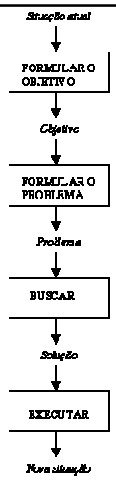
IA

Busca Não Informada – Revisão

Busca Heurística



Solução de problemas

Custo total da resolução do problema =

Custo de busca + Custo da execução

Objetivo

- É um conjunto de estados do mundo.
- Os estados do objetivo satisfazem a solução do problema.

Formular Objetivos

- O que é?
- Por que fazer?
- Em que se baseia?

Formular objetivos

- É simplificação do problema de decisão do agente através da rejeição, sem consideração adicional, de cursos de ações do agente que não satisfazem o problema
- Os objetivos ajudam a organizar o comportamento limitando os objetivos que o agente está tentando alcançar
- É baseada na situação atual e na medida de desempenho do agente

Formular problemas

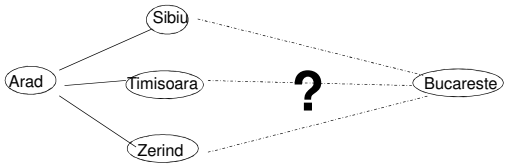
- Formulei meus objetivos, e daí?!
- Primeiramente você deve decidir que espécie de estados e ações devem ser consideradas

Formular problemas

- Agora o agente começa a trabalhar?
- É o processo de decidir que ações e estados devem ser considerados, dado um objetivo;
- Um agente com várias opções imediatas de valor desconhecido pode decidir o que fazer examinando primeiro diferentes seqüências de ações possíveis que levam a estados de valor conhecido, e depois escolhendo a melhor seqüência

Exemplo

- Considerando que as ações do agente são no nível de dirigir de uma cidade até outra.
- E os Estados?



Buscar Solução

- É o processo de procurar a seqüência de ações
- Como funciona?
- Um algoritmo de busca recebe um problema como entrada, e retorna uma **solução** sob a forma de seqüência de ações

Busca de soluções (cont.)

- Percorrer o espaço de estados a partir de uma ***árvore de busca (ou arborescência de busca);***
- *Expandir* o estado atual aplicando a função sucessor, *gerando* novos estados;
- Busca: seguir um caminho, deixando os outros para depois;
- A ***estratégia de busca*** determina qual caminho seguir.

Executar

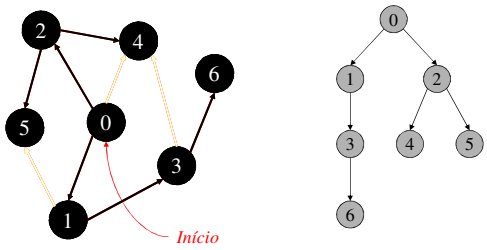
- É a execução das ações a partir de uma solução encontrada

Medição de desempenho de um algoritmo

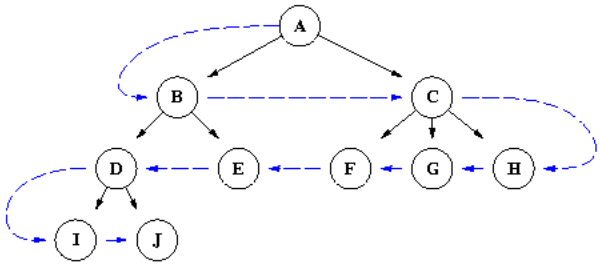
- **Completeza:** Sempre encontra uma solução se esta existir;
- **Otimização**: Encontra sempre a solução ótima?
- **Complexidade de tempo**: Tempo gasto para encontrar uma solução;
- **Complexidade de espaço**: Memória necessária para encontrar uma solução.

Busca em Grafos

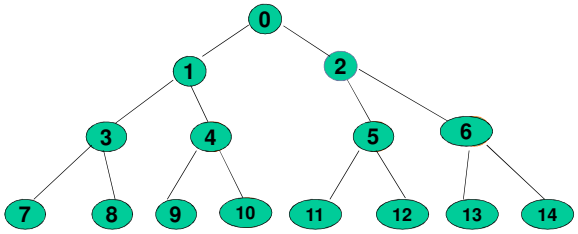
Utilizando as arestas escolhidas na ordem da busca, é possível montar uma árvore de busca:



Busca em Largura ou Breadth-First Search (BFS)

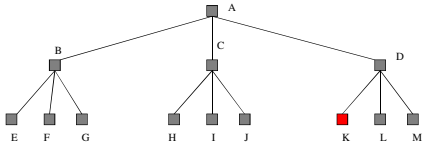


Busca em Largura ou Breadth-First Search (BFS)



O números representam a ordem de pesquisa dos nós

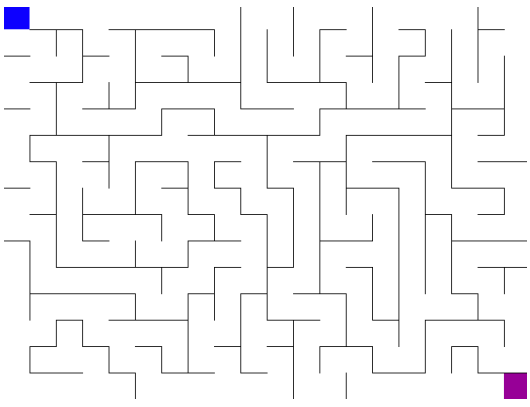
Busca em Largura - Algoritmo



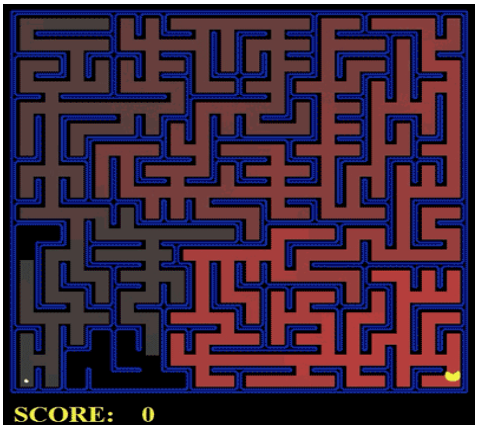
- Buscando K, situação da pilha

1) L={A}
2) L={B C D}
3) L={C D E F G}
4) L={D E F G H I J}
5) L={E F G H I J K L M}
6) L={F G H I J K L M}
8) L={G H I J K L M}
9) L={H I J K L M}
9) L={I J K L M}
10) L={J K L M}
11) L={K L M}
12) Achou K! Interrompe o processo

