



SCC-5774 - Capítulo 1

Métodos de Busca

João Luís Garcia Rosa¹

¹Departamento de Ciências de Computação
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo - São Carlos
<http://www.icmc.usp.br/~joaoluis>

2020

Sumário

1 Formulação do Problema

- Solução
- Tipos
- Sistema de Produção

2 Controle

- Backtracking
- Busca em Grafos

3 Busca

- Busca Cega
- Busca Heurística

Sumário

1 Formulação do Problema

- Solução
- Tipos
- Sistema de Produção

2 Controle

- Backtracking
- Busca em Grafos

3 Busca

- Busca Cega
- Busca Heurística

Férias na Romênia

- Férias na Romênia; atualmente em Arad.
- Os vôos partem amanhã de Bucharest.
- Formular meta:
 - estar em Bucharest
- Formular problema:
 - estados: várias cidades
 - ações: dirigir entre cidades
- Achar solução:
 - seqüência de cidades, e.g., Arad, Sibiu, Fagaras, Bucharest

Romênia

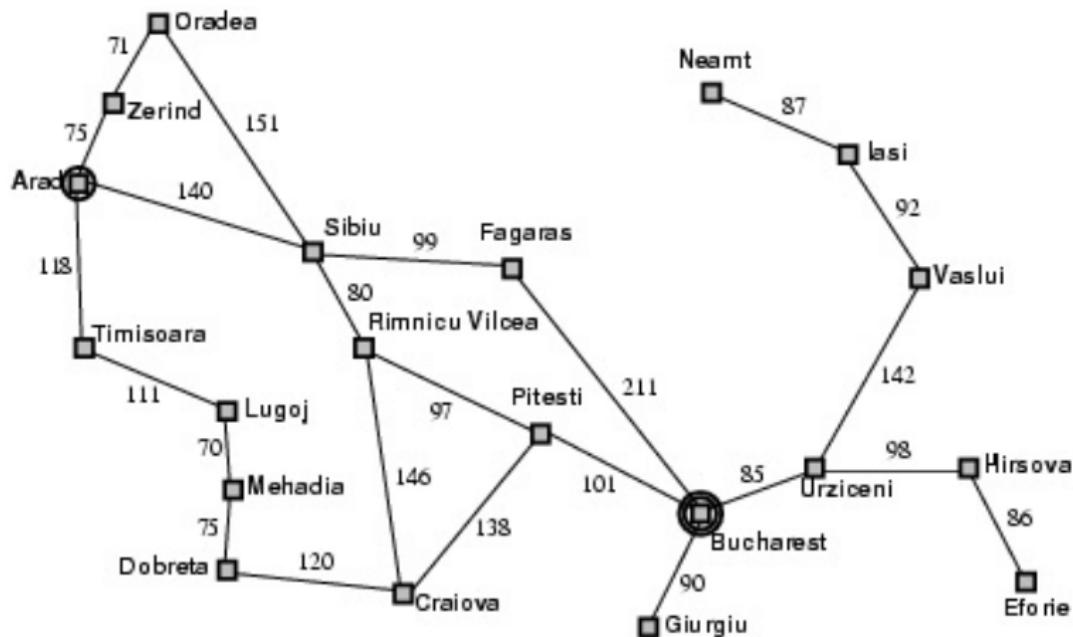


Figure 1: Fonte: Russell e Norvig [3]

Sumário

1 Formulação do Problema

- Solução
- Tipos
- Sistema de Produção

2 Controle

- Backtracking
- Busca em Grafos

3 Busca

- Busca Cega
- Busca Heurística

Tipos de Problemas

- Determinístico, totalmente observável ⇒ problema de único estado
 - O agente sabe exatamente em qual estado estará; solução é uma seqüência
- Não observável ⇒ problema sem sensores
 - O agente pode não ter idéia de onde está; solução é uma seqüência
- Não-determinístico e/ou parcialmente observável ⇒ problema de contingência
 - Os perceptores provêem informação *nova* sobre o estado corrente
- Espaço de estados desconhecido ⇒ problema de exploração

Aspirador

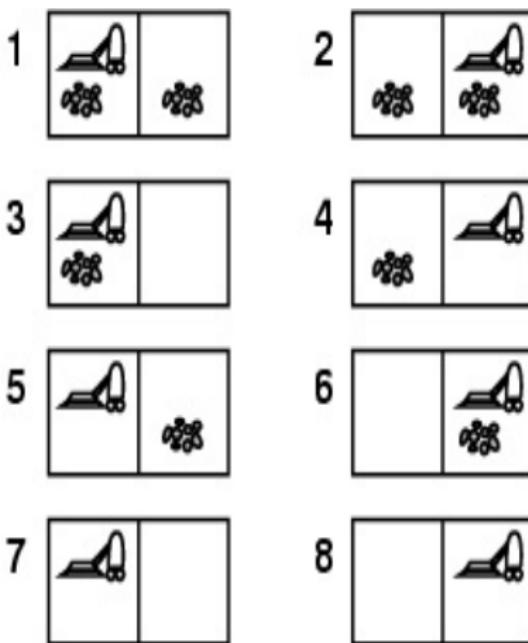


Figure 2: Fonte: Russell e Norvig [3]

Aspirador

- Estado único, começa no #5.
Solução? [Direita, Aspira]
- Sem sensores, começa em $\{1,2,3,4,5,6,7,8\}$ e.g., *Direita* vai para $\{2,4,6,8\}$
Solução? [Direita, Aspira, Esquerda, Aspira]

Contingência

- Não-determinístico: Aspirar pode sujar um carpete limpo
- Parcialmente observável: local, sujeira no local corrente.
- Perceptor: [E, Limpa], i.e., começa no #5 ou #7
Solução? [Direita, se sujeira então Aspira]

Formulação de um Problema de Estado Único

- Um problema é definido por quatro itens:
 - ① estado inicial e.g., "em Arad"
 - ② ações ou função sucessor $S(x) = \text{conjunto de pares ação-estado}$
 - e.g., $S(Arad) = \{<Arad \rightarrow Zerind, Zerind>, \dots\}$
 - ③ teste da meta, pode ser
 - explícito, e.g., $x = \text{"em Bucharest"}$
 - implícito, e.g., $\text{Checkmate}(x)$
 - ④ custo do caminho (aditivo)
 - e.g., soma das distâncias, número de ações executadas, etc.
 - $c(x,a,y)$ é o custo do passo, assumido ser = 0
- Uma solução é uma seqüência de ações que leva do estado inicial a um estado meta

Selecionando um Espaço de Estados

- O mundo real é absurdamente complexo
→ o espaço de estados deve ser restrinido para a resolução de problemas
- Estado (restrito) = conjunto de estados reais
- Ação (restrita) = combinação complexa de ações reais
 - e.g., “Arad → Zerind” representa um conjunto complexo de possíveis rotas, desvios, paradas, etc.
- Para a garantia da realização, qualquer estado real “em Arad” deve levar a algum estado real “em Zerind”
- Solução (restrita) =
 - conjunto de caminhos reais que são soluções no mundo real
- Cada ação restrita deve ser “mais fácil” que o problema original

Grafo do Espaço de Estados do Aspirador

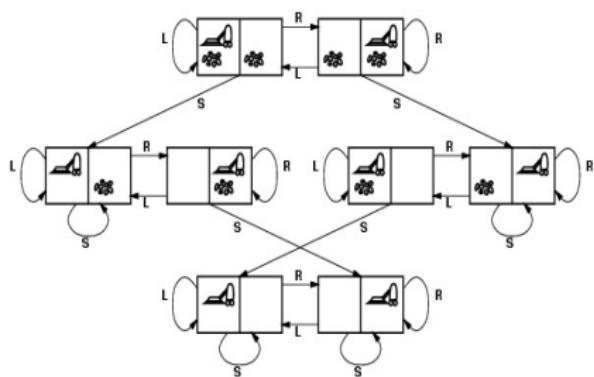


Figure 3: Fonte: Russell e Norvig [3]

- estados? posição da sujeira e do robô: inteiros
- ações? *Esquerda, Direita, Aspirar*
- teste da meta? sem sujeira em todas as posições
- custo do caminho? 1 por ação

Exemplo: Quebra-cabeça de 8 Peças

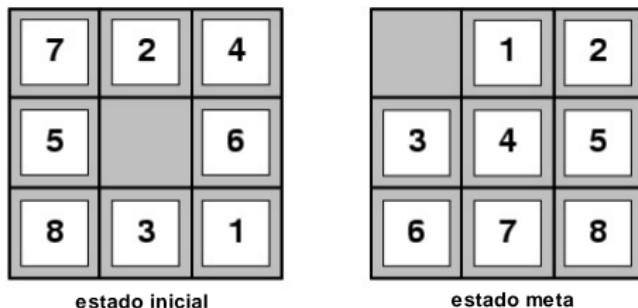


Figure 4: Fonte: Russell e Norvig [3]

- estados? posições das peças
- ações? mover branco para esquerda, direita, cima, baixo
- teste da meta? = estado meta (dado)
- custo do caminho? 1 por movimento

