Inteligência Artificial

Resolução de Problemas

Sumário

- Agentes Solucionadores de Problemas
- Exemplos de Problemas
  - Problemas Simples
  - Problemas Reais
- Busca por Soluções
- Estratégias de Busca Exaustiva
  - Busca em Largura
  - Busca em Profundidade
  - Busca Bidirecional
- Evitando Estados Repetidos
- Busca com Informação Parcial

Resolução de Problemas

Agentes Solucionadores de Problemas

Agentes Solucionadores de Problemas

- Agentes Inteligentes buscam maximizar a Medida de Desempenho
  - Agentes Solucionadores de Problema são do tipo *Agentes Baseados em Objetivo*
    - Objetivos ajudam a reduzir o *espaço de busca*
    - Ações que não levam ao objetivo não precisam ser consideradas

Agente Solucionador de Proble-mas Simples

- Formulação do Objetivo
  - Objetivo é um conjunto de estados do mundo
- Formulação do Problema
  - Definição das ações e estados que devem ser consi-derados dado o objetivo
- Busca
  - Processo de analisar diferentes sequências de ações e escolher a melhor, dado o objetivo e a medida de desempenho
- Execução
  - Execução da sequência de ações encontrada durante a busca

Agente Solucionador de Proble-mas Simples

```
função AGENTE-SOLUCIONADOR-DE-PROBLEMAS-SIMPLES(percepção) retorna uma ação
  entradas: percepção, uma percepção
  estático: seq, uma sequência de ações, inicialmente vazia
            estado, uma descrição do estado atual do mundo
            objetivo, um objetivo, inicialmente nulo
            problema, uma formulação de um problema

  estado ← ATUALIZA-ESTADO(estado, percepção)
  se seq está vazia então
    objetivo ← FORMULA-OBJETIVO(estado)
    problema ← FORMULA-PROBLEMA(estado,objetivo)
    seq ← BUSCA(problema)

  ação ← PRIMEIRA(seq)
  seq ← RESTO(seq)
  retorne ação
```

Agente Solucionador de Proble-mas Simples

```
função AGENTE-SOLUCIONADOR-DE-PROBLEMAS-SIMPLES(percepção) retorna uma ação
  entradas: percepção, uma percepção
  estático: seq, uma sequência de ações, inicialmente vazia
            estado, uma descrição do estado atual do mundo
            objetivo, um objetivo, inicialmente nulo
            problema, uma formulação de um problema

  estado ← ATUALIZA-ESTADO(estado, percepção)
  se seq está vazia então
    objetivo ← FORMULA-OBJETIVO(estado)
    problema ← FORMULA-PROBLEMA(estado,objetivo)
    seq ← BUSCA(problema)

  ação ← PRIMEIRA(seq)
  seq ← RESTO(seq)
  retorne ação
```

Enquanto está executando a sequência, ignora as percepções. (Malha Aberta)

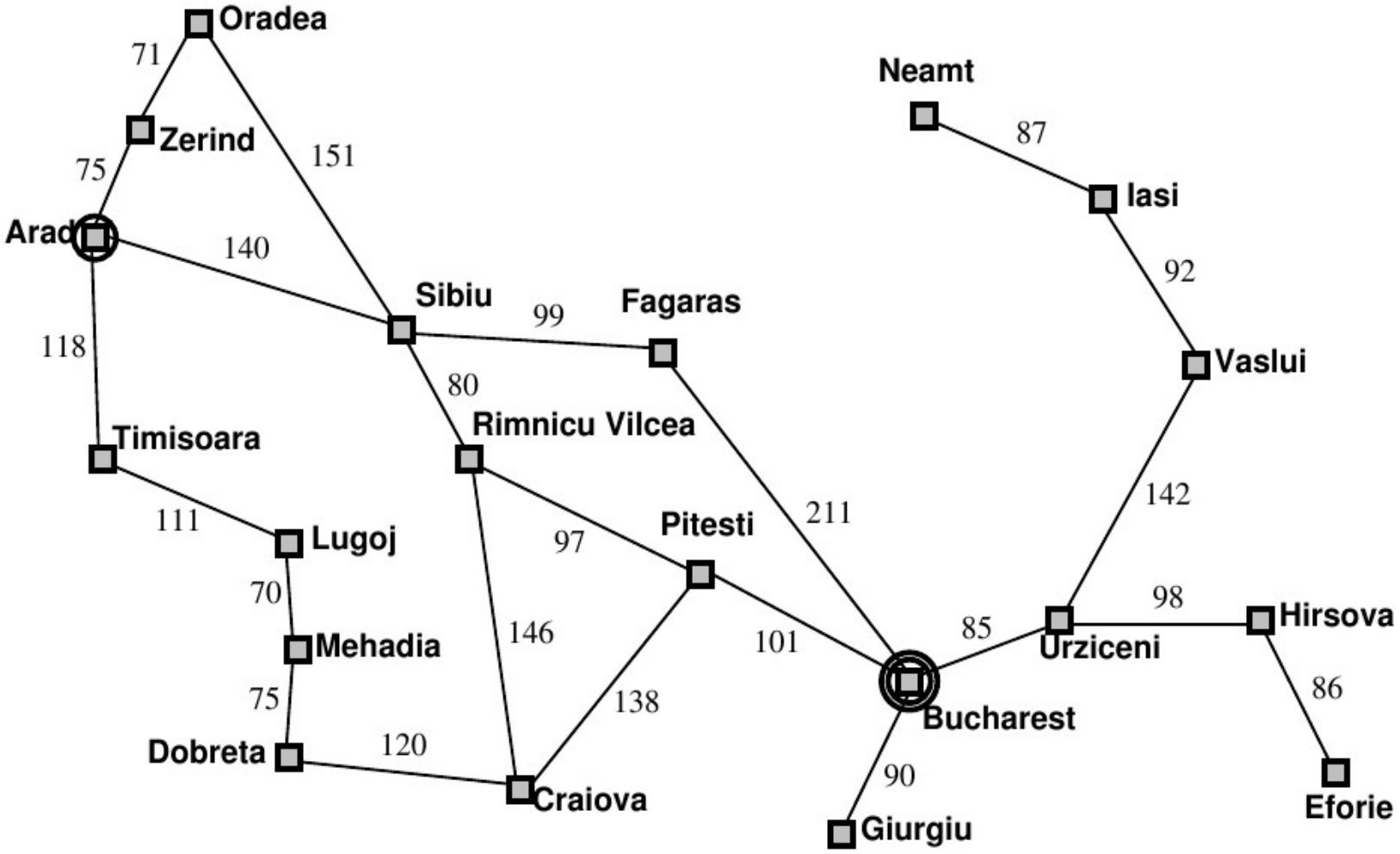
Agente Solucionador de Proble-mas Simples

- O agente descrito assume que o ambiente é:
  - Estático
  - Totalmente Observável
    - O primeiro estado é conhecido
  - Discreto (enumeração de caminhos de ação)
  - Determinístico
    - Malha aberta (ou *offline*)
  - Sequencial

Agente: Viajante na Romênia

- Estado Inicial
  - Está em Arad
- Medidas de Desempenho
  - Aperfeiçoar o Romeno
  - Apreciar a paisagem
  - Aproveitar a vida noturna
  - Economizar gasolina
- Objetivo
  - Chegar a Bucareste no dia seguinte
    - Passagem aérea comprada

Mapa Simplificado da Romênia



Definição de um Problema

- Estado inicial
  - Ex: *Em(Arad)*
- Função Sucessor
  - Dado um estado *x*, FUNÇÃO-SUCESSOR(*x*) retorna um par ordenado [*ação*, *sucessor*] onde?
    - *ação* é uma ação legal no estado *x*
    - *sucessor* é o estado alcançado se *ação* for executada em *x*
  - Ex: Se a função sucessor for aplicada no estado *Em(Arad)*, o retorno seria:
    - {⟨*Vá(Sibiu)*, *Em(Sibiu)*⟩, ⟨*Vá(Timisoara)*, *Em(Timisoara)*⟩, ⟨*Vá(Zerind)*, *Em(Zerind)*⟩}
  - O estado inicial, juntamente com a função sucessor, definem implicitamente o Espaço de Estados
    - Estados Alcançáveis a partir do inicial

Definição de um Problema

- Teste do Objetivo
  - Define se o objetivo foi alcançado
    - Ex: *Em(Bucareste)*
  - Pode ser uma propriedade abstrata, em vez de um estrado
    - Ex: X*eque-mate*
- Função de Custo do Caminho
  - Define quão bom é um caminho em relação a outros
    - O Caminho, num espaço de estados, é uma sequência de estados conectados por uma sequência de ações
    - Ex: *c(Em(Arad), Go(Sibiu), Em(Sibiu))* retorna 140
  - Uma solução do problema leva até o objetivo
    - Uma solução ótima é uma solução que tem o menor custo

Formulação de Problemas

- Abstração
  - Remover detalhes da representação
  - Facilita o tratamento da informação
    - Diminui o espaço de estados
    - Ex: Não é preciso representar no espaço de estados se o carro está com o rádio ligado ou não
  - A abstração é válida se pode-se expandi-la em soluções para um mundo mais detalhado

Resolução de Problemas

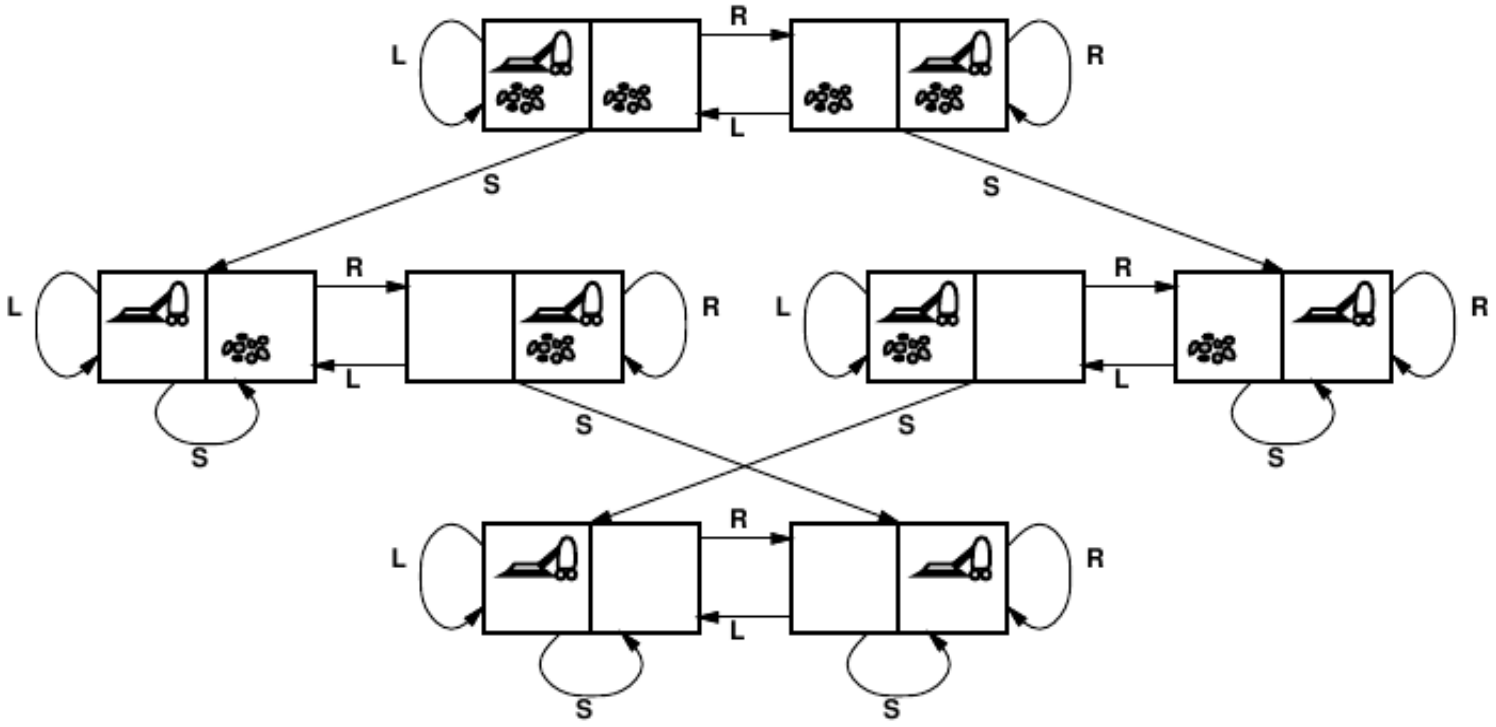
Exemplos de Problemas
Problemas Simples
(*Toy Problems*)

Mundo do Aspirador de Pó

- Formulação
  - Estados
    - *n* quadrados que podem ou não estar sujos (ignora quantidade de sujeira)
      - Total: $n \times 2^n = 8$ possíveis estados para 2 quadrados
  - Estado inicial
    - Qualquer estado pode ser o inicial
  - Função Sucessor
    - Gera os estados a partir das ações válidas
      - (*Left*, *Right*, *Suck*)
  - Teste do Objetivo
    - Testa se todos os quadrados estão limpos
  - Custo do Caminho
    - Sempre 1 por ação. O custo do caminho é o número de passos.

Mundo do Aspirador de Pó

- Espaço de Estados



Quebra-cabeça de Blocos Desli-záveis

- Formulação
  - Estados
    - *n* blocos deslizáveis, numerados de *1* a *n*, dispostos aleatoriamente em um tabuleiro quadrado com *n+1* posições
      - Total: *(n+1)!/2 = 181.440* possíveis estados para um tabuleiro com 8 blocos (*8-puzzle*)
  - Estado inicial
    - Somente metade dos estados podem ser iniciais [JOHNSON e STORY, 1879]
  - Função Sucessor
    - Gera os estados a partir das ações válidas
      - (*Left*, *Right*, *Up*, *Down*)
  - Teste do Objetivo
    - Testa se o estado objetivo foi alcançado
  - Custo do Caminho
    - Sempre 1 por ação. O custo do caminho é o número de passos.

Quebra-cabeça de Blocos Desli-záveis

- Espaço de Estados

7	2	4
5		6
8	3	1

Estado Inicial

1	2	3
4	5	6
7	8	

Estado Objetivo

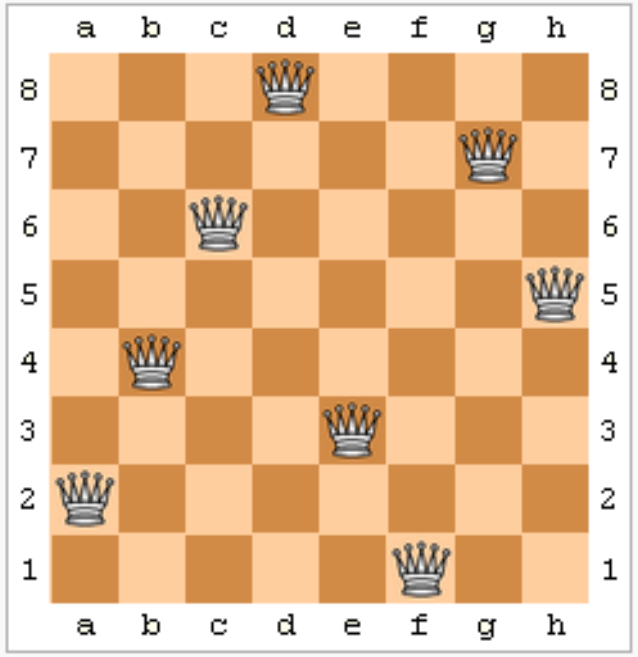
Problema das n Rainhas

- Formulação
  - Estados
    - Qualquer arranjo de *0* a *n* rainhas no tabuleiro
      - Total: $n^2!/(n^2 - n)! \approx 1.8 \times 10^{14}$ possíveis estados para 8 rainhas
  - Estado inicial
    - Nenhuma rainha no tabuleiro
  - Função Sucessor
    - Adiciona uma rainha num quadrado vazio
  - Teste do Objetivo
    - *n* rainhas estão no tabuleiro, nenhuma atacável por outra
  - Custo do Caminho
    - Todos os caminhos têm o mesmo custo

Problema das n Rainhas

- Formulação Melhor
  - Estados
    - Qualquer arranjo de *k rainhas*, $0 \leq k \leq n$, no tabuleiro, uma por coluna, nas *k* colunas mais à esquerda, com nenhuma rainha sendo atacada
      - Total: 2057 possíveis estados para 8 rainhas
  - Estado inicial
    - Nenhuma rainha no tabuleiro
  - Função Sucessor
    - Adiciona uma rainha em um quadrado da coluna vazia mais à esquerda de forma que esta não seja atacada por outra rainha
  - Teste do Objetivo
    - *n* rainhas estão no tabuleiro, nenhuma atacável por outra
  - Custo do Caminho
    - Todos os caminhos têm o mesmo custo (por esta formulação)

Problema das n Rainhas



Uma solução possível

Resolução de Problemas

Exemplos de Problemas
Problemas Reais
(*Real-world Problems*)

Busca de Rota

- Formulação
  - Estados
    - Cada representa um lugar (e.g. Aeroporto) e o tempo atual
  - Estado inicial
    - Definido pelo problema
  - Função Sucessor
    - Retorna os estados resultantes de utilizar um meio de locomoção (e.g. um avião) agendado após o tempo atual, somando ao tempo atual o tempo de deslocamento até o lugar de destino
  - Teste do Objetivo
    - Atingiu-se o lugar de destino dentro do tempo especificado?
  - Custo do Caminho
    - Pode incluir o custo financeiro, tempo de espera, tempo de deslocamento, conforto do deslocamento, bonificação de milhas, etc.

