Inteligência Artificial

Aula 04- Solução de Problemas [1]

Sílvia M.W. Moraes

Faculdade de Informática - PUCRS

March 20, 2018

[1] Este material não pode ser reproduzido ou utilizado de forma parcial sem a permissão dos autores.

Sinopse

- Nesta aula, apresentamos uma **introdução a solução de problemas por busca**.
- Este material foi construído com base nos capítulos:
  - 3 do livro Artificial Intelligence – a Modern Approach de Russel & Norvig
  - 3 do livro Inteligência Artificial de Luger.

Sumário

Aula anterior

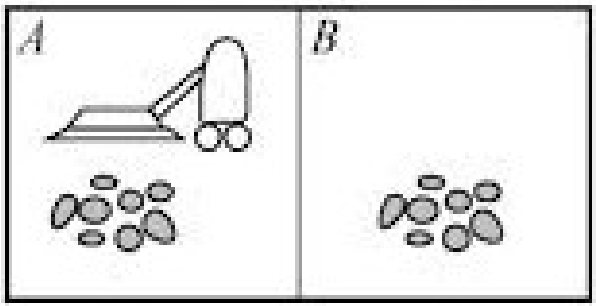
- Introdução a Agentes
  - Conceito de Agente
  - Tipos clássicos: Reativos e Cognitivos

Abstração

- A tarefa de um agente baseado em objetivo **é descobrir a sequência de ações que o levará a solução do problema que é seu objetivo.**
- O primeiro passo no caminho que leva à solução é a definição de **uma abstração de mundo que capture apenas os elementos essenciais do problema.**
- Vamos considerar inicialmente alguns **miniproblemas cujos ambientes são completamente observáveis, discretos e determinísticos.**

Abstração

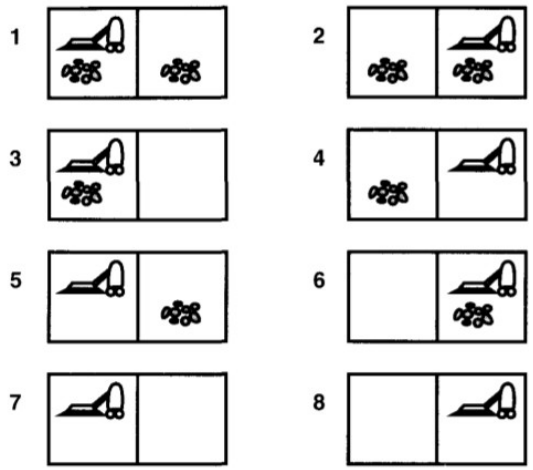
- No caso do agente-exemplo **Aspirador de pó**, apresentado na aula passada, que é capaz de
  - **Perceber** em que **local** está e se há **sujeira** nesse local;
  - mover-se para **esquerda** ou **direita**; **aspirar** sujeira ou não fazer **nada**.



- é suficiente criar uma representação que indique a **posição** atual do agente no ambiente (A ou B) e o **estado** (sujo ou limpo) de cada posição.
  - **[posicao, estadoDeA, estadoDeB]**

Estados do Mundo

- Com essa abstração conseguimos representar todos os 8 estados possíveis desse ambiente:



- **[posicao, estadoDeA, estadoDeB]**
  - [A, sujo, sujo]; [B, sujo, sujo];
  - [A, sujo, limpo]; [B, sujo, limpo];
  - [A,limpo, sujo]; [B,limpo, sujo];
  - [A,limpo,limpo]; [B,limpo,limpo],

Estados do Mundo

- No ambiente do **Aspirador de pó** é apenas **por meio das ações do agente que o estado do mundo pode se modificar**.
  - Exemplos:

Estado Atual	**Ação**	**Novo Estado**
[A,limpo,limpo]	direita	[B,limpo,limpo]
[A,sujo,limpo]	aspirar	[A,limpo,limpo]
[B,limpo,sujo]	esquerda	[A,limpo,sujo]

Definição de Problema

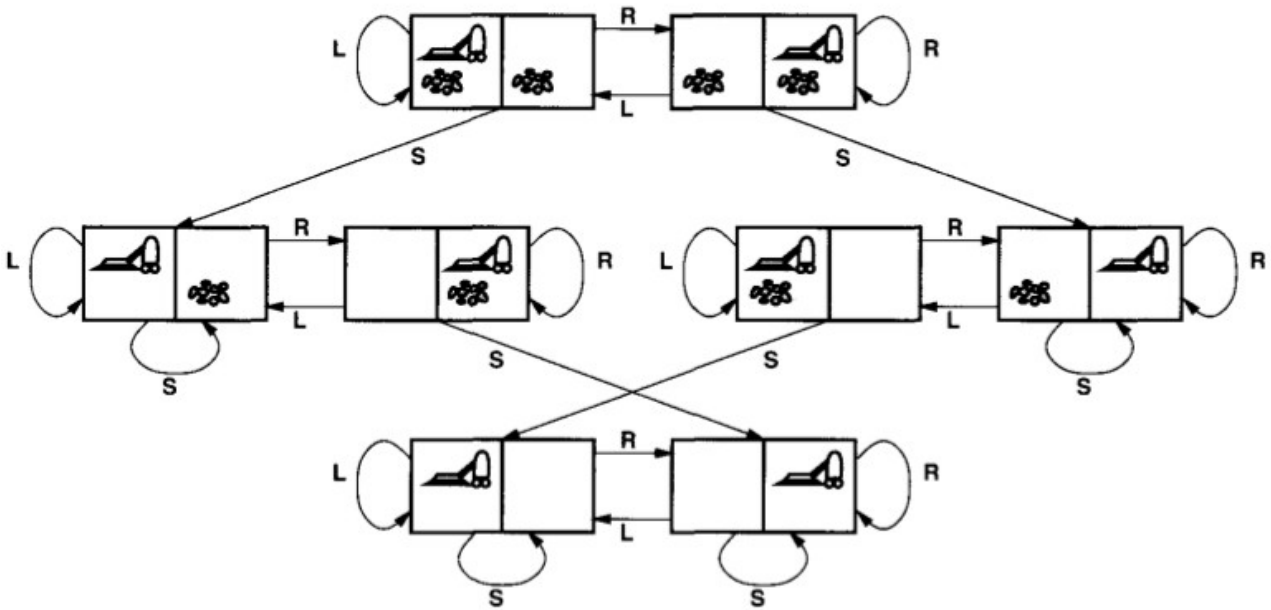
- Um problema pode ser definido formalmente por quatro componentes:
  - **estado inicial**: estado de mundo em que o agente começa a sua execução.
  - **descrição das ações**: define as ações possiveis do agente.
    - Em geral, usa uma **função sucessor** para gerar os estados válidos.
    - Dado um estado x, a função sucessor(x) retorna o conjunto de pares <ação, estado sucessor de x>
  - **teste de objetivo**: usada para verificar se um dado estado é o estado objetivo.
  - **custo do caminho**: define o custo numérico a cada caminho que leva o estado inicial ao estado objetivo.

