Busca

Profª Lucília Ribeiro

Ideias que possibilitam a IA

01.

Busca

Encontrar solução para o problema

02.

Conhecimento

Representar informações e tirar inferências delas

03.

Incerteza

Lidar com eventos incertos usando probabilidade

Ideias que possibilitam a IA

04.

Otimização

Encontrar a melhor maneira de resolver um problema

05.

Aprendizado

Melhorar o desempenho com base no acesso a dados e experiência

06.

Redes Neurais

Estrutura inspirada no cérebro, capaz de executar tarefas de forma eficaz

07.

Linguagem

Processar a linguagem natural que é produzida e compreendida pelos humanos

01.

Busca

Problemas de Busca

Envolvem um **agente** que recebe um **estado inicial** e um **objetivo** e retorna uma **solução** de como passar do primeiro para o último

Quebra cabeça dos 15

Agente

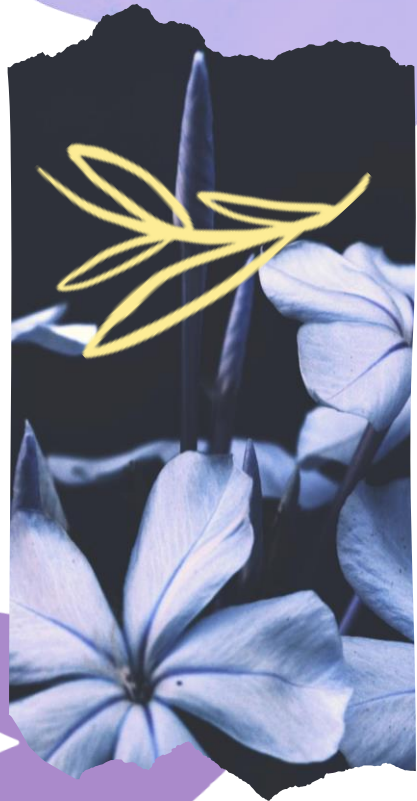
Uma entidade que percebe seu ambiente e age sobre esse ambiente. Em um aplicativo navegador, por exemplo, o agente seria a representação de um carro que precisa decidir quais ações tomar para chegar ao destino.

Estado (estado inicial)

Uma configuração de um agente em seu ambiente. Por exemplo, no quebra-cabeça de 15 , um estado é qualquer maneira em que todos os números estão organizados no tabuleiro.

Ações

- Escolhas que podem ser feitas em um estado.
- Podem ser definidas como uma função.
- Ao receber o estado s como entrada, Actions(s) retorna como saída o conjunto de ações que podem ser executadas no estado s.
- Ex: em um *quebra-cabeça 15* , as ações de um determinado estado são as maneiras pelas quais você pode deslizar os quadrados na configuração atual (4 se o quadrado vazio estiver no meio, 3 se estiver próximo a um lado, 2 se estiver no canto).



Modelo de Transição

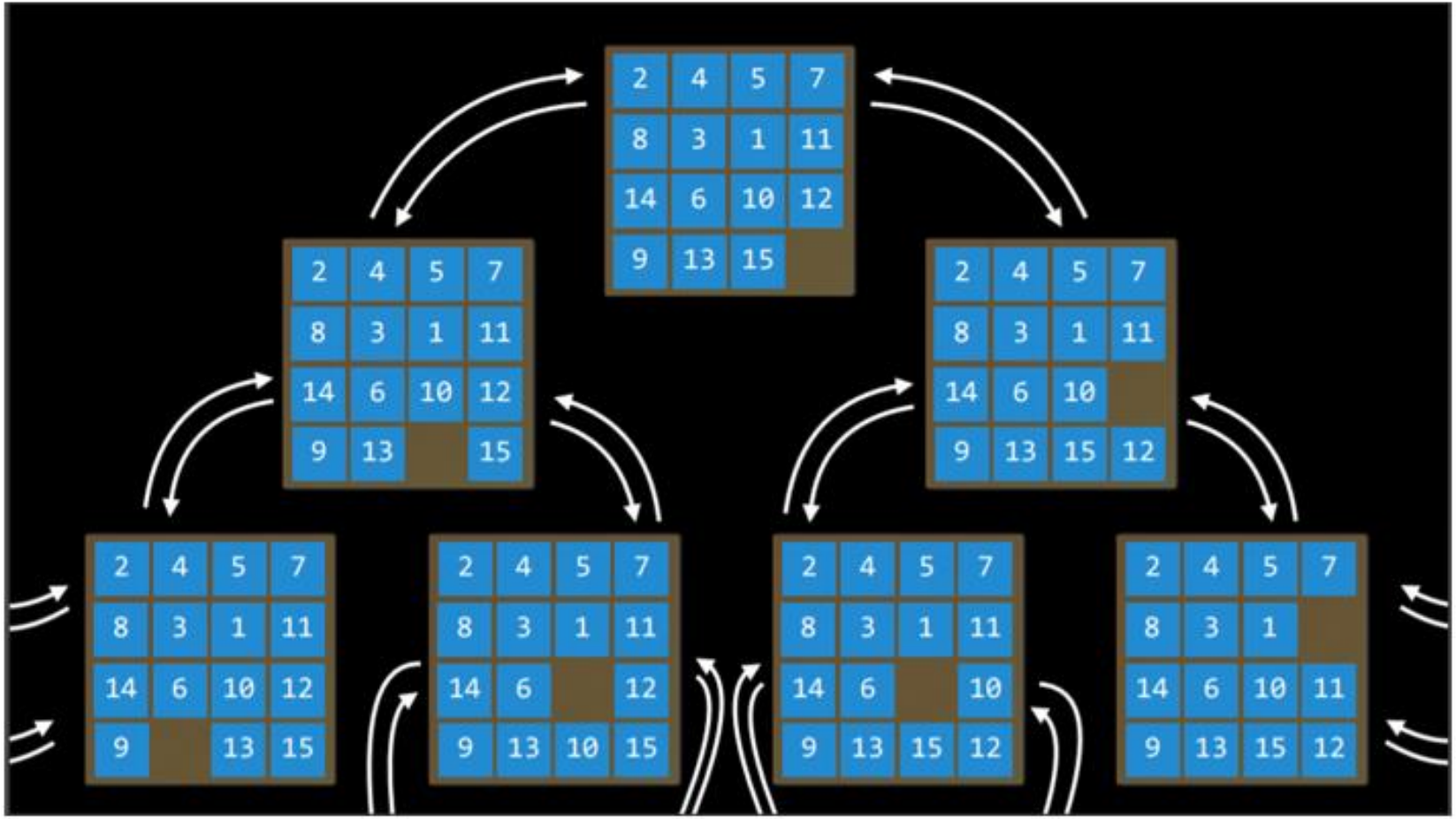
Uma descrição do estado resultante da execução de qualquer ação aplicável em qualquer estado.

Pode ser definido como uma função. Ao receber o estado **`s`** e a ação **`a`** como entrada, **`Results(s, a)`** retorna o estado resultante da execução da ação **`a`** no estado **`s`**.

Por exemplo, dada uma certa configuração de um *quebra-cabeça de 15* (estado s), mover um quadrado em qualquer direção (ação a) levará a uma nova configuração do quebra-cabeça (o novo estado).

Espaço de Estado

- O conjunto de todos os estados alcançáveis a partir do estado inicial por qualquer sequência de ações.
- Ex: no quebra-cabeça 15, o espaço de estados consiste em todas as configurações (16!/2) no tabuleiro que podem ser alcançadas a partir de qualquer estado inicial.
- O espaço de estados pode ser visualizado como um gráfico direcionado com estados, representados como nós, e ações, representadas como setas entre os nós.


2 4 5 7
8 3 1 11
14 6 10 12
9 13 15
2 4 5 7
8 3 1 11
14 6 10 12
9 13 15
2 4 5 7
8 3 1 11
14 6 10
9 13 15 12
2 4 5 7
8 3 1 11
14 6 10 12
9 13 15
2 4 5 7
8 3 1 11
14 6 12
9 13 10 15
2 4 5 7
8 3 1 11
14 6 10
9 13 15 12
2 4 5 7
8 3 1
14 6 10 11
9 13 15 12

Teste de Meta (objetivo)

A condição que determina se um determinado estado é um estado objetivo.