Touring Problems

- Problemas similares aos de Rota
  - Todas os estados incluem TODAS as localidades visitadas anteriormente
    - Ex: *Em(Vaslui); Visitou {Bucareste, Urziceni, Vaslui}*
  - O objetivo é chegar a um local visitando todos os outros
    - Chegar a Bucareste visitando todas as outras 20 cidades
- Ex: Caixeiro-Viajante
  - Problema do tipo NP-Difícil

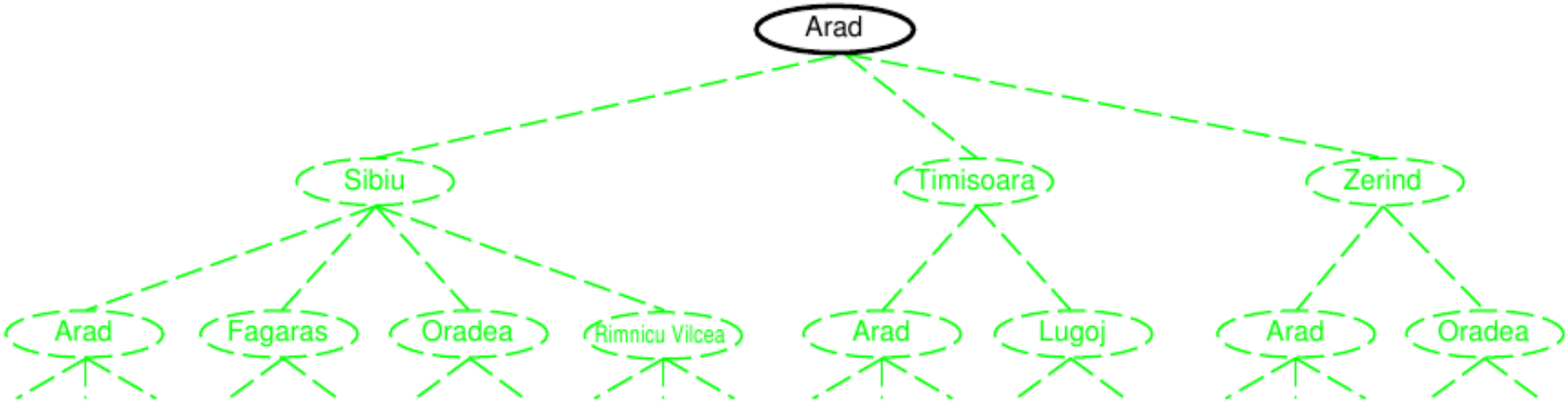
Outros Problemas

- *Layout* de sistemas VLSI
- Navegação de Robôs
  - Similar ao Problema de Rota, mas o ambiente é contínuo
- Sequência Automática de Montagem
  - Qual peça vem antes de qual?
- Projeto de Proteínas
- Busca na Internet

Resolução de Problemas

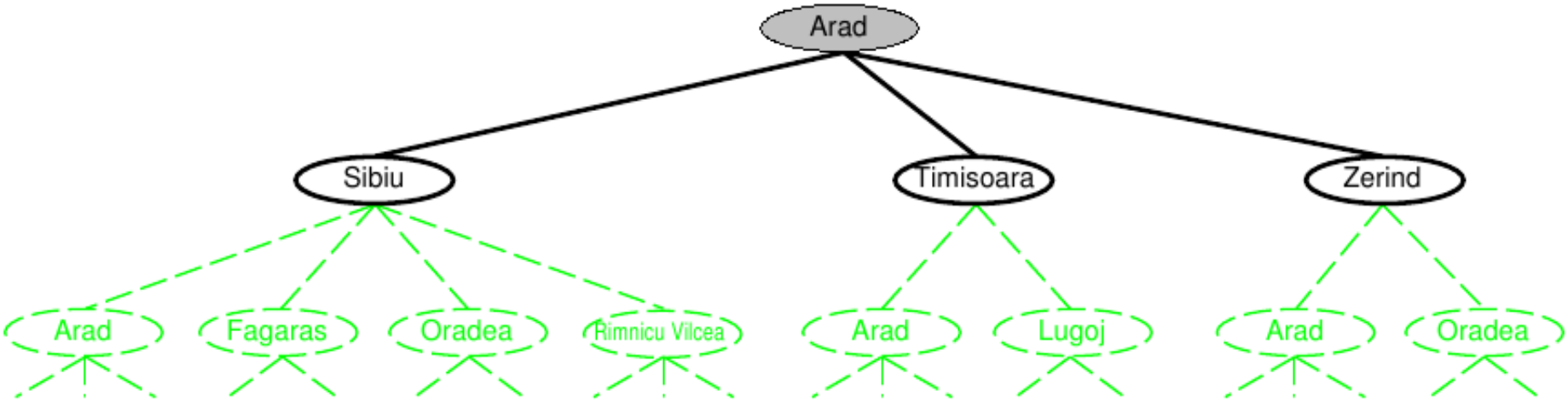
Busca por Soluções

Árvore de Busca



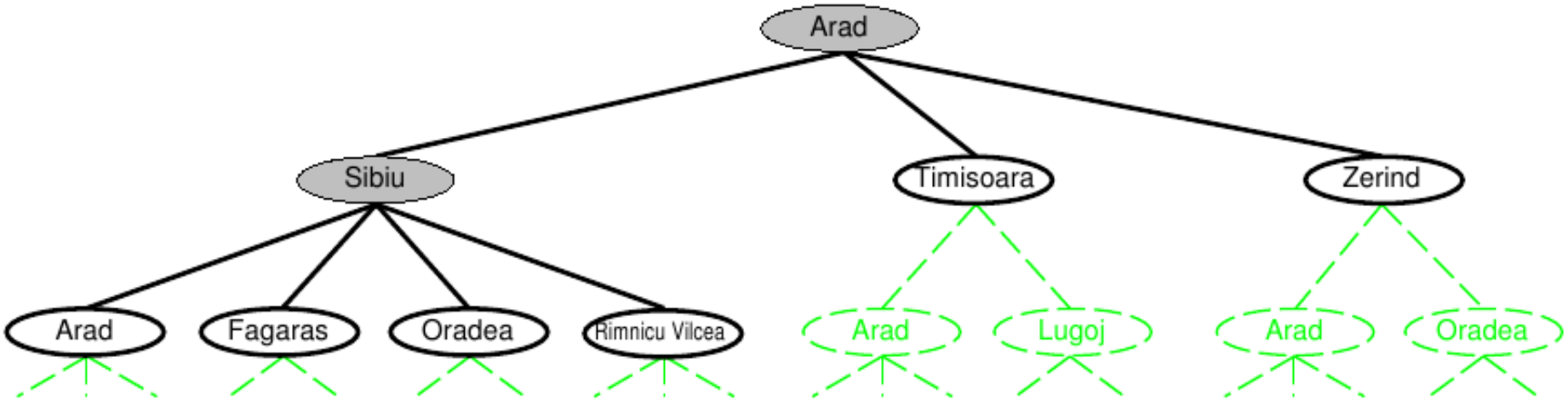
- Nós gerados estão contornados de preto
- Nós expandidos estão preenchidos de cinza

Árvore de Busca



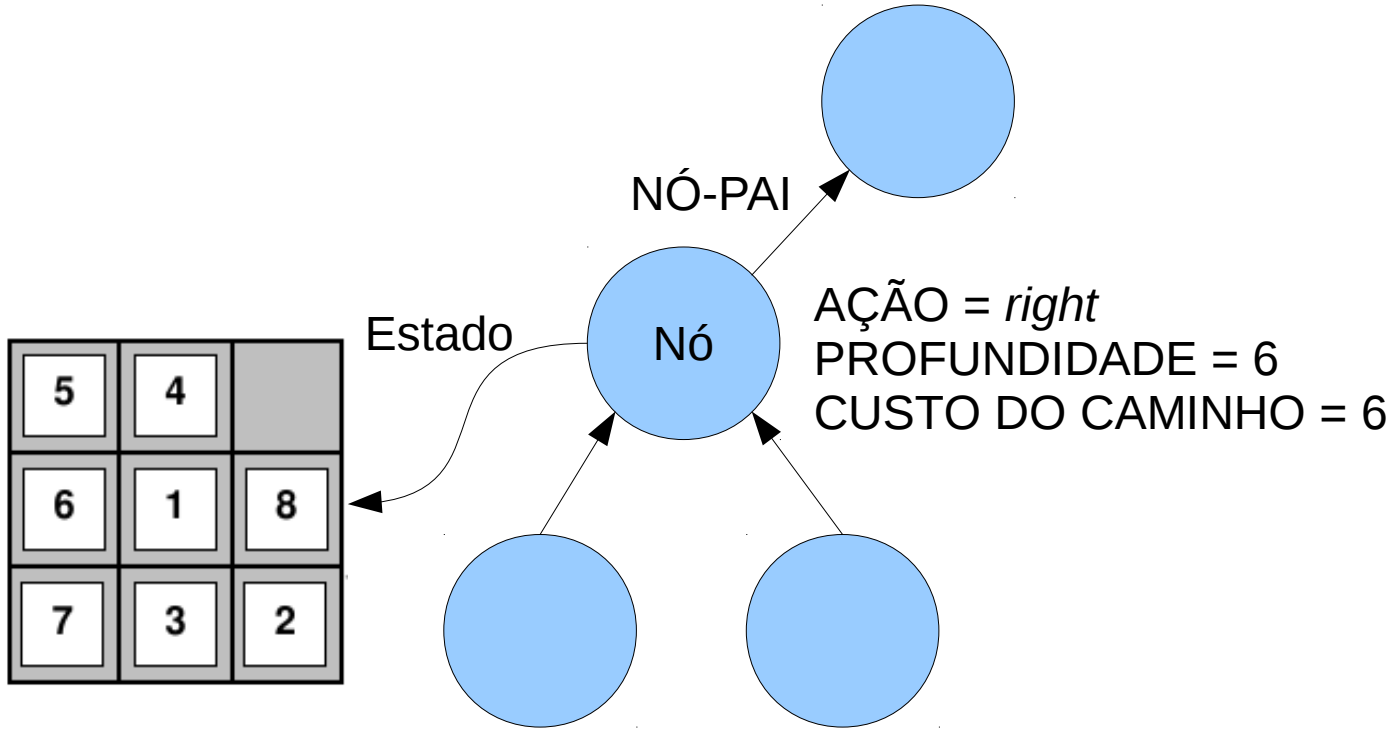
- Nós gerados estão contornados de preto
- Nós expandidos estão preenchidos de cinza

Árvore de Busca



- Nós gerados estão contornados de preto
- Nós expandidos estão preenchidos de cinza

Estrutura de um Nó



- Pode ser um grafo em vez de uma árvore
- Note que NÓ ≠ ESTADO

Franja

- Utilizada para armazenar nós que foram gerados, mas não expandidos
  - Cada um dos elementos da franja é um nó folha
    - Não possui descendentes na árvore
- A franja é uma estrutura abstrata
  - Pode ser concretizada por meio de uma fila, pilha, fila de prioridades, conforme a necessidade
- A estratégia de busca será definida pela função que seleciona o próximo nó da franja a ser expandido

Métodos da Franja

- CRIA-FRANJA(*elemento*,...)
  - Cria a franja com os elementos passados
- VAZIA*(franja)*
  - Retorna verdadeiro somente se a franja está vazia
- PRIMEIRO(*franja*)
  - Retorna o primeiro elemento da franja
- REMOVE-PRIMEIRO*(franja)*
  - Retorna PRIMEIRO() e o remove da franja
- INSERE(*elemento, franja*)
  - Insere o elemento na franja e retorna a franja resultante
- INSERE-TODOS(*elementos, franja*)
  - Insere um conjunto de elementos na franja e retorna a franja resultante

Algoritmo Genérico para Busca em Árvores

```
função BUSCA-EM-ARVORE(problema, franja) retorna uma solução, ou falha
  entradas: problema, a formulação do problema
            franja, a franja vazia

  franja ← INSERT(CRIA-NO(ESTADO-INICIAL[problema]),franja)
  laço faça
    se VAZIA(franja) então retorne falha
    nó ← REMOVE-PRIMEIRO(franja)
    se TESTE-OBJETIVO[problema](ESTADO[nó]) então retorne SOLUÇÃO(nó)
    franja ← INSERE-TODOS(EXPANDA(nó, problema),franja)
```

A função SOLUÇÃO retorna a sequência de ações obtida, seguindo os ponteiros dos nós pais até a raiz da árvore

Função Expanda

```
função EXPANDA(nó, problema) retorna um conjunto de nós
  entradas: nó, um nó da árvore
            problema, a formulação do problema

  descendentes ← o conjunto vazio
  para cada ⟨ação, resultado⟩ em FUNÇÃO-SUCESSOR[problema](ESTADO[nó]) faça
    s ← um novo nó
    ESTADO[s] ← resultado
    NÓ-PAI[s] ← nó
    AÇÃO[s] ← ação
    CUSTO-DO-CAMINHO[s] ← CUSTO-DO-CAMINHO[nó] + CUSTO-DO-PASSO(ESTADO[nó],ação,resultado)
    PRONFUNDIDADE[s] ← PROFUNDIDADE[nó] + 1
    adiciona s a descendentes

  retorna descendentes
```

Medindo o Desempenho do Método de Resolução de Problemas

- A saída de um algoritmo de resolução de problemas é uma *falha* ou uma *solução*
  - Alguns algoritmos podem ficar presos em um *loop* infinito
- O desempenho do algoritmo será medido em 4 critérios
  - Completude
    - O algoritmo garante que uma solução será encontrada?
  - Optimalidade
    - O algoritmo encontra a melhor solução?
  - Complexidade no Tempo
    - Quanto tempo leva para encontrar uma solução
  - Complexidade no Espaço
    - Quanta memória é necessária para realizar a tarefa

Complexidade

- Geralmente, a complexidade é medida em termos do tamanho do grafo do espaço de estados
  - Em IA este grafo frequentemente é infinito!
- A complexidade será medida usando:
  - ***b*** → Fator de ramificação
    - Máximo número de descendentes do nó
  - ***d*** → Profundidade do nó objetivo menos profundo
  - ***m*** → Profundidade do nó mais profundo da árvore
- Complexidade de tempo é geralmente medida em número de nós gerados
- Complexidade de espaço é medida em número de nós armazenados na memória

Eficiência do Algoritmo

- Custo da Busca
  - Complexidade de Tempo
  - Uso de memória
- Custo Total
  - Custo da Busca
  - Custo do Caminho encontrado na Solução
  - O Custo Total pode requerer a soma de grandezas diferentes (e.g. *Km* e milissegundos)
    - Não existe conversão padrão!
    - No caso da viagem na Romênia, pode-se estimar a média de velocidade do carro e converter *km* em milissegundos
      - Pode-se determinar quando buscar outras soluções se torna contra-produtivo

Resolução de Problemas

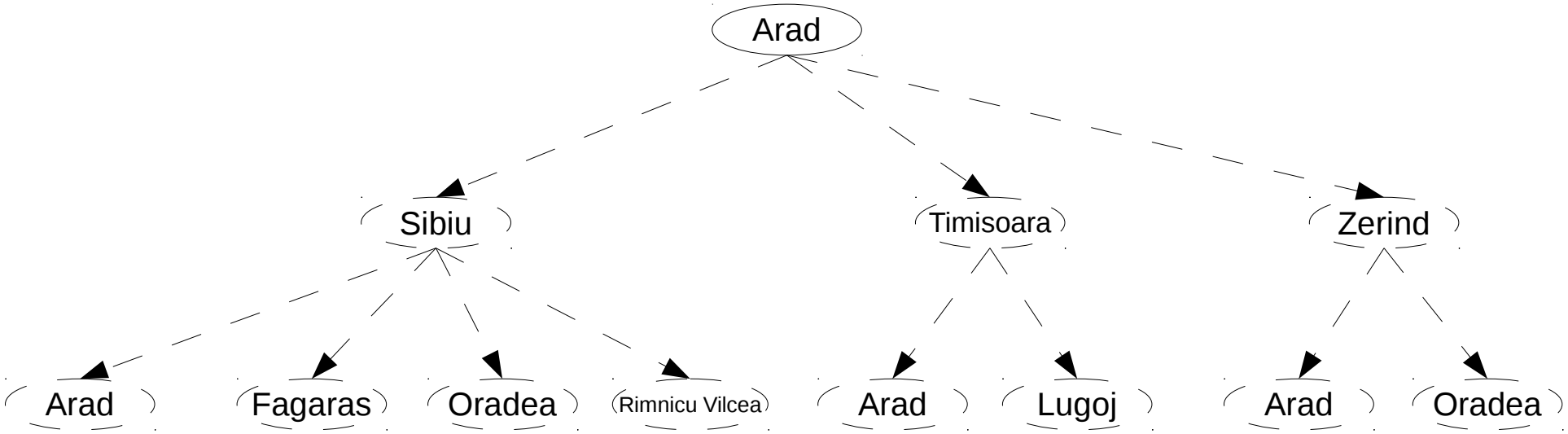
Estratégias de Busca Exaustiva
(ou busca às cegas)

Busca Exaustiva

- Busca Exaustiva
  - Busca às cegas
  - Busca desinformada
  - Não existe informação adicional além da definição do problema
    - Tudo que pode ser feito é gerar estados e verificar se é um objetivo
- As estratégias de Busca Exaustivas são diferenciadas pela ordem na qual os nós são expandidos

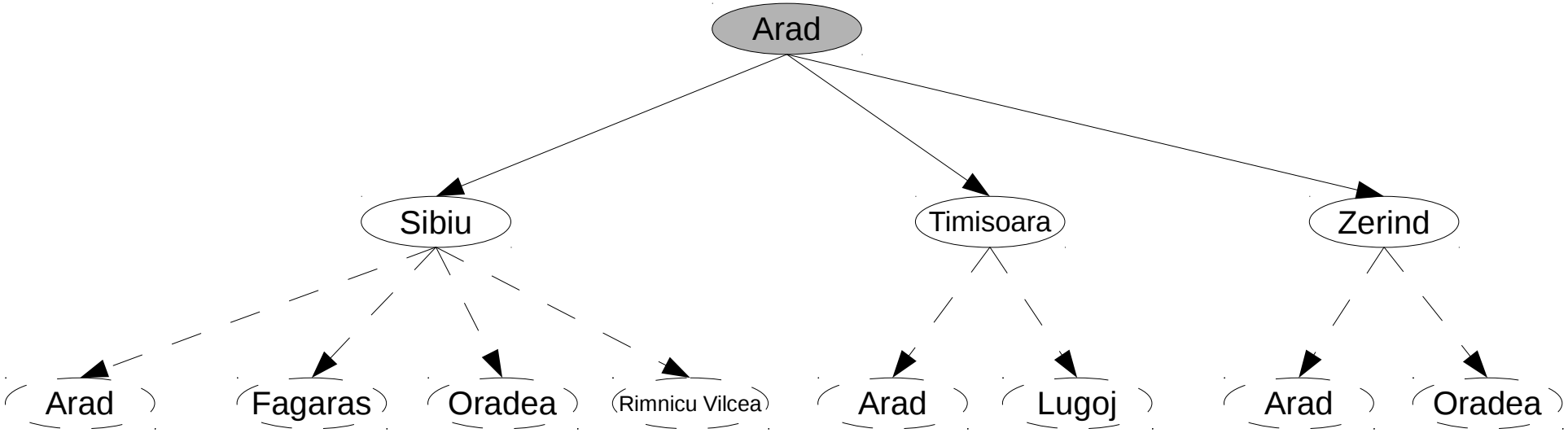
Busca em Largura

- O nó menos profundo não expandido é expandido
  - Utiliza uma FIFO como *franja*



