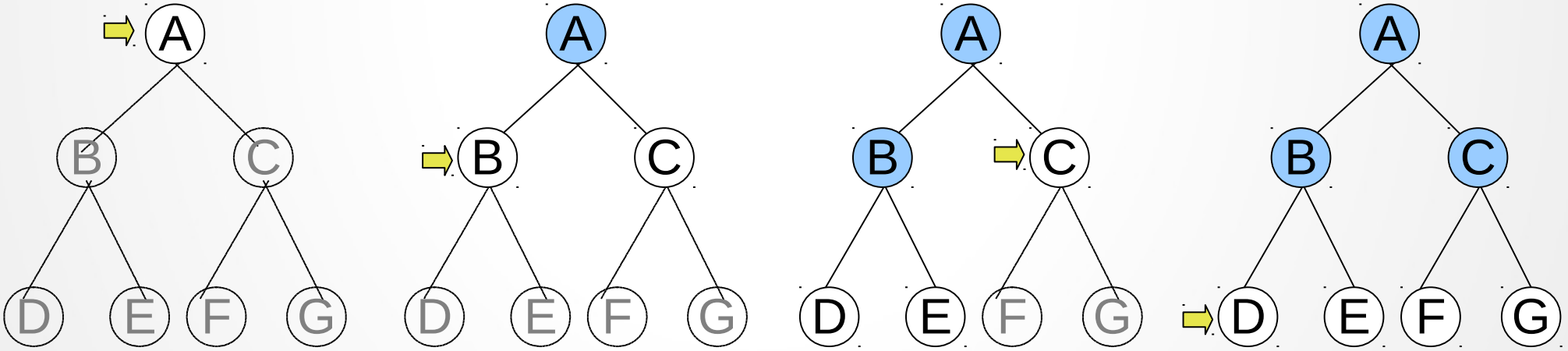


A B C D E F G

Busca em largura

- Fronteira pode ser vista como fila
  - Novos sucessores são colocados no final
  - O primeiro da fila é selecionado a cada passo

Árvore de busca:



Fila:

A

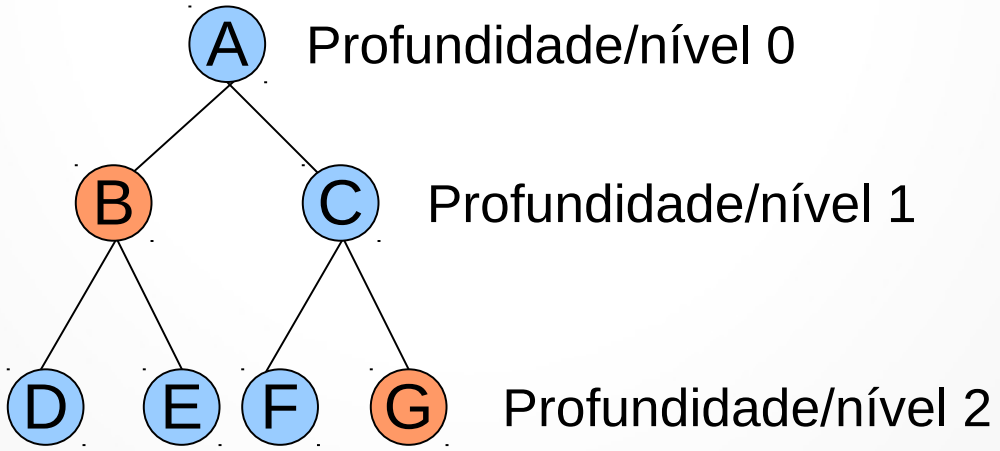
B C

C D E

D E F G

Desempenho

- Exemplo:
  - b = fator de ramificação = 2
  - d = profundidade da solução mais rasa = 1
  - m = profundidade máxima da árvore de busca = 2



Busca em largura

- É *completa*
  - Quando fator de ramificação é finito

- É *ótima* se custo de caminho cresce com a profundidade do nó
  - Ou seja, quando:

$$\forall n', n \quad \text{profundidade}(n') \geq \text{profundidade}(n) \Rightarrow$$
$$\text{custo de caminho}(n') \geq \text{custo de caminho}(n)$$

Ex.: quando todas operações tiverem o mesmo custo
Pois sempre explora profundidades mais rasas primeiro

- Pode não ser solução de menor **custo de caminho,** caso operadores tenham valores diferentes
  - ex. ir para uma cidade D passando por B e C pode ser mais perto do que passando só por E

Busca em largura

- *Complexidade de tempo e memória*
  - Considere que cada estado tem b sucessores
    - Número de nós em nível i = b^i
  - Suponha que primeira solução está no nível d
    - No pior caso, expande todos os nós exceto o último no nível d, gerando $b^{d+1} – b$ nós no nível d+1
    - Número total de nós gerados = $1+ b + b^2 +...+ (b^{d+1} – b)$
      - $O(b^{d+1})$: complexidade exponencial
      - Impraticável para problemas grandes
    - Tem que manter na memória fronteira e ancestrais dos nós da fronteira

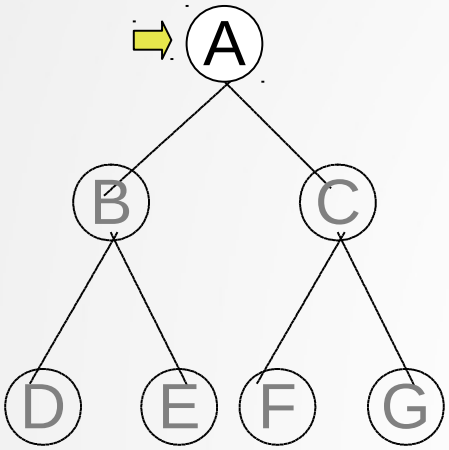
Busca em largura

- Fator de ramificação b = 10
  - Assumindo que:
    - 1 milhão de nós podem ser gerados por segundo
    - Cada nó ocupa 1000 bytes de espaço

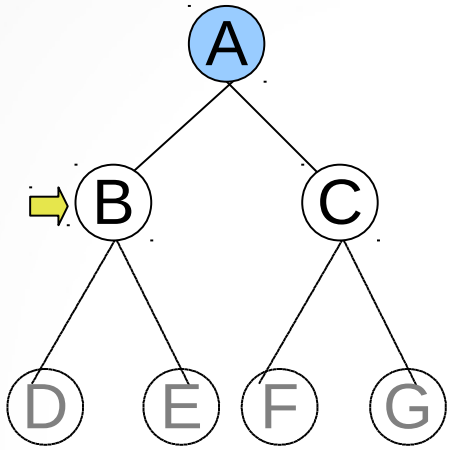
Profundidade	Nós	Tempo	Memória
2	110	0,11 ms	107 KB
4	1110	11 ms	10,6 MB
6	10^6	1,1 s	1 GB
8	10^8	2min	103 GB
10	10^10	3 h	10 TB
12	10^12	13 dias	1 PB
14	10^14	3,5 anos	99 PB
16	10^16	350 anos	10 EB