Exemplo: Problema das n Rainhas

- Colocar n rainhas em um tabuleiro $n \times n$ com uma única rainha em cada linha, coluna e diagonal.
- Para $n = 8$:

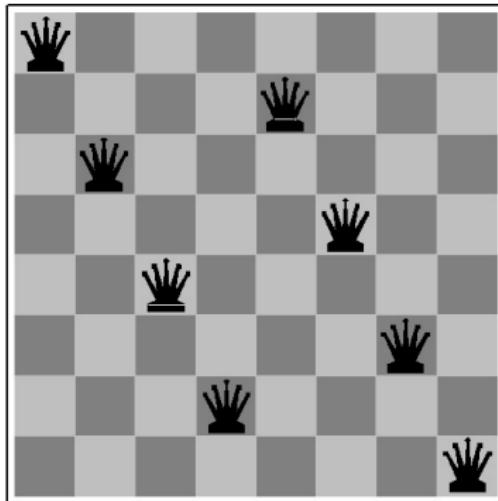
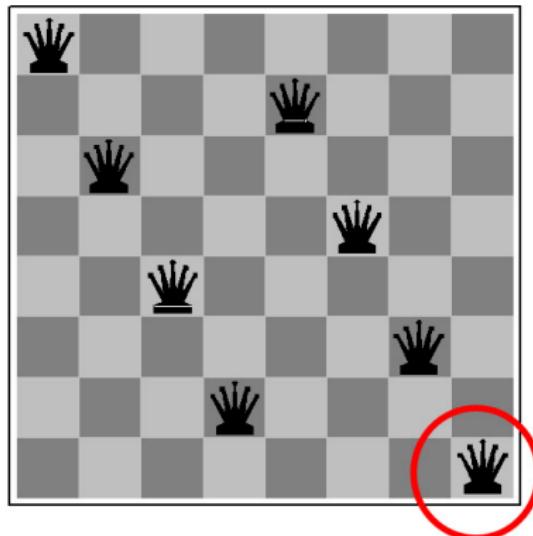


Figure 5: Fonte: Russell e Norvig [3]

Exemplo: Problema das n Rainhas

- Colocar n rainhas em um tabuleiro $n \times n$ com uma única rainha em cada linha, coluna e diagonal.
- Para $n = 8$:



Sumário

1 Formulação do Problema

- Solução
- Tipos
- Sistema de Produção

2 Controle

- Backtracking
- Busca em Grafos

3 Busca

- Busca Cega
- Busca Heurística

Definição

Um Sistema de Produção (SP) é composto por três elementos principais:

- Base de dados global
- Regras de produção
- Sistema de controle

Exemplo: Quebra-cabeça de 8 Peças

2	8	3
1	6	4
7		5

Inicial



1	2	3
8		4
7	6	5

Meta

Produção

Procedimento PRODUÇÃO

- ① DADOS ← base de dados inicial
- ② até DADOS satisfazer a condição de terminação, faça:
 - ① selecione alguma regra, R, no conjunto de regras que possa ser aplicada a DADOS
 - ② DADOS ← resultado da aplicação de R a DADOS

Sistema de Controle

- *Controle*: Passo 2 dentro do loop: maior problema
- Procedimentos de busca:
 - não-informados (busca cega)
 - informados (busca heurística)

Regime de Controle

- *Regime de Controle*: A forma como é conduzido um processo de busca
- Dois tipos principais:
 - irrevogável
 - tentativo

Estratégias de Controle

- Dentro do regime de controle existem as estratégias de controle.
- Dois tipos de estratégia de controle dentro do regime de controle tentativo:
 - *Backtracking*
 - Busca em grafos

Sumário

1 Formulação do Problema

- Solução
- Tipos
- Sistema de Produção

2 Controle

- Backtracking
- Busca em Grafos

3 Busca

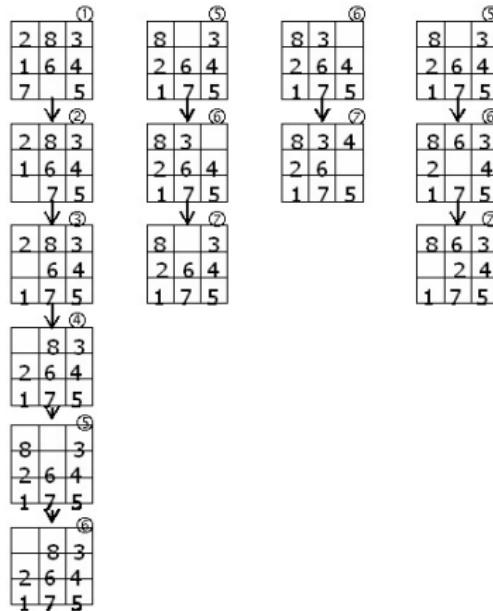
- Busca Cega
- Busca Heurística

Backtracking

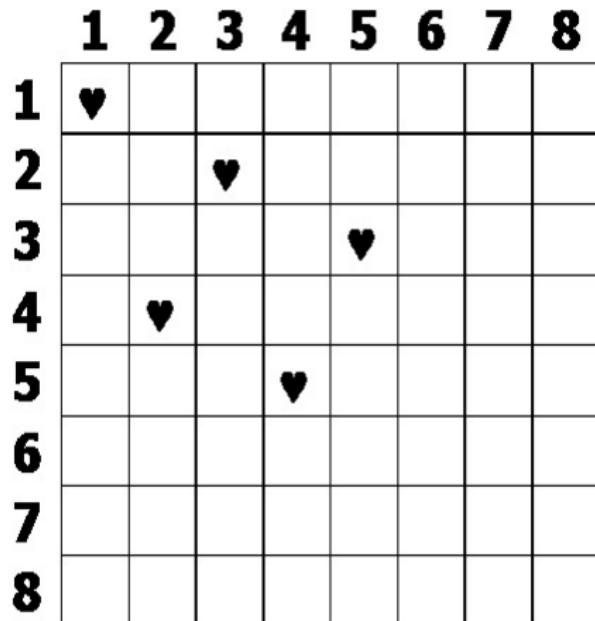
Procedimento Recursivo BACKTRACK (DADOS) [1]

- ① se TERMO(DADOS), retorne []
- ② se DEADEND(DADOS), retorne *FALHA*
- ③ REGRAS \leftarrow REGRASAPL(DADOS)
- ④ **LOOP:** se NULL(REGRAS), retorne *FALHA*
- ⑤ R \leftarrow PRIMEIRA(REGRAS)
- ⑥ REGRAS \leftarrow CAUDA(REGRAS)
- ⑦ RDADOS \leftarrow R(DADOS)
- ⑧ CAMINHO \leftarrow BACKTRACK(RDADOS)
- ⑨ se CAMINHO = *FALHA*, vá para **LOOP**
- ⑩ retorne CONC(R,CAMINHO)

Exemplo: Quebra-cabeças de 8 Peças



Exemplo: Problema das 8 Rainhas



Exemplo: Problema das 8 Rainhas

Uma solução depois de 91 backtrackings:

	1	2	3	4	5	6	7	8
1	♥							
2					♥			
3							♥	
4						♥		
5			♥					
6							♥	
7		♥						
8				♥				

Sumário

1 Formulação do Problema

- Solução
- Tipos
- Sistema de Produção

2 Controle

- Backtracking
- **Busca em Grafos**

3 Busca

- Busca Cega
- Busca Heurística

Busca em Grafos

- No backtracking: o sistema de controle armazena somente o caminho correntemente sendo estendido.
- Um procedimento mais flexível envolve o armazenamento explícito de todos os caminhos de tal forma que qualquer um deles pode ser candidato a futura extensão: BUSCA EM GRAFOS.

Busca em Grafos

Notação de grafos:

- **Grafo:** Um grafo **G** consiste num conjunto de nós (ou vértices) N e num conjunto de arestas (ou arcos) A . Cada aresta num grafo é especificada por um par de nós (u, v) .
- **Arestas direcionadas:** Certos pares de nós são conectados por arestas direcionadas de um membro do par ao outro (grafo direcionado - dígrafo).
- **Sucessor/pai:** Se uma aresta é direcionada do nó n_i para o nó n_j , então o nó n_j é o sucessor do nó n_i , e o nó n_i é o pai do nó n_j .
- **Árvore:** caso especial de um grafo no qual cada nó tem, no máximo, um pai.
- **Raiz:** Um nó na árvore que não tem pai é chamado de nó raiz.
- **Folha:** Um nó na árvore que não tem sucessores é chamado de nó folha.

Busca em Grafos

Notação de grafos:

- *Profundidade*: Diz-se que o nó raiz é de profundidade zero. A profundidade de qualquer outro nó na árvore é, por definição, a profundidade de seus pais mais 1.
- *Caminho*: Uma seqüência de nós (n_1, n_2, \dots, n_k) , com cada n_i um sucessor de n_{i-1} , para $i = 2, \dots, k$, é chamada de um caminho de comprimento k do nó n_1 para o nó n_k . Se um caminho existe entre o nó n_i para o nó n_j , então o nó n_j é acessível a partir do nó n_i .
- *Descendente e ancestral*: O nó n_j é então um descendente do nó n_i , e o nó n_i é um ancestral do nó n_j .