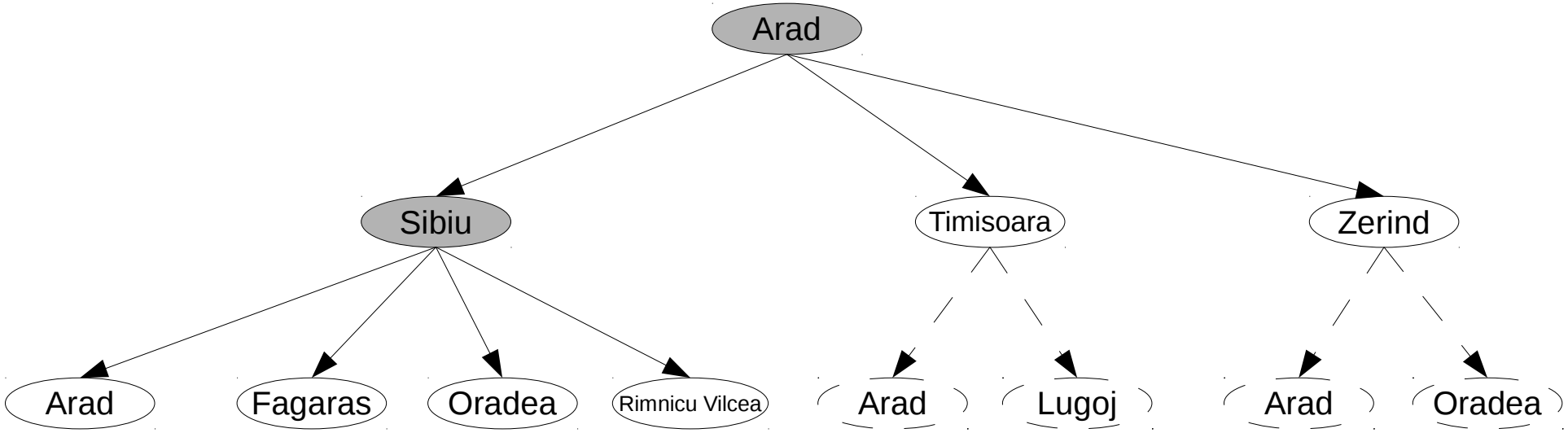
Busca em Largura

- O nó menos profundo não expandido é expandido
  - Utiliza uma FIFO como *franja*



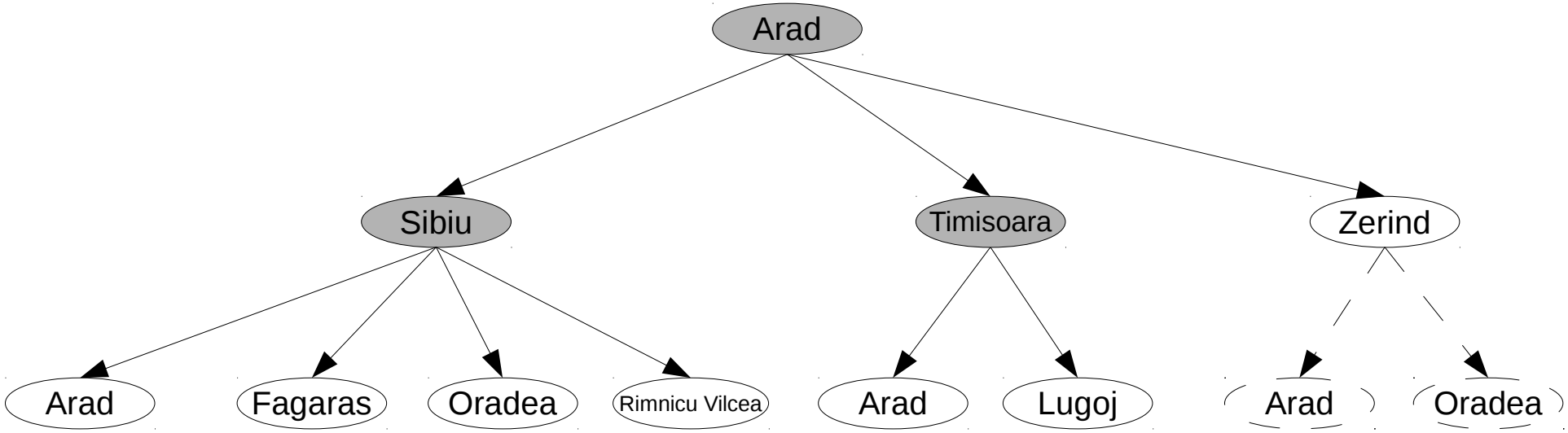
Busca em Largura

- O nó menos profundo não expandido é expandido
  - Utiliza uma FIFO como *franja*



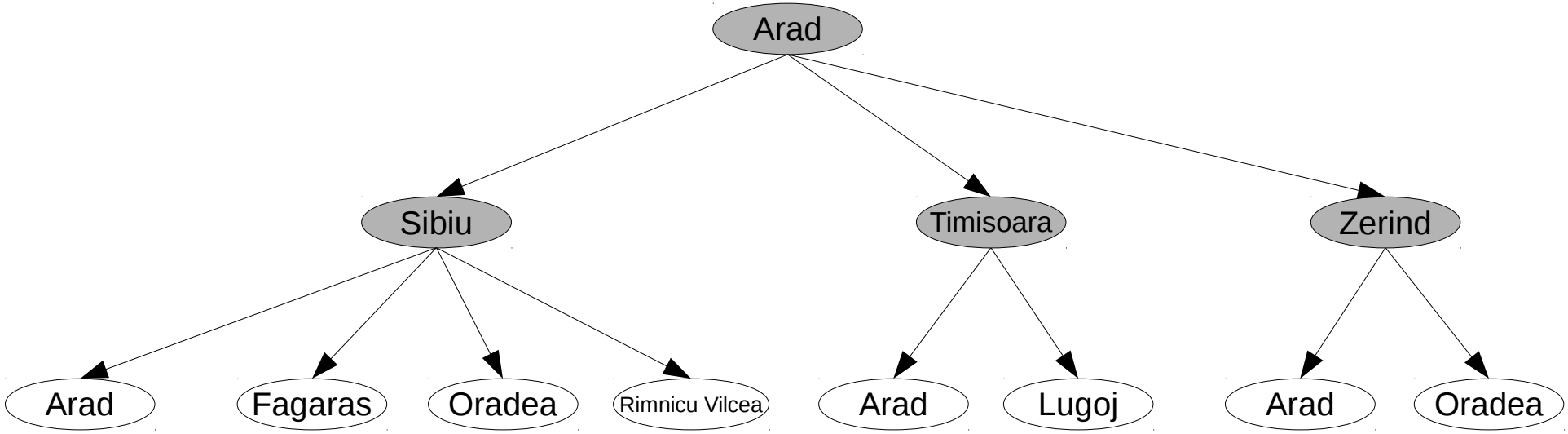
Busca em Largura

- O nó menos profundo não expandido é expandido
  - Utiliza uma FIFO como *franja*



Busca em Largura

- O nó menos profundo não expandido é expandido
  - Utiliza uma FIFO como *franja*



Análise

- Completude
  - Garante (Se *b* é finito)
- Optimalidade
  - Garante se o custo for igual para todas ações
  - Não garante em caso contrário
- Complexidade no Tempo
  - Número de nós Gerados (Pior caso)

$$b + b^2 + b^3 + ... + b^d + (b^{d+1} - b) = O(b^{d+1})$$

- Complexidade no espaço
  - Todos os nós gerados permanecem na memória

$$b + b^2 + b^3 + ... + b^d + (b^{d+1} - b) = O(b^{d+1})$$

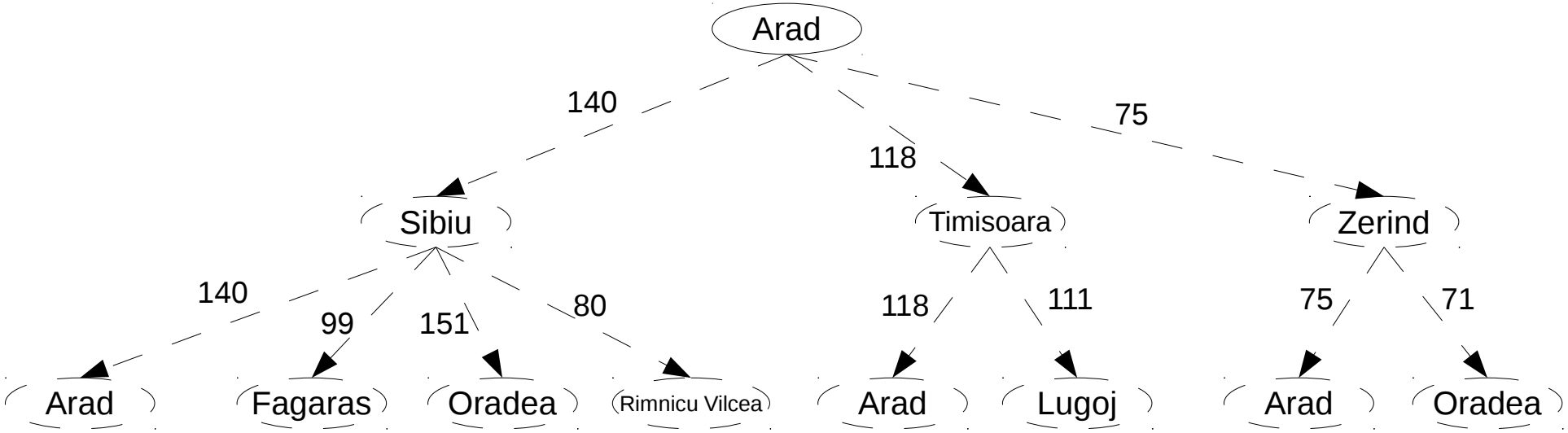
Análise em Números

Profundidade	Nós	Tempo	Memória
2	1.100	,11 segundos	1 MB
4	111.100	11 segundos	106 MB
6	10^7	19 minutos	10 GB
8	10^9	31 horas	1 TB
10	10^{11}	129 dias	101 TB
12	10^{13}	35 anos	10 PB
14	10^{15}	3.523 anos	1 EB

Assumindo *b*=10; 10.000 nós por segundo; 1KB/nó

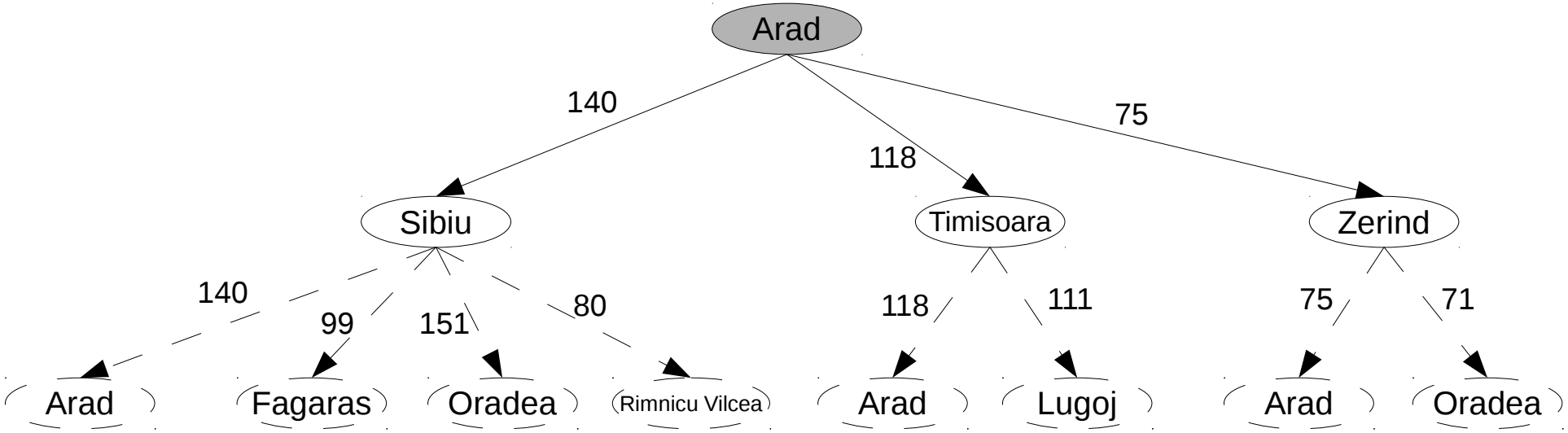
Busca em Custo Uniforme

- O nó não expandido com menor custo é expandido
  - Utiliza uma fila de prioridades como *franja*



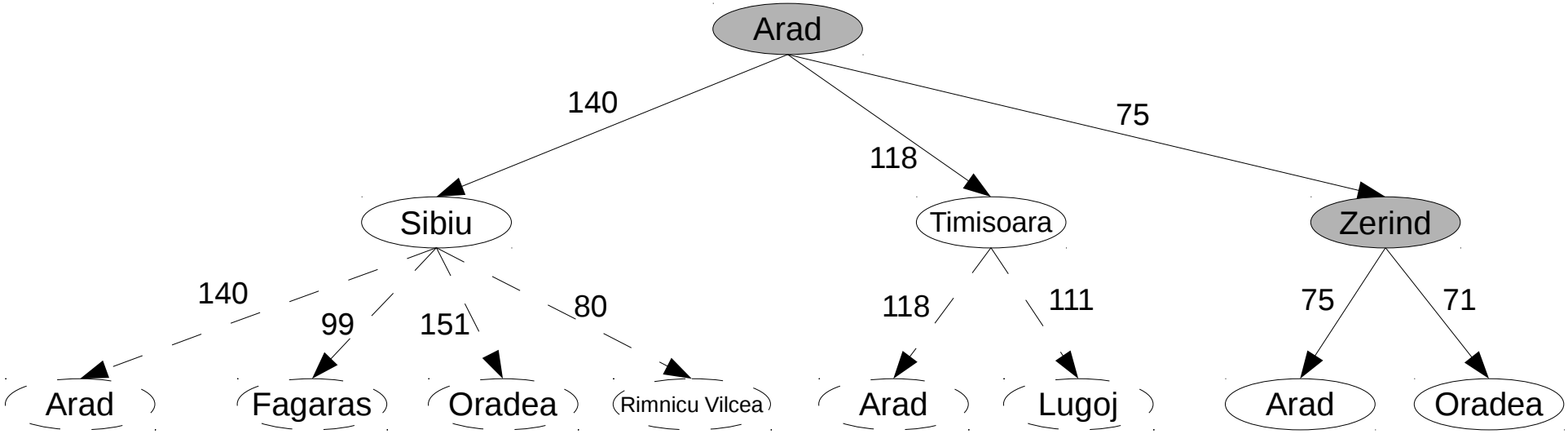
Busca em Custo Uniforme

- O nó não expandido com menor custo é expandido
  - Utiliza uma fila de prioridades como *franja*



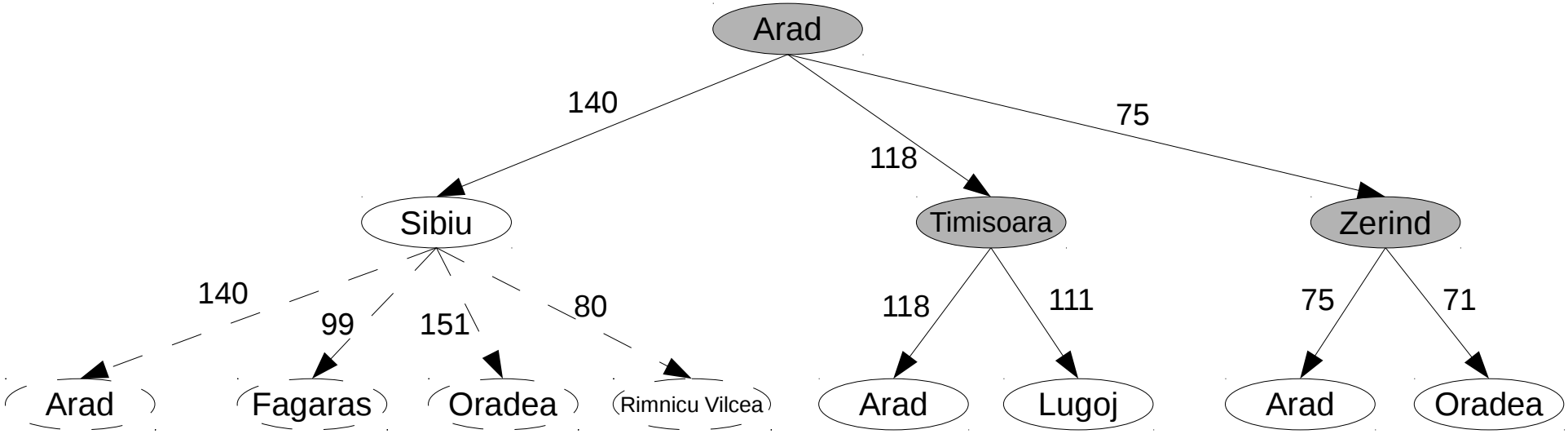
Busca em Custo Uniforme

- O nó não expandido com menor custo é expandido
  - Utiliza uma fila de prioridades como *franja*



