

Algoritmos de Busca

05/09/2022



Estruturas e Estratégias para busca em espaço de estados

- Para projetar e implementar algoritmos de busca, um programador precisa ser capaz de analisar e prever o seu comportamento. Algunhas perguntas que precisam ser respondidas:

- * O resolvelor do problema encontrará, garantidamente uma solução?
- * O resolvelor sempre terminará? Ele ficará preso em algum laço infinito?
- * Quando uma solução for encontrada, há garantias de que ela será a ideal?
- * Qual a complexidade do processo de busca em termos de tempo e memória?

- A teoria da busca em espaço de estados é nossa principal ferramenta para responder estas questões.
- Representamos problemas como um **grafo** de espaço de estados, podemos usar teoria dos grafos para analisar a estrutura e complexidade tanto do problema quanto dos procedimentos de busca que sempre vamos para resolvê-lo

1. Introdução

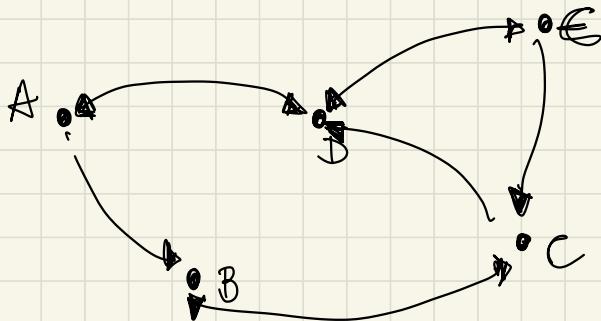
- Conjunto de nós e conjunto de arcos que conectam estes nós.
- No modelo de espaço de estados para a solução de problemas, os nós de um grafo são usados para representar estados discretos, como diferentes configurações de um

tabuleiro. Os arcos representam as transições entre estados. Essas transições correspondem a movimentos válidos em um jogo.

1.1 Grafos

↳ $G = (V, E)$, onde:

- V é um conjunto finito de vértices
 - E é um conjunto de arcos, que são pares de vértices $\in V$
- Exemplo de grafo direcionado e rotulado:



$$V = \{a, b, c, d, e\}$$

$$E = \{(a, b), (b, a), (b, c), (c, b), (c, d), (d, a), (d, e), (e, c), (e, d)\}$$

- algumas propriedades :

- caminhos
- adjacência

1.2 Máquina de Estados Finitos (MEF)

- É um grafo finito, direcionado, conectado, tendo um conjunto de estados, um conjunto de valores de entrada, e uma função de transição de estado que descreve o efeito que os elementos do fluxo de entrada têm sobre os estados do grafo.

→ modelo abstrato de computação

- MTF tem um uso importante no processo de reconhecimento de linguagens.