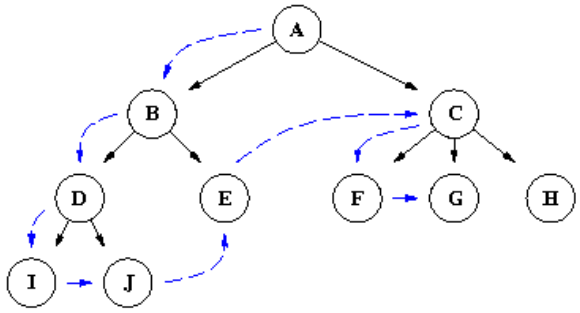
BFS – Exemplo em labirinto



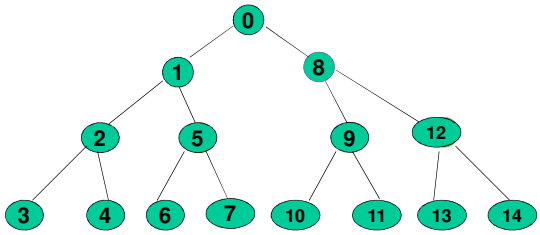
BFS – Exemplo no jogo Pac Man



Busca em Profundidade

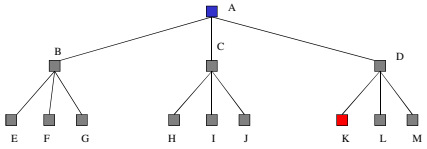


Busca em Profundidade



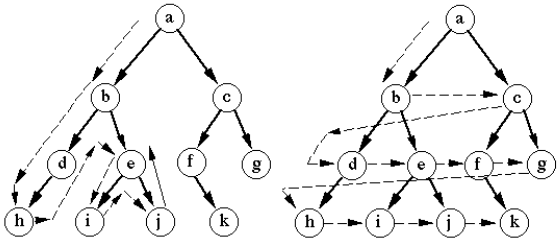
O números representam a ordem de pesquisa dos nós

Busca em Profundidade ou Depth-First Search (DFS)



• Buscando K, situação da pilha:

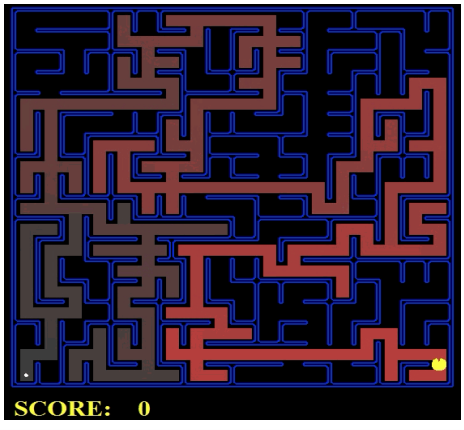
1) L={A}
2) L={B C D}
3) L={E F G C D}
4) L={F G C D}
5) L={G C D}
6) L={C D}
7) L={H I J D}
8) L={I J D}
9) L={J D}
10) L={D}
11) L={K L M}
12) Achou K! Interrompe



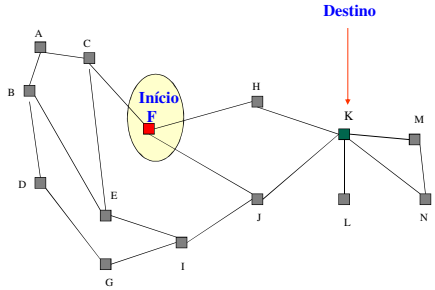
Depth-first search

Breadth-first search

DFS – Exemplo no jogo Pac Man



Caminhos entre Cidades



Estados, Objetivos, Operadores

- Estados - Cidades visitadas
- Estado inicial - Cidade de partida
- Estado final - Cidade de chegada
- Operador - Próximas cidades

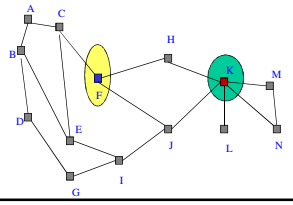
  p. ex. - pega cidades conectadas que ainda

  não tenham sido visitadas

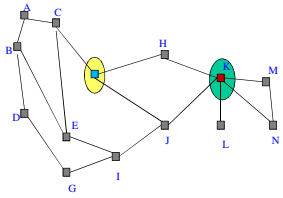
- Lembrete: O algoritmo de busca não armazena tudo (mapa pode estar numa base de dados, por exemplo); e guarda o caminho percorrido

Exemplo (em Profundidade)

- Estado Inicial: F
- Estado Final: K
- Busca em Profundidade
  - F (aplicando "prox. não visitada) ->
  - C J H ->
  - A E J H ->
  - B E J H ->
  - D E E J H ->

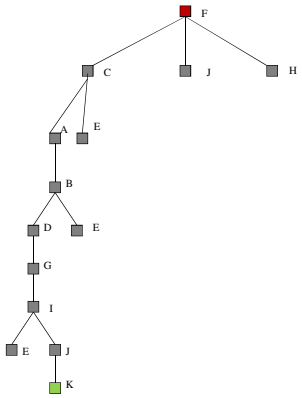


Exemplo (cont.)



