1

Busca não informada

Capítulo 3
Parte II

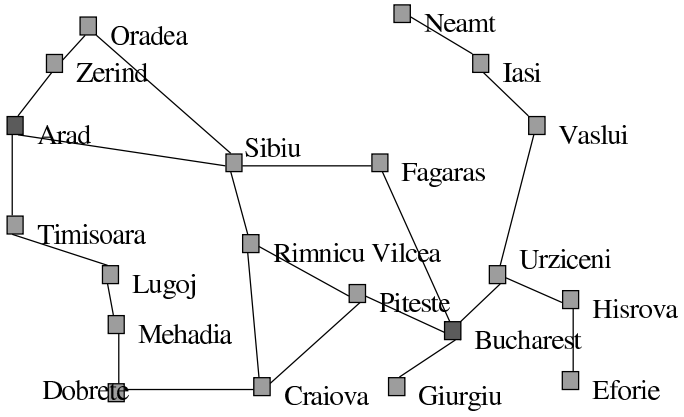
2

O que um algoritmo de busca faz:

- gera sequência de ações/gera conjunto de estados (expande nós)
- múltiplas escolhas de caminho
- essência de busca: escolher uma opção e deixar outras de lado p/ mais tarde
  (divide-and-conquer ???)
- a decisão de quem escolher primeiro define uma estratégia de busca

3

Espaço de estado → 20 nós
Árvore de busca → infinitos nós



4

Estrutura de dados para árvores de busca

- Nó → formado por 5 componentes:
  - estado
  - nó pai
  - operador (usado para gerá-lo)
  - profundidade
  - custo do caminho

  função *expande*: gera os 5 componentes de um nó
- Lista que guarda os nós a serem expandidos

  OPEN: lista que guarda os nós a serem expandidos

  + CLOSED: lista que guarda os nós já expandidos

5

Algorítmo de busca geral

função Busca-Geral(*problema*, *estratégia*) **retorna** uma solução, ou falha

inicializa a árvore de busca com o estado inicial do *problema*

laço faça

se não há mais candidatos para expandir **então retorna** falha

escolha um nó folha para expandir

se o nó contém um estado gol **então retorna** a solução

senão expanda o nó de acordo com a *estratégia*

fim

Obs: *estratégia* = QUEUING-FN

6

Estratégias de busca

- A estratégia de busca é definida pela ordem em que nós são expandidos
- Estratégias são avaliadas através das seguintes dimensões:
  - completeza (encontra a solução se existir uma)
  - complexidade de tempo (número de nós gerados e expandidos)
  - complexidade de espaço (número de nós na memória)
  - solução ótima (encontra a solução de menor custo)
- Complexidade é medida em termos de:
  - *b* - fator de ramificação da árvore de busca
  - *d* - profundidade da solução de menor custo
  - *m* - profundidade máxima do espaço de busca

7

Estratégias de busca não-informada

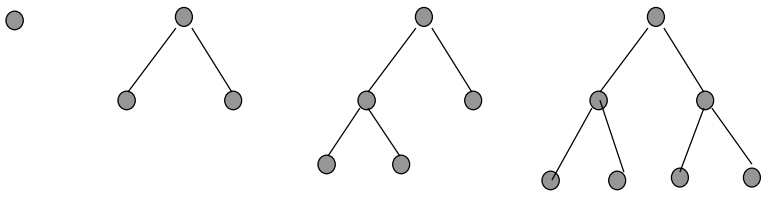
- BUSCA CEGA ou FORÇA BRUTA: não usa heurística. Variam quanto a ordem em que nós são expandidos:
  - Busca em largura - *Breadth first*
  - Busca de custo uniforme
  - Busca em profundidade - *Depth first*
  - Busca em profundidade limitada
  - Busca em profundidade iterativa
  - Busca bidirecional

Não possuem informação sobre a distância do nó e o goal.