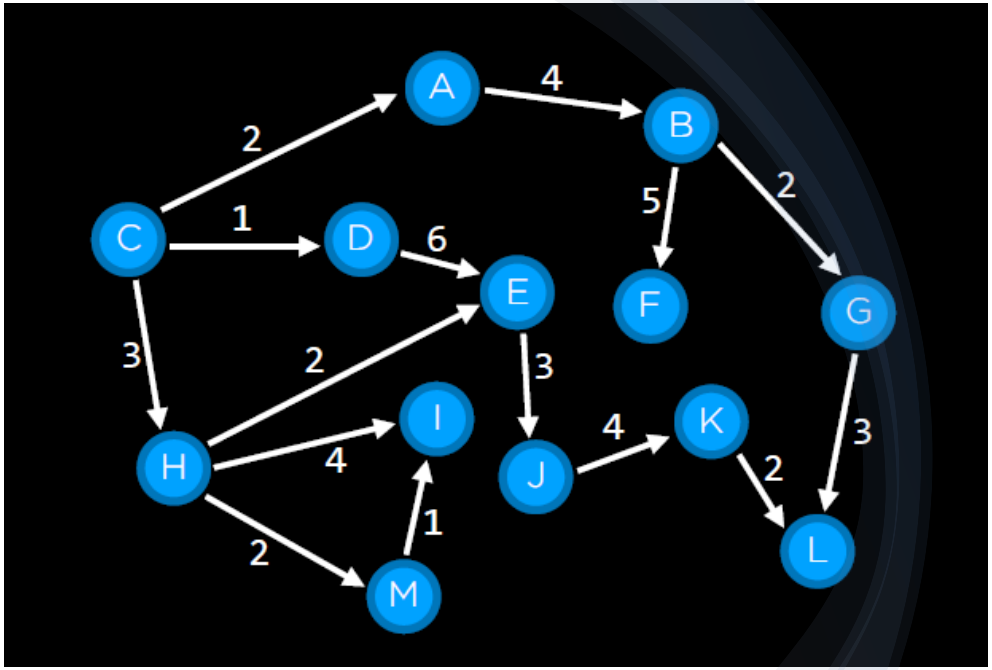
Por exemplo, em um aplicativo de navegação, o teste de objetivo seria se a localização atual do agente (a representação do carro) está no destino. Se for – problema resolvido. Se não estiver, continuamos pesquisando.

Custo do Caminho

Um custo numérico associado a um determinado caminho.

Por exemplo, um aplicativo de navegação não leva você simplesmente ao seu objetivo; isso é feito ao mesmo tempo em que minimiza o custo do caminho, encontrando o caminho mais rápido possível para você chegar ao seu estado objetivo.

Custo do Caminho



Número que nos diz o quão caro é escolher esta opção específica

02.

Resolvendo Problemas de Busca

Solução

Conjunto de ações que leva do estado inicial ao estado objetivo

Solução Ideal: solução que possui o menor custo de caminho entre todas as soluções

Nós (Nodos)

Num processo de pesquisa, os dados são frequentemente armazenados num ***nó*** , uma estrutura de dados que contém os seguintes dados:

- Um ***Estado***
- Seu ***nó pai*** , por meio do qual o nó atual foi gerado
- A ***ação*** que foi aplicada ao estado do pai para chegar ao nó atual
- O ***custo do caminho*** do estado inicial até este nó

Nós

- Contêm informações que os tornam muito úteis para fins de algoritmos de busca.
- Eles contêm um ***estado*** , que pode ser verificado usando o ***teste de meta*** para ver se é o **estado final**. Se for, o ***custo do caminho*** do nó pode ser comparado com *os custos do caminho* de outros nós , o que permite escolher a ***solução ideal*** .
- Uma vez escolhido o nó, em virtude de armazenar o *nó pai* e a *ação* que levou do *nó pai* ao nó atual, é possível rastrear cada passo do caminho desde o *estado inicial* até este nó, e esta sequência de ações é a **solução** .

Fronteira

Mecanismo que “gerencia” os *nós* . A *fronteira* começa contendo um estado inicial e um conjunto vazio de itens explorados, e então repete as seguintes ações até que uma solução seja alcançada:

Repita:

1.Se a fronteira estiver vazia,

- *Parar.* Não há solução para o problema.

2.Remova um nó da fronteira. Este é o nó que será considerado.

3.Se o nó contiver o estado objetivo,

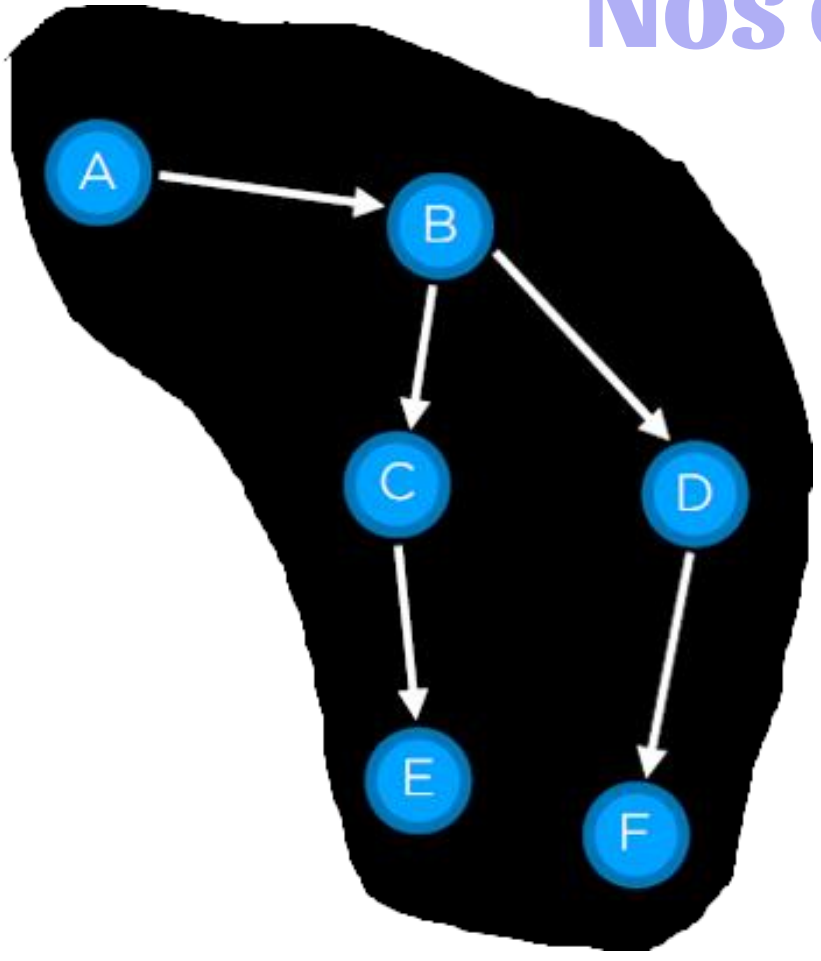
- Devolva a solução. *Parar* .

Senão,

4. Expanda o nó (encontre todos os novos nós que podem ser alcançados a partir deste nó) e adicione os nós resultantes à fronteira.

5. Adicione o nó atual ao conjunto explorado.

Nós explorados



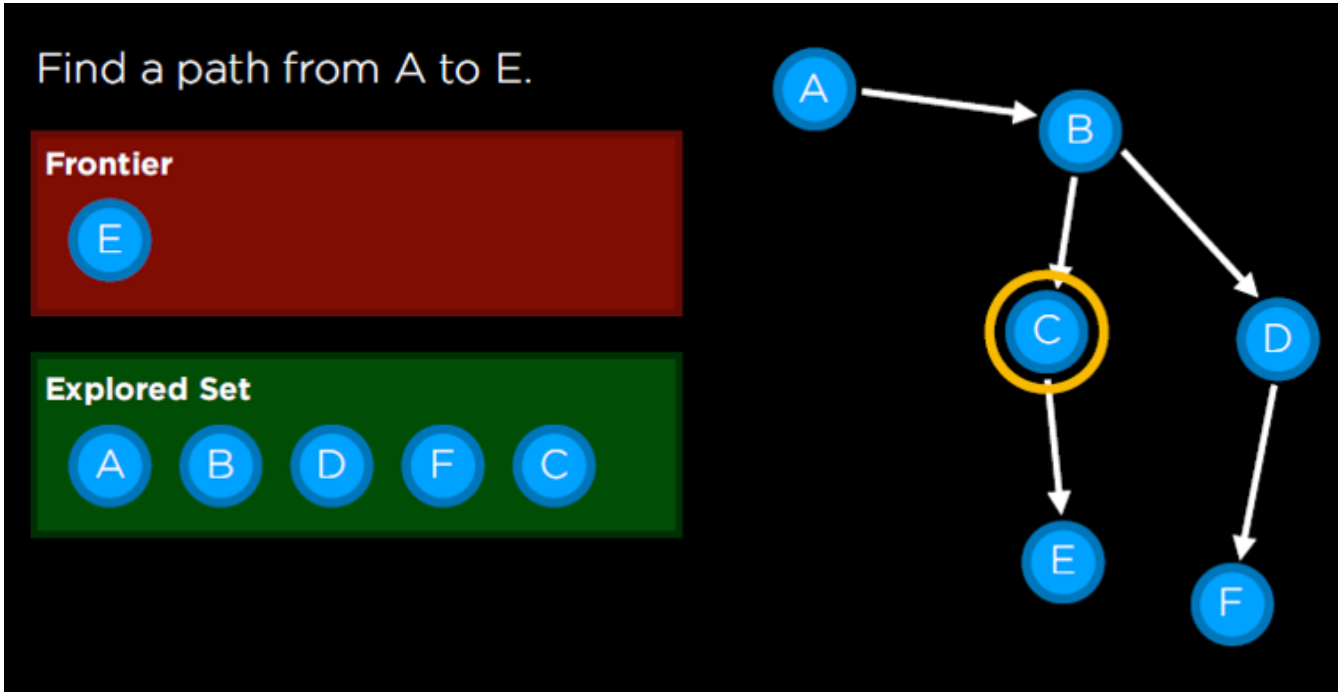
03.

