Introdução a Inteligência Artificial

Busca em grafos

Alneu de Andrade Lopes

Busca

- Imagine que você está em uma nova cidade e deseja encontrar um bar para beber um chopp
  - Se você tem um mapa, você pode descobrir como sair de onde você está para chegar ao bar
  - Se você não tem um mapa, você poderia caminhar sem rumo até encontrar um bar
  - Ou você poderia procurar sistematicamente por um bar

Busca

- Você poderia, por exemplo, utilizar o seguinte algoritmo de “encontrar bar”
  1. Procure um bar
  2. Caso não encontrou um bar, vá para um local não visitado por você e repita o passo 1
  3. Se você encontrou um bar, vá e beba um chop
  4. Após o décimo chopp, saia e caia na sarjeta

Busca

- Um sistema de IA pode resolver problemas da mesma forma:
  - Ele sabe onde ele está (conjunto de informações inicial)
  - Ele sabe onde deseja ir (estado objetivo)
  - Ele sabe como ir para um próximo estado
- Resolver problema em IA envolve a busca pelo estado objetivo
- Simples sistemas de IA reduzem raciocínio a busca

Busca

- Problemas de busca são frequentemente descritos utilizando diagramas de árvores
  - Nó inicial = onde a busca começa
  - Nó objetivo = onde ela termina (Fig 1)
- Objetivo: Encontrar um caminho que ligue o nó inicial a um nó objetivo