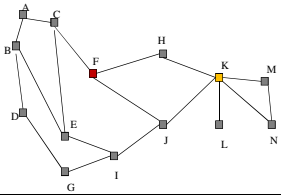
- G E E J H->
- I E E J H->
- E J E E J H -> (E geraria C e B, que já foram visitados. Logo, sai da lista)
- J E E J H-> (J geraria F e K, mas F já foi. Logo, só entra o K)
- K E E J H -> estado final atingido

- Caminho da busca: F C A B D G I E J K



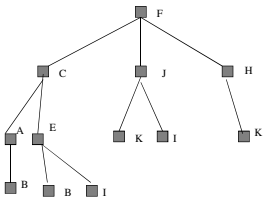
Exemplo (em largura)

- Estado Inicial: F
- Estado Final: K
- Busca em Profundidade
  - F (aplicando "prox. não visitada) ->
  - C J H ->
  - J H A E ->
  - H A E K I ->
  - A E K I K ->
  - E K I K B ->



Exemplo (em largura)

- K I K B B I -> Estado Final atingido



Exercício

- Jogo das 8 Peças (8-Puzzle)
- Estado final: Configuração da fig. 1
- Estado final: Configuração da fig. 2
- Objetivo: buscar, a partir de um estado inicial, o estado final

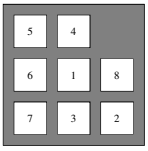


Fig. 1

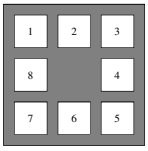
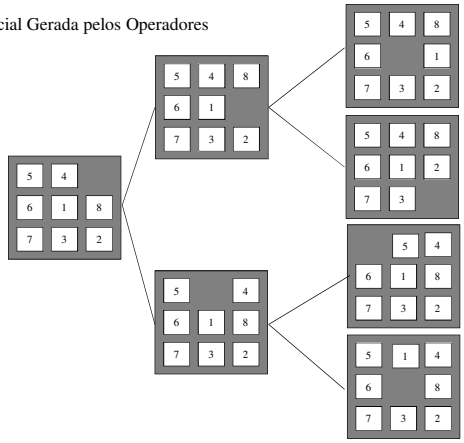


Fig. 2

Exercício

- Árvore: construída por operadores que são os movimentos possíveis
- Operadores: vazio se move para direita, esquerda, cima ou baixo

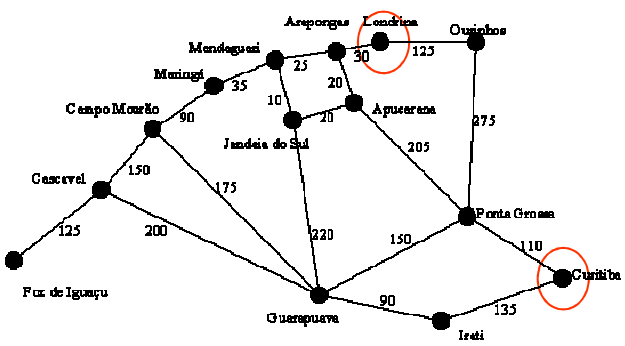
Árvore Parcial Gerada pelos Operadores

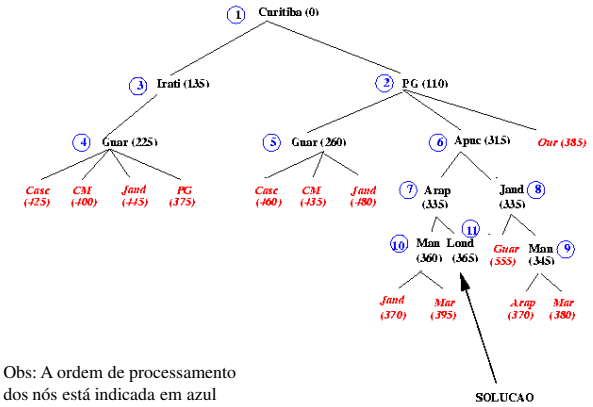


Busca em largura a custo uniforme (Branch-and-Bound)

- Modificação do algoritmo de busca em largura para aumentar o conjunto de problemas pelos quais o método retorna uma solução ótima;
- Ao invés de dar prioridade aos nodos que se encontram no nível menos profundo, o algoritmo escolhe o nodo que tem o menor custo;
- Agora, a condição para obter uma solução ótima é que o custo para passar ao próximo estado nunca seja negativo.

Exemplo: Busca do caminho mais curto entre Curitiba e Londrina:





Obs: A ordem de processamento dos nós está indicada em azul

B&B: Vantagens & Desvantagens

- Vantagens desse algoritmo:
  - Completo
  - Ótimo, se o custo até o próximo nodo nunca for negativo.
- Desvantagens:
  Complexidade em memória e tempo igual à da busca em largura: $O(b^p)$
  - Onde "b" é o fator de ramificação e "p" a profundidade.

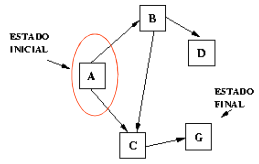
•Variações sobre busca em profundidade:

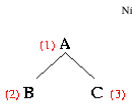
- *Busca com profundidade limitada:*
  - *Escolhe-se um valor limite de profundidade que a busca não pode ultrapassar;*
  - *Isso vai dar certo somente se pudermos confiar que a solução encontra-se dentro desse limite;*
  - *Se não escolhermos um bom valor de limite de profundidade, pode não retornar uma solução;*
  - *Esse tipo de busca resolve o problema da incompletude, mas ainda é possível que seja retornada uma solução não ótima.*

•Variações sobre busca em profundidade:

- *Busca em profundidade iterativa:*
  - *Compromisso entre a busca em largura e a busca em profundidade.*
  - *Nesse caso, tentamos primeiro uma busca em profundidade com limite de profundidade 0.*
  - *Se não encontramos uma solução, repetimos com limite de profundidade 1, 2, 3 e assim por diante até achar uma solução.*
  - *Essa solução tem a vantagem de economia em memória, pois é a busca em profundidade que é utilizada, e também vai achar a solução ótima (se a função de custo é uniforme), pois não passamos a um nível superior de profundidade antes de ter esgotado o nível precedente.*

Busca em Profundidade Iterativa





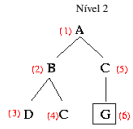
•A busca em profundidade tradicional pode retornar a solução não ótima ABCG

•Na busca em profundidade iterativa, uma primeira busca será realizada até o nível 0.