8

Busca em largura

QUEUING-FN at-end



9

Propriedades da busca em largura

- **Completa?** Sim. Garante encontrar a solução (se existir uma)
- **Solução ótima?** Sempre encontra primeiro a solução mais rasa. Solução ótima primeiro se a função custo for não-decrescente com a profundidade do nó.
- **Tempo?** $1 + b + b^2 + b^3 + ... + b^d = O(b^d)$
- **Espaço?** $O(b^d)$ (guarda todos os nós na memória)

10

Busca em largura

Prof.	Nós	Tempo	Memória
0	1	1 ms	100 bytes
2	111	.1 s	11 Kbytes
4	11.111	11 s	1 Mbytes
6	10^6	18 min	111 Mbytes
8	10^8	31 hs	11 Gbytes
10	10^{10}	128 dias	1 Tbytes
12	10^{12}	35 anos	111 Tbytes
14	10^{14}	3500 anos	11.111 Tbytes

B=10 1000 nós/seg 100 bytes/nó

11

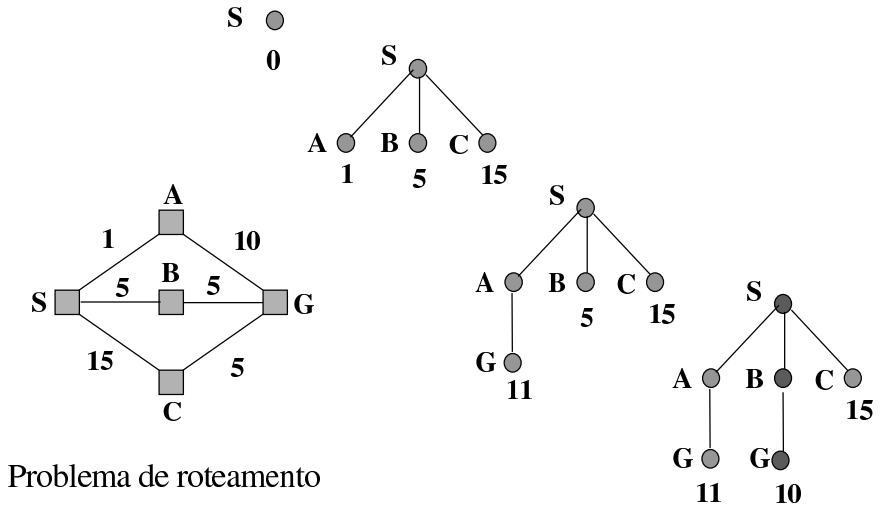
Busca em largura

- Desvantagens:
  - complexidade exponencial: $O(b^d)$
  - problema de memória maior que o de tempo
- Vantagens:
  - garante encontrar a solução
  - encontra a solução mais próxima da raiz

    g(n) = Profundidade(n)