Busca em Grafos

IMPORTANTE

- Para este modelo, os *nós* são rotulados por **bases de dados** e as *arestas* são rotuladas por **regras**.
- Note que o problema de achar uma seqüência de regras para transformar uma base de dados em outra é equivalente ao problema de achar um caminho num grafo.

Busca em Grafos

Custo:

- É conveniente atribuir custos positivos às arestas, para representar o custo da aplicação da regra correspondente.
- Usa-se a notação $c(n_i, n_j)$ para denotar o custo de uma aresta direcionada do nó n_i para o nó n_j .
- O custo de um caminho entre dois nós é a soma dos custos de todas as arestas que conectam os nós no caminho.
- Em alguns problemas, deseja-se achar o caminho que tenha custo mínimo entre dois nós.

Busca em Grafos

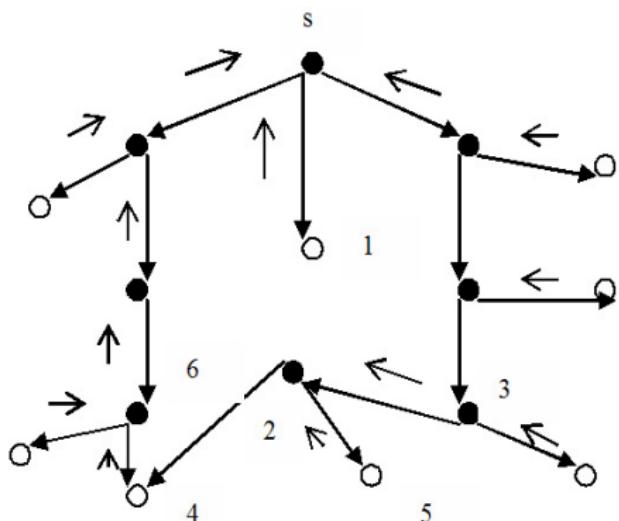
No tipo mais simples de problema, deseja-se achar um caminho (talvez tendo custo mínimo) entre um nó dado s , representando a base de dados inicial e um outro nó dado t , representando alguma outra base de dados. A situação mais usual envolve achar um caminho entre o nó s e qualquer membro do conjunto de nós $\{t_i\}$ que representa as bases de dados que satisfazem a condição de terminação. Chama-se o conjunto $\{t_i\}$ o conjunto meta e cada nó t em $\{t_i\}$ é um nó meta.

Procedimento Busca em Grafos [1]

- ➊ Crie um grafo de busca, G , consistindo somente do nó inicial, s . Ponha s numa lista chamada $ABERTOS$.
- ➋ Crie uma lista chamada $FECHADOS$ que está inicialmente vazia.
- ➌ **LOOP:** se $ABERTOS$ está vazia, saia com falha.
- ➍ Selecione o primeiro nó em $ABERTOS$, remova-o de $ABERTOS$, ponha-o em $FECHADOS$. Chame este nó de n .
- ➎ Se n é um nó meta, saia com sucesso com a solução obtida traçando um caminho entre os ponteiros de n para s em G .
- ➏ Expanda o nó n , gerando o conjunto, M , de seus sucessores que não são ancestrais de n . Instale estes membros de M como sucessores de n em G .
- ➐ Estabeleça um ponteiro para n a partir daqueles membros de M que ainda não estejam nem em $ABERTOS$ nem em $FECHADOS$. Adicione estes membros de M a $ABERTOS$.
- ➑ (Re)ordene a lista $ABERTOS$, arbitrariamente ou de acordo com mérito heurístico.
- ➒ Vá para **LOOP**.

Procedimento Busca em Grafos

Passo 7: Para cada membro de M que já esteja em *ABERTOS* ou *FECHADOS*, decida se direciona ou não seu ponteiro para n . Para cada membro de M já em *FECHADOS*, decida para cada um de seus descendentes em G se redireciona ou não o seu ponteiro.



Sumário

1 Formulação do Problema

- Solução
- Tipos
- Sistema de Produção

2 Controle

- Backtracking
- Busca em Grafos

3 Busca

- **Busca Cega**
- Busca Heurística

Procedimentos Não-informados de Busca em Grafos

As estratégias não-informadas usam apenas a informação disponível na definição do problema

- Busca em largura
- Busca em profundidade
- Busca em profundidade limitada
- Busca por aprofundamento iterativo
- Outras.

Procedimentos Não-informados de Busca em Grafos

- Estratégias de busca: Uma estratégia é definida através da ordem da expansão dos nós
- Avaliação:
 - Completeza: sempre acha a solução se ela existir?
 - Complexidade temporal: número de nós gerados / expandidos
 - Complexidade espacial: número máximo de nós na memória
 - Otimalidade: sempre acha a solução de custo mínimo?

b = fator de ramificação máximo

d = profundidade da solução de menor custo

m = profundidade máxima do espaço de estados (pode ser ∞)

Busca em Largura

- Expanda o nó de menor profundidade ainda não expandido
- Implementação:
 - ABERTOS é uma lista FIFO, i.e., novos sucessores entram no final

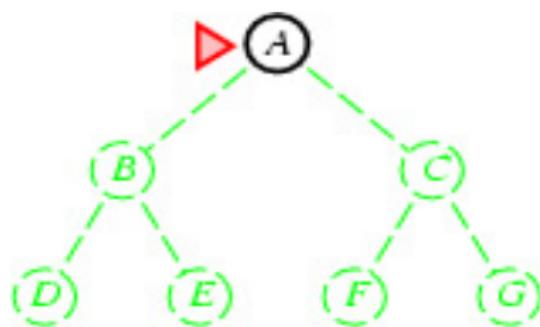
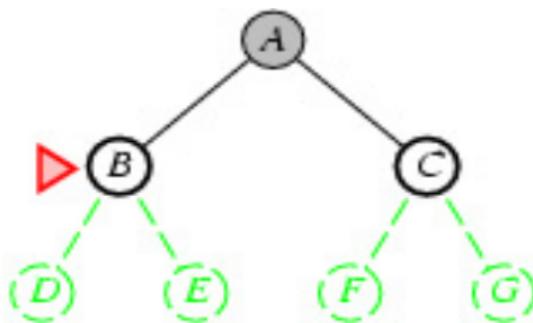


Figure 6: Fonte: Russell e Norvig [3]

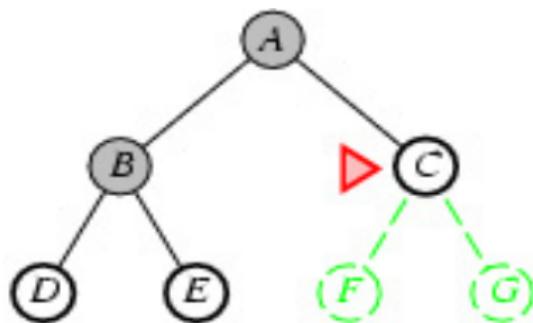
Busca em Largura

- Expanda o nó de menor profundidade ainda não expandido
- Implementação:
 - ABERTOS é uma lista FIFO, i.e., novos sucessores entram no final



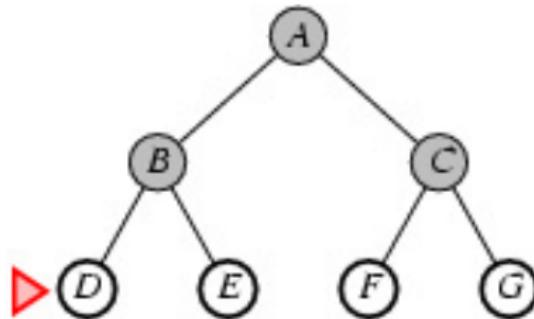
Busca em Largura

- Expanda o nó de menor profundidade ainda não expandido
- Implementação:
 - ABERTOS é uma lista FIFO, i.e., novos sucessores entram no final



Busca em Largura

- Expanda o nó de menor profundidade ainda não expandido
- Implementação:
 - ABERTOS é uma lista FIFO, i.e., novos sucessores entram no final



Propriedades da Busca em Largura

- Completa? Sim (se b é finito).
- Tempo? $1 + b + b^2 + \dots + b^d + b(b^d - 1) = \mathcal{O}(b^{d+1})$.
- Espaço? $\mathcal{O}(b^{d+1})$ (mantém todo nó na memória).
- Ótima? Sim (se custo = 1 por passo); não ótima em geral.

Espaço é o maior problema:

pode-se facilmente gerar nós a 10MB/s, tal que 24h = 860GB.

