

Algoritmos de Busca

05/03/2022



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. Algumas 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

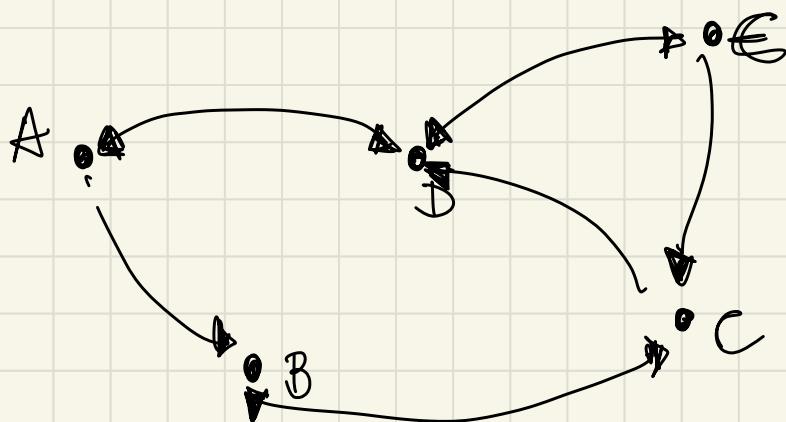
- Conjunto de nós e conjunto de arcos que conectam estes nós.
- No modelo de espaço de estados para resolução de problemas, os nós de um grafo são usados para representar estados diferentes, como diferentes configurações de um

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 arestas, 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, conexo, 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

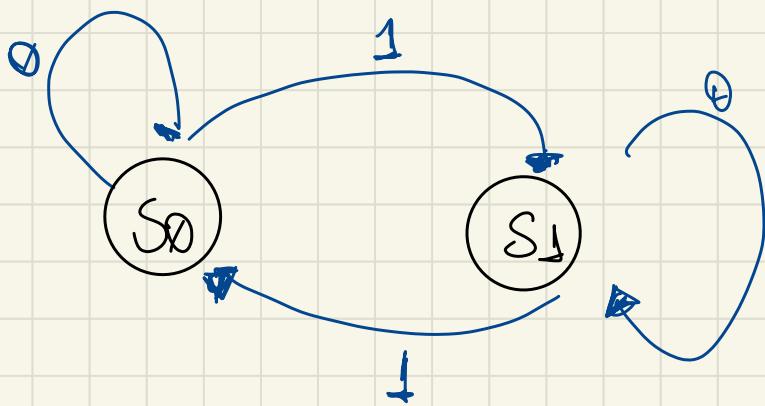
Σ : alfabeto, símbolos manipulados

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

q_0 : estado inicial

F : conjunto de estados finais

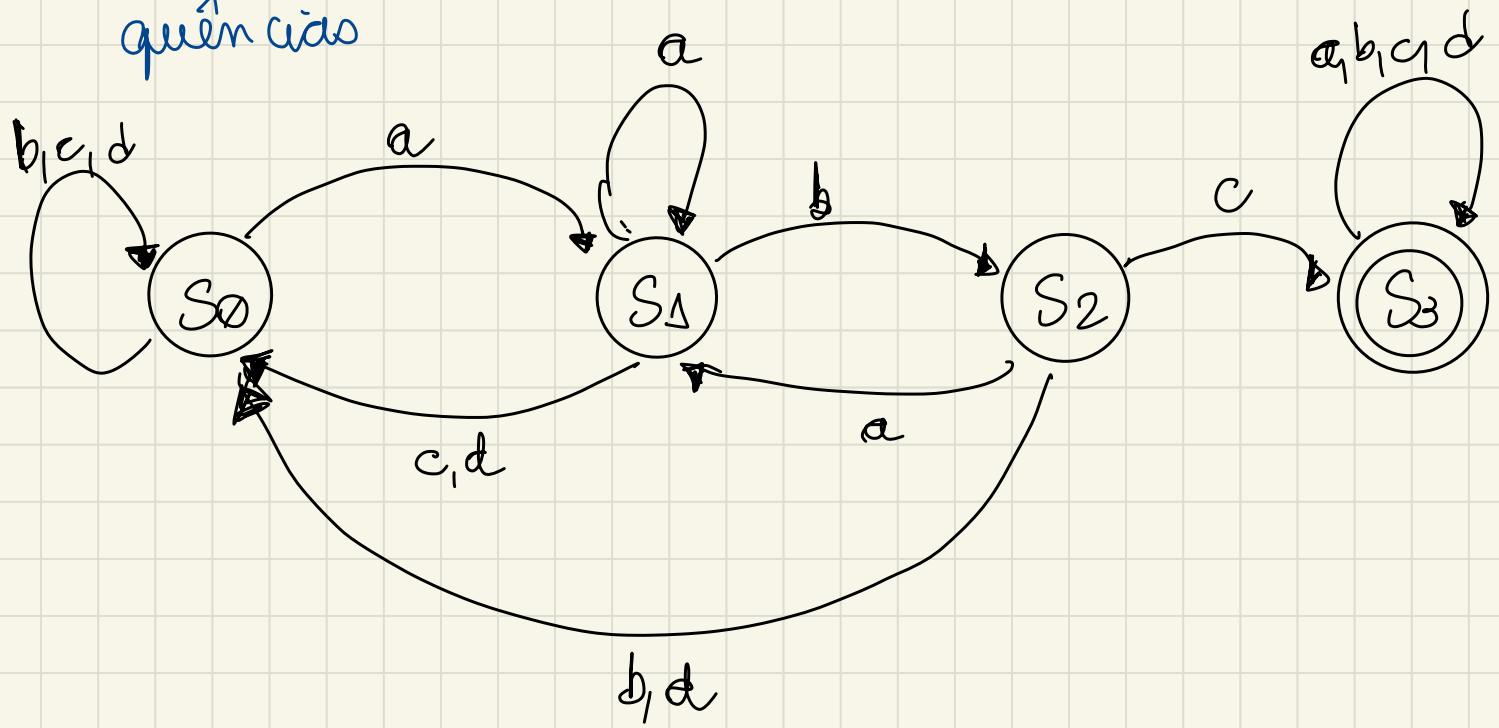
- Exemplo de MTF para um flip-flop



S	0	1
S0	S0	S1
S1	S1	S0

Q5

- Exemplo de MAF para reconhecimento de se_ quebrado



δ	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

06

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 real 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 solução 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:

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

- N : conjunto de nós/estados do grafo
 - A : arcos/arestas/transições. Passos de um processo de solução de problema
 - I : estado(s) inicial(is) do problema
 - D : estado(s) objetivo(s) do problema
- * Um caminho de solução é um caminho através de um nó em I para um nó em D .