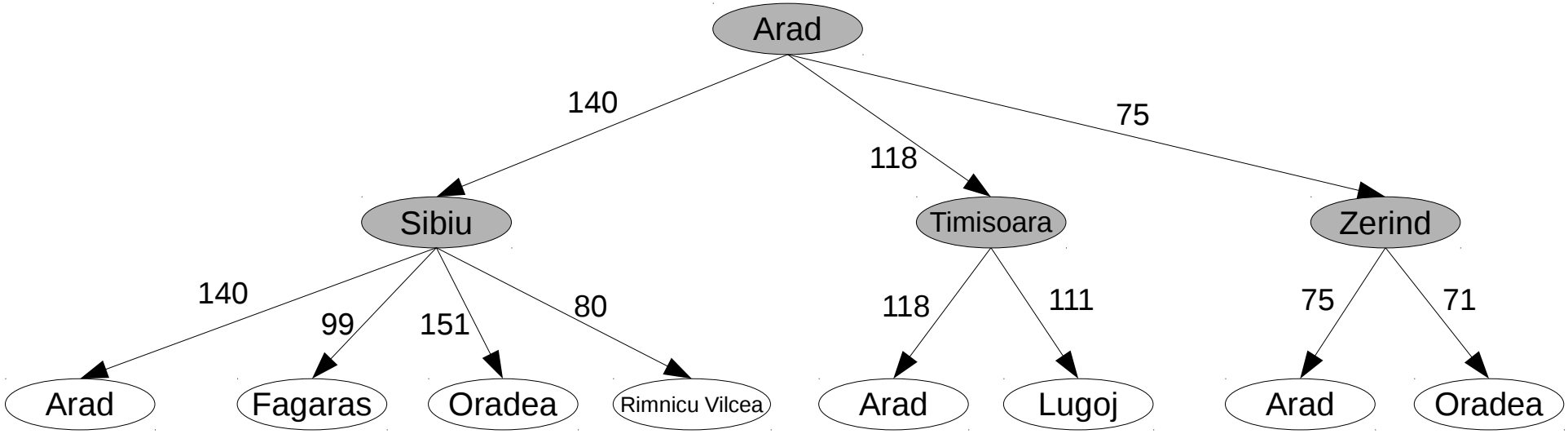
Busca em Custo Uniforme

- O nó não expandido com menor custo é expandido
  - Utiliza uma fila de prioridades como *franja*



Busca em Custo Uniforme

- O nó não expandido com menor custo é expandido
  - Utiliza uma fila de prioridades como *franja*



Análise

- Completude
  - Em geral não garante (*Loops* com custo 0)
    - Para resolver, cada caminho pode ter um custo mínimo ε
- Optimalidade
  - Garante se o custo mínimo do passo for ε
- Complexidade no Tempo
  - Não é simples de ser calculada

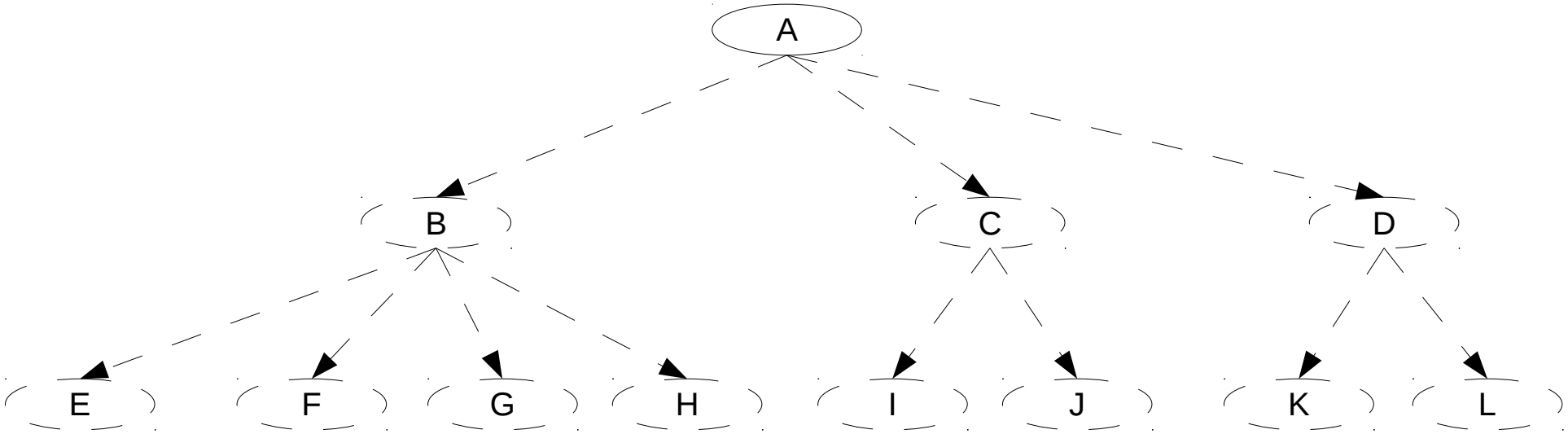
$$O(b^{1+[C^*/\epsilon]}), \text{ onde } C^* \text{ é o custo da solução ótima}$$

- Complexidade no espaço
  - Todos os nós gerados permanecem na memória

$$O(b^{1+[C^*/\epsilon]}), \text{ onde } C^* \text{ é o custo da solução ótima}$$

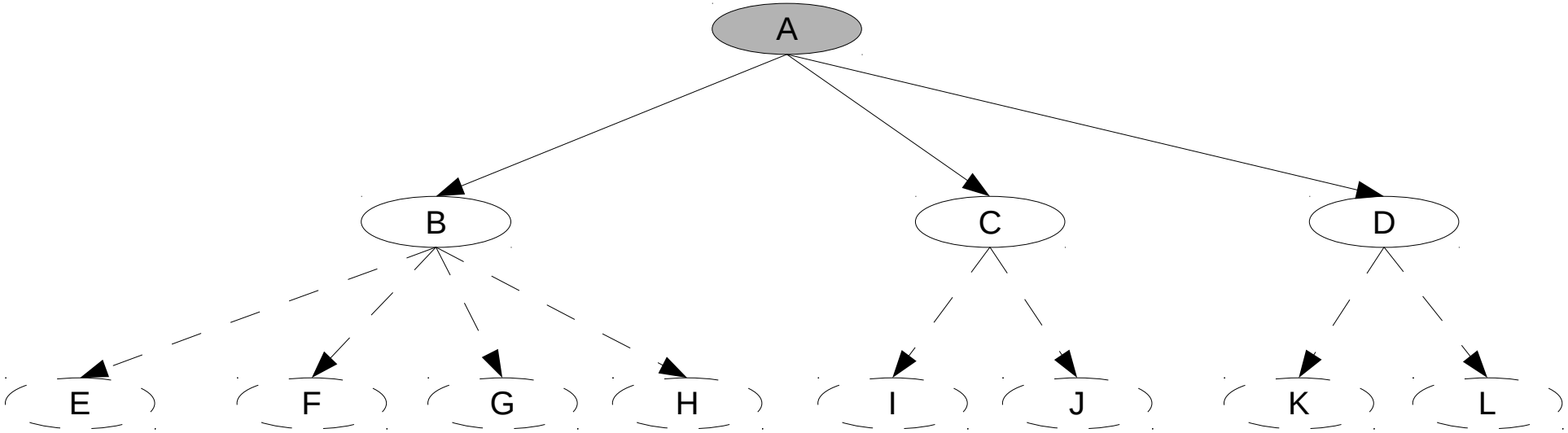
Busca em Profundidade

- O nó mais profundo não expandido é expandido
  - Utiliza uma LIFO como *franja*



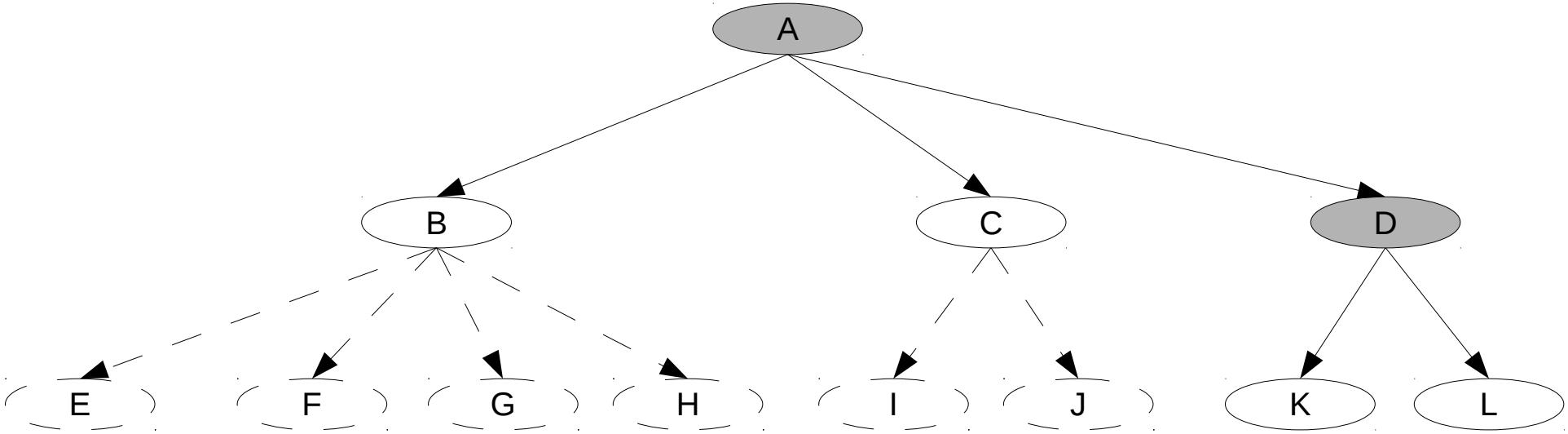
Busca em Profundidade

- O nó mais profundo não expandido é expandido
  - Utiliza uma LIFO como *franja*



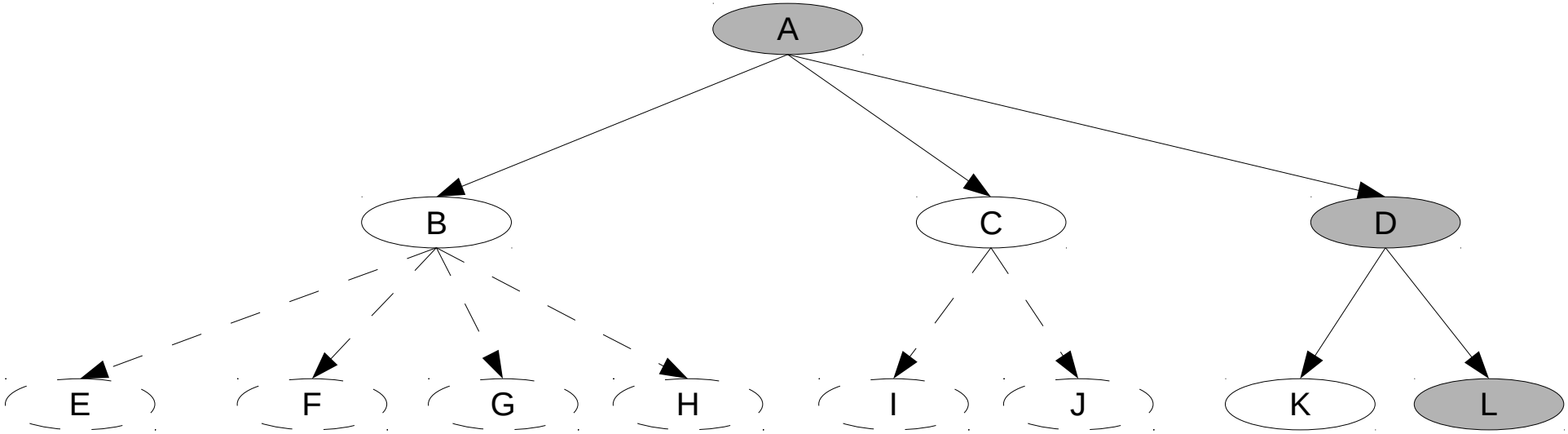
Busca em Profundidade

- O nó mais profundo não expandido é expandido
  - Utiliza uma LIFO como *franja*



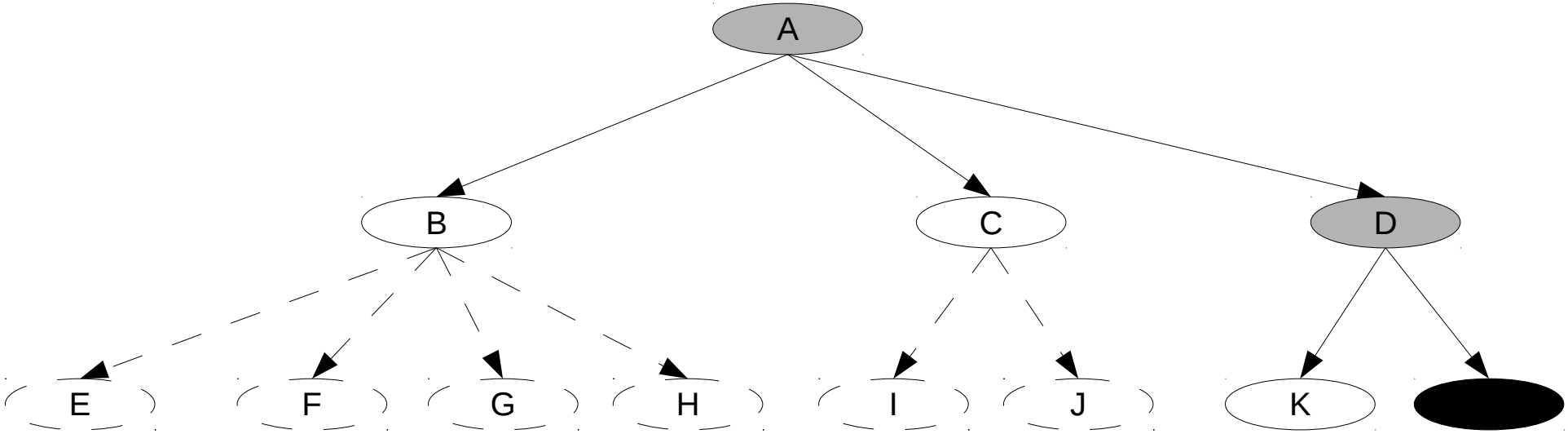
Busca em Profundidade

- O nó mais profundo não expandido é expandido
  - Utiliza uma LIFO como *franja*



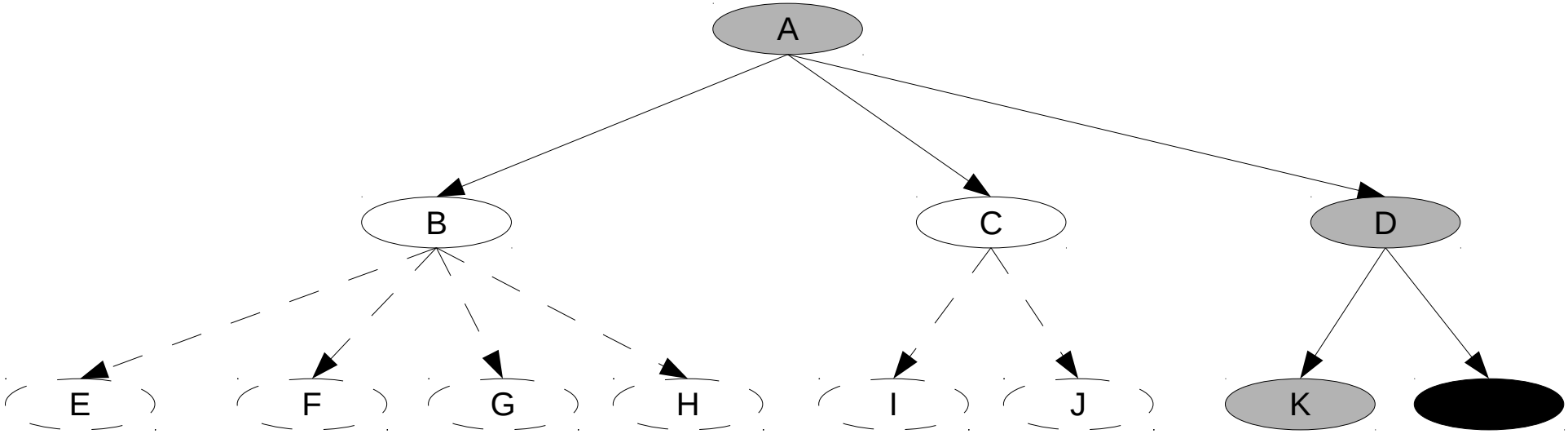
Busca em Profundidade

- O nó mais profundo não expandido é expandido
  - Utiliza uma LIFO como *franja*



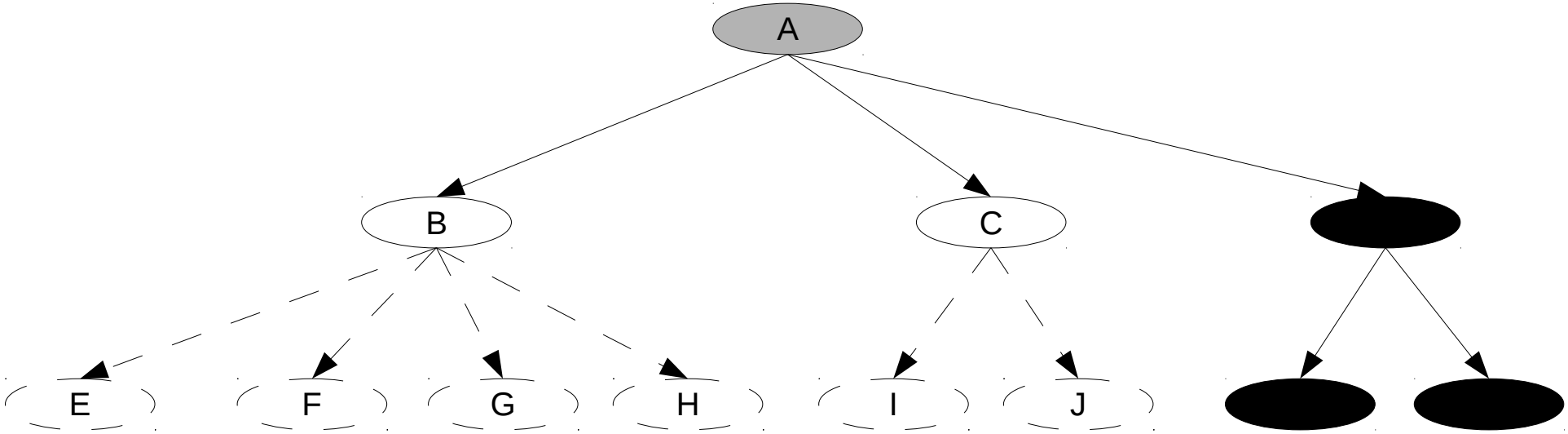
Busca em Profundidade

- O nó mais profundo não expandido é expandido
  - Utiliza uma LIFO como *franja*



