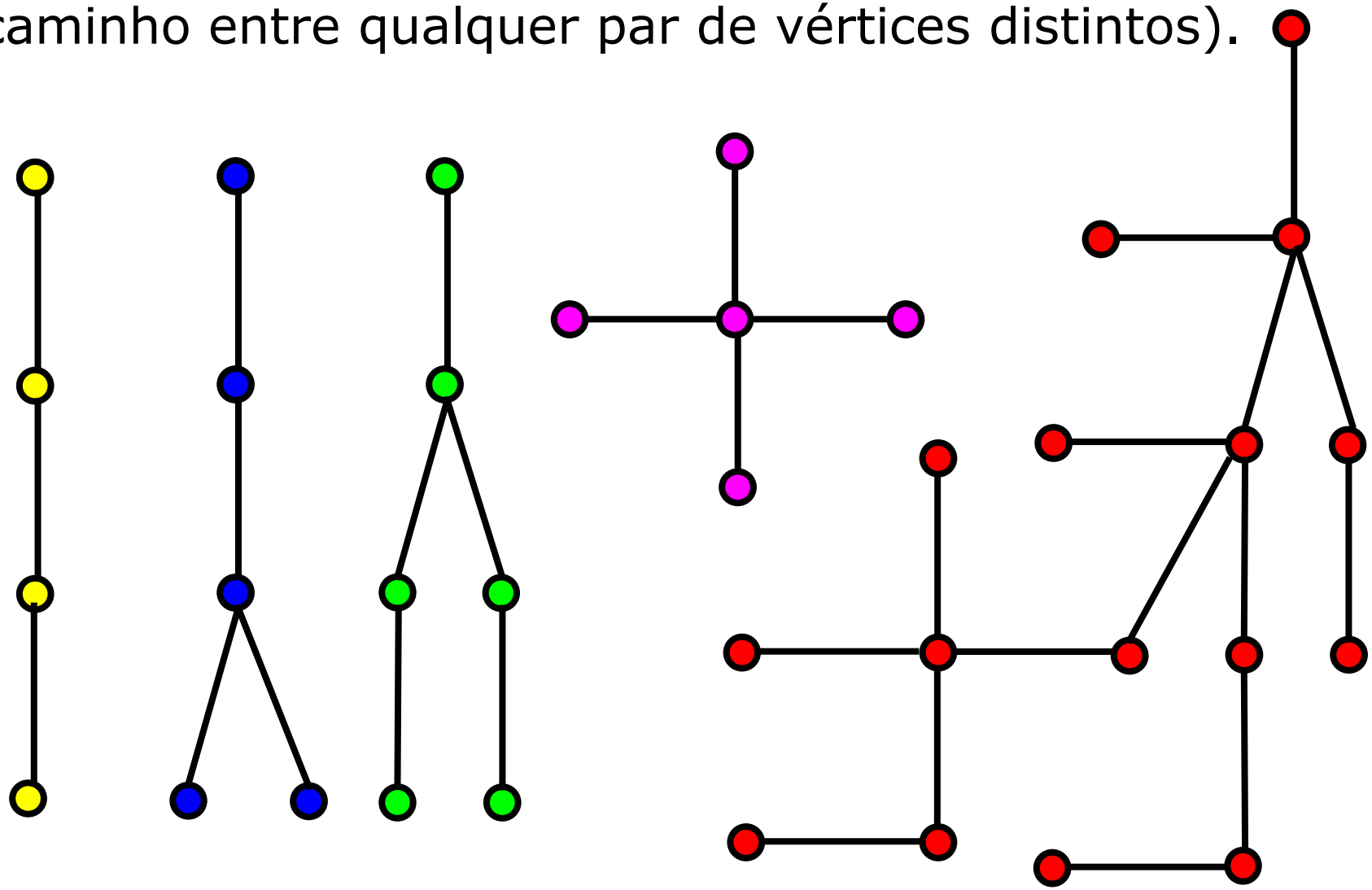
Estratégias de Busca sem Informação

Material gentilmente cedido por **José Augusto Baranauska**s
Departamento de Computação e Matemática – FFCLRP-USP

Cristiane Neri Nobre

Árvore (Tree)

Grafo acíclico (não possui ciclos) e conexo (existe um caminho entre qualquer par de vértices distintos).



Busca

- Algoritmos inteligentes devem agir de maneira tal que o ambiente passa por uma sequência de estados que maximizam alguma medida de desempenho

- Dessa forma, o algoritmo adota um objetivo e tenta satisfazê-lo

- O algoritmo pode decidir o que realizar primeiramente examinando as sequências possíveis de ações que levam a estados com valores conhecidos e então escolher o melhor entre eles

- O processo de procurar por uma sequência é chamado de busca

Busca em Espaço de Estados

- Um grafo pode ser usado para representar um espaço de estados onde:

  - ✓ Os nós correspondem a situações de um problema
  - ✓ As arestas correspondem a movimentos permitidos ou ações ou passos da solução
  - ✓ Um dado problema é solucionado encontrando-se um caminho no grafo

- Um problema é definido por
  - ✓ Um espaço de estados (um grafo)
  - ✓ Um estado (nó) inicial
  - ✓ Uma condição de término ou critério de parada; estados (nós) terminais são aqueles que satisfazem a condição de término

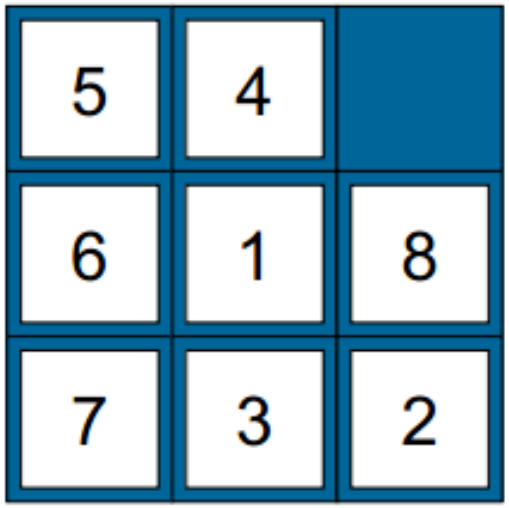
Busca em Espaço de Estados

- Se não houver custos, não há interesse em soluções de caminho mínimo

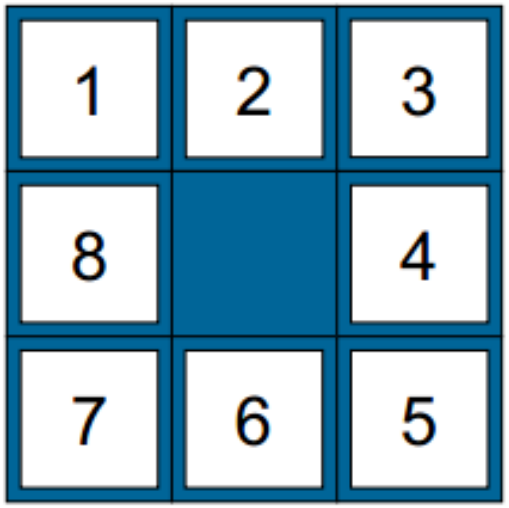
- No caso em que custos são adicionados aos movimentos normalmente há interesse em soluções de custo mínimo

  - ✓ O custo de uma solução é o custo das arestas ao longo do caminho da solução

Exemplo: Quebra-Cabeça-8



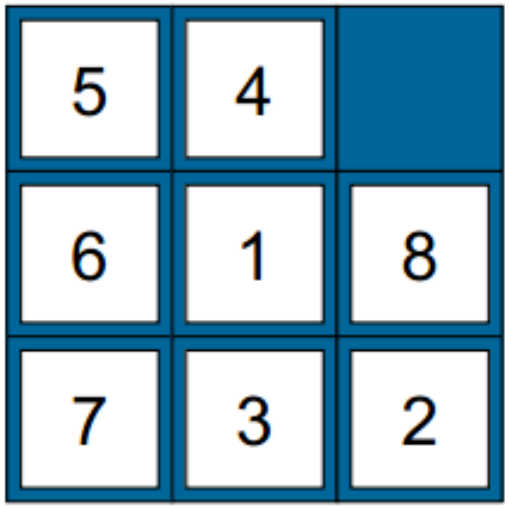
Estado Inicial



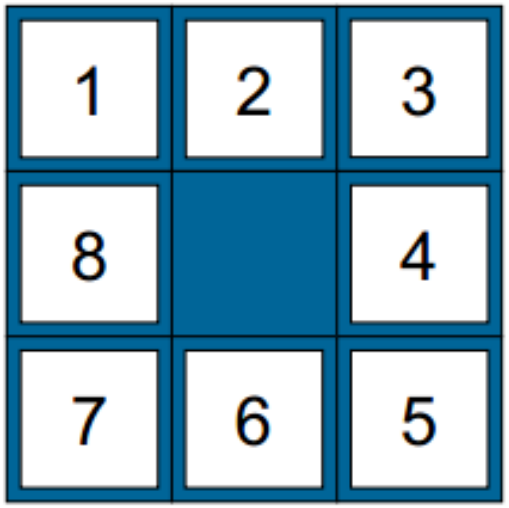
Estado Final

Estados?
Operadores?
Estado Final: = estado fornecido
Custo do caminho?

Exemplo: Quebra-Cabeça-8



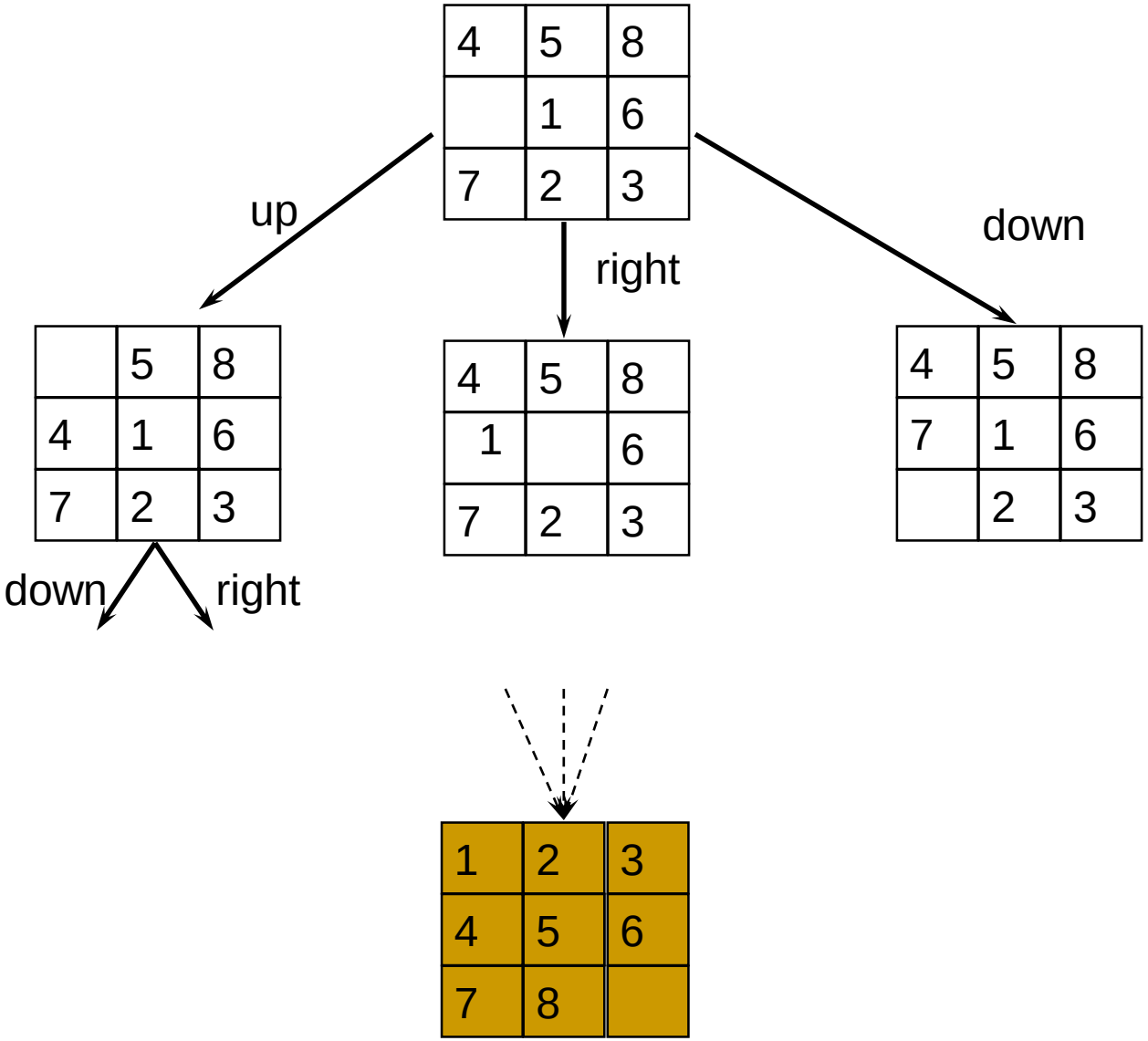
Estado Inicial



Estado Final

Estados: posições inteiras dos quadrados (ignorar posições intermediárias)
Operadores: mover se branco à esquerda, direita, em cima, em baixo (ignorar ação de desalojar, etc)
Estado Final: = estado fornecido
Custo do caminho: 1 por movimento

Árvore de busca para o "Quebra-Cabeça de 8"



Exemplo: Quebra-Cabeça de 8
Problema NP-Completo

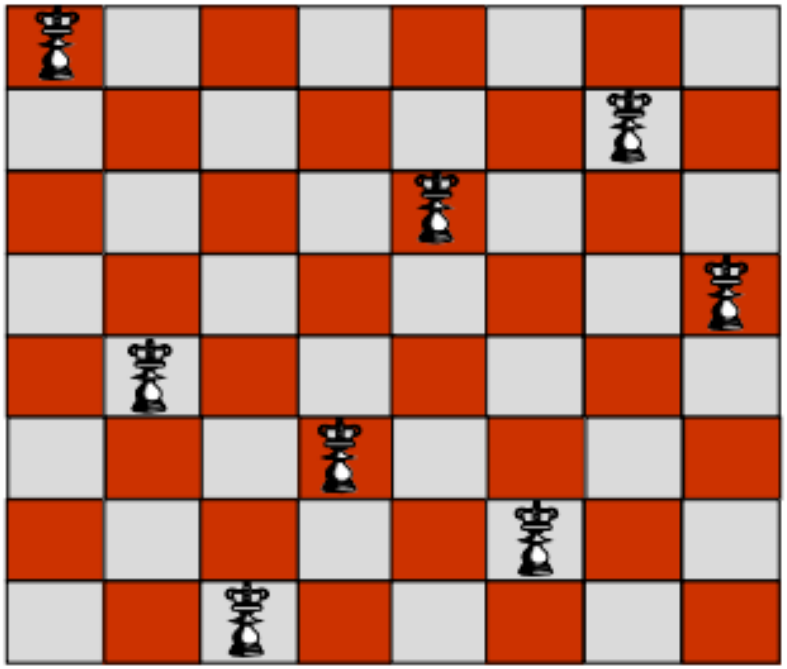
Qual a complexidade?

8 peças:

15 peças:

24 peças:

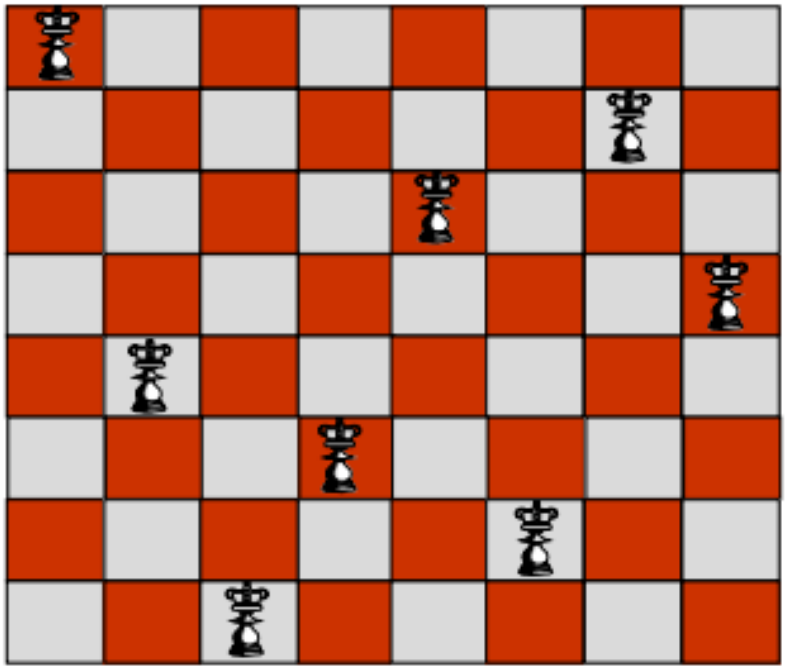
Exemplo: 8 Rainhas



Dispor 8 rainhas no tabuleiro de forma que não possam se "atacar"

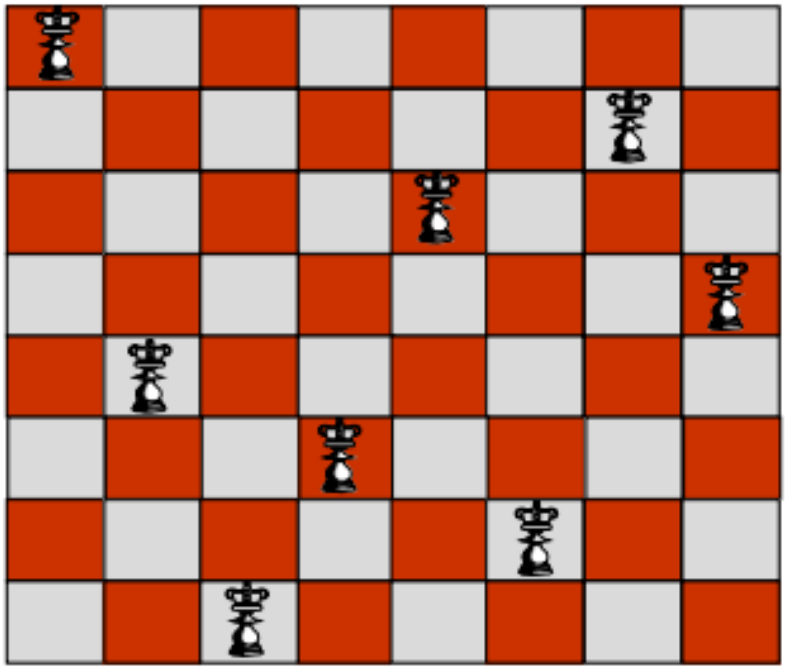
- ✓ não pode haver mais de uma rainha em uma mesma linha, coluna ou diagonal

Exemplo: 8 Rainhas



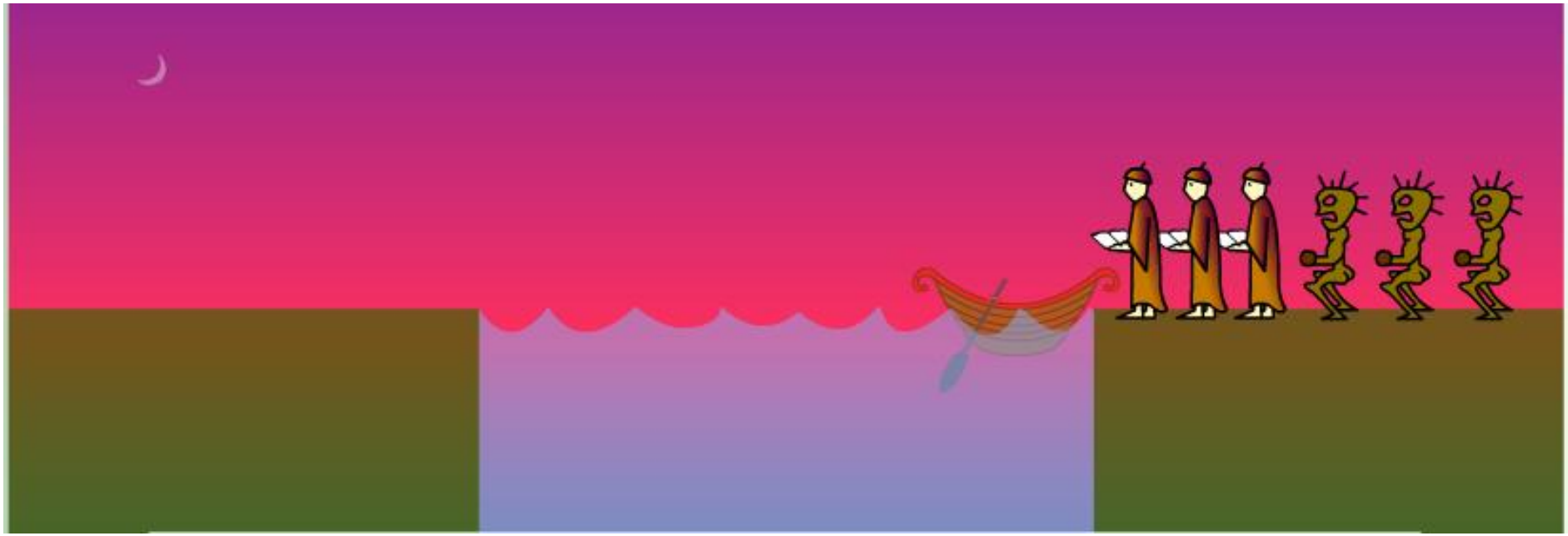
Estados?
Operadores?
Estado Final?
Custo do caminho?

Exemplo: 8 Rainhas



Estados: qualquer configuração das rainhas no tabuleiro
Estado inicial: tabuleiro vazio
Função sucessor: coloca uma rainha em um espaço vazio d
tabuleiro
Meta: colocar 8 rainhas de modo que elas não se ataquem
Número de estados: 64 x ... x 57 ≈ 3 x 10^14 estados

Exemplo: Missionários e Canibais



http://www.plastelina.net/games/game2.html

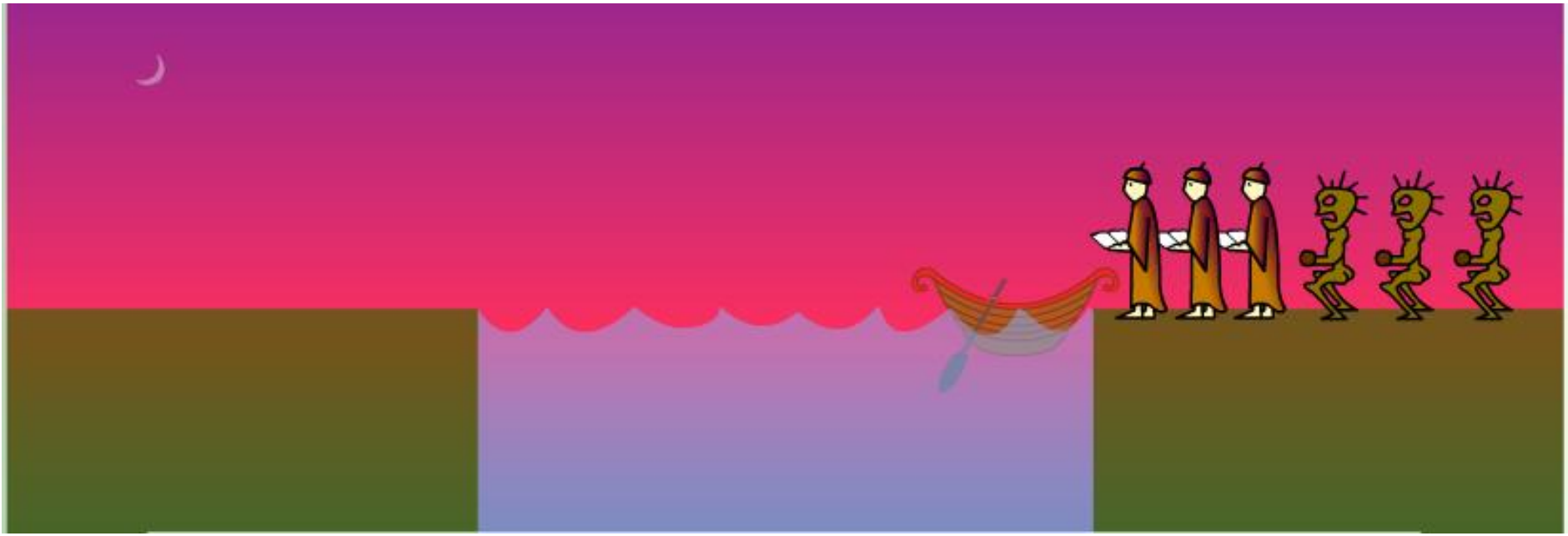
Exemplo: Missionários e Canibais

- 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.

- O número de canibais não pode exceder o número de missionários em nenhuma das margens do rio.

- Encontre uma forma de levar todos para a outra margem do rio, utilizando este barco.

Exemplo: Missionários e Canibais



http://www.plastelina.net/games/game2.html

Estados?
Estado Final?

Operadores?
Custo do caminho?