Exemplo qt: Jogo da velha

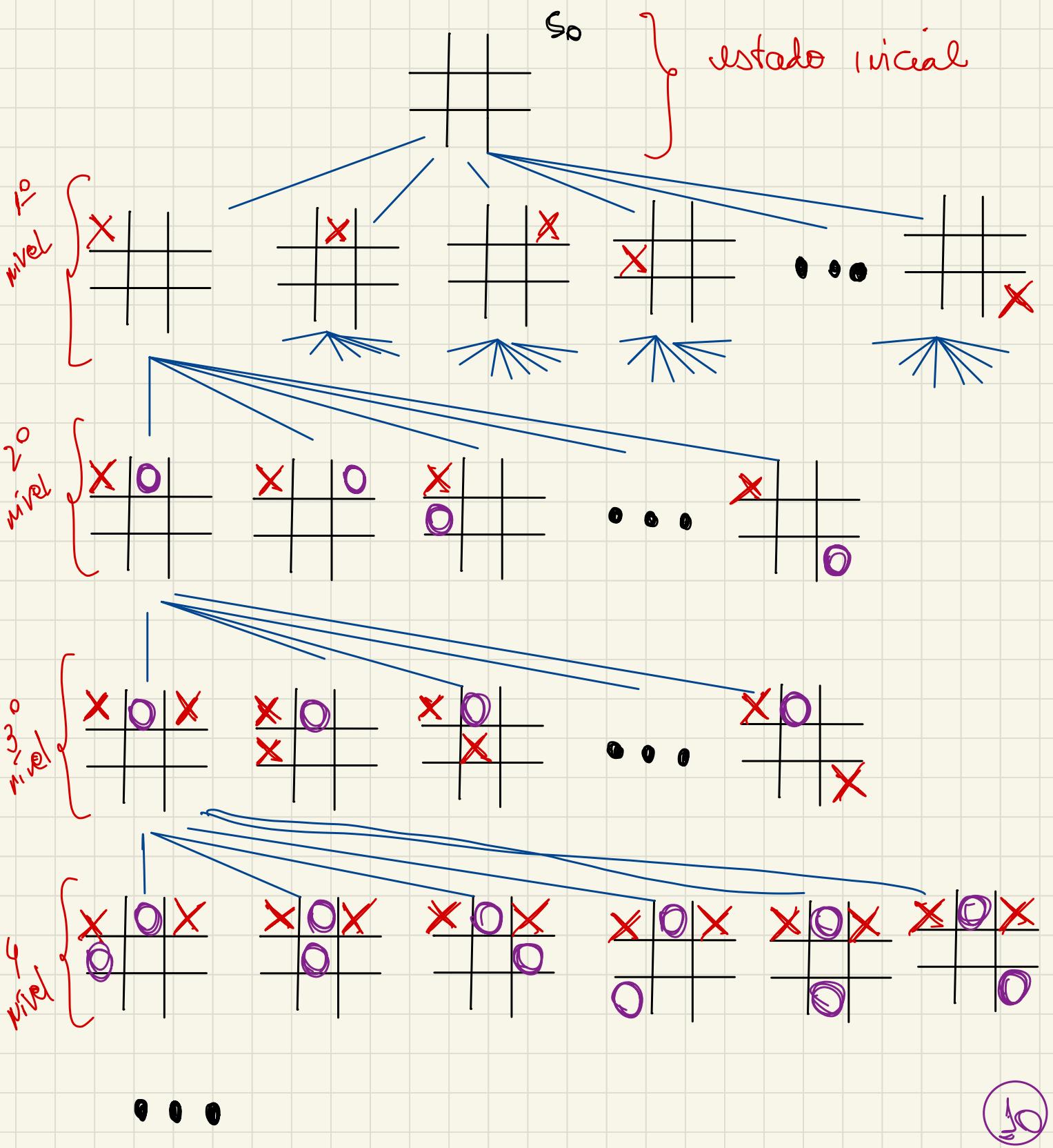
Estado inicial = tabuleiro vazio

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 nunca ocorrem em um jogo real

! espaços difíceis ou impossíveis de ser 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

metendo
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

à Esq

à dir

	1	3
7	4	6
5	8	2

1	3	
7	4	6
5	8	2

...

- Objetivo:

1	2	3
8		4
7	6	5

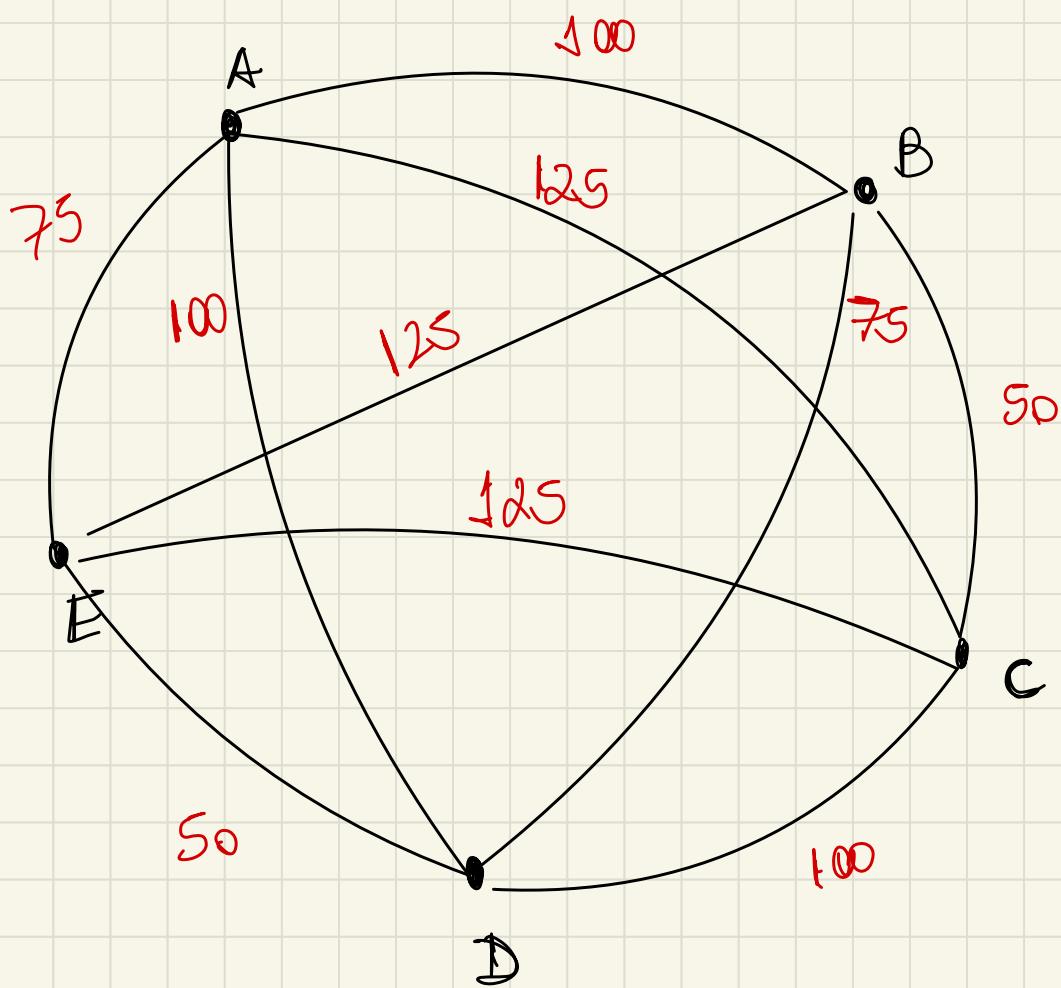
de

1	2	3
4	5	6
7	8	

Exemplo 03: Caixiero - viajante

- Vendedor precisa visitar 5 cidades e depois voltar para casa
- Objetivo: encontrar o caminho mais curto

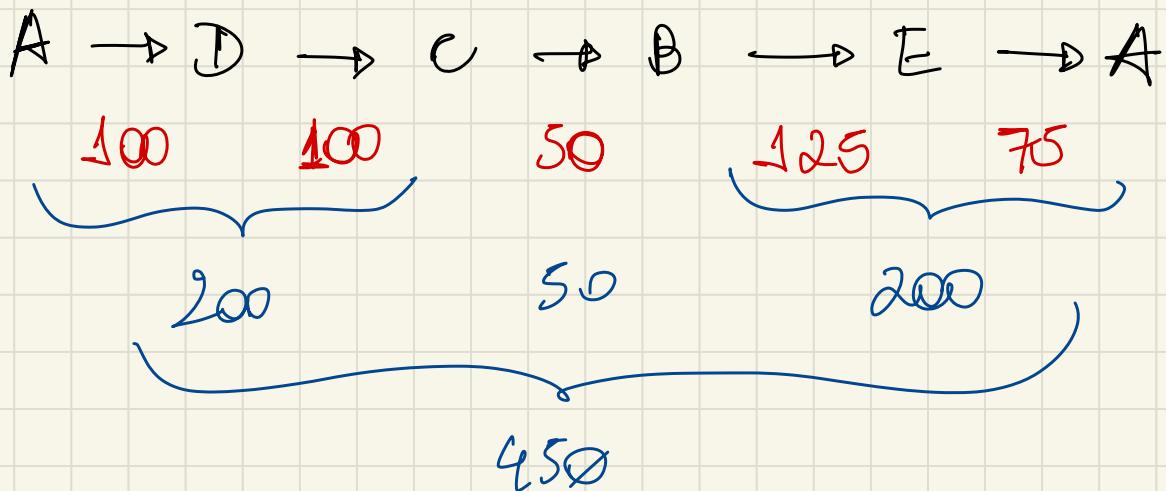
Ex:



15

- exemplo de caminho:

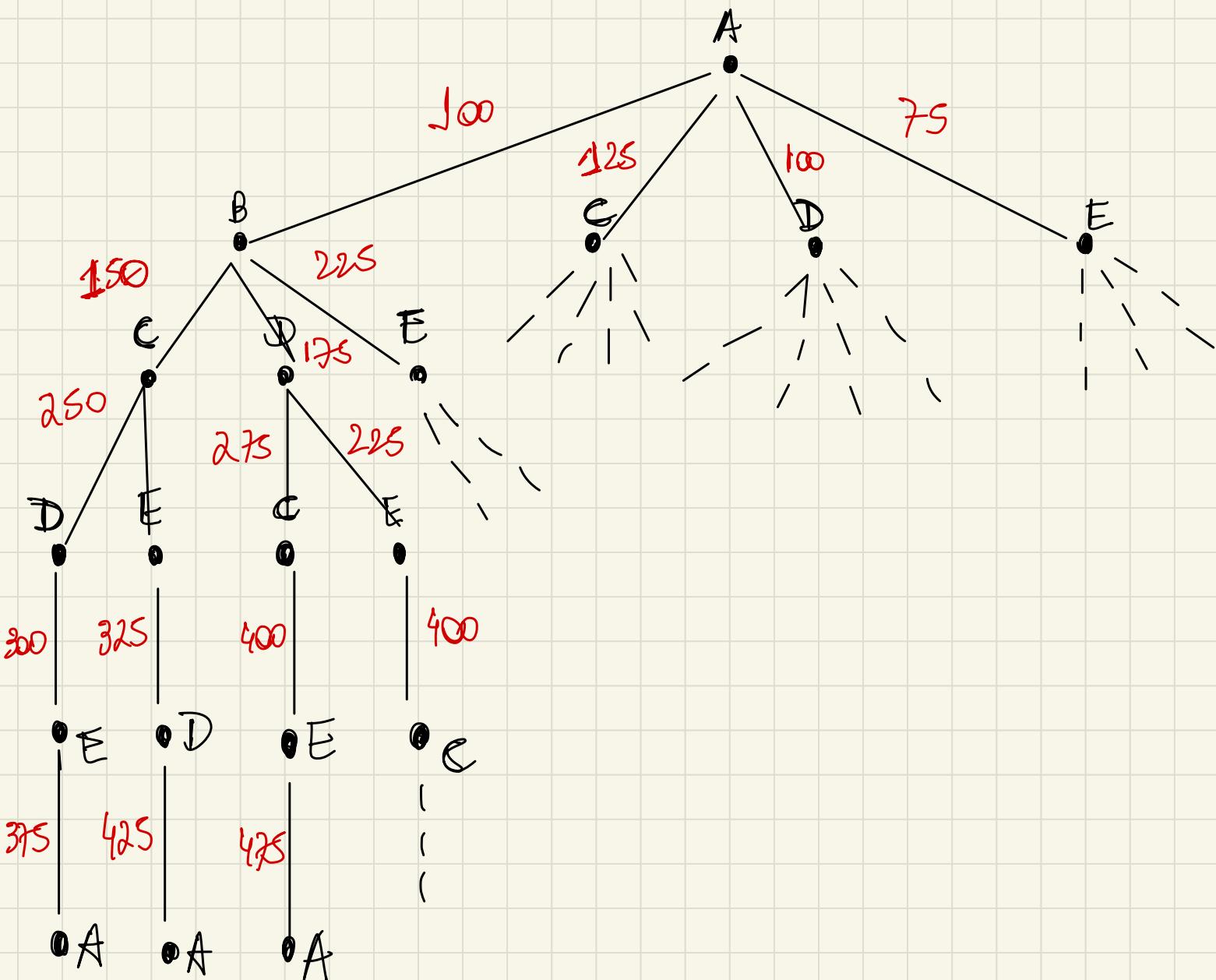
$[ADCBEA] \rightarrow$ com custo 450 Km



- 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
escoha

② percorre ótimo

Museu britânico
tornificar e ligar
Programação Dinâmica
A*

③ jogos

minimax
aprofundamento progressivo
toda 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 quicada por objetivos

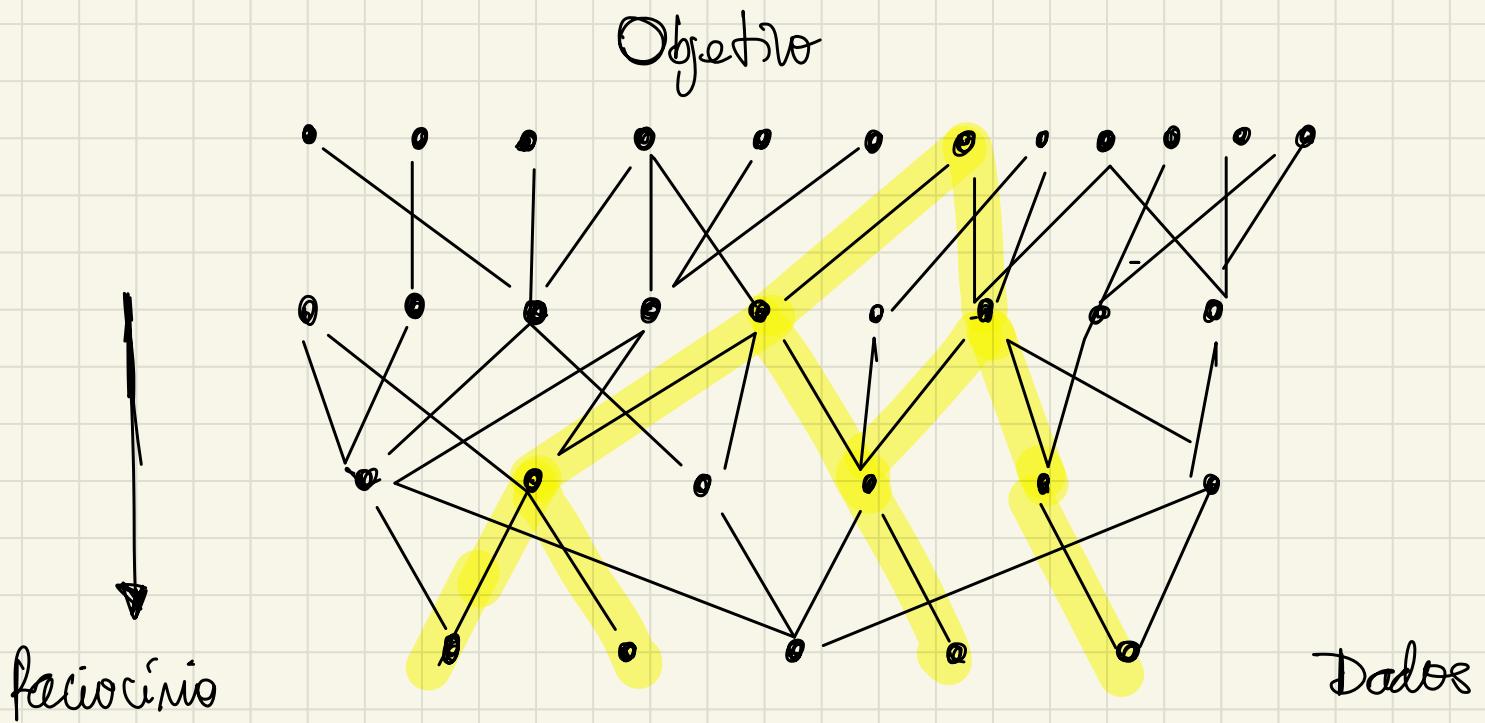
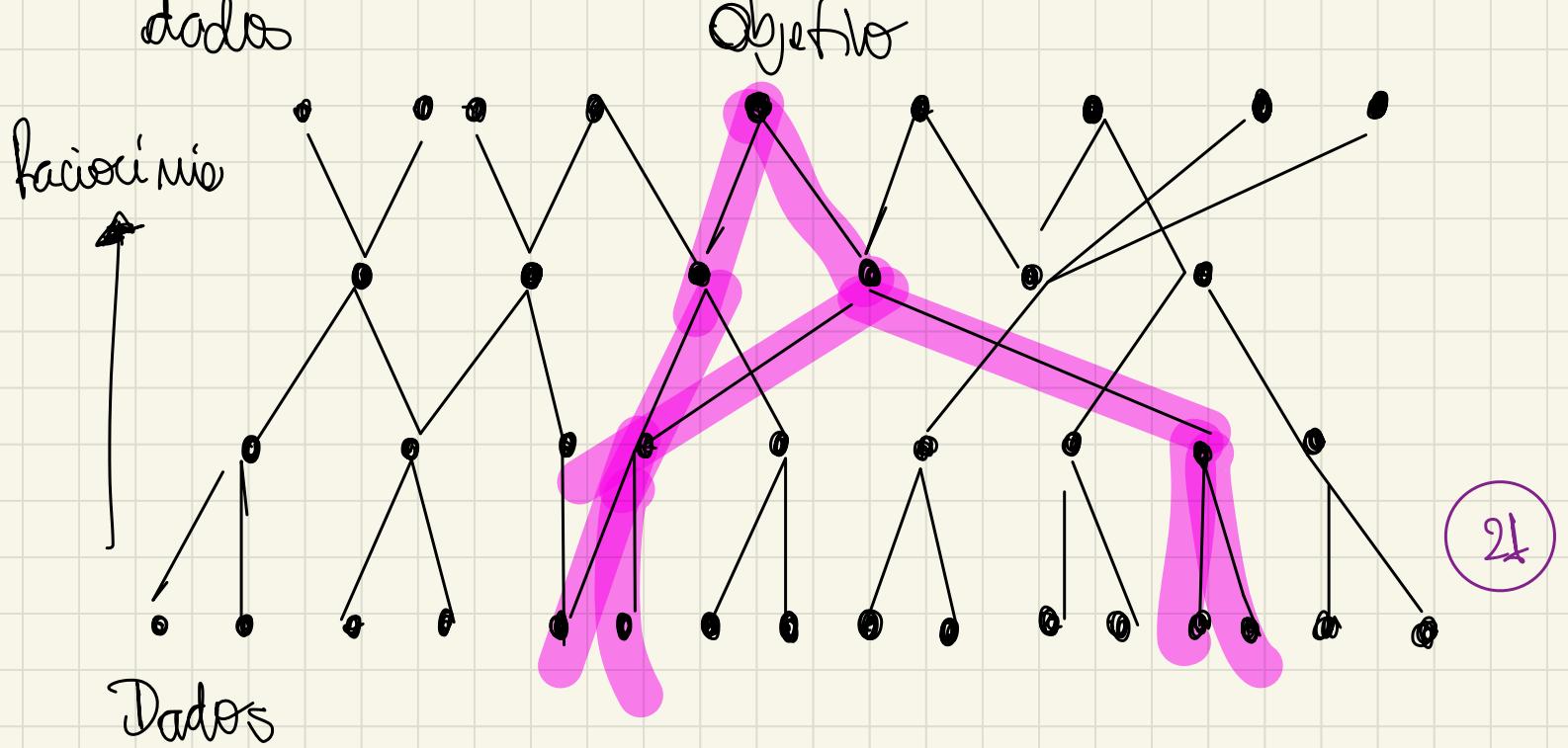