Com uma modificação no algoritmo básico que torna complexidade $O(b^d)$

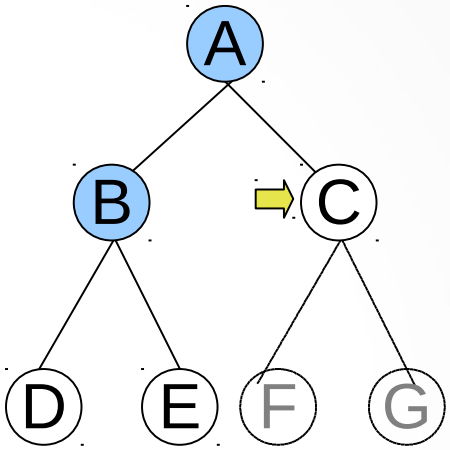
Busca em largura - armazenamento



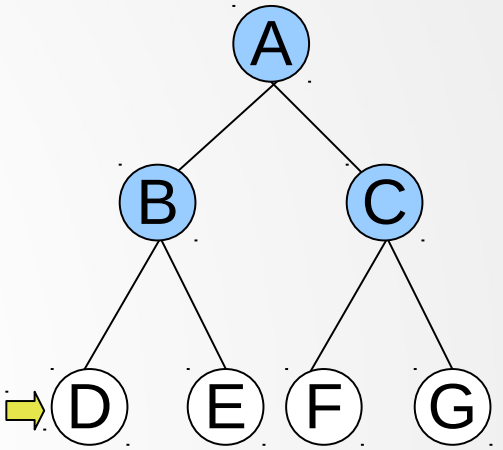
A



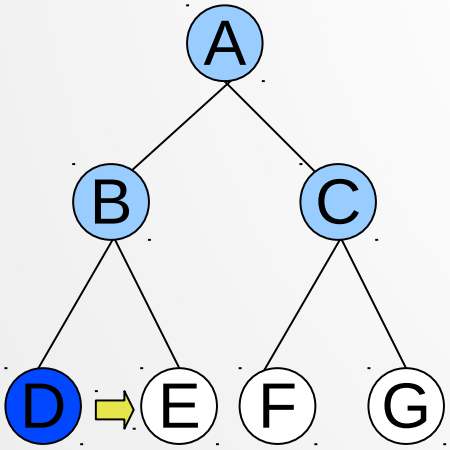
A B C



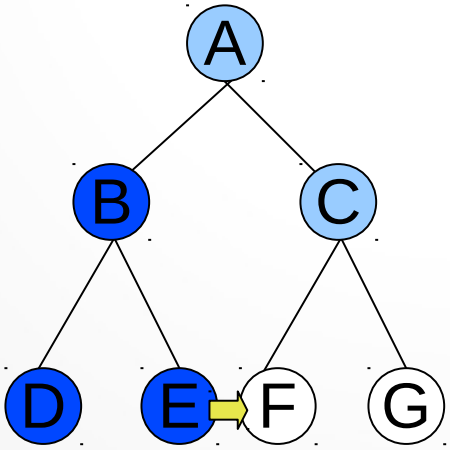
A B C D E



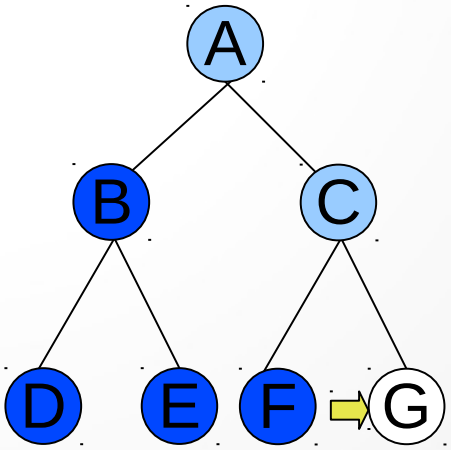
A B C D E F G



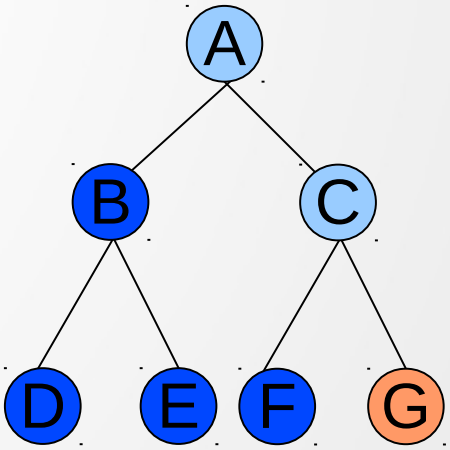
A B C E F G



A C F G



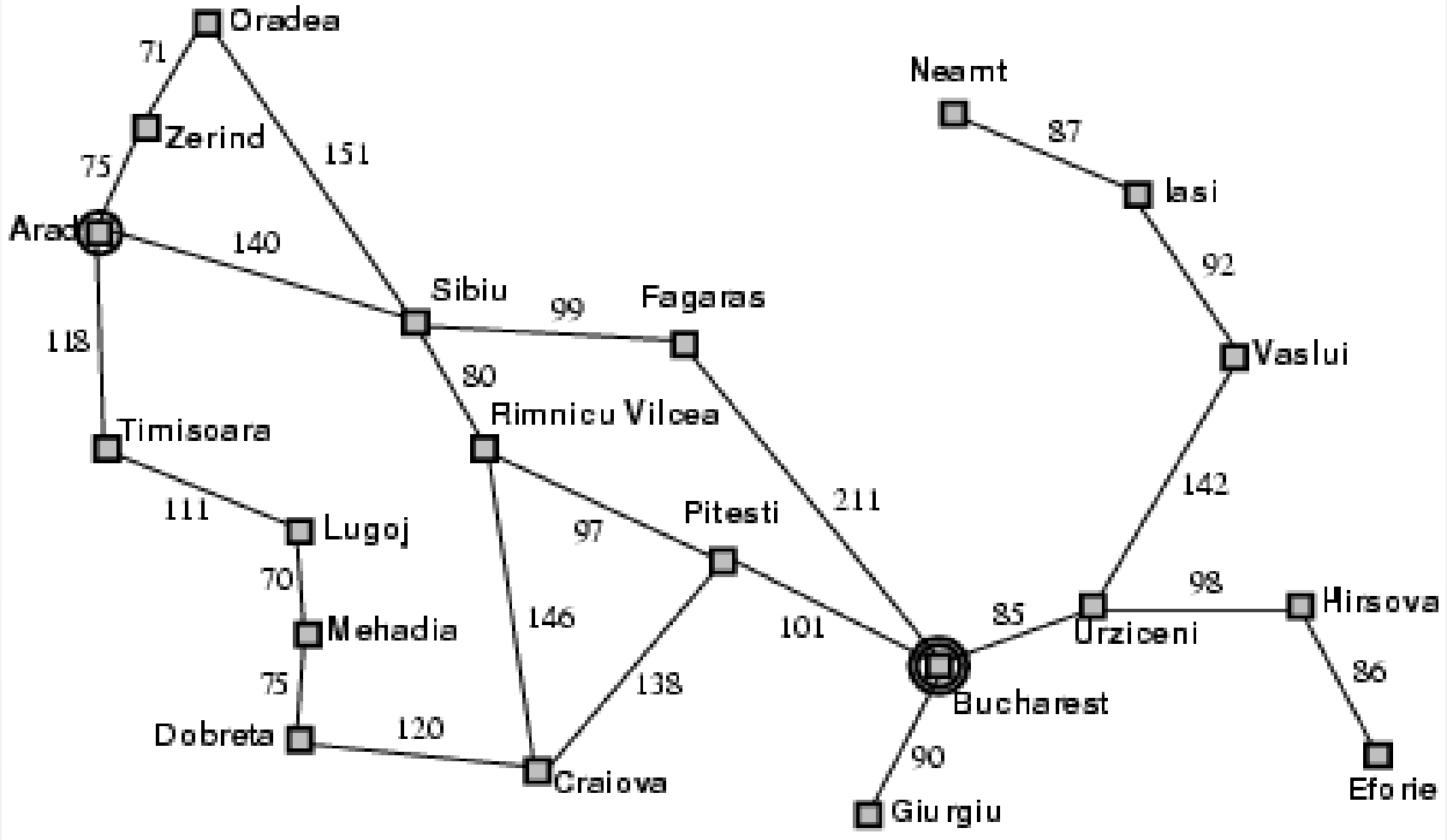
A C G



A C G