Exemplo: Missionários e Canibais

- Estados: uma sequência de três números representando <missionário, canibais, barco>

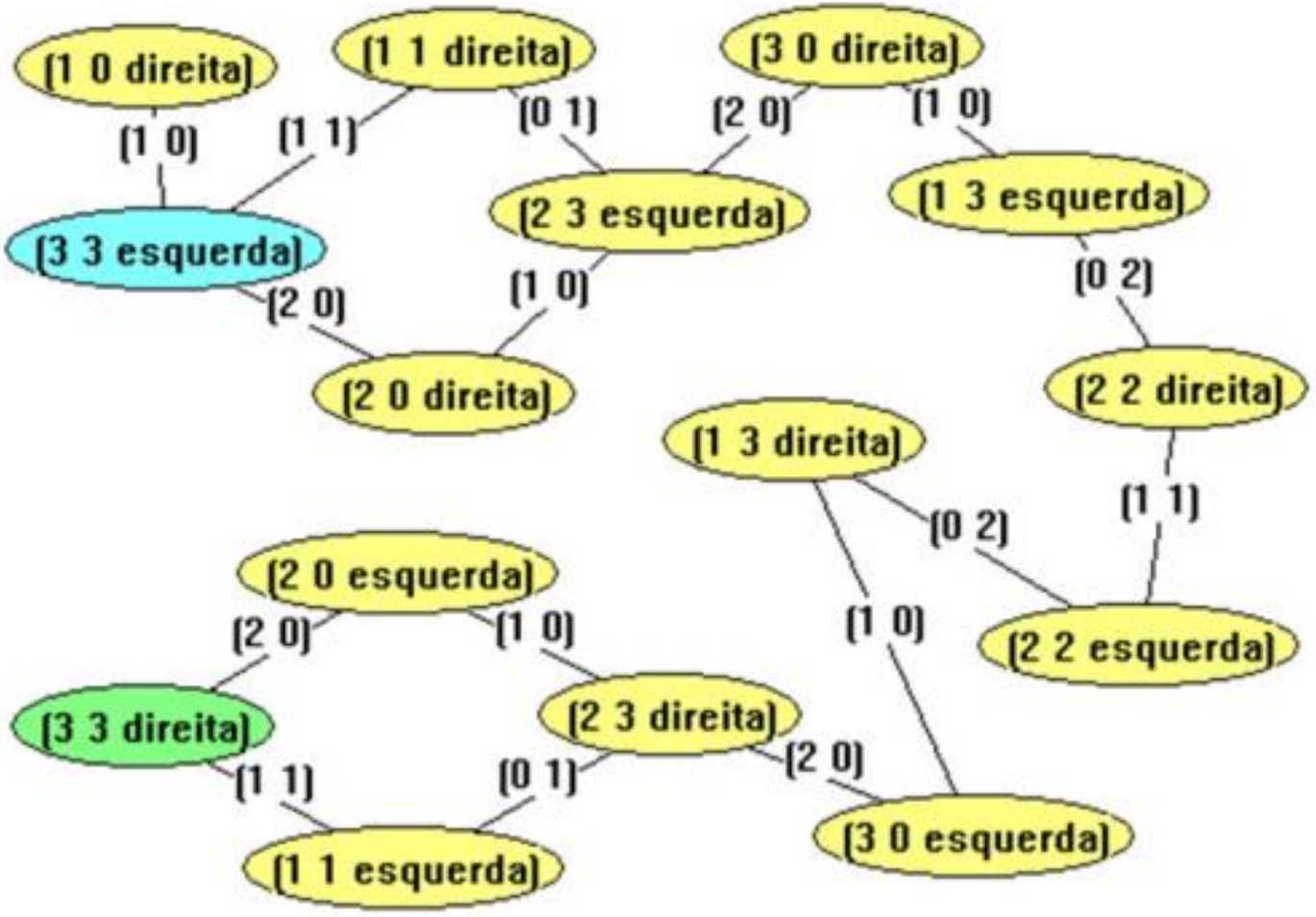
- Estado inicial <3,3,1>

- Função sucessor:
  - ✓ Levar 1 missionário e 1 canibal
  - ✓ Levar 2 missionários
  - ✓ Levar 2 canibais
  - ✓ Levar 1 canibal
  - ✓ Levar 1 missionário

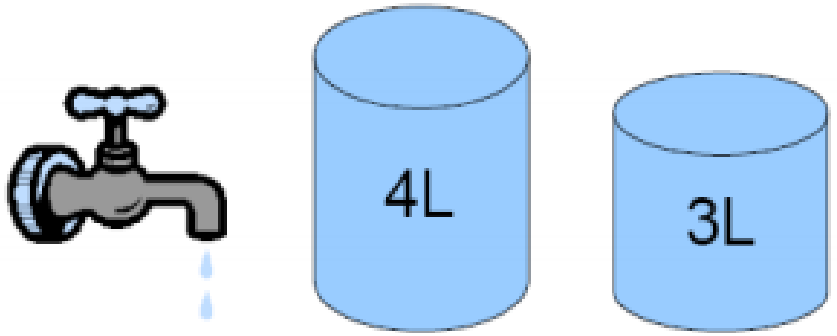
Teste Objetivo : <0,0,0>

Custo : passagens pelo rio

Exemplo: Missionários e Canibais



Problemas das jarras d'água



Como colocar exatamente 2 litros na jarra de 4 litros?

Estados: $\{(x,y) \mid 0 \leq x \leq 4 \text{ e } 0 \leq y \leq 3\}$

Estado Inicial (0,0)

Estado Final (2,0)

Considerações sobre a busca

O que está sendo procurado?

O que é interessante: a solução ou o caminho para a solução?

Completude

O algoritmo de busca garante encontrar uma solução quando há uma?

Busca exaustiva versus busca heurística (com poda)

Complexidade de tempo e espaço

E ótima?

A estratégia encontra a solução de melhor qualidade quando há diferentes soluções?

Árvore de busca

- A busca no espaço de estados pode ser representada como uma árvore de busca
  - ✓ Cada nó na árvore de busca representa um estado
  - ✓ Cada aresta na árvore representa uma ação, correspondendo a uma alteração no ambiente partindo do nó pai para o nó filho
- A cada aresta pode ser associado um custo ou utilidade
- Cada caminho a partir da raiz da árvore até outro nó representa uma sequência de ações

Árvore de busca

- Alguns nós folhas da árvore de busca representam estados finais
- O problema de navegar no espaço de busca então pode ser entendido como gerar e manter uma árvore de busca até que uma solução (nó final) ou caminho seja gerado

Busca em espaço de estados

- Estratégias Básicas de Busca (Uniforme, Exaustiva ou Cega)

  - ✓ não utiliza informações sobre o problema para guiar a busca. Ou seja, não tem nenhuma informação adicional sobre os estados, além daquelas fornecidas na definição do problema.

  - ✓ estratégia de busca exaustiva aplicada até uma solução ser encontrada (ou falhar)

    - ➢ Profundidade (Depth-first)
    - ➢ Profundidade limitada (Depth-first Limited)
    - ➢ Profundidade iterativa (Iterative Deepening)
    - ➢ Largura (Breadth-first)
    - ➢ Custo Uniforme (Uniform Cost)
    - ➢ Bidirecional

- Estratégias Heurísticas de Busca (Busca Informada)

Busca em Profundidade

- A busca em profundidade(do inglês depth-first search -DFS) é um algoritmo para caminhar no grafo;

- Seu núcleo se concentra em buscar, sempre que possível, o mais fundo no grafo.

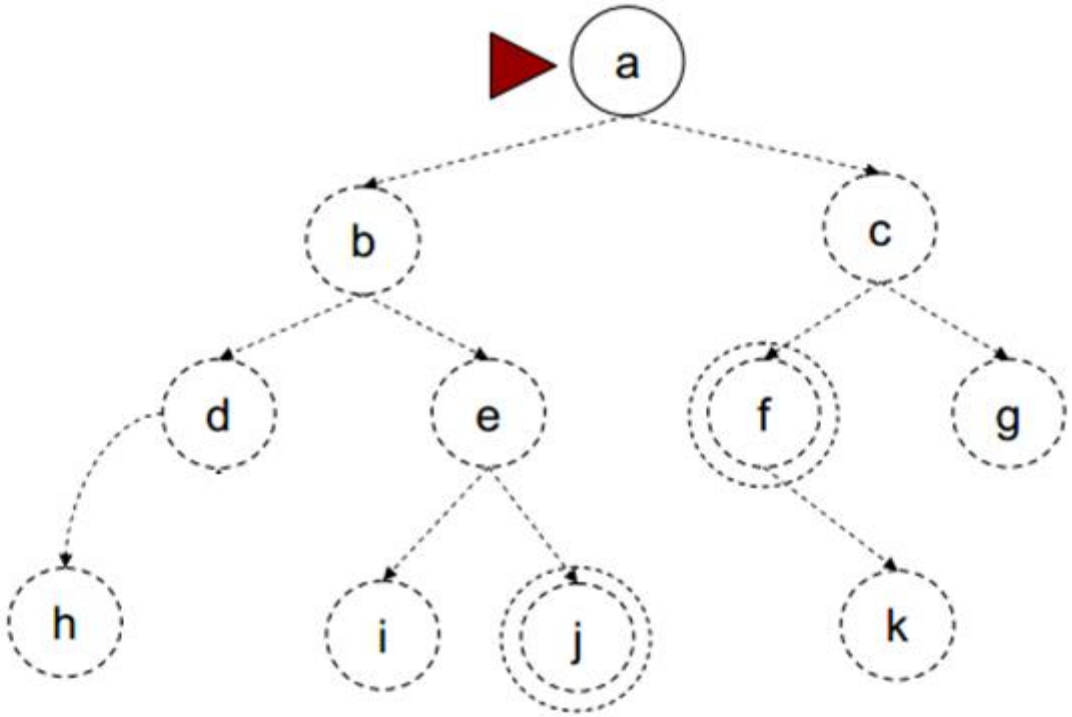
- As arestas são exploradas a partir do vértice *v* mais recentemente descoberto que ainda possui arestas não exploradas saindo dele

- Quando todas arestas adjacentes a *v* tiverem sido exploradas, a busca "anda para trás"(do inglês *backtracking*) para explorar vértices do qual *v* foi descoberto;

- O processo continua até que sejam descobertos todos os vértices que são alcançáveis a partir do vértice original;

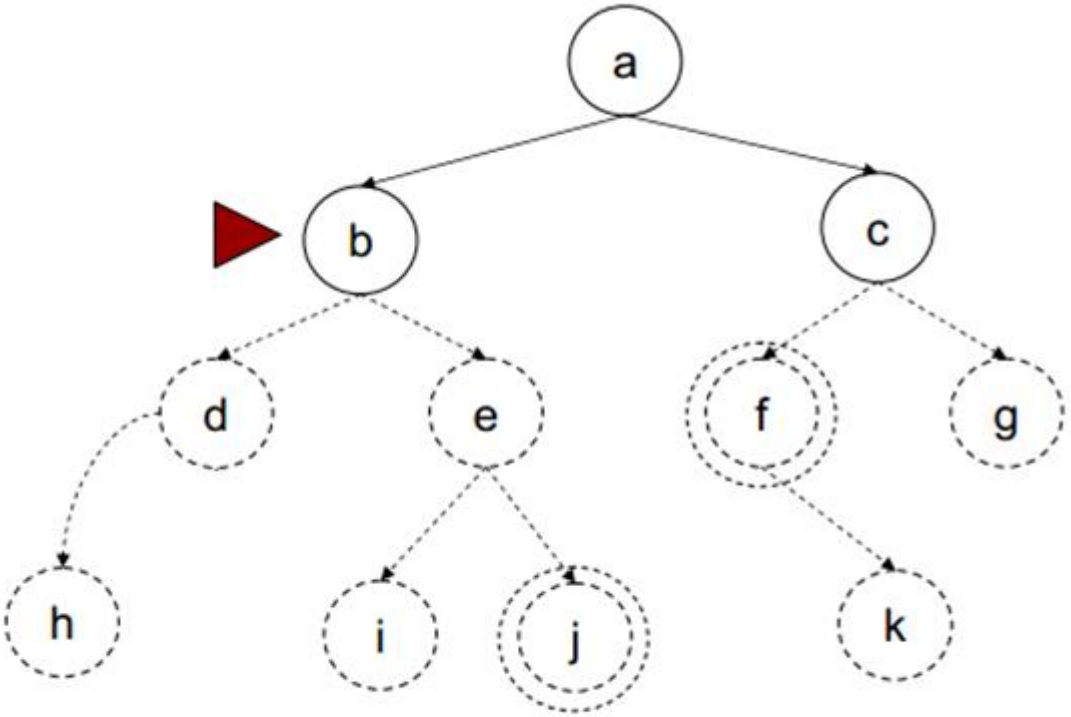
- Se todos os vértices já foram descobertos, então é o fim.

Busca em profundidade



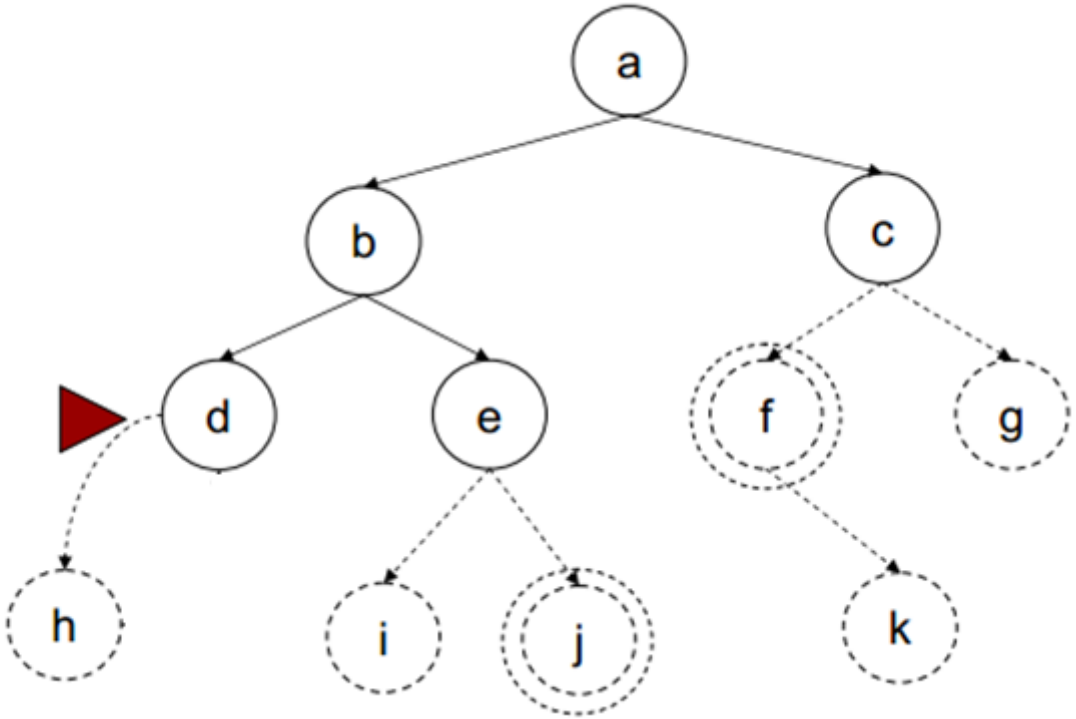
Inserir na frente, remover da frente: a

Busca em profundidade

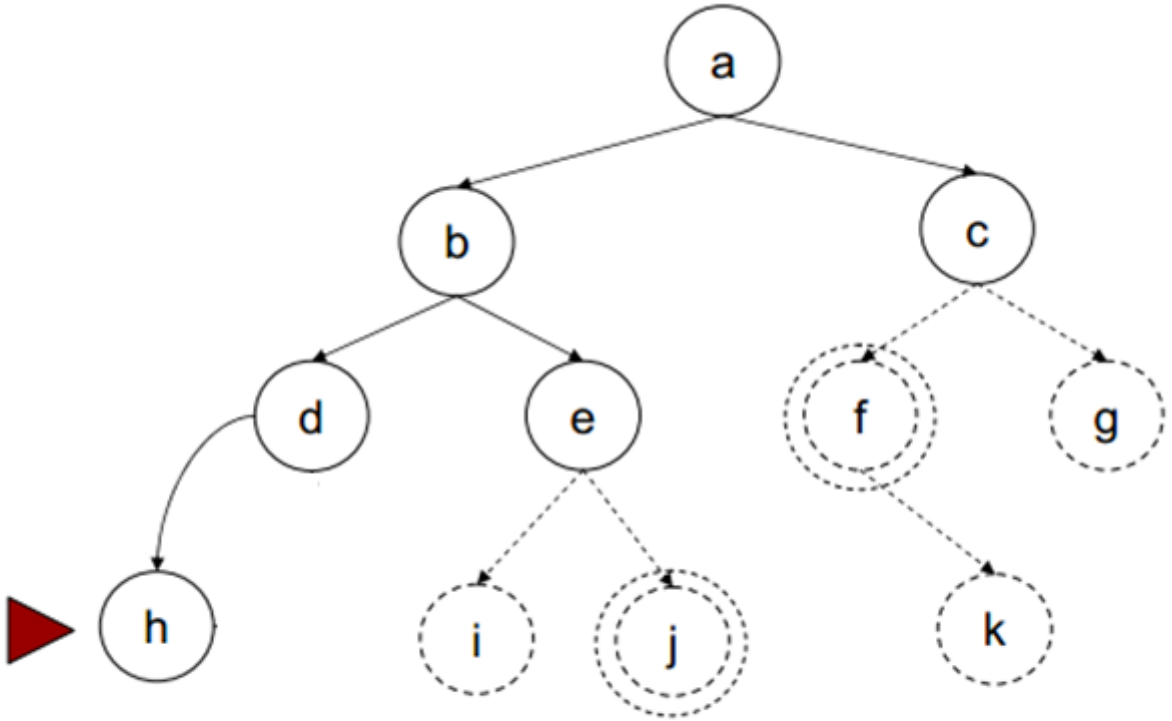


Inserir na frente, remover da frente: b, c

Busca em profundidade

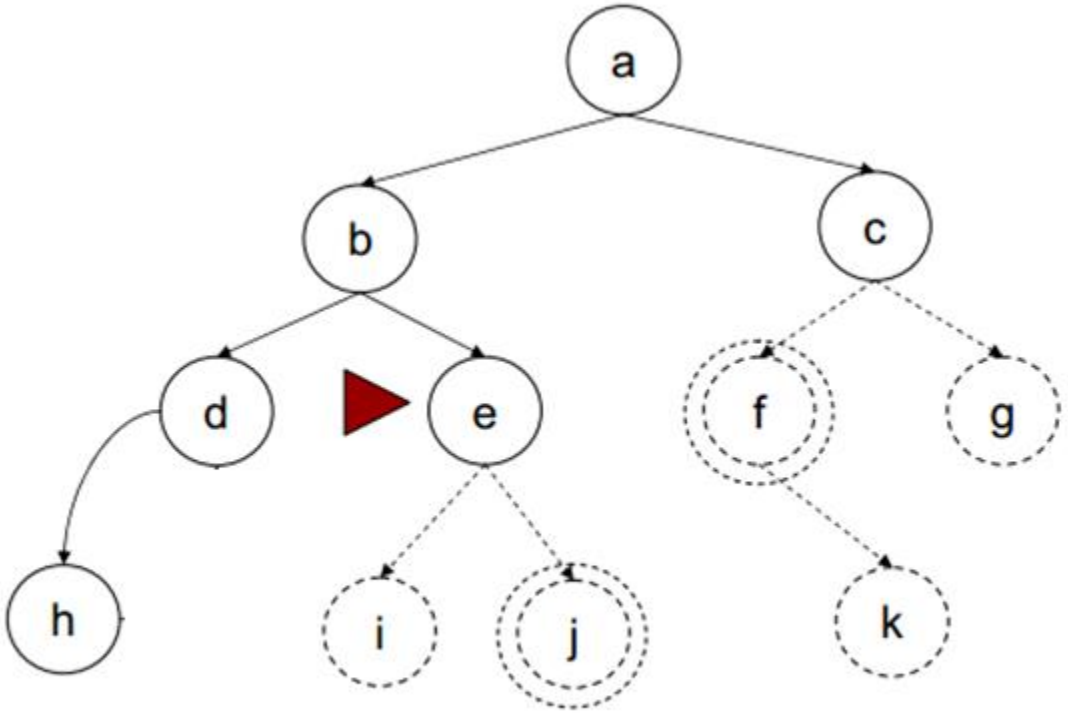


Busca em profundidade



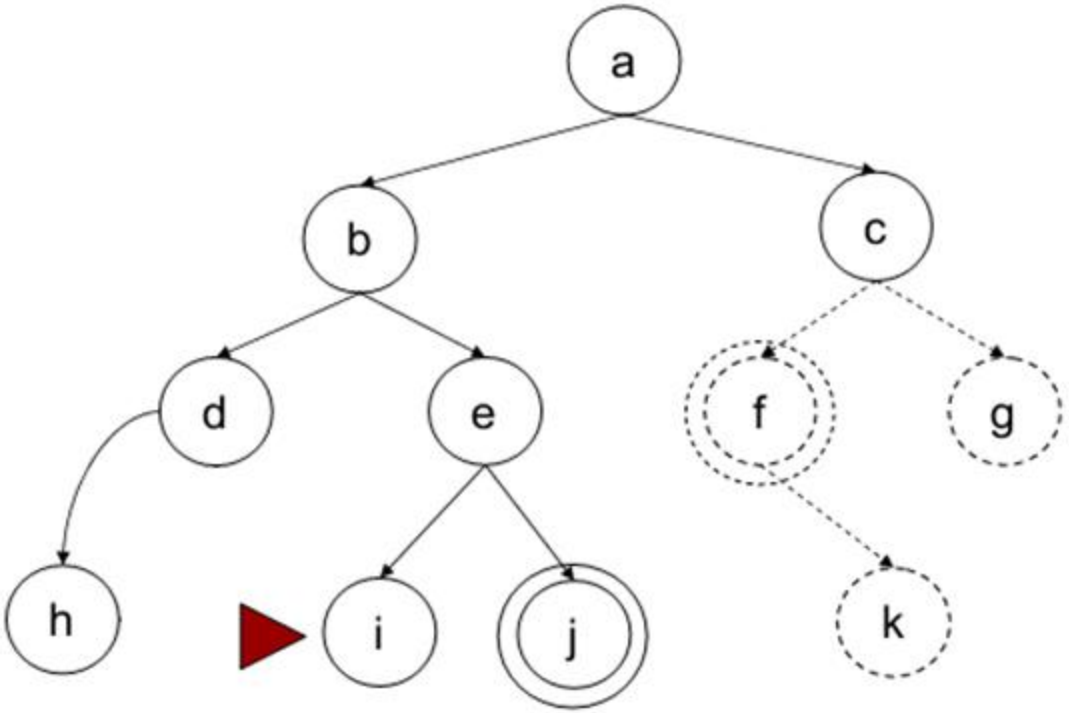
Inserir na frente, remover da frente: h, e, c

Busca em profundidade



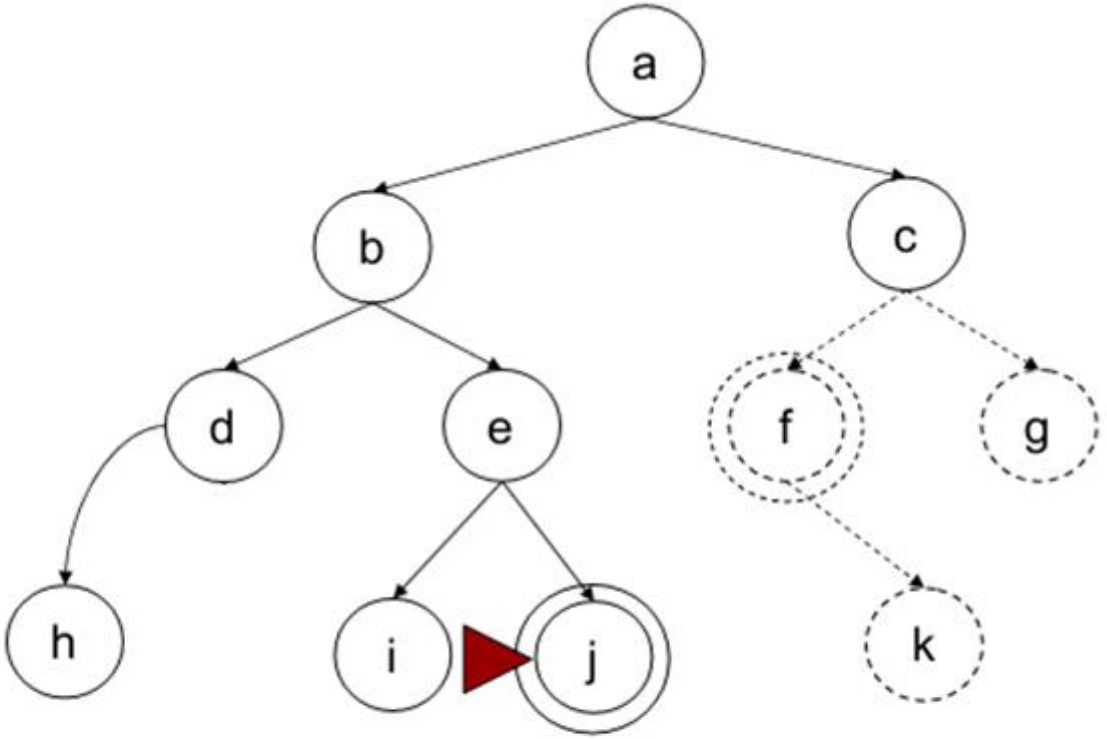
Inserir na frente, remover da frente: e, c