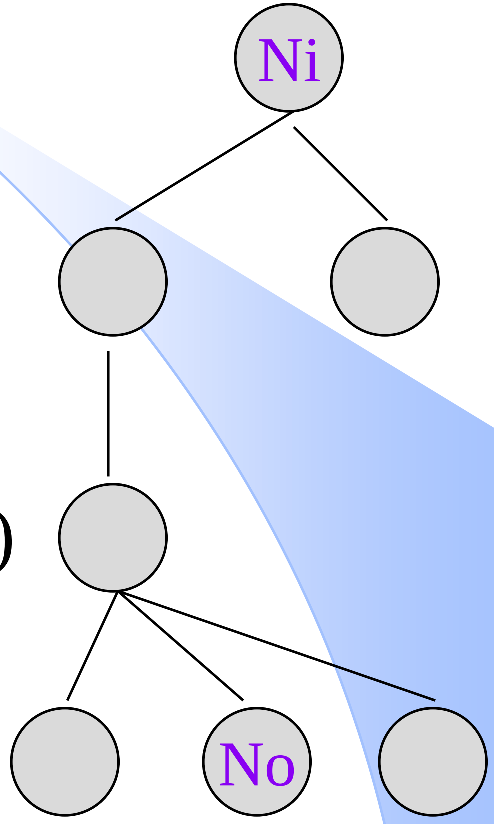


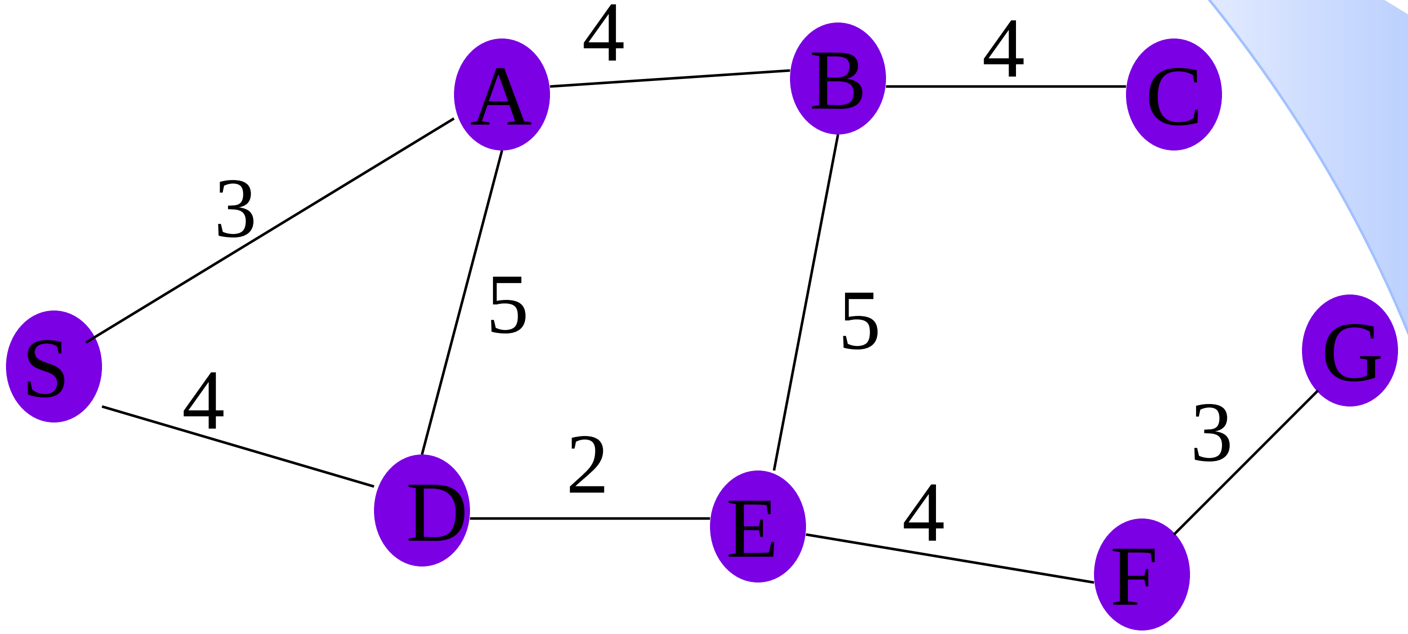
Fig 1

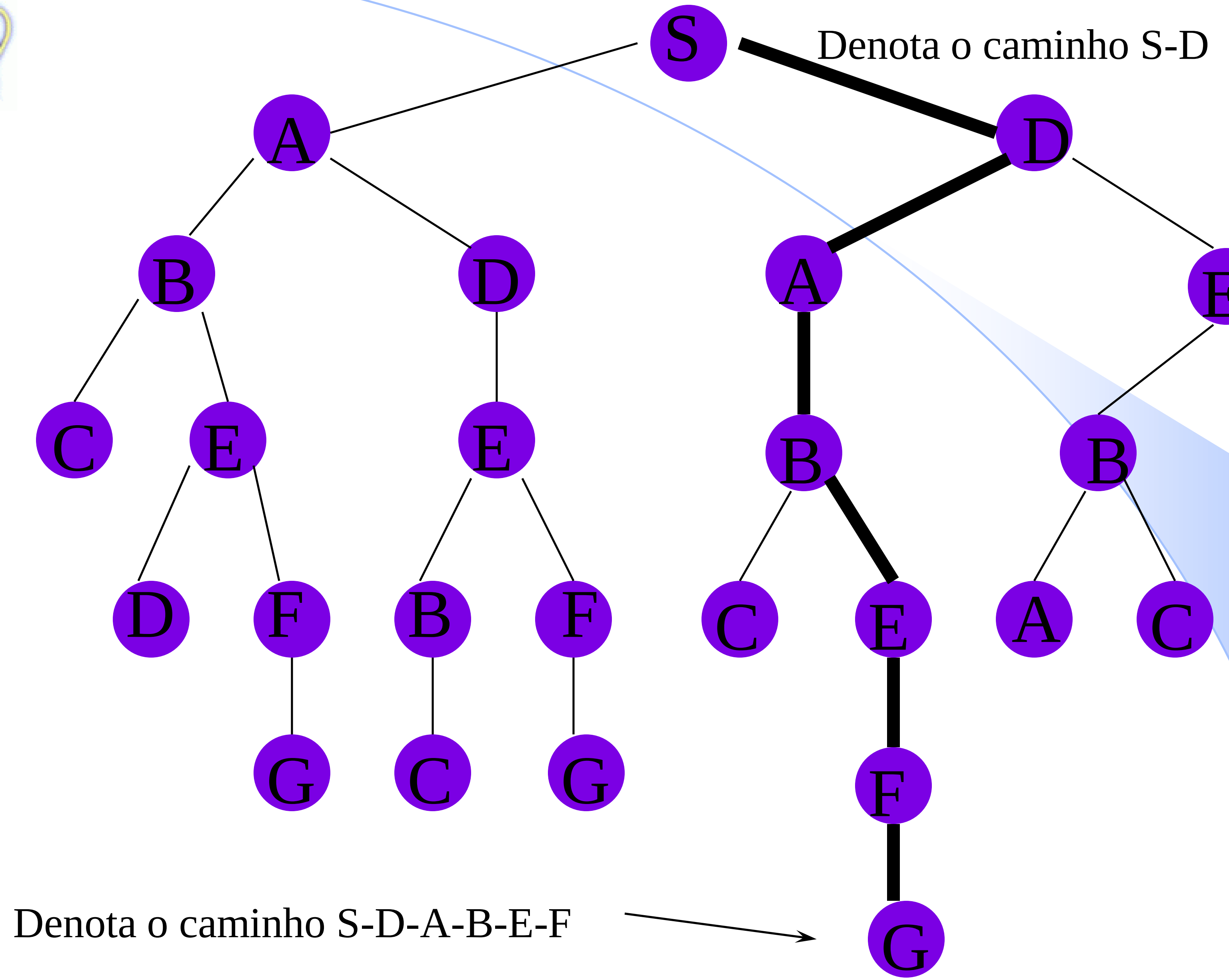
Busca

- Entrada:
  - Descrição dos nós inicial e objetivo
  - Procedimento que produz os sucessores de um nó arbitrário
- Saída:
  - Sequência legal de nós iniciando com o nó inicial e terminando com o nó objetivo
  - Exemplos: encontrar o menor caminho entre S e G no mapa a seguir.

Exemplo

- Dado o mapa (grafo) abaixo, encontrar a menor distância de S a G




S
Denota o caminho S-D
A
D
B
D
A
E
C
E
E
B
B
F
D
F
B
F
C
E
A
C
G
G
C
G
F
Denota o caminho S-D-A-B-E-F
G