Busca em Profundidade

- Expanda o nó de maior profundidade ainda não expandido
- Implementação:
 - ABERTOS é uma lista LIFO, i.e., novos sucessores entram no início

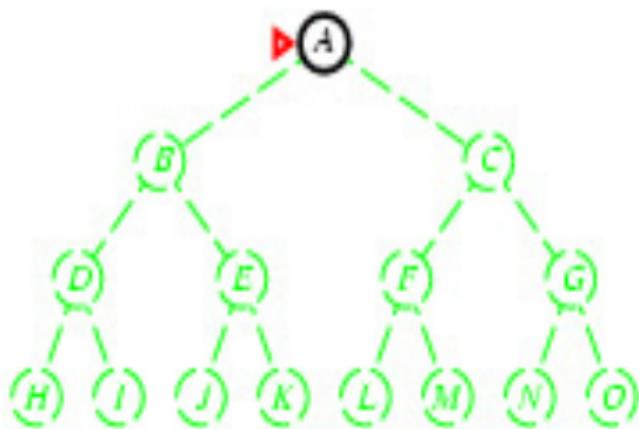
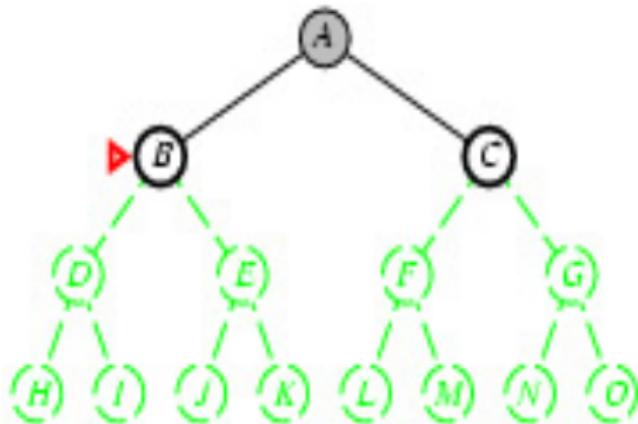


Figure 7: Fonte: Russell e Norvig [3]

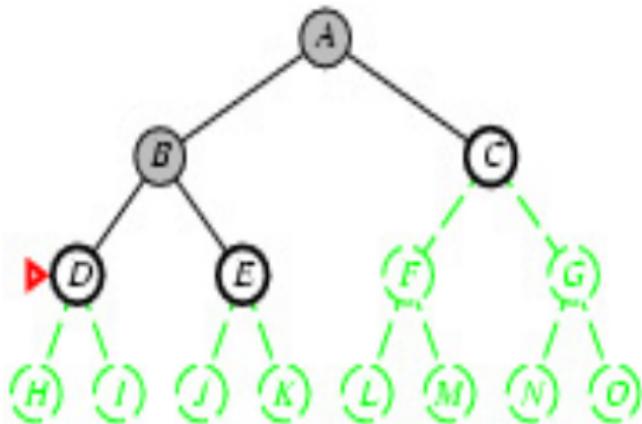
Busca em Profundidade

- Expanda o nó de maior profundidade ainda não expandido
- Implementação:
 - ABERTOS é uma lista LIFO, i.e., novos sucessores entram no início



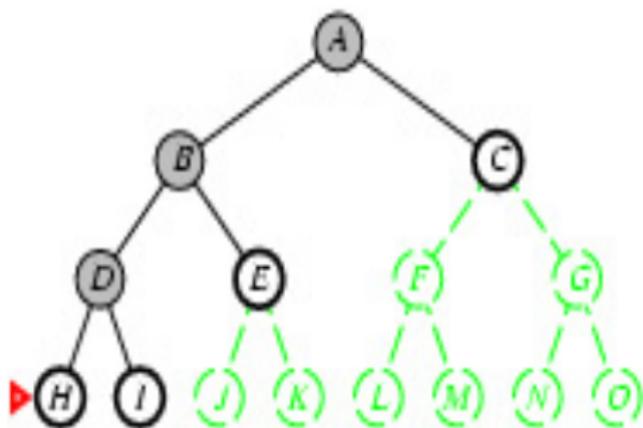
Busca em Profundidade

- Expanda o nó de maior profundidade ainda não expandido
- Implementação:
 - ABERTOS é uma lista LIFO, i.e., novos sucessores entram no início



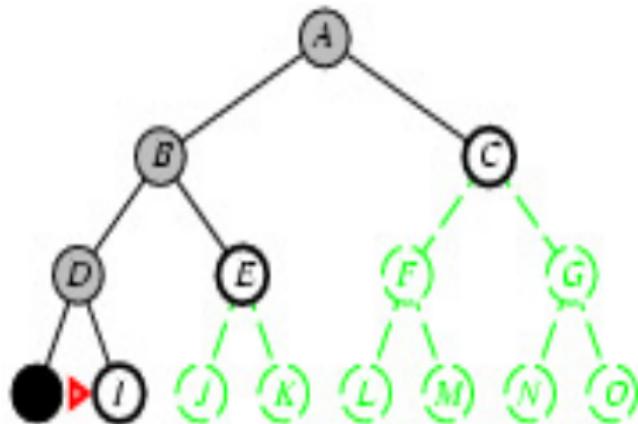
Busca em Profundidade

- Expanda o nó de maior profundidade ainda não expandido
- Implementação:
 - ABERTOS é uma lista LIFO, i.e., novos sucessores entram no início



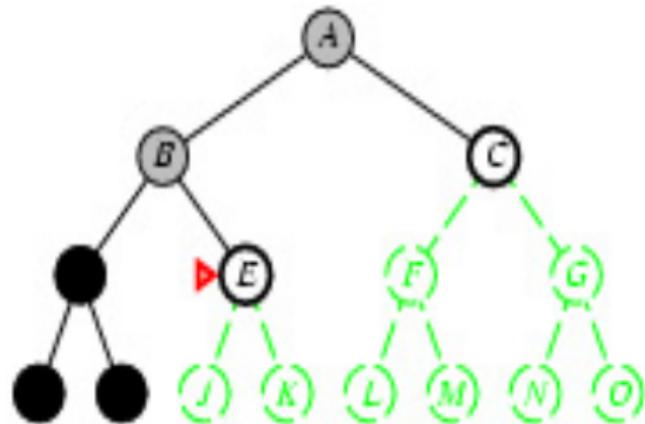
Busca em Profundidade

- Expanda o nó de maior profundidade ainda não expandido
- Implementação:
 - ABERTOS é uma lista LIFO, i.e., novos sucessores entram no início



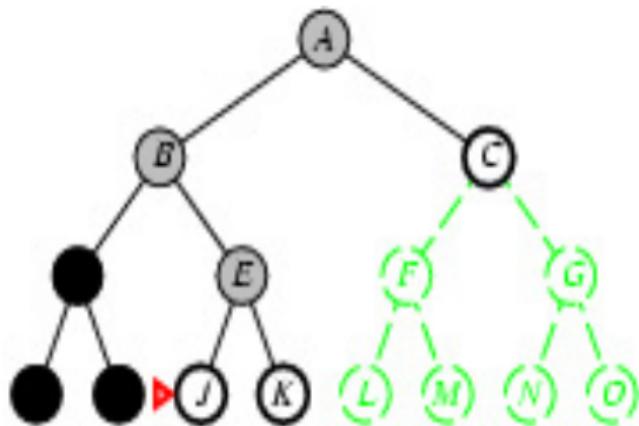
Busca em Profundidade

- Expanda o nó de maior profundidade ainda não expandido
- Implementação:
 - ABERTOS é uma lista LIFO, i.e., novos sucessores entram no início



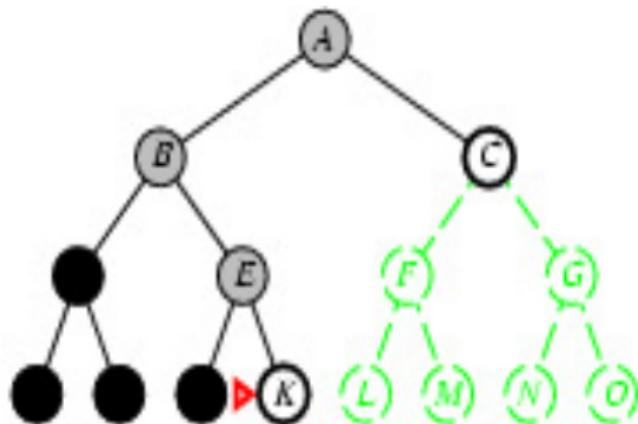
Busca em Profundidade

- Expanda o nó de maior profundidade ainda não expandido
- Implementação:
 - ABERTOS é uma lista LIFO, i.e., novos sucessores entram no início