Busca por Profundidade

PILHA (LIFO – Last In, First Out)

DFS (Busca em profundidade)

- Prós:
  - Na melhor das hipóteses, esse algoritmo é o mais rápido. Se tiver “sorte” e sempre escolher o caminho certo para a solução (por acaso), então a pesquisa *em profundidade* leva o menor tempo possível para chegar a uma solução.
- Contras:
  - É possível que a solução encontrada não seja a ideal.
  - Na pior das hipóteses, este algoritmo explorará todos os caminhos possíveis antes de encontrar a solução, demorando assim o maior tempo possível antes de chegar à solução.



Exploramos sempre o nó mais profundo

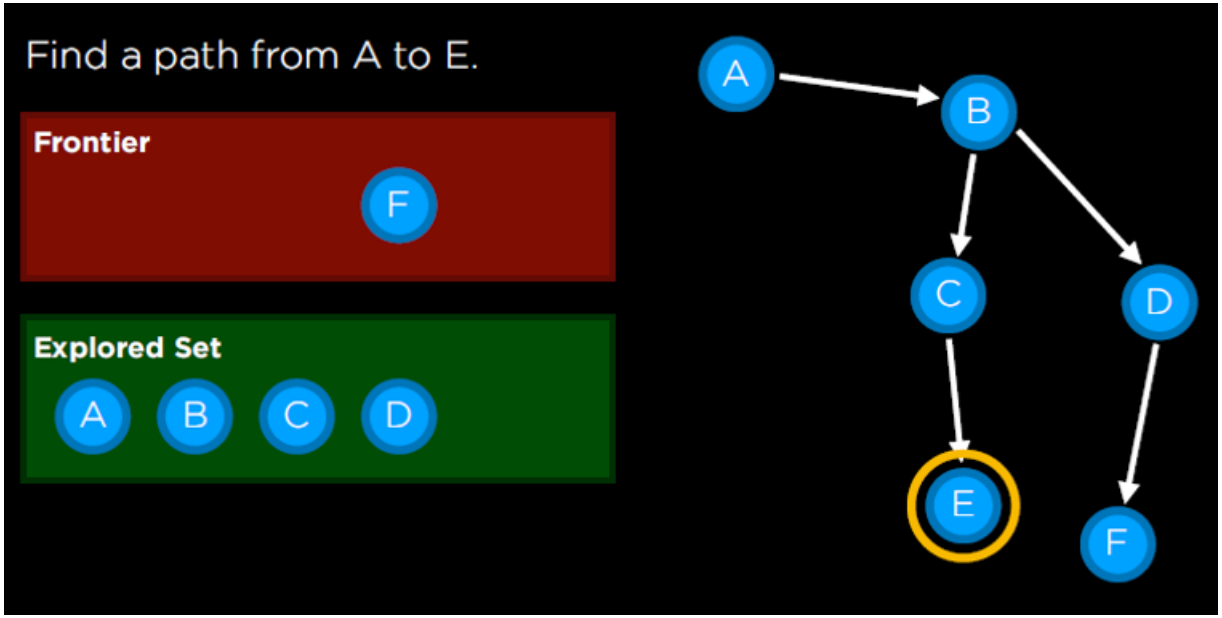
04.

Busca em Largura

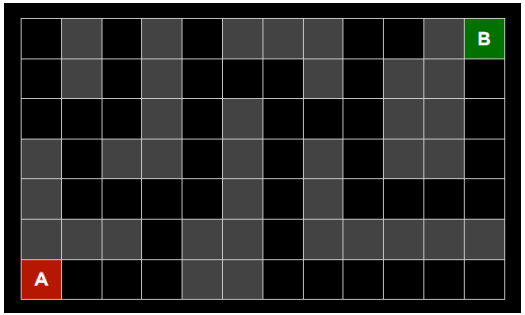
FILA (FIFO – First In, First Out)

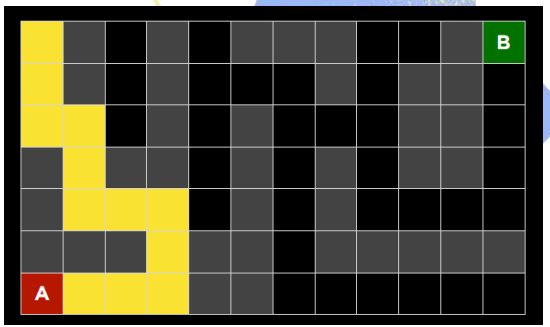
BFS (Busca em largura)

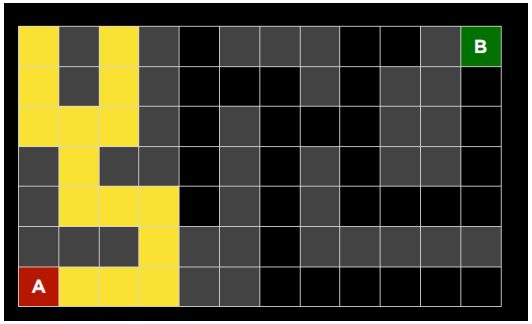
- Prós:
  - Este algoritmo tem garantia de encontrar a solução ótima.
- Contras:
  - É quase certo que esse algoritmo levará mais tempo do que o tempo mínimo para ser executado.
  - Na pior das hipóteses, esse algoritmo leva o maior tempo possível para ser executado.

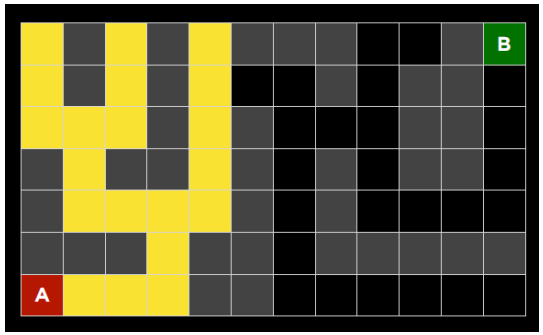


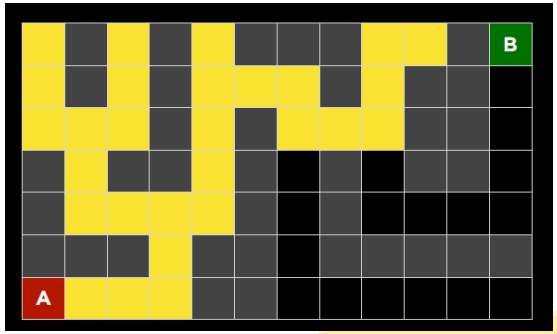
Exploramos sempre o nó mais raso da fronteira

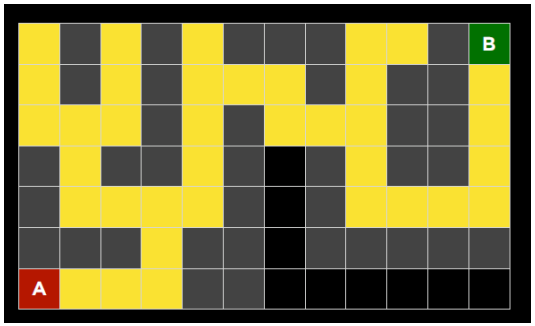












Busca em Profundidade

Busca por Largura

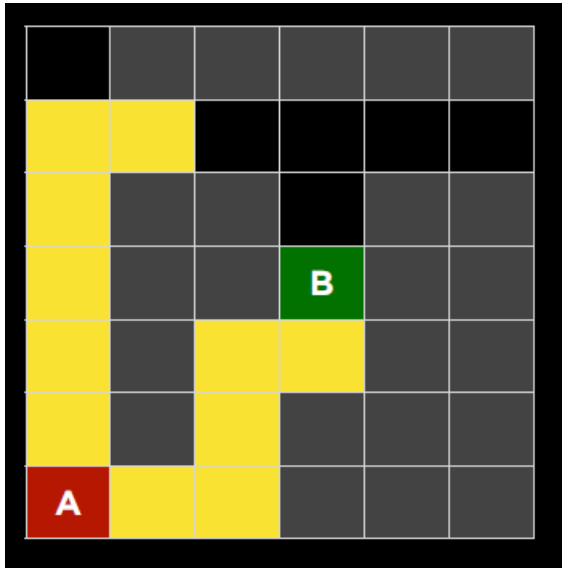


Figura 3.8 Espaço de estados do quebra-cabeça dos 8 gerado por operações do tipo "mova o vazio".