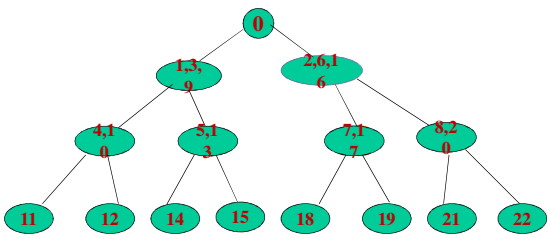
•Nesse caso somente o estado A será visitado.

• Como isso não é o estado final, recomeçamos uma busca até o nível 1 .

•A solução ótima é encontrada no nível 2



Busca em Profundidade Iterativa ou Depth-First Iterative Deepening (IDDF)



O números representam a ordem de pesquisa dos nós

Busca Bidirecional

- Duas buscas são realizadas em paralelo:
  - Uma a partir do estado inicial e outra a partir do objetivo.
  - Temos uma solução quando as duas se encontram.
  - Para poder utilizar essa solução, temos que respeitar as seguintes exigências:
    - definir qual tipo de busca realizada nas duas direções de maneira a maximizar as chances delas se encontrarem;
    - estabelecer um método para verificar o encontro.

Busca Bidirecional

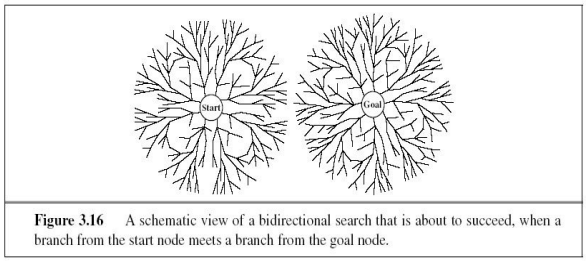


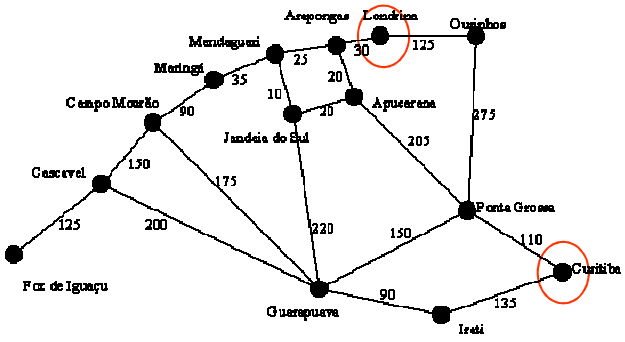
Figure 3.16 A schematic view of a bidirectional search that is about to succeed, when a branch from the start node meets a branch from the goal node.

Complexidade dos algoritmos básicos de busca

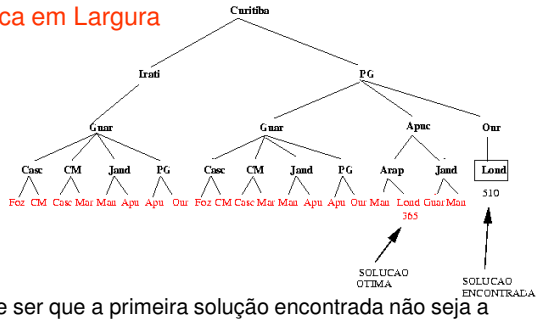
	Profund.	Largura	*Custo unif.*	*Prof. limit.*	*Prof. iterat.*
Tempo	b^m	b^d	b^d	b^l	b^d
Memória	bm	b^d	b^d	bl	bd
Sol. ótima	Não	Sim*	Sim**	Não	Sim*
Completude	Não	Sim	Sim	Sim, se l >= d	Sim

b = fator de ramificação
d = profundidade da solução
l = limite de profundidade especificado
m = profundidade máxima atingida na busca
* Somente se o custo para de um estado ao próximo é sempre o mesmo (função de custo uniforme).
** Somente se o custo não diminuir quando o caminho aumenta.

Exemplo: Busca do caminho mais curto entre Curitiba e Londrina:

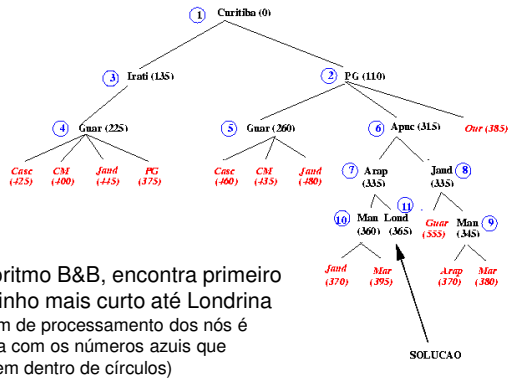


Busca em Largura



- Pode ser que a primeira solução encontrada não seja a ótima, pois dentre os nós ainda esperando para serem visitados, pode existir um que tenha valor menor;
- Note que nesse caso eliminamos os nodos redundantes;
- Isso foi feito por que é impossível que o caminho mais curto passe duas vezes pela mesma cidade.

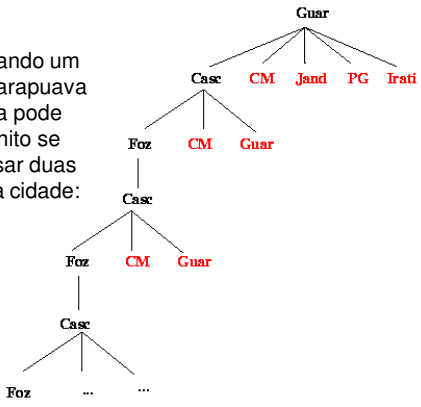
Busca em largura a custo uniforme (Branch-and-Bound)



O algoritmo B&B, encontra primeiro o caminho mais curto até Londrina
(a ordem de processamento dos nós é indicada com os números azuis que aparecem dentro de círculos)

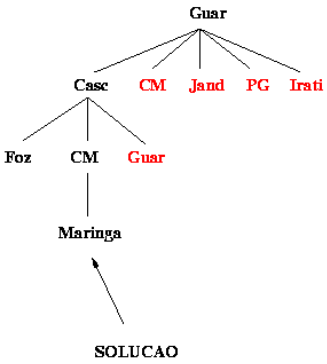
Busca em profundidade (depth-first)

Se por exemplo, estivermos procurando um caminho entre Guarapuava e Maringá, a busca pode entrar em loop infinito se não evitamos passar duas vezes pela mesma cidade:



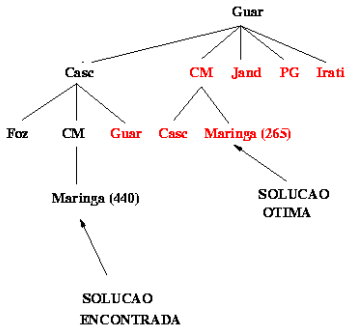
Busca em profundidade (depth-first)

Evitando retornar à mesma cidade, achamos rapidamente uma solução.