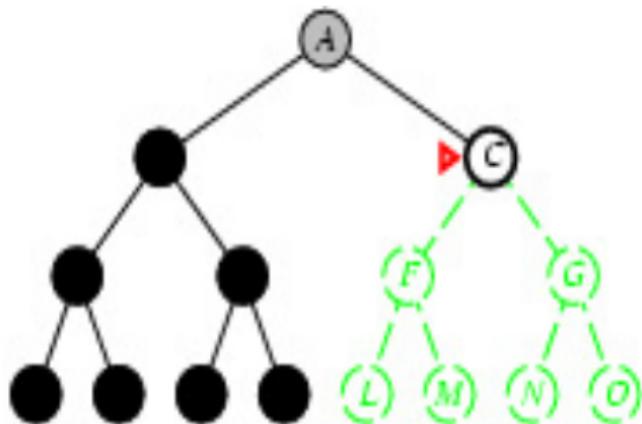
Busca em Profundidade

- Expanda o nó de maior profundidade ainda não expandido
- Implementação:
 - ABERTOS é uma lista LIFO, i.e., novos sucessores entram no início



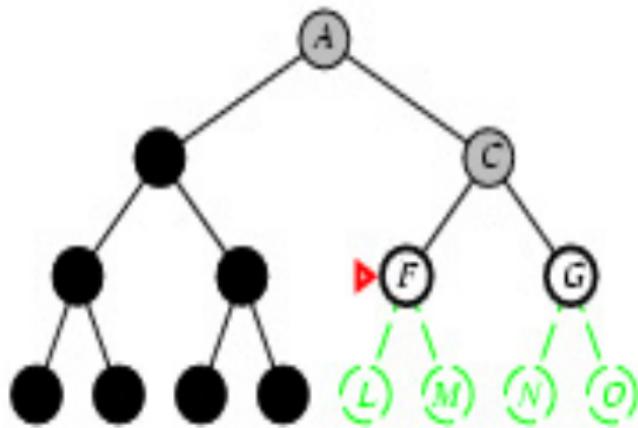
Busca em Profundidade

- Expanda o nó de maior profundidade ainda não expandido
- Implementação:
 - ABERTOS é uma lista LIFO, i.e., novos sucessores entram no início



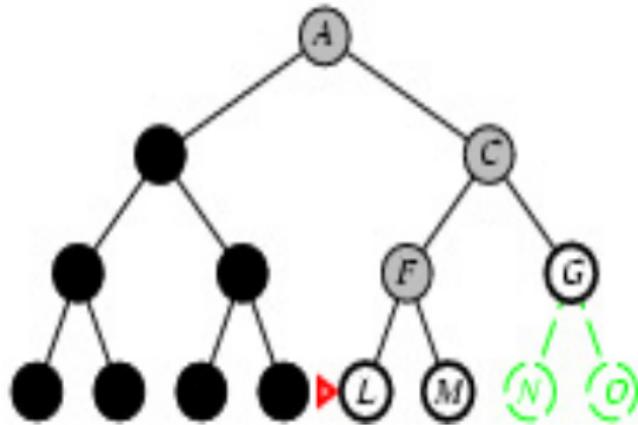
Busca em Profundidade

- Expanda o nó de maior profundidade ainda não expandido
- Implementação:
 - ABERTOS é uma lista LIFO, i.e., novos sucessores entram no início



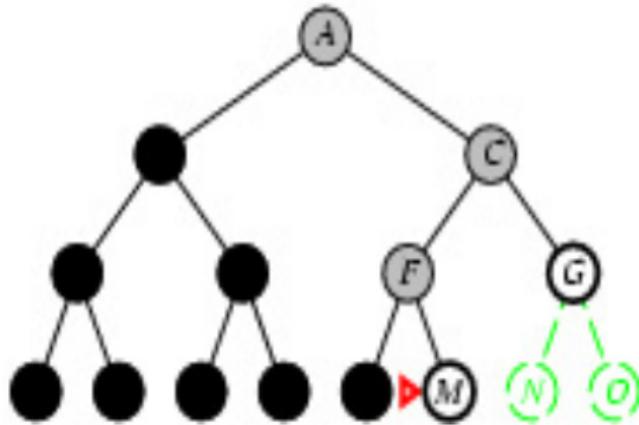
Busca em Profundidade

- Expanda o nó de maior profundidade ainda não expandido
- Implementação:
 - ABERTOS é uma lista LIFO, i.e., novos sucessores entram no início



Busca em Profundidade

- Expanda o nó de maior profundidade ainda não expandido
- Implementação:
 - ABERTOS é uma lista LIFO, i.e., novos sucessores entram no início



Propriedades da Busca em Profundidade

- Completa? Não: falha em espaços de profundidade infinita, espaços com loops. Modificar para evitar estados repetidos no caminho → completa em espaços finitos.
- Tempo? $\mathcal{O}(b^m)$ terrível se m é muito maior que d , mas se soluções são densas, pode ser mais rápido que a busca em largura.
- Espaço? $\mathcal{O}(bm)$ i. e. espaço linear!
- Ótima? Não.

Busca em Profundidade Limitada

- Busca em Profundidade Limitada = Busca em Profundidade com limite de profundidade l , i.e., nós na profundidade l não têm sucessores.
- Implementação recursiva.

Busca por Aprofundamento Iterativo: $l = 0$ 

Figure 8: Fonte: Russell e Norvig [3]

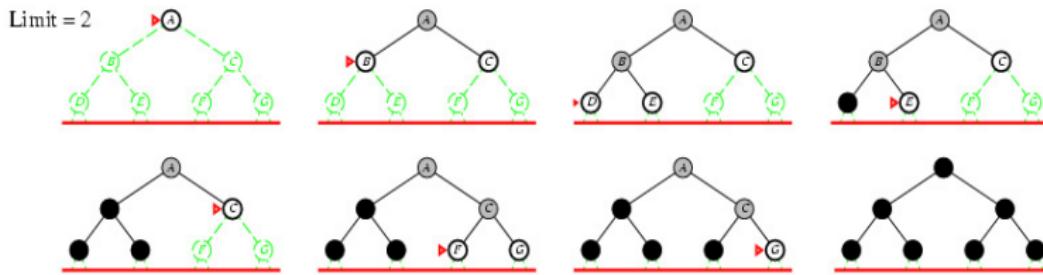
Busca por Aprofundamento Iterativo: $l = 1$

Limit = 1

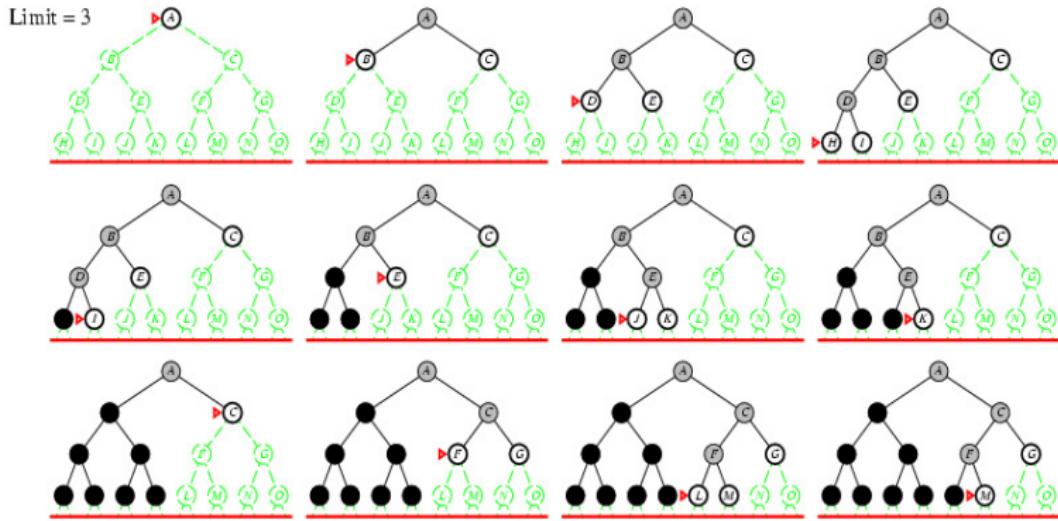


Busca por Aprofundamento Iterativo: $l = 2$

Limit = 2



Busca por Aprofundamento Iterativo: $l = 3$



Busca por Aprofundamento Iterativo

- Número de nós gerados em uma busca em profundidade limitada até a profundidade d com fator de ramificação b :
 - $N_{BPL} = b^0 + b^1 + b^2 + \dots + b^{d-2} + b^{d-1} + b^d$
- Número de nós gerados em uma busca por aprofundamento iterativo até a profundidade d com fator de ramificação b :
 - $N_{BAI} = (d+1)b^0 + db^1 + (d-1)b^2 + \dots + 3b^{d-2} + 2b^{d-1} + 1b^d$
- Para $b = 10$, $d = 5$:
 - $N_{BPL} = 1 + 10 + 100 + 1.000 + 10.000 + 100.000 = 111.111$
 - $N_{BAI} = 6 + 50 + 400 + 3.000 + 20.000 + 100.000 = 123.456$
- $Overhead = (123.456 - 111.111) / 111.111 = 11\%$

Propriedades da Busca por Aprofundamento Iterativo

- Completa? Sim.
- Tempo? $(d + 1)b^0 + db^1 + (d - 1)b^2 + \dots b^d = \mathcal{O}(b^d)$.
- Espaço? $\mathcal{O}(b^d)$ (mantém todo nó na memória).
- Ótima? Sim (se custo = 1 por passo).