Definição de Problema

- Exemplo **Aspirador de pó**
  - **estado inicial**: pode ser qualquer um, tal como **[A,sujo,sujo]**.
  - **descrição das ações:** aspirar, direita e esquerda.
    - sucessor([A,sujo,sujo]) =
      { (aspirar, [A,limpo,sujo]),
      (esquerda, [A,sujo,sujo]),
      (direita,[B,sujo,sujo]) }
  - **teste de objetivo**: verifica se todos os locais estão limpos, ou seja, se atingiu um dos estados [A,limpo,limpo] ou [B,limpo,limpo].
  - **custo do caminho**: cada passo custa 1, logo é o número de passos do caminho.

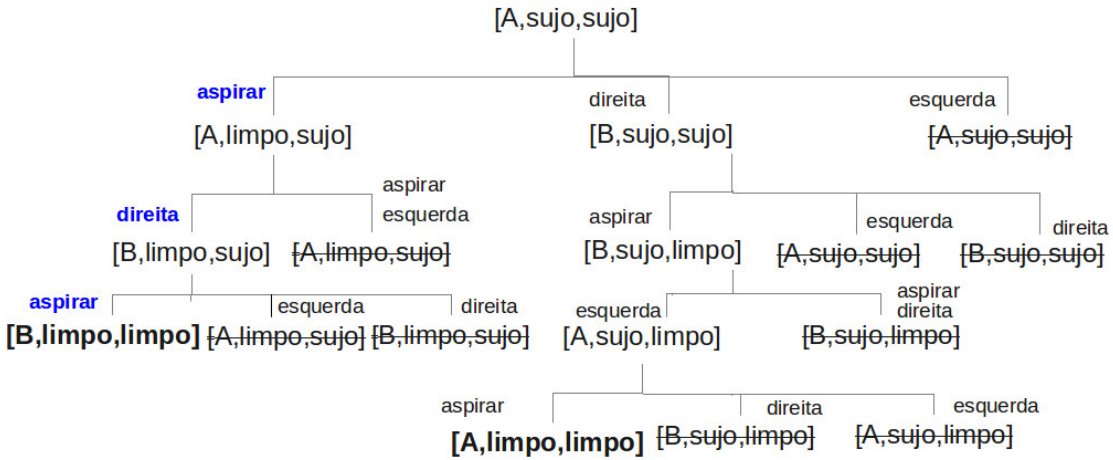
Espaço de Estados

- **Espaço de estados completo** para o Aspirador de Pó é representado por um **grafo**.
  - Os nodos são os estados
  - Os arcos identificam as ações:
    L = esquerda, R = direita e S = aspirar



Espaço de Estados

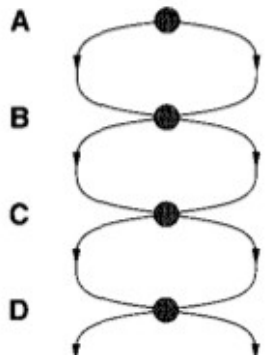
- A tarefa do agente **é dentro do espaço de busca encontrar a sequência de ações que o leva do estado inicial ao estado objetivo**.
  - Exemplo - Considerando **estado inicial: [A,sujo, sujo]** e o **estado objetivo: [_,limpo,limpo]**

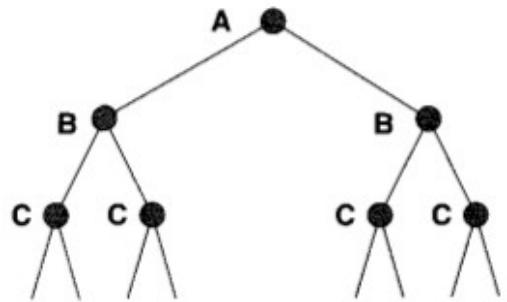


- Para isso, são usados **algoritmos de busca**.
  - [aspirar, direita, aspirar] tem custo 3.
  - [direita, aspirar, esquerda, aspirar] tem custo 4.

Espaço de Estados

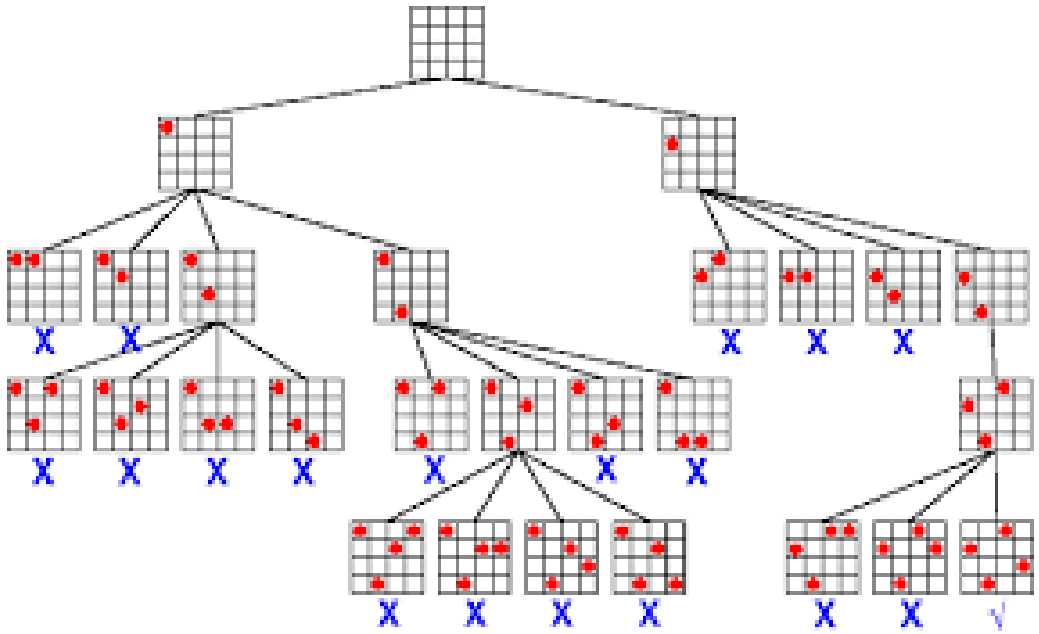
- A tarefa de um agente baseado em objetivo **é descobrir a sequência de ações que o levará a solução do problema que é seu objetivo.**
  - Para isso **usamos algoritmos de busca**.
- Durante o **processo de busca existe a possibilidade de disperçarmos tempo expandindo espaços que já foram encontrados** e expandidos antes.