Busca em profundidade



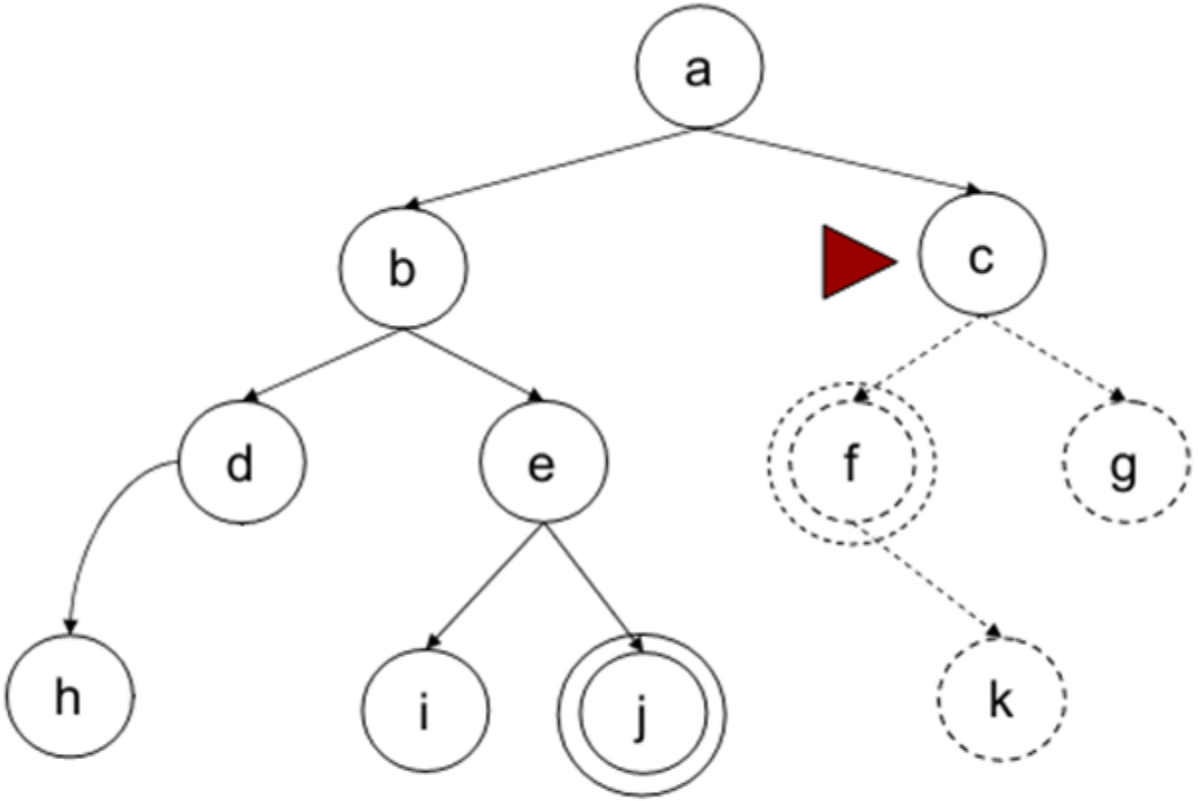
Inserir na frente, remover da frente: i, j, c

Busca em profundidade



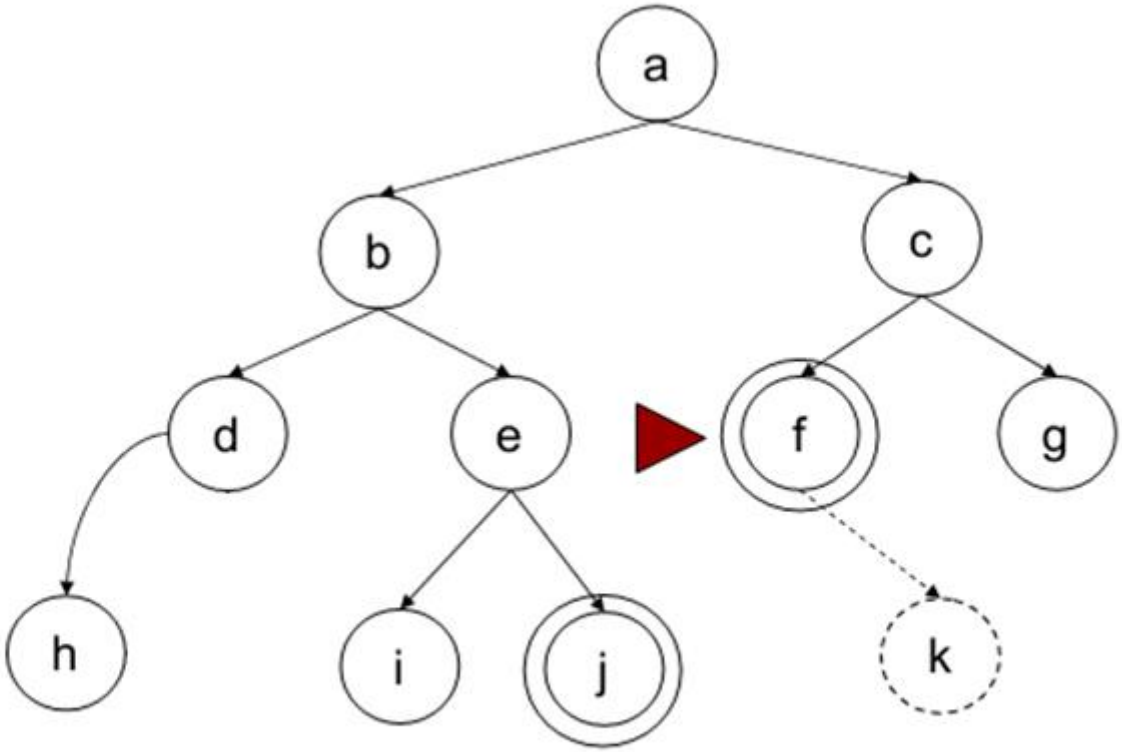
Inserir na frente, remover da frente: j, c

Busca em profundidade



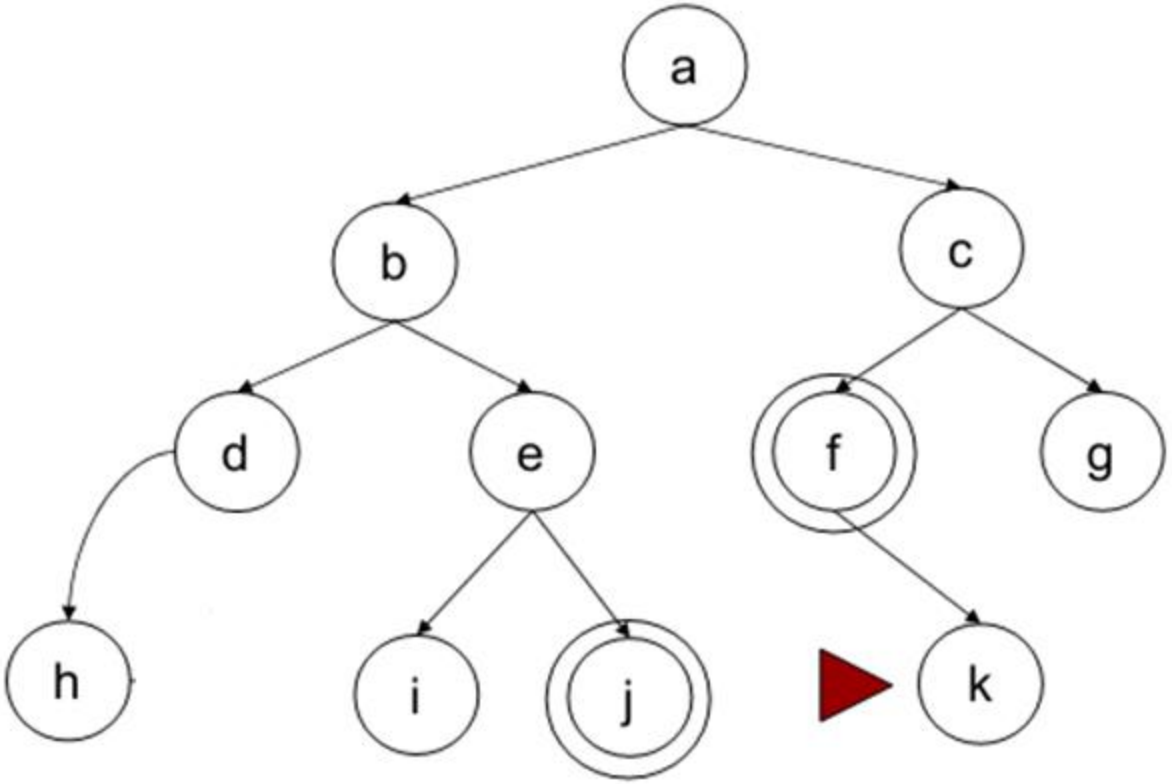
Inserir na frente, remover da frente: c

Busca em profundidade



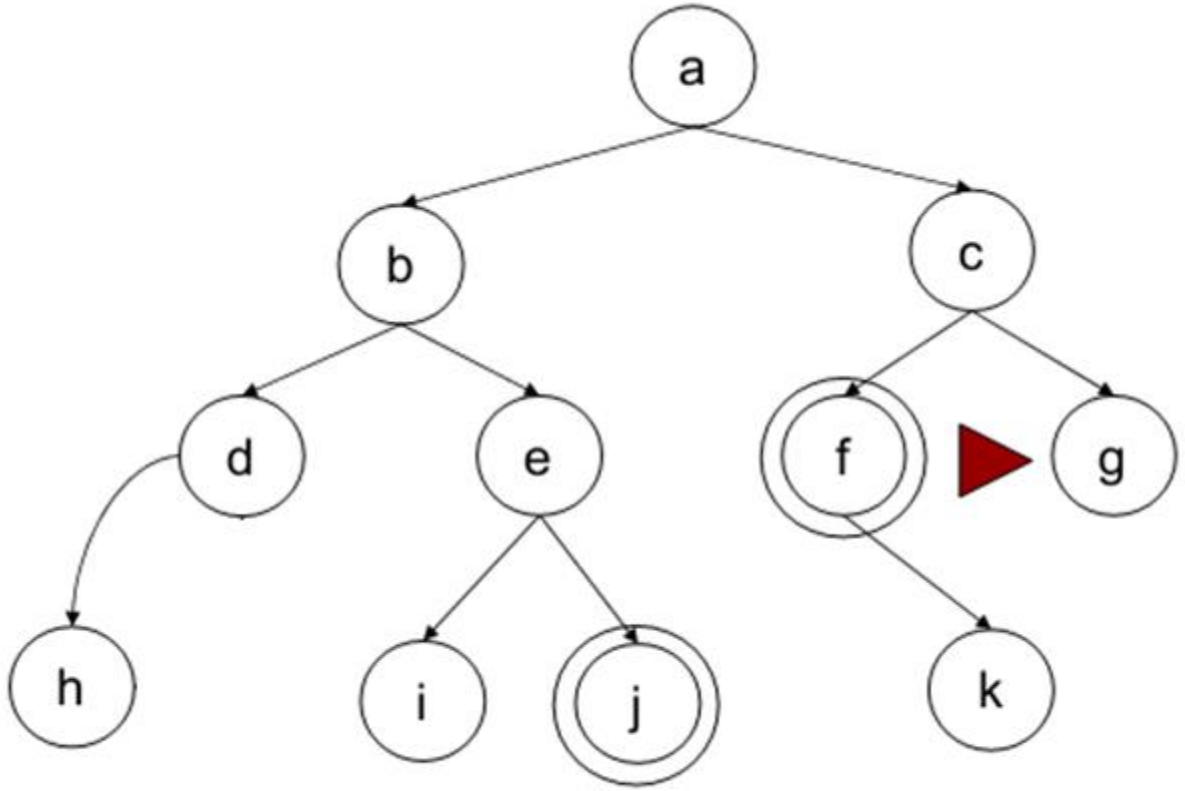
Inserir na frente, remover da frente: f, g

Busca em profundidade



Inserir na frente, remover da frente: k, g

Busca em profundidade



Inserir na frente, remover da frente: g

Busca em profundidade - Resumo

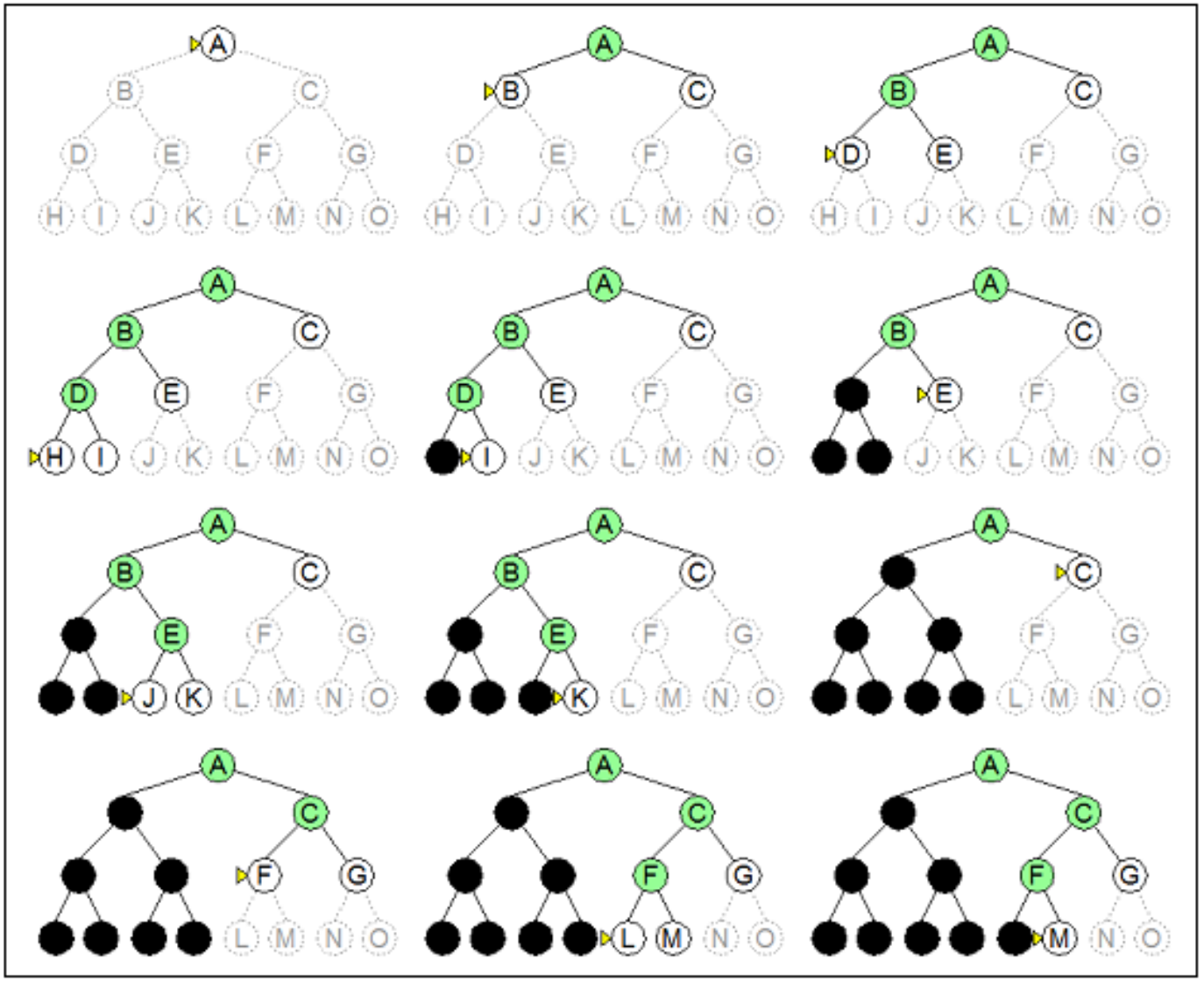
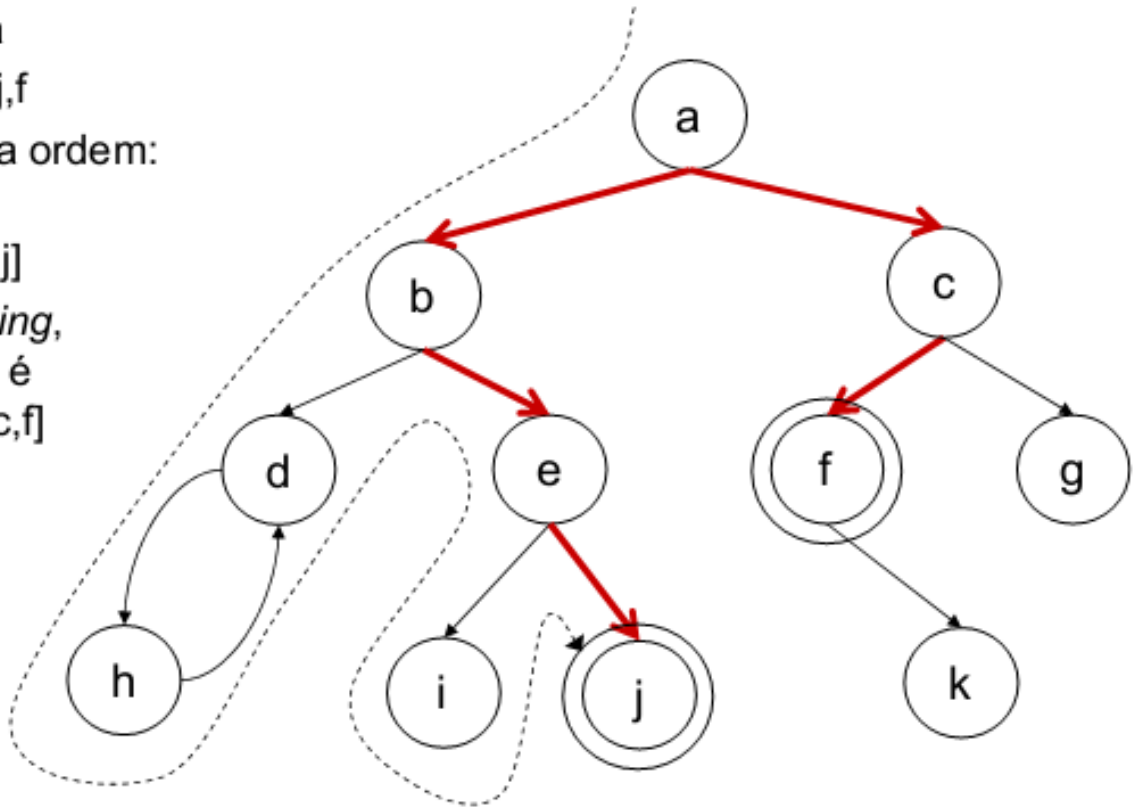


Figura 2.6 – Busca em Profundidade. Os nós que foram expandidos e não têm descendentes na borda podem ser removidos da memória; eles são mostrados em cor preta. Supomos que os nós na profundidade 3 não têm sucessores e que M é o único nó objetivo (RUSSELL, 2003).

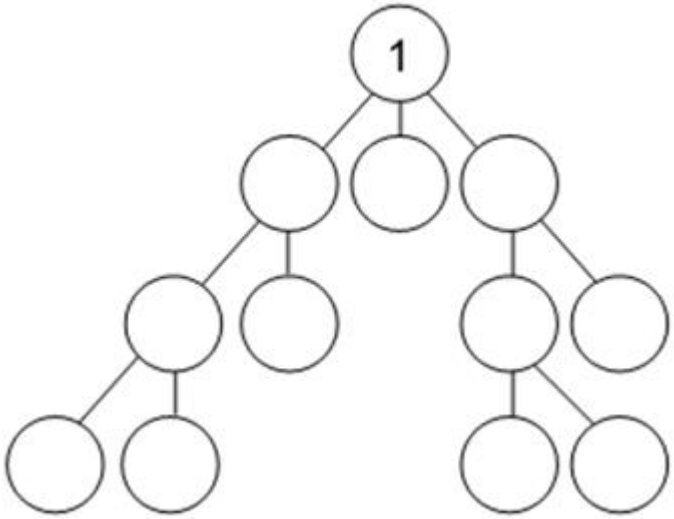
Busca em profundidade

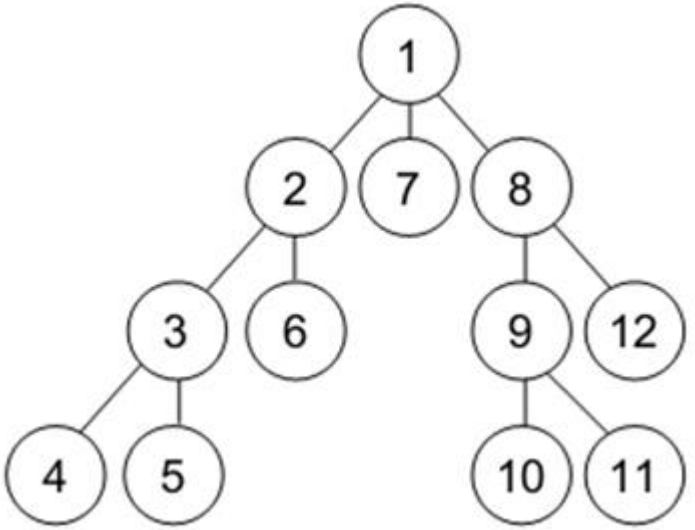
- Estado inicial: a
- Estados finais: j,f
- Nós visitados na ordem: a,b,d,h,e,i,j
- Solução: [a,b,e,j]
- Após *backtracking*, a outra solução é encontrada: [a,c,f]



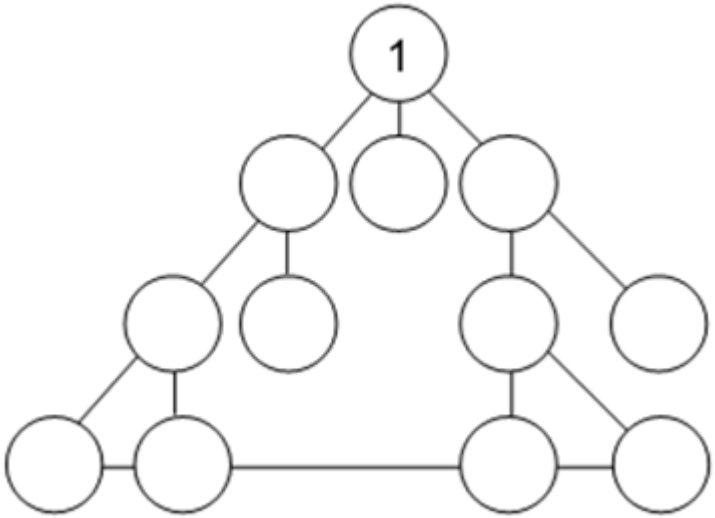
Exercício: Indique a ordem na qual os nós são visitados na busca em profundidade

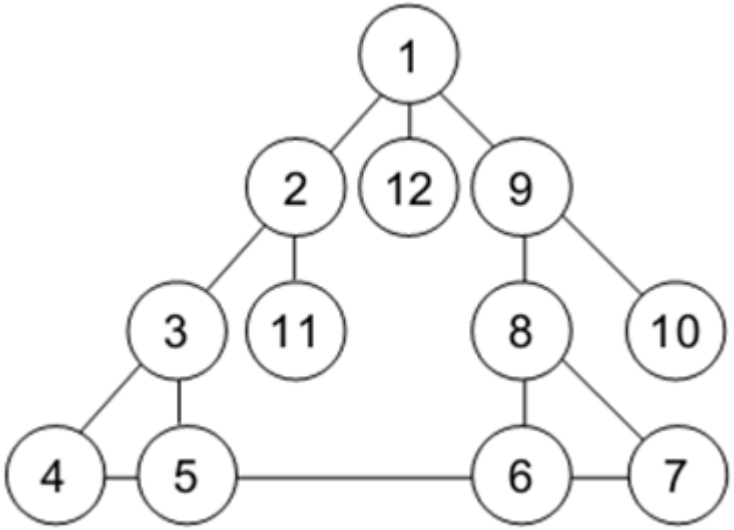
Assuma que os sucessores são definidos da esquerda para a direita e, depois de cima para baixo