Exemplo: palavras cruzadas

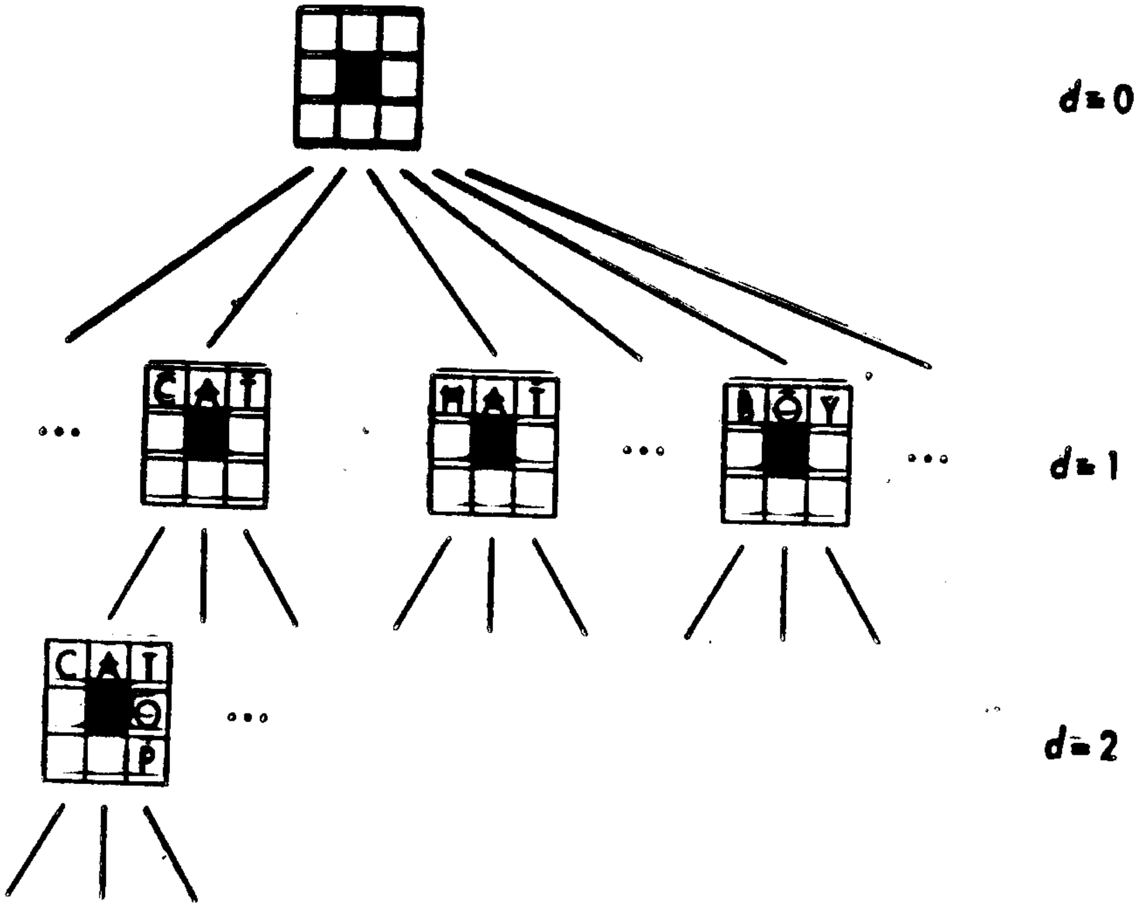
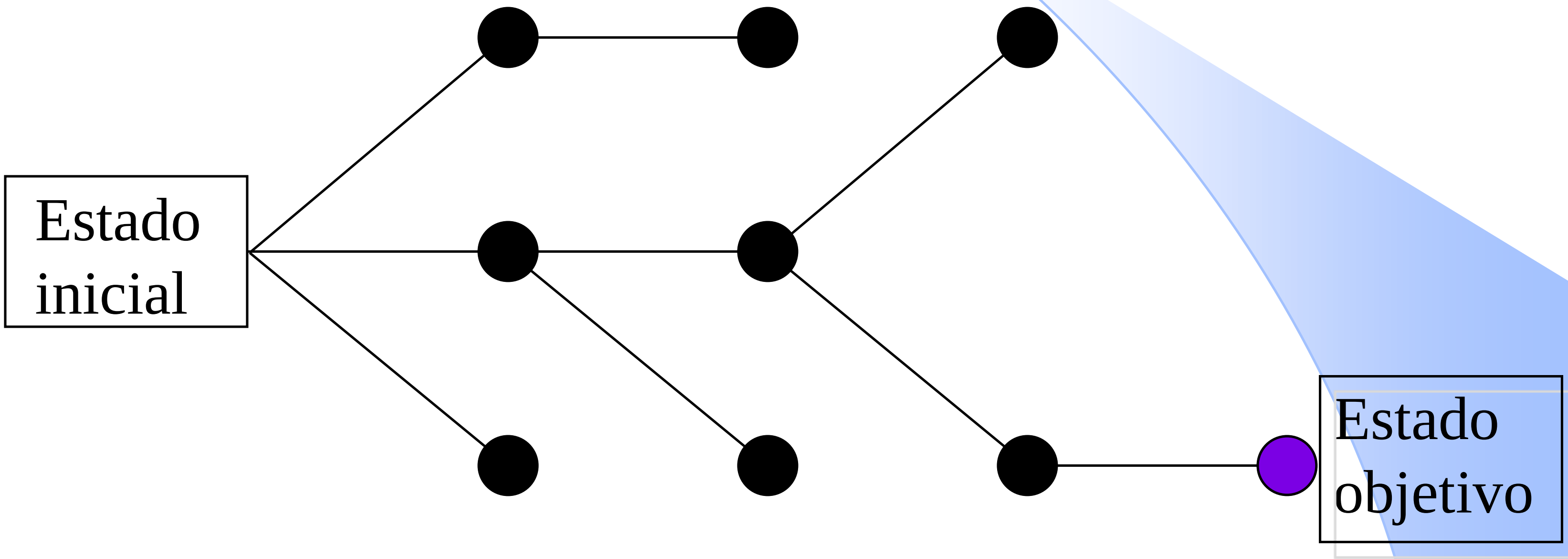