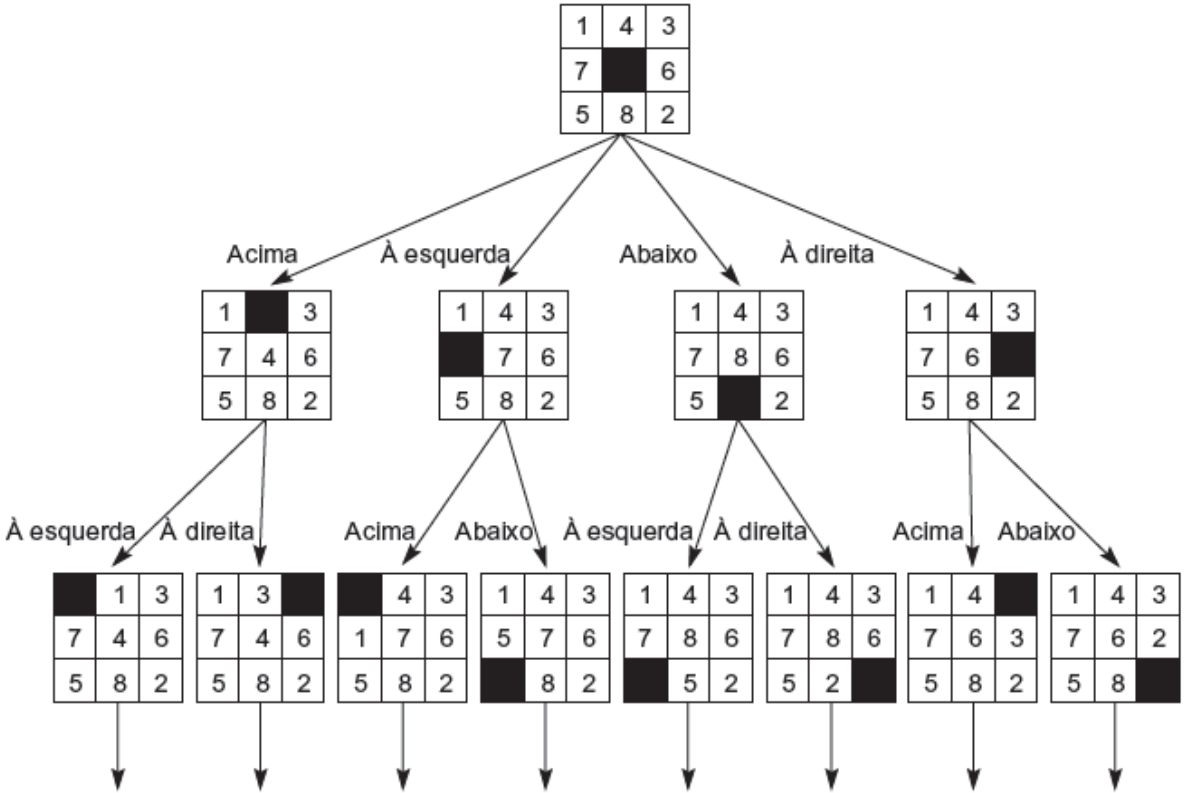


Figura 3.9 Um exemplo do problema do caixeiro-viajante.

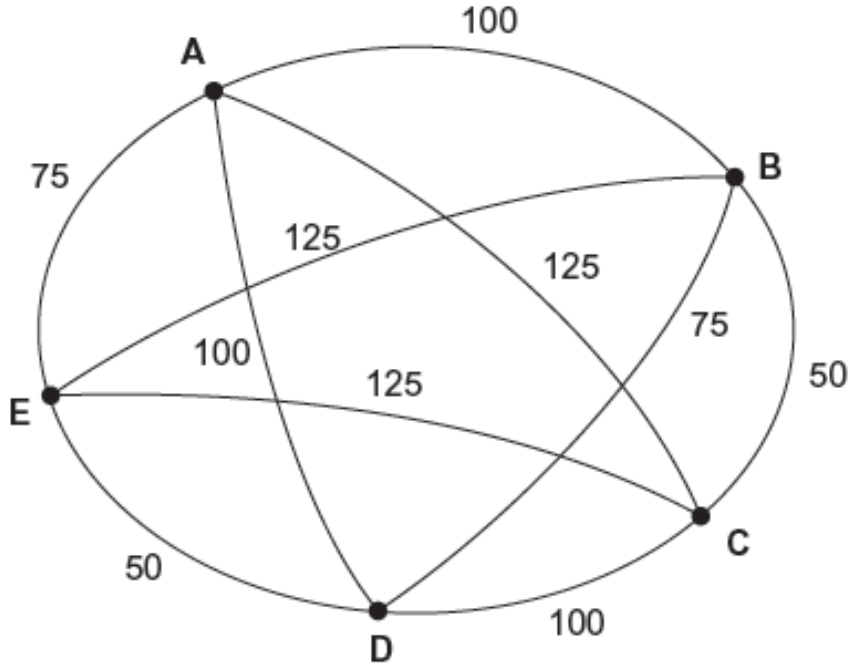


Figura 3.10 Busca para o problema do caixeiro-viajante. Cada arco é rotulado com o peso total de todos os caminhos desde o nó inicial (A) até o ponto final.

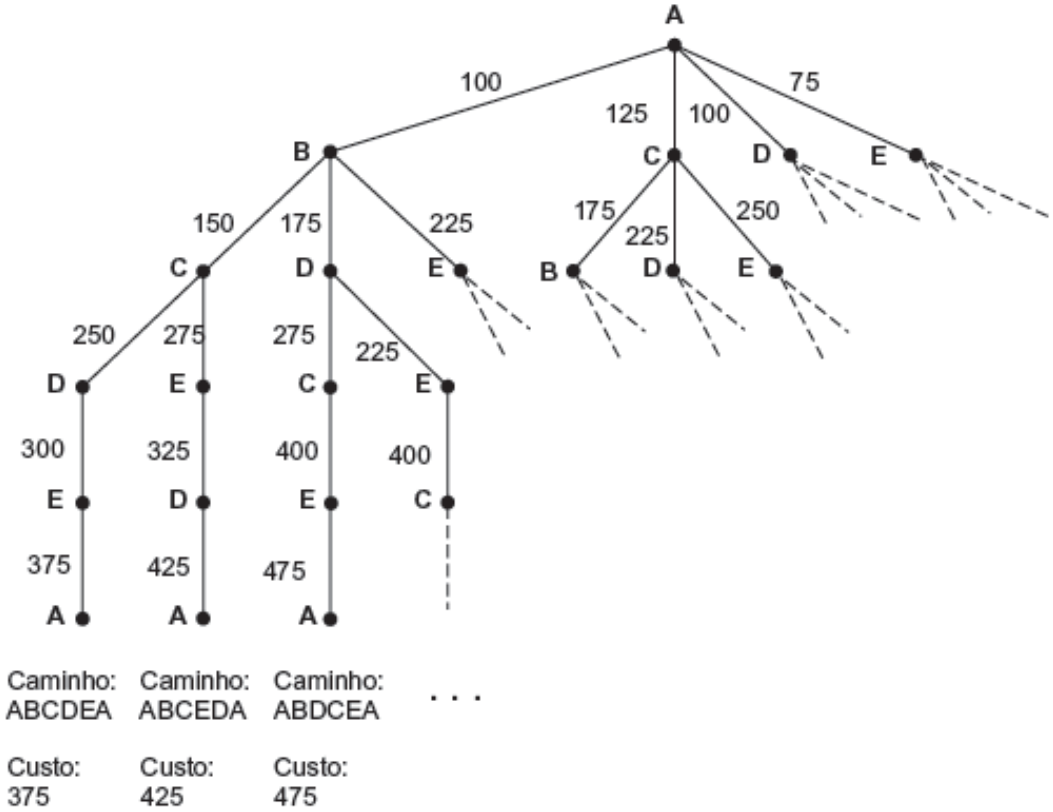
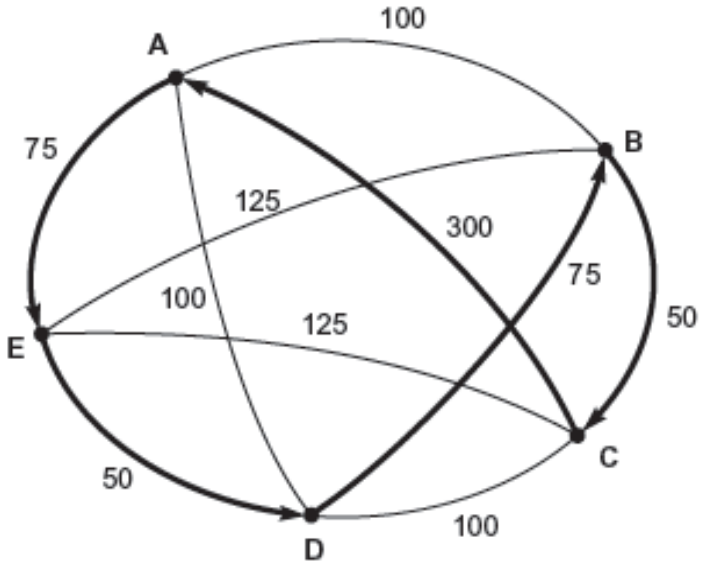


Figura 3.11 Um exemplo de problema do caixeiro-viajante com o caminho do vizinho mais próximo indicado pelas setas mais grossas. Note que esse caminho (A, E, D, B, C, A), com custo de 550, não é o caminho mais curto. O alto custo relativo do arco (C, A) é responsável pelo insucesso da heurística.