Resumo dos Algoritmos

Table 1: Algoritmos de Busca Cega.

<i>critério</i>	Largura	Profundidade	BPL	BAI
<i>Completa?</i>	Sim	Não	Não	Sim
<i>Tempo?</i>	$\mathcal{O}(b^{d+1})$	$\mathcal{O}(b^m)$	$\mathcal{O}(b^l)$	$\mathcal{O}(b^d)$
<i>Espaço?</i>	$\mathcal{O}(b^{d+1})$	$\mathcal{O}(bm)$	$\mathcal{O}(bl)$	$\mathcal{O}(b^d)$
<i>Ótimo?</i>	Sim	Não	Não	Sim

- BPL = Busca em Profundidade Limitada
- BAI = Busca por Aprofundamento Iterativo

Sumário

1 Formulação do Problema

- Solução
- Tipos
- Sistema de Produção

2 Controle

- Backtracking
- Busca em Grafos

3 Busca

- Busca Cega
- **Busca Heurística**

Exemplo: Problema do Caixeiro Viajante

- A palavra *heurística* vem da palavra grega *heuriskein*, que significa “descobrir,” que é também a origem de *eureka*, a famosa exclamação de Arquimedes (“Eu achei”).
- Problema do caixeiro viajante:
 - Um vendedor tem uma lista de cidades, que ele deve visitar apenas uma vez. Existem estradas diretas ligando cada par de cidades da lista. Ache a rota que o vendedor deve seguir para a viagem mais curta possível que começa e termina em uma das cidades.

Heurística do Vizinho Mais Próximo

Heurística do vizinho mais próximo:

- ① Arbitrariamente selecione uma cidade inicial.
- ② Para selecionar a próxima cidade, olhe todas as cidades ainda não visitadas e selecione a mais próxima a cidade corrente. Vá a ela.
- ③ Repita o passo 2 até que todas as cidades tenham sido visitadas.

Este procedimento é executado em tempo proporcional a N^2 , uma melhora significativa sobre $N!$, que seria o tempo gasto caso fosse usada uma estratégia não informada (que geraria uma *explosão combinatorial*).

Busca pela Melhor Escolha

- Idéia: usar uma função de avaliação $f(n)$ para cada nó
 - estimativa da “desejabilidade”
 - expandir o nó não expandido mais desejável
- Implementação:
 - Ordenar os nós na lista ABERTOS na ordem decrescente de desejabilidade
- Casos especiais:
 - busca gulosa pela melhor escolha
 - busca A*

Romênia com Custo

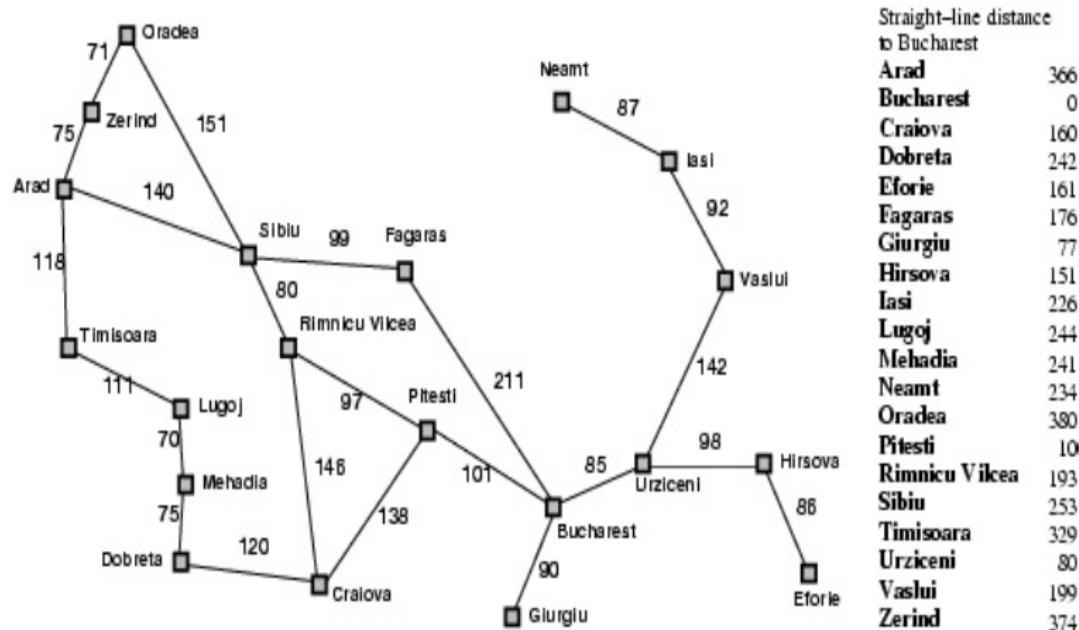


Figure 9: Fonte: Russell e Norvig [3]

Busca Gulosa pela Melhor Escolha

- Função de avaliação $f(n) = h(n)$ (**heurística**)
= estimativa de custo de n para a meta
- e.g., $h_{DLR}(n)$ = distância em linha reta de n até Bucharest
- Busca gulosa pela melhor escolha expande o nó que *parece* mais perto da meta

Exemplo de Busca Gulosa pela Melhor Escolha

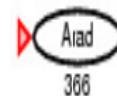
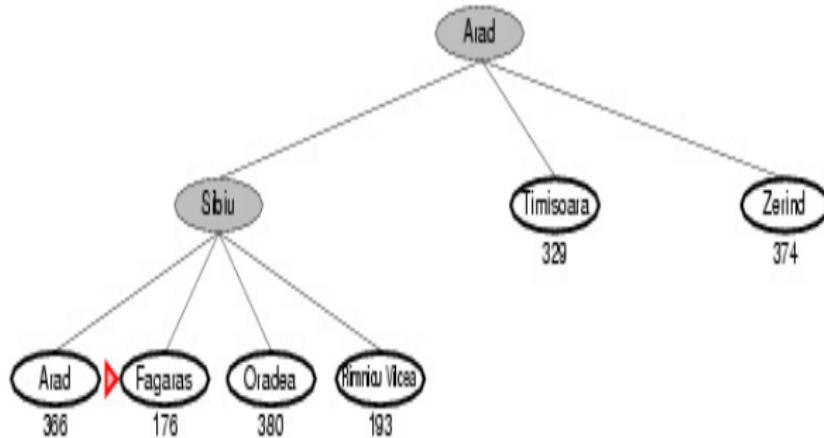


Figure 10: Fonte: Russell e Norvig [3]

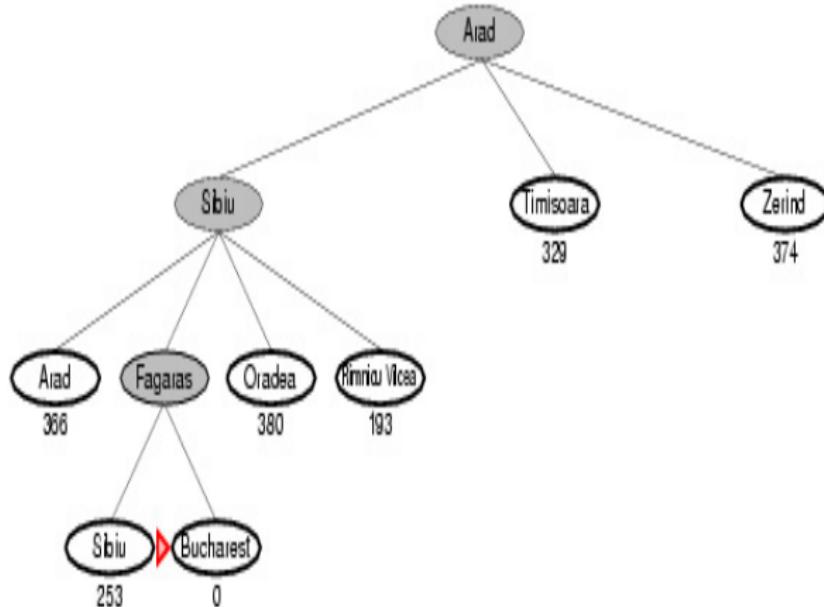
Exemplo de Busca Gulosa pela Melhor Escolha



Exemplo de Busca Gulosa pela Melhor Escolha



Exemplo de Busca Gulosa pela Melhor Escolha



Propriedades da Busca Gulosa pela Melhor Escolha

- Completa? Não – pode ficar presa em loops, e.g., Iasi ⇒ Neamt ⇒ Iasi ⇒ Neamt ⇒.
- Tempo? $\mathcal{O}(bm)$, mas uma boa heurística pode melhorar dramaticamente.
- Espaço? $\mathcal{O}(bm)$ – mantém todos os nós na memória.
- Ótima? Não.