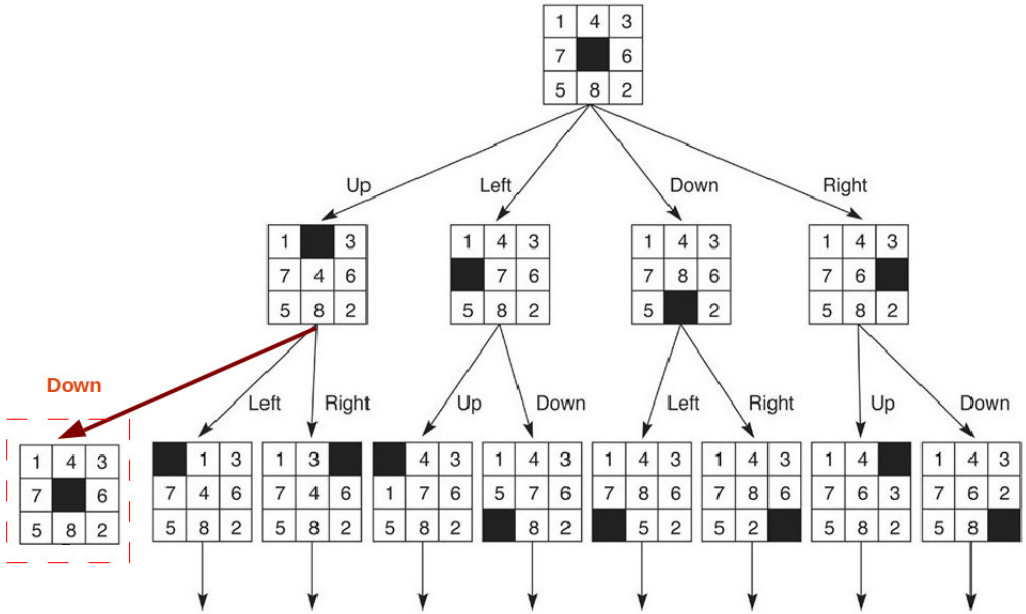
Espaço de Estados

- **Para alguns problemas, essa possibilidade nunca surge.**
  - O **espaço de estados é uma árvore e só existe um caminho até cada estado**. Ex: Problema das **n** rainhas.



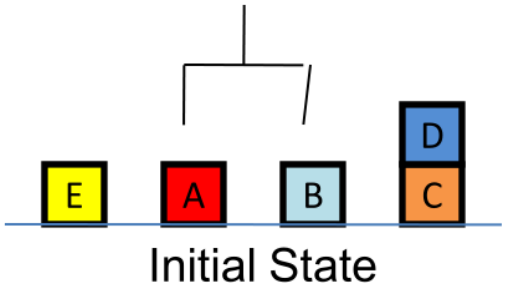
Espaço de Estados

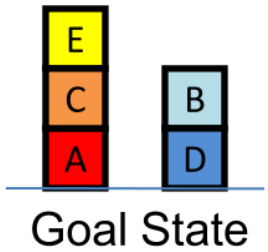
- **Para outros, os estados repetidos são inevitáveis**.
  - Inclui problemas de localização de rotas ou quebra-cabeças deslizantes.
  - **As árvores de busca para esses problemas são infinitas** (presença de ciclos).



Espaço de Estados

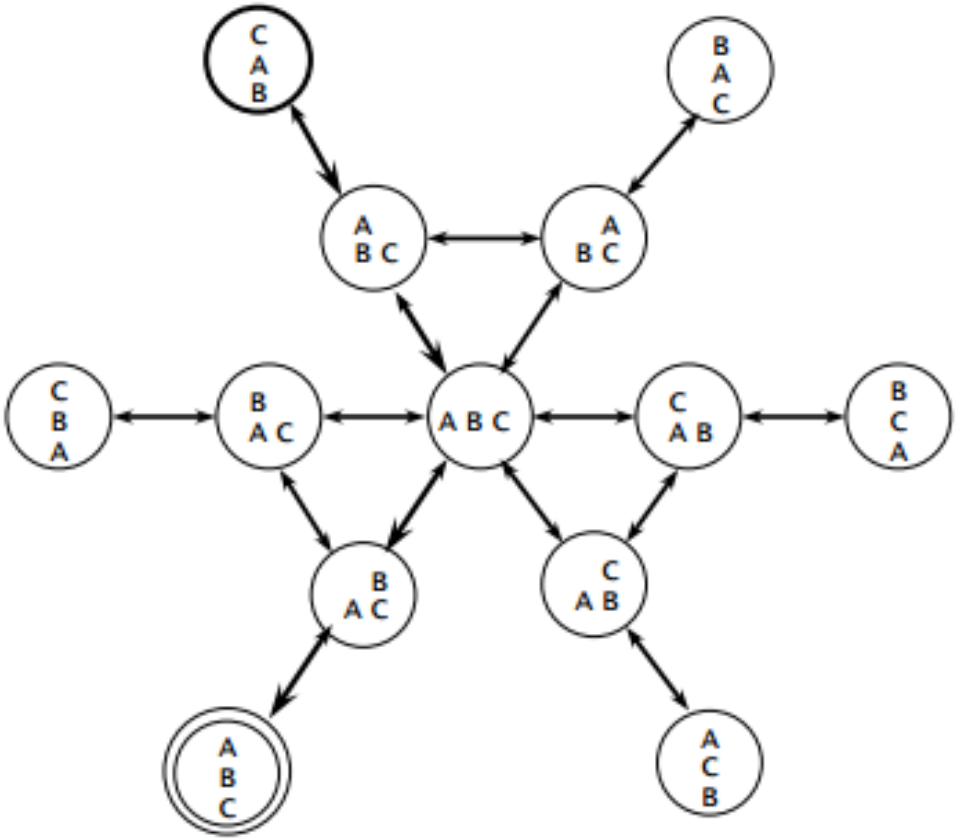
- O que fazer ?
  - **Podar alguns estados para reduzir a árvore a um tamanho finito** (ex. profundidade máxima fixa).
  - **Memorizar os estados visitados**.
    - Relação inversamente proporcional **espaço x tempo**.
    - *"Algoritmos que esquecem sua história estão condenados a repeti-la".*
  - Em problemas com muitos estados repetidos, **Busca-Em-Grafo é mais eficiente** do que Busca-Em-Árvores.