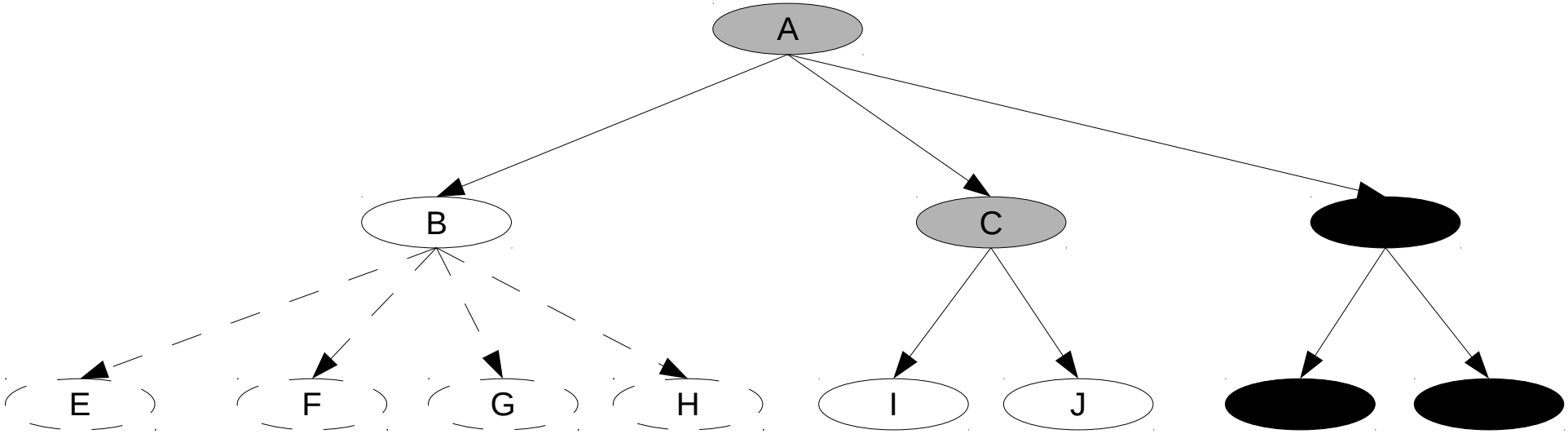
Busca em Profundidade

- O nó mais profundo não expandido é expandido
  - Utiliza uma LIFO como *franja*



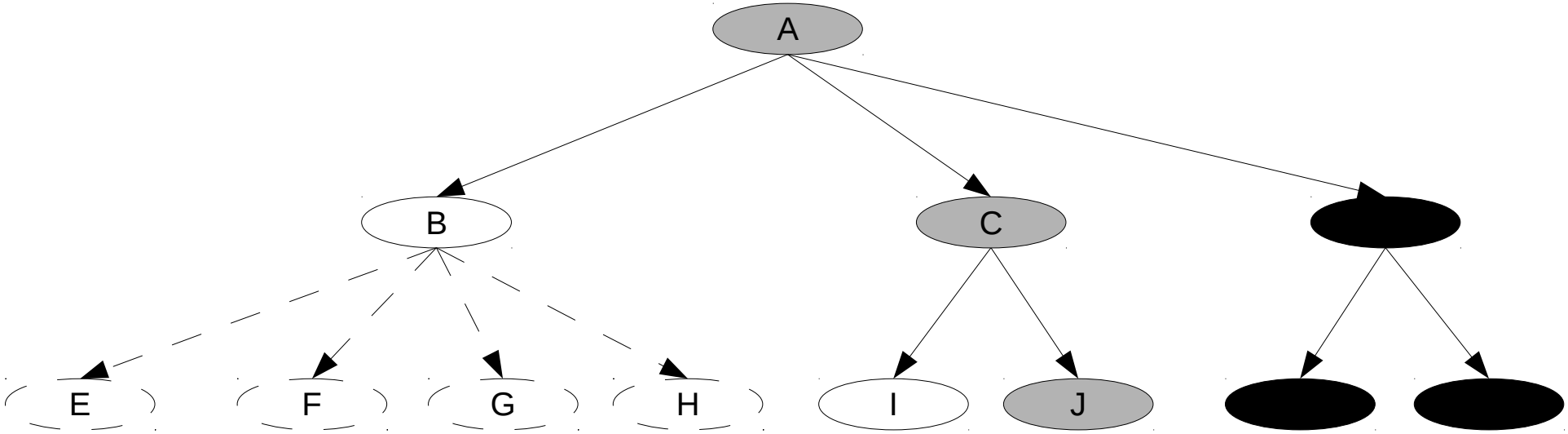
Busca em Profundidade

- O nó mais profundo não expandido é expandido
  - Utiliza uma LIFO como *franja*



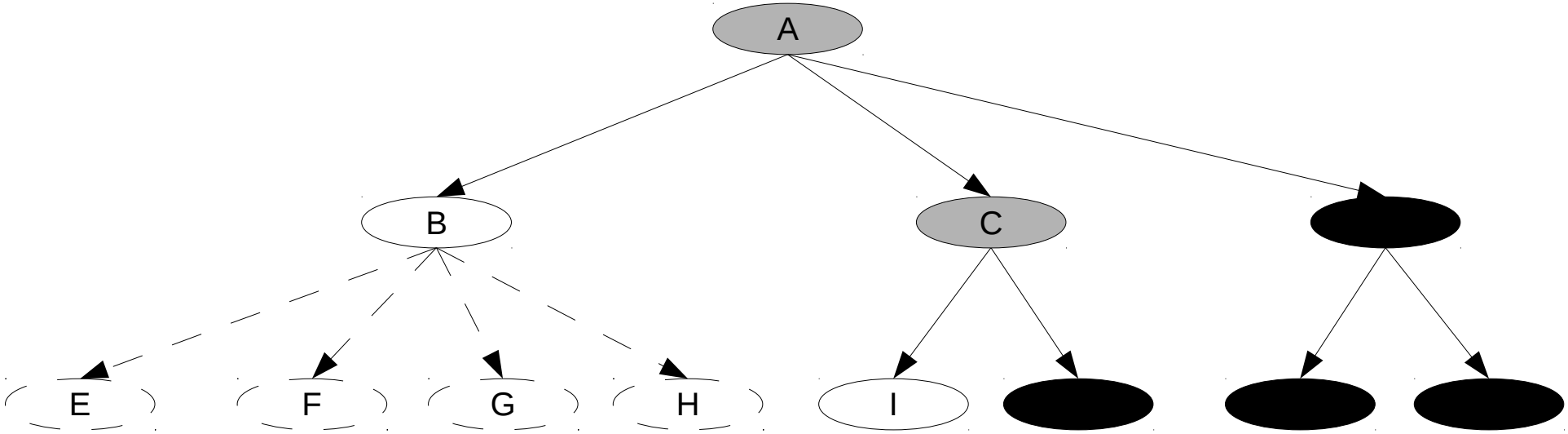
Busca em Profundidade

- O nó mais profundo não expandido é expandido
  - Utiliza uma LIFO como *franja*



Busca em Profundidade

- O nó mais profundo não expandido é expandido
  - Utiliza uma LIFO como *franja*



Análise

- Completude
  - Garante (Se o espaço de estados é finito)
  - Não garante se o espaço for infinito ou com *loops*
    - Necessário uma abordagem para evitar caminhos repetidos
- Optimalidade
  - Não
- Complexidade no Tempo
  - Número de nós Gerados (Pior caso)

    $O(b^m)$
- Complexidade no espaço linear

  $O(bm)$

Problema

- O valor de *m* pode ser muito maior que o de *d*
  - *m* pode ser infinito, inclusive!

Modificações

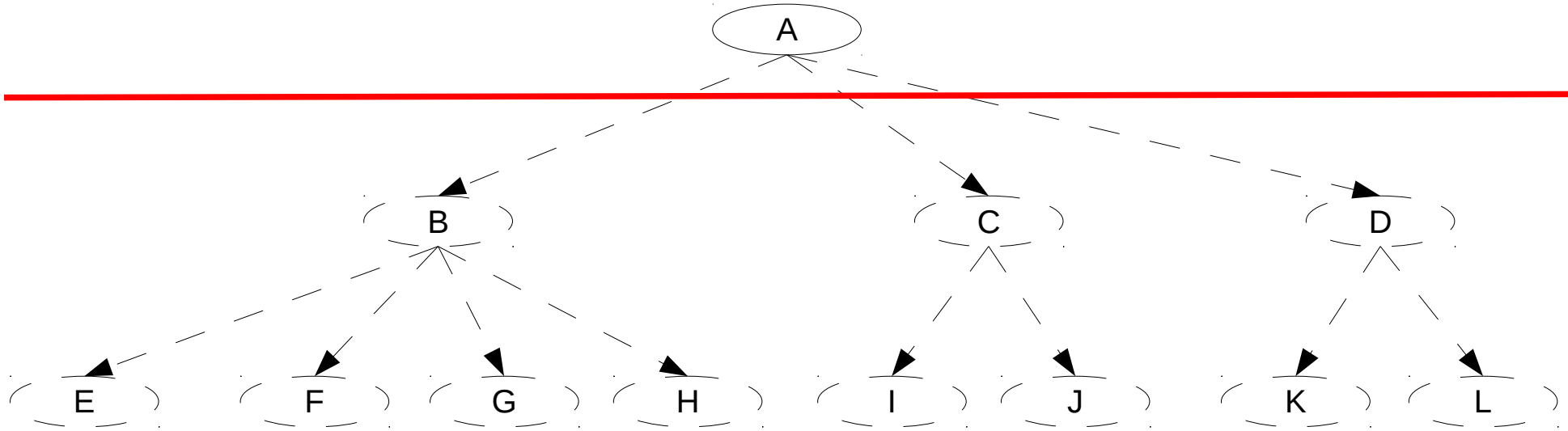
- *Backtracking*
  - Somente um descendente é gerado de cada vez
    - Em vez de todos os descendentes
    - Cada um dos nós parcialmente expandidos deve se *lembrar* quais nós já foram gerados
    - Complexidade de Espaço $O(m)$
  - Pode-se também modificar o estado do nó, em vez de copiar e modificar
    - Neste caso, armazena-se apenas 1 estado e a complexidade no espaço diz respeito apenas às ações

Busca com Profundidade Limitada

- Busca em profundidade com profundidade limitada em *l*
  - A busca torna-se mais incompleta, pois se $l < d$, a solução não pode ser encontrada
- Estudando o problema pode-se chegar a um valor de *l* que garanta a solução
  - No caso da viagem pela Romênia, pode-se fazer $l = 19$, pois são, somente, 20 cidades
    - Mais estudo pode levar à conclusão que $l = 9$ é melhor, pois é a maior profundidade (diâmetro do grafo) entre as cidades
  - Busca com profundidade limitada pode terminar por falha, solução, ou quando o limite é alcançado (*cut-off)*

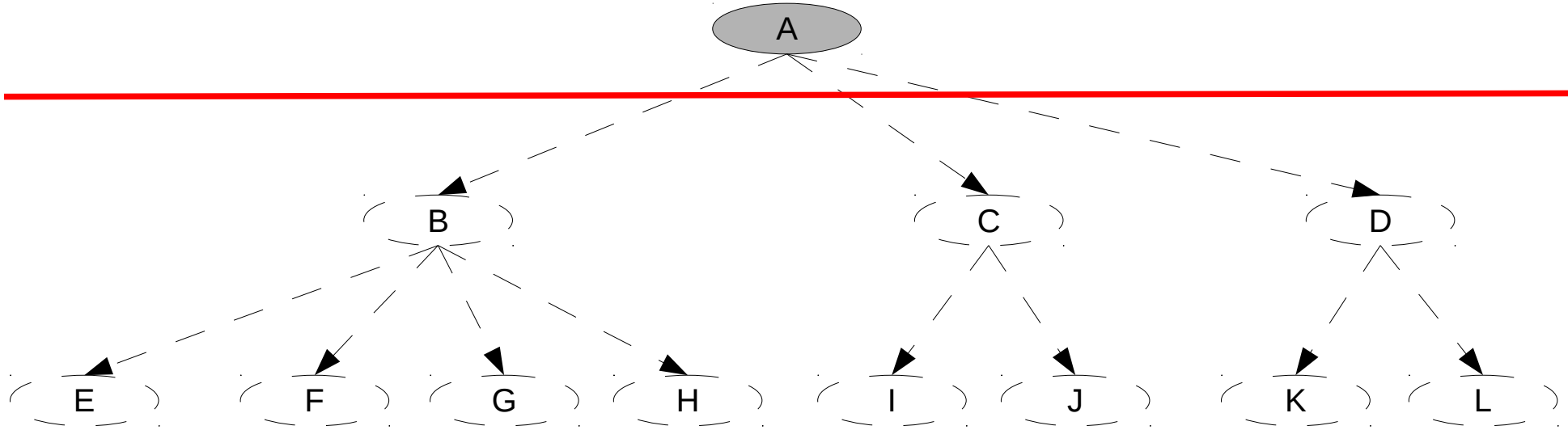
Busca em Profundidade Iterativa

- O nó mais profundo não expandido dentro do limite atual é expandido
  - Utiliza uma LIFO como *franja*



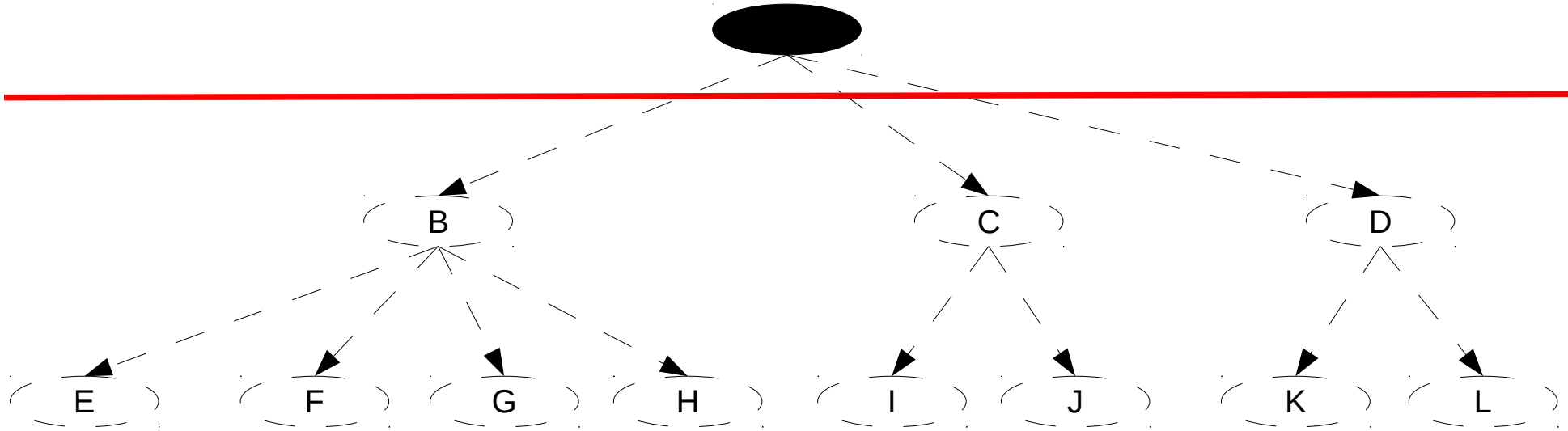
Busca em Profundidade Iterativa

- O nó mais profundo não expandido dentro do limite atual é expandido
  - Utiliza uma LIFO como *franja*



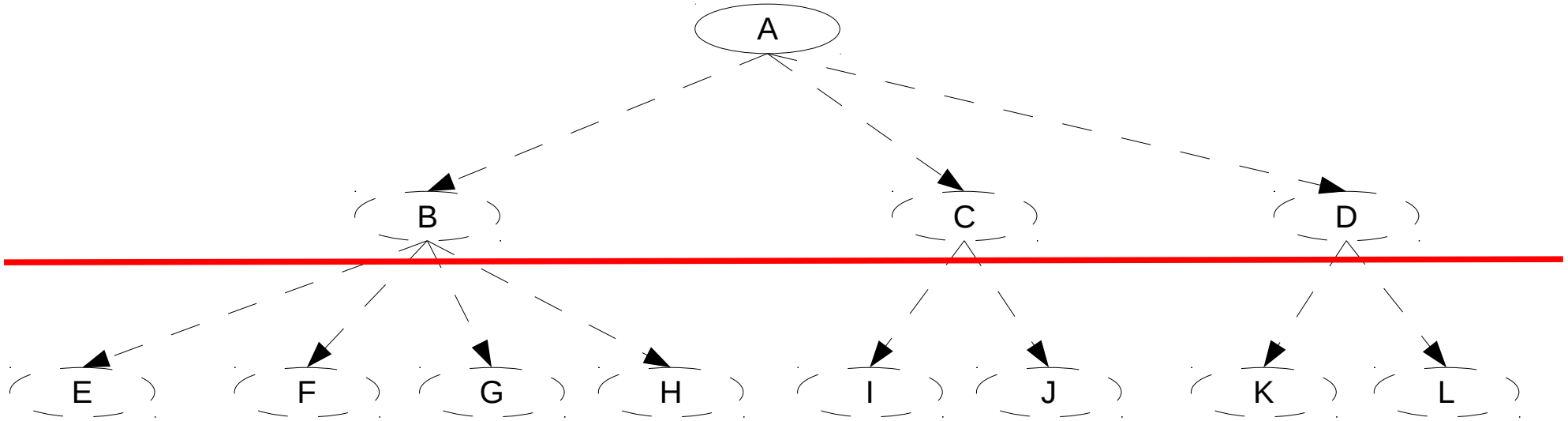
Busca em Profundidade Iterativa

- O nó mais profundo não expandido dentro do limite atual é expandido
  - Utiliza uma LIFO como *franja*



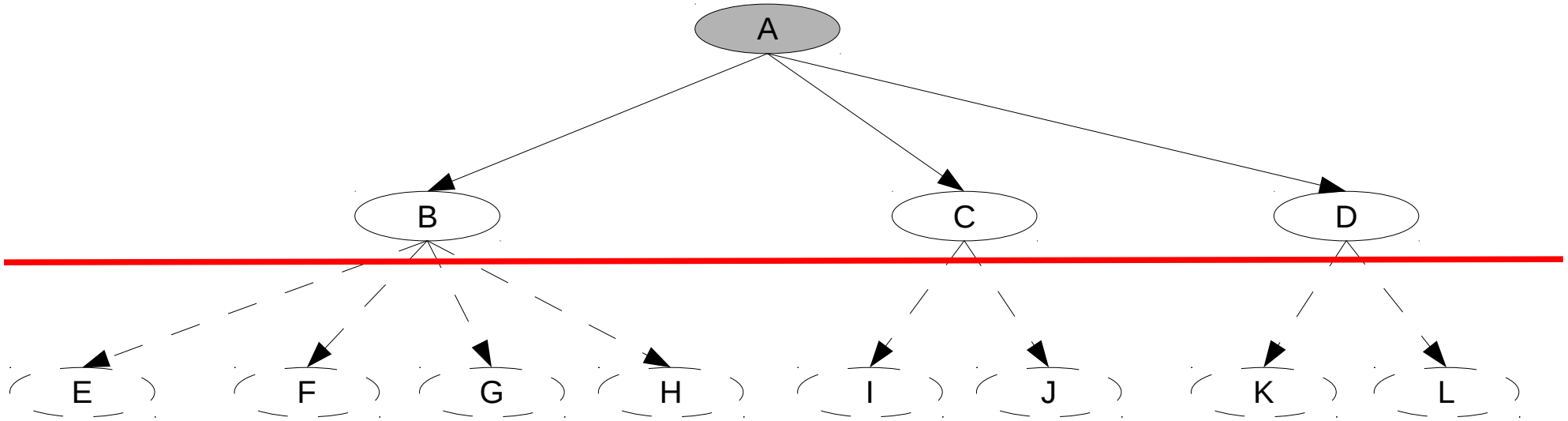
Busca em Profundidade Iterativa

- O nó mais profundo não expandido dentro do limite atual é expandido
  - Utiliza uma LIFO como *franja*