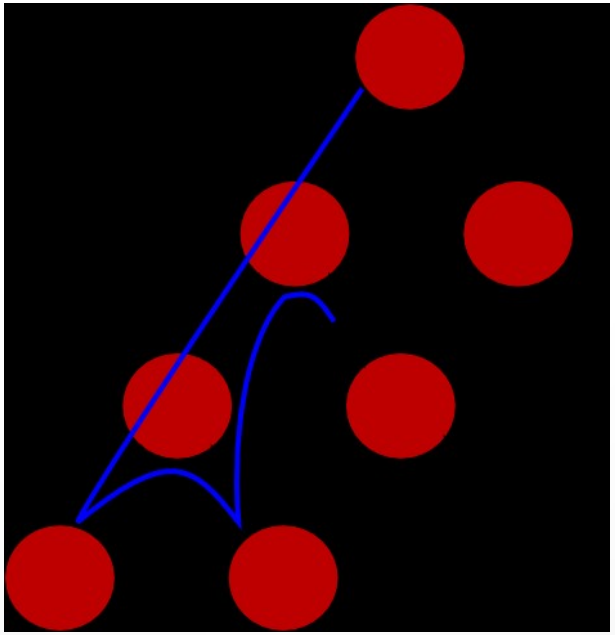
Exercício 1

- Aplique a busca em largura para o problema de achar o caminho de Arad a Bucharest



Busca em profundidade

- Sempre expande o nó atual mais profundo
  - Até chegar em objetivo ou em nó folha
    - Neste caso, retorna então ao nó seguinte mais raso



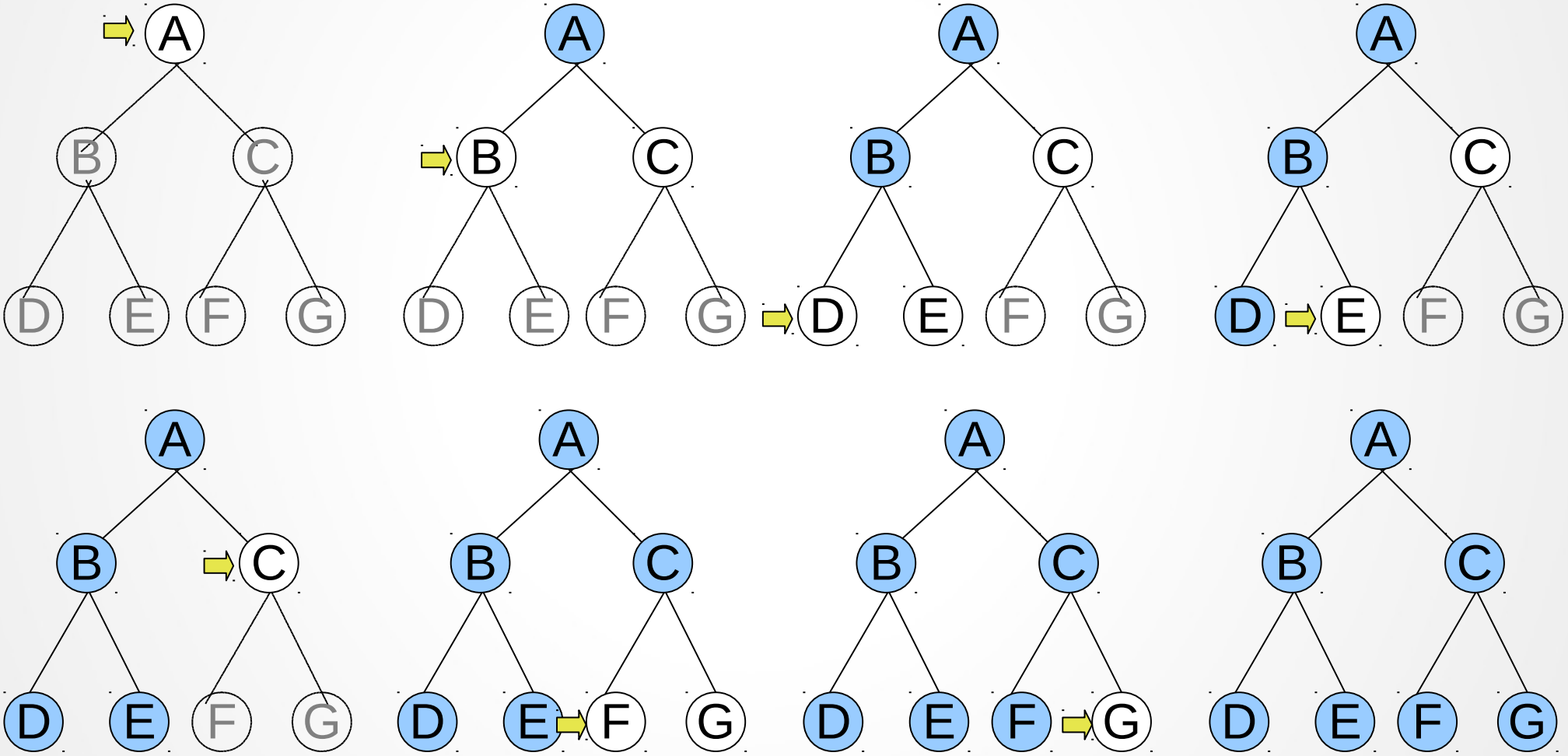
Busca em profundidade

- Ordem de ramificação dos nós:

  1. nó raiz
  2. primeiro nó de profundidade 1
  3. primeiro nó de profundidade 2, etc.