Espaço de Estados

- Exemplo: **Mundo dos Blocos**
  - Em problemas com muitos estados repetidos, **Busca-Em-Grafo é mais eficiente** do que Busca-Em-Árvores.

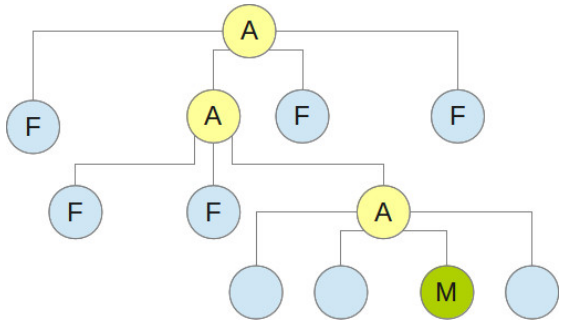


Algoritmos de Busca

- A **busca em um espaço de estados** é o **processo de procurar um caminho de solução** iniciando no estado inicial até alcançar o estado objetivo.
- **Algoritmos de Busca**
  - **sem informação (ou busca cega)**
    - Busca em largura ou amplitude ou extensão
    - Busca em profundidade
  - **com informação (usam funções heurísticas)**

Busca Com Informação pela Melhor Escolha

- **Busca com informação (ou heurística)** utiliza conhecimento específico do problema para encontrar a solução.
  - Pode ser mais eficiente que a busca cega.
- **Busca pela melhor escolha**: escolhe para expansão o nó com custo mais baixo a partir de uma função de avaliação.



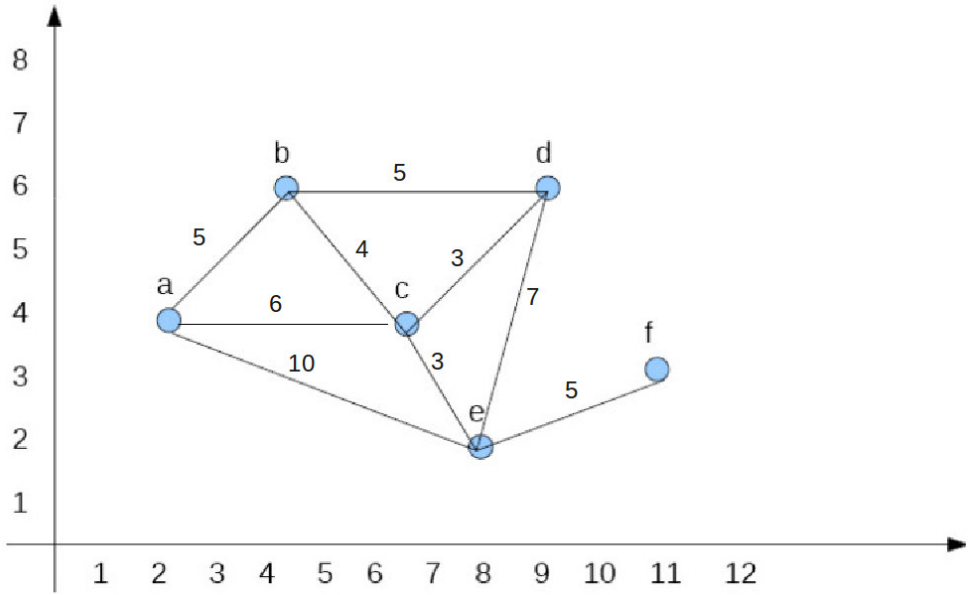
- **Função de avaliação:**
  - mede (estima) a distância do nó atual até o objetivo.
  - usa **heurística** para calcular essa distância

Busca Com Informação pela Melhor Escolha

- **Função de avaliação** (heurística): $f(n) = h(n)$
  - **Heurística**:
    - probabilidade ou suposição a respeito da resolução de um problema
    - regra para escolher aqueles ramos em um espaço de estado que têm maior probabilidade de levarem a uma solução aceitável.
  - **Razões para usá-la**:
    - **O problema não tem uma solução exata** por causa das **ambiguidades na formalização do problema ou nos dados disponíveis**. Ex: diagnóstico médico
    - **O problema tem solução exata, mas o custo computacional é proibitivo**. Ex: jogo de xadrez

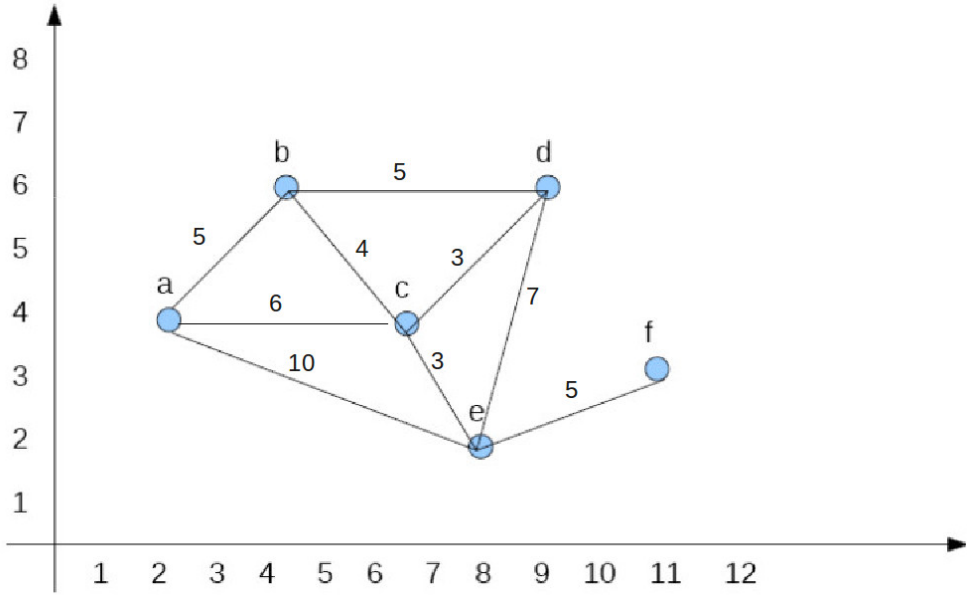
Busca Com Informação pela Melhor Escolha

- **Exemplo**: Considere **5 cidades: a, b, c, d, e, f.** As distâncias reais (entre elas) são apresentadas na imagem a seguir. As coordenadas referentes a latitude e logitude das cidades são definidas como:
  - coord(a,2,4),coord(b,4,6), coord(c,6,4), coord(d,8,6), coord(e,7,2) e coord(f,10,3).