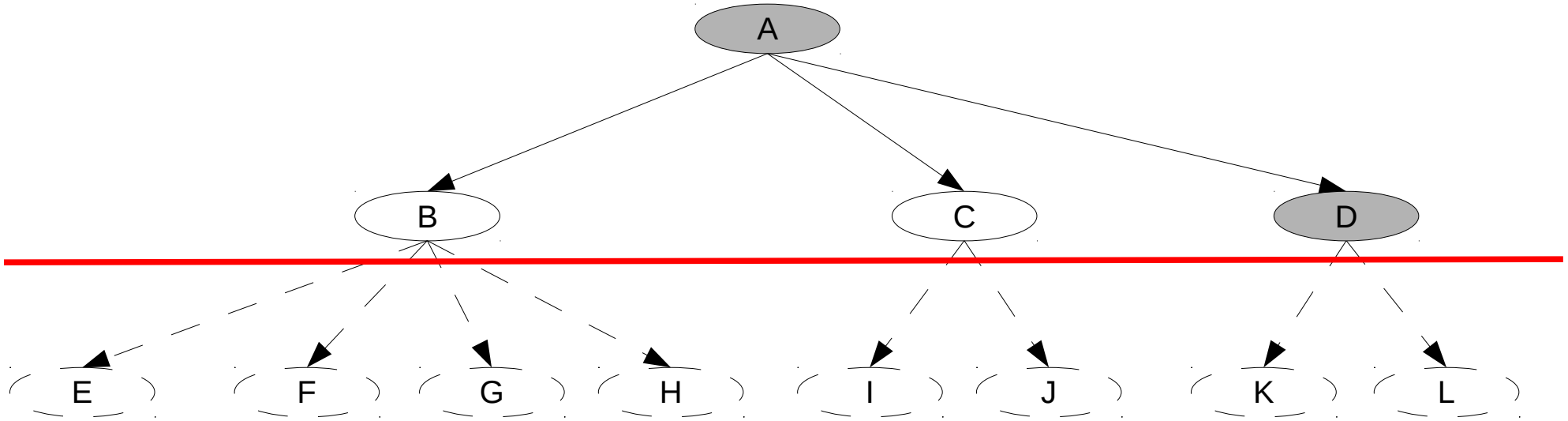
Busca em Profundidade Iterativa

- O nó mais profundo não expandido dentro do limite atual é expandido
  - Utiliza uma LIFO como *franja*



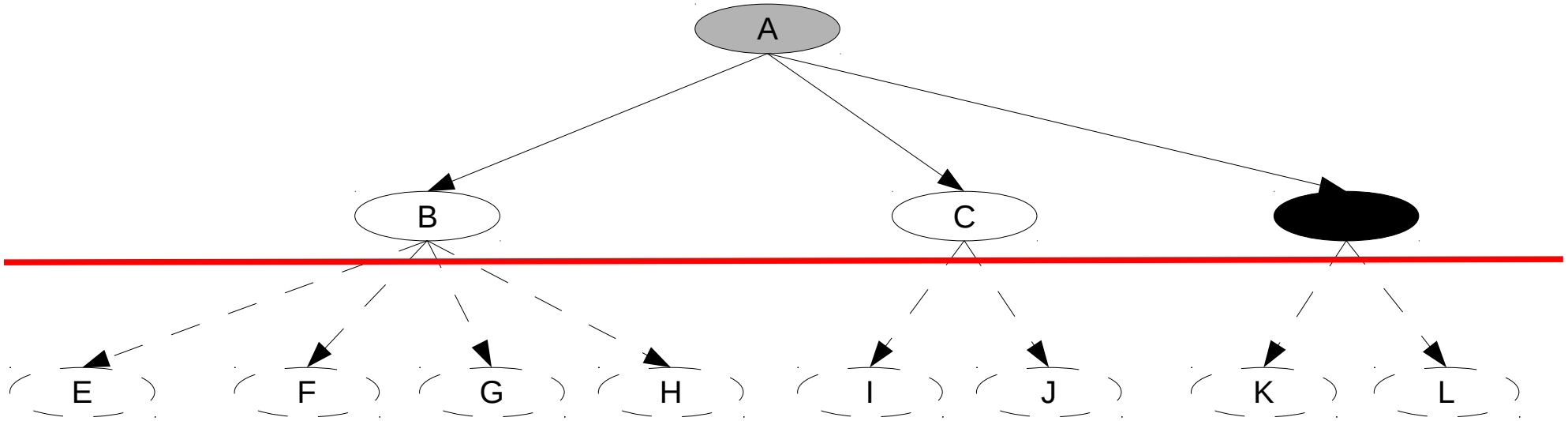
Busca em Profundidade Iterativa

- O nó mais profundo não expandido dentro do limite atual é expandido
  - Utiliza uma LIFO como *franja*



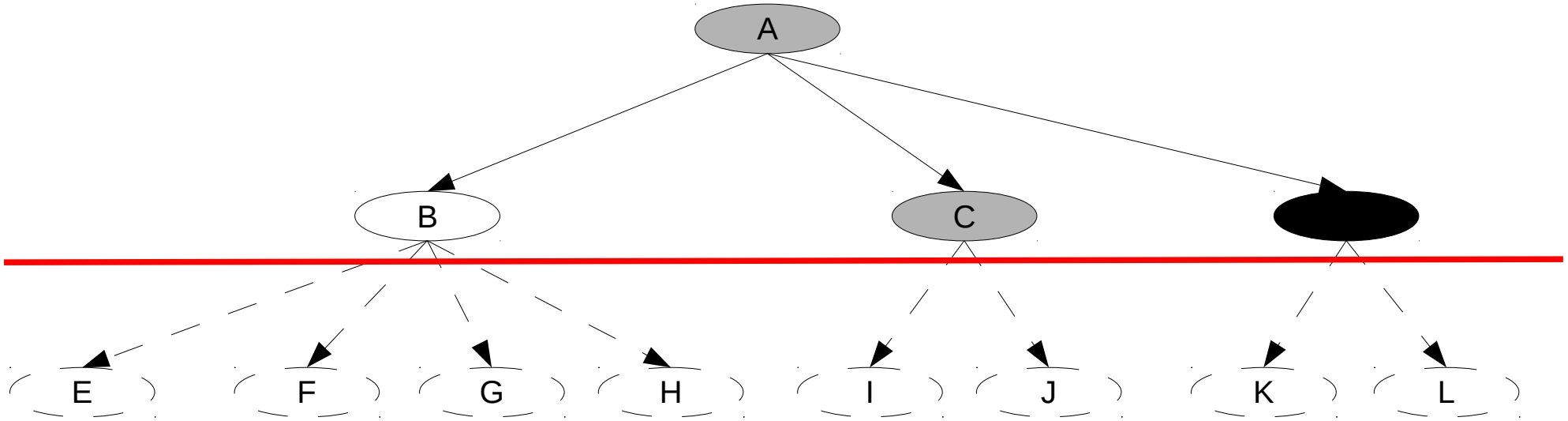
Busca em Profundidade Iterativa

- O nó mais profundo não expandido dentro do limite atual é expandido
  - Utiliza uma LIFO como *franja*



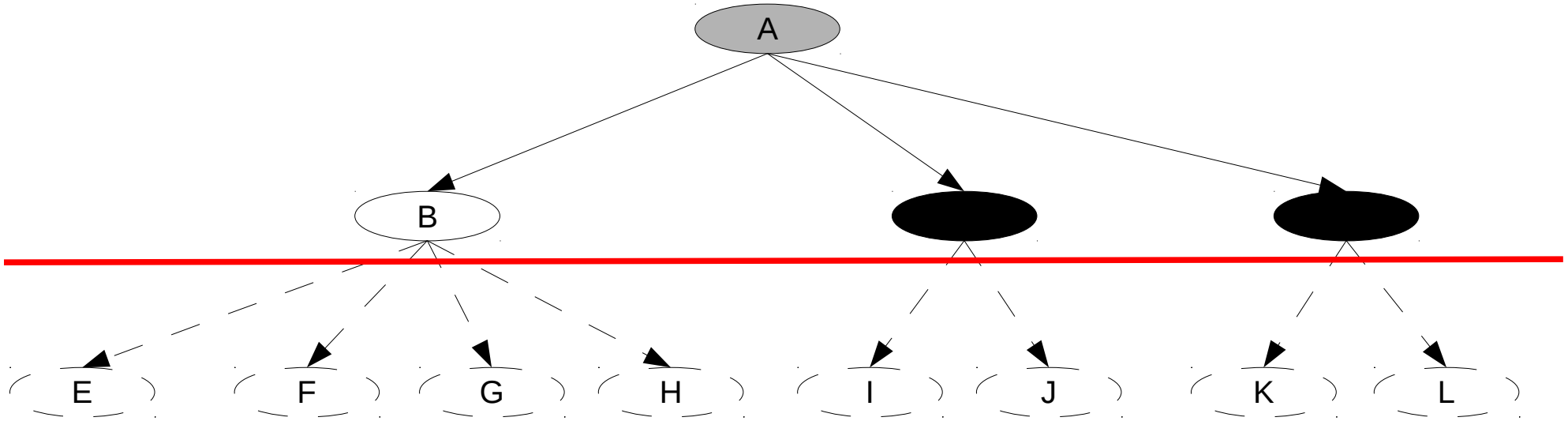
Busca em Profundidade Iterativa

- O nó mais profundo não expandido dentro do limite atual é expandido
  - Utiliza uma LIFO como *franja*



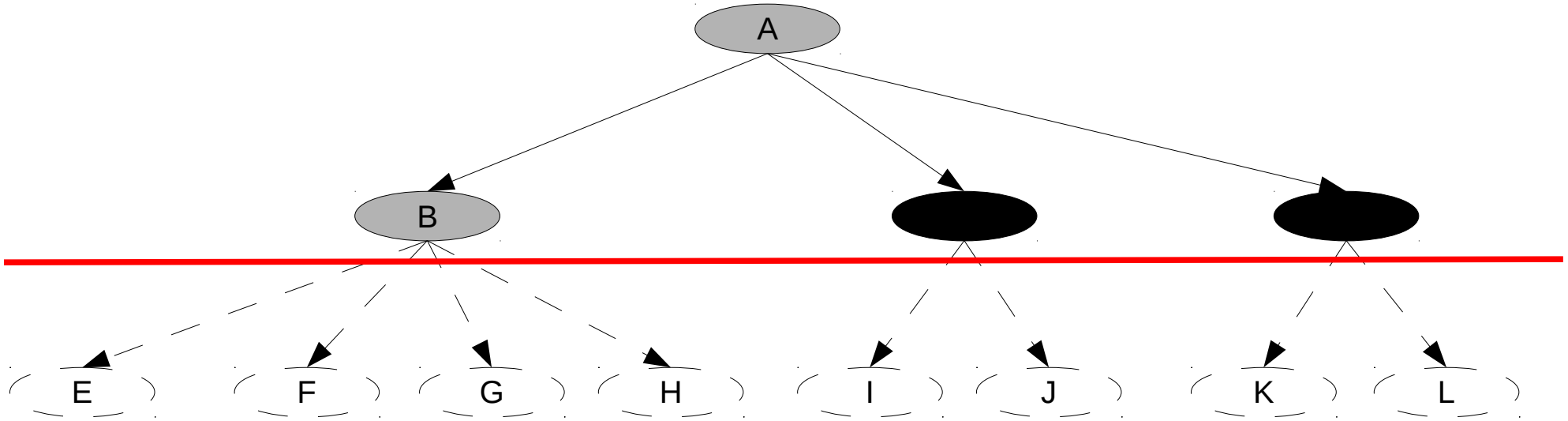
Busca em Profundidade Iterativa

- O nó mais profundo não expandido dentro do limite atual é expandido
  - Utiliza uma LIFO como *franja*



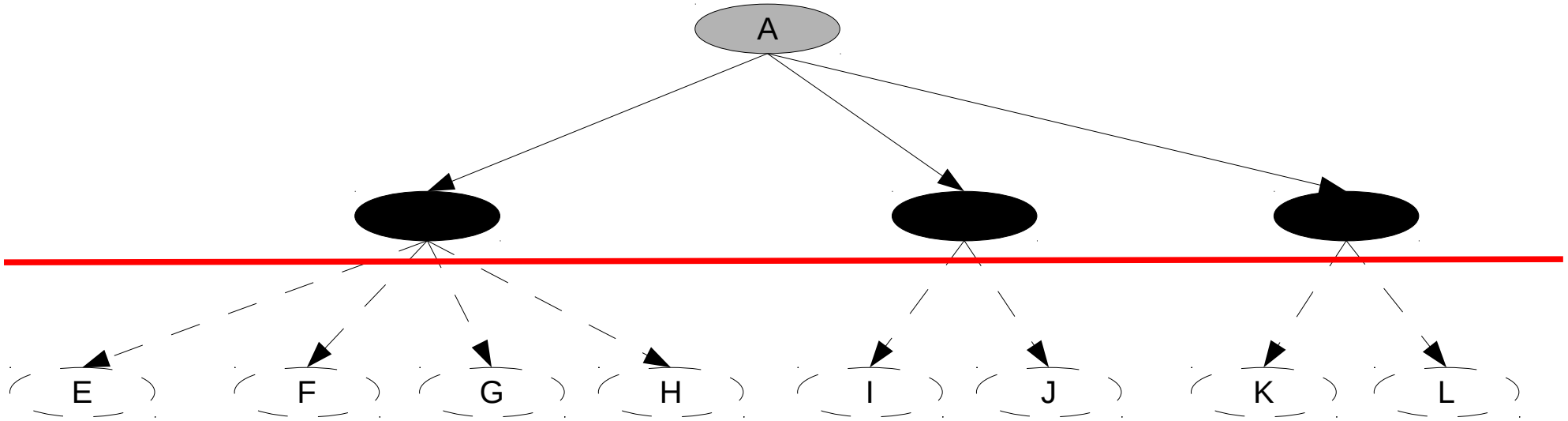
Busca em Profundidade Iterativa

- O nó mais profundo não expandido dentro do limite atual é expandido
  - Utiliza uma LIFO como *franja*



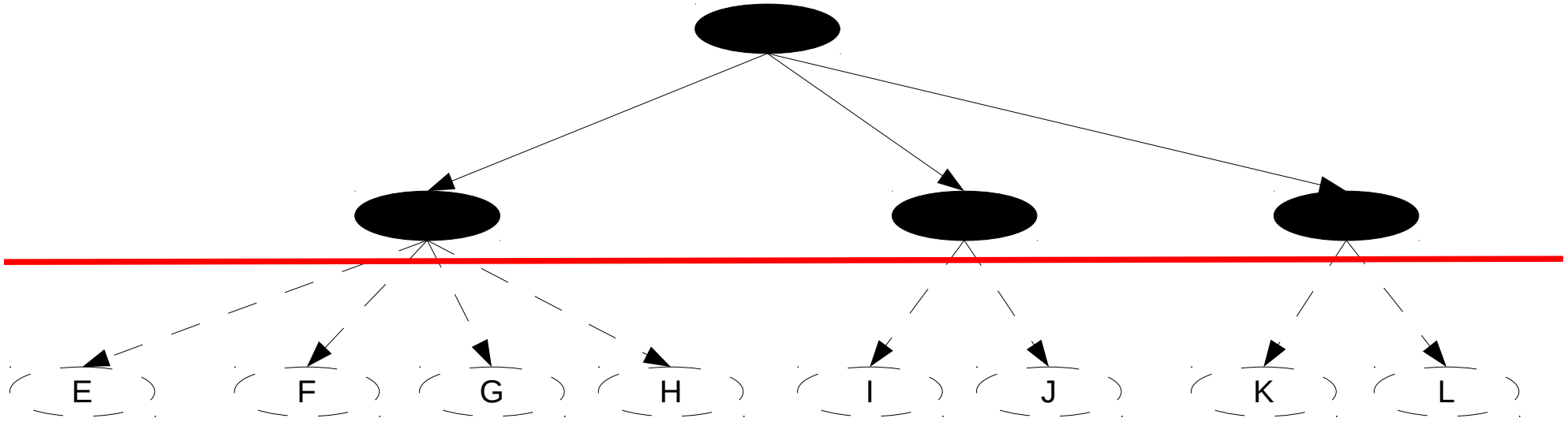
Busca em Profundidade Iterativa

- O nó mais profundo não expandido dentro do limite atual é expandido
  - Utiliza uma LIFO como *franja*