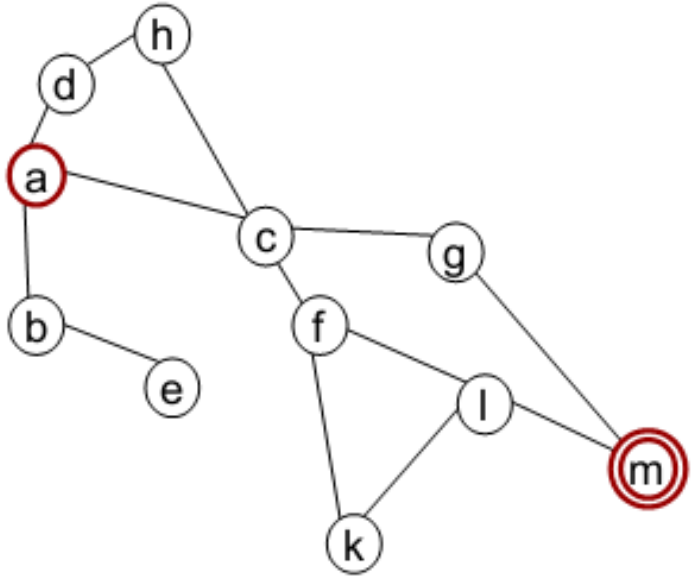


Exercício: Indique a ordem na qual os nós são visitados na busca em profundidade





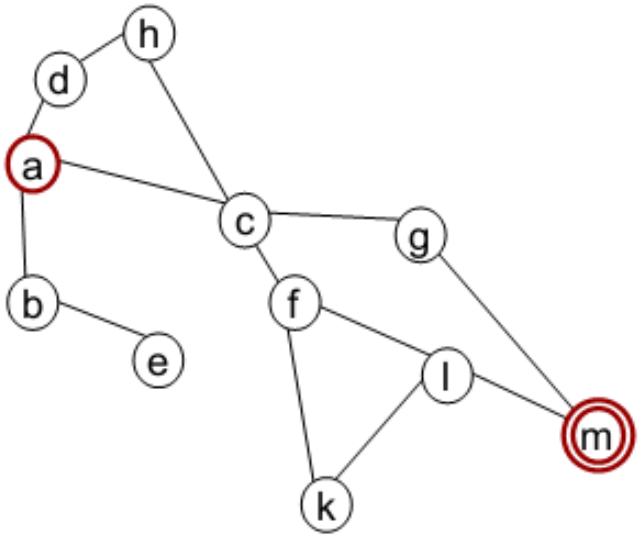
Exercício

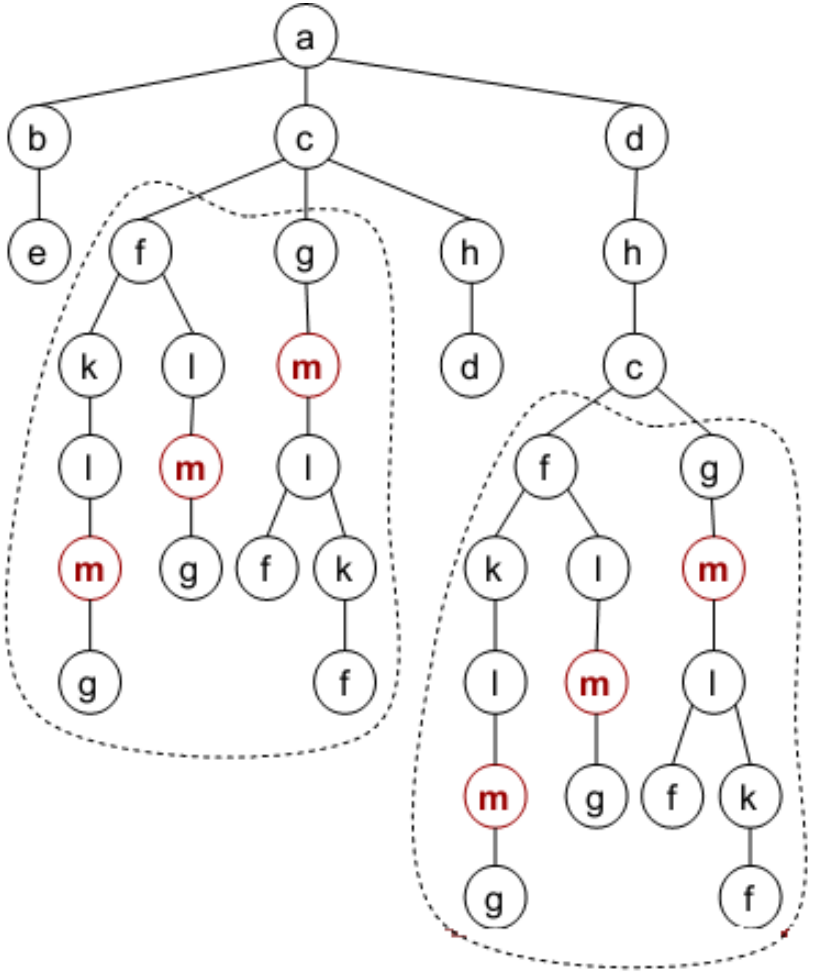


Assuma que os sucessores de um nó são definidos em ordem alfabética

- Mostre a árvore de busca definida pelo algoritmo de busca em profundidade, partindo de **a** e chegando até **m**
- Mostre também os caminhos encontrados na ordem em que são encontrados pela busca em profundidade

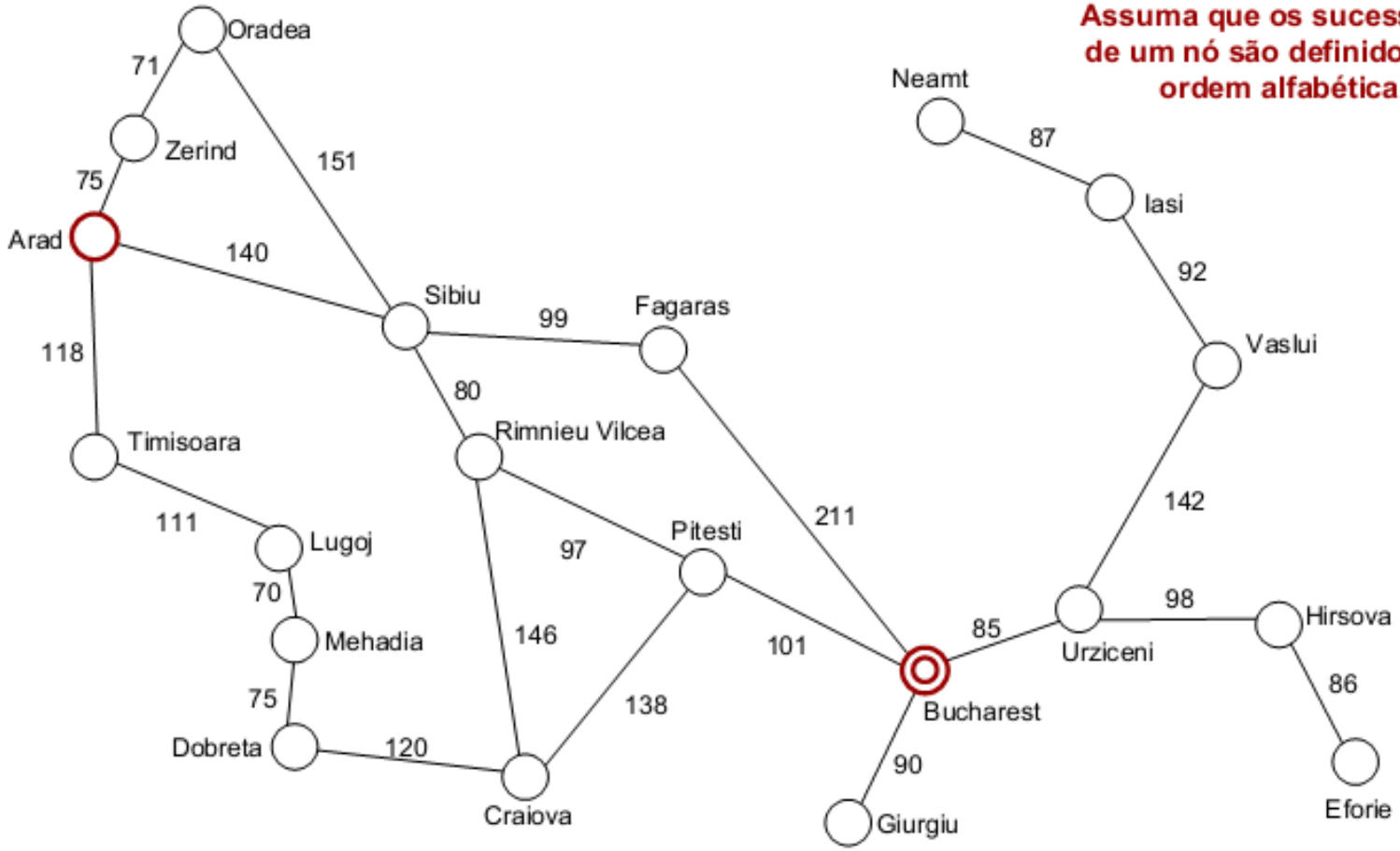
Exercício



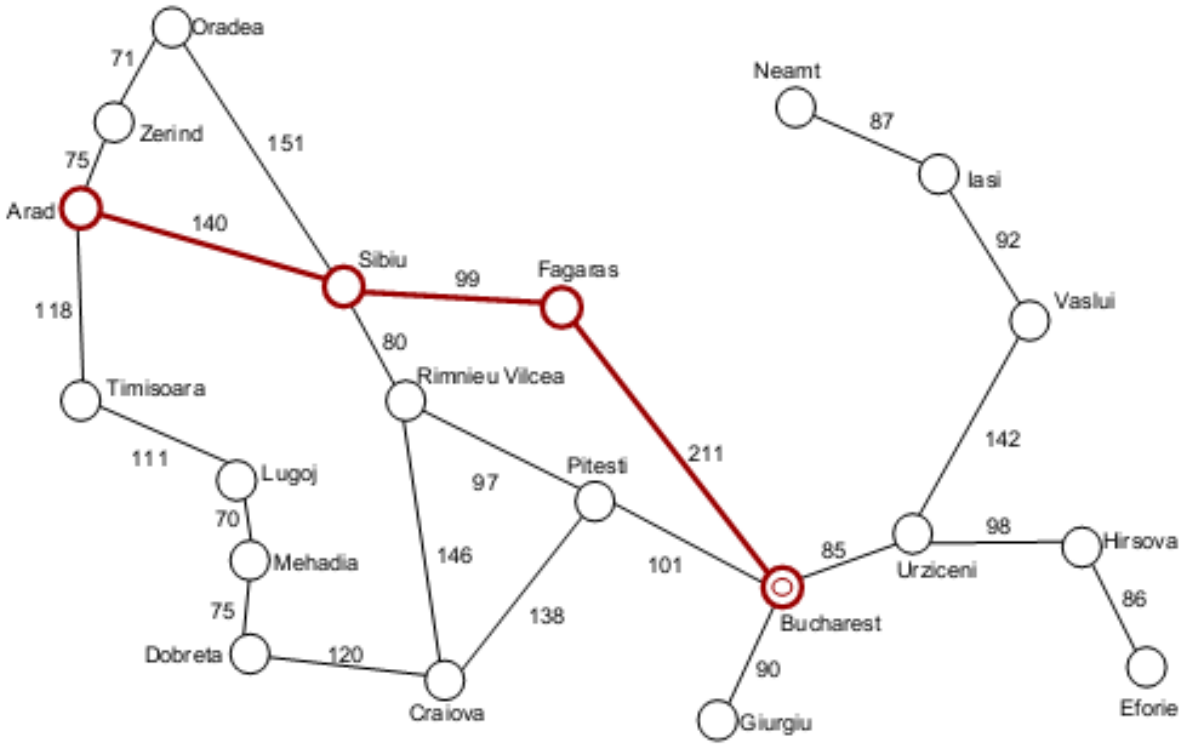


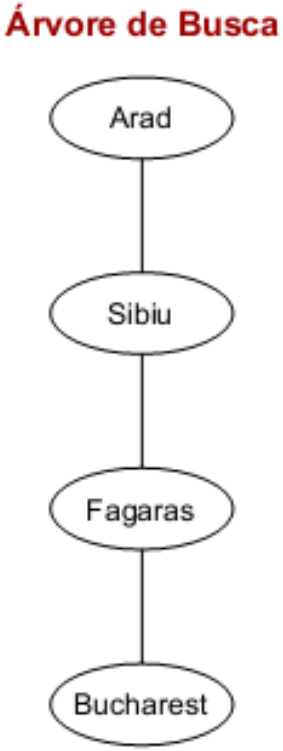
Caminhos encontrados:
[a,c,f,k,l,m]
[a,c,f,l,m]
[a,c,g,m]
[a,d,h,c,f,k,l,m]
[a,d,h,c,f,l,m]
[a,d,h,c,g,m]

Encontre o Caminho e Árvore de Busca de Arad até Bucharest usando Busca em Profundidade



Encontre o Caminho e Árvore de Busca de Arad até Bucharest usando Busca em Profundidade





Busca em profundidade

- Um problema com a busca em profundidade é que existem espaços de estados nos quais o algoritmo se perde

- Muitos espaços de estado são infinitos e, nesse caso, o algoritmo de busca em profundidade pode perder um nó final, prosseguindo por um caminho infinito no grafo

- O algoritmo então explora esta parte infinita do espaço, nunca chegando perto de um nó final

- Para evitar caminhos infinitos (sem ciclos), um refinamento pode ser adicionado à busca em profundidade: limitar a profundidade de busca

Algoritmo

Algoritmo:

função Busca-em-Profundidade (*problema*)

retorna uma solução ou falha

Busca-Genérica (*problema*, Insere-no-Começo)

Busca em Profundidade com Pilha

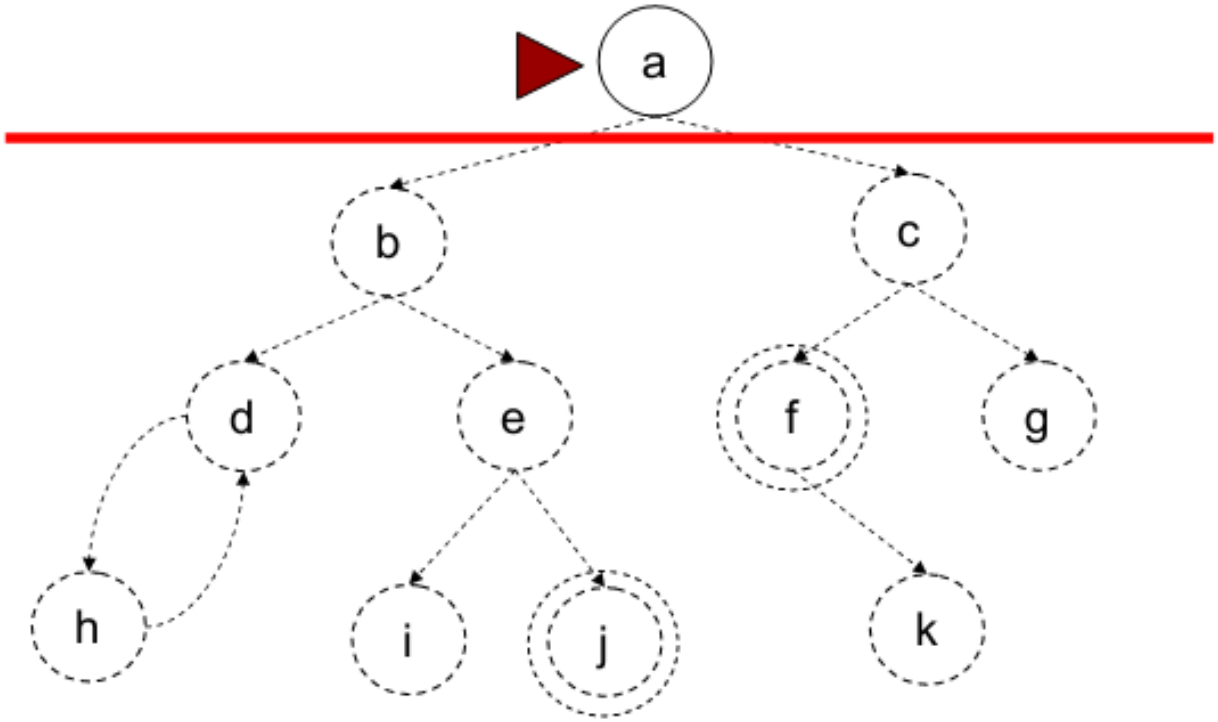
```
function Busca_Profundidade(Inicio, Alvo)
{
   empilha(Inicio)
   while nao pilhaVazia()
   {
      no = desempilha()
      foiVisitado(no)
      if no == Alvo
      {
         return no
      }
      for each Filho in Expande(no)
      {
          if nao Visitado(Filho)
          {
             empilha(Filho)
          }
      }
   }
}
```

Busca em Profundidade com recursão

```
function Busca_Profundidade(Inicio, Alvo)
{
      foiVisitado(Inicio)
      if Inicio == Alvo
      {
         return Inicio
      }
      for each Filho in Expande(Inicio)
      {
          if nao Visitado(Filho)
          {
             Busca_Profundidade(Filho, Alvo)
          }
      }
   }
}
```

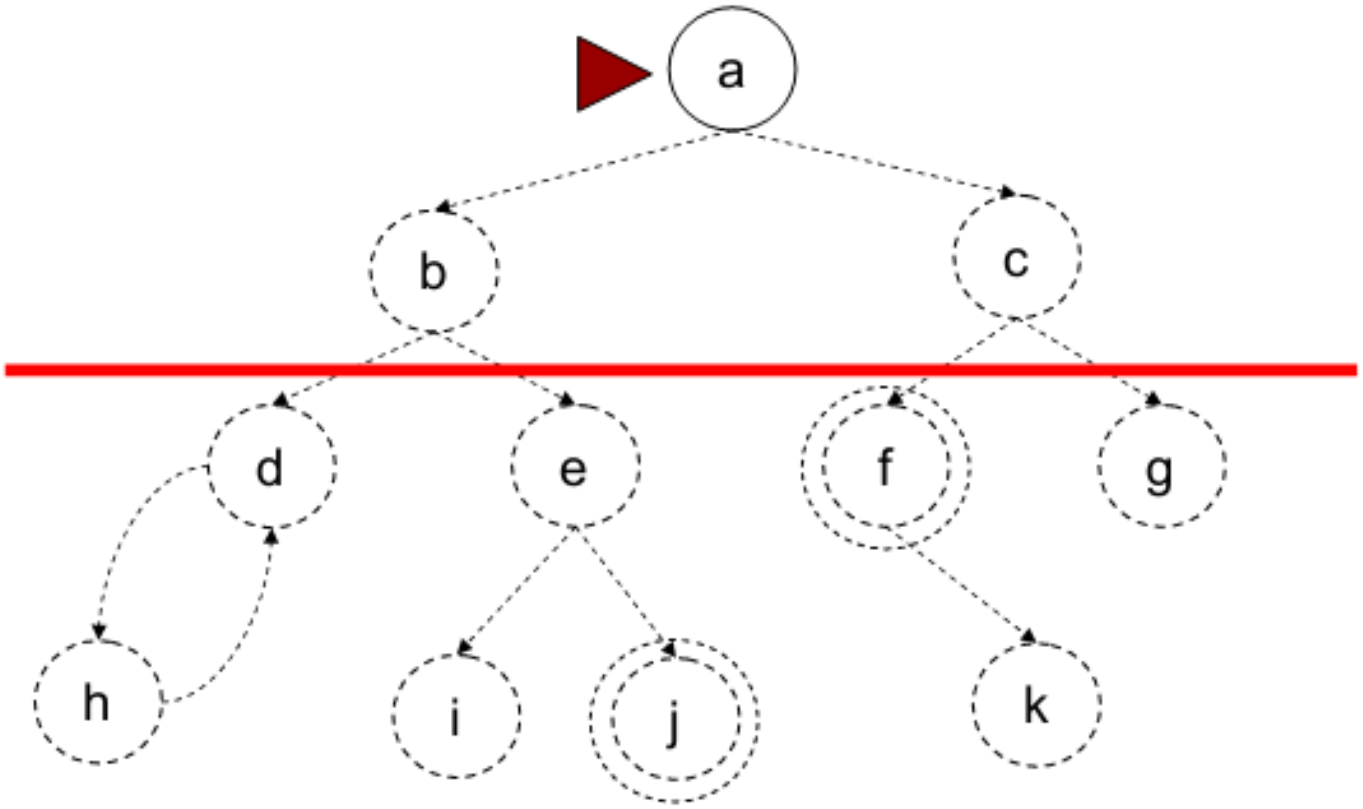
Busca em Profundidade Limitada

Busca em profundidade limitada (l=0)



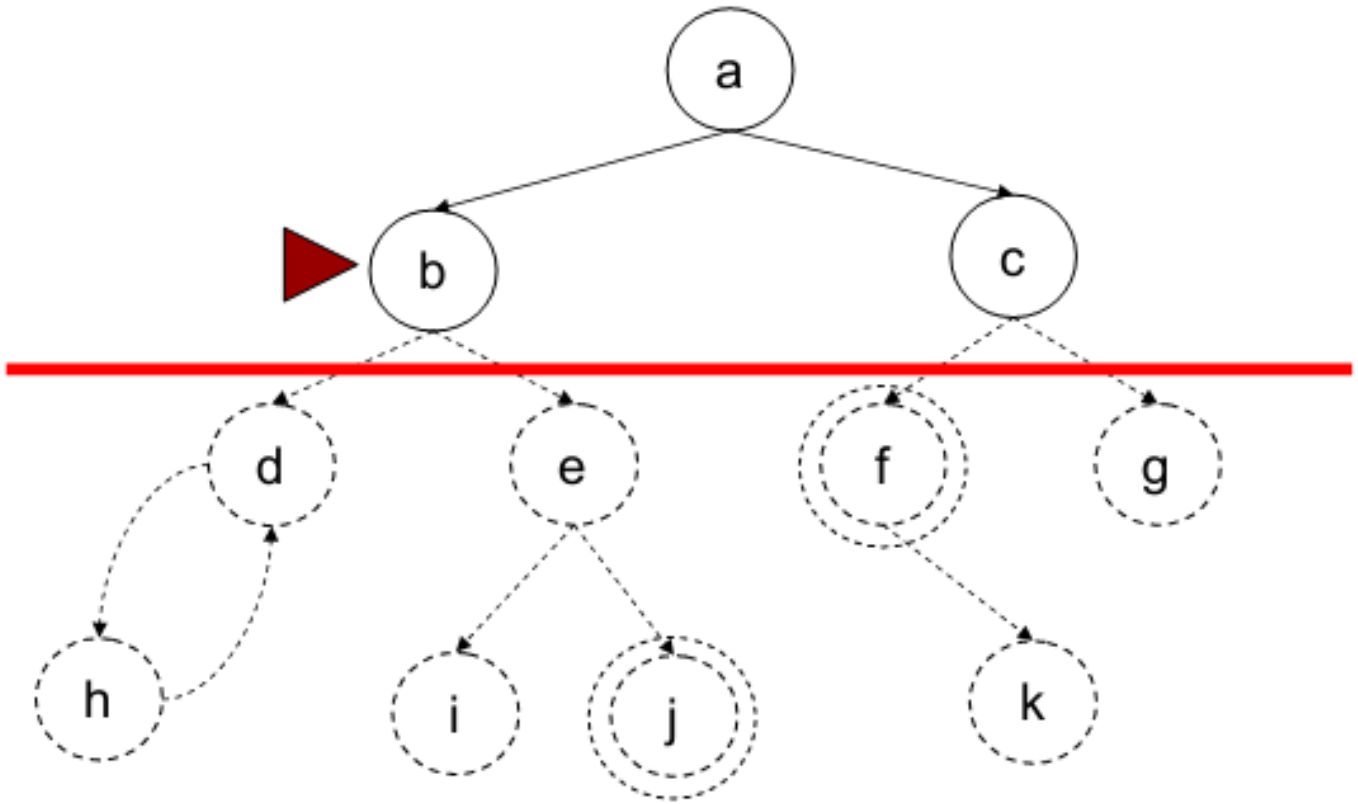
Inserir na frente, remover da frente: a

Busca em profundidade limitada (l=1)



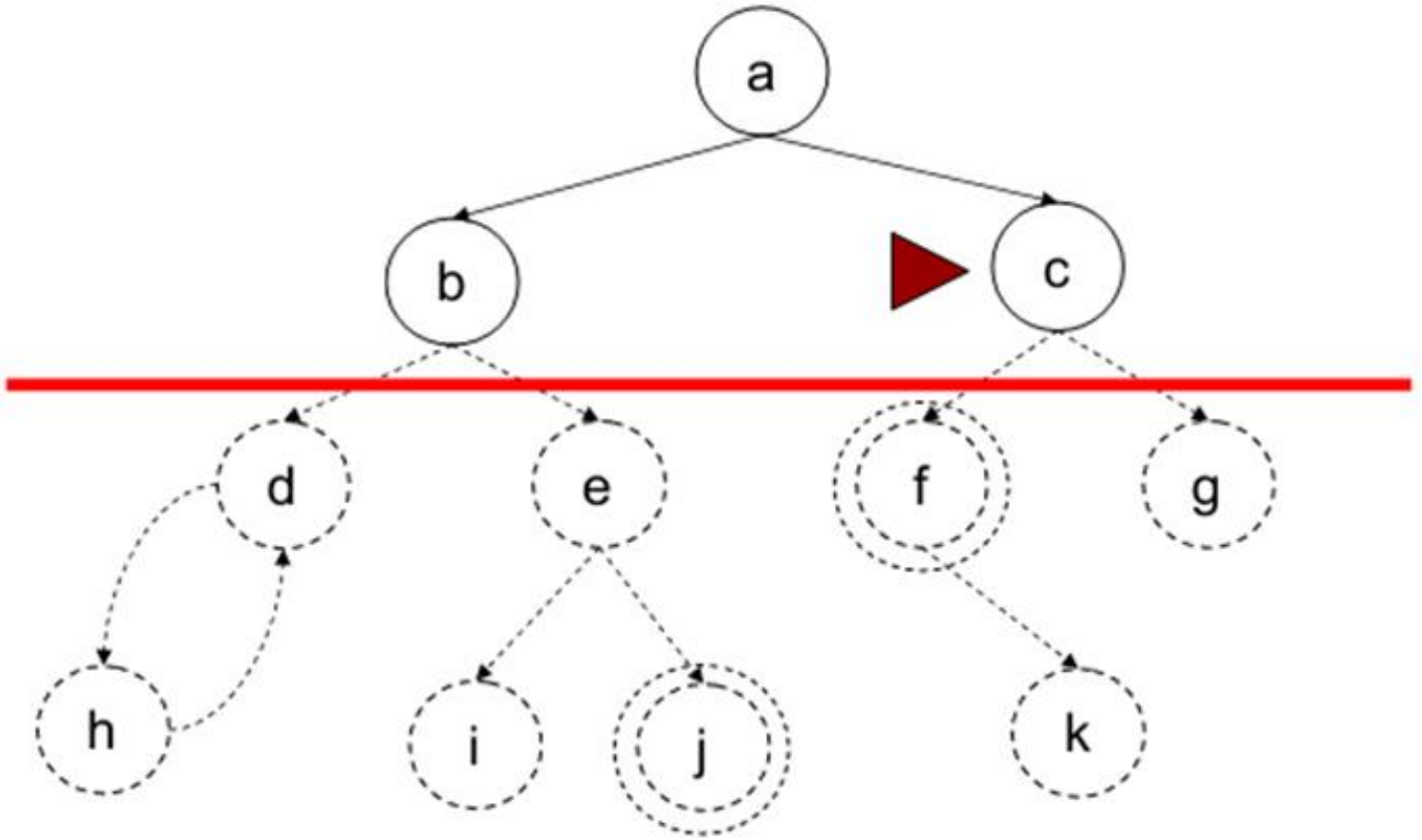
Inserir na frente, remover da frente: a

Busca em profundidade limitada (l=1)