Busca Com Informação pela Melhor Escolha

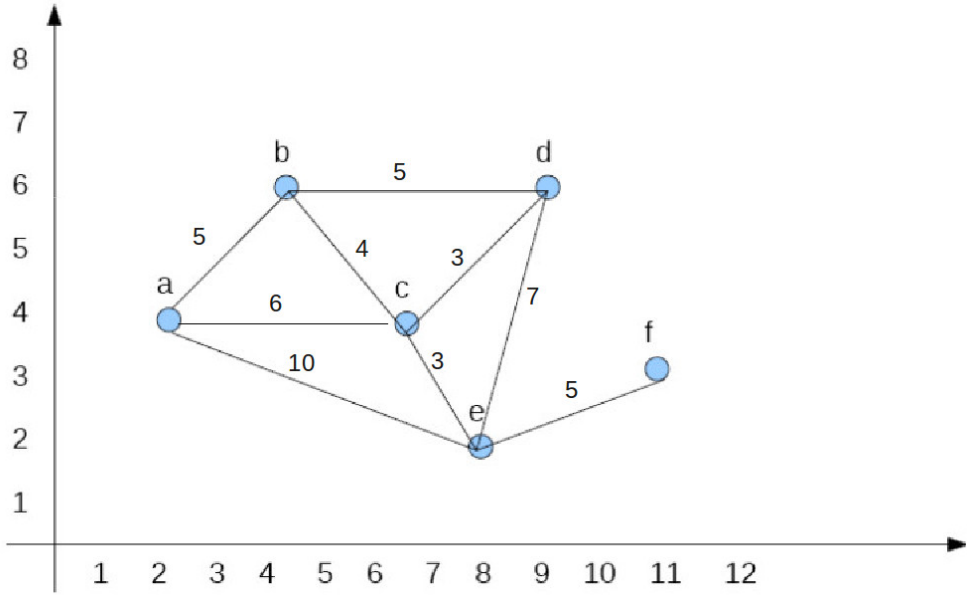
- **Exemplo**: Considere **5 cidades: a, b, c, d, e, f...**
- Com base nas informações dadas, qual o melhor caminho de **a** para **f** ?



- O caminho mais curto é **a-c-e-f**, ou seja, 14 km.
- Usando apenas as distâncias reais podemos encontrar esse caminho, sem visitar todos os outros ?

Busca Com Informação pela Melhor Escolha

- **Exemplo**: Considere **5 cidades: a, b, c, d, e, f...**
  - Vamos usar uma função heurística para estimar a distância das cidades em linha reta.
  - Best First: $h(cidade) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$, onde (x_1, y_1) são as coordenadas de cidade e (x_2, y_2) da cidade objetivo.



Busca Com Informação pela Melhor Escolha

- **Exemplo**: Considere **5 cidades: a, b, c, d, e, f...**
  - coord(a,2,4),coord(b,4,6), coord(c,6,4), coord(d,8,6), coord(e,7,2) e coord(f,10,3).
  - $h(cidade) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

  - Ciclo 0 - Caminho: [**a**]
  - Cidade Atual: **a**
    - Distância estimada das cidades vizinhas de **a** (b, c, e) em relação ao objetivo (cidade **f**).

      $h(b) = \sqrt{(10-4)^2 + (3-6)^2} = \sqrt{45} \sim 6,7$

      $h(c) = \sqrt{(10-6)^2 + (3-4)^2} = \sqrt{17} \sim 4,1$

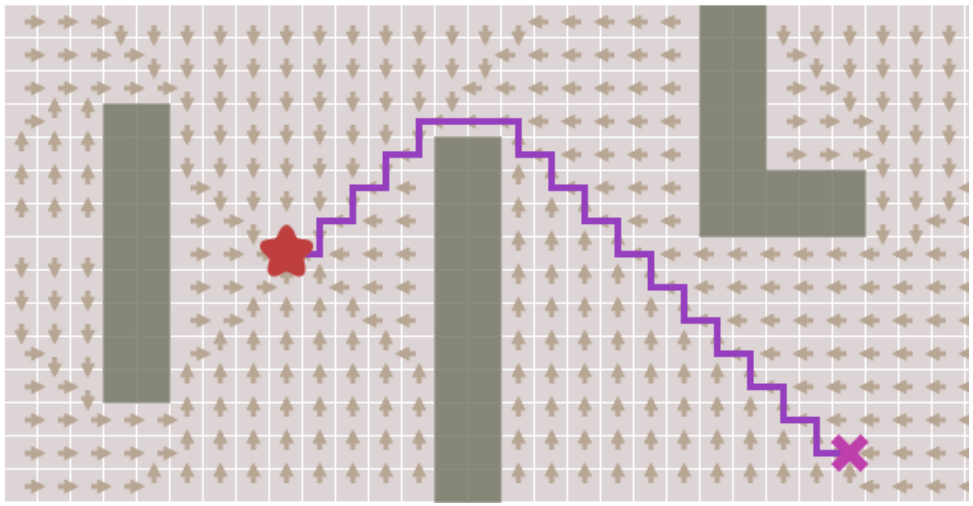
      $h(e) = \sqrt{(10-7)^2 + (3-2)^2} = \sqrt{10} \sim 3,1$

Busca Com Informação pela Melhor Escolha

- **Exemplo**: Considere **5 cidades: a, b, c, d, e, f...**
  - Ciclo 1 - Caminho: [**a**,**e**]
  - Cidade Atual: **e**
    - Distância estimada das cidades vizinhas de **e** (a, c, d, f) em relação ao objetivo (cidade **f**): como **f** aparece entre as cidades vizinhas, o algoritmo pára.
    - Retorna Caminho: [**a**,**e**,**f**]

Busca Com Informação pela Melhor Escolha

- Dado o seguinte problema: encontrar o caminho que leva do ponto inicial(estrela) ao ponto meta(xis).