Fig 2 . Espaço de estados da busca quicada por dados



- 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 em potencial, mas há poucas formas de manipular os dados

3. É difícil formular um objetivo ou hipóteses

• Implementando a busca de grafo

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

↳ um resolvelor 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 "refrigide" 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 "falha" para o nó inicial S.

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

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

Le : 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 busca-dos

BSS: bco sem saída, lista os estados cujos descendentes não contém objetivo. 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 reem-gerado pertence a qual

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

BUSCA COM RETROCESSO

INÍCIO

1. $LE = [\text{Início}] ;$ // Inicialização
2. $LNE = [\text{Início}] ;$
3. $BSS = [] .$
4. $EC = \text{Início} ;$ // estado corrente
5. Enquanto $LNE \neq []$ faça:
 6. se $EC = \text{Objetivo}$ (ou atende descrição do objetivo)
|
7. | então reforma $LE ;$
| // se houver sucesso, reforma lista de estados no caminho

8.

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

então:

9.

enquanto LE não está vazio e
EC = o primeiro elemento de LE
faça:

10.

- acrescenta EC em BSS;
- remove primeiro elemento de LE;
- remove primeiro elemento de LNE;
- EC = primeiro elemento de LNE

11.

acrescenta EC a LE;

12.

Sendo:

13.

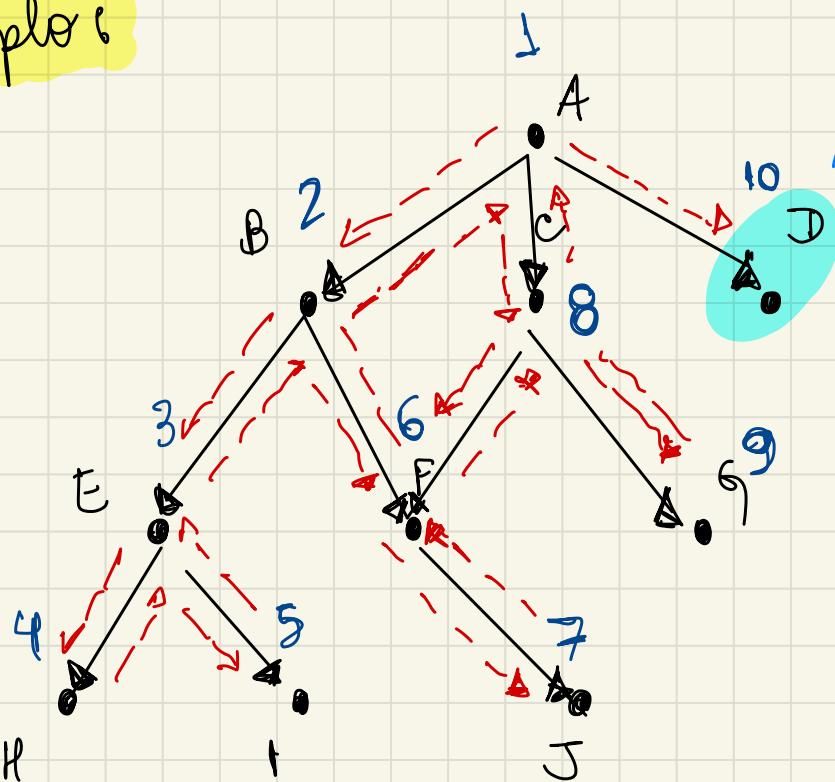
- coloca filhos de EC em LNE
// exceto nós já em BSS, LE ou LNE

14.