A busca em profundidade examina os estados na ordem A, B, E, K, S, L, T, F, M, C, G, N, H, O, P, U, D, I, Q, J, R.

A busca em amplitude, por outro lado, explora o espaço nível por nível. Apenas quando não houver mais estados a serem explorados em um determinado nível é que o algoritmo se moverá para o próximo mais profundo.

Uma busca em amplitude do grafo considera os estados na ordem A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U.

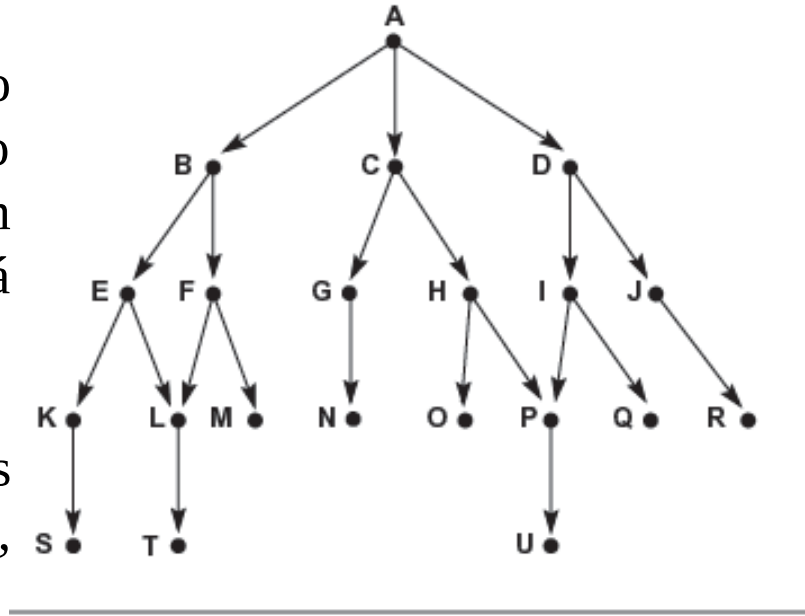


Figura 3.19 Busca em profundidade para o quebra-cabeça dos 8 com uma profundidade limitada a 5.

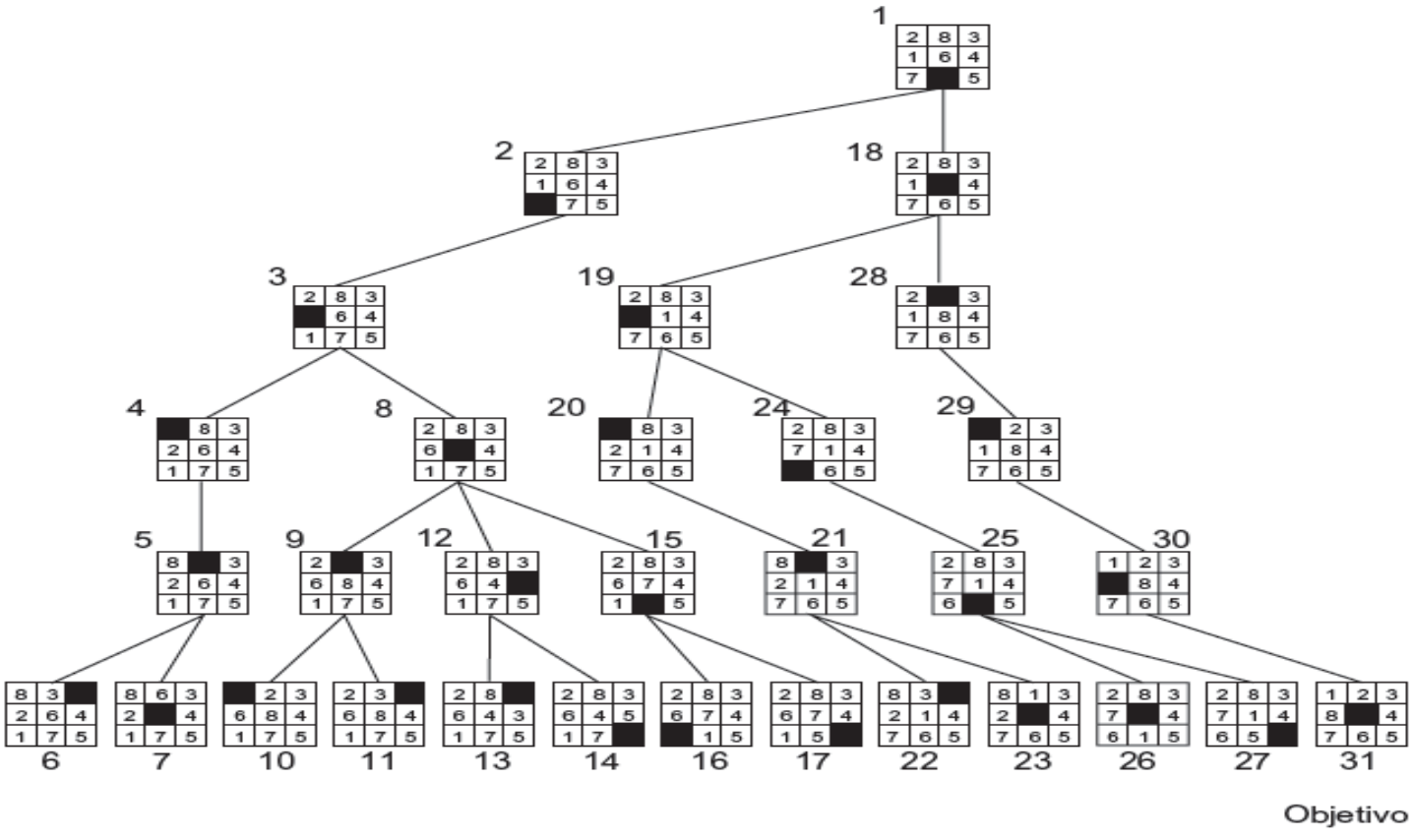
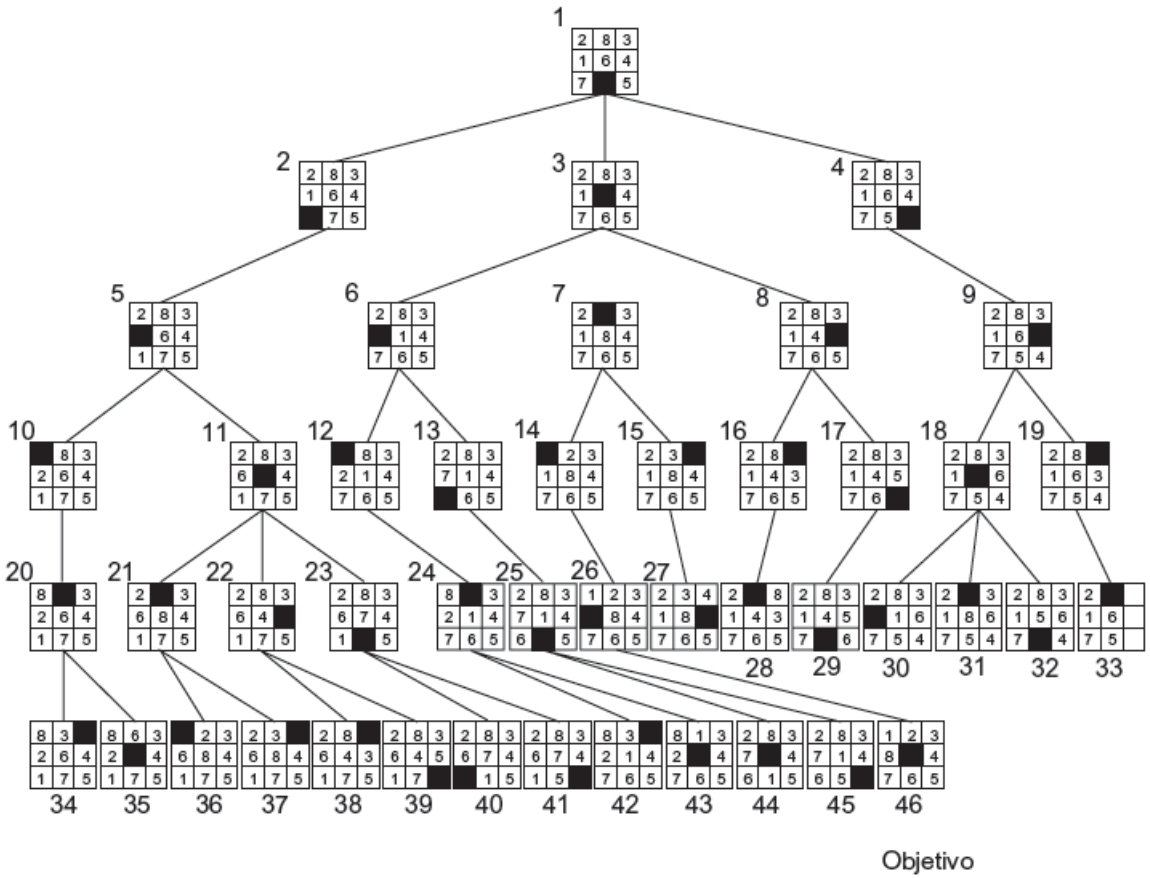