$$A = (Q, \Sigma, \delta, q_0, F) \text{ onde}$$

Q : conjunto de estados

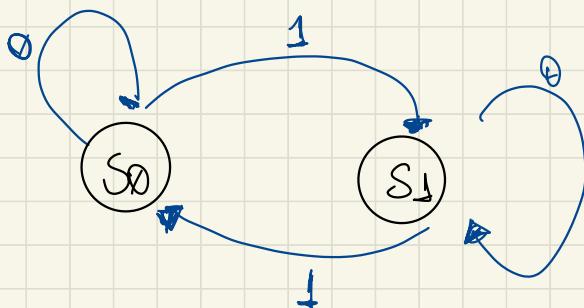
Σ : alfabeto, símbolos manipulados

δ : função de transição $\delta: Q \times \Sigma \rightarrow Q$

q_0 : estado inicial

F : conjunto de estados finais

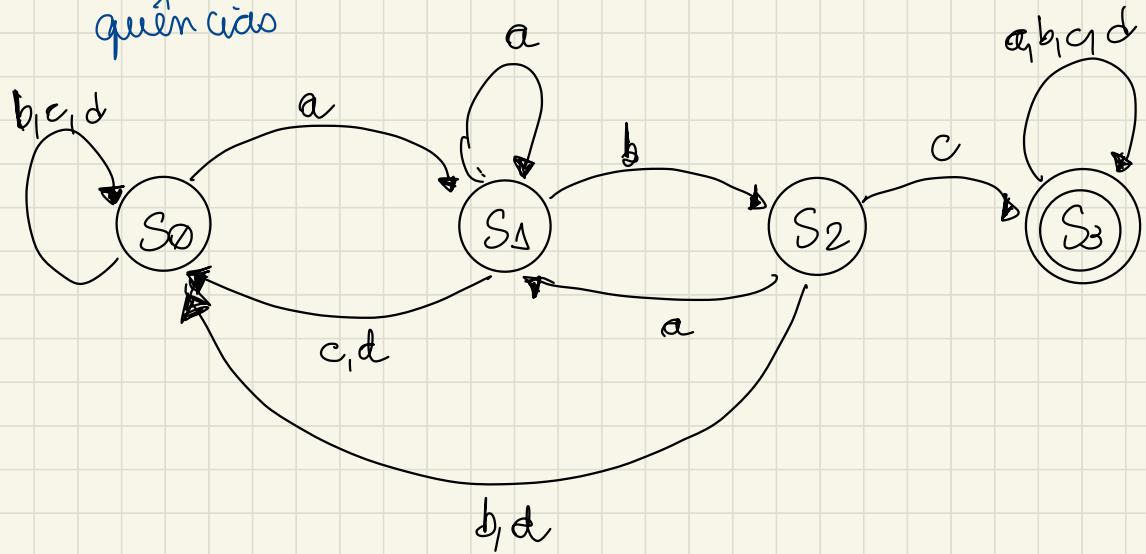
- Exemplo de MTF para um flip-flop



S	0	1
S_0	S_0	S_1
S_1	S_1	S_0

(Q5)

- Exemplo de MEF para reconhecimento de seções



S	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_1	S_3	S_3	S_3

2. Representação de Problemas por Espaço de Estados

- os nós de um grafo correspondem a estados de soluções parciais do problema, e os vértices correspondem a passos de um processo de solução de um problema
- um ou mais estados iniciais, que correspondem à informação fornecida de um problema, formam a raiz do grafo.
- o grafo também define uma ou mais condições relativas aos objetivos, que são as soluções para uma instância do problema
- A busca em espaço de estados caracteriza a solução de um problema como o processo de procurar um caminho de solução partindo do estado inicial até um objetivo.

- Um objetivo pode descrever:
 - um estado (jogo da velha)
 - configuração-alvo (quebra-cabeça)
- A criação ideal de novos estados ao longo do caminho é feita pela aplicação de operadores, como "movimentos válidos" em um jogo
- A tarefa do algoritmo de busca é encontrar um caminho de seleção por esse espaço de problemas.

→ os algoritmos devem acompanhar os caminhos de um nó inicial até um nó objetivo

* Formalmente:

um espaço de estados é representado por uma quadra:

$[N, A, I, DO]$, onde:

- N: conjunto de nós / estados do grafo
- A: arcos / arestas / transições . passos de um processo de solução de problema
- I: estado(s) inicial(ais) do problema
- DO: estado(s) objetivo(s) do problema

* Um caminho de solução é um caminho através de um nó em I para um nó em DO.