  - Quando um nó final não é solução, o algoritmo volta para expandir os nós que ainda estão na fronteira do espaço de estados (*backtracking*)

Busca em profundidade

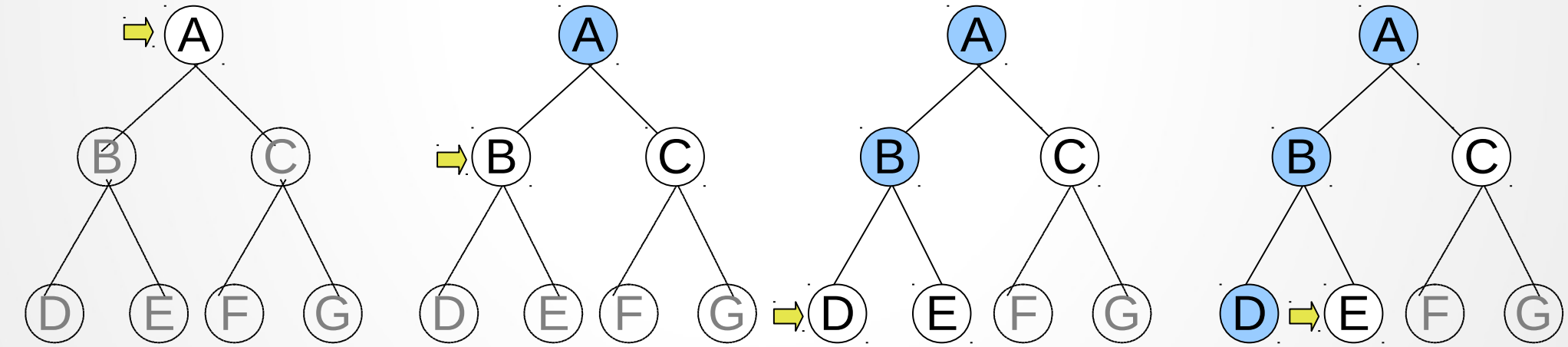


A B D E C F G

Busca em profundidade

- Fronteira pode ser vista como pilha
  - Novos sucessores são colocados no final
  - O último ou topo da pilha é selecionado a cada passo

Árvore de busca:



Pilha:

A

B
C

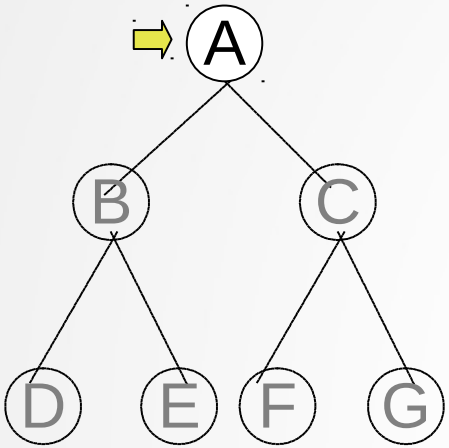
D
E
C

E
C

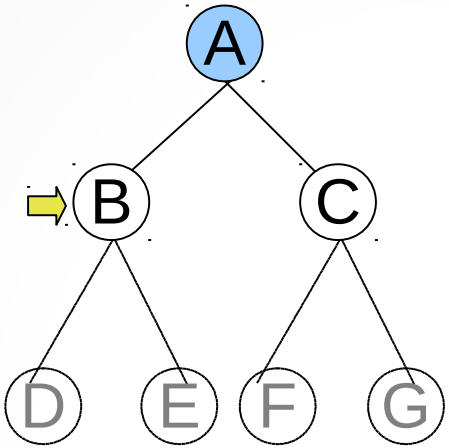
Busca em profundidade

- Requisitos de memória modestos
  - Precisa armazenar:
    - um único caminho de raiz a uma folha,
    - com os nós-irmãos não expandidos no caminho
      - Nós expandidos podem ser removidos da memória quando seus descendentes forem todos explorados
  - Em espaço de busca com:
    - Fator de ramificação b
    - Profundidade máxima m
    - Exige armazenamento de bm+1 nós: O(bm) (linear!)
    - No ex.: d = 12
      - Profundidade 118 KB
      - Largura 1 PB