Busca Com Informação pela Melhor Escolha

- Trechos de Código em Python:

```
frontier = PriorityQueue()
frontier.put(start, 0)
came_from = {}
came_from[start] = None

while not frontier.empty():
   current = frontier.get()

   if current == goal:
      break

   for next in graph.neighbors(current):
      if next not in came_from:
         priority = heuristic(goal, next)
         frontier.put(next, priority)
         came_from[next] = current
```

```
current = goal
path = [current]
while current != start:
   current = came_from[current]
   path.append(current)
path.append(start) # optional
path.reverse() # optional
```

```
def heuristic(a, b):
   # Manhattan distance on a square grid
   return abs(a.x - b.x) + abs(a.y - b.y)
```

Busca Com Informação pela Melhor Escolha

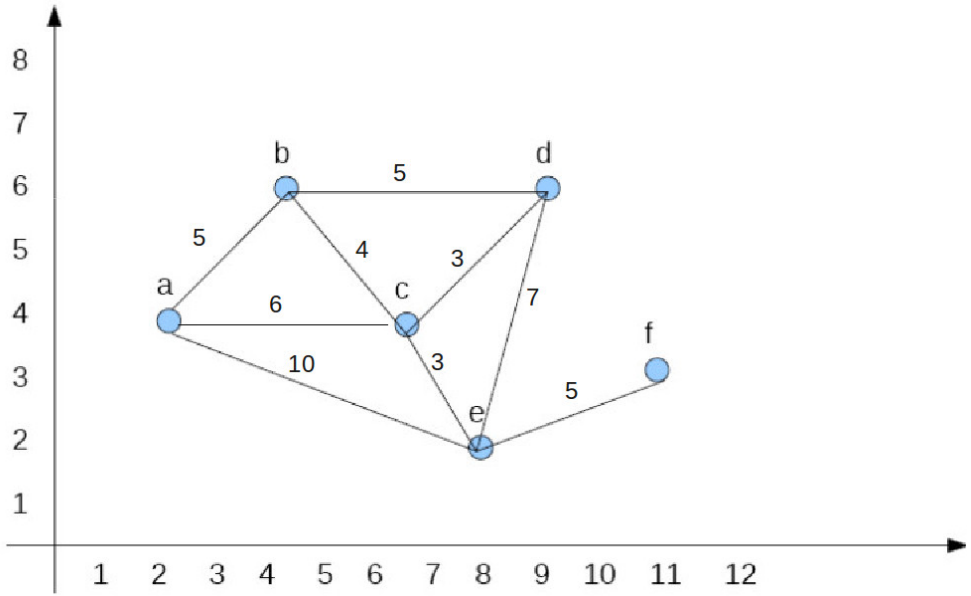
- A **busca gulosa pela melhor escolha (best first) é semelhante a busca em profundidade** em vários aspectos:
  - **segue um único caminho**;
  - **não garante** que encontrará a **solução ótima**;
  - pode ser **incompleta**: ficar presa em ciclos (estados repetidos)
  - complexidade de tempo - **pior caso:** $O(b^m)$, onde b é o fator de ramificação (número máximo de sucessores de um nó) e m é a profundidade máxima do espaço de busca.
- No entanto, a **busca pela melhor escolha pode reduzir substanciamente essa complexidade** dependendo da **qualidade da função heurística** e das especificidades do problema em questão.

Busca Com Informação: A*

- Algoritmo amplamente conhecido.
- **Usa ao menos 2 funções heurísticas**: $f(n) = g(n) + h(n)$, onde:
  - $g(n)$ é o custo para alcançar cada nó n (estado) a partir de um nó anterior; (custo real)
  - $h(n)$ é o custo para ir do nó n ao nó objetivo. (custo estimado)
- **A* alcançará a solução ótima** se for usada **busca em árvore** e a heurística de $h(n)$ for admissível:
  - **admissivel**: não superestima o custo para alcançar o objetivo (otimista, gera valores menores que a realidade).
    - Ex: estimar a distância em linha reta é admissivel, pois representa a menor distância entre 2 pontos, logo não pode ser considerada uma superestimativa.

Busca Com Informação: A*

- Voltando ao exemplo das **5 cidades: a, b, c, d, e, f.**
  - $f(n) = g(n) + h(n)$, onde:
    - $g(n)$: custo real de ir até a cidade n, distância em km.
    - $h(n) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$: custo estimado da cidade n até a cidade objetivo.
  - Melhor caminho de **a** a **f** ?



Busca Com Informação: A*

- Voltando ao exemplo das **5 cidades: a, b, c, d, e, f.**
  - $f(n) = g(n) + h(n)$, onde:
    - $g(n)$: custo real de ir até a cidade n, distância em km.
    - $h(n) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$: custo estimado da cidade n até a cidade objetivo.
  - **Ciclo 0** - Caminho: [**a**]
  - Cidade Atual: **a**
    - Distância estimada das cidades vizinhas de **a** (b, c, e) em relação ao objetivo (cidade **f**).
      $g(b) = 5,\ h(b) = \sqrt{(10-4)^2 + (3-6)^2} = 6,7$
      ▶ $f(b) = 11,7$
      $g(c) = 6,\ h(c) = \sqrt{(10-6)^2 + (3-4)^2} = \sqrt{17} \sim 4,1$
      ▶ $f(c) = 10,1$
      $g(e) = 10,\ h(e) = \sqrt{(10-7)^2 + (3-2)^2} = \sqrt{10} \sim 3,1$
      ▶ $f(e) = 13,1$

Busca Com Informação: A*

- Voltando ao exemplo das **5 cidades: a, b, c, d, e, f.**
  - $f(n) = g(n) + h(n)$, onde:
    - $g(n)$: custo real de ir até a cidade n, distância em km.
    - $h(n) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$: custo estimado da cidade n até a cidade objetivo.
  - **Ciclo 1** - Caminho: [**a**,**c**]
  - Cidade Atual: c
    - Distância estimada das cidades vizinhas de **c** (a, b, d,e) em relação ao objetivo (cidade **f**).
      $g(b) = 4,\ h(b) = \sqrt{(10-4)^2 + (3-6)^2} = 6,7$
      ▶ $f(b) = 10,7$
      $g(d) = 3,\ h(c) = \sqrt{(10-8)^2 + (3-6)^2} = \sqrt{13} \sim 3,6$
      ▶ $f(d) = 6,6$
      $g(e) = 3,\ h(e) = \sqrt{(10-7)^2 + (3-2)^2} = \sqrt{10} \sim 3,1$
      ▶ $f(e) = 6,1$