Fig 2

Uma árvore de busca

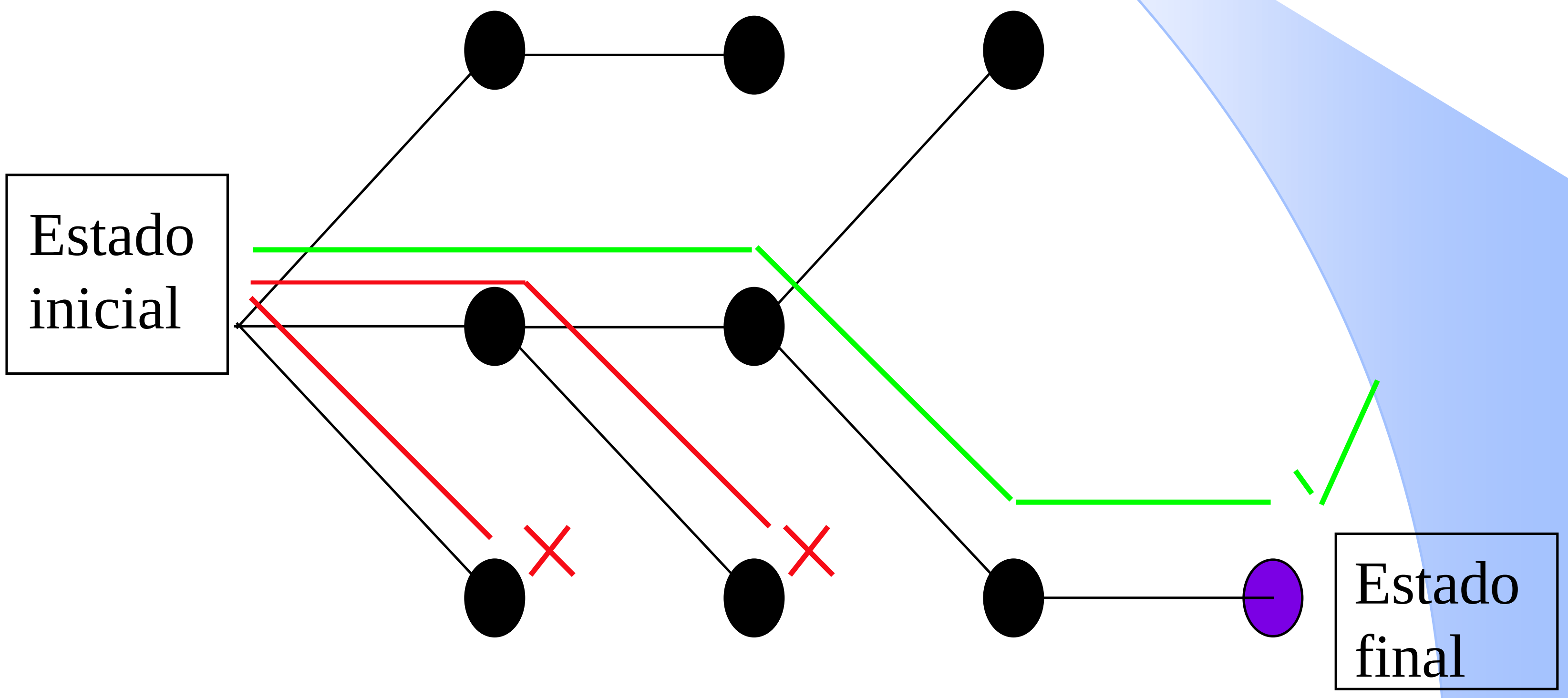
- Uma busca pode ser definida graficamente:



- O objetivo é atravessar a árvore partindo do estado inicial até o estado objetivo