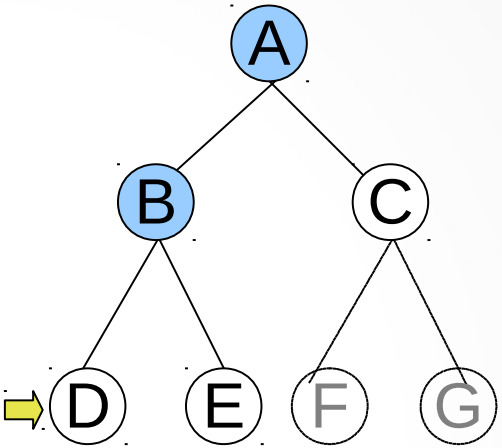
Busca em profundidade - armazenamento



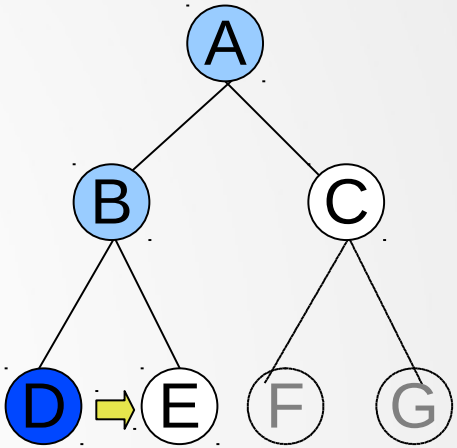
A



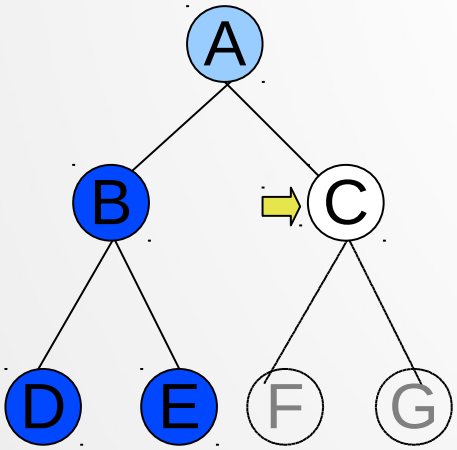
A B C



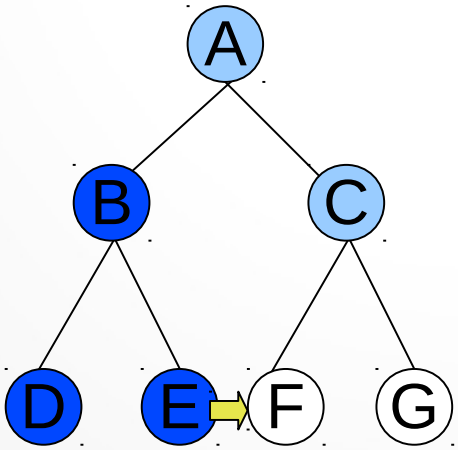
A B C D E



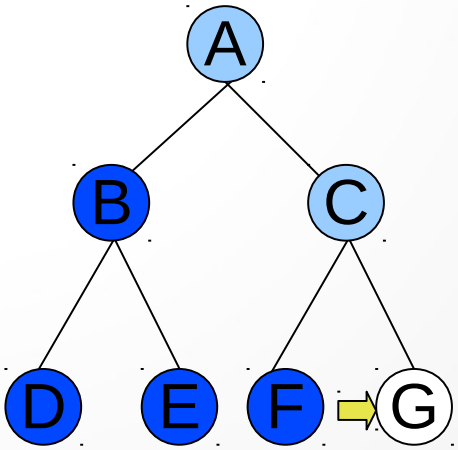
A B C E



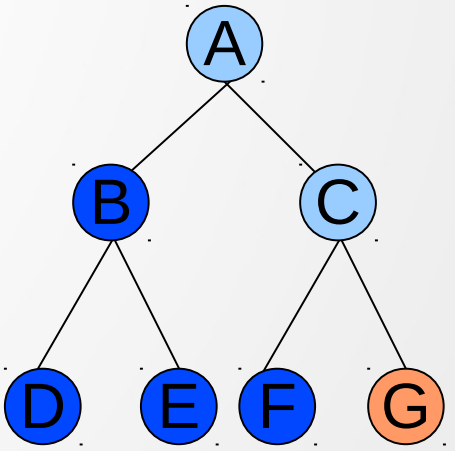
A C



A C F G



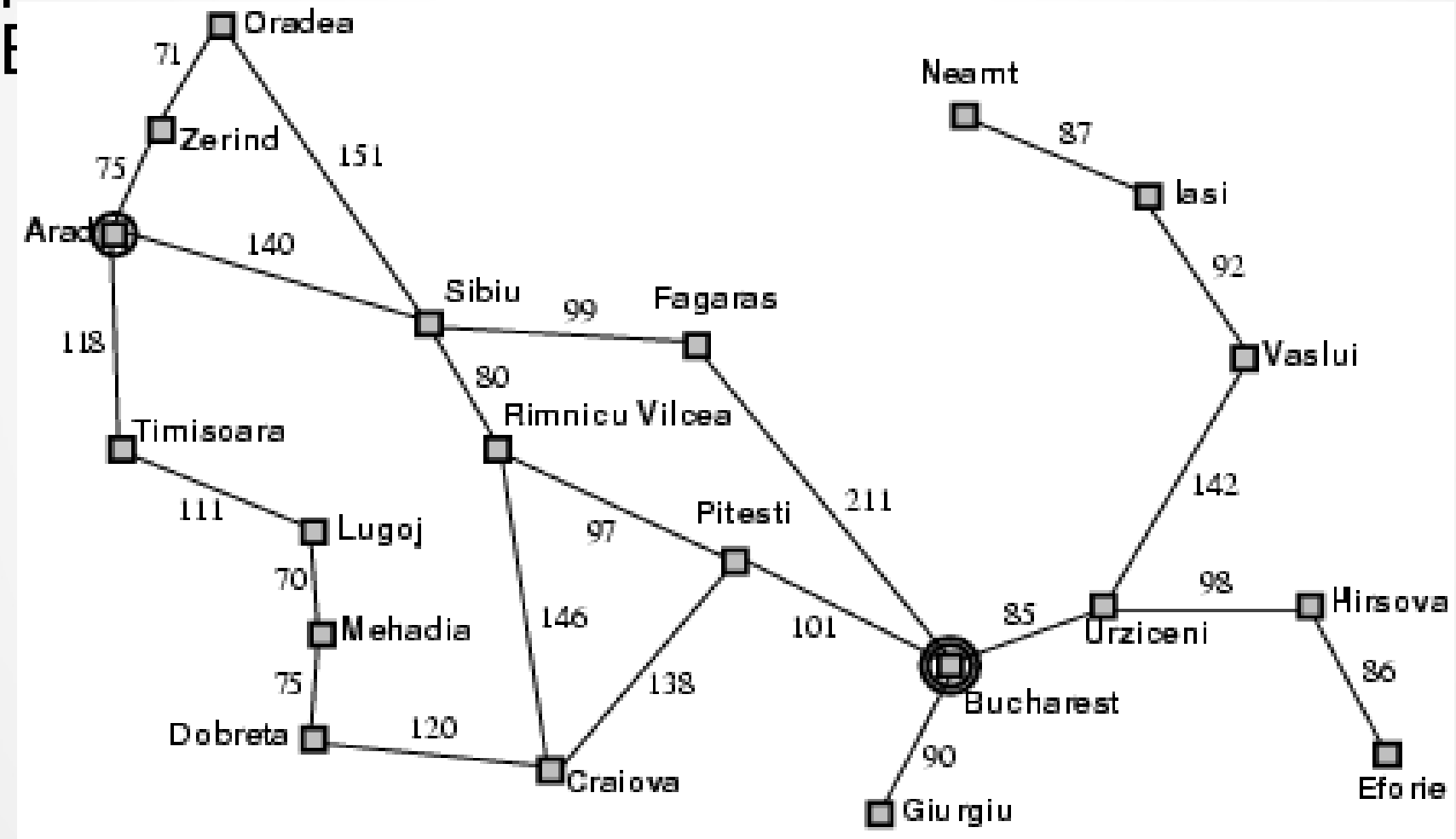
A C G



A C G

Exercício 2

- Aplique a busca em profundidade para o problema de achar o caminho de Arad a B



Busca profundidade x largura

- Tempo
  - *m* = *d*: BP tipicamente ganha
  - *m* > *d*: BL pode ganhar
  - *m é* ***infinito***: BL provavelmente irá melhor
- Espaço
  - BP geralmente é melhor que BL

Busca em profundidade

- Pode ficar paralisada ao descer um caminho muito longo (ou infinito)
  - Quando uma opção diferente levaria a uma solução próxima à raiz da árvore de busca
  - Por essa razão também *não é completa*
- Também *não é ótima*
  - Pode retornar solução em profundidade maior na árvore de busca do que a mais rasa