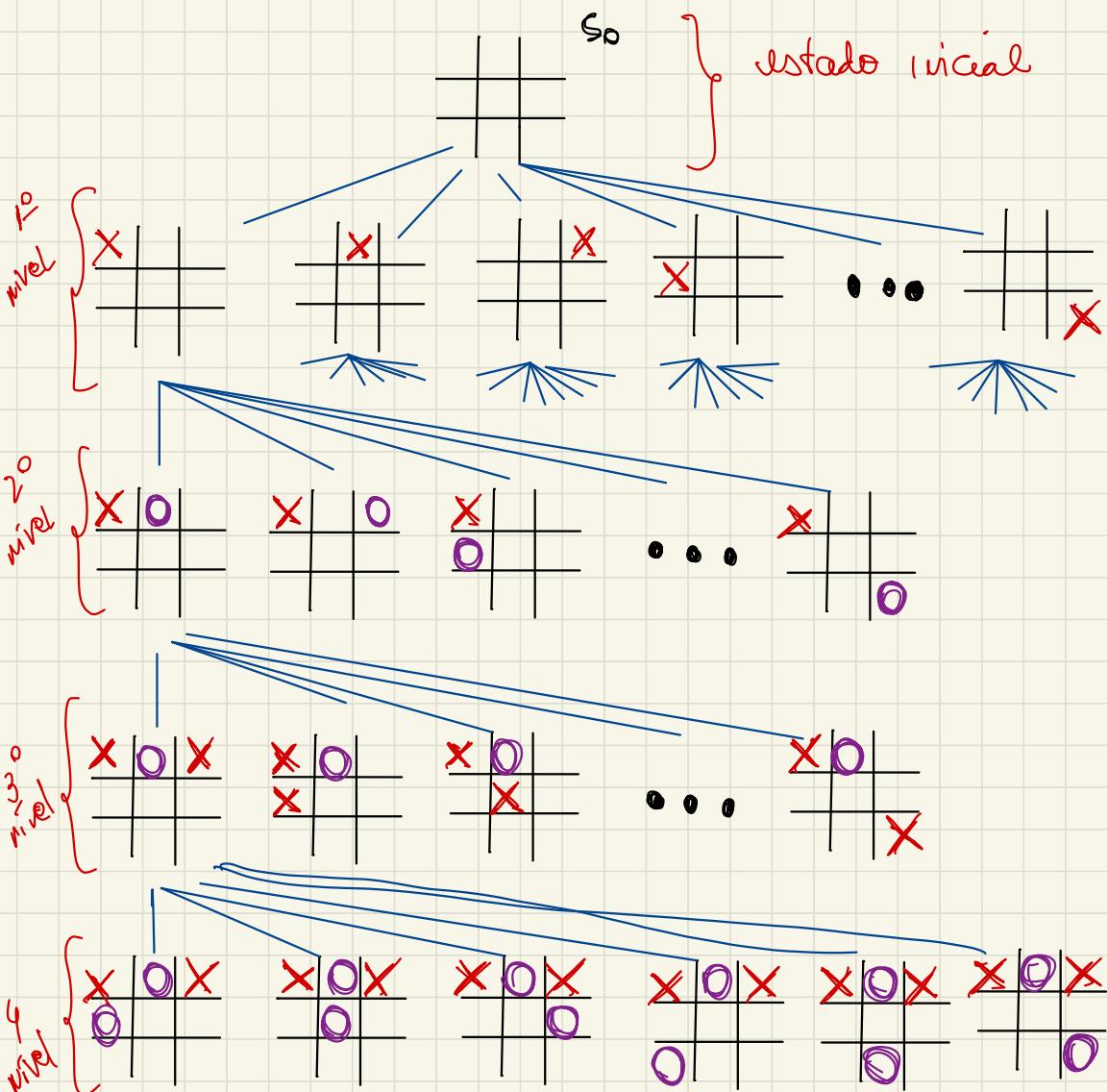
Exemplo qt: Jogo da velha

Estado inicial = tabuleiro vazio

Estado final = três X em linha, coluna ou diagonal

(Supondo que objetivo é vitória de X)

- Estados possíveis são todas configurações diferentes de Xs e Os que o jogo pode ter.



• • •

- as transições geram apenas movimentos válidos do jogo
- o espaço de estados é um grafo, porém não há ciclos, porque movimentos não podem ser desfeitos
- no jogo da velha há $9!$ caminhos diferentes que podem ser gerados
 - exaustivamente: 362.880

- outros jogos:

$=$ xadrez: 10^{120}
 $=$ damas: 10^{40}
} alguns caminhos nula
ocorrem em um jogo real

!

espaços difíceis ou impossíveis de se buscar exaustivamente

→ Usamos estratégias de busca e heurísticas para reduzir a complexidade da busca.