- EC = primeiro elemento de LNE

18. | | • acrescenta Ec a LE;
19. | | fim enquanto
20. | | retorna FALHA,
21. | | fim algoritmo
- 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 algoritmo "retraseca")

Exemplo 6



Objetivo

Ordem de visita

côde:

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]	[GCDAB]	[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 ideias, 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 "guinchados" (BSS) para encorajar que o algoritmo tente novamente caminhos inúteis

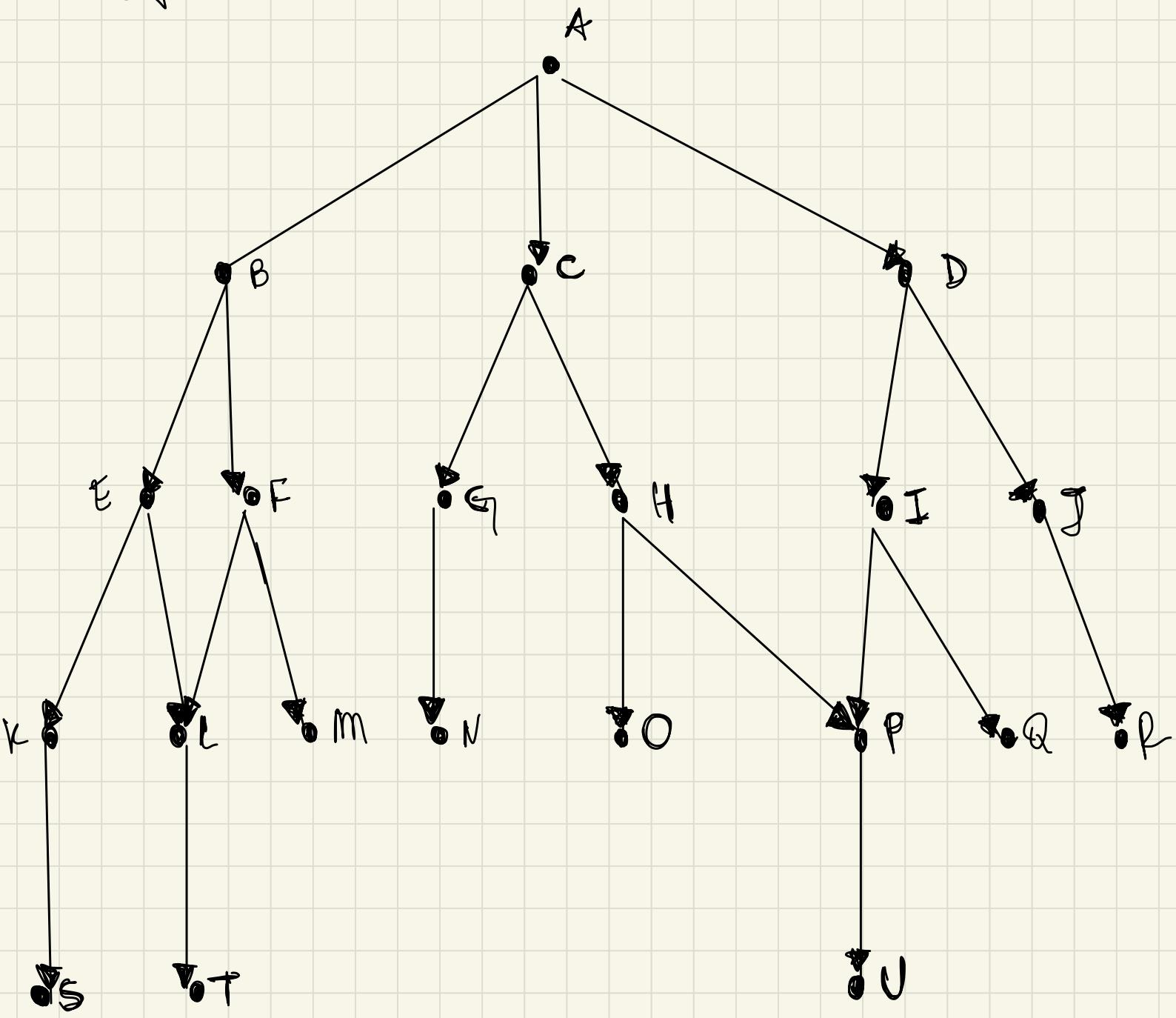
3. Uma lista de nós (LE) sobre o caminho da solução atual, que é retornada se um objetivo for encontrado

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

04. Busca em profundidade e busca em Amplitude

- além de especificar uma direção de busca, um algoritmo de busca deve determinar a ordem na qual os estados são examinados
- vamos ver duas possibilidades
 - busca em profundidade
 - = busca em largura
- Considere o grafo da Figura X.1. Os estados são rotulados (A, B, C, D, \dots) de modo que possam ser referenciados na discussão que segue.

Fig X.1 - Grafo para os exemplos de Bfs e Dfs



• na busca em profundidade, quando um estado é examinado; todos os seus filhos e os descendentes deles são examinados antes de qualquer um dos irmãos.

→ avança a busca se profundando no espaço de estados sempre que possível

→ apenas quando não forem encontrados mais descendentes de um estado é que seus irmãos serão considerados

→ a busca em profundidade (dfs) examina os estados do grafo da figura X. 1 na ordem: A, B, E, K, S, L, T, F, M, C, G, N, H, O, P, U, D, I, Q, J, R.

→ o algoritmo de busca com retrocesso já apresentado, implementa uma dfs

• Já a busca em amplitude, explora o espaço nível por nível.

→ afinal quando não houver mais estados a serem explorados em um determinado nível é que o algoritmo se move para o próximo estado mais profundo.

→ Uma busca em largura (bfs) 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

→ implementamos a busca em amplitude usando listas (**abertos**, **fechados**) para registrar o progresso através do espaço de estados

Lista abertos: lista os estados que foram gerados, mas cujos filhos não foram examinados

A ordem segundo a qual os estados são removidos de **abertos** determina a ordem da busca

Lista Fechados: registra os estados que já foram examinados e corresponde à união das listas BSS e LE do algoritmo de busca com retrocesso.