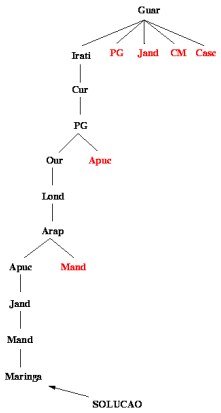
Busca em profundidade (depth-first)

- Mas esta solução não é ótima.
- A solução ótima está escondida nos nodos que não foram anteriormente visitados



Busca em profundidade

- o algoritmo de busca em profundidade, se ele é menos exigente em memória, não é melhor que a busca em largura no que se refere ao tempo de execução.
- Depois da expansão de um nodo, a ordem da aparição dos nodos filhos na pilha influencia muito a busca.
- Veja por exemplo como a ordem pode tornar muito ruim a busca do caminho entre Guarapuava e Maringá:



Busca Informada ou Heurística

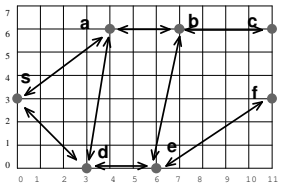
Heurística

- Na criação e elaboração de algoritmos busca-se fazer algoritmos que tenham um tempo de execução sempre aceitável, e ser a solução ótima ou provavelmente boa para o problema em todos os casos;
- Um algoritmo heurístico pode ser ou um algoritmo que encontra boas soluções a maioria das vezes, mas não tem garantias de que sempre a encontrará,
  - Ou um algoritmo que tem processamento rápido, mas não tem provas de que será rápido para todas as situações.
- Diz-se que se tem uma *boa* (ou alta) heurística se o objeto de avaliação está muito próximo do objetivo;
  - Diz-se de *má* (ou baixa) heurística se o objeto avaliado estiver muito longe do objetivo.
- Etimologicamente a palavra **heurística** vem da palavra grega *Heuriskein*, que significa descobrir (e que deu origem também ao termo *Eureca*).

Busca Heurística

Se soubermos que estamos chegando mais próximos ao destino podemos fazer uma busca um pouco mais esperta.

Nó	Vizinhos	Coordenadas
s	(a d)	(0 3)
a	(s b d)	(4,6)
b	(a c e)	(7,6)
c	(b)	(11,6)
d	(s a e)	(3,0)
e	(b d f)	(6,0)
f	(e)	(11,3)

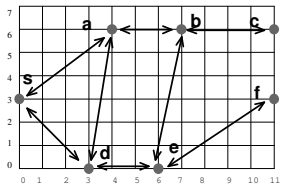


Hill Climbing: Seleciona o ramo que nos leva mais próximo ao objetivo.

Best First: Ordena os nós baseado na distância aos nós mais próximos.

Hill Climbing versus Best First

- *Best First e Hill-Climbing são frequentemente equivalentes;*
- *H-C é local, ao passo que B-F é mais global.*



Roteiro para B-F e HC

```
((sa)(sd))
((sab)(sad)(sd))
((sabc)(sabe)(sad)(sd))
((sabef)(sabed)(sad)(sd))
```

Busca Heurística

- Uma busca heurística é uma busca que utiliza uma função *h(n)* que, para cada nodo *n* do espaço de busca, fornece uma avaliação do custo para atingir o estado final.
- A função *h(n)* é chamada **função heurística**.
- Variações de algoritmos:
  - **Melhor escolha (ou best-first);**
  - **Busca em largura com custo uniforme;**
  - **Busca gulosa (ou Greedy);**
  - **Algoritmo A***

Busca Heurística

- Adiciona-se alguma informação relativa ao domínio, para selecionar qual é o melhor caminho a ser pesquisado;
- Define-se uma função heurística, h(n), que estima a "vantagem" de um nó n com respeito a atingir o objetivo;
- Especificamente, **h(n)** = custo estimado (ou distância) do caminho de mínimo custo de n ao estado objetivo;
- **h(n) tenta contabilizar o custo da pesquisa futura, enquanto g(n) custeia a pesquisa passada (da origem até o nó atual).**

Busca Heurística

- **h(n)** é um valor estimado, baseado em informação específica do domínio - que é **computável** a partir da descrição do estado corrente;
- A heurística não garante soluções factíveis e frequentemente não tem lastro na teoria;
- Em geral:
  - h(n) >= 0 para todo nó n;
  - h(n) = 0, implica que n é um nó objetivo;
  - h(n) = infinito, implica que n é um caminho sem saída, a partir do qual não se consegue atingir o objetivo.