Exemplo 02: Quebra-Cabeça dos 8

- 8 peças com números diferentes são ajustadas em 9 espaços em uma grade.
- Um espaço em branco, de modo que as peças possam ser movimentadas para formar diferentes padrões
- Movimentos válidos:

- ↑ mover o Vazio para cima
- mover o Vazio para direita
- ↓ mover o Vazio para baixo
- ← mover o Vazio para esquerda

- Obs: Nem todos os quatro movimentos se aplicam o tempo todo, pois temos que garantir que o Vazio não se moverá para fora do tabuleiro

1	4	3
7		6
5	8	2

metido
acima

à esq

abaixo

à dir

1		3
7	4	6
5	8	2

1	4	3
	7	6
5	8	2

1	4	3
7	8	6
5		2

1	4	3
7	6	
5	8	2

1		3
7	4	6
5	8	2

1	3	
7	4	6
5	8	2

...

- Objective:

1	2	3
8		4
7	6	5

all

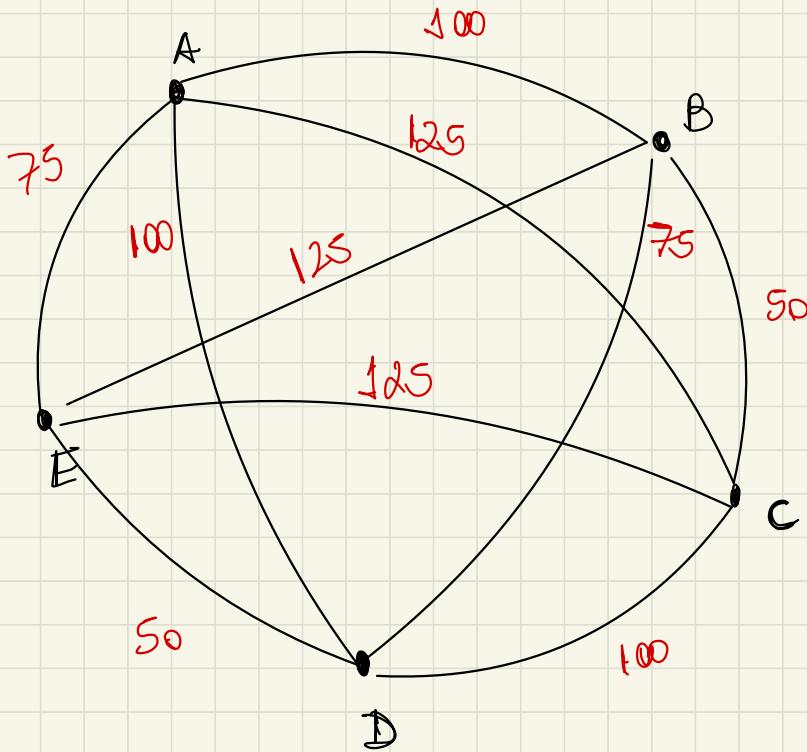
1	2	3
4	5	6
7	8	

Exemplo Q3: Caixeiro - Viajante

→ Vendedor precisa visitar 5 cidades e depois voltar para casa

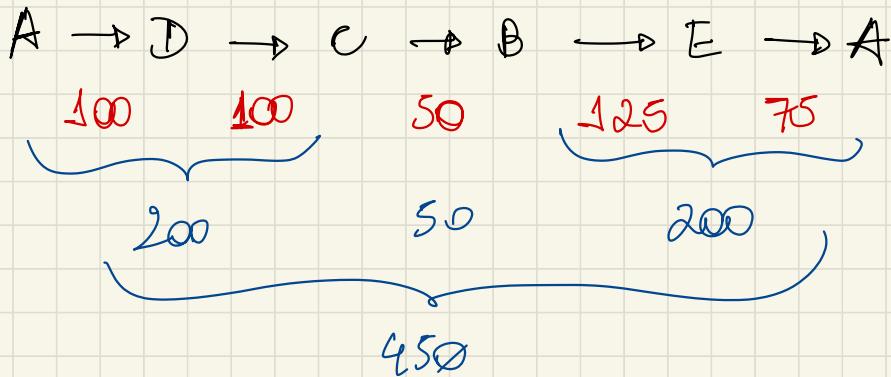
→ Objetivo: encontrar o caminho mais curto

Ex:



- exemplo de caminho:

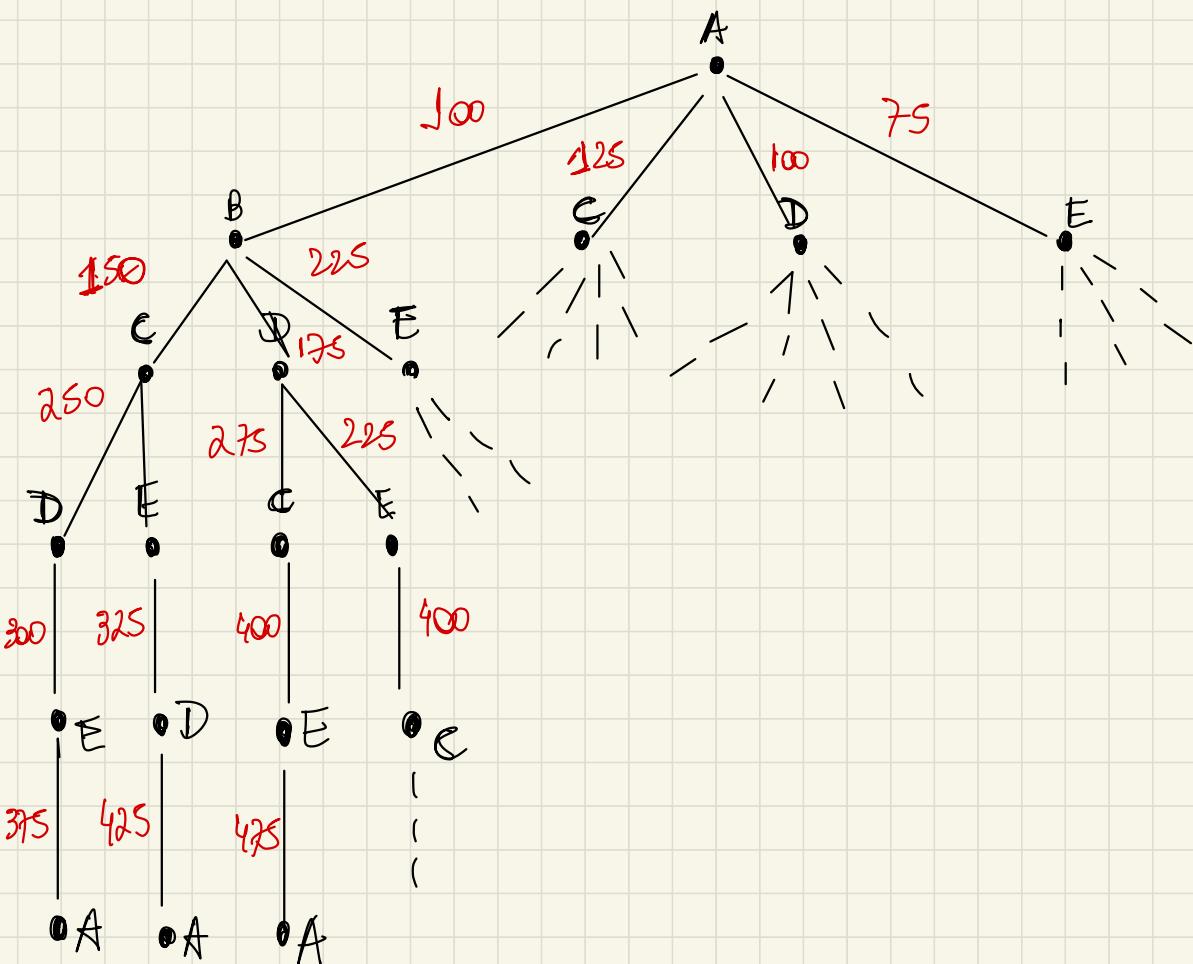
$[ADCBEA] \rightarrow$ com custo 450 Km



- a complexidade da busca exaustiva no problema do caixeiro-viajante é $(N-1)!$, onde N é o número de cidades no grafo

→ 50 cidades gera um grafo onde a busca exaustiva fica impraticável

Exemplo de busca para o grafo anterior:



↳ Caminho $[A, B, D, C, E, A]$, custo: 475

↳ Caminho $[A, B, C, E, D, A]$, custo: 425

↳ Caminho: $[A, B, C, D, E, A]$, custo: 375

• diferentes estratégias para reduzir a complexidade da busca

- * Ramificação e Poda (branch and bound)
- * Gulosa

B
V
S
C
A
S

① Algum percurso

busca com profundidade
subida de encosta
busca em amplitude
busca em feixe
busca pela melhor
escolha

② percurso ótimo

mesmo britânico
ramificar e ligar
Programação Dinâmica
 A^*

③ jogo

minimax
aprofundamento progressivo
fada alfa-beta

* 3. Estratégias para busca em espaço de estados

→ busca guiada por dados x busca guiada por objetivo

- podemos explorar o espaço de estados em duas direções:

dados → objetivo

objetivo → dados

- A decisão de entre a abordagem guiada por dados e a guiada por objetivo é baseada na estrutura do problema a ser resolvido.

Fig 1. Espaço de estados da busca guiada por objetivos

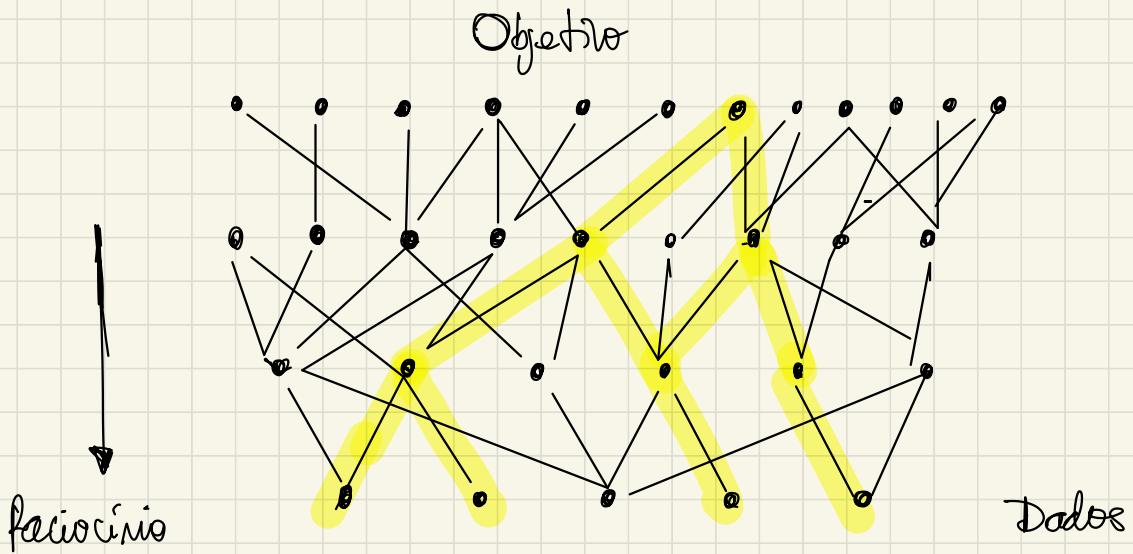
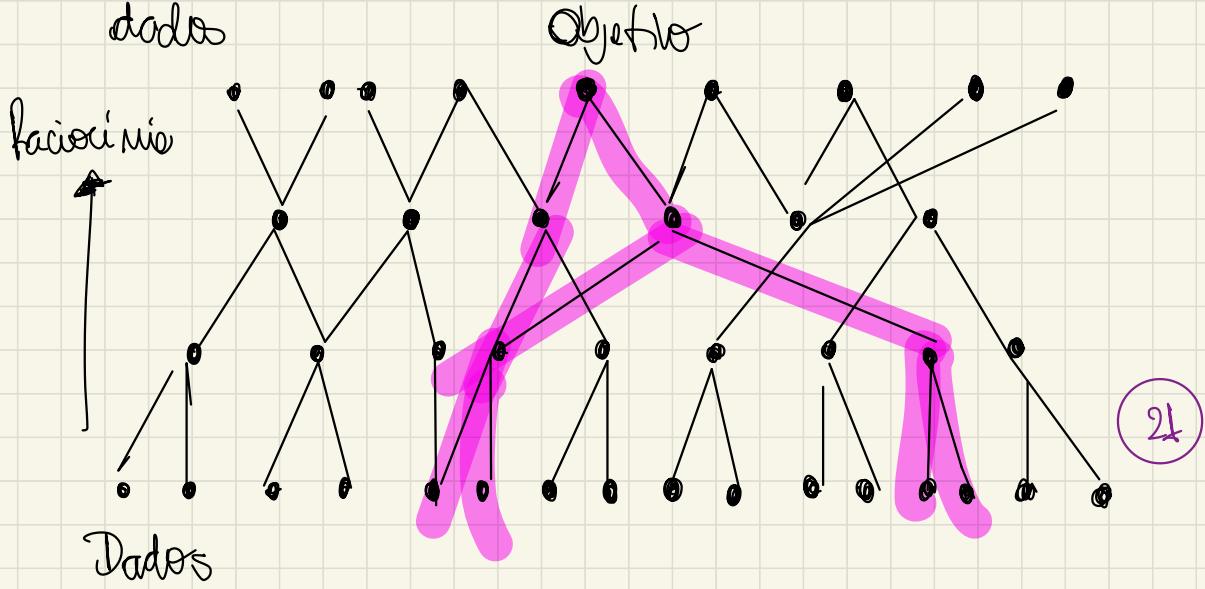