ALGORITMO: BUSCA EM AMPLITUDE

BFS (Início)

1. **Abertos** = [Inicial] // é uma Fila
2. **Fechados** = [] // Inicializado
3. ENQUANTO **Abertos** != [] faça :
4. | Remova o estado mais à esquerda
 | de **Abertos**, chame-o de X

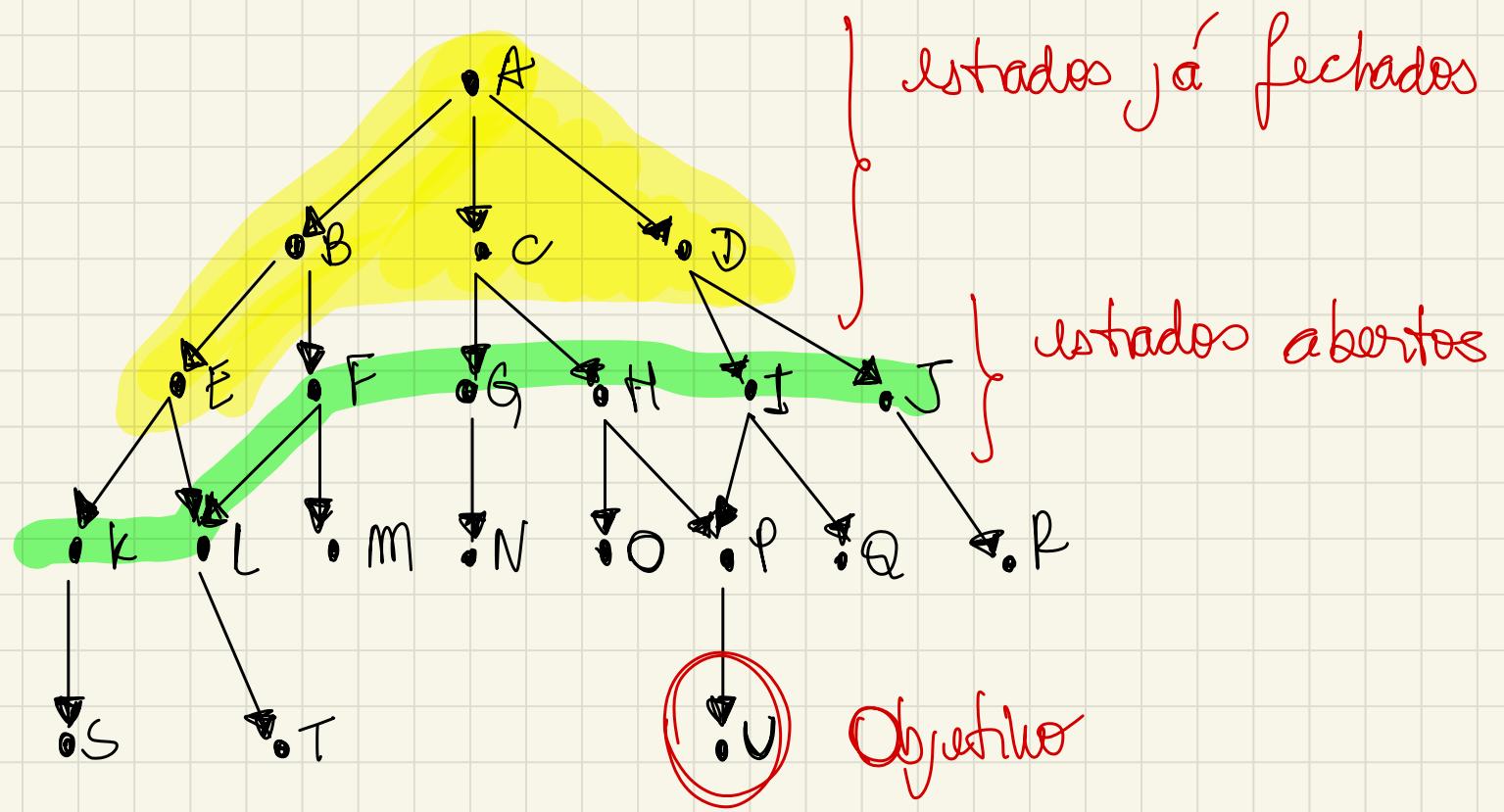
5. Se X for um objetivo, então
6. | retorna **SUCESSO** // objetivo encontrado
7. Senão
8. | • gera filhos de X
9. | • coloque X em **Fechados**
10. | • descarte filhos de X se já estiverem
| em **ABERTOS** de **Fechados**
11. | • coloque os filhos que restam no
| final à direita de **ABERTOS**
12. | fime se
13. | fime loop
14. | retorna **FALHA** // não restam estados
15. | fime Bfs

- Note que a lista **Abertos** é mantida como uma **fila**. Os estados são adicionados à direita e removidos da extremidade esquerda.

→ Isto orienta a busca em direção aos estados que permanecem em **Abertos** por mais tempo, fazendo com que a busca se dê em amplitude.

- Os estados filhos que já foram descobertos (já aparecem em **Abertos** ou **Fechados**) são descartados.
- Frequentemente é útil manter outras informações além dos nomes dos estados. Note que Bfs não mantém uma lista de estados sobre o caminho atual até um objetivo. A solução pode ser manter uma informação de seu pai junto com cada estado.

* Fig. Exemplo de execução de Bfs



Exemplo de caminho gerado pela busca na quarta iteração:

Abertos: $[(D, A), (E, B), (F, B), (G, C), (H, C)]$

Fechados: $[(C, A), (B, A), (A, \text{nada})]$ estado inicial

O caminho (A, B, F) pode ser facilmente criado com essa informação

• **Exercício:** Implementar Bfs para o quebra-cabeça de 8 peças:

2	8	3
1	6	4
7		5

estado
inicial

1	2	3
8		4
7	6	5

estado
objetivo

Obs: se tudo estiver certo, seu algoritmo de busca em largura vai precisar de 46 iterações para solucionar o problema

* Teste depois com outros estados iniciais

• A seguir criamos um algoritmo de busca em profundidade, uma simplificação do algoritmo de retrocesso.