12

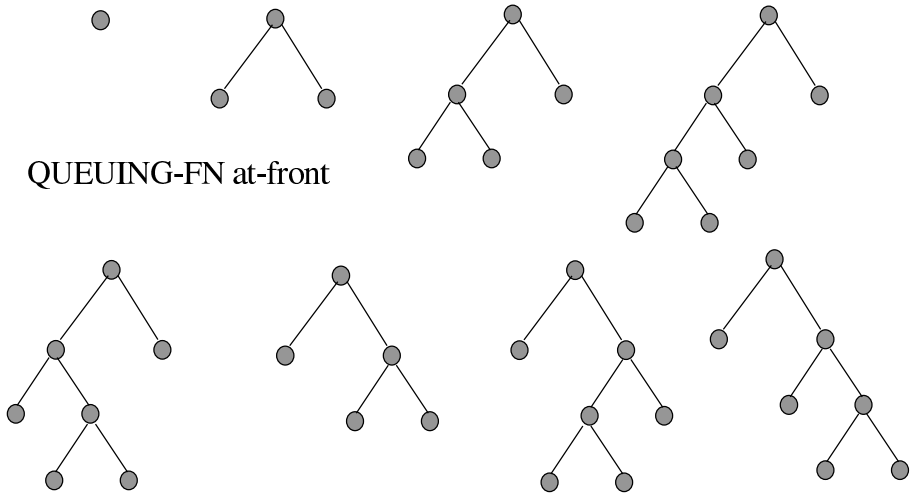
Busca de custo uniforme



Problema de roteamento

13

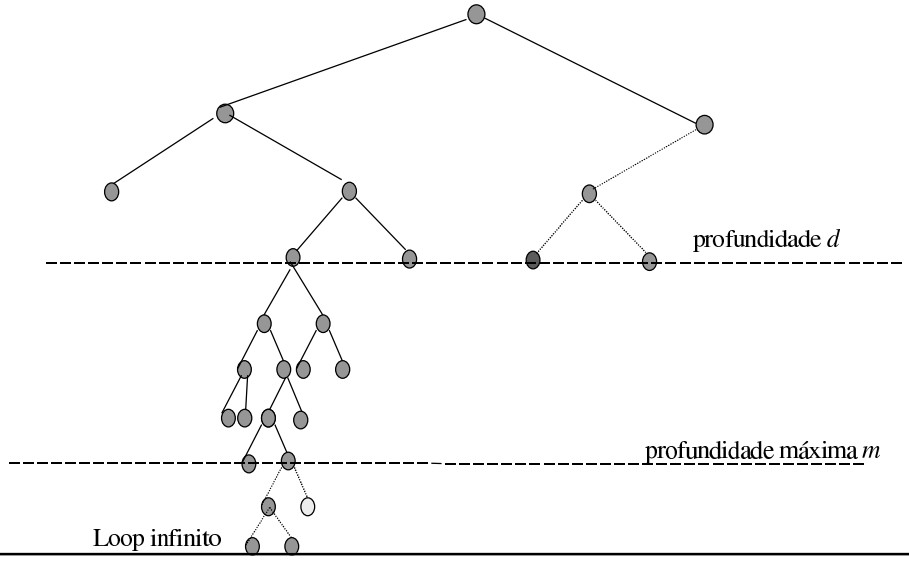
Busca em profundidade (DFS)



Expande os nós de profundidade maior / b*m nós armazenados

14

Busca em profundidade



15

Propriedades da busca em profundidade

- **Completa?** Não. Falha em espaços com profundidade infinita, ou com loops. Se modificado para evitar estados repetidos *é completo em espaços finitos*
- **Tempo?** $O(b^m)$*:* ruim se m é muito maior que d
- **Espaço?** $O(bm)$ linear com m
- **Solução ótima?** Não

DFS pode realizar ciclos infinitos. Requer um espaço de busca finito e não cíclico (cheque por estados repetidos)

16

Busca em profundidade (DFS)

- Expande os nós de profundidade maior
  - requer pouca memória (*bm versus* b^m)

Busca	**Prof.**	**Nós**	**Memória**
BFS	**12**	**10^{12}**	**111 Tbytes**
DFS	**12**	**120**	**12 Kbytes**

  - não garante encontrar a solução ótima.
  - não recomendado para grandes árvores de busca ou quando a profundidade máxima é desconhecida.

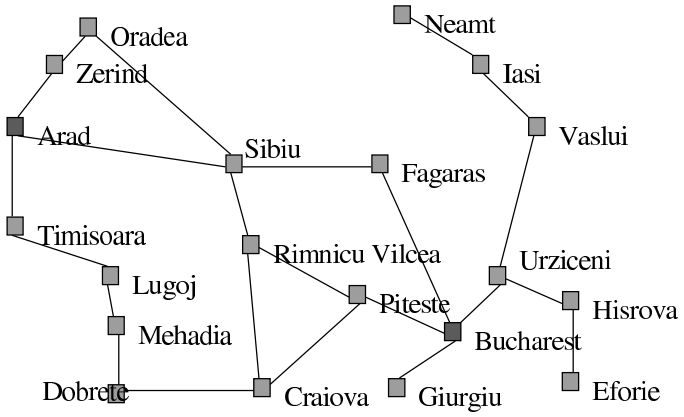
17

Busca em profundidade limitada

- Exemplo de roteamento no mapa da Romenia: caminho máximo: passar pelas 20 cidades
- Mesma complexidade da DFS
- **Tempo?** $O(b^l)$
- **Espaço?** $O(bl)$