Fig 2 . Espaço de estados da busca guiada por dados



- A busca por objetivo é sugerida quando:
 1. Um objetivo é dado na formulação do problema ou pode ser facilmente formulado
 2. Existe um grande número de regras que se aplicam aos fatos do problema, produzindo um número crescente de conclusões ou objetivos
 3. Os dados do problema não são fornecidos, mas devem ser adquiridos pelo sistema para a resolução de um problema.

- A busca guiada por dados é apropriada quando:
 1. Todos os dados (ou a maioria) são fornecidos na formulação do problema
 2. Existe um grande conjunto de objetivos com potencial, mas há poucas formas de manipular os dados

3. É difícil formular um objetivo ou hipóteses

• Implementando a busca de grafo

↳ resolvidor deve achar um caminho de um estado inicial a um objetivo através do grafo do espaço de estados.

↳ um resolvidor precisa considerar diferentes caminhos pelo espaço até achar um objetivo. A busca com retrocesso (backtracking) é uma técnica para tentar sistematicamente todos os caminhos para um espaço de estados.

→ a busca com retrocesso começa no estado inicial e segue por um caminho até atingir um objetivo ou "beço sem saída"



→ ela "refrigera" até o nó mais recente no caminho que contenha irmãos não examinados e continua por um desses caminhos

ALGORITMO : BACKTRACKING

1. Se o estado atual S não atender os requisitos da descrição do objetivo, então:

gere seu primeiro descendente S_{filho1} e aplique o procedimento de retrocesso recursivamente a este nó.

2. Se o retrocesso não achar um nó de objetivo no subgrafo radicado em S_{filho1} , repita o procedimento para o seu irmão S_{filho2} .

3. Continue até que algum descendente de um filho seja um nó objetivo, ou até que todos os filhos tenham sido buscados.

4. Se nenhum dos filhos de S levar a um objetivo, então o algoritmo "falta" para o nó inicial S.

↳ o algoritmo continua até encontrar um objetivo ou esgotar o espaço de estados

↳ um algoritmo que realiza um retrocesso, usando três listas para acompanhar os nós no espaço de estados:

Lc: lista de estados - lista os estados no caminho atual sendo experimentado

LNE: lista de novos estados — contém os nós que aguardam avaliação, nós cujos descendentes ainda não foram gerados e buscados

BSS: beco sem saída, lista os estados cujos descendentes não contém objetivos. Se esses estados forem encontrados novamente, eles serão desconsiderados imediatamente

→ na definição do algoritmo de busca com retrocesso geral, é necessário detectar múltiplas ocorrências de qualquer estado, de modo que ele não seja reentrado e cause laços no caminho.

- isso é realizado testando se cada estado gerado-pertence a qual

que uma das listas. Se um novo endereço pertencer a qualquer uma das listas, então ele já terá sido visitado e poderá ser ignorado.