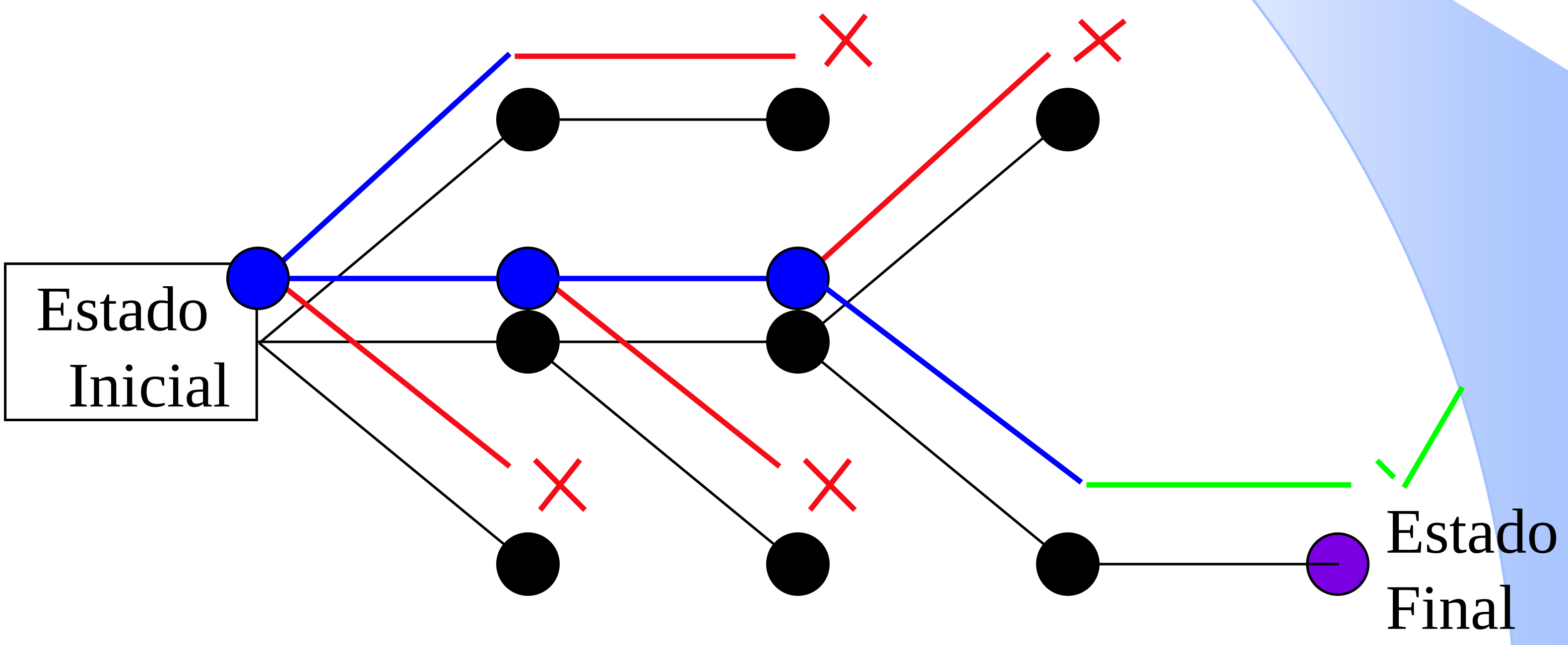
Tipos de busca

- Busca em profundidade envolve buscar o final de um caminho antes de tentar um caminho alternativo:



Tipos de busca (cont)

- Busca em largura envolve escolher um caminho e segui-lo até o próximo ponto de decisão ou até o objetivo ser atingido:



Problemas da busca

- Com o aumento da árvore de decisão e do número de possíveis caminhos, o tempo de busca aumenta
- Existem várias formas de reduzir o tempo de busca, algumas das quais serão discutidas mais adiante

Possíveis situações

- Mais de um nó objetivo
- Mais de um nó inicial
- Nestas situações
  - Encontrar qualquer caminho de um nó inicial para um nó objetivo
  - Encontrar melhor caminho

Definições importantes

- Profundidade: número de ligações entre um dado nó e o nó inicial
- Largura: número de sucessores (filhos) de um nó

Algoritmo básico de busca

1 Definir um conjunto L de nós iniciais;

2 Se L é vazio

Então Busca não foi bem sucedida

Senão Escolher um nó n de L;

3 Se *n* é um nó objetivo

Então Retornar caminho do nó inicial até n;

Parar

Senão Remover n de L;

Adicionar a L todos os filhos de n, rotulando cada um   com o seu caminho até o nó inicial;

Voltar ao passo 2

Algoritmos de Busca

- Existem vários algoritmos de busca diferentes, o que os distingue é a maneira como o nó *n* é escolhido no passo 2
- Métodos de busca
  - Busca cega: a escolha depende da posição do nó na árvore de busca
  - Busca heurística: A escolha utiliza informações específicas do domínio para ajudar na decisão

Busca cega

- Técnicas de busca cega
  - Grande número de métodos
    - Busca em Profundidade
    - Busca em Largura

Técnicas de busca cega

- Busca em Profundidade (BP)
  - A árvore é examinada de cima para baixo
  - Aconselhável nos casos onde os caminhos improdutivos não são muito longos
- Busca em Largura (BL)
  - A árvore é examinada da esquerda para a direita
  - Aconselhável quando o número de ramos não é muito grande

Algoritmo BP

1 Definir um conjunto L de nós iniciais

2 Seja n o primeiro nó de L;

Se L é vazio

Então Busca não foi bem sucedida

3 Se n é um nó objetivo

Então Retornar caminho do nó inicial até n;