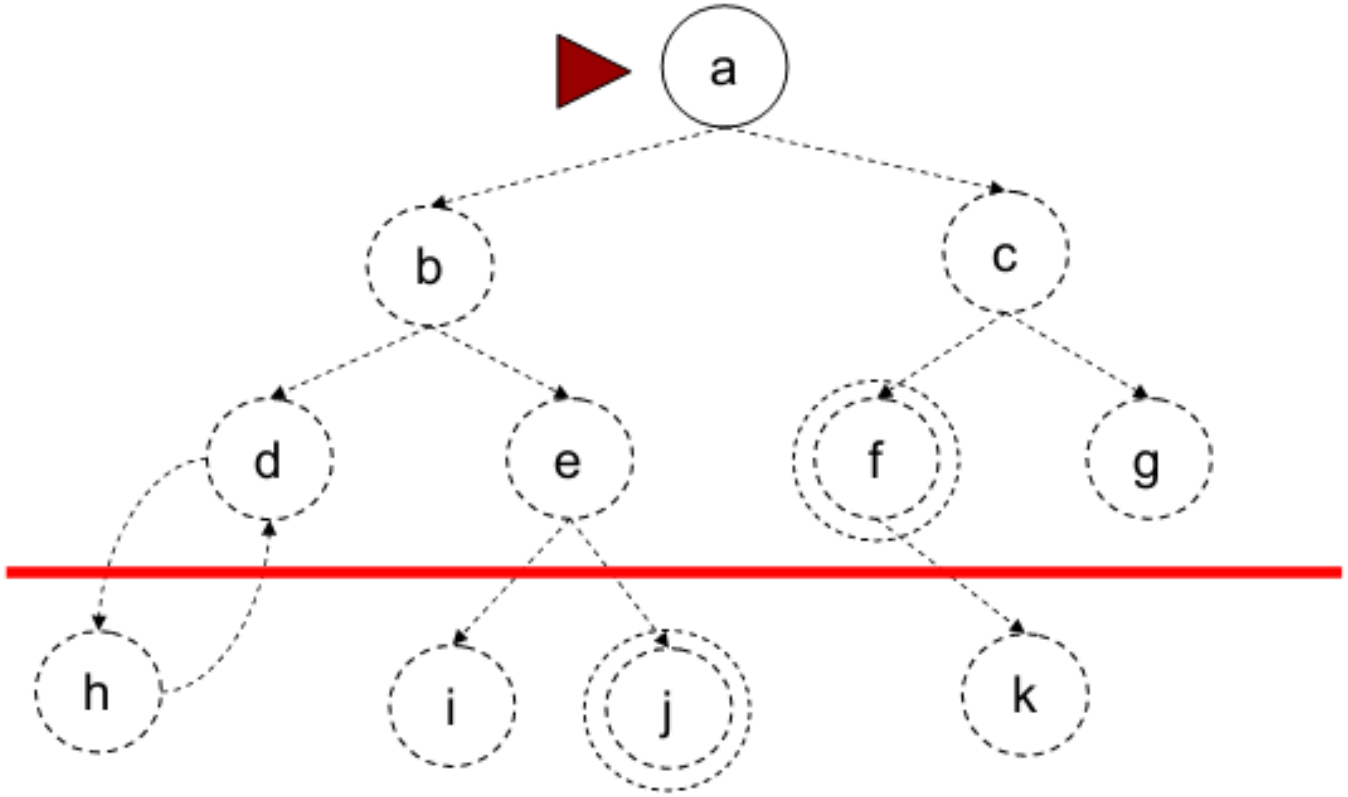
Inserir na frente, remover da frente: b, c

Busca em profundidade limitada (l=1)



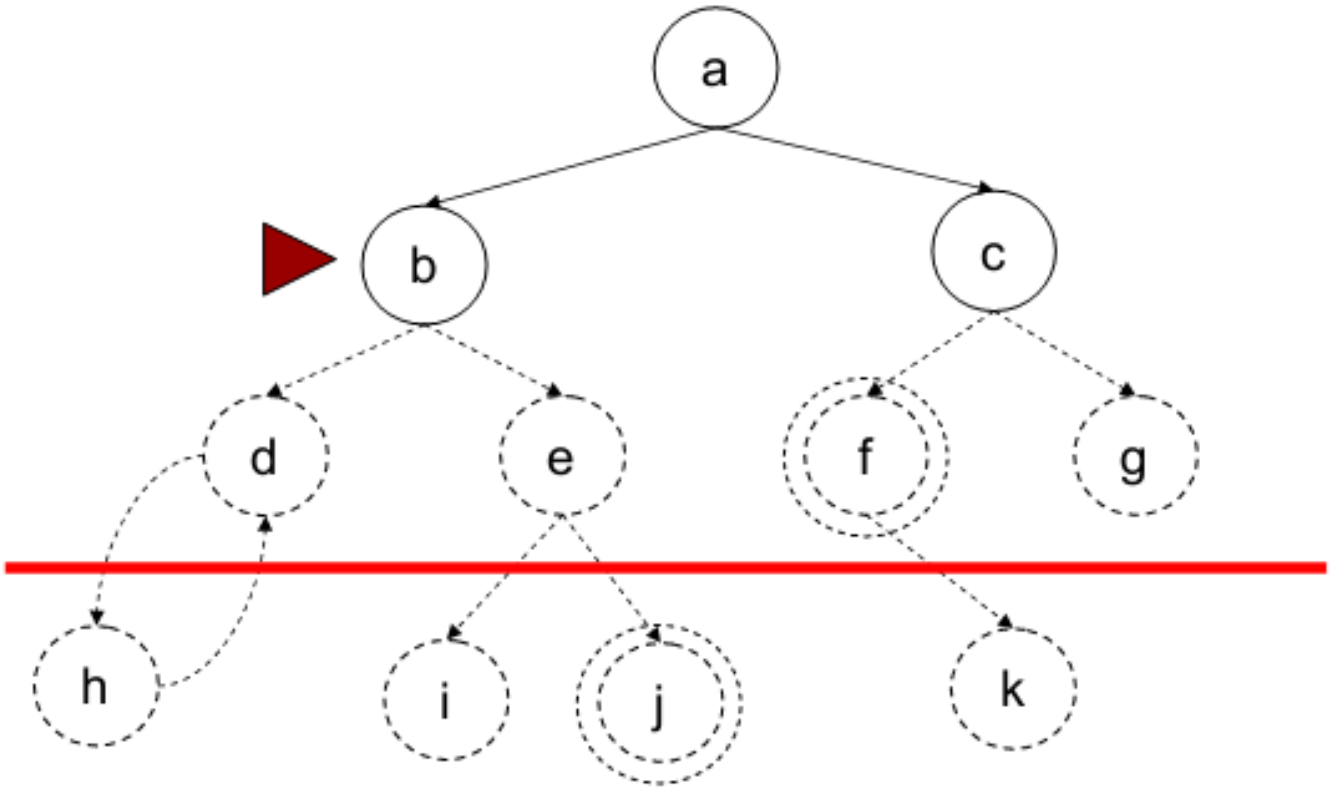
Inserir na frente, remover da frente: b, c

Busca em profundidade limitada (l=2)



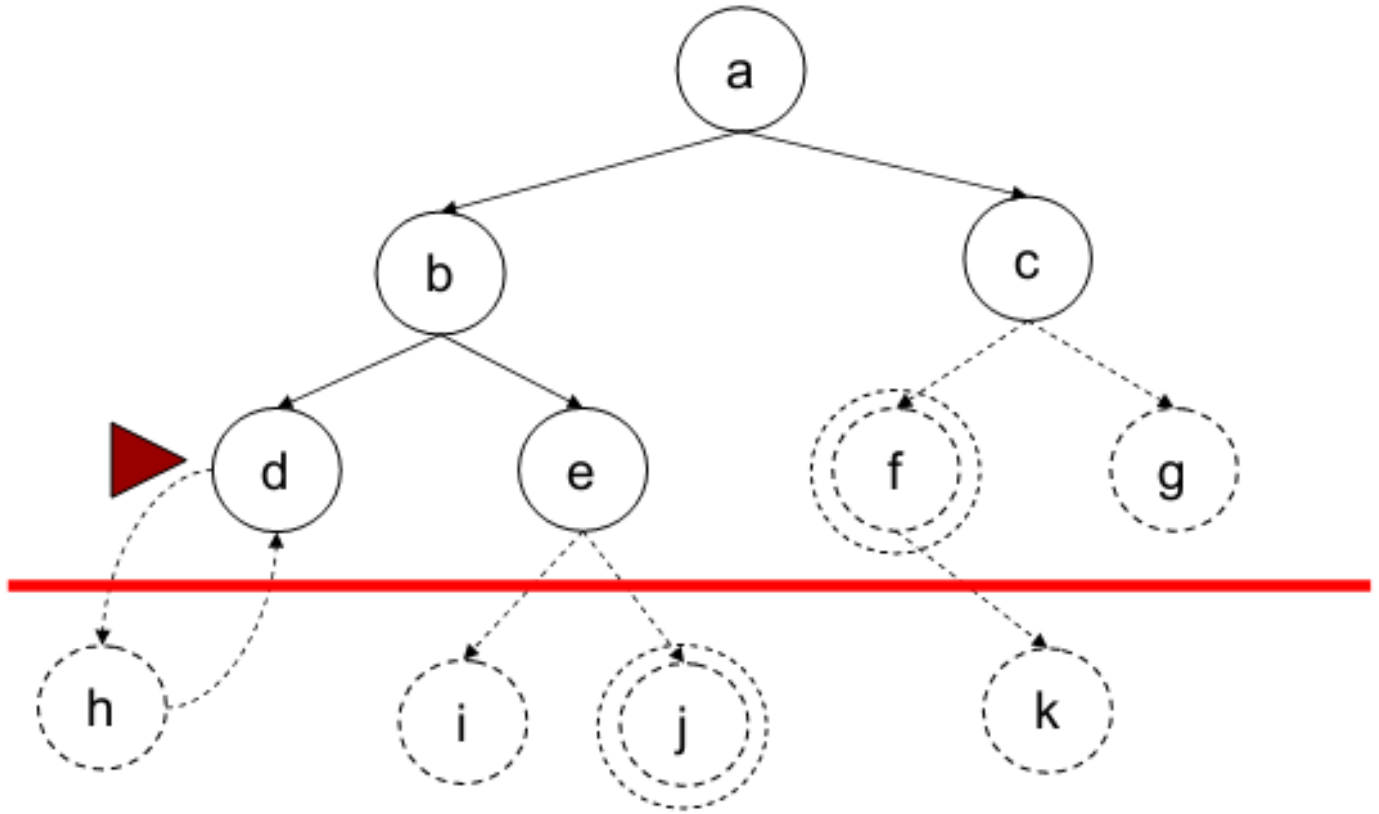
Inserir na frente, remover da frente: a

Busca em profundidade limitada (l=2)



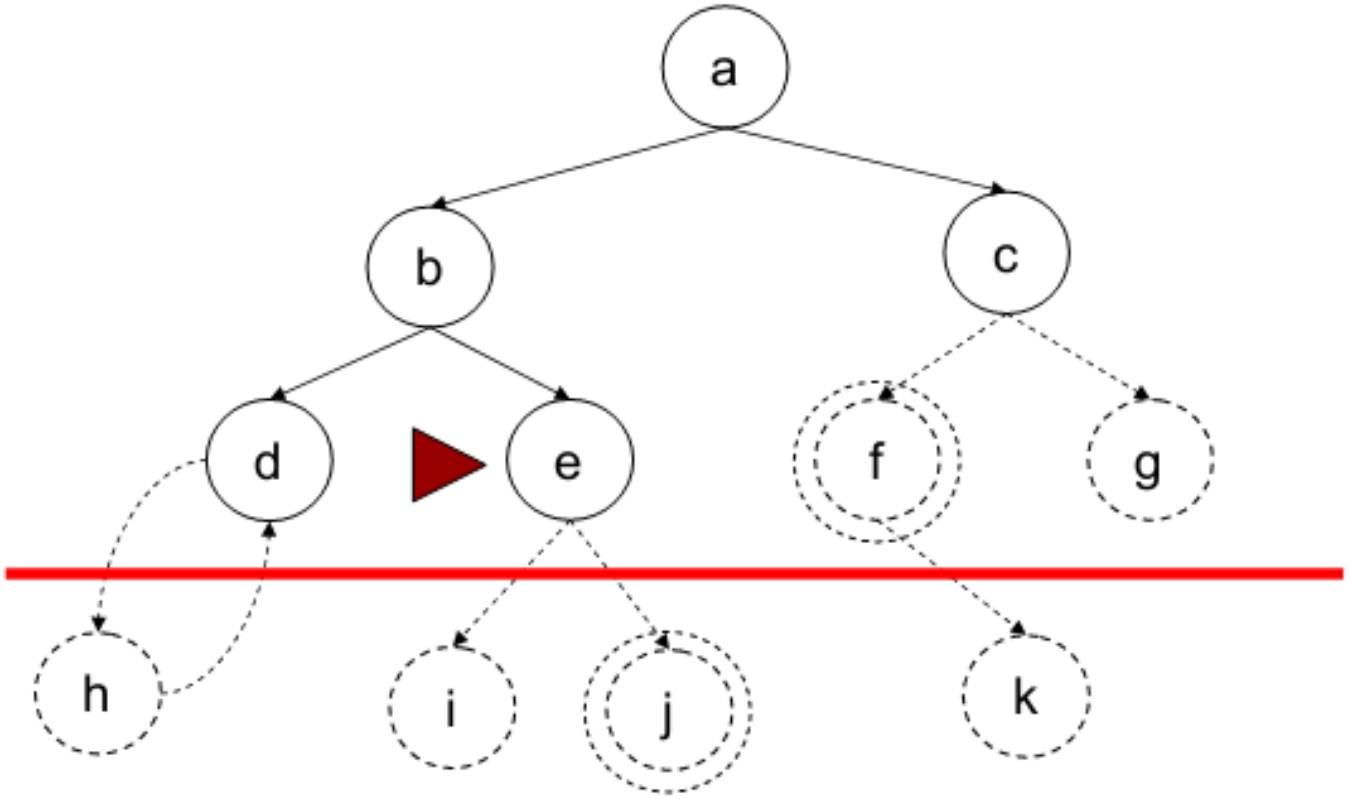
Inserir na frente, remover da frente: b, c

Busca em profundidade limitada (l=2)



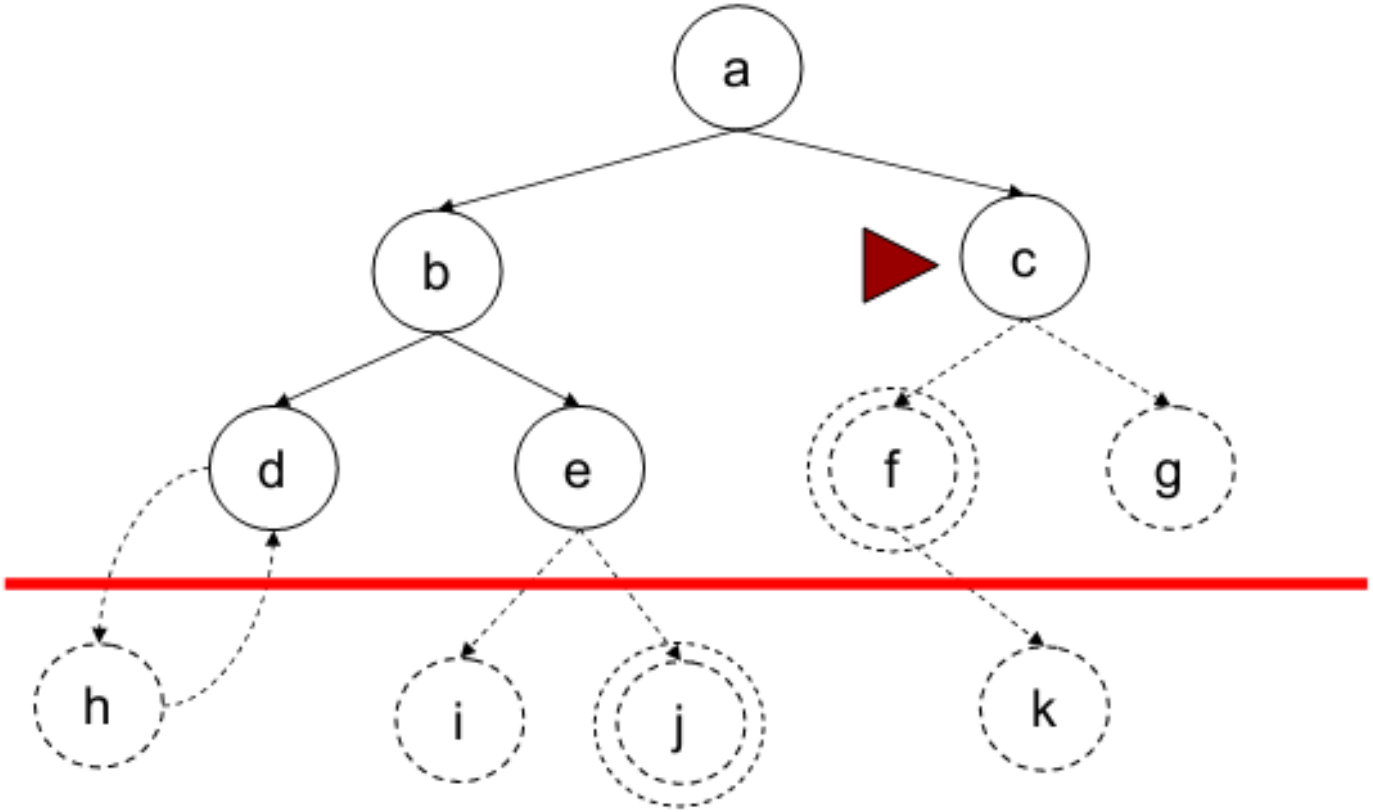
Inserir na frente, remover da frente: d, e, c

Busca em profundidade limitada (l=2)



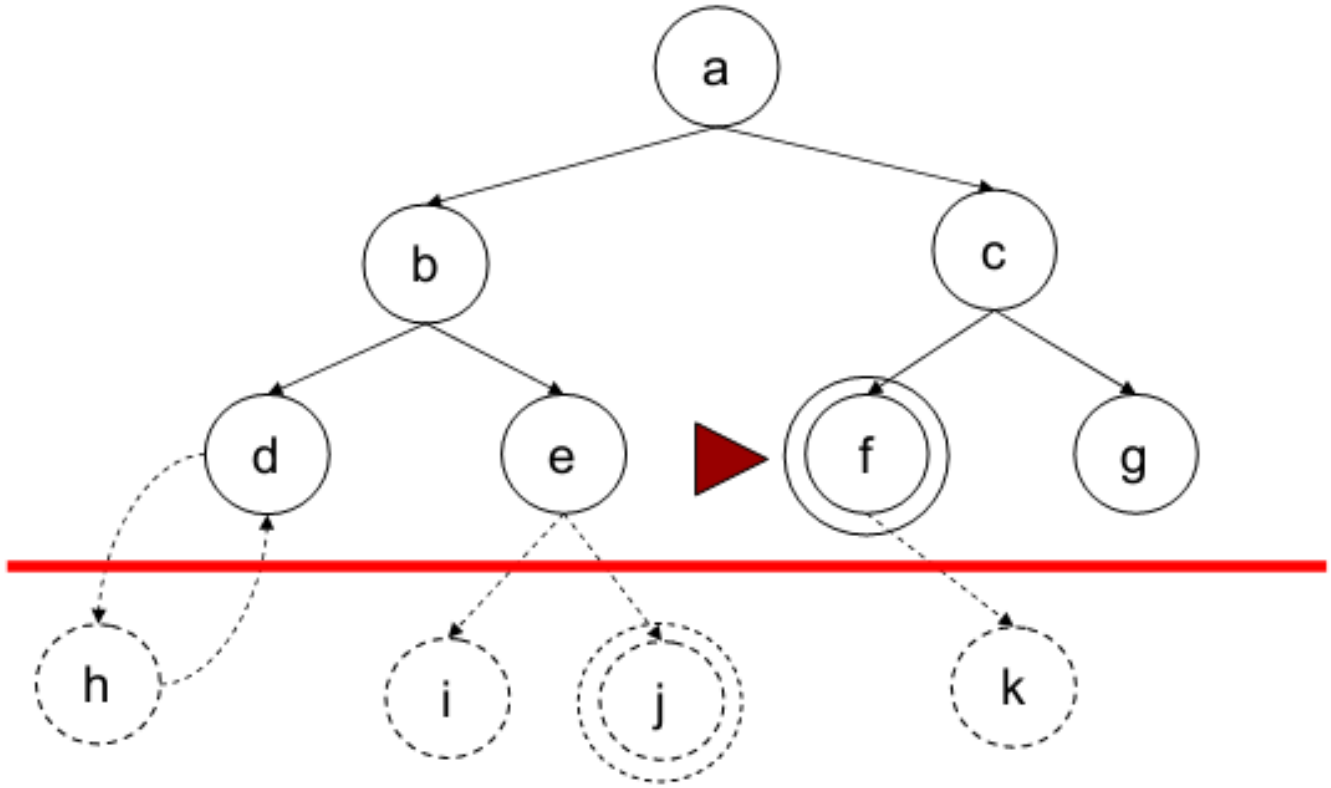
Inserir na frente, remover da frente: e, c

Busca em profundidade limitada (l=2)



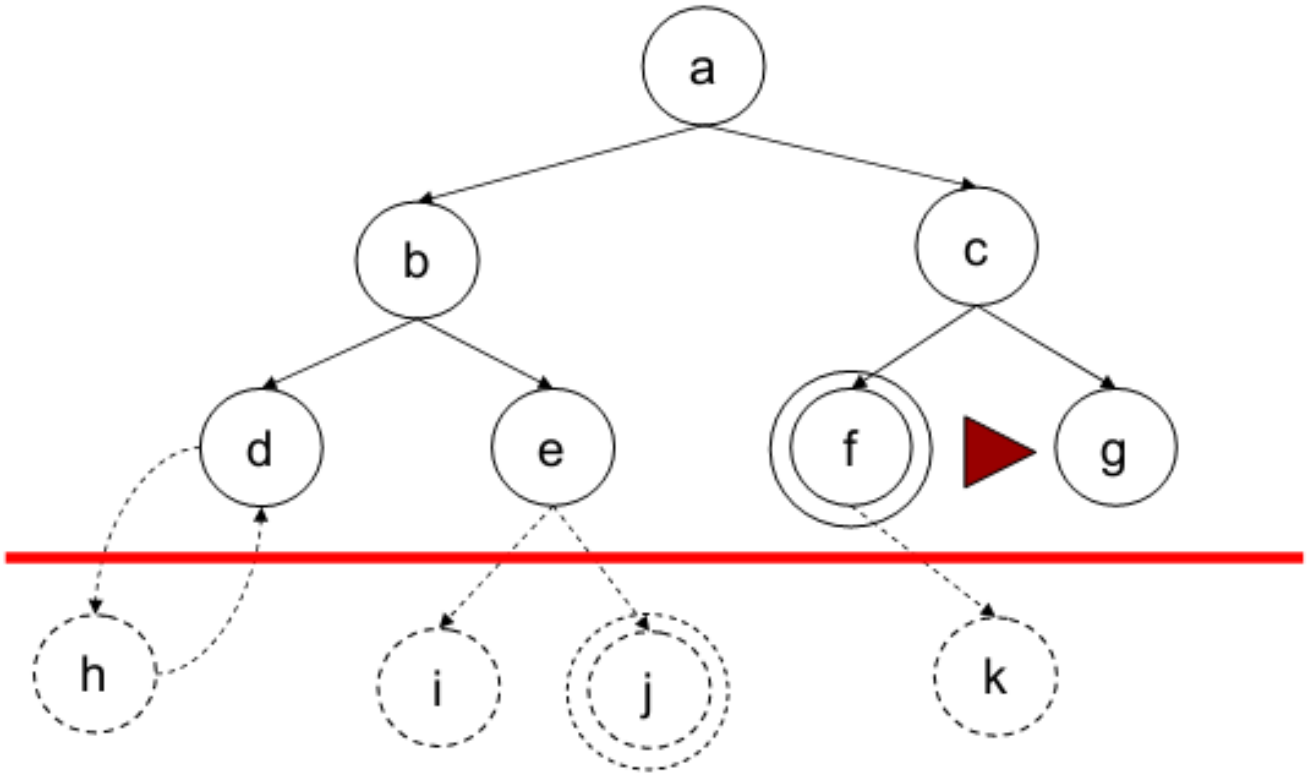
Inserir na frente, remover da frente: c

Busca em profundidade limitada (l=2)



Inserir na frente, remover da frente: f, g

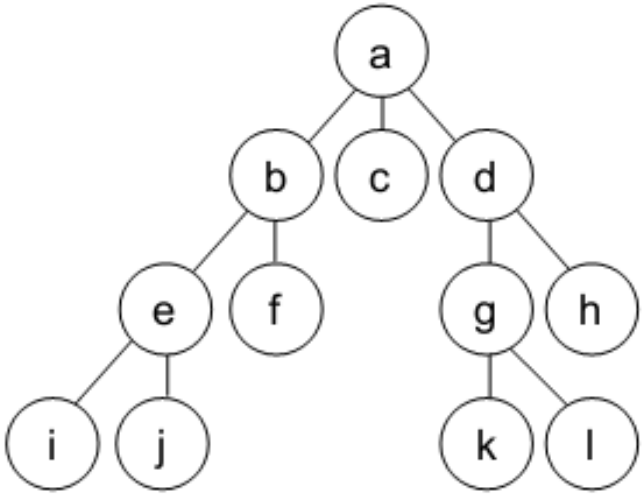
Busca em profundidade limitada (l=2)