Busca A*

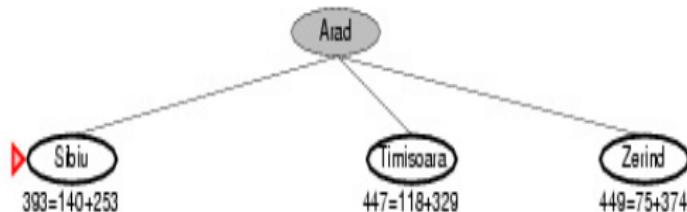
- Idéia: evitar expandir caminhos que já são caros
- Função de avaliação $f(n) = g(n) + h(n)$
 - $g(n)$ = custo para chegar a n ,
 - $h(n)$ = custo estimado de n a meta,
 - $f(n)$ = custo total estimado do caminho através de n até a meta.

Exemplo de Busca A*

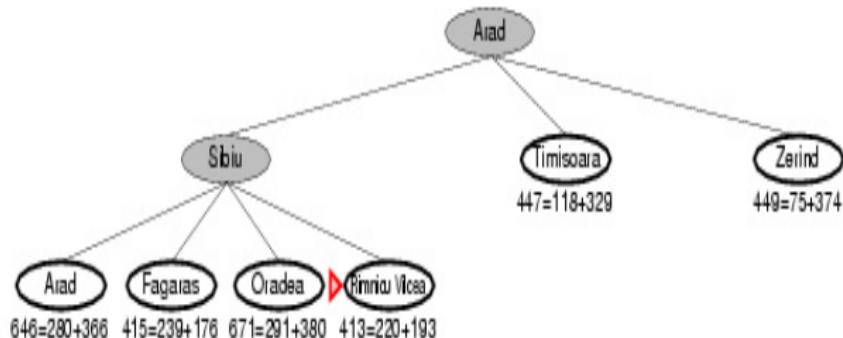


Figure 11: Fonte: Russell e Norvig [3]

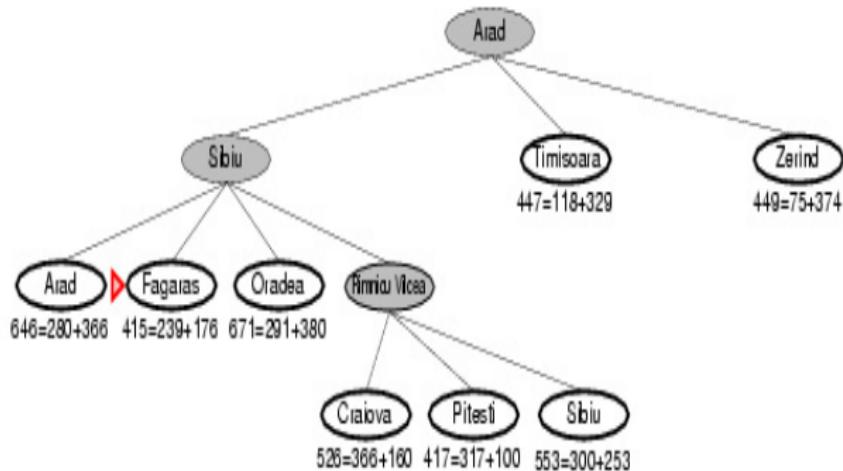
Exemplo de Busca A*



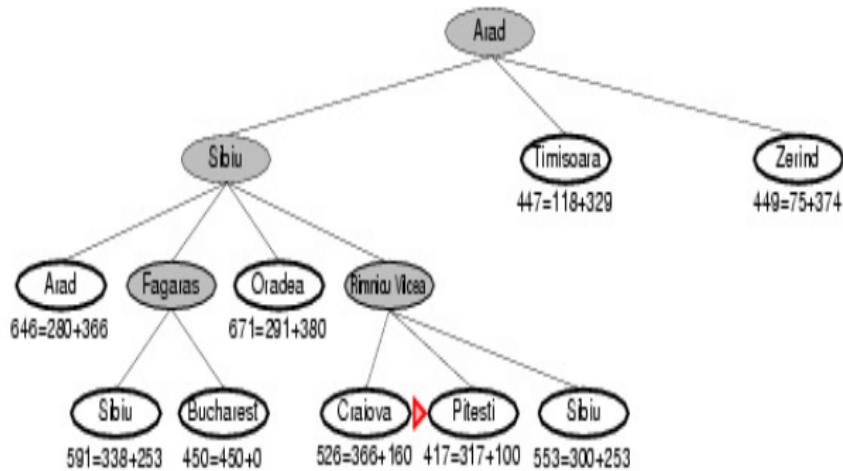
Exemplo de Busca A*



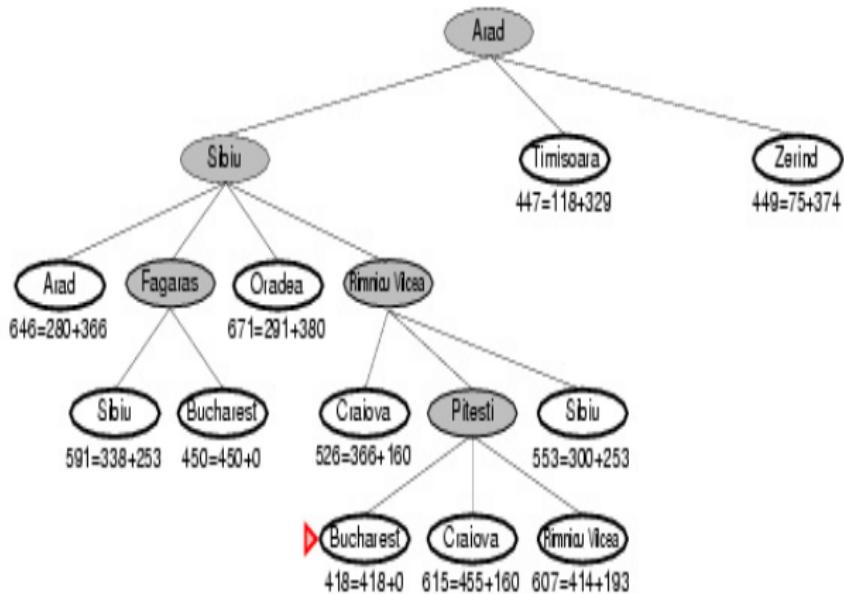
Exemplo de Busca A*



Exemplo de Busca A*



Exemplo de Busca A*



Heurísticas Admissíveis

- Uma heurística $h(n)$ é *admissível* se para todo nó n , $h(n) \leq h^*(n)$, onde $h^*(n)$ é o custo *real* para chegar ao estado meta a partir de n .
- Uma heurística admissível *nunca superestima* o custo para chegar a meta, i.e., ela é *ótima*.
 - Exemplo: $h_{DLR}(n)$ (nunca superestima a distância rodoviária real)

Teorema

Se $h(n)$ é admissível, A que usa BUSCA EM GRAFOS é ótima.*

Heurísticas Consistentes

- Uma heurística é *consistente* se para todo nó n , todo sucessor n' de n gerado por qualquer ação a ,
 - $h(n) \leq c(n, a, n') + h(n')$
- Se h é consistente, tem-se
 - $f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n)$
 i.e., $f(n)$ é não decrescente ao longo de qualquer caminho.
- Teorema:** Se $h(n)$ é consistente, A* que usa BUSCA EM GRAFOS é ótima.

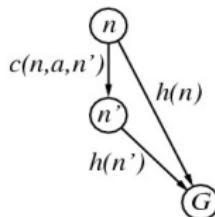


Figure 12: Fonte: Russell e Norvig [3]

Propriedades de A*

- Completa? Sim.
- Tempo? Exponencial.
- Espaço? Mantém todos os nós na memória.
- Ótima? Sim.

Heurísticas Admissíveis

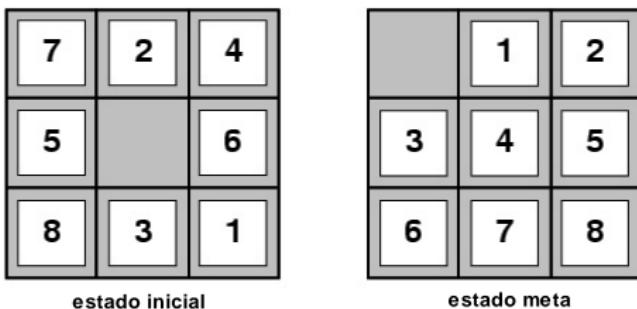


Figure 13: Fonte: Russell e Norvig [3]

- E.g., para o quebra-cabeça de 8 peças:
 - $h_1(n)$ = número de peças fora do lugar
 - $h_2(n)$ = total da distância de Manhattan (i.e., número de quadrados da posição desejada de cada peça)
 - $h_1(S) = 8$
 - $h_2(S) = 3+1+2+2+2+3+3+2 = 18$

Dominância

- Se $h_2(n) \geq h_1(n)$ para todo n (ambas admissíveis), então h_2 **domina** h_1
- h_2 é melhor para busca

Table 2: Custos de busca típicos (número médio de nós expandidos).

d	BAI	$A^*(h_1)$	$A^*(h_2)$
12	3.644.035	227	73
24	muitos nós	39.135	1.641

- BAI = Busca por Aprofundamento Iterativo

Problemas Relaxados

- Um problema com poucas restrições nas ações é chamado de **problema relaxado**.
- O custo de uma solução ótima para um problema relaxado é uma heurística admissível para o problema original
- Se as regras do quebra-cabeça de 8 peças são relaxadas tal que uma peça possa ser movida para *qualquer lugar*, então $h_1(n)$ fornece a solução mais curta
- Se as regras são relaxadas tal que uma peça possa ser movida para *qualquer quadrado adjacente*, então $h_2(n)$ fornece a solução mais curta

Algoritmos de Busca Local

- Em muitos problemas de otimização, o caminho para a meta é irrelevante; o estado meta por si só é a solução
 - Espaço de estados = conjunto de configurações “completas”
- Achar a configuração que satisfaz as restrições, e.g., problema das n rainhas
 - Em tais casos, pode-se usar *algoritmos de busca local*.
 - Mantenha um único estado “corrente”, tente melhorá-lo.