Figura 3.17 Busca em amplitude do quebra-cabeça dos 8 mostrando a ordem em que os estados foram removidos de abertos.



05.

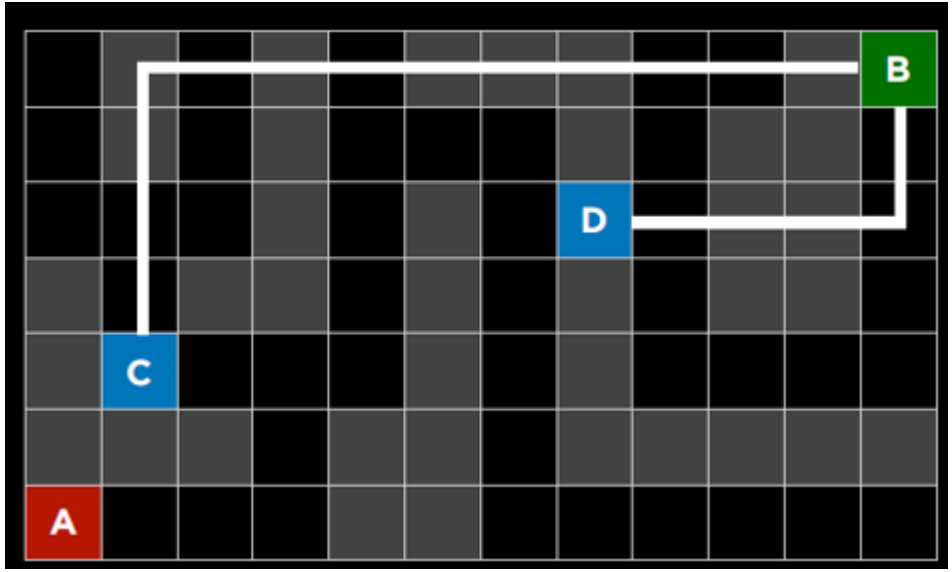
Busca informada

Busca Informada

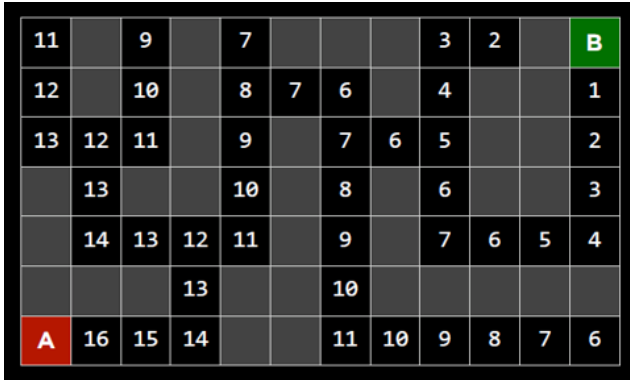
Estratégias de busca que utilizam conhecimentos específicos do problema para melhor encontrar uma solução

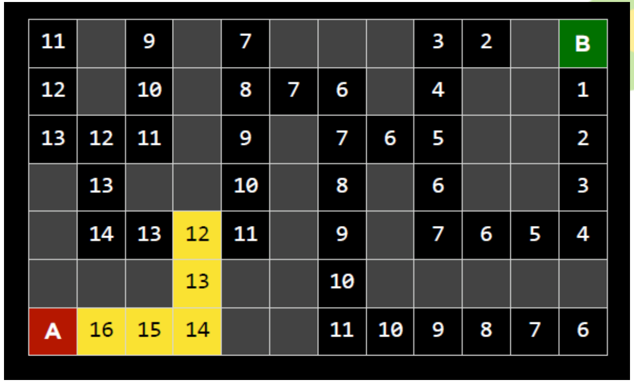
função heurística, convencionalmente chamada h(n), que pega um estado de entrada e retorna nossa estimativa de quão próximos estamos do objetivo (A qualidade da heurística afetará a qualidade do algoritmo)

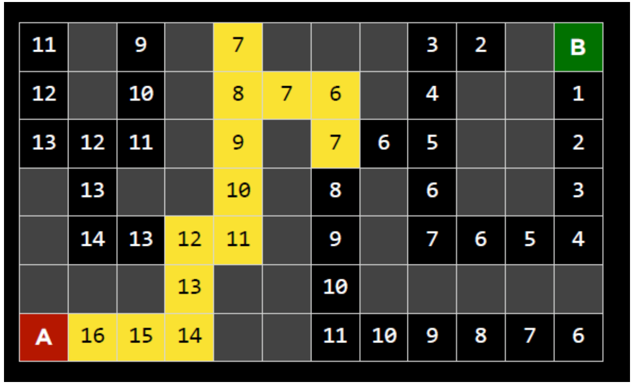
Distância de Manhattan

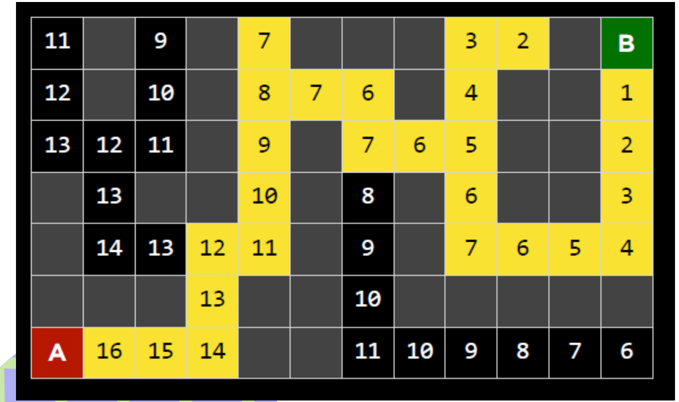


Busca Gulosa

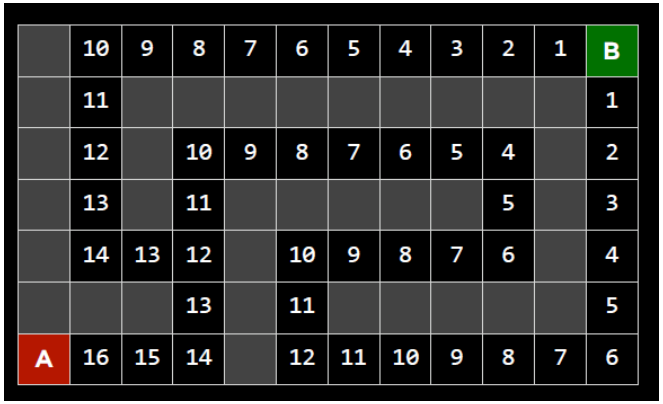


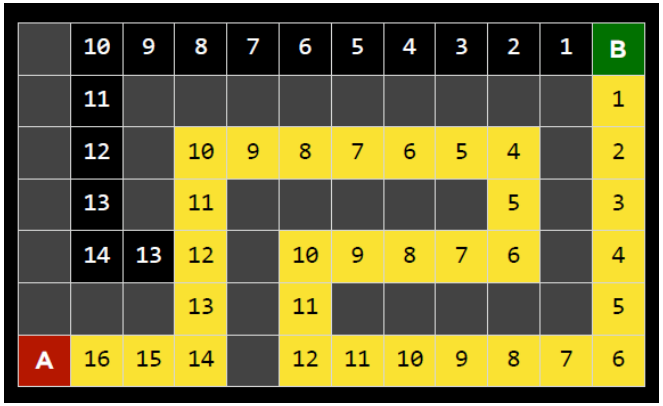




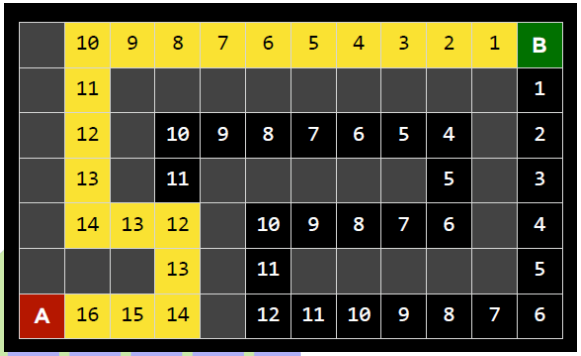


Busca Gulosa





- Este algoritmo não tem garantia de encontrar a solução ótima.