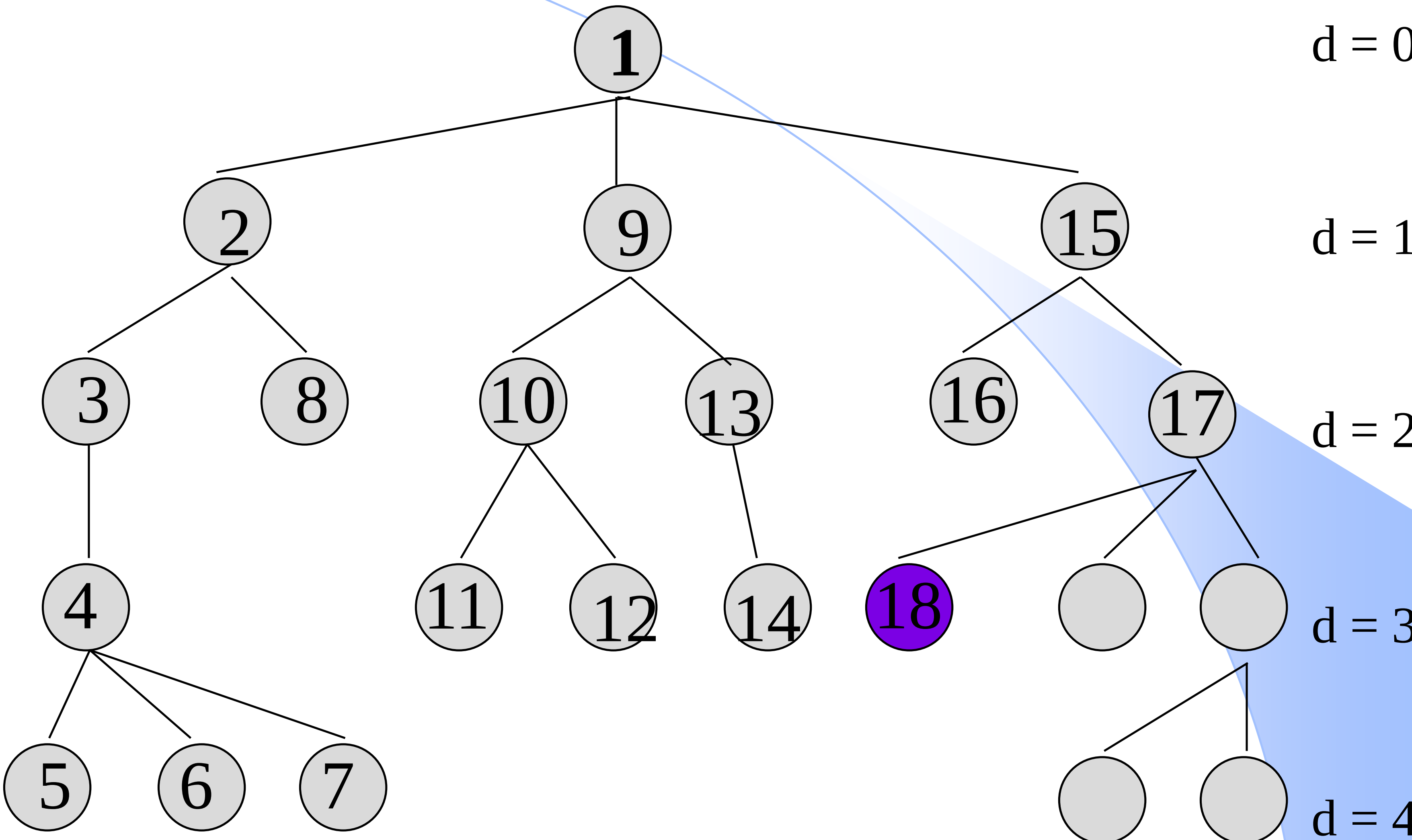
Parar

Senão Remover n de L;

Adicionar ao início de L todos os filhos de n, rotulando cada um com o seu caminho até o nó inicial;

Voltar ao passo 2;

Busca em profundidade



BP Em Prolog

```
% implementando o Alg.
caminho1(O,D,Cam) :-
        caminho2([[O]],D,Cam).
caminho2([[D|C]|_],D,[D|C]).
caminho2([[A|R]|Outros],D,C):-
        todos_filhos(A,R,L),
        append(L,Outros,L1),
        caminho2(L1,D,C).
```

```
todos_filhos(A,R,L):-
        findall([X,A|R],
        (aresta(A,X),
        not(member(X,R))),L).

append([],L,L).
append([H|T],L,[H|T1]):-
        append(T,L,T1).
```