BUSCA COM RETRÔCESSO

INÍCIO

$LE = [Início]$; // Inicialização

$LNE = [Início]$;

$BSS = []$;

$EC = Início$; // estado corrente

Enquanto $LNE \neq []$ faça:

se $EC = \text{objetivo}$ (ou atende descrição do objetivo)

então retorna LE ;

// se houver sucesso, retorna lista de estados no caminho

Se EC não tem filhos (excluindo os nós já em BSS, LE, LNE)
então:

enquanto LE não está vazio e
 $EC = \sigma$ primeiro elemento de LE
fazer:

- acrescenta EC em BSS;
- remove primeiro elemento de LE;
- remove primeiro elemento de LNE;
- $EC = \text{primeiro elemento de LNE}$

acrescenta EC a LE;

Semão:

- coloca filhos de EC em LNE
// exceto nós já em BSS, LE ou LNE
- $EC = \text{primeiro elemento de LNE}$

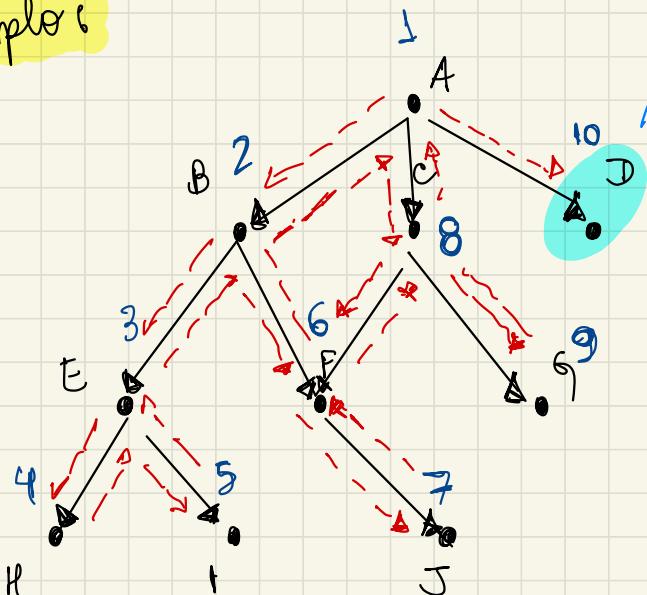
- acrescenta Ee à LE;

lín algortmo
retorna FALHA,

lín algortmo

- na busca com retrocesso, o estado atual é identificado por EC
- as regras de inferência, regras de um jogo, ou outros operadores apropriados para solução de um problema são ordenados e aplicados ao EC. O resultado é um conjunto ordenado de estados (filhos de EC)
- Se EC não tiver filhos, ele é removido de LE (o algortmo "retrócede")

Exemplo 6



Objetivo

Obterm de visita

cada:

A, B, E, H, I, F, J

C, G, D

It	EC	LE	LNE	BSS
0	A	[A]	[A]	[]
1	B	[BA]	[BCDA]	[]
2	E	[EBA]	[EFBCDA]	[]
3	H	[HEBA]	[HIEFBCDA]	[]
4	I	[IEBA]	[IEFBCDA]	[H]
5	F	[FBA]	[FBCDA]	[EIH]
6	J	[JFBA]	[JFBCDA]	[EIH]
7	C	[CA]	[CDA]	[BFJEIH]
8	G	[GCA]	[GCDA]	[BFJEIH]

- a busca com retrocesso implementa uma busca guiada por dados, tomando a raiz como estado inicial, e avaliando os seus filhos para buscar o objetivo
- a busca com retrocesso é um algoritmo para busca em grafos de espaço de estados.
- Outros algoritmos exploram algumas de suas idéias, como:

1. uso de uma lista de estados não-processados (LNE) para permitir que o algoritmo retorne (retroceda) a qualquer um destes estados

2. uma lista de estados "guinô" (BSS) para evitar que o algoritmo tente novamente caminhos inúteis

3. Uma lista de nós (L_t) sobre o caminho da solução atual, que é retomada se um objetivo for encontrado

4. Verificação explícita da pertinência de novos estados nestas listas para evitar lacos