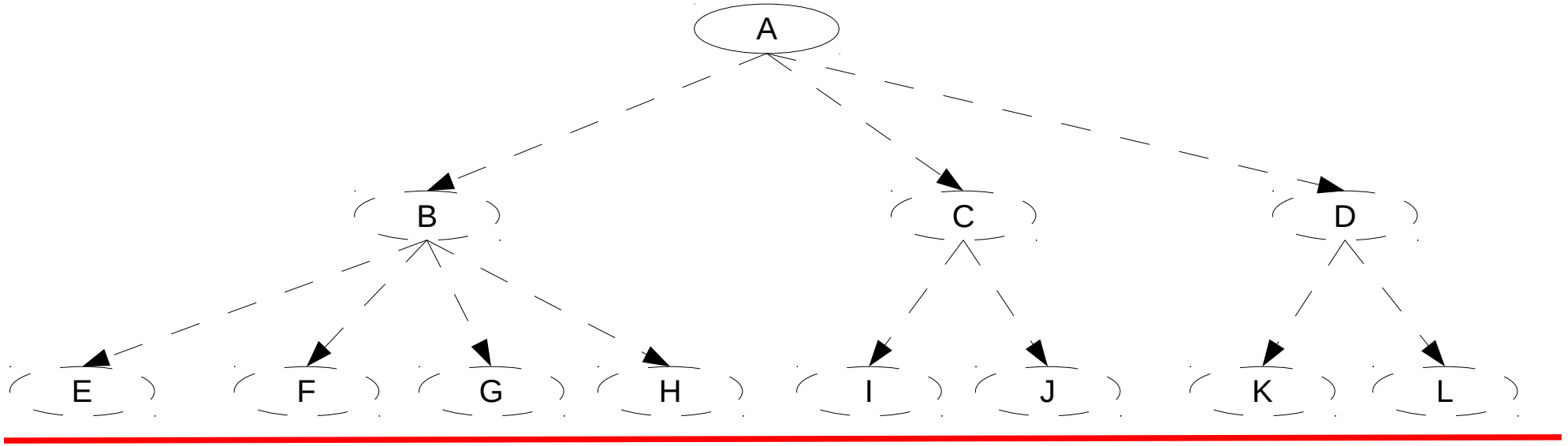
Busca em Profundidade Iterativa

- O nó mais profundo não expandido dentro do limite atual é expandido
  - Utiliza uma LIFO como *franja*



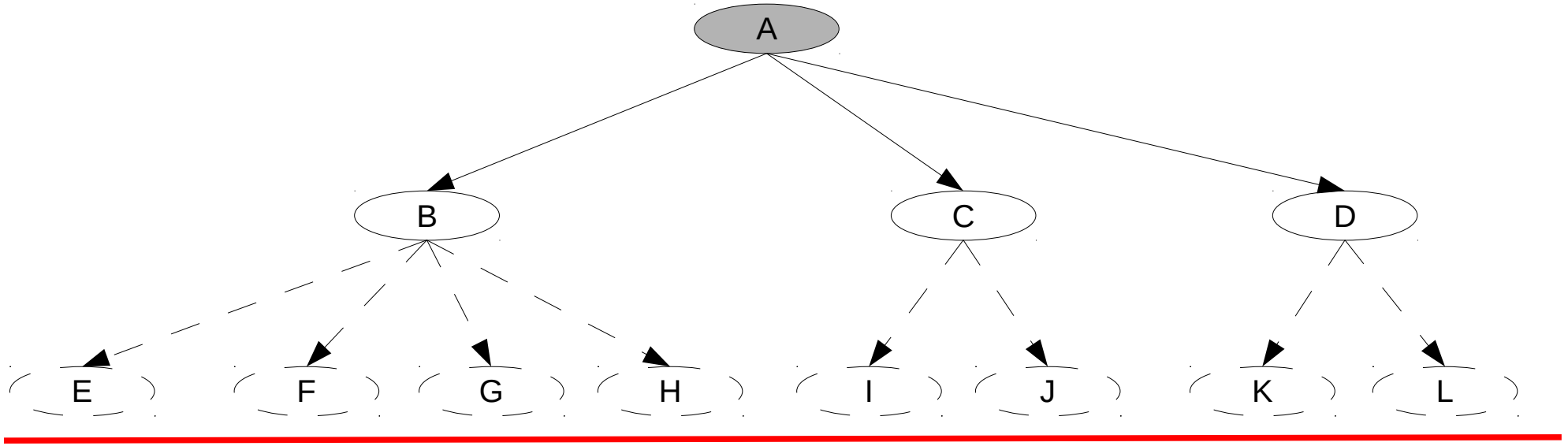
Busca em Profundidade Iterativa

- O nó mais profundo não expandido dentro do limite atual é expandido
  - Utiliza uma LIFO como *franja*



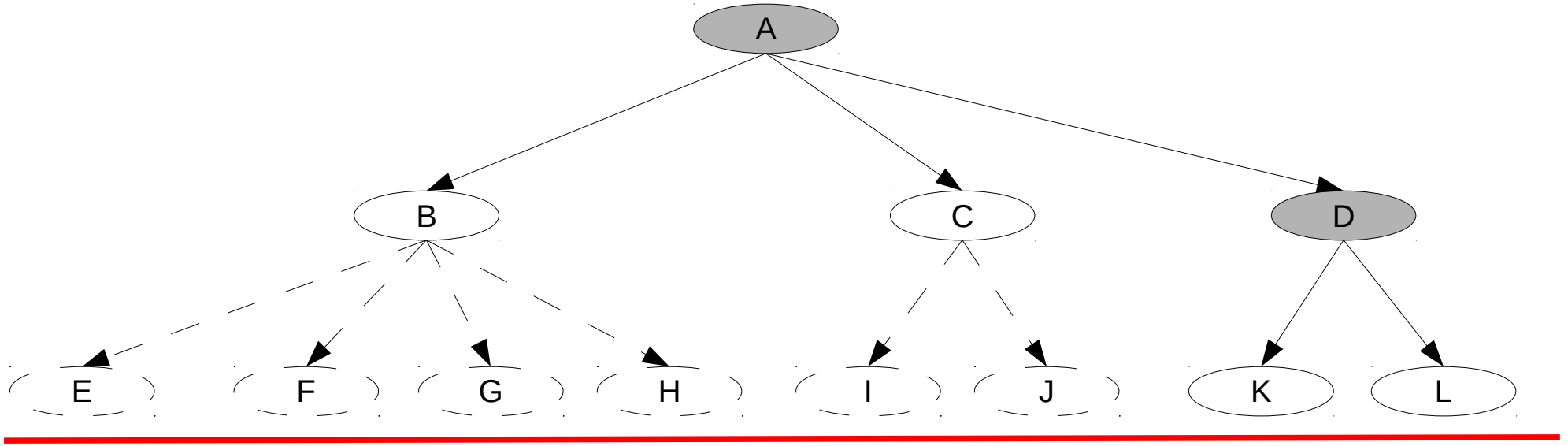
Busca em Profundidade Iterativa

- O nó mais profundo não expandido dentro do limite atual é expandido
  - Utiliza uma LIFO como *franja*



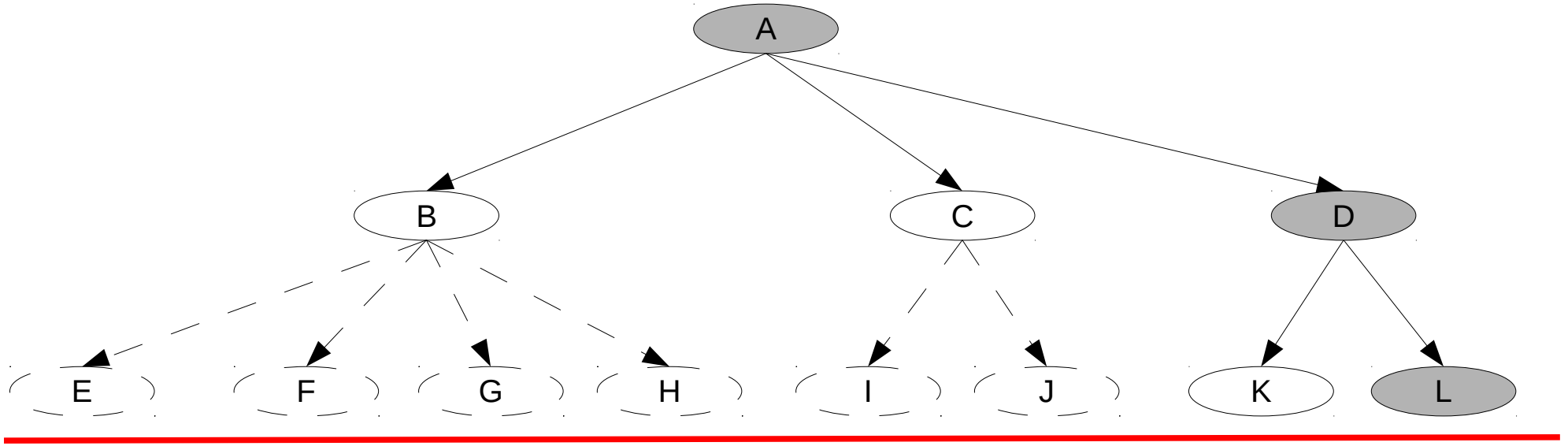
Busca em Profundidade Iterativa

- O nó mais profundo não expandido dentro do limite atual é expandido
  - Utiliza uma LIFO como *franja*



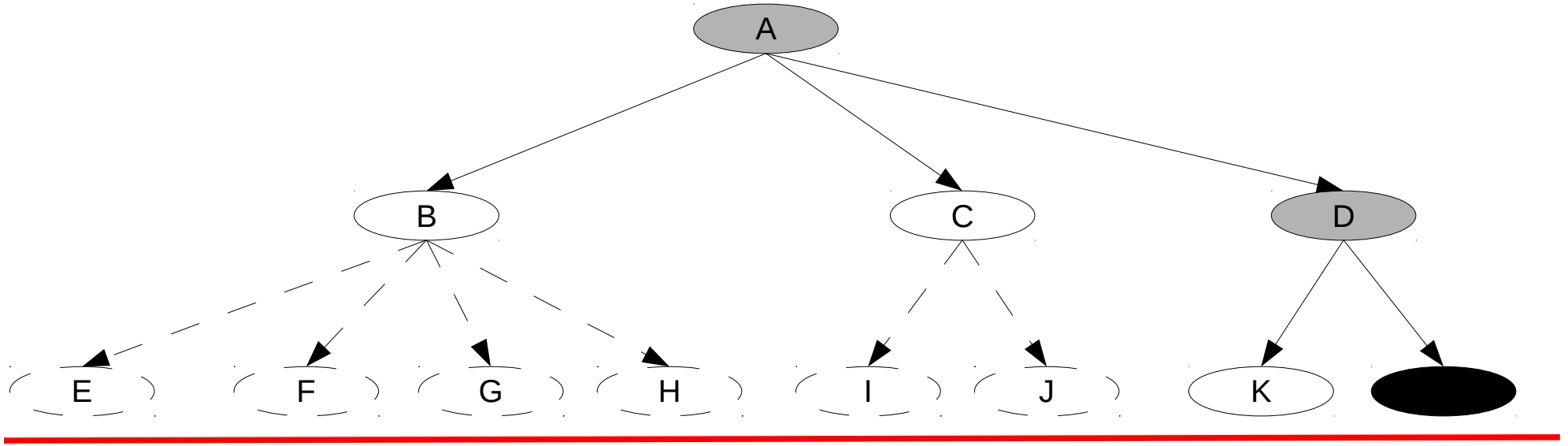
Busca em Profundidade Iterativa

- O nó mais profundo não expandido dentro do limite atual é expandido
  - Utiliza uma LIFO como *franja*



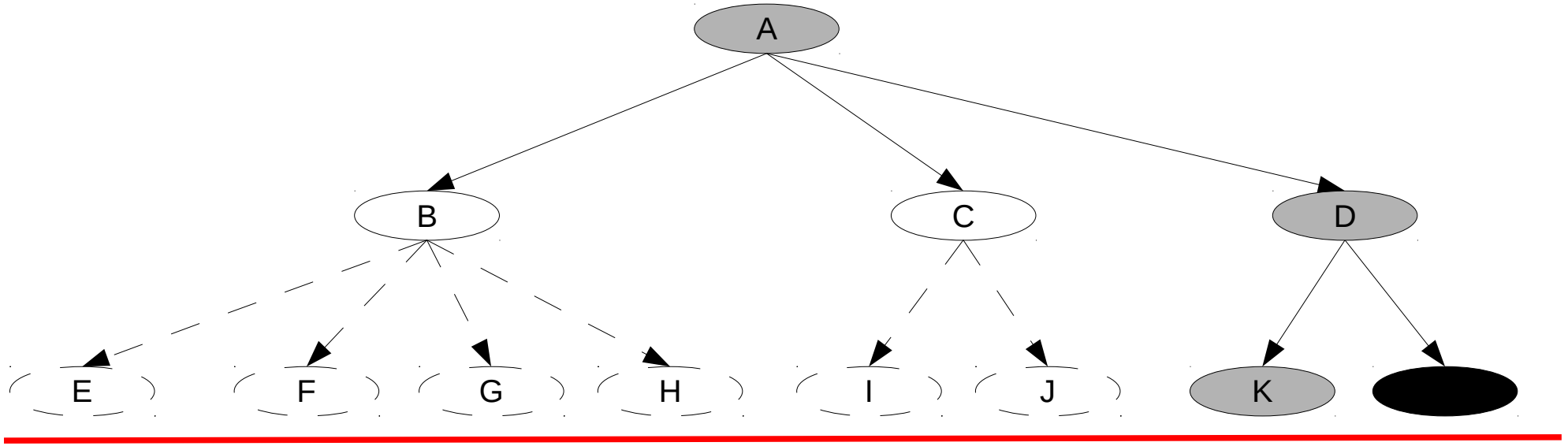
Busca em Profundidade Iterativa

- O nó mais profundo não expandido dentro do limite atual é expandido
  - Utiliza uma LIFO como *franja*



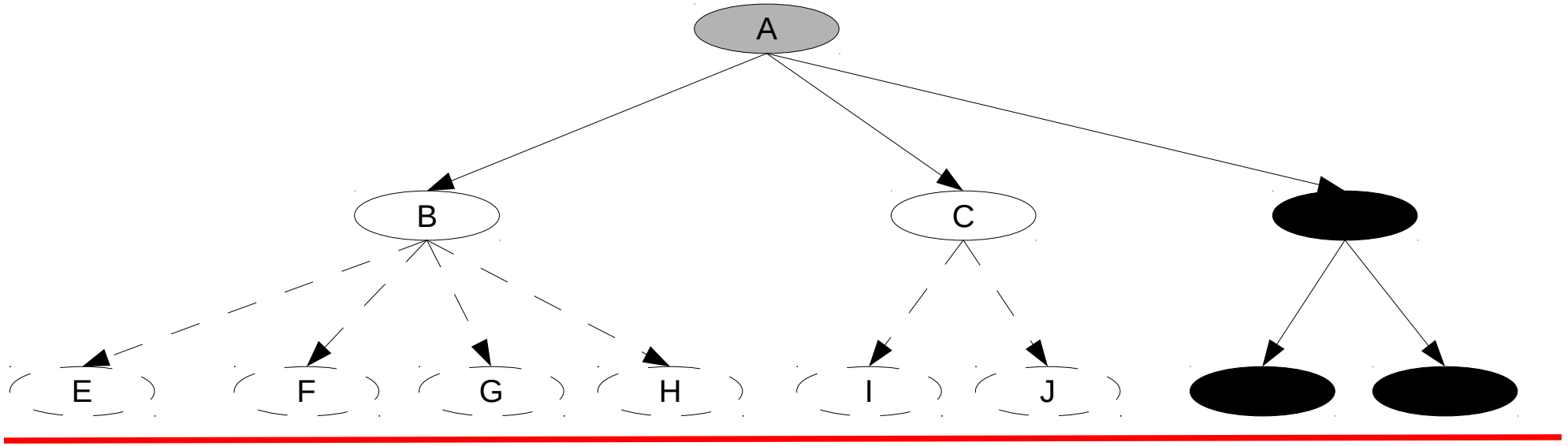
Busca em Profundidade Iterativa

- O nó mais profundo não expandido dentro do limite atual é expandido
  - Utiliza uma LIFO como *franja*



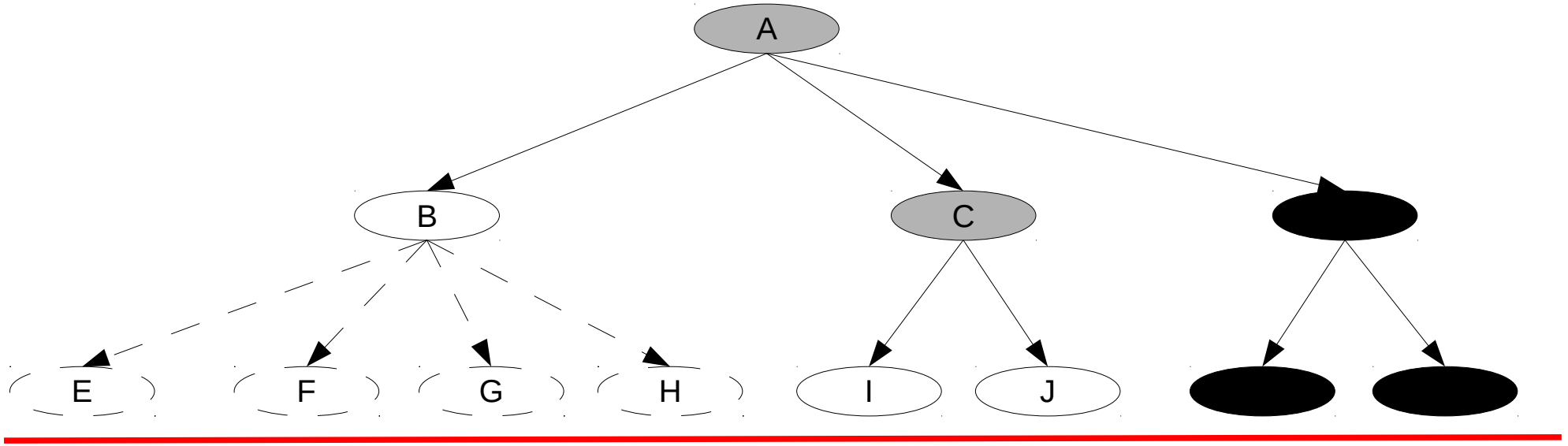
Busca em Profundidade Iterativa

- O nó mais profundo não expandido dentro do limite atual é expandido
  - Utiliza uma LIFO como *franja*