18

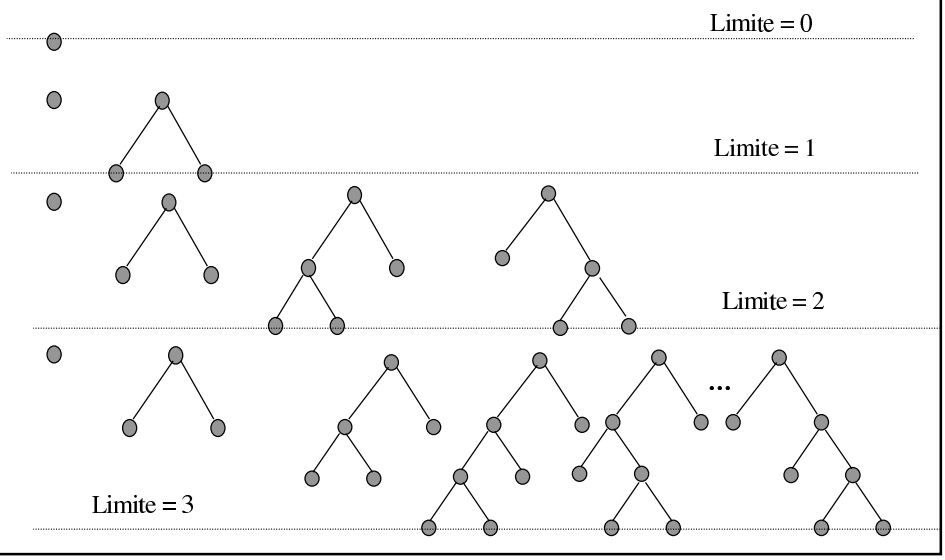
Busca em profundidade limitada



Profundidade máxima para 20 cidades: 19

19

Busca em profundidade iterativa

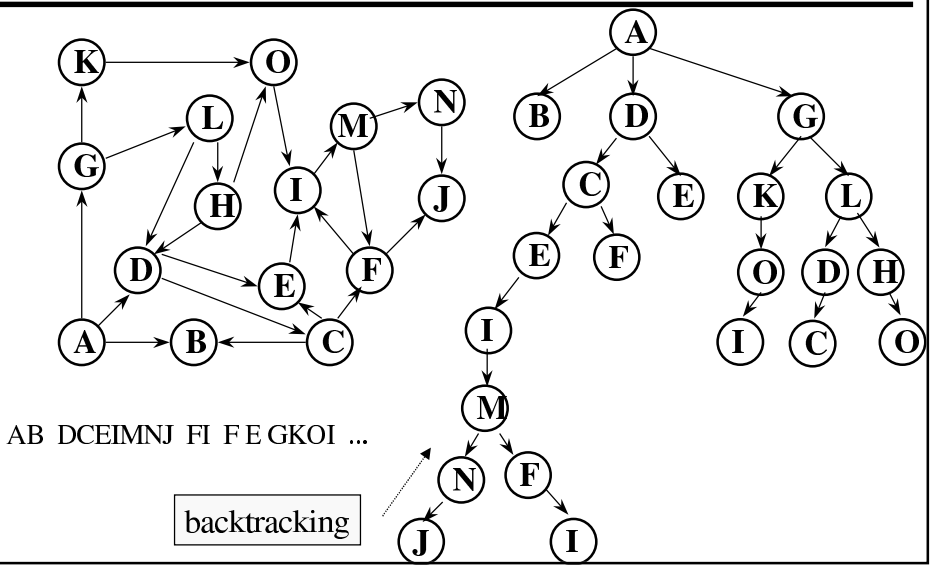


20

Propriedades da busca em profundidade iterativa

- **Completa?** Sim
- **Tempo?** $O(b^d)$
- **Espaço?** $O(bd)$
- **Solução ótima?** Sim, se custo = 1 por passo.

