Aula 07 e 08 - Métodos de busca

terça-feira, 19 de abril de 2022 08:09

Algoritmos de busca
Teoria de busca em espaço de estados é uma boa ferramenta
Analisar e prever o comportamento de busca
Algumas perguntas

- O resolvedor encontrará garantidamente a solução?
- O resolvedor terminará (não entrará em looping)?
- A solução encontrada é ideal?
- Qual a complexidade do processo de busca (tempo, memória)?
- Como o interpretador pode reduzir eficientemente a complexidade?

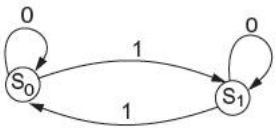
Problema das pontes de Königsberg
Como passear a pé pela cidade, passando uma única vez por cada uma das sete pontes, e retornar ao ponto de partida?
Euler propôs um método usando teoria dos grafos para resolver o problema - não tinha solução

- Eliminou detalhes geométricos (comprimento, forma, distâncias...) -> grafo
- Teoria dos Grafos pode ser utilizado em algoritmos de busca

Máquina de estado (um lado –munda estado-> outro lado da ponte)

- Tripla ordenada de estados (S, I, F)
  - S: conjunto finito de estados de um grafo
  - I: conjunto finito de valores de entrada (input)
  - F: função de transição de estado - descreve o efeito de qualquer $i \in I$ sobre o estado S da máquina
    - $\forall\, i \in I,\ F_i : (S \to S)$
    - O estado seguinte a um estado s_j é definido por $F_i\,(s_j)$

Flip-flop
A saída depende do valor das entradas e/ou dos estados armazenados
Opera sob o comando de pulsos de clock



(a)

	0	1
S_0	S_0	S_1
S_1	S_1	S_0

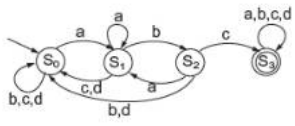
(b)

(a) Grafo de estados finitos
(b) matriz de transição

Máquina de Moore
Reconhecedor de estados finitos: máquina de estados finitos (S, I, F) na qual:

- $\exists\ s_0 \in S$ tal que o fluxo de entrada comece em s_0
- $\exists\ s_n \in S$, um estado de aceitação
  - Fluxo de entrada é aceito se terminar nesse estado
  - Pode haver um conjunto de estados de aceitação

Representação da máquina: (S, s_0, {s_n}, I, F)
Ex. reconhecedor de todas as sequências {a, b, c, d} que contenha a sequência exata "abc"



(a)

	a	b	c	d
→ S_0	S_1	S_0	S_0	S_0
S_1	S_1	S_2	S_0	S_0
S_2	S_1	S_0	S_3	S_0
(S_3)	S_3	S_3	S_3	S_3

(b)

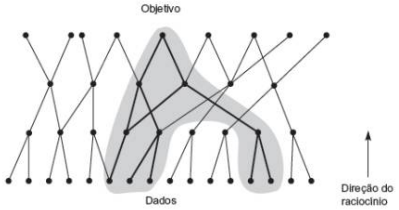
Buscas guiadas por dados e por objetivo

Busca guiada por dados
Parte de fatos para chegar ao objetivo

- Usa os dados para criar regras que cheguem ao objetivo

Recomendada nos casos

- Todos os dados ou a maioria dos dados são fornecidos na formulação do problema
- Problemas com muitos objetivos ou com objetivos difíceis de serem formulados
  - Vai tirando inferências a partir dos dados

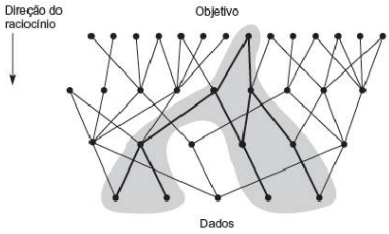


Busca guiada por objetivos
Parte do objetivo

- Encontrar regras que possam produzir o objetivo
- Vai encontrando os dados (fatos) fornecidos do problema

Recomendada em casos

- Objetivo ou hipótese é facilmente formulado ou já é conhecido
- Grande número de regras se aplicam ao objetivo
  - Número crescente de conclusões ou objetivos
- Dados não são fornecidos
  - Precisam ser adquiridos para resolver o problema