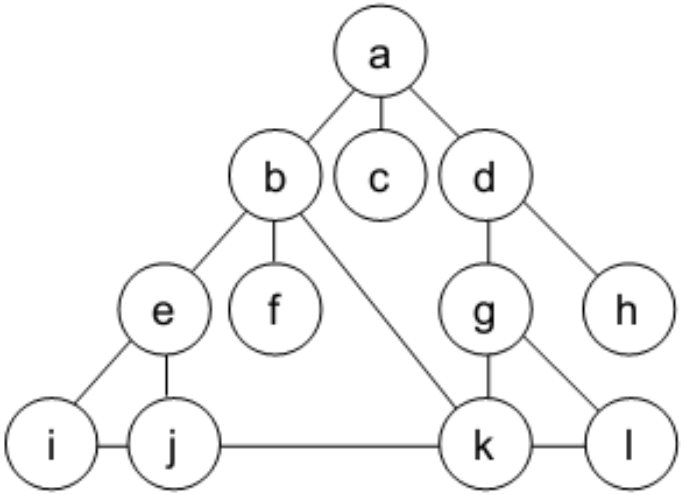


Inserir na frente, remover da frente: g

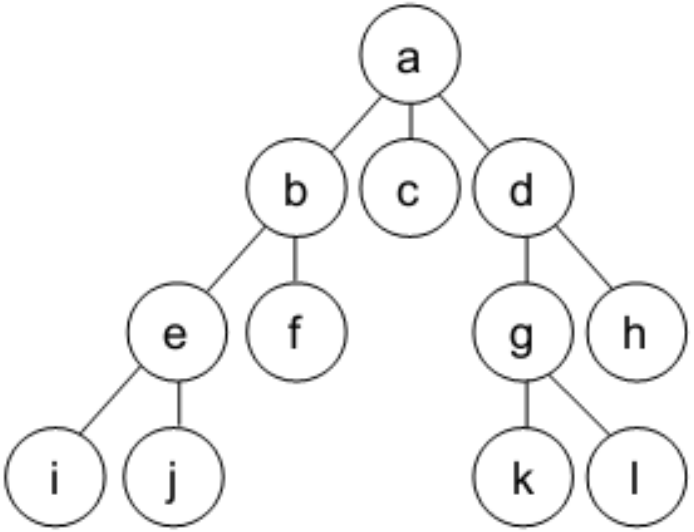
Exercício: Indique a ordem na qual os nós são visitados na busca em profundidade limitada (L = 0, 1, 2 e 3)

Assuma que os sucessores são definidos da esquerda para a direita e, depois, de cima pra baixo

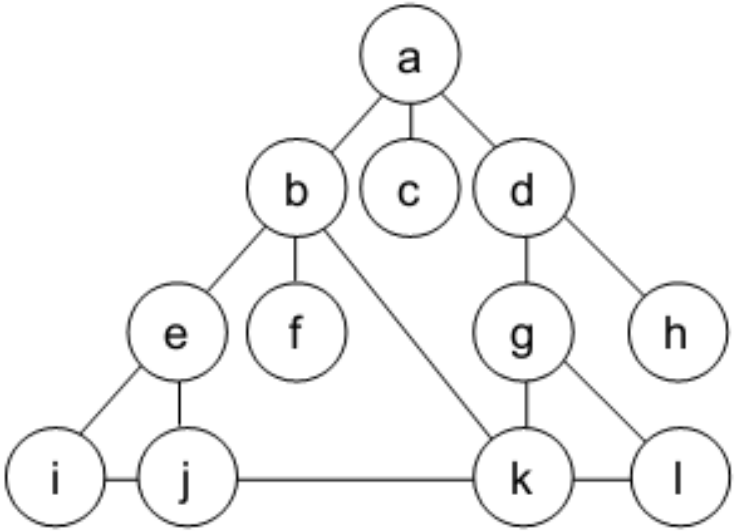




Exercício: Indique a ordem na qual os nós são visitados na busca em profundidade limitada (L = 0, 1, 2 e 3)



L=0: a
L=1: a,b,c,d
L=2: a,b,e,f,c,d,g,h
L=3: a,b,e,i,j,f,c,d,g,k,l,h



L=0: a
L=1: a,b,c,d
L=2: a,b,e,f,k,c,d,g,h
L=3: a,b,e,i,j,f,k,l,g,c,d,g,h

Busca em Profundidade Limitada

Um problema com a busca em profundidade limitada é que **não se tem previamente um limite razoável**

Se o limite for muito pequeno (menor que qualquer caminho até uma solução) então a busca falha

Se o limite for muito grande, a busca se torna muito complexa

Para resolver este problema a busca em profundidade limitada pode ser executada de forma iterativa, variando o limite: comece com um limite de profundidade pequeno e aumente gradualmente o limite até que uma solução seja encontrada

Busca em Profundidade Iterativa

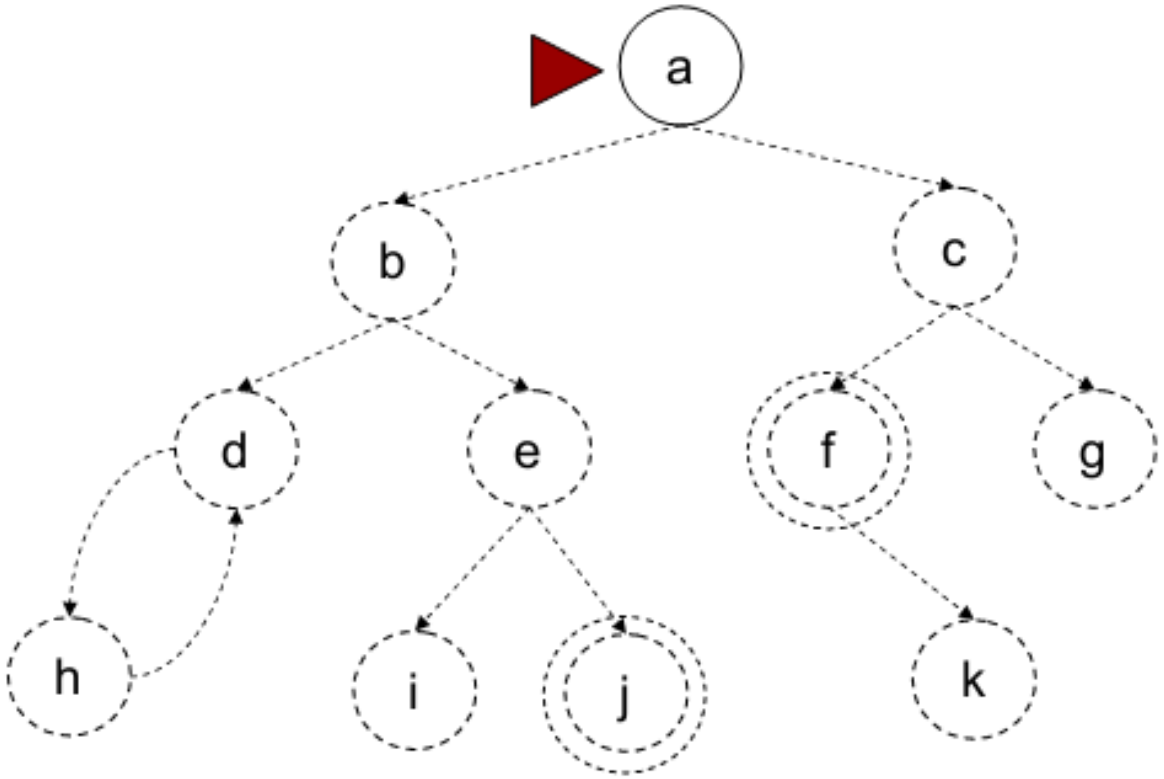
Como seria o seu funcionamento?

Busca em Largura

Busca em Largura

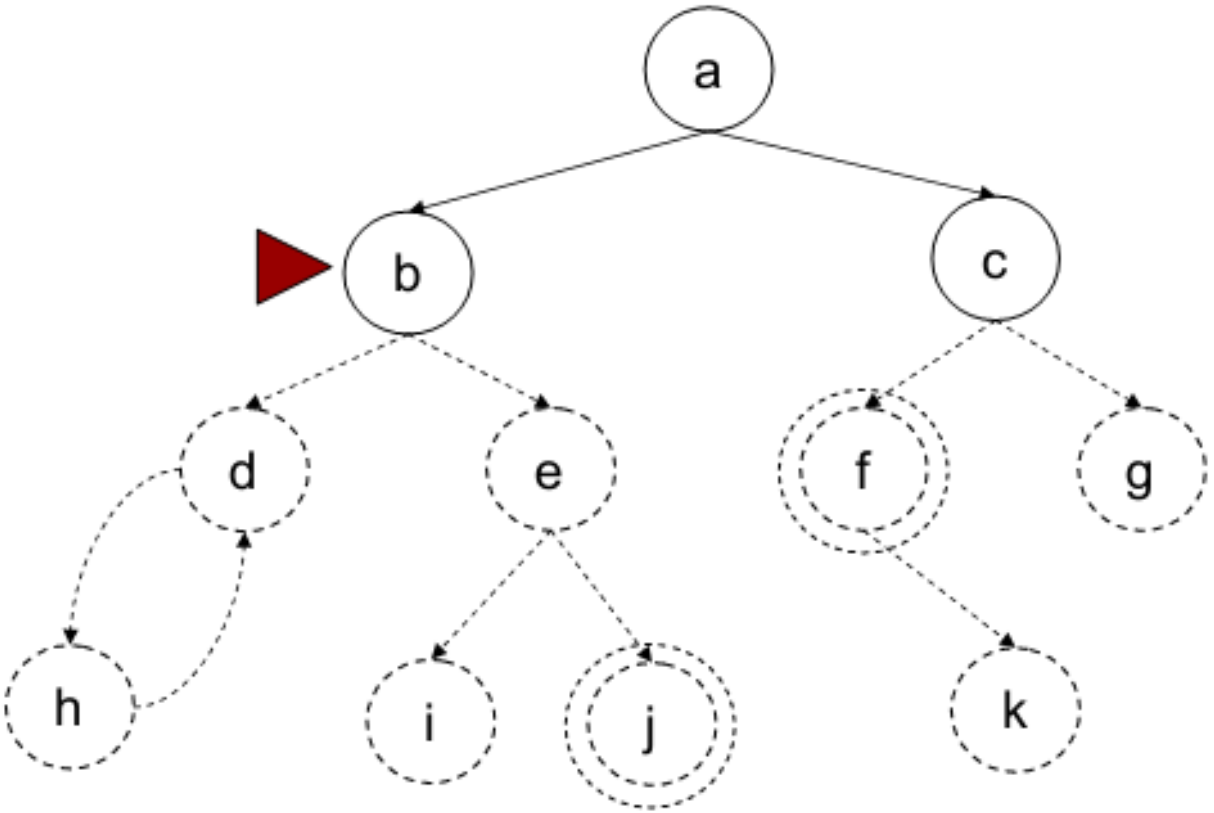
- Em contraste com a busca em profundidade, a busca em largura escolhe primeiro visitar aqueles nós mais próximos do nó inicial
- O algoritmo não é tão simples, pois é necessário manter um **conjunto** de nós candidatos alternativos e não apenas um único, como na busca em profundidade
- O conjunto é todo o nível inferior da árvore de Busca

Busca em Largura



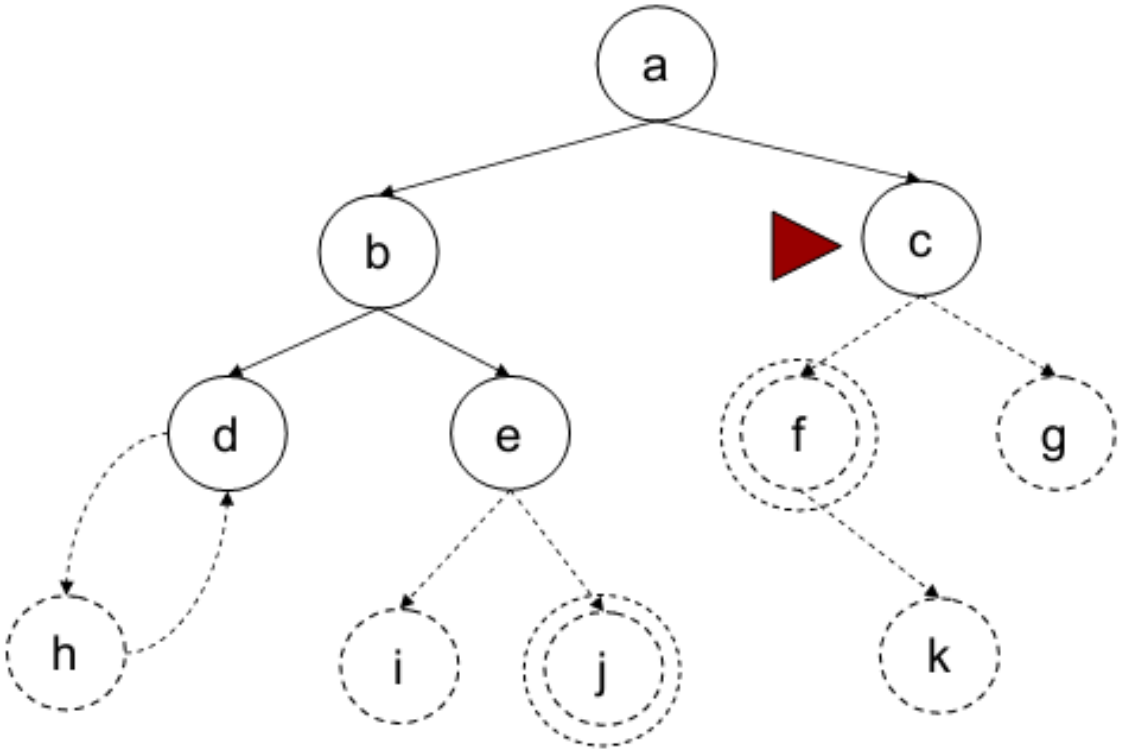
Inserir no final, remover da frente: a

Busca em Largura



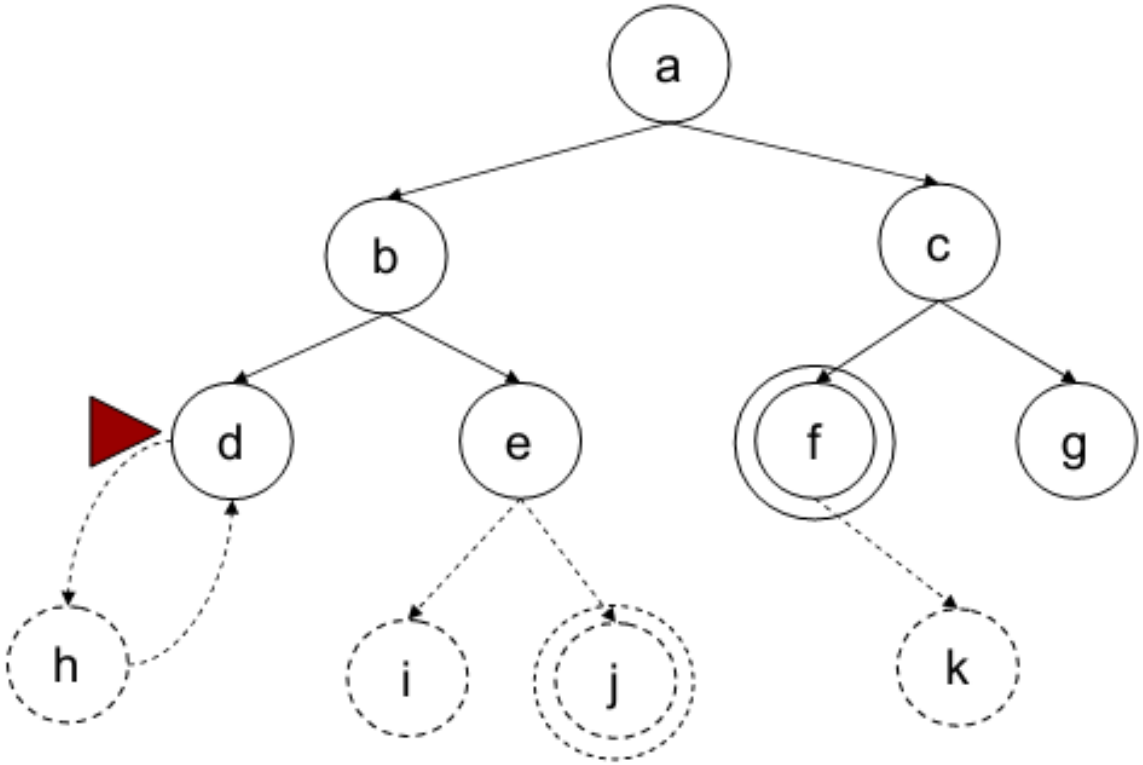
Inserir no final, remover da frente: b, c

Busca em Largura



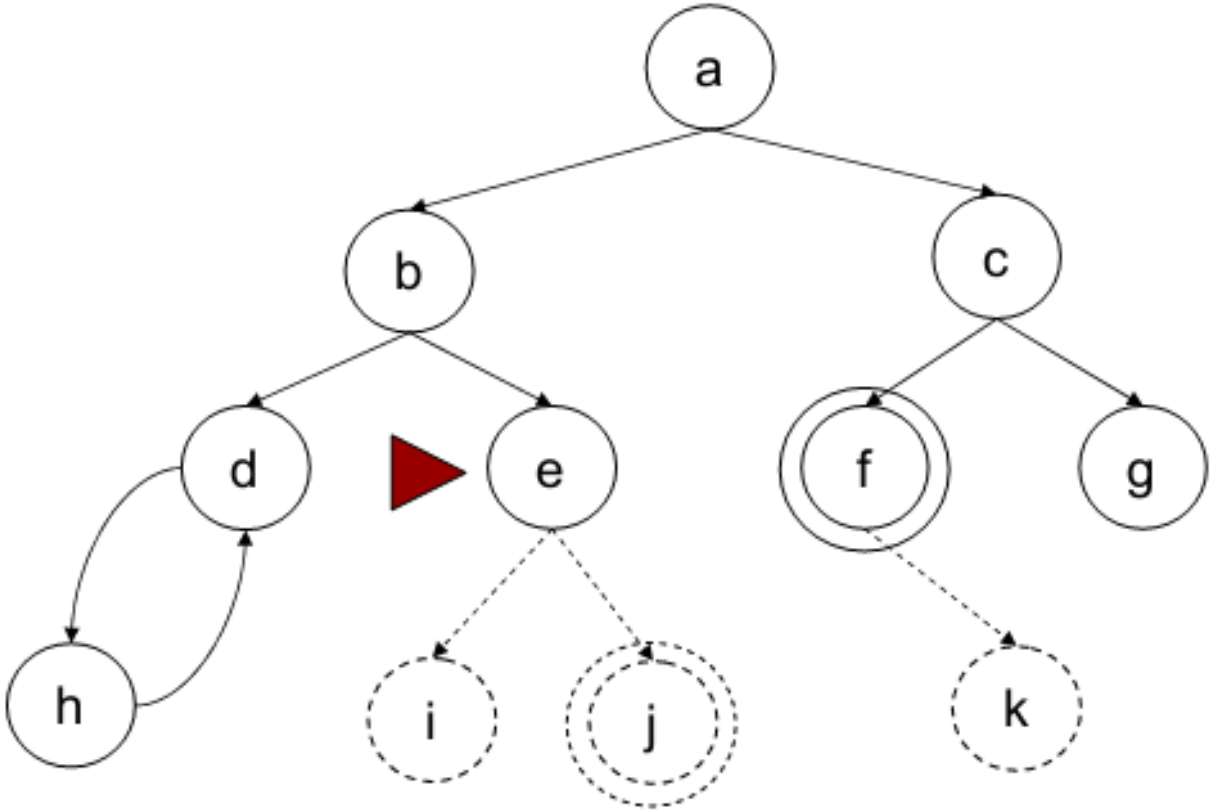
Inserir no final, remover da frente: c, d, e

Busca em Largura



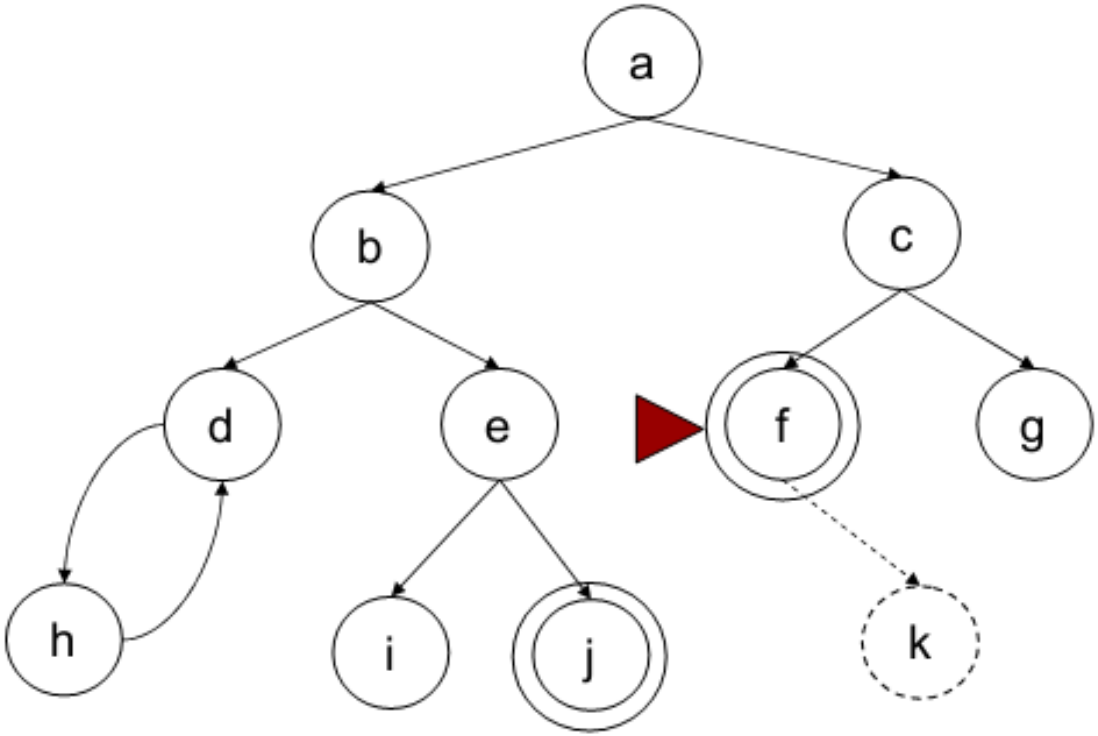
Inserir no final, remover da frente: d, e, f, g

Busca em Largura



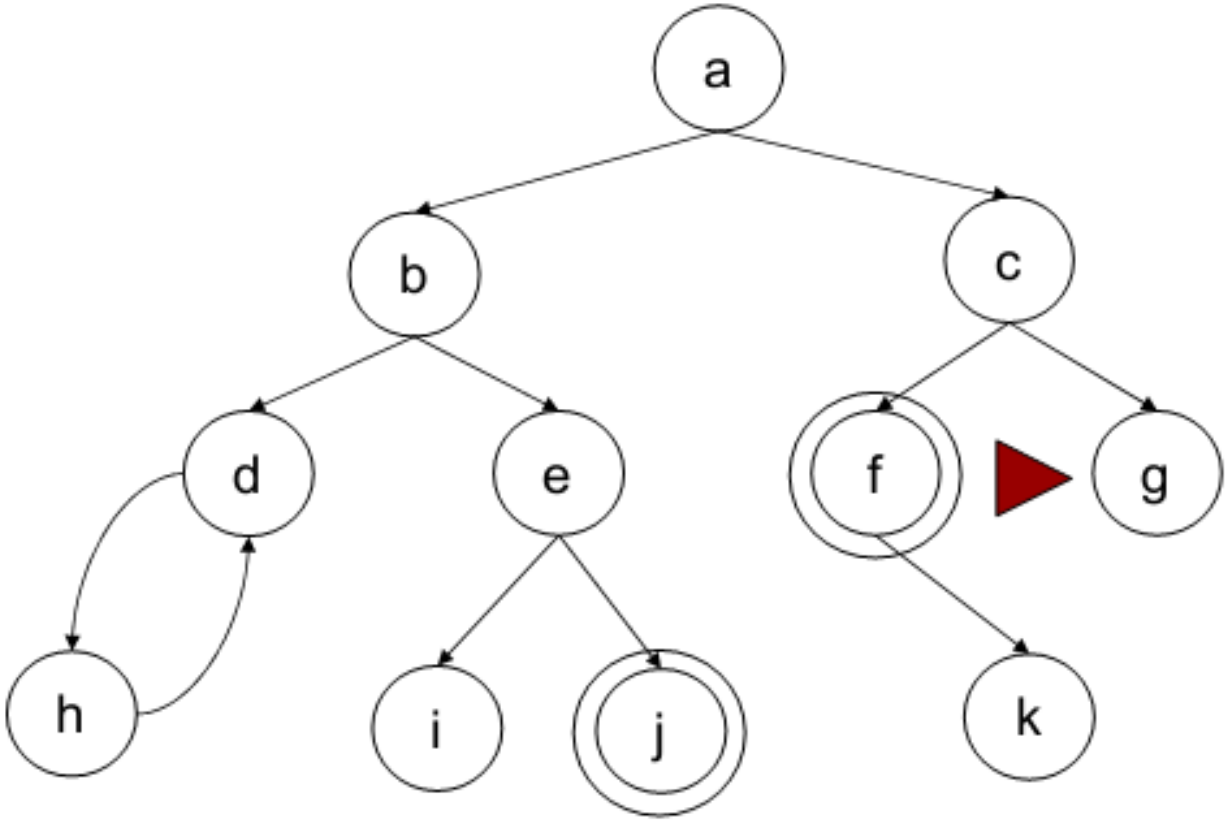
Inserir no final, remover da frente: e, f, g, h