Busca Com Informação: A*

- Voltando ao exemplo das **5 cidades: a, b, c, d, e, f.**
  - $f(n) = g(n) + h(n)$, onde:
    - $g(n)$: custo real de ir até a cidade n, distância em km.
    - $h(n) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$: custo estimado da cidade n até a cidade objetivo.
  - **Ciclo 2** - Caminho: [**a**,**c**,**e**]
  - Cidade Atual: e
    - Distância estimada das cidades vizinhas de **e** (a, c, d, f) em relação ao objetivo (cidade **f**): como **f** aparece entre as cidades vizinhas, o algoritmo pára.
    - Retorna Caminho: [**a**,**c**,**e**,**f**]

Dijkstra

- Algoritmo que resolve o problema de encontrar o menor caminho entre dois nodos de um grafo.
- Desenvolvido nos anos 50 por Edsger Dijkstra
- Subdividido em duas partes:
  - Cálculo da tabela de distâncias e antecessores
  - Geração da sequência final, via tabela

Dijkstra - Parte 1

```
 1  function Dijkstra(Graph, source):
 2      for each vertex v in Graph:
 3          dist[v] := infinity ;
 4          previous[v] := undefined ;
 5      end for ;
 6      dist[source] := 0 ;
 7      Q := the set of all nodes in Graph ;
 8      while Q is not empty:
 9          u := vertex in Q with smallest dist[] ;
10          if dist[u] = infinity:
11              break ;
12          end if ;
13          remove u from Q ;
14          for each neighbor v of u:
15              alt := dist[u] + dist_between(u, v) ;
16              if alt < dist[v]:
17                  dist[v] := alt ;
18                  previous[v] := u ;
19              end if ;
20          end for ;
21      end while ;
22      return dist[] ;
23  end Dijkstra.
```

Dijkstra - Parte 2

```
1  S := empty sequence
2  u := target
3  while previous[u] is defined:
4      insert u at the beginning of S
5      u := previous[u]
```

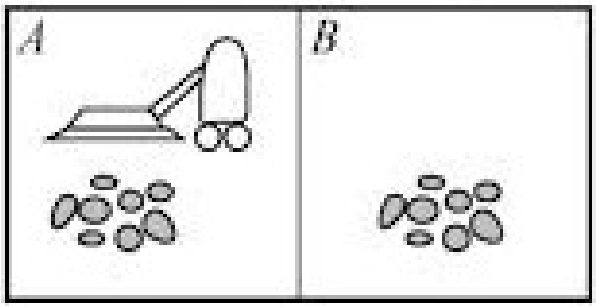
Busca Com Informação: A*

- Trecho do código do A* em Python

```
frontier = PriorityQueue()
frontier.put(start, 0)
came_from = {}
cost_so_far = {}
came_from[start] = None
cost_so_far[start] = 0

while not frontier.empty():
   current = frontier.get()

   if current == goal:
      break

   for next in graph.neighbors(current):
      new_cost = cost_so_far[current] + graph.cost(current, next)
      if next not in cost_so_far or new_cost < cost_so_far[next]:
         cost_so_far[next] = new_cost
         priority = new_cost + heuristic(goal, next)
         frontier.put(next, priority)
         came_from[next] = current
```

Exercícios Práticos

- **Atividade 1:** Implementar o agente aspirador de pó visto em aula. Use um algoritmo de busca para encontrar a sequência de ações que levam de um estado inicial qualquer ao estado objetivo (locais limpos).

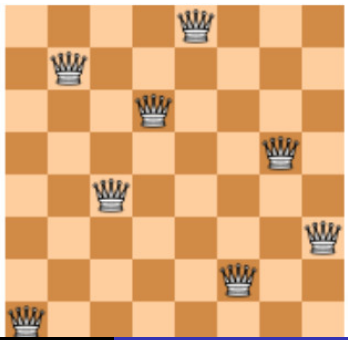


Exercícios Práticos

- **Atividade 2:** Considere agora que o agente aspirador de pó é responsável por limpar 4 salas de aula, sendo que duas ficam no térreo e duas ficam no segundo andar. O agente pode estar em 5 locais diferentes: em uma das 4 salas ou fora delas. Para isso, ele possui o seguinte conjunto de ações: {subir, descer, entrarNaSala1, entrarNaSala2, sair e aspirar}. Para subir, descer ou entrar nas salas, ele deve estar fora delas. A ação sair sempre o leva para fora das salas. Ele aspirar apenas a sala em que está.
  - Defina uma representação adequada para o problema;
  - Crie uma função sucessor;
  - Defina uma função para testar o estado objetivo;
  - Defina uma função para calcular o custo (1 por ação).
  - Implemente a busca da solução por meio de um algoritmo de busca. Devolver a solução com menor custo.

Exercícios Práticos

- **Atividade 3** - O Problema das 8 rainhas consiste em dispor 8 rainhas em tabuleiro sem que uma ataque a outra. Uma rainha ataca a outra se esta estiver na mesma linha, coluna ou diagonal.
  - Quantos estados tem esse problema ? E se fossem **n** rainhas ?
  - Defina uma representação para esse problema, uma função sucessor e uma função para testar o objetivo.
  - Tente encontrar a solução, tendo como ponto de partida o tabuleiro vazio. A cada passo, vá inserindo rainha por rainha no tabuleiro. Considere na sua solução, apenas estados cuja a inserção de uma nova rainha no tabuleiro não é confliante com as que já estão no tabuleiro.



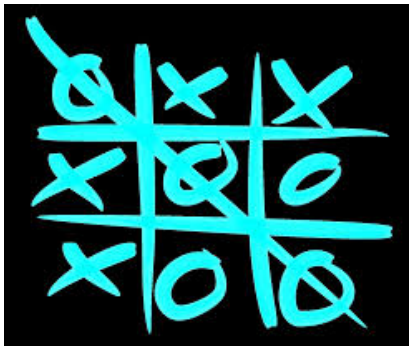
Exercícios Práticos

- **Atividade 4:** Considere agora que o agente dispõe de 2 jarros. O primeiro jarro tem 3 litros de capacidade e o segundo, 4 litros. Considere também que o agente dispõe das seguintes ações:
  - **encher1**: enche o jarro1 até a sua capacidade máxima (3 litros).
  - **encher2**: enche o jarro2 até a sua capacidade máxima (4 litros).
  - **esvaziar1**: retira todo líquido do jarro1.
  - **esvaziar2**: retira todo líquido do jarro2.
  - **despejar1em2**: despeja o líquido do jarro1 no jarro2, sendo que o jarro2 não pode conter mais de 4 litros. Quando a soma dos jarros for maior que 4, jarro2 ficará com 4 litros e jarro1, com os litros restantes.
  - **despejar2em1**: despeja o líquido do jarro2 no jarro1, sendo que o jarro1 não pode conter mais de 3 litros. Quando a soma dos jarros for maior que 3, jarro1 ficará com 3 litros e jarro2, com os litros restantes.