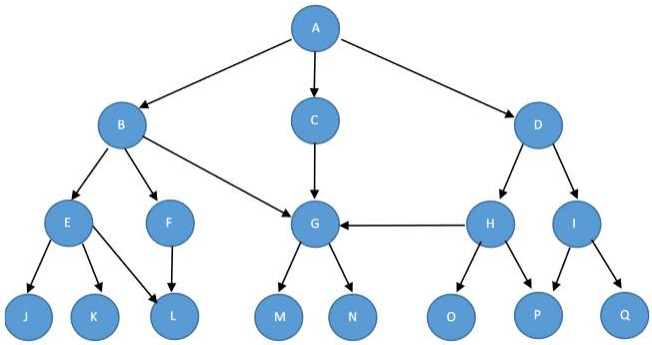
Busca por retrocesso (backtracking)

Tenta sistematicamente todos os caminhos por um espaço de estados

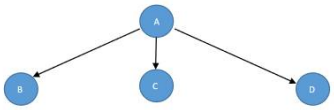
- Percorre cada estado ligado ao estado que está sendo analisado
- "Abre" os grafos dependentes até chegar ao estado objetivo

Semelhante a um labirinto: chega a um ponto sem saída, volte um passo

Ex. Estado inicial: A; Objetivo: O; Estratégia de controle: ordem alfabética



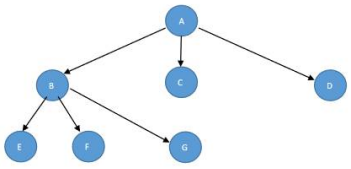
Grafo do problema



Abrir o estado inicial e verificar. Se ele não é o meu objetivo, abrir os estados ligados a ele. Após abrir os três estados começo a analisar um por um dentro da estratégia de controle que foi estabelecida.

Se não é o estado objetivo
Então abre os estados filhos

Analisando A

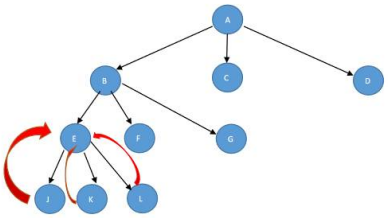


Abrir o estado inicial e verificar. Se ele não é o meu objetivo, abrir os estados ligados a ele. Após abrir os três estados começo a analisar um por um dentro da estratégia de controle que foi estabelecida.

Se B não é o estado objetivo
Então abre os estados filhos

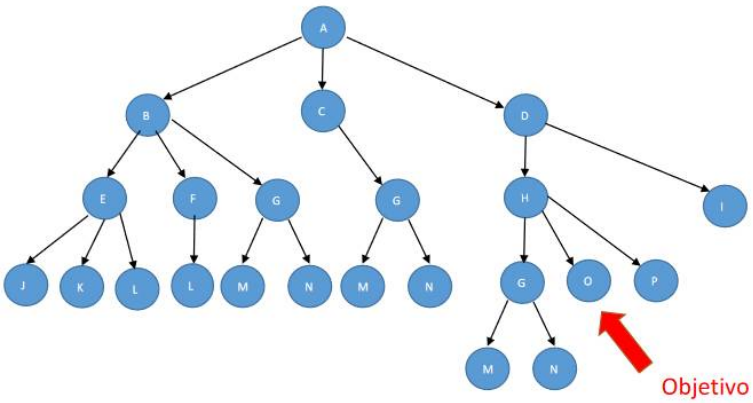
Agora, se nos filhos eu não encontrar o estado objetivo ou não tenha estados filhos, então realizo um Backtracking.

Analisando B



Chegamos ao estado J. Se ele não for meu objetivo e não tiver estados filhos, então devo fazer um Backtracking para o estado E e daí analisar o próximo estado que neste segundo a regra de controle (ordem alfabética) e a direção será de E para K. Se K não for meu objetivo e não tiver filhos, então realizo um Backtracking para E. Depois faço o mesmo para L.