Busca em Largura



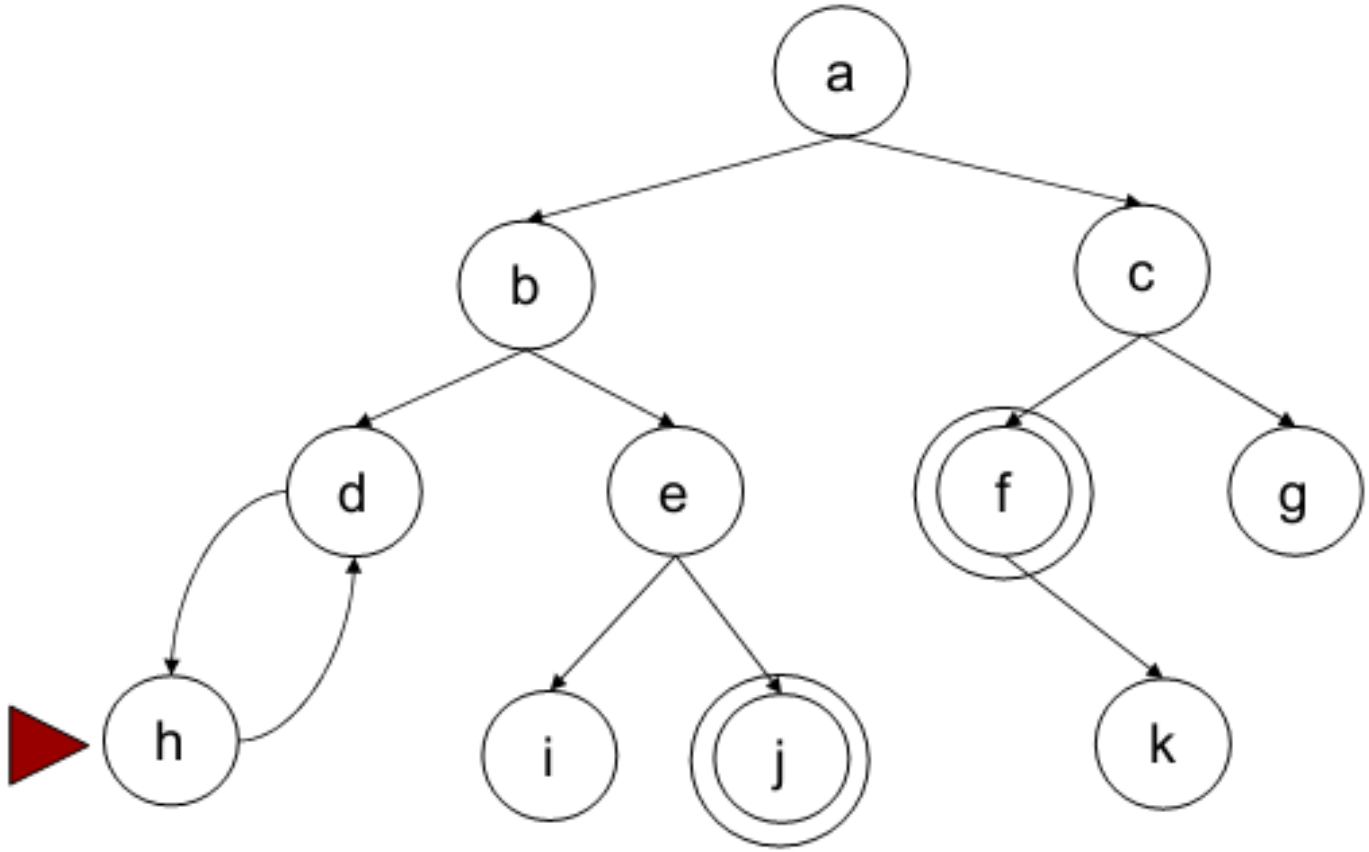
Inserir no final, remover da frente: f, g, h, i, j

Busca em Largura



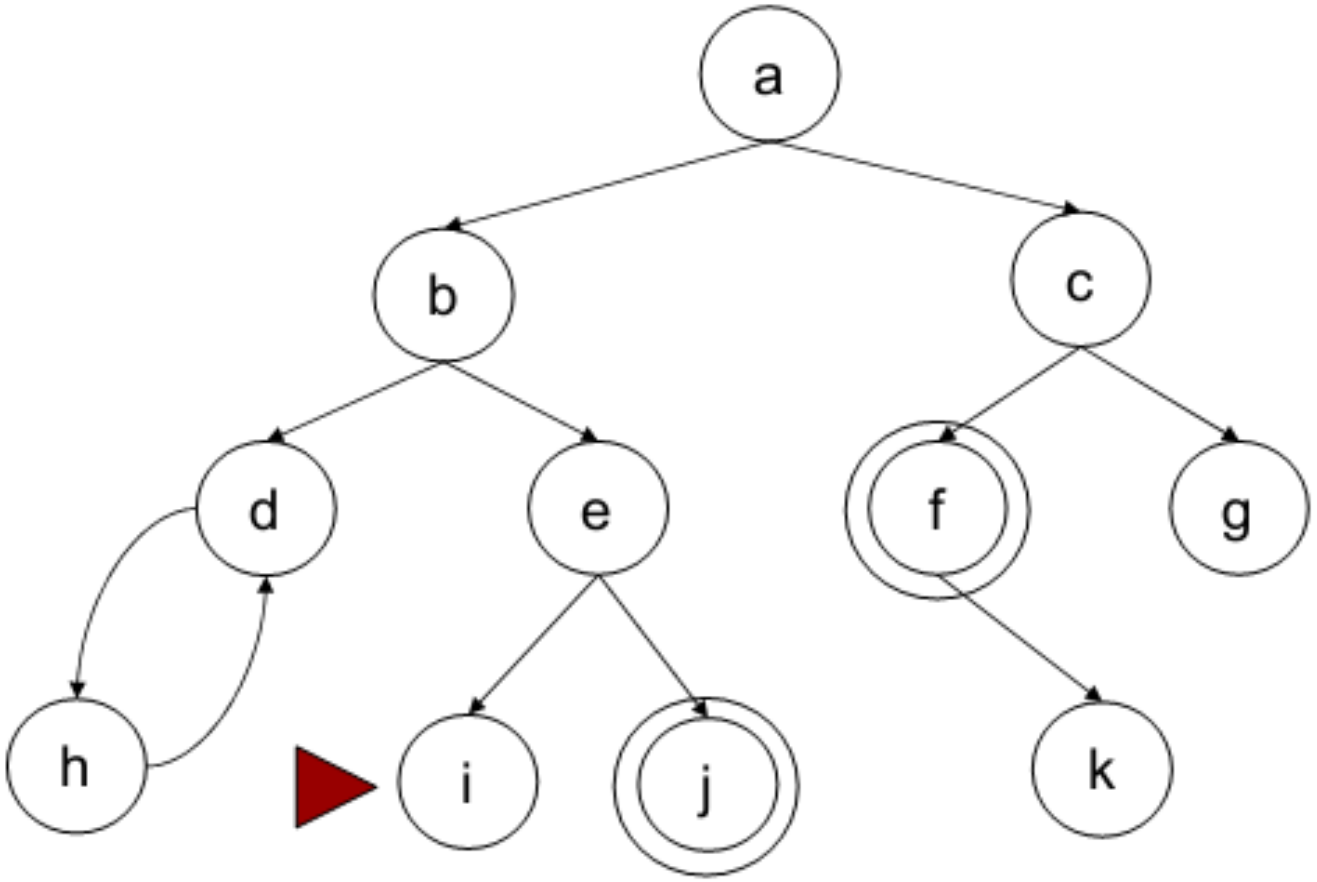
Inserir no final, remover da frente: g, h, i, j, k

Busca em Largura



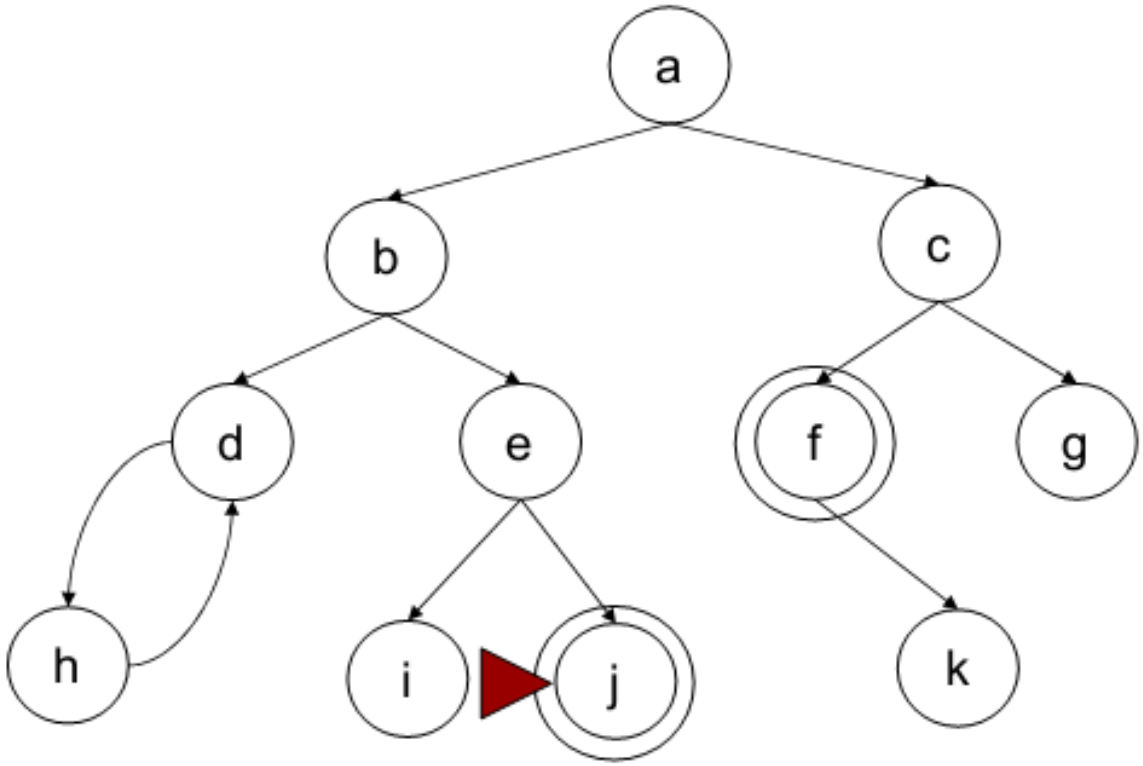
Inserir no final, remover da frente: h, i, j, k

Busca em Largura



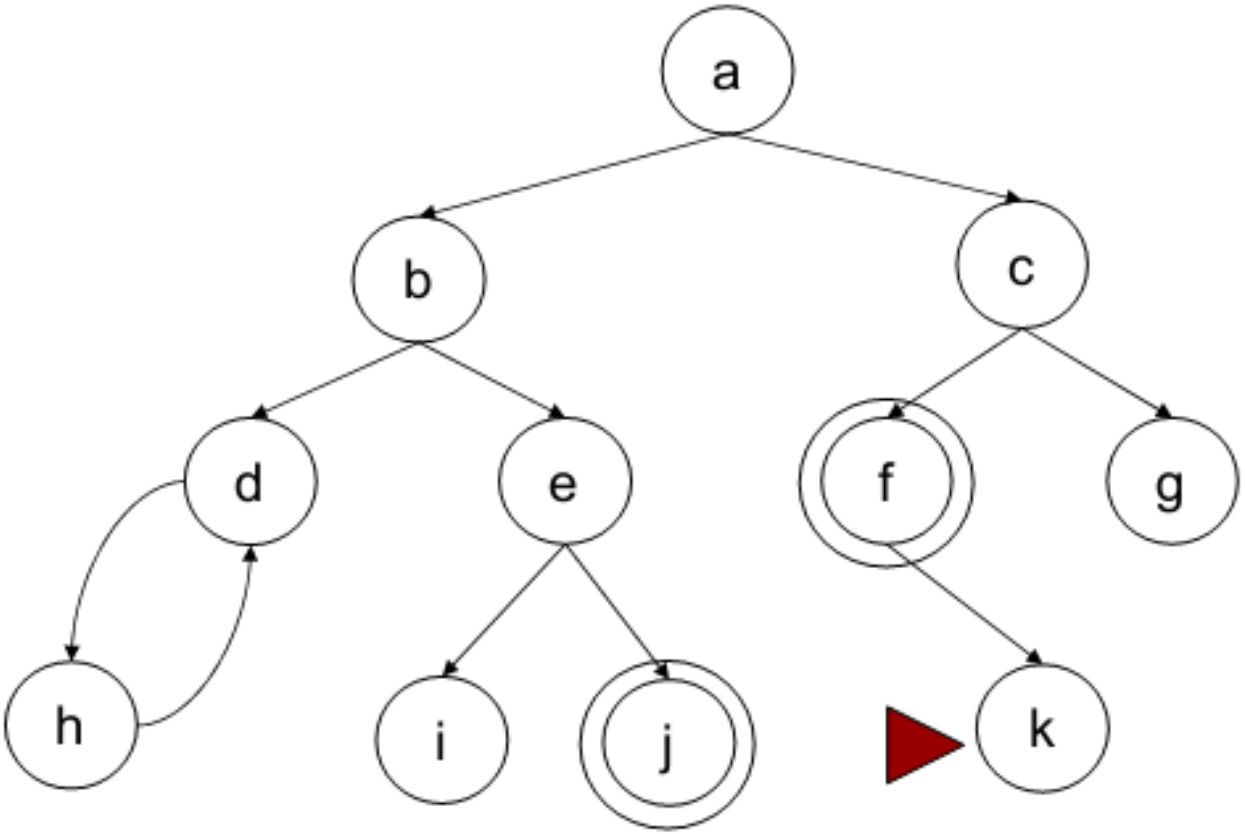
Inserir no final, remover da frente: i, j, k

Busca em Largura



Inserir no final, remover da frente: j, k

Busca em Largura



Inserir no final, remover da frente: k

Busca em Largura - Resumo

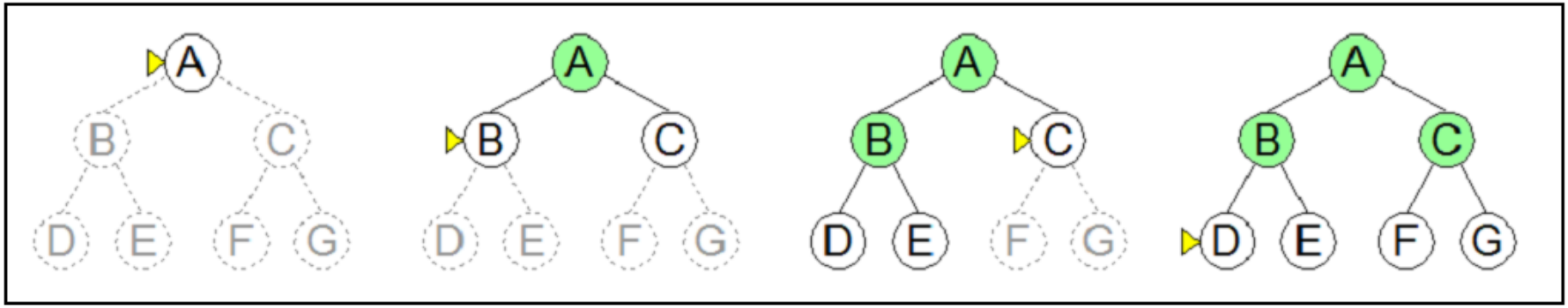
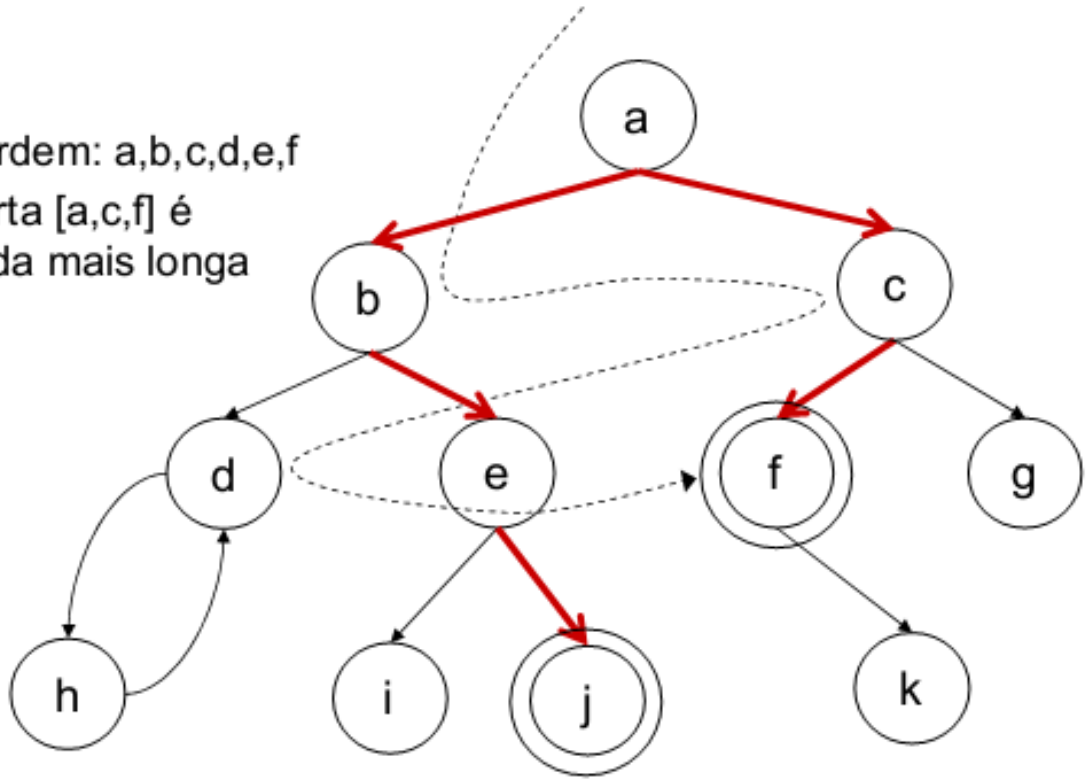


Figura 2.5 – Busca em Largura. Em cada fase o nó a ser expandido em seguida é indicado por um marcador (RUSSELL, 2003).

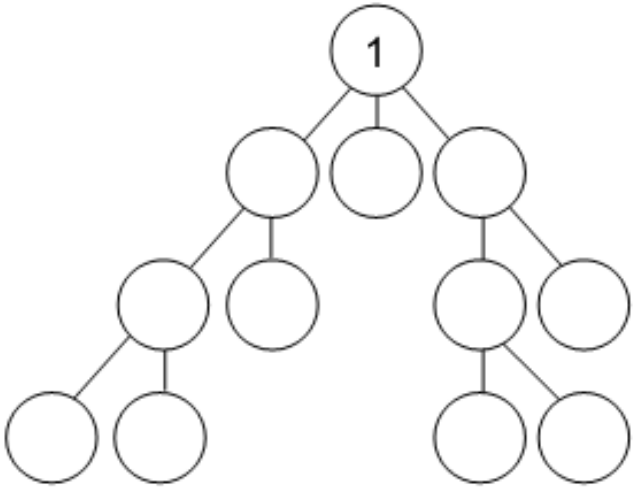
Busca em Largura

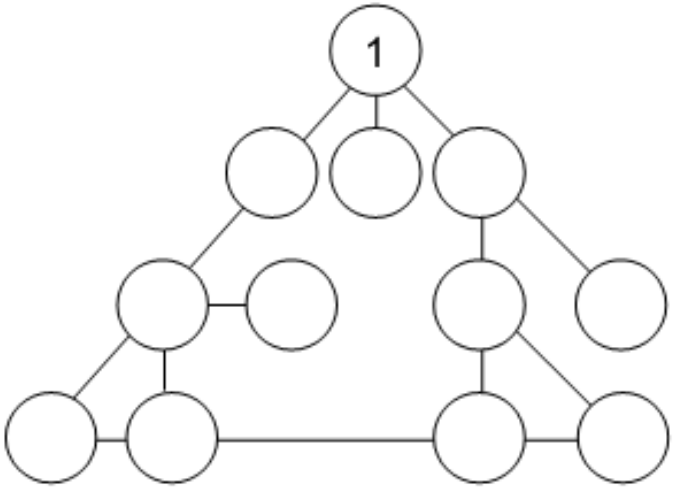
- Estado inicial: a
- Estados finais: j,f
- Nós visitados na ordem: a,b,c,d,e,f
- A solução mais curta [a,c,f] é encontrada antes da mais longa [a,b,e,j]



Exercício: Indique a ordem na qual os nó são visitados na busca em largura

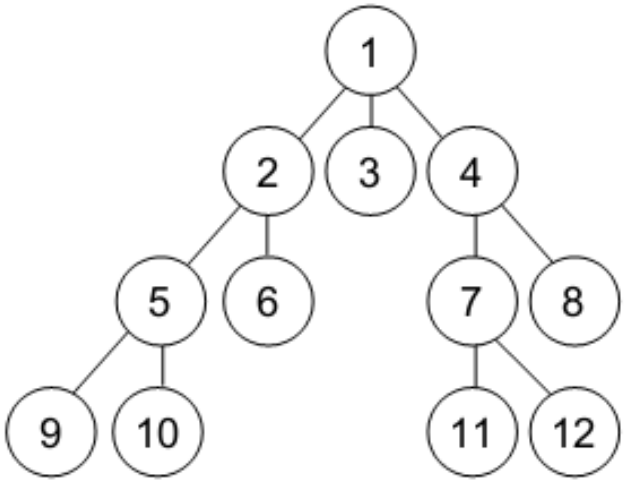
Assuma que os sucessores são definidos da esquerda para a direita e, depois, de cima pra baixo

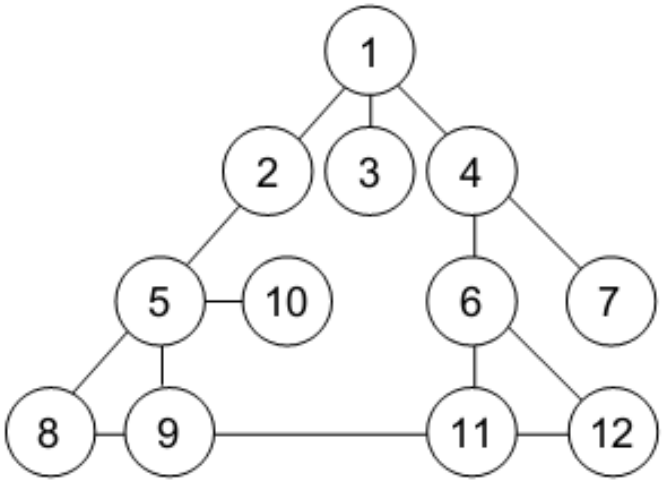




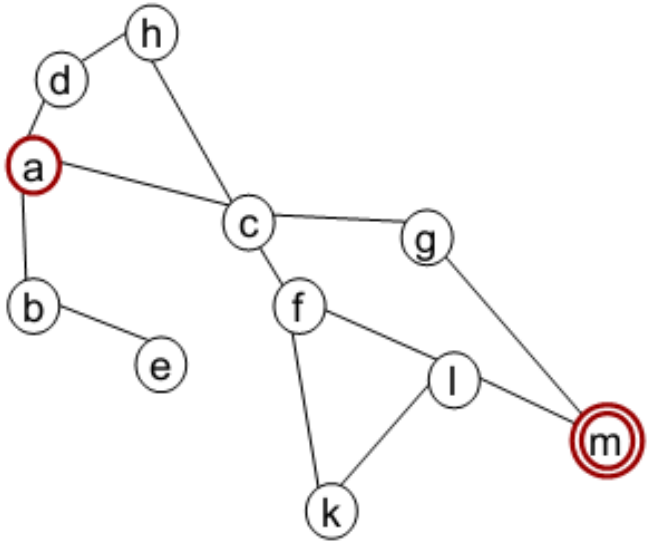
Solução

Assuma que os sucessores são definidos da esquerda para a direita e, depois, de cima pra baixo





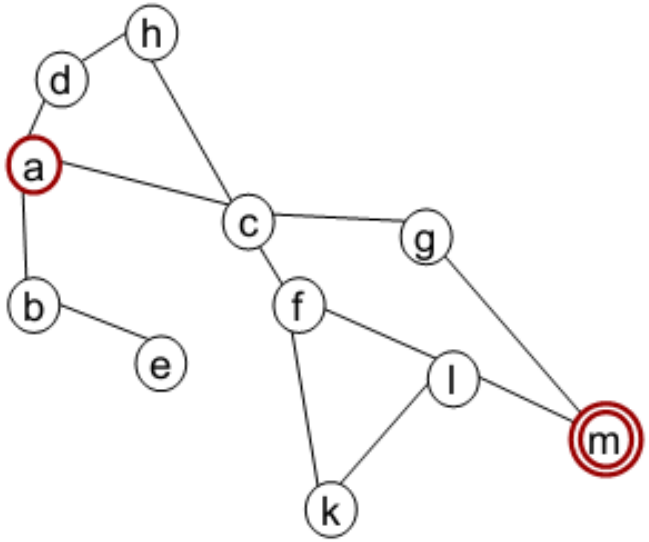
Exercício



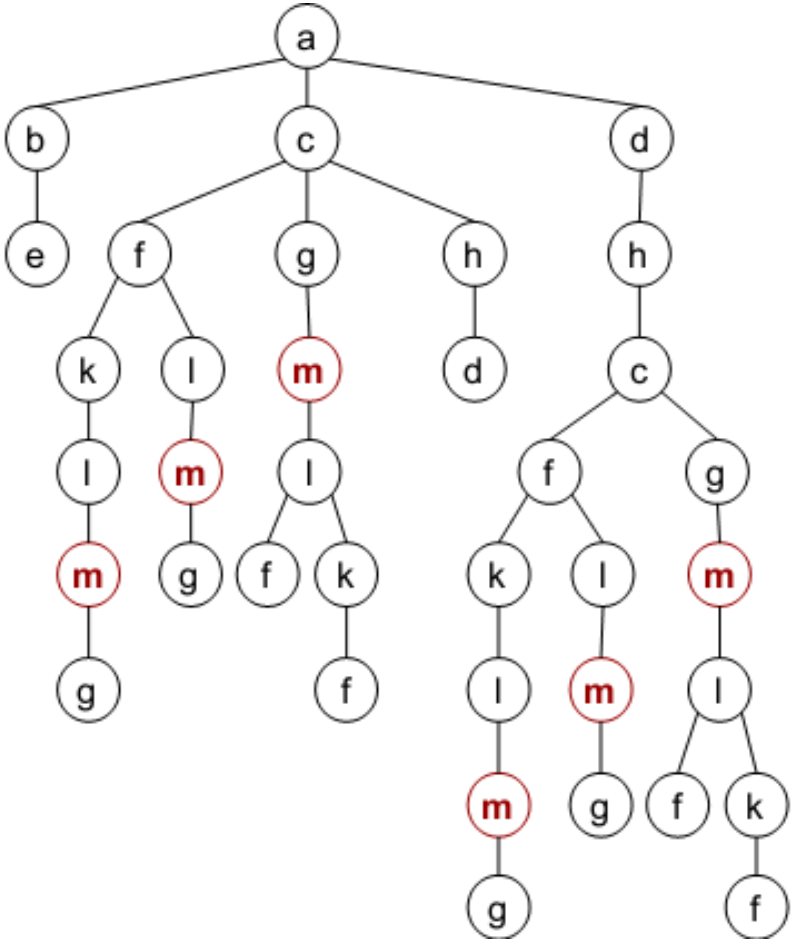
Assuma que os sucessores de um nó são definidos em ordem alfabética