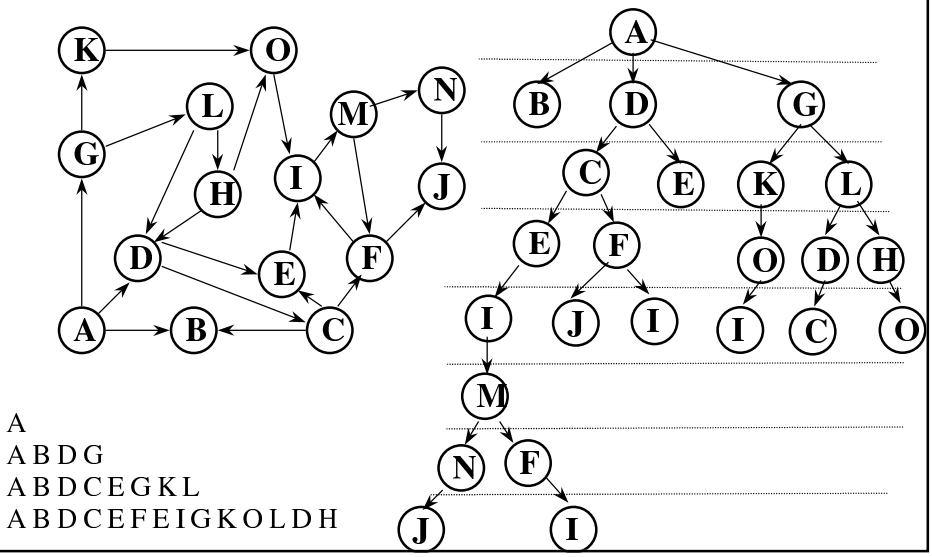
Busca em profundidade



AB DCEIMNJ FI F E GKOI ...

backtracking

Busca em profundidade iterativa



A
A B D G
A B D C E G K L
A B D C E F E I G K O L D H

Direções de busca

- **Encadeamento para frente** *(Forward chaining)*:
  - começa do estado atual até encontrar o estado gol (gera predecessores)
- **Encadeamento para trás** *(Backward chaining)*:
  - começa do estado gol até encontrar o estado atual do mundo (só possível se o gol é conhecido). Gera sucessores (operadores reversíveis ?)
- **Biderecional ...**

depende da estrutura do espaço de problema

Exemplos de direções:

- 8-puzzle: gol e estado inicial são conhecidos
- 8-rainhas, xadrez: o gol não é explícito

Busca bidirecional

- útil se o estado inicial e o gol são conhecidos
- similar a busca de custo uniforme com a diferença de começar a busca com o estado inicial e gol, expandindo nós em ambas as direções com a esperança de se encontrarem no meio
- Alternativas:
  - Expande alternativamente a partir dos dois lados
  - Expande o lado com menor número de nós na lista

Propriedades da busca biderecional

- **Completa?** Sim
- **Tempo?** $O(b^{d/2})$
- **Espaço?** $O(b^{d/2})$ (guarda todos os nós na memória)
- **Solução ótima?** Sim

Agente que resolve problemas

- Agentes que "raciocinam" versus agentes que consultam uma tabela
- Decidem o que fazer encontrando uma sequência de ações que o levam ao estado desejado (goal)
- O agente deve ser capaz de
  - formular o goal e o problema
  - encontrar a solução (busca)
  - executar

FORMULAÇÃO
BUSCA
EXECUÇÃO

Agente "deliberativo"

FORMULAÇÃO
BUSCA
EXECUÇÃO

função Agente-Deliberativo-Simples (*p*) **devolve** *ação*
  ***entradas*:** *p,* uma percepção
  estático: *s*, um sequência de estados, inicialmente vazia
    estado, alguma descrição do estado do mundo
    g, um goal, inicialmente nulo
    problema, uma formulação de problema
  estado ← ATUALIZA-ESTADO (*estado, p*)
  se *s* vazia ***então***
    g ← FORMULA-GOAL (*estado)*
    problema ← FORMULA-PROBLEMA (*estado, g*)
    s ← BUSCA (*problema)*
  ação ← RECOMENDA (*s, estado*)
  s ← RESTO (*s, estado)*
  devolve *ação*

Encontrar solução através de uma busca no espaço de estados

- Processo de examinar diferentes possíveis sequências de ações que levam a estados desejados (goals) e escolher a melhor
- goal: conjunto de estados do mundo
- ações: causam transições de um estado para o outro / descrevem mudanças no mundo