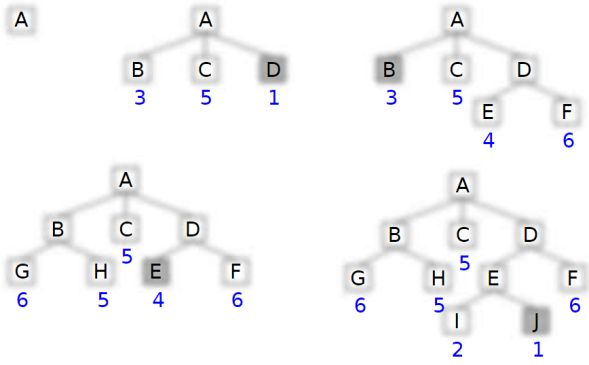
Melhor escolha (ou Best-First)

- Combina as vantagens dos algoritmos DFS e BFS em um único método;
  - Vantagem herdada da Busca em Profundidade: Nem todos os caminhos necessitam ser percorridos;
  - Vantagem herdada da Busca em Largura: Não fica estacionada em caminhos sem saída.
- Casos especiais: Busca Gulosa e A*

Melhor escolha (ou Best-First)

- A diferença entre este algoritmo e o algoritmo geral é que ele usa uma função *f(n)*, que dá um valor estimado para o caminho de menor custo a partir do nó *n* até o destino;
- Um nó é selecionado para expansão com base em uma função de avaliação f(n);
- O nodo que tiver o menor valor é escolhido para continuar a busca;
- Esta é uma forma genérica de se referir aos métodos informados de busca.

Melhor escolha (ou Best-First)



Busca em largura com custo uniforme

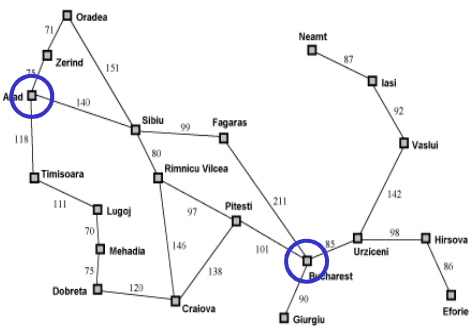
- Utiliza a função $f(n) = g(n)$, onde $g(n)$ dá o custo a partir do estado inicial até o nodo n;
- Esse algoritmo não garante uma solução ótima.

Busca Gulosa ou Greedy

- Usa $f(n) = h(n)$, onde $h(n)$ é uma estimativa do custo do caminho mais curto do nodo n até o objetivo;
- O resultado é uma busca "gulosa" , porque sempre vai escolher o maior passo possível, sem se preocupar se ao final vai se obter a melhor solução;
- Por exemplo, para achar o caminho de Guarapuava até Curitiba, a escolha inicial é entre as seguintes cidades: Cascavel, Campo Mourão, Jandaia do sul, Ponta Grossa e Irati;
  - O algoritmo vai escolher Ponta Grossa, pois é o mais perto de Curitiba.
- Então, o algoritmo vai dar uma resposta não ótima, pois o caminho que passa por Irati é mais curto;

Busca Gulosa: exemplo

Roteiro para ir de Arad a Bucharest ??



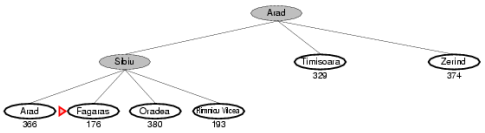
Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy: exemplo

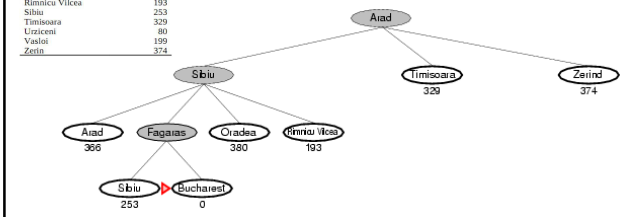
Greedy: exemplo

Greedy: exemplo



Greedy: exemplo

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Faragas	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mechadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vasloi	199
Zerin	374



Problemas com a busca Greedy

- Não completa. Vide por exemplo o caminho entre Iasi e Fagaras?
- Estaciona em mínimo local e platôs;
- Loops Infinitos;
- Como incorporar heurísticas em busca sistemática?

Algoritmo A*

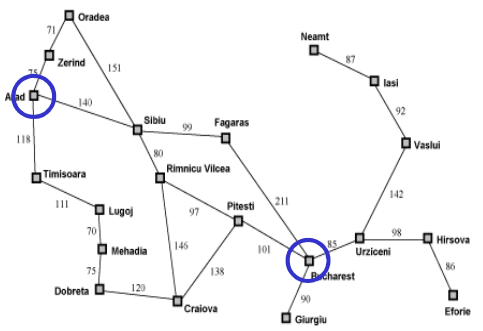
- Combina as duas funções, de forma que: *f(n) = g(n) + h(n);*
- g(n) é o custo real da origem até o nó n;
- h(n) é a distância em linha reta do objetivo até o nó n.

  |<-- **g** --->|<---- **h** ---->|

  |<-------- **f** --------->| **f = g + h**
- Essa função estima o custo total do estado inicial até o objetivo;
- A obtenção da solução ótima depende da função heurística;
- Se *h(n)* não fizer uma boa avaliação do custo até o objetivo, tem chance de "perder" a solução ótima.