  Defina uma representação adequada para o problema, a função sucessor e a função objetivo. Use um algoritmo de busca para encontrar a sequência de ações que leva do estado inicial em que os jarros estão vazios ao estado objetivo em que o jarro1 está vazio e o jarro2 tem exatamente 2 litros.

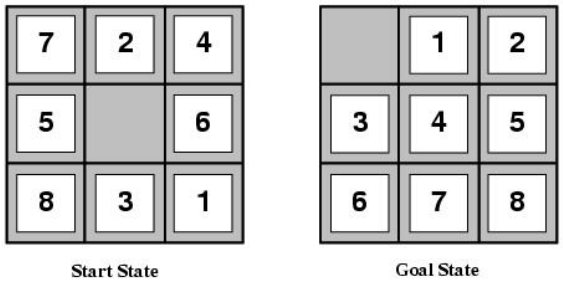
Exercícios Práticos

- **Atividade 5**: Crie uma representação para o jogo da velha.
  - Quantos estados tem esse problema ?
  - Implemente a função sucessor.



Exercícios Práticos

- **Atividade 6:** Considere o quebra-cabeça de peças deslizantes abaixo, a seguir defina:
  - Uma representação para o problema
  - Uma função sucessor
  - Uma função de teste do objetivo



Para encontrar a solução para este problema, a repetição de estados pode acontecer ? Por que?

Exercícios Práticos

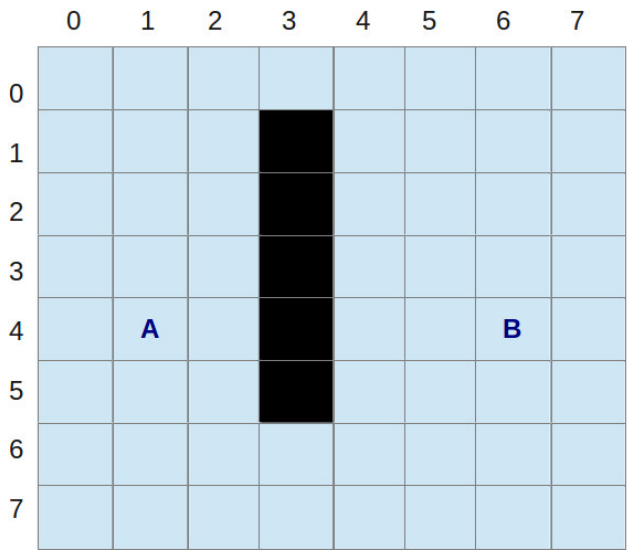
- **Atividade 7:** Para responder as questões abaixo, consulte o capítulo 3 do livro do Russel e Norvig.
  1. Qual a diferença entre um algoritmo de busca em extensão de um algoritmo de custo uniforme ?
  2. Qual a diferença entre um algoritmo de busca em profundidade de um algoritmo com busca limitada ou iterativa ? Todos podem alcançar a solução ótima ?
  3. Como funciona uma busca bidirecional ? Ela é sempre aplicável ? O que é necessário para aplicá-la ?
  4. Quem consome mais memória os algoritmos de busca em extensão ou em profundidade ?

Exercícios Práticos

- **Atividade 8:**Volte ao exercicio 4 e monte a árvore de busca, usando a função heurística dada abaixo que resolve esse problema, considerando que
  - os estados são representados da seguinte forma $[J1, J2]$, onde $J1$ e $J2$ são valores inteiros que correspondem a quantidade atual de líquido em cada jarro.
    - a função heurística é dada por $h([J1, J2]) = |J1 - JA| + |J2 - JB|$, onde $[JA, JB]$ representa o estado objetivo.
    - o estado inicial é $[0,0]$ e o estado objetivo é $[0,2]$.

Exercícios Práticos

- **Atividade 9:** Considere agora o ambiente abaixo. Sabendo que o agente está na posição A e que o objetivo do agente é a posição B, qual o caminho que o agente deve seguir para chegar ao seu objetivo ? As células pretas correspondem a uma parede intransponível.



Exercícios Práticos

- **Atividade 10:** ... Use o algoritmo A* ($f(n) = g(n) + h(n)$) para resolver esse problema, levando em conta que
  - o agente pode se movimentar em qualquer direção.
  - $g(n)$: o custo de se movimentar no sentido horizontal ou vertical é 10 e na diagonal é 14.
  - $h(h)$: distancia * custo de movimentação (estimado com no mínimo 10).
    - Teste a distância de manhattan ($|x_2 - x_1| + |y_2 - y_1|$) e a distância eucliadiana ($\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ e verifique qual é a admissível para esse problema.

Exercícios Práticos

- **Atividade 10:** ... Exemplo usando a distância de manhattan:
  - Posição Inicial: (4,1)
  - Posição Objetivo: (4,6)

Posição Atual de A = (4,1)			
n = Próximo Movimento	**g(n)**	**h(n)**	**f(n)**
Diagonal Para cima à esquerda (3,0)	14	$(\|3-4\|+\|0-6\|)*10=70$	84
Para cima (3,1)	10	$(\|3-4\|+\|1-6\|)*10=60$	70
Diagonal Para cima à direita (3,2)	14	$(\|3-4\|+\|2-6\|)*10=50$	64
Para direita (4,2)	**10**	$(\|4-4\|+\|2-6\|)*10=40$	50
Diagonal Para baixo à direita (5,2)	14	$(\|5-4\|+\|2-6\|)*10=50$	64
Para baixo (5,1)	10	$(\|5-4\|+\|1-6\|)*10=60$	70
Diagonal Para baixo à esquerda (5,0)	14	$(\|5-4\|+\|0-6\|)*10=70$	84
Para esquerda (4,0)	10	$(\|4-4\|+\|0-6\|)*10=60$	70

Exercícios Práticos

- **Atividade 11:** Voltando ao problema do quebra-cabeça de peças deslizantes abaixo, resolva o problema usando o algoritmo A* ($f(n) = g(n) + h(n)$). Em sua implementação, use as seguintes funções:
  - $g(n)$= quantidade de peças fora do lugar em relação ao estado objetivo.
  - $h(n)$= a soma das distâncias das posições atuais dos blocos de suas posições objetivos. Use a distância de manhattan.