- Mostre a árvore de busca definida pelo algoritmo de busca em largura, partindo de **a** e chegando até **m**
- Mostre também os caminhos encontrados na ordem em que são encontrados pela busca em largura

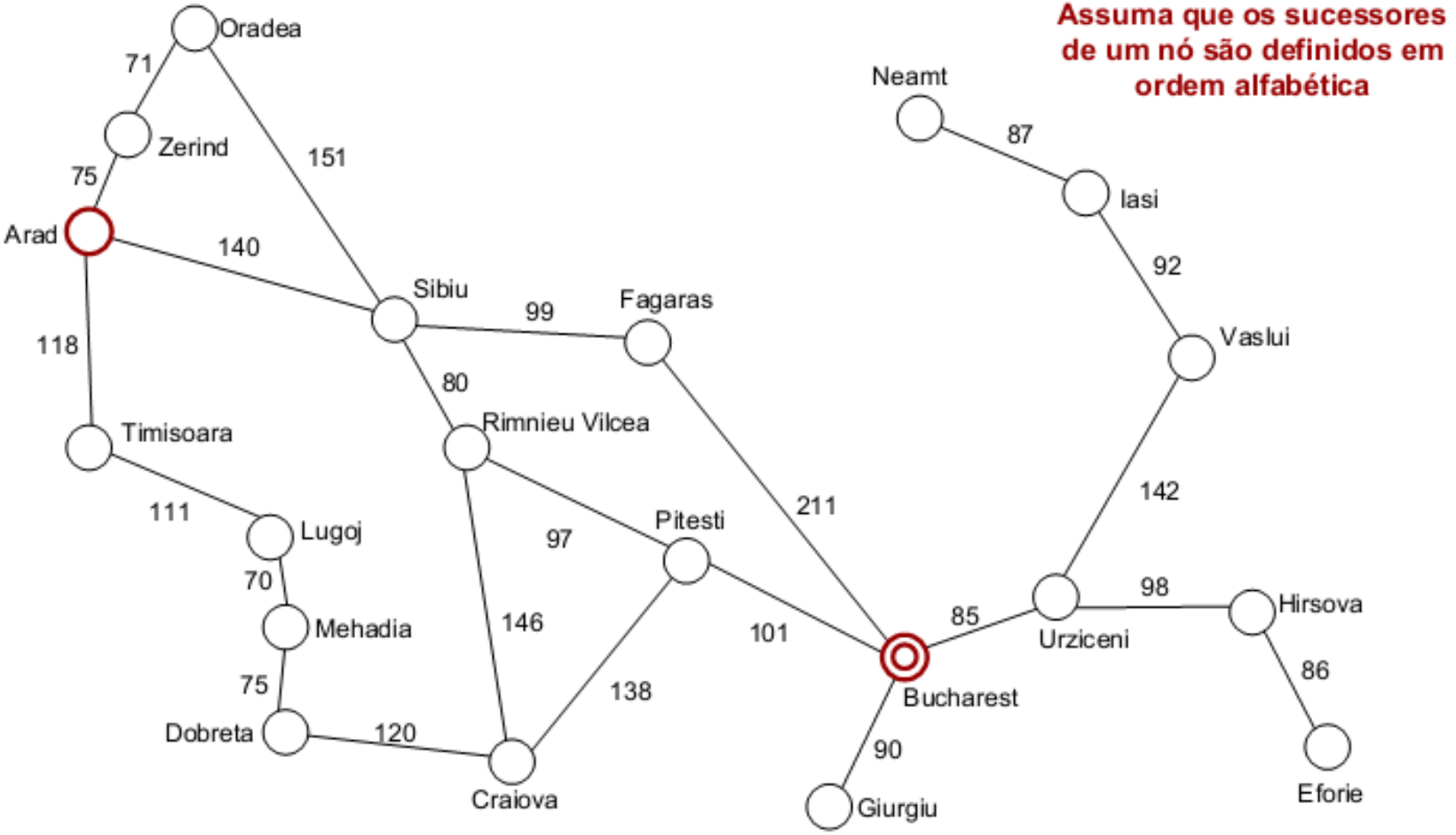
Solução



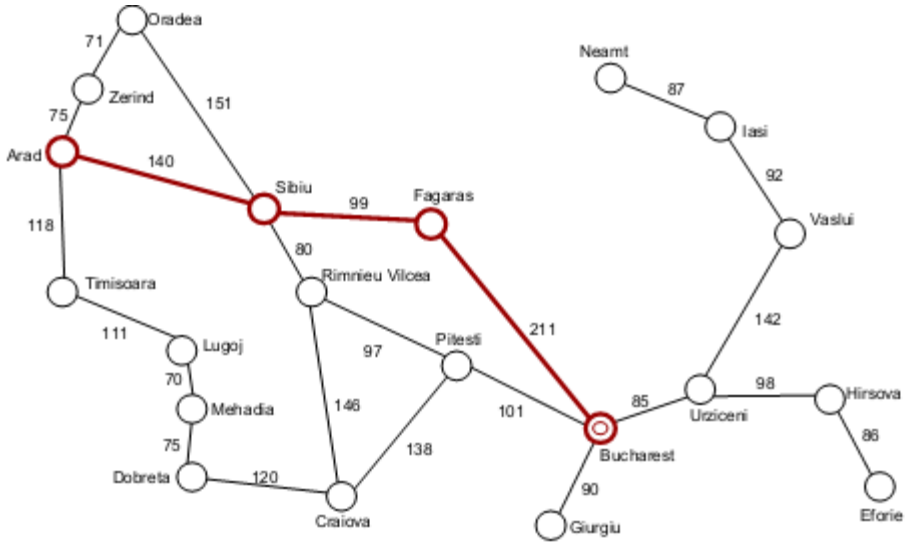
Caminhos encontrados:
[a,c,g,m]
[a,c,f,l,m]
[a,c,f,k,l,m]
[a,d,h,c,g,m]
[a,d,h,c,f,l,m]
[a,d,h,c,f,k,l,m]

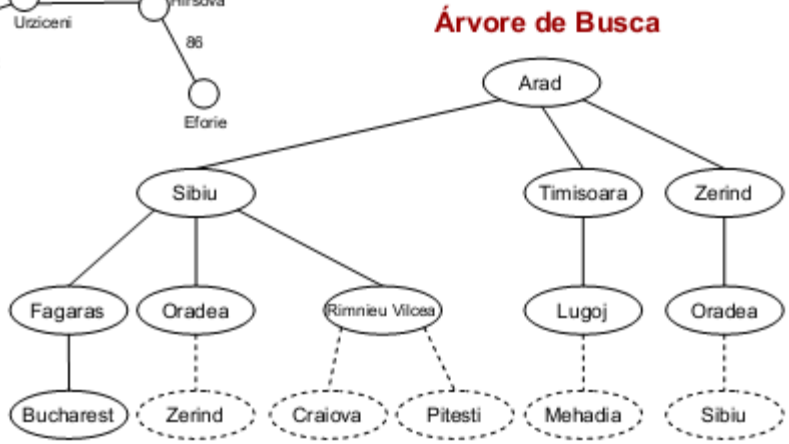


Encontre o Caminho e Árvore de Busca de Arad até Bucharest usando Busca em Largura



Encontre o Caminho e Árvore de Busca de Arad até Bucharest usando Busca em Largura





Complexidade dos algoritmos de Busca

Tempo e espaço gastos na busca em largura:
b = 10
1000 nós por segundo
100 bytes por nó

Profundidade	Nodos	Tempo	Memória
0	1	1 milisegundo	100 bytes
2	111	0.1 segundo	11 kilobytes
4	11111	11 segundos	1 megabytes
6	10^6	18 minutos	111megabytes
8	10^8	31 horas	11 gigabytes
10	10^10	128 dias	1 terabytes
12	10^12	35 anos	111 terabytes
14	10^14	3500 anos	11111 terabytes

Algoritmo

Algoritmo:

função Busca-em-Largura (*problema*)
retorna uma solução ou falha
Busca-Genérica (*problema*, Insere-no-Fim)

Busca de Custo Uniforme

Busca de custo uniforme

- A busca de custo uniforme (*uniform-cost search*) é similar à busca em largura, exceto pelo fato que os caminhos são colocados em uma **fila de prioridades**
  - ✓ A fila de prioridade se comporta como uma fila, exceto pelo fato que os elementos que a compõem são sempre ordenados em relação a algum valor, normalmente o custo associado

- O custo da raiz da busca até o nó atual ou o número de nós percorridos são exemplos de valores que são tipicamente colocados na fila de prioridades da busca de custo uniforme
  - ✓ g(n)= custo da raiz da busca até o nó *n*

□

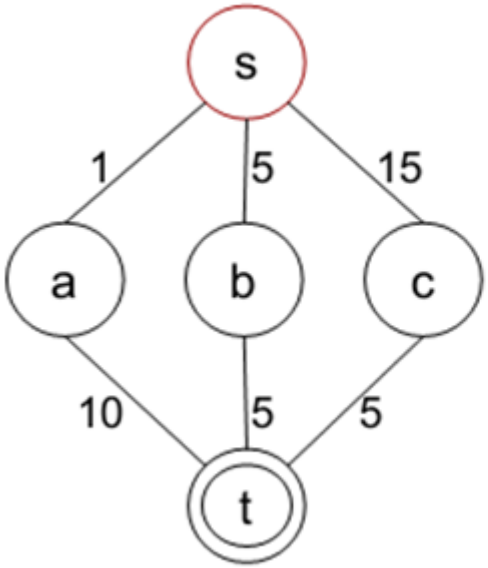
Busca de custo uniforme

- Assim, ao invés de expandir o nó mais próximo à raiz da busca, a busca de custo uniforme expande o nó com o caminho de custo mais baixo g(n)

- Se os custos forem iguais, a busca de custo uniforme torna-se idêntica à busca em largura

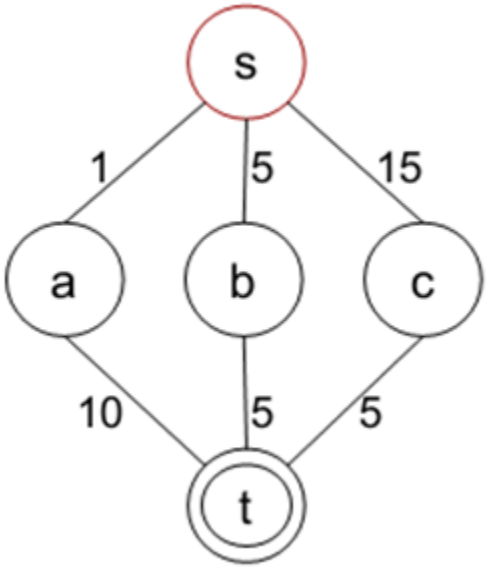
- O algoritmo é ótimo e completo se todos os custos forem maiores ou iguais a alguma constante positiva pequena ε

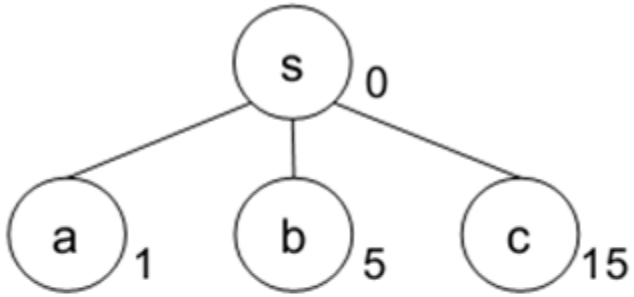
Busca de custo uniforme



Nó	Fila ordenada pelo valor de g
S	

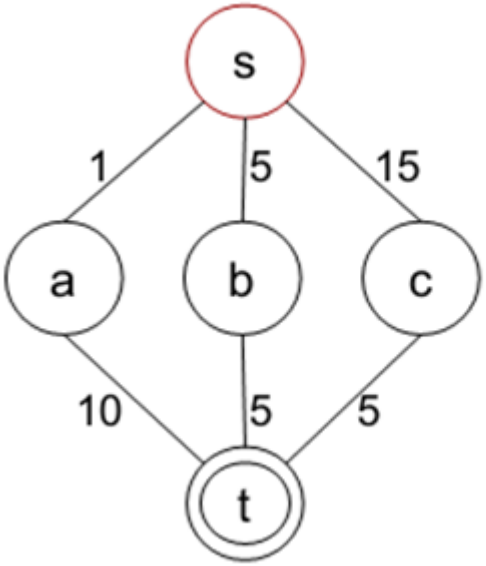
Busca de custo uniforme

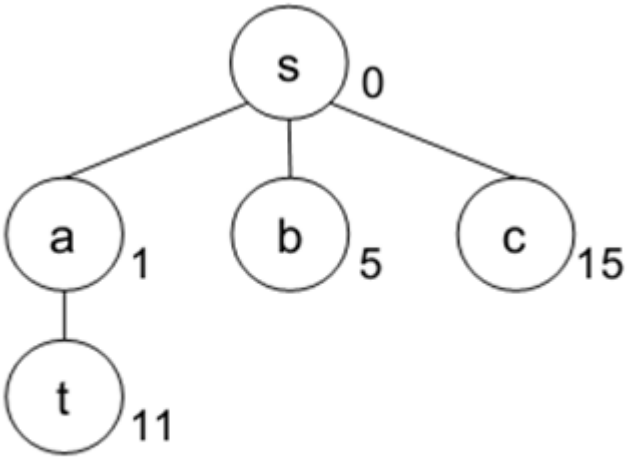




Nó	Fila ordenada pelo valor de g
S	[s,a]:1, [s,b]:5, [s,c]:15

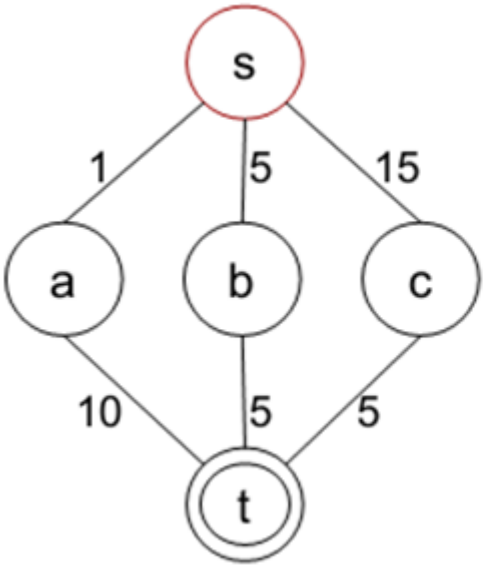
Busca de custo uniforme

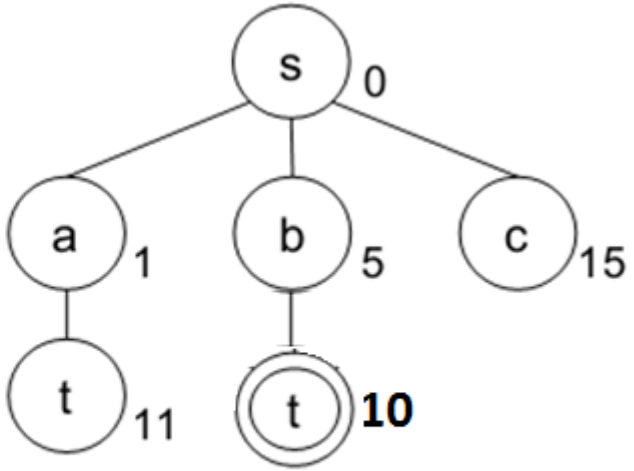




Nó	Fila ordenada pelo valor de g
S	[s,a]:1, [s,b]:5, [s,c]:15
a	[s,b]:5, [s,a,t]:11, [s,c]:15

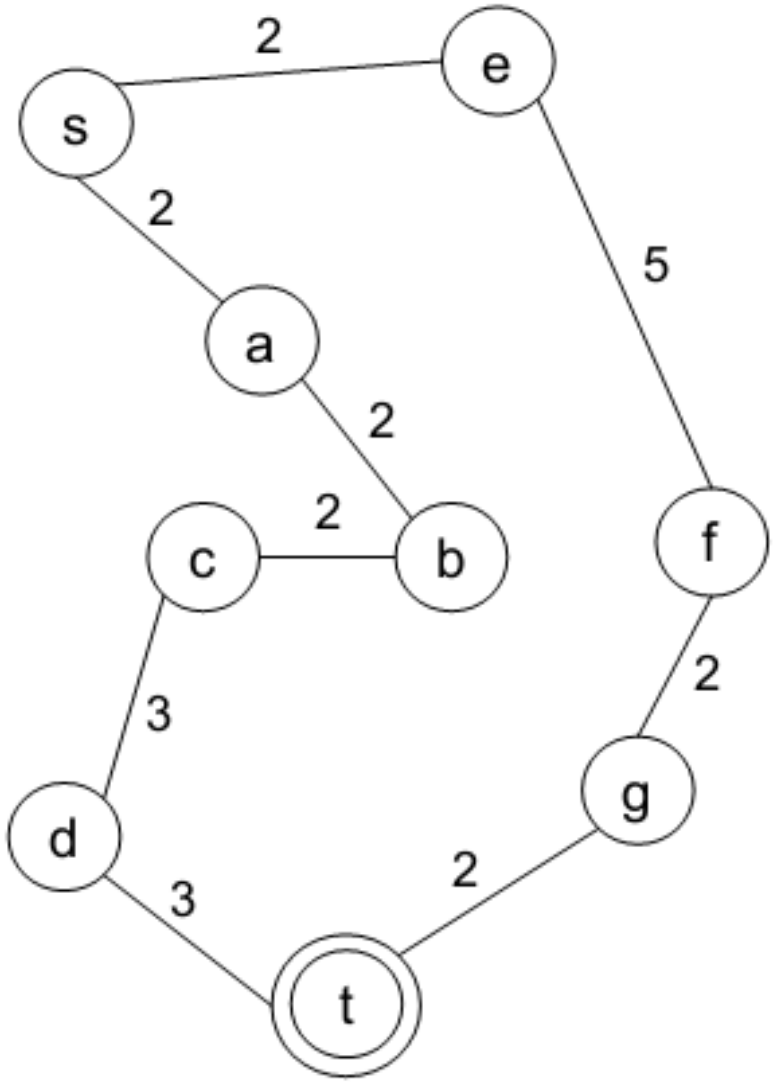
Busca de custo uniforme





Nó	Fila ordenada pelo valor de g
s	[s,a]:1, [s,b]:5, [s,c]:15
a	[s,b]:5, [s,a,t]:11, [s,c]:15
b	[s,b,t]:10, [s,a,t]:11, [s,c]:15
t	Solução (início da fila): [s,b,t]

Busca de Custo Uniforme: Exercício



Nó expandido?

Fila ordenada?

Busca de Custo Uniforme

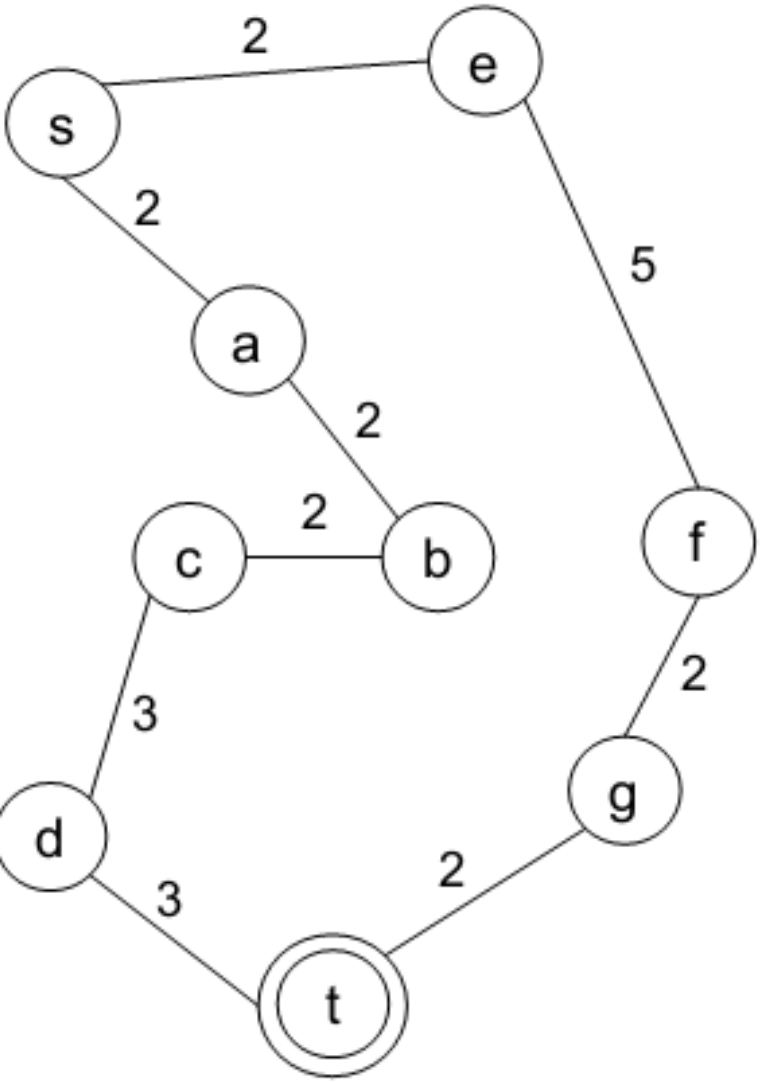
- A partir do nó inicial, a busca continua para o próximo nó de menor custo até a raiz

- Uma forma de busca do primeiro melhor (best-first search)
  Utilizando f(n) = g(n)

- Se o custo de se atingir cada nó é o mesmo torna-se igual à busca em largura

- Se a heurística é uma função constante, torna-se um caso particular de A*

Busca de Custo Uniforme: Exercício



Nó	Fila ordenada pelo valor de g
S	[a,s]:2, [s,e]:2
A	[s,e]:2, [b,a,s]:4
E	[b,a,s]:4, [f,e,s]:7
B	[c,b,a,s]:6 [f,e,s]:7,
C	[f,e,s]:7, [d,c,b,a,s]:9
F	[d,c,b,a,s]:9, [g,f,e,s]:9
D	[g,f,e,s]:9, [t,d,c,b,a,s]:12
G	[t,g,f,e,s]:11 [t,d,c,b,a,s]:12,
T	

Algoritmo

Algoritmo:

função Busca-de-Custo-Uniforme (*problema*)

retorna uma solução ou falha

Busca-Genérica (*problema*, Insere-Ordem-Crescente)

Avaliação de estratégias de busca

Métodos	Completa?	Ótimo?	Tempo	Espaço
Largura	Sim[a]	Sim[c]	$O(b^{d+1})$	$O(b^{d+1})$
Custo uniforme	Sim[a,b]	Sim[b]	$O(b^{\lceil C*/\varepsilon \rceil})$	$O(b^{\lceil C*/\varepsilon \rceil})$
Profundidade	Não	Não	$O(b^{m})$	$O(bm)$
Profundidade limitada	Não	Não	$O(b^{l})$	$O(bl)$
Profundidade iterativa	Sim[a]	Sim[c]	$O(b^{d})$	$O(bd)$

b é o fator de ramificação; *d* é a profundidade da solução mais rasa; *m* é a profundidade máxima da árvore de busca; *l* é o limite de profundidade. As notações sobrescritas são: [a] completa se *b* for finito; [b] completa se o custo do passo é $\geq \varepsilon$ para ε positivo; [c] ótima se os custos dos passos são todos idênticos.

Referências

http://dcm.ffclrp.usp.br/~augusto/ia/

http://adm-net-a.unifei.edu.br/phl/pdf/0036188.pdf

http://www-usr.inf.ufsm.br/~pozzer/disciplinas/pj3d_busca.pdf

http://www.allgoritmos.com/2009/08/busca-em-profundidade-dfs.html

http://www.dca.fee.unicamp.br/~gomide/courses/EA072/transp/EA072BuscaBasicaProgramas.pdf

http://professor.ufabc.edu.br/~leticia.bueno/classes/teoriagrafos/materiais/dfs.pdf

http://www.professeurs.polymtl.ca/michel.gagnon/Disciplinas/Bac/Grafos/index_grafos.html

http://professor.ufabc.edu.br/~ronaldo.prati/InteligenciaArtificial/pratica1.html - PAC-MAN

http://en.wikipedia.org/wiki/Breadth-first_search