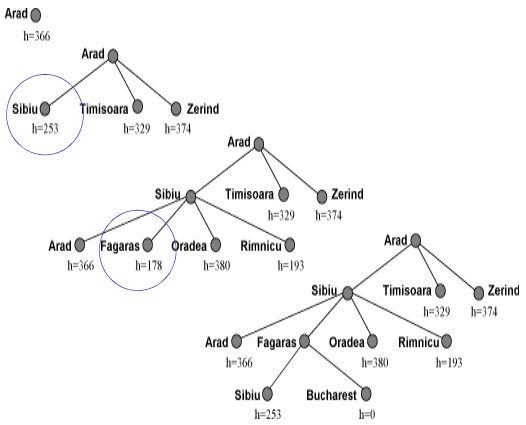
Algoritmo A* - Exemplo 2

Caminho entre Arad e Bucharest

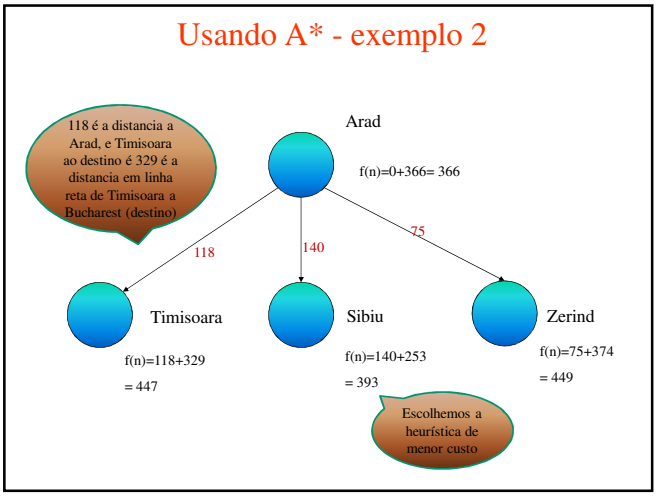


Straight line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

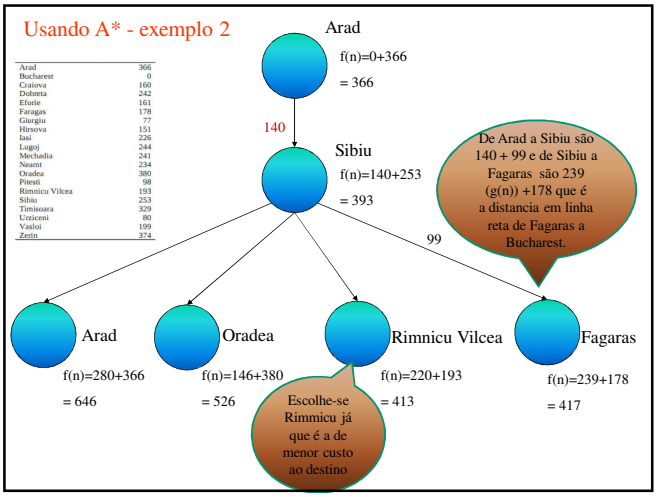
Se fosse pela Busca Gulosa (ou Greedy)...



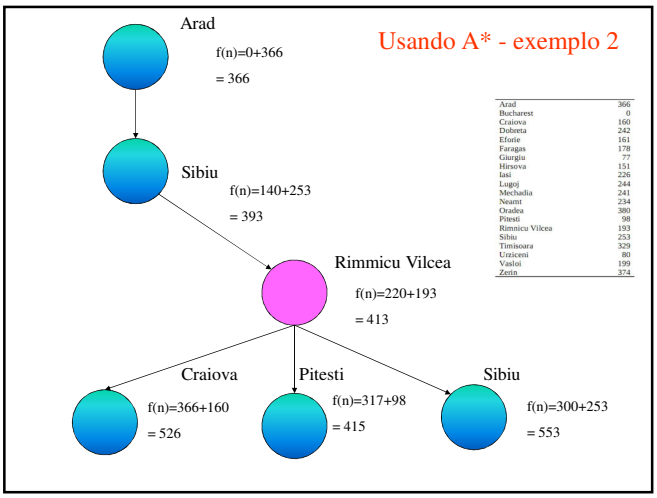
Usando A* - exemplo 2



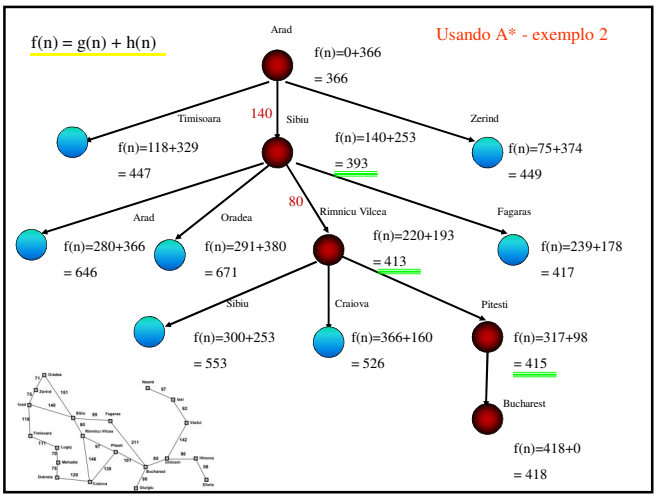
Usando A* - exemplo 2



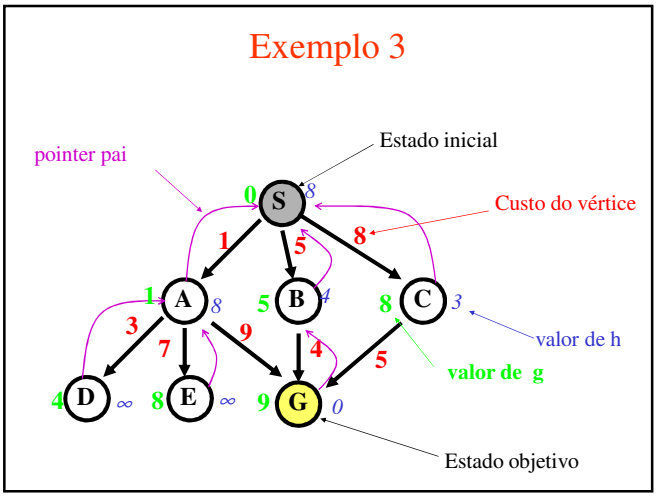
Usando A* - exemplo 2



Usando A* - exemplo 2

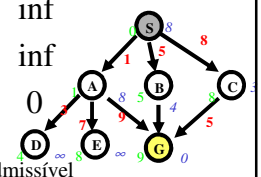


Exemplo 3



N	g(n)	h(n)	f(n)	h*(n)
S	0	8	8	9
A	1	8	9	9
B	5	4	9	4
C	8	3	11	5
D	4	inf	inf	inf
E	8	inf	inf	inf
G	9	0	9	0

- **Exemplo**

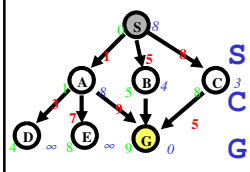


- h*(n) é a função heurística.
- Como h(n) <= h*(n) para todo n, h é admissível
- Caminho ótimo = S B G com custo 9.

Busca Greedy ou Gulosa

f(n) = h(n)

Nós expandidos **lista de nós**



{ S(8) }
S { C(3) B(4) A(8) }
C { G(0) B(4) A(8) }
G { B(4) A(8) }

- O caminho encontrado foi: S C G, 3 nós expandidos.
- Embora seja rápido, não é ótimo.