Analisando os filhos de E



Fim

Estratégias de busca

Num espaço de estados, a busca pode ser cega ou heurística
Todas as estratégias se distinguem pela ordem em que os nós são expandidos
Precisamos especificar a direção de busca (guiada por dados ou por objetivos) e a ordem na qual os estados são examinados na árvore ou no grafo (ordem alfabética, maior para o menor, direita para a esquerda... ou vice-versa)

Busca Cega

Não tem nenhuma informação, mas precisa chegar ao objetivo
- Não sabe qual sucessor é mais promissor para atingir a meta

Existem duas estratégias: busca em profundida e busca em largura

Busca em profundidade
Tem várias possibilidades, examina todos os filhos e descendentes antes dos irmãos
- Examina toda uma ramificação até o final, depois parte para a ramificação adjacente

Os nós são percorridos com backtracking
- Se não houver um saída (outro nó) para seguir, vai voltando até ter

Cada nó pode estar aberto ou fechado
- Lista de abertos: estados gerados, mas cujos filhos não foram examinados
  - A ordem de remoção de estados da lista de abertos determina a ordem de busca
- Lista de fechados: estados que já foram examinados
  - Passou pelo nó -> marcador muda (fechado)
  - Lista ordenada de estados no caminho da solução

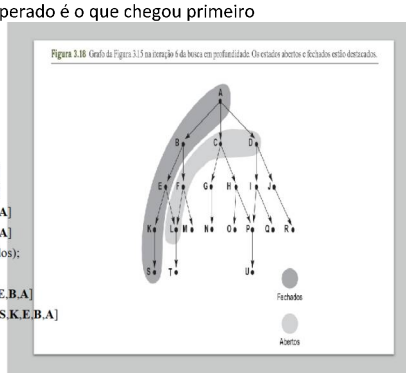
Pilha de dados: novos dados vão sendo empilhados
- Primeiro dado a ser recuperado é o mais recente

Fila de dados: dados são enfileirados
- Primeiro dado a ser recuperado é o que chegou primeiro

Algoritmo busca em profundidade

1. abertos = [**A**]; fechados = [ ]
2. abertos = [**B**,**C**,**D**]; fechados = [**A**]
3. abertos = [**E**,**F**,**C**,**D**]; fechados = [**B**,**A**]
4. abertos = [**K**,**L**,**F**,**C**,**D**]; fechados = [**E**,**B**,**A**]
5. abertos = [**S**,**L**,**F**,**C**,**D**]; fechados = [**K**,**E**,**B**,**A**]
6. abertos = [**L**,**F**,**C**,**D**]; fechados = [**S**,**K**,**E**,**B**,**A**]
7. abertos = [**T**,**F**,**C**,**D**]; fechados = [**L**,**S**,**K**,**E**,**B**,**A**]
8. abertos = [**F**,**C**,**D**]; fechados = [**T**,**L**,**S**,**K**,**E**,**B**,**A**]
9. abertos = [**M**,**C**,**D**] (como **L** já está em fechados); fechados = [**F**,**T**,**L**,**S**,**K**,**E**,**B**,**A**]
10. abertos = [**C**,**D**]; fechados = [**M**,**F**,**T**,**L**,**S**,**K**,**E**,**B**,**A**]
11. abertos = [**G**,**H**,**D**]; fechados = [**C**,**M**,**F**,**T**,**L**,**S**,**K**,**E**,**B**,**A**]

e assim por diante, até que **U** seja descoberto
ou, então, que abertos = [ ].

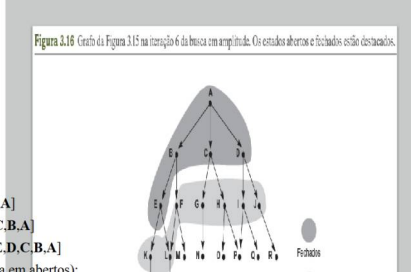


Busca em largura
Explora nível por nível
- Explora horizontalmente e depois aprofunda (avalia os filhos)

Cada iteração produz todos os filhos do estado analisado e seus filhos são adicionados à lista de abertos
- FIFO (first-in-first-out): primeiro a chegar é o primeiro a ser analisado
  - Primeiro os irmãos (chegaram primeiro na lista de abertos), depois os filhos (chegaram depois)

Algoritmo busca em largura (amplitude)

1. abertos = [**A**]; fechados = [ ]
2. abertos = [**B**,**C**,**D**]; fechados = [**A**]
3. abertos = [**C**,**D**,**E**,**F**]; fechados = [**B**,**A**]
4. abertos = [**D**,**E**,**F**,**G**,**H**]; fechados = [**C**,**B**,**A**]
5. abertos = [**E**,**F**,**G**,**H**,**I**,**J**]; fechados = [**D**,**C**,**B**,**A**]
6. abertos = [**F**,**G**,**H**,**I**,**J**,**K**,**L**]; fechados = [**E**,**D**,**C**,**B**,**A**]
7. abertos = [**G**,**H**,**I**,**J**,**K**,**L**,**M**] (pois **L** já esta em abertos);



fechados = [F,E,D,C,B,A]
8. abertos = [H,I,J,K,L,M,N]; fechados = [G,F,E,D,C,B,A]
9. e assim por diante, ate que U seja encontrado ou abertos = [ ].