Busca A*

Considera não apenas h(n) , o custo estimado da localização atual até a meta, mas também g(n) , o custo que foi acumulado até a localização atual.

Ao combinar esses dois valores, o algoritmo tem uma maneira mais precisa de determinar o custo da solução e otimizar suas escolhas em movimento.

O algoritmo acompanha (custo do caminho até agora + custo estimado para a meta)

A*

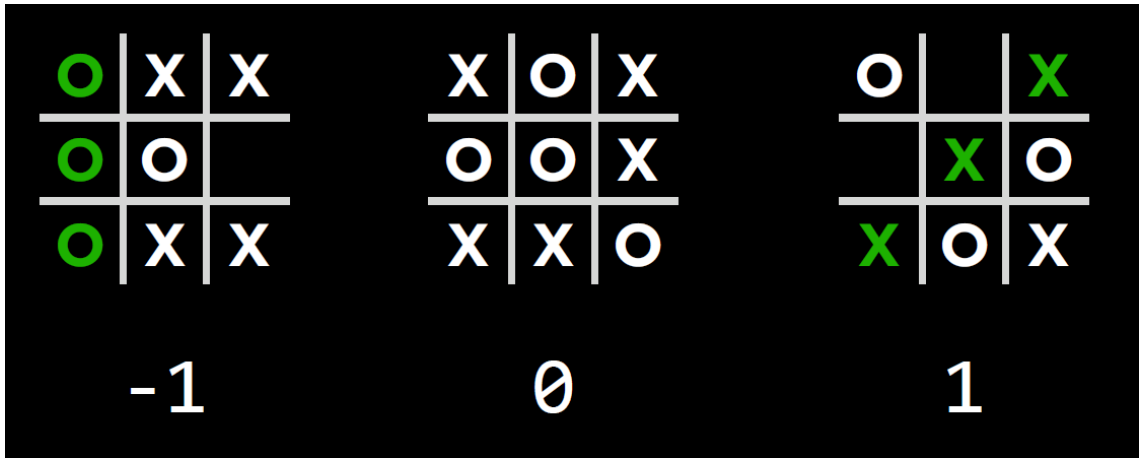
	10	9	8	7	6	5	4	3	2	1	B
	11										1
	12		7+10	8+9	9+8	10+7	11+6	12+5	13+4		2
	13		6+11						14+5		3
	14	6+13	5+12		10	9	8	7	6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

	11+10	12+9	13+8	14+7	15+6	16+5	17+4	18+3	19+2	20+1	B
	10+11										1
	9+12		7+10	8+9	9+8	10+7	11+6	12+5	13+4		2
	8+13		6+11						14+5		3
	7+14	6+13	5+12		10	9	8	7	15+6		4
			4+13		11						5
A	1+16	2+15	3+14		12	11	10	9	8	7	6

06.

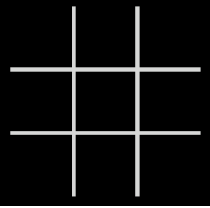
Busca adversária Algoritmo MinMax

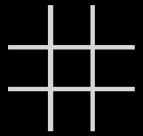
Função Utilidade

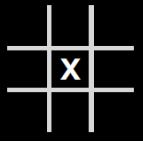


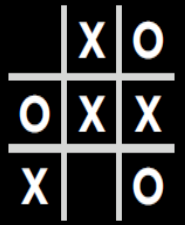
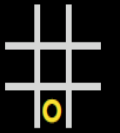
- S_0 : Estado inicial (no nosso caso, um tabuleiro 3X3 vazio)
- *Jogador(es)* : função que, dado um estado *s* , retorna qual é a vez do jogador (X ou O).
- *Action(s)* : uma função que, dado um estado *s* , retorna todos os movimentos legais neste estado (quais posições estão livres no tabuleiro).
- *Resultado(s, a)* : uma função que, dado um estado *s* e uma ação *a* , retorna um novo estado. Este é o tabuleiro que resultou da realização da ação *a* no estado *s* (fazer uma jogada no jogo).
- *Terminal(s)* : função que verifica se esta é a última etapa do jogo, ou seja, se alguém ganhou ou se houve empate. Retorna *True* se o jogo terminou, *False* caso contrário.
- *Utilidade(s)* : uma função que, dado um estado terminal *s* , retorna o valor de utilidade do estado: -1, 0 ou 1.

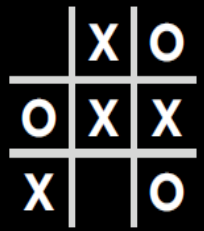
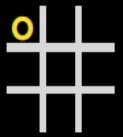
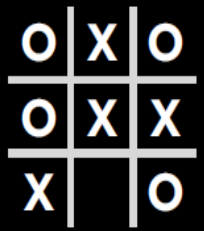
Initial State