↳ os estados descendentes são adicionados e removidos a partir do final esquerdo de Abertos, que já mantida como uma **Filha**

↳ esta organização direciona a busca para estados gerados mais recentemente, produzindo uma ordem de busca que avança em profundidade

ALGORITMO : BUSCA EM PROFUNDIDADE (DFS)

DFS (Início)

1. **Abertos** = [Início] // Inicializa
2. **Fechados** = []

3. Enquanto Abertos ≠ [] faça // gesto em estados
4. • Remova o estado mais à esquerda em ABERTOS, chame-o de X
5. Se X é um objetivo, então
6. retorna SUCESSO // objetivo encontrado
7. Senão
8. • gera filhos de X
9. • coloque X em fechados
10. • descarte filhos de X que já estão em ABERTOS ou FECHADOS
11. • coloque os filhos que restaram no final esquerdo de ABERTOS
12. fim se
13. fim enqto
14. retorna FALHA // não geraram mais estados
15. fim algoritmo

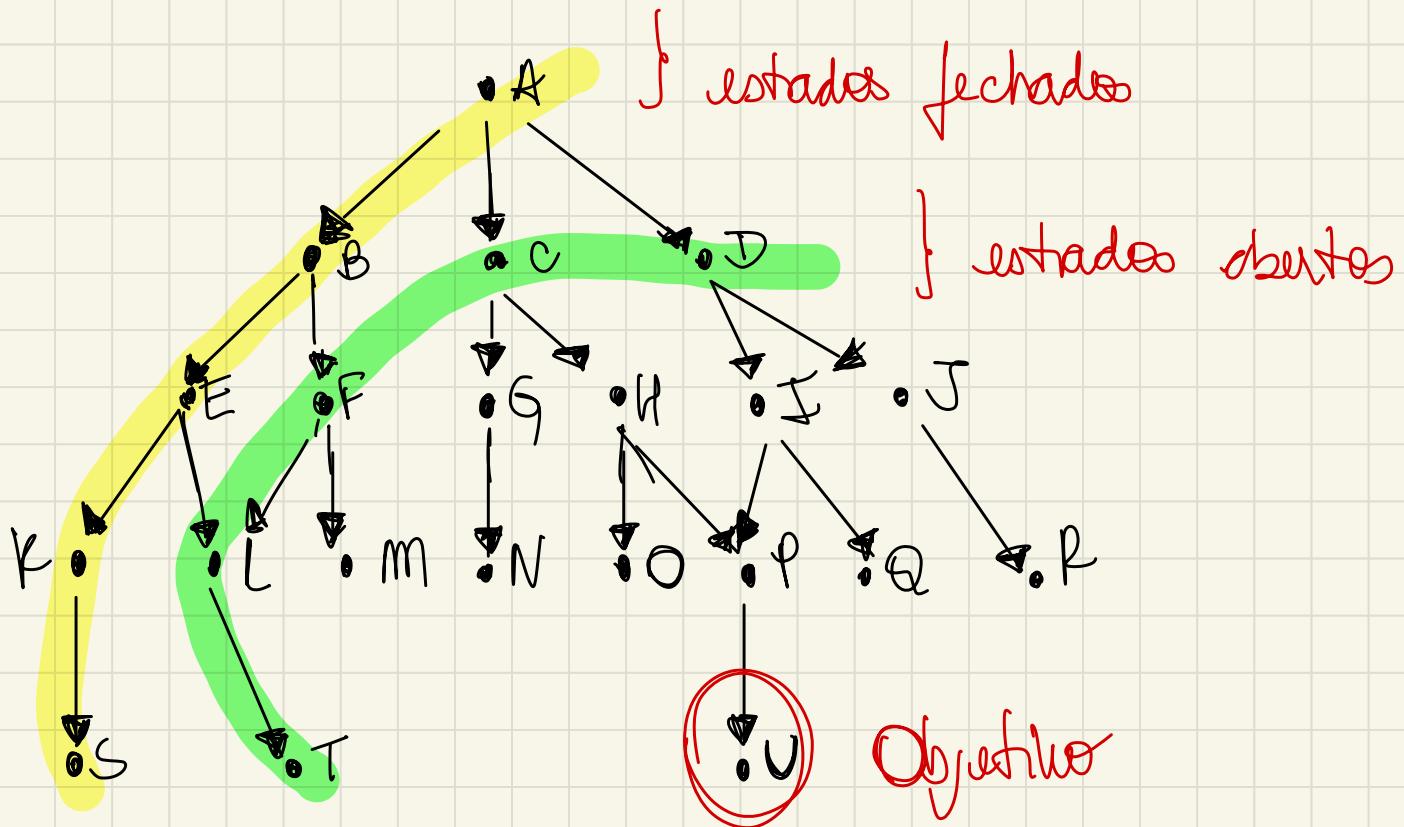


Fig. Exemplo de Busca em Profundidade

- Como na bfs, a Abertos lista todos os estados descobertos mas ainda não avaliados
- diferente da bfs, a dfs não tem garantias de achar o caminho mais curto. Mais adiante na busca, um caminho diferente pode ser encontrado

* Comparativo entre Bfs e Dfs

- A escolha de qual algoritmo usar depende do problema em específico
 - características importantes incluem:
 - Importância de achar o caminho mais curto para um objetivo
 - o fator de ramificação do espaço
 - o tempo de computação
 - o tamanho médio dos caminhos para um nó de objetivos
 - se queremos todas as soluções ou apenas a primeira
- Busca em Amplitude (BFS)
 - sempre acha o caminho mais curto até um nó objetivo

→ em problemas de solução simples, sabe-se que essa solução será encontrada

→ se houver um alto fator de ramificação, pode "explodir" a memória com o alto número de estados abertos na memória

- Se cada estado tiver B filhos, o número de estados em nível n é B^n
- Impraticável para problemas como o xadrez

• Busca em Profundidade (Dfs)

→ chega rapidamente a um espaço de busca profundo. Se é sabido que o caminho da solução é longo, não desperdice tempo buscando estados superficiais

→ por outro lado, pode-se perder na profundidade

dade

→ é muito mais eficiente para espaços de busca com muitas ramificações. O uso do espaço de busca é uma função linear do tamanho do caminho (em cada nível, a lista Abertos contém apenas os filhos de um único estado)

- se um grafo tem em média B filhos por estado, isso requer $B \times n$ estados para avançar com profundidade de n níveis no espaço
- Para o jogo de xadrez, a busca em amplitude simplesmente não é possível. Em jogos mais simples, não só pode ser possível, como também a única maneira de evitar a derrota, já que reforma o melhor caminho.

• **Exercício 02:** Implementar Dfs para o quebra-cabeça de 8 peças:

2	8	3
1	6	4
7	█	5

estado
inicial

1	2	3
8	█	4
7	6	5

estado
objetivo

Obs: implemente um controle de recursão para que a busca execute com uma profundidade limitada a 5 (5 níveis a partir do nível 0)

Obs 2: se tudo estiver certo, seu algoritmo de busca em largura vai precisar de 31 iterações para solucionar o problema

* Teste depois com outros estados iniciais