Busca em profundidade

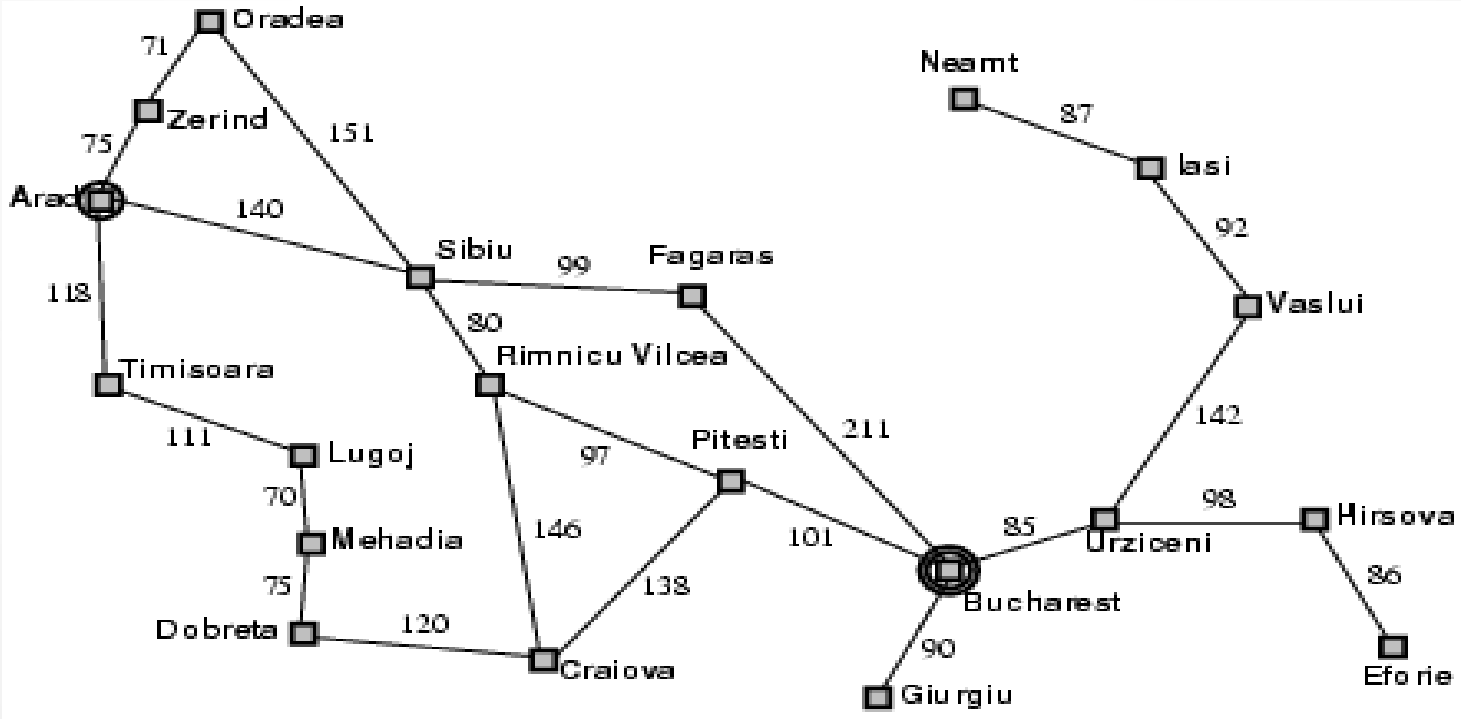
- *Complexidade* de pior caso
  - Irá gerar todos os $O(b^m)$ nós na árvore de busca
    - m é a profundidade máxima de qualquer nó
      - m pode ser muito maior que d
        - Até mesmo infinito
- Esta estratégia deve ser evitada quando as árvores geradas são muito *profundas* ou geram *caminhos infinitos*

Busca em profundidade limitada

- Colocar um limite máximo de profundidade a ser explorado
  - Lidando assim com árvores ilimitadas
    - Resolve problema de caminhos infinitos
    - Busca em profundidade é caso especial de busca em profundidade limitada com limite $l = \infty$
  - Também pode ser *incompleto*
    - Se solução mais rasa estiver abaixo de limite l
  - Se $d < l$, *não é ótima*
  - *Complexidade de tempo*: $O(b^l)$
  - *Complexidade de espaço*: $O(bl)$

Busca em profundidade limitada

- Às vezes, conhecimento sobre problema pode dar limite de profundidade
  - Ex.: Romênia

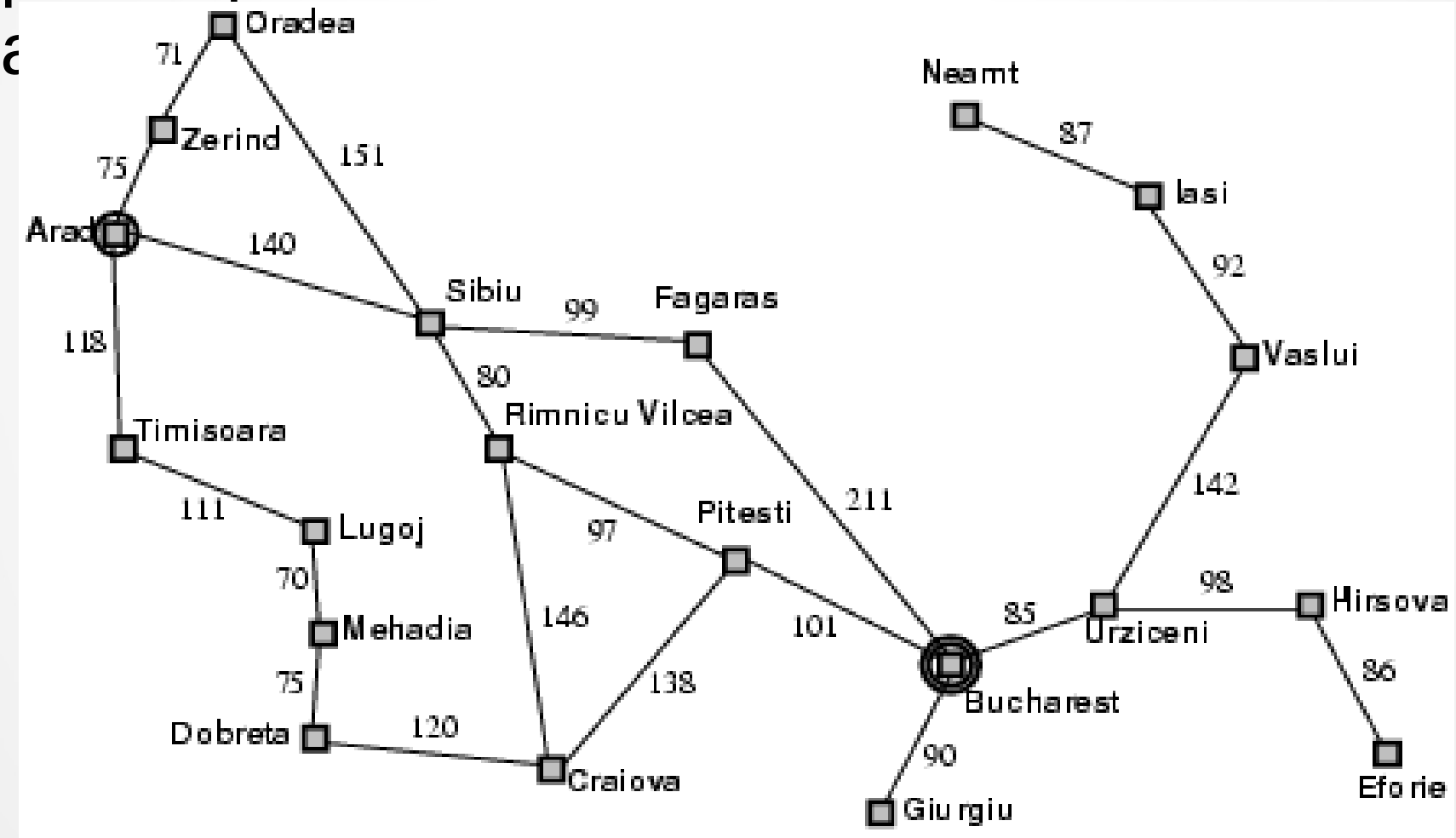


Busca em profundidade limitada

- Ex.: Romênia
  - Há 20 cidades
  - Então o comprimento mais longo pode ser l=19
  - Estudando o mapa, qualquer cidade pode ser alcançada a partir de outra com no máximo 9 passos
    - Diâmetro do espaço de estados
      - Infelizmente, não é conhecido para maioria dos problemas