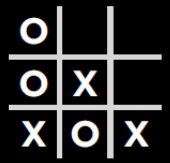


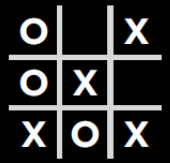
PLAYER() = X

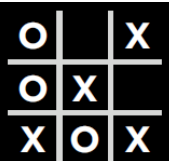
PLAYER() = O

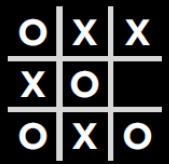
ACTIONS() = {  ,  }

RESULT( , ) = 

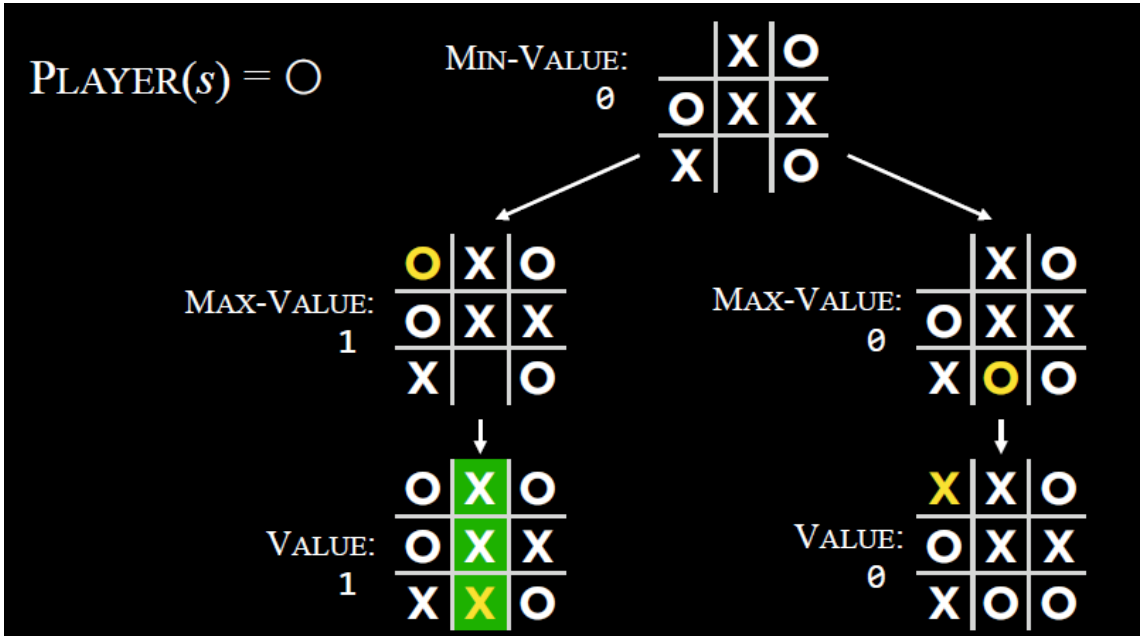
TERMINAL() = false

TERMINAL() = true

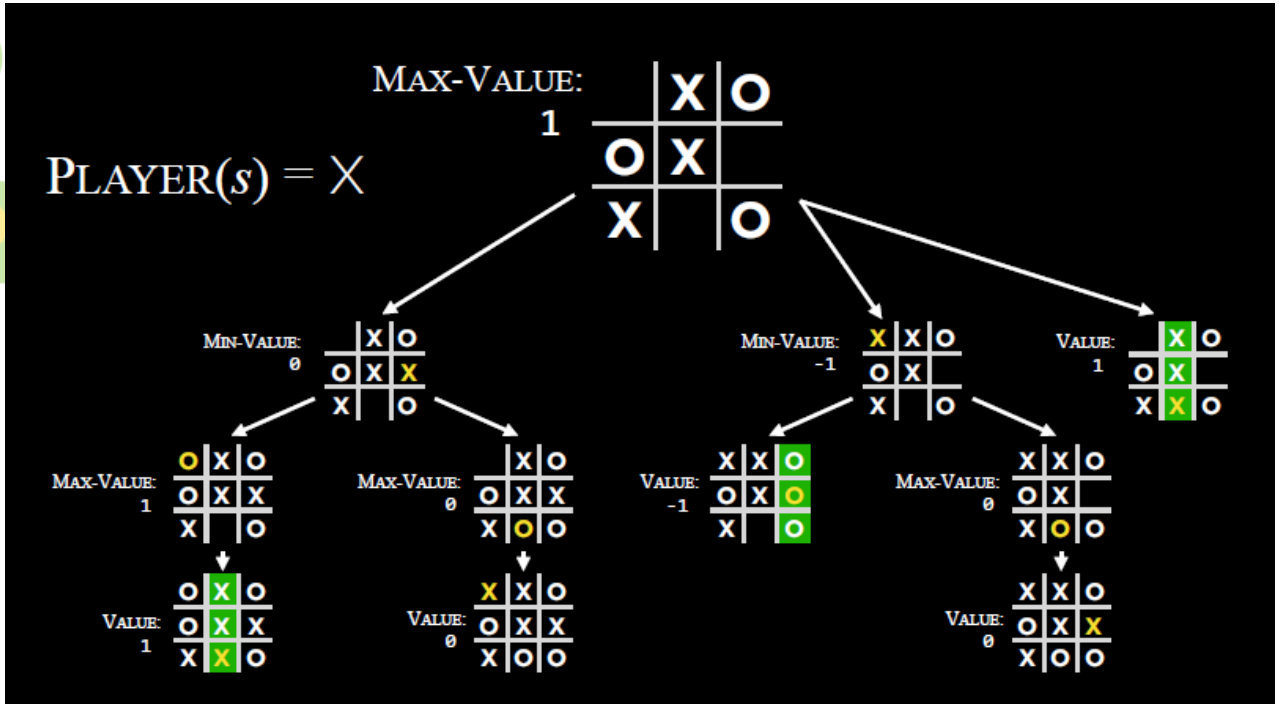
UTILITY() = 1

UTILITY() = -1

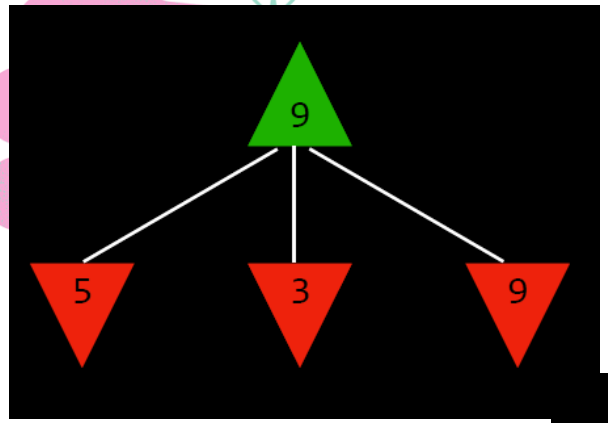
MiniMax

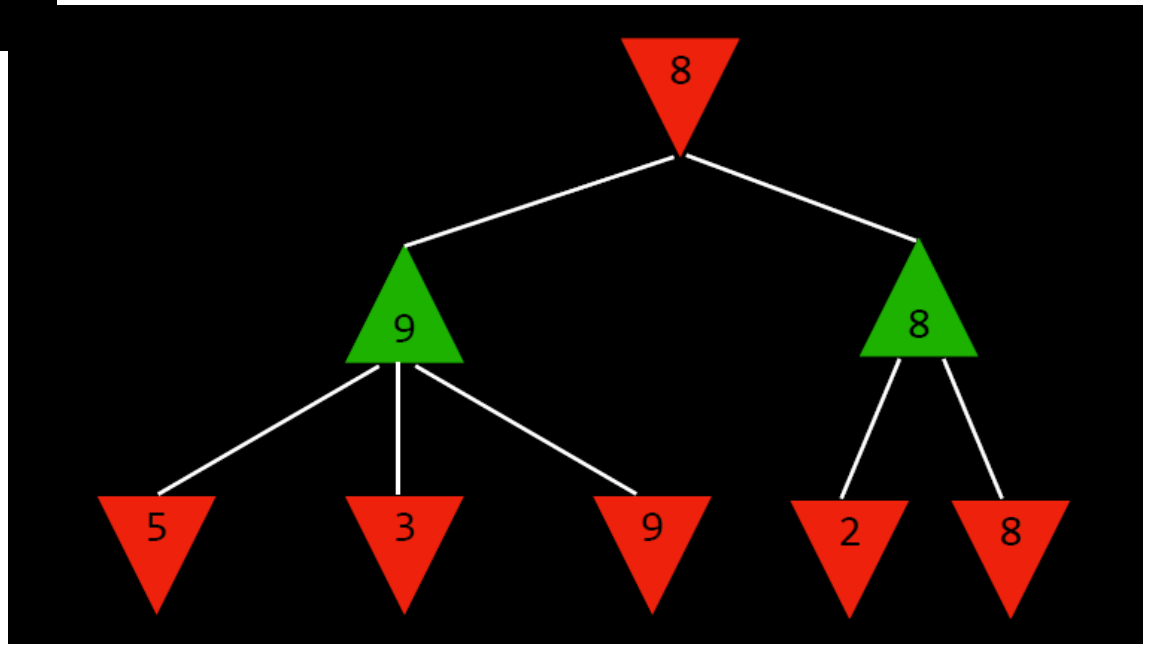


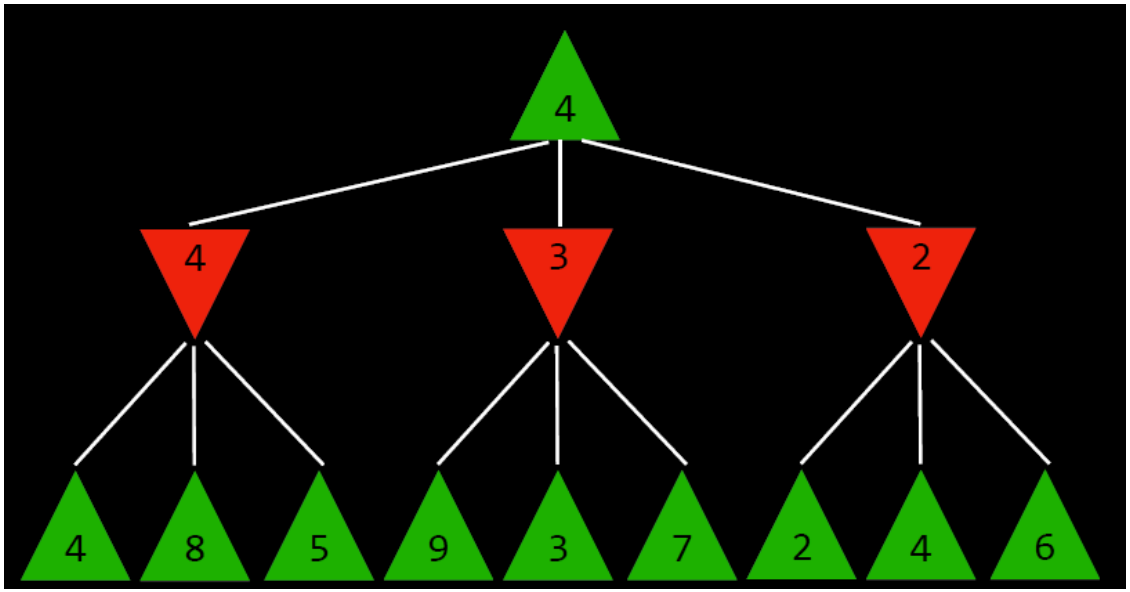
MiniMax


MAX-VALUE:
1
PLAYER(s) = X
MIN-VALUE:
0
MIN-VALUE:
-1
VALUE:
1
MAX-VALUE:
1
MAX-VALUE:
0
VALUE:
-1
MAX-VALUE:
0
VALUE:
1
VALUE:
0
VALUE:
0

MiniMax






4
4
3
2
4
8
5
9
3
7
2
4
6