Exemplo: n Rainhas



Figure 14: Fonte: Russell e Norvig [3]

Busca de Subida de Encosta

- Problema: dependendo do estado inicial pode ficar preso em um máximo local

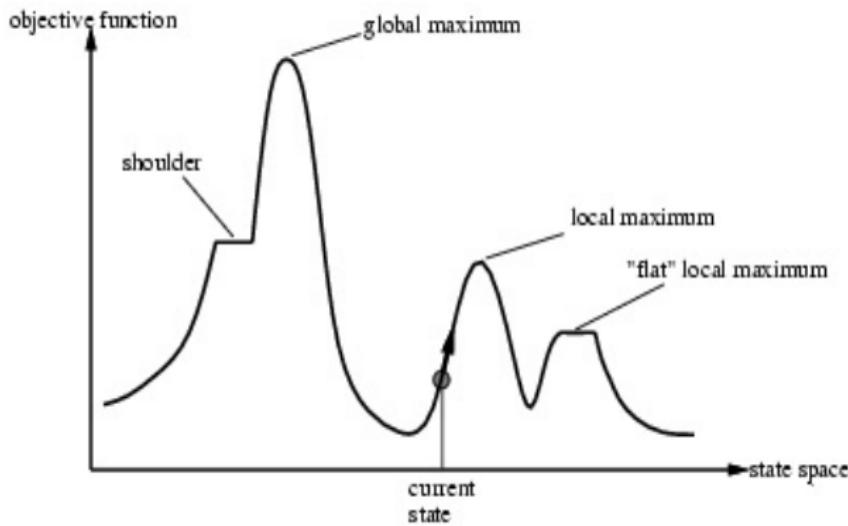


Figure 15: Fonte: Russell e Norvig [3]

Subida de Encosta: Problema das 8 Rainhas

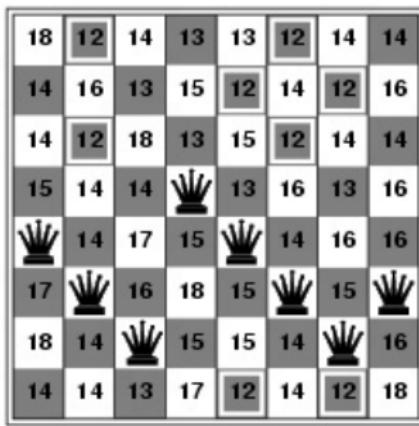


Figure 16: Fonte: Russell e Norvig [3]

- $h =$ número de pares de rainhas que se atacam, direta ou indiretamente
- $h = 17$ para o estado acima

Subida de Encosta: Problema das 8 Rainhas

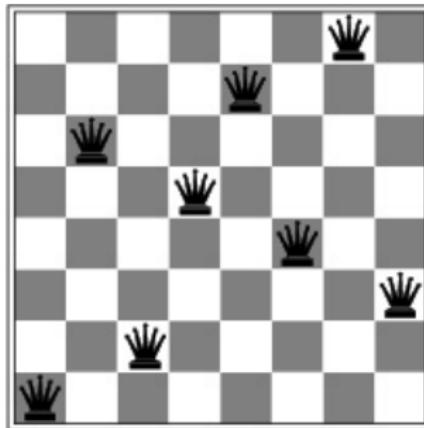


Figure 17: Fonte: Russell e Norvig [3]

- Um mínimo local com $h = 1$

Subida de Encosta: Quebra-cabeça de 8 Peças

-4

2	8	3
1	6	4
7		5

↓

-3

2		3
1	8	4
7	6	5

↓

-1

1	2	3
	8	4
7	6	5

↓

-3

2	8	3
1		4
7	6	5

↓

-2

	2	3
1	8	4
7	6	5

↓

0

1	2	3
8		4
7	6	5

Busca de Recozimento Simulado

- Idéia: escapar dos máximos locais permitindo alguns movimentos “ruins,” mas gradualmente diminuindo sua freqüência.
- Propriedades:
 - Pode-se provar: Se T diminui lentamente, então a busca de recozimento simulado (*simulated annealing*) achará um ótimo global com probabilidade de aproximação 1.
 - Muito usada em layout de VLSI, escalonamento de empresas aéreas, etc.

Busca em Feixe Local

- Mantém a trilha de k estados ao invés de apenas um.
- Começa com k estados gerados aleatoriamente.
- A cada iteração, todos os sucessores de todos os k estados são gerados.
- Se um deles for o estado meta, pára; caso contrário seleciona os k melhores sucessores a partir da lista completa e repete.

Algoritmos Genéticos

- Um estado sucessor é gerado combinando dois estados pais.
- Começa com k estados gerados aleatoriamente (população).
- Um estado é representado como uma cadeia de caracteres de um alfabeto finito (freqüentemente uma cadeia de 0's e 1's).
- Função de avaliação (função de *fitness*). Valores mais altos para estados melhores.
- Produz a próxima geração de estados através de seleção, cruzamento e mutação.

Algoritmos Genéticos



Figure 18: Fonte: Russell e Norvig [3]

- Função de *fitness*: número de pares de rainhas que não se atacam ($\min = 0$, $\max = 8 \times 7/2 = 28$).
 - $24/(24+23+20+11) = 31\%$.
 - $23/(24+23+20+11) = 29\%$ etc.

Algoritmos Genéticos

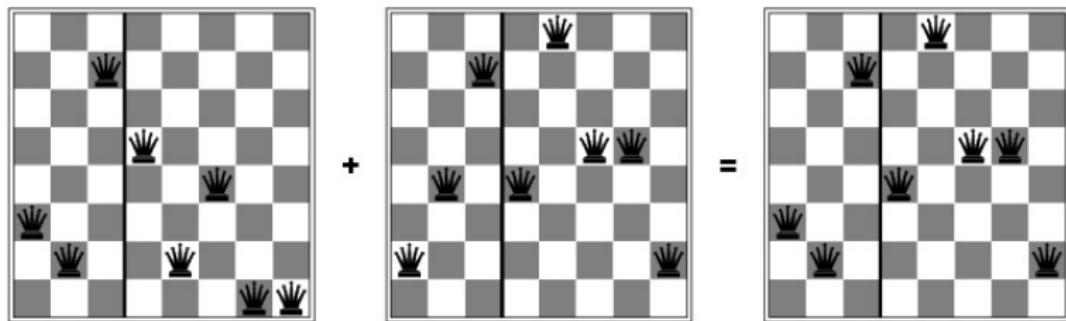


Figure 19: Fonte: Russell e Norvig [3]

Procedimentos Heurísticos de Busca em Grafos

Argumentos a favor do uso de heurística:

- Raramente precisa-se da solução ótima; uma boa aproximação normalmente serve muito bem. De fato, existem evidências de que as pessoas, quando resolvem problemas, procuram qualquer solução que satisfaça algum conjunto de requisitos, e assim que encontram, dão-se por satisfeitas.
- Ainda que as aproximações produzidas por heurísticas possam não ser muito boas no pior caso, os piores casos raramente acontecem na vida real.
- A tentativa de entender por que uma heurística funciona, ou por que não funciona, leva a um entendimento mais aprofundado do problema.

Exercício

Especifique uma base de dados global, regras e uma condição de terminação para um sistema de produção para resolver o seguinte problema dos jarros de água:

- Dado um jarro de 5 litros cheio de água e um jarro de 2 litros vazio, como se pode obter precisamente 1 litro no jarro de 2 litros? A água pode ser desperdiçada ou transferida de um jarro para outro; entretanto, só os 5 litros iniciais estão disponíveis.

Referências I

- [1] Nilsson, N. J.
Principles of Artificial Intelligence.
Springer-Verlag; 1982.
- [2] Rosa, J. L. G.
Fundamentos da Inteligência Artificial.
Editora LTC. Rio de Janeiro, 2011.
- [3] Russell, S., Norvig, P.
Inteligência Artificial. 2a. Edição.
Editora Campus, 2004.