Exercício 3

- Aplique a busca em profundidade limitada para o problema de achar o caminho de Arad a



Busca em profundidade por aprofundamento iterativo

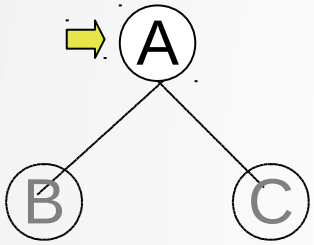
- Busca por aprofundamento iterativo
  - Estratégia geral usada com frequência em conjunto com busca em profundidade
  - Encontra o melhor limite de profundidade
    - Aumentando gradualmente o limite
      - 0, 1, 2, etc
      - Até encontrar um objetivo
        - Acontece quando alcançar d, profundidade do nó objetivo mais raso

Busca por aprofundamento iterativo

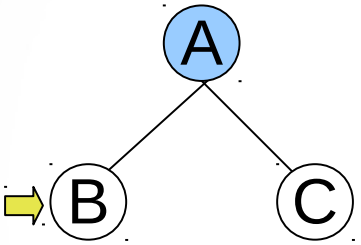
Limite = 0

Busca por aprofundamento iterativo

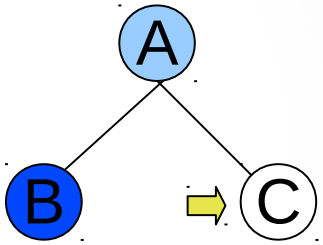
Limite = 1



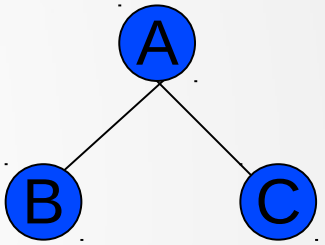
A



A B C

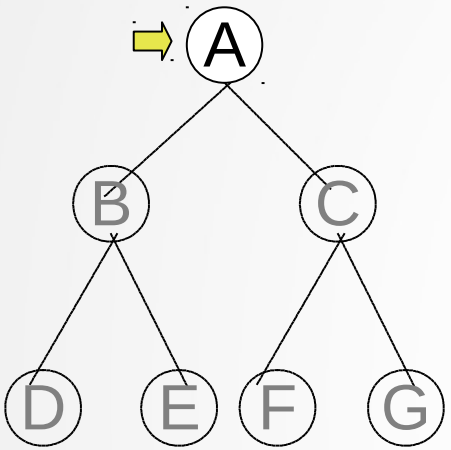


A C

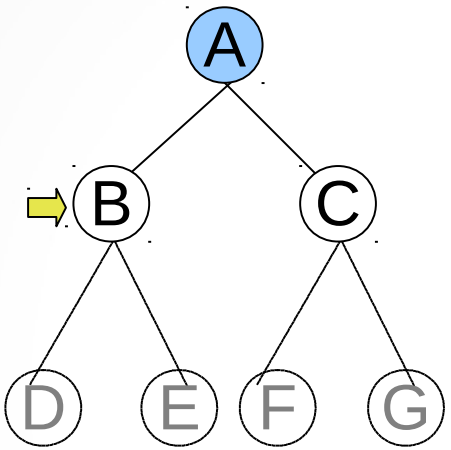


Busca por aprofundamento iterativo

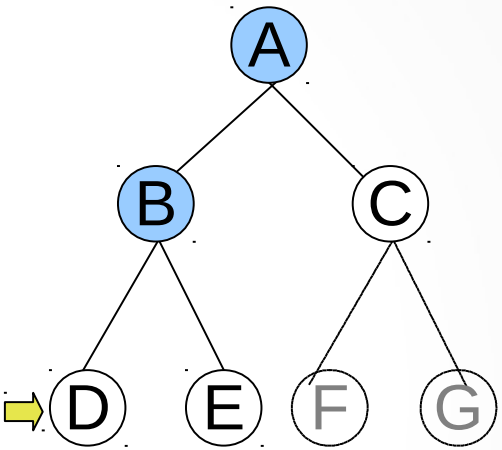
Limite = 2



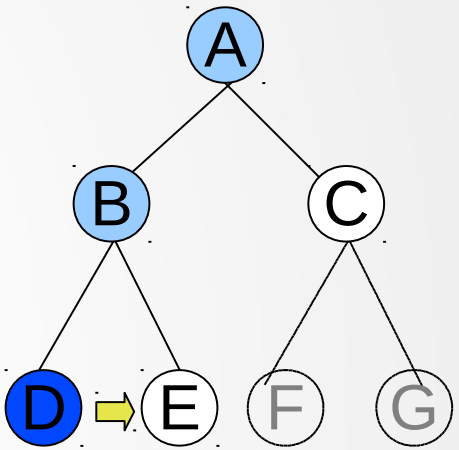
A



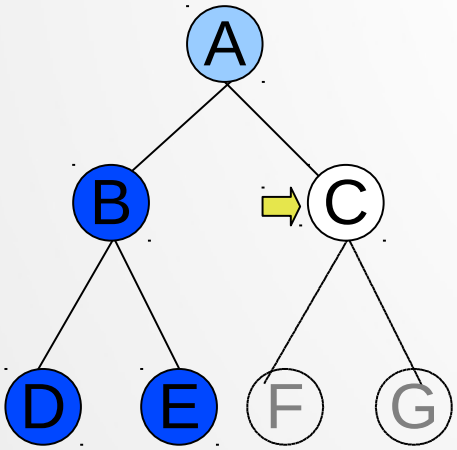
A B C



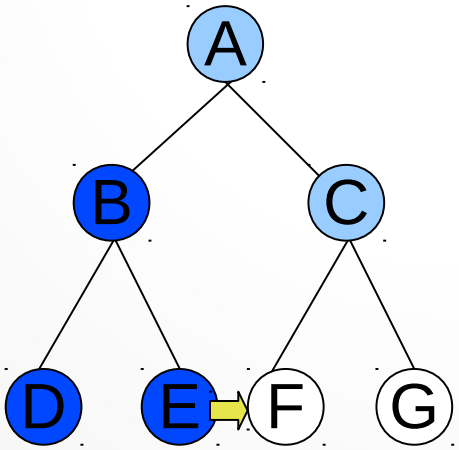
A B C D E



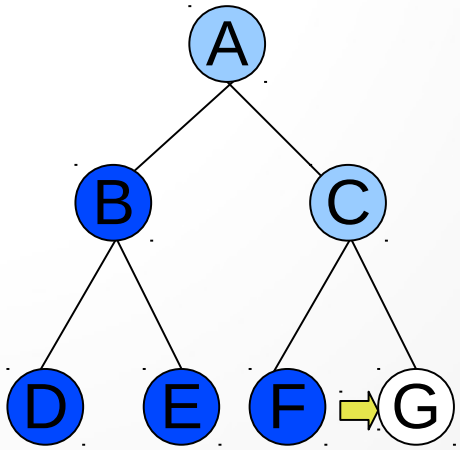
A B C E



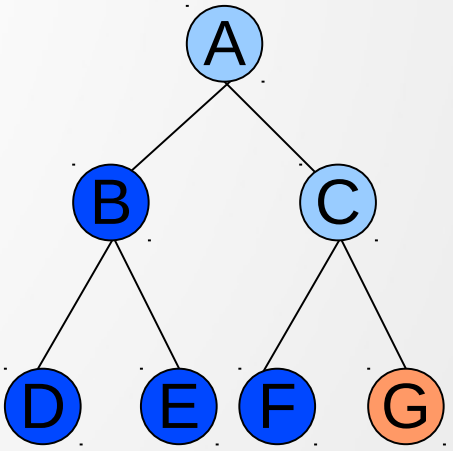
A C



A C F G



A C G



A C G

Busca por aprofundamento iterativo

- Pode parecer desperdício
  - Estados são gerados várias vezes
    - Custo não é muito alto, pois a maior parte dos nós estará em níveis inferiores
      - Nós do nível inferior (profundidade d) são gerados uma vez...
      - Os do penúltimo são gerados duas vezes...
      - E assim por diante...
      - Até filhos da raiz, gerados d vezes

Busca por aprofundamento iterativo

- Número total de nós gerados é:
  - nós(BAI) = $db + (d-1)b^2 + ... + (1)b^d$
    - *Complexidade de tempo* $O(b^d)$
    - É mais rápido que busca em largura
    - nós(BL) = $b + b^2 + ... + b^d + (b^{d+1} – b)$
    - nós(BP) = $b + b^2 + ... + b^d$
      - Limitando profundidade em d
  - Ex.: b = 10, d = 5;
    - nós(BAI) = 123450
    - nós(BL) = 1111100
    - nós(BP) = 111111
      - Overhead = (123,456 - 111,111)/111,111 = 11%

Busca em profundidade por aprofundamento iterativo

- Busca por aprofundamento iterativo
  - *Requisitos de memória modestos*: O(bd)
  - *Complexidade de tempo*: $O(b^d)$
  - *Completa* quando fator de ramificação é finito
  - *Ótima* quando custo de caminho cresce com a profundidade do nó

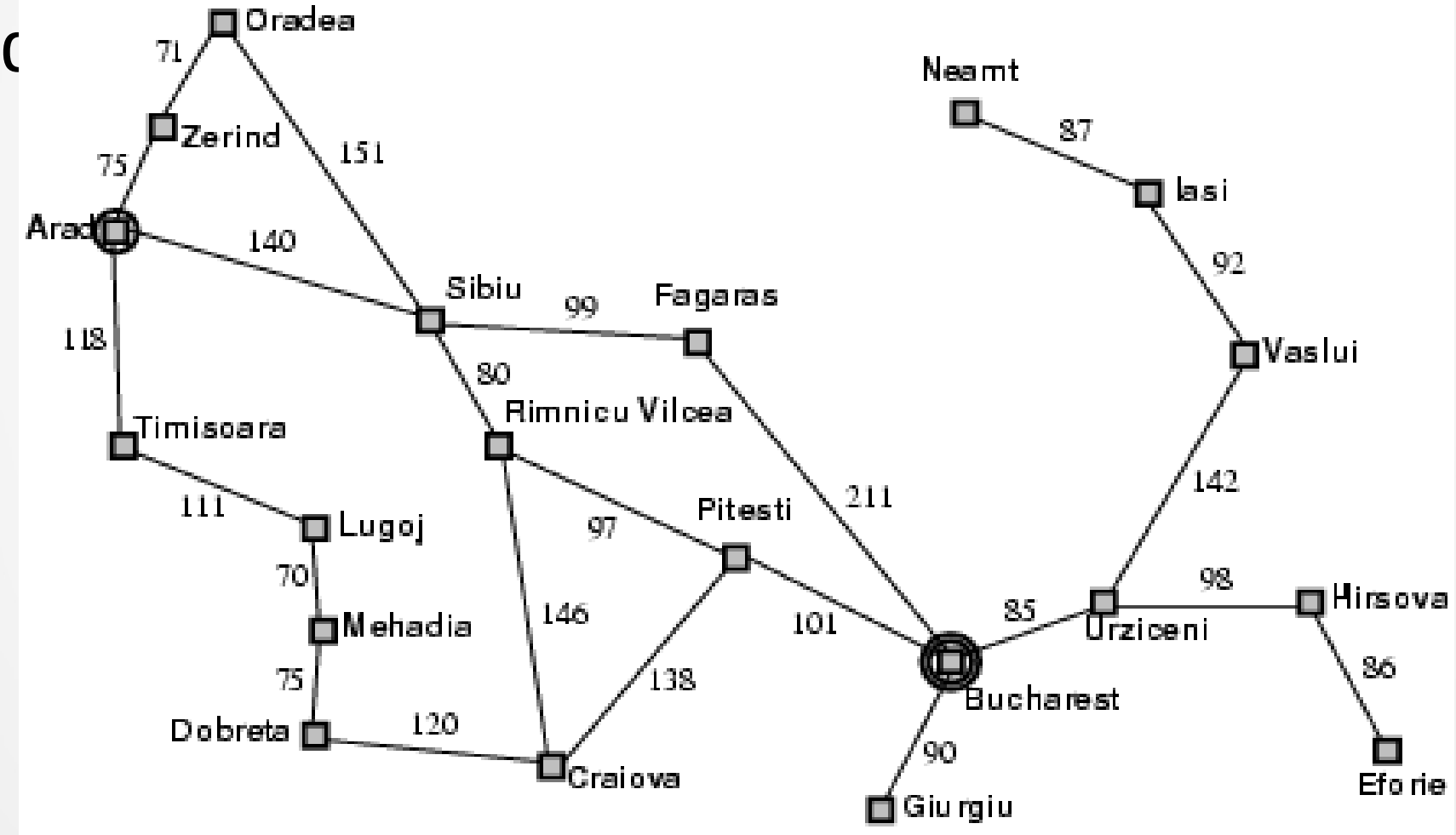
Busca por aprofundamento iterativo

Em geral, é método de busca sem informação preferido quando existe um espaço de busca grande e a profundidade da solução não é conhecida

É similar à busca em largura, pois explora um nível de nós em cada iteração, porém mais eficiente em tempo e espaço

Exercício 4

- Aplique a busca com aprofundamento iterativo para o problema de achar o caminho d



Comparação entre estratégias de busca sem informação

Critério	BL	BP	BPL	BAI
Completa	Sim*	Não	Não	Sim*
Tempo	$O(b^{d+1})$	$O(b^{m})$	$O(b^{l})$	$O(b^{d})$
Espaço	$O(b^{d+1})$	O(bm)	O(bl)	O(bd)
Ótima	Sim***	Não	Não	Sim***

b é fator de ramificação

d é profundidade da solução mais rasa

m é profundidade máxima da árvore de busca

l é limite de profundidade

* Se b é finito
** Se b é finito e ambos sentidos usam busca em extensão
*** Se custos dos passos são iguais

Referências

- Capítulo 3 Russel e Norvig

Referências

- Livro Russel e Norvig, capítulo 3