Busca A*

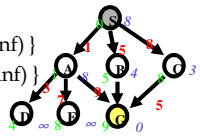
f(n) = g(n) + h(n)

N	g(n)	h(n)	f(n)
S	0	8	8
A	1	8	9
B	5	4	9
C	8	3	11
D	4	inf	inf
E	8	inf	inf
G	9	0	9

nós exp. Lista de nós

{ S(8) }

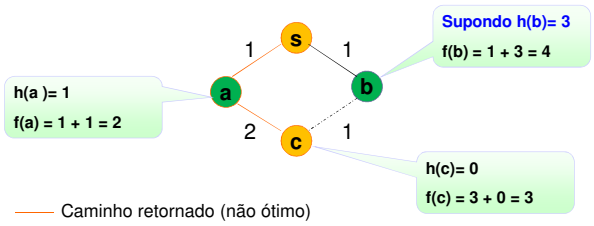
S { A(9) B(9) C(11) }
A { B(9) G(10) C(11) D(inf) E(inf) }
B { G(9) G(10) C(11) D(inf) E(inf) }
G { C(11) D(inf) E(inf) }



- O caminho encontrado foi: S B G, com expansão de 4 nós.
- Muito rápido e também ótimo.

A* - Qualidade da Solução

- Depende da função Heurística, mas de modo geral, A* retorna a solução ótima.
- **Exemplo** partindo de **s**, com destino a **c**



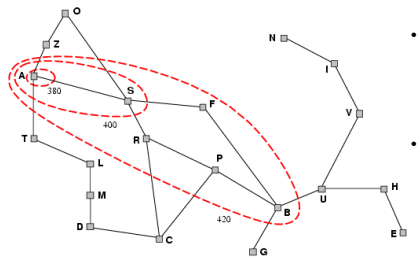
- O problema neste caso é que o valor da heurística para o nó b (3), superestima o custo para atingir o objetivo a partir do nó b, que é somente 1.

Heurística Admissível

- Uma heurística *h(n)* é admissível, se para cada nó *n*,
  - $h(n) \leq h^*(n)$, onde $h^*(n)$ é o custo verdadeiro de alcançar o estado objetivo a partir de *n*.
- Uma heurística admissível nunca superestima o custo de alcançar o objetivo, isto é, ela é otimista;
- Exemplo: $h_{DLR}(n)$ (distância em linha reta nunca é maior que distância pela estrada);
- Teorema: Se *h(n)* é admissível, A* fornece a solução ótima.

A* é ótima com heurística consistente

- A* expande nós em ordem crescente de valores de *f*.
- Gradualmente adiciona "*contornos*" de nós.
- Contorno *i* tem todos os nós com $f=f_i$, onde $f_i < f_{i+1}$



- Se h(n)=0 temos uma busca de custo uniforme ⇒ círculos concêntricos.
- Quanto melhor a heurística, mais direcionados ao objetivo serão os círculos

Propriedades da Busca A*

- Completa? Sim (a não ser que exista uma quantidade infinita de nós com f ≤ *f(G)*)
- Tempo? Complexidade Exponencial ($O2^n$) no pior caso
- Espaço? Mantém todos os nós na memória. Essa ineficiência levou ao desenvolvimento do algoritmo IDA*
- Ótima? Sim
- Otimamente eficiente:
  - Nenhum outro algoritmo de busca ótimo tem garantia de expandir um número de nós menor que A*;
  - Isso porque qualquer algoritmo que não expande todos os nós com f(n) < C* corre o risco de omitir uma solução ótima.

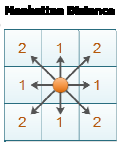
Busca Heurística - Busca com Limite de Memória

Adaptação da técnica de aprofundamento iterativo ao conceito de busca heurística, com a finalidade de reduzir as exigências de memória do A*.

- IDA* (*Iterative Deepening* A*)
  - Semelhante ao aprofundamento iterativo, sua principal diferença é que seu limite é dado pela função de avaliação (f) (contornos), e não pela profundidade (d);
  - Necessita de menos memória do que A*, mas continua ótima.
- RBFS* (*Recursive Best-First-Search*)
  - Limita o best-first-search através da utilização de um espaço linear;
  - Semelhante ao busca em profundidade recursiva;
  - Mantém no nó corrente a melhor alternativa a partir do ancestral do nó corrente.
- SMA* (*Simplified Memory-Bounded* A*)
  - O número de nós guardados em memória é fixado previamente:
    - Conforme vai avançando, descarta os piores ;
    - Mantém no nó corrente a melhor alternativa a partir do ancestral do nó corrente;
    - É completa e ótima se a memória alocada for suficiente;
    - Atividade constante da paginação causando a degradação do desempenho do sistema.

A* - Eficiência

- A eficiência do algoritmo A* depende da escolha da heurística;
- Distância Euclideana:
  - É sempre admissível
- Distância Manhattan:
  - Considera os quarteirões da cidade;
  - Admissível em problemas de mapas de cidades
  - Considere dois pontos de coordenadas:
    p1: (x1, y1) e p2: (x2, y2)
  - A distância Manhattan é dado por:



$$|x_1 - x_2| + |y_1 - y_2|$$

A* - Heurísticas em mapas

- Comparação das distâncias Manhattan e Euclideana

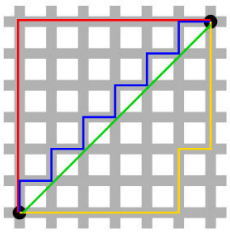


Image Credit: Wikimedia Commons

E num jogo em tempo real ?

- Entre 1990 e 2017 a Nintendo patrocinou uma competição de jogos eletrônicos denominada Nintendo World Championships (ou NWC)
  - Baseia-se em seu cartucho personalizado para o Nintendo Entertainment System, considerado o cartucho de NES mais valioso e mais raro que existe.

- A competição Super Mário;
- Remake do jogo clássico;
- Uma API foi fornecida para controlar Mario;
- O vencedor da competição de 2009 utilizou o algoritmo A*.