Comparando buscas em profundidade e em largura

Cenário	Profundidade	Largura
Caminhos longos ou infinitos	Ruim	Bom
Caminhos com comprimentos parecidos	Bom	Bom
Todos os caminhos com comprimentos parecidos e todos levam a um estado objetivo	Bom	Ruim perde tempo e memória
Muitas ramificações	Desempenho depende de outros fatores	Precário

Lógica proposicional
Descrição por espaço de estados de um sistema lógico
Modus ponendo ponens (a maneira que afirma afirmando): eliminação de implicação

- Simples regra de inferência
- "P implica Q, P é afirmado verdade, portanto, Q deve ser verdade."
  - Terça-feira tem aula de Inteligência Artificial. Hoje é terça-feira. Então, tem aula de Inteligência Artificial

Lógica aristotélica

- Todo humano é mortal, Sócrates é humano, logo Sócrates é mortal

Grafos E/OU
Extensão do modelo básico de grafos: inclui operadores lógicos "e" e "ou"
Importantes para problemas de IA

- Provadores de teoremas
- Sistemas especialistas baseados em lógica

p -> q (se p, então q)
q ^ r -> p (se q interseção (e) r, então p)

q V r -> p (se q união (ou) r, então p)

Busca heurística

Heurística: conjectura informada sobre o próximo passo a ser tomado na solução do problema
Tem informação estimada de qual sucessor é mais promissor para atingir a meta

- É uma busca cega com orientação

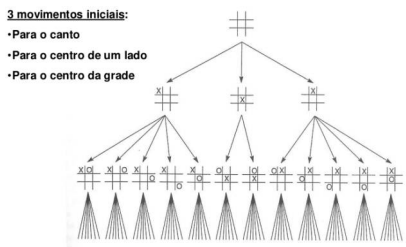
Baseada em experiência e intuição
Pode levar a uma solução subótima ou, inclusive, falhar
Heurísticas são empregadas em problemas de IA quando

- Não tem solução exata por causa de ambiguidades na formulação do problema ou nos dados disponíveis
- Pode até ter solução exata, mas o custo computacional para obtê-la é proibitivo

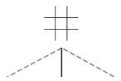
Ex. heurísticas do jogo da velha
O tabuleiro tem 9 espaços -> 1: 9 possibilidades -> 2: 8 possibilidades -> ... -> 9: 1 possibilidades

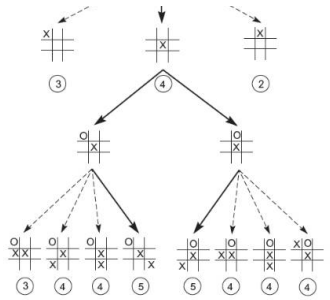
- Total de possibilidades: 9*8*7*...*1 = 9!

Podemos reduzir o problema por simetria



Heurística do maior número de vitórias aplicada ao primeiros filhos





X no centro de um lado: 2 chances de vitória
X no canto: 3 chances de vitória
X no centro: 4 chances de vitória

Algoritmo de subida de encosta

Expande o estado atual de busca e verifica os filhos (vai abrindo os nós)

- "Melhor" filho é selecionado
- Irmãos e "sobrinhos" não são considerados

Tendência de ficar presa em máximos locais

É importante lembrar que, como estratégia de busca local, a subida de encosta não retorna caminhos

Função heurística h(n)

Estima o custo do caminho de menor custo de n até um nó objetivo

- Ex. verificar distância do caminho entre duas cidades, pega a distância entre elas em linha reta
- Se n já for o objetivo, h(n) = 0

Forma mais comum de adicionar conhecimento do problema

- Específica para cada caso

Admissibilidade

Algoritmo é admissível se, seguramente, encontra um caminho mínimo até uma solução

- Solução sempre existe

Quebra o código em três partes

1. Calcular a função de "custo" do nó
2. Comparar as funções dos nós para determinar o menor caminho
3. Verifica se o nó é o objetivo