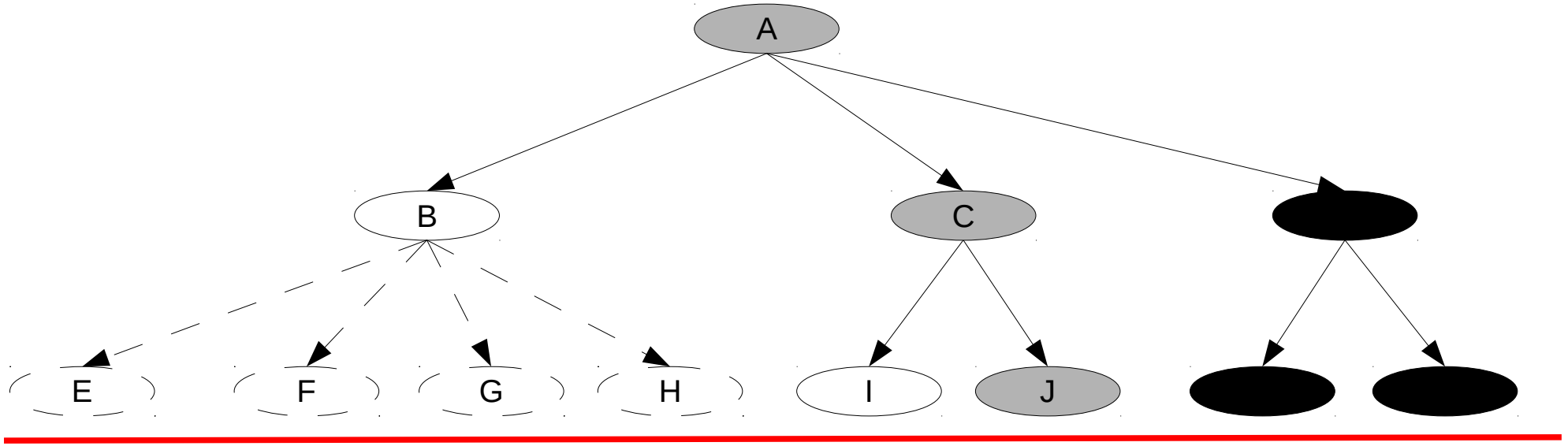
Busca em Profundidade Iterativa

- O nó mais profundo não expandido dentro do limite atual é expandido
  - Utiliza uma LIFO como *franja*



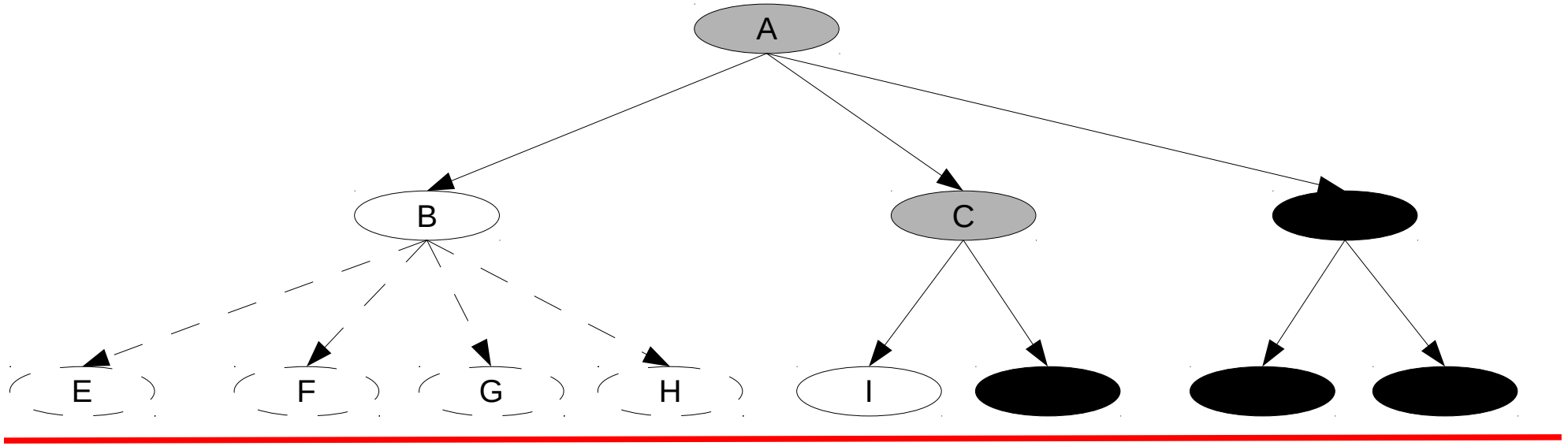
Busca em Profundidade Iterativa

- O nó mais profundo não expandido dentro do limite atual é expandido
  - Utiliza uma LIFO como *franja*



Busca em Profundidade Iterativa

- O nó mais profundo não expandido dentro do limite atual é expandido
  - Utiliza uma LIFO como *franja*



Análise

- Completude
  - Garante (Se *b* é finito)
- Optimalidade
  - Sim, se o custo for igual em todos os passos
- Complexidade no Tempo
  - Número de nós Gerados (Pior caso)

$$(d)b + (d-1)b^2 + ... + (1)b^d = O(b^d)$$

- Complexidade no espaço linear

$$O(bd)$$

Busca Bidirecional

- Se a Busca em Largura for executada simultaneamente a partir da raiz e a partir da solução do problema, a complexidade do problema é reduzida (tempo e espaço)

$$b^{d/2} + b^{d/2} << b^{d}$$

- O problema da Busca Bidirecional é que deve-se voltar a partir da solução
  - Deve haver uma FUNÇÃO-PREDESCESSOR = FUNÇÃO-SUCESSOR
  - O problema é muito maior se o objetivo for uma propriedade, como “Xeque-Mate”

Resolução de Problemas

Evitando Estados Repetidos

Evitando Estados Repetidos

- Alguns problemas podem levar o espaço de estados a ser um grafo, em vez de uma árvore
  - Alguns são até mesmo infinitos
    - Problemas de rota
    - Problemas de blocos deslizantes
- Nestes casos, remover alguns estados repetidos podem reduzir a árvore de busca a um tamanho finito

Evitando Estados Repetidos

- O algoritmo deve se lembrar dos estados já visitados
  - *Algoritmos que esquecem-se de sua história estão condenados a repeti-la*
  - Uma *lista fechada* pode ser mantida para armazenar todos os nós que foram expandidos
    - A franja dos nós não expandidos é, as vezes, chamada de *lista aberta*
    - Se o nó atual estiver na *lista fechada* ele é descartado em vez de expandido

Busca em Grafos

```
função BUSCA-EM-GRAFO(problema, franja) retorna uma solução, ou falha
  entradas: problema, a formulação do problema
            franja, a franja vazia

  fechado ← um conjunto vazio
  franja ← INSERT(CRIA-NO(ESTADO-INICIAL[problema]),franja)
  laço faça
    se VAZIA(franja) então retorne falha
    nó ← REMOVE-PRIMEIRO(franja)
    se TESTE-OBJETIVO[problema](ESTADO[nó]) então retorne SOLUÇÃO(nó)
    se ESTADO[nó] não está em fechado então
      adiciona ESTADO[nó] a fechado
      franja ← INSERE-TODOS(EXPANDA(nó, problema),franja)
```

Observações

- Garante optimalidade para utilização com Busca em C*usto Uniforme* e *largura* se o custo do passo for constante
- Busca em *Profundidade* e *Profundidade Iterativa* deixam de ser lineares no espaço
  - A *lista fechada* armazena todos os nós visitados
  - Algumas buscas podem se tornar impossíveis devido a limitações de memória

Resolução de Problemas

Busca com Informação Parcial

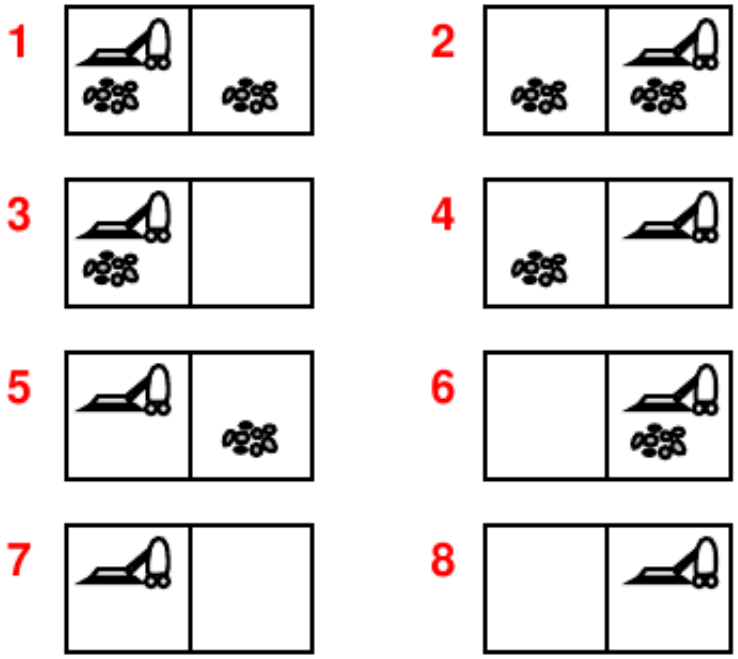
Busca com Informação Parcial

- Problemas sem sensores
  - Se o agente não tem sensores, então ele pode estar em vários estados iniciais possíveis
  - Cada ação pode levar a vários estados possíveis
- Problemas de Contingência
  - Se o ambiente é parcialmente observável, ou se as ações forem incertas, então a percepção do agente fornece informações novas depois de cada ação
  - Não será abordado neste curso
- Problemas de Exploração
  - Quando os estados e resultados das ações no ambiente são desconhecidas, o agente tem que descobri-las
  - Será abordado no tópico Busca Heurística

Problemas sem Sensores

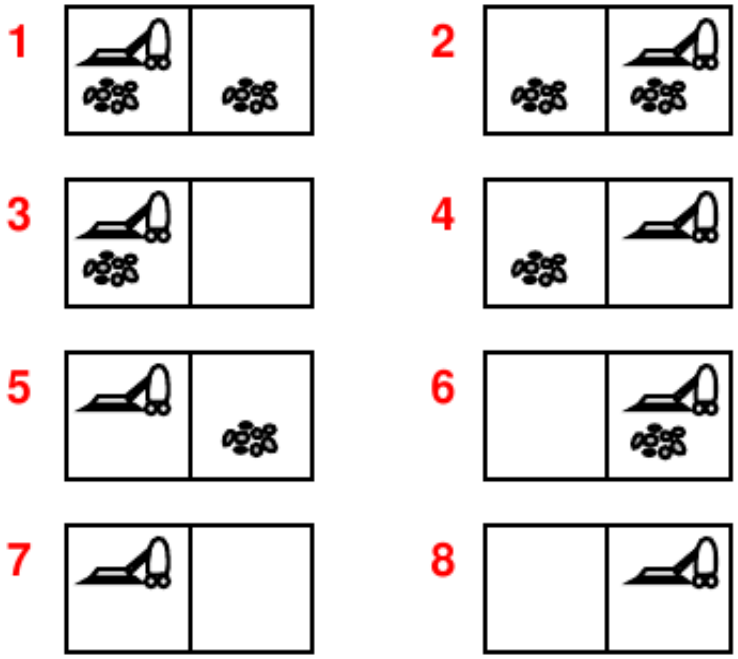
- Estado inicial é um conjunto de estados possíveis
- Cada ação leva a um conjunto de estados de crença
  - O agente pode tentar *forçar* o mundo a estar no estado objetivo
  - Em geral, se o problema sem Sensores tem *S* estados físicos, ele terá 2^S estados de crença

Problema sem Sensores



- Sem sensores o trabalho parece sem esperança

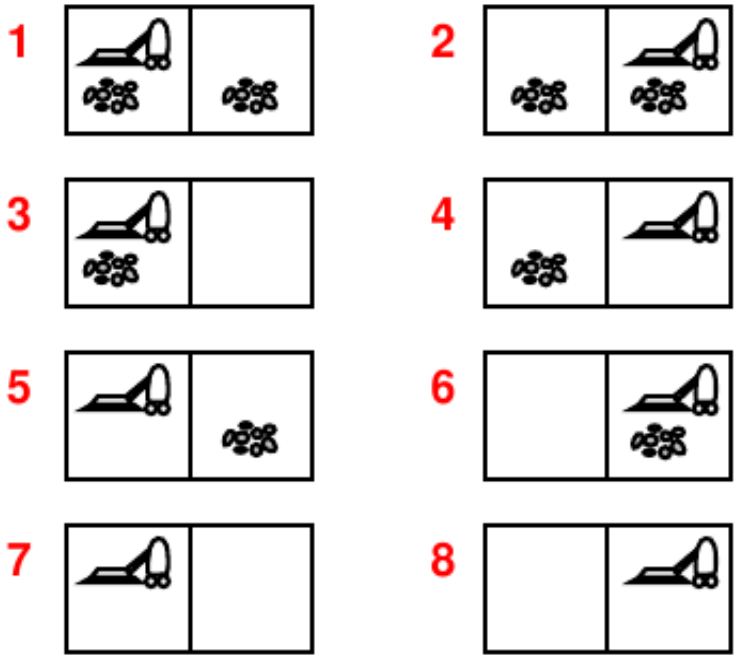
Problema sem Sensores



- Estado de crença inicial

  {1,2,3,4,5,6,7,8}

Problema sem Sensores

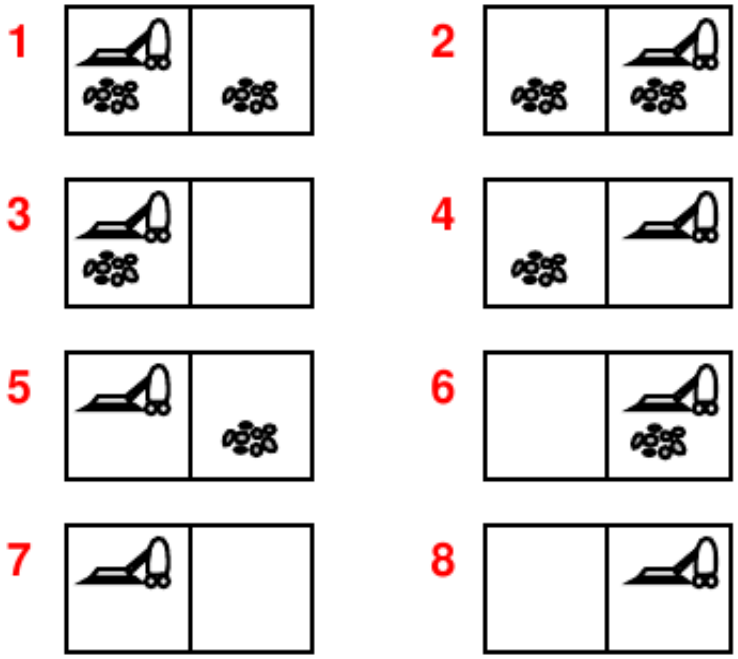


- Estado de crença inicial

  {1,2,3,4,5,6,7,8}
- Ação *Right*

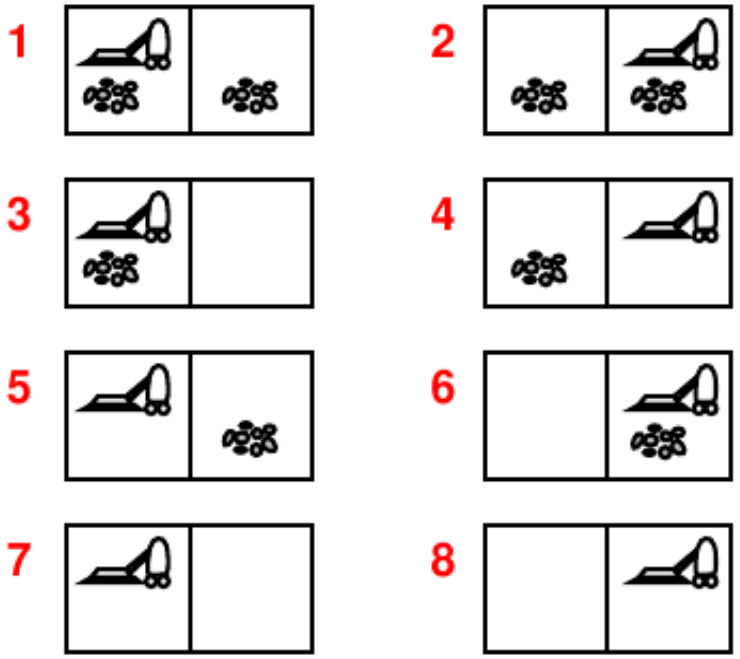
  {2,4,6,8}

Problema sem Sensores



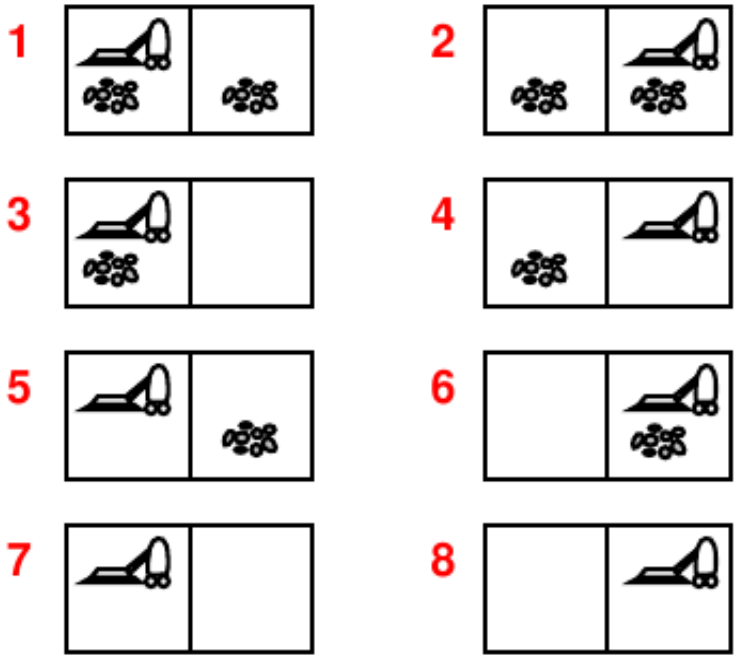
- Estado de crença inicial
  {1,2,3,4,5,6,7,8}
- Ação *Right*
  {2,4,6,8}
- Ação *Suck*
  {4,8}

Problema sem Sensores



- Estado de crença inicial
  {1,2,3,4,5,6,7,8}
- Ação *Right*
  {2,4,6,8}
- Ação *Suck*
  {4,8}
- Ação *Left*
  {3,7}

Problema sem Sensores



- Estado de crença inicial
  {1,2,3,4,5,6,7,8}
- Ação *Right*
  {2,4,6,8}
- Ação *Suck*
  {4,8}
- Ação *Left*
  {3,7}
- Ação *Suck*
  {7}

Problema sem Sensores

- Só foi possível resolver porque o ambiente era determinístico
  - Se sujeira aparecesse de maneira aleatória, não seria possível resolver o problema

Referências

- [JOHNSON e STORY, 1879] Johnson, Wm. Woolsey; e Story, William E. *Notes on the "15" Puzzle*. In American Journal of Mathematics. The Johns Hopkins University Press, 1879.