Implementação 2

% busca usando o backtracking do Prolog

```
caminho(O,D,Cam) :-
        caminho(O,D,[],Cam).
caminho(D,D,C,[D|C]).
caminho(A,D,Ac,C):-
        aresta(A,X),
        not(member(X,Ac)),
        caminho(X,D,[A|Ac],C).
```

Algoritmo BL

1 Definir um conjunto L de nós iniciais

2 Seja n o primeiro nó de L;

Se L é vazio

Então Busca não foi bem sucedida

3 Se n é um nó objetivo

Então Retornar caminho do nó inicial até n;

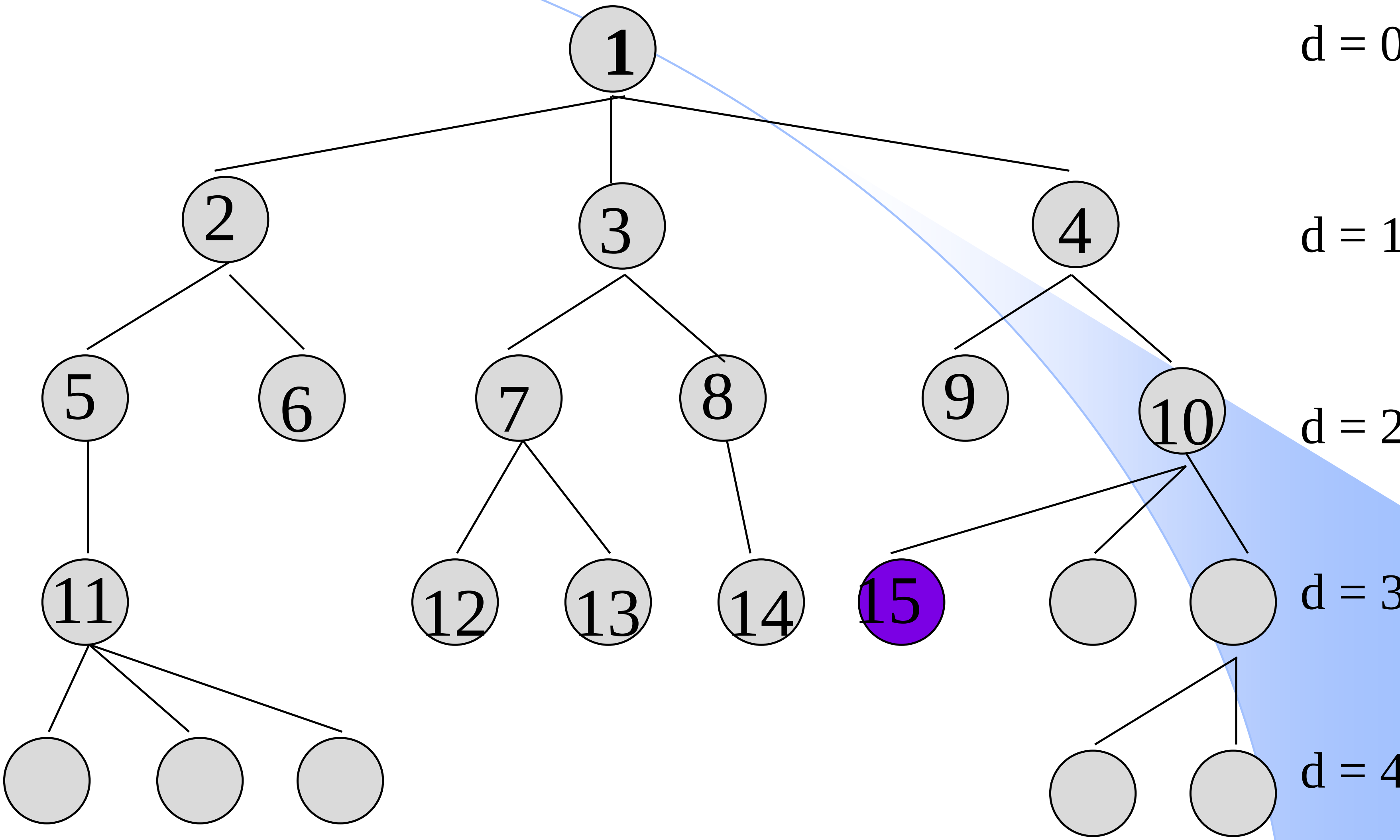
Parar

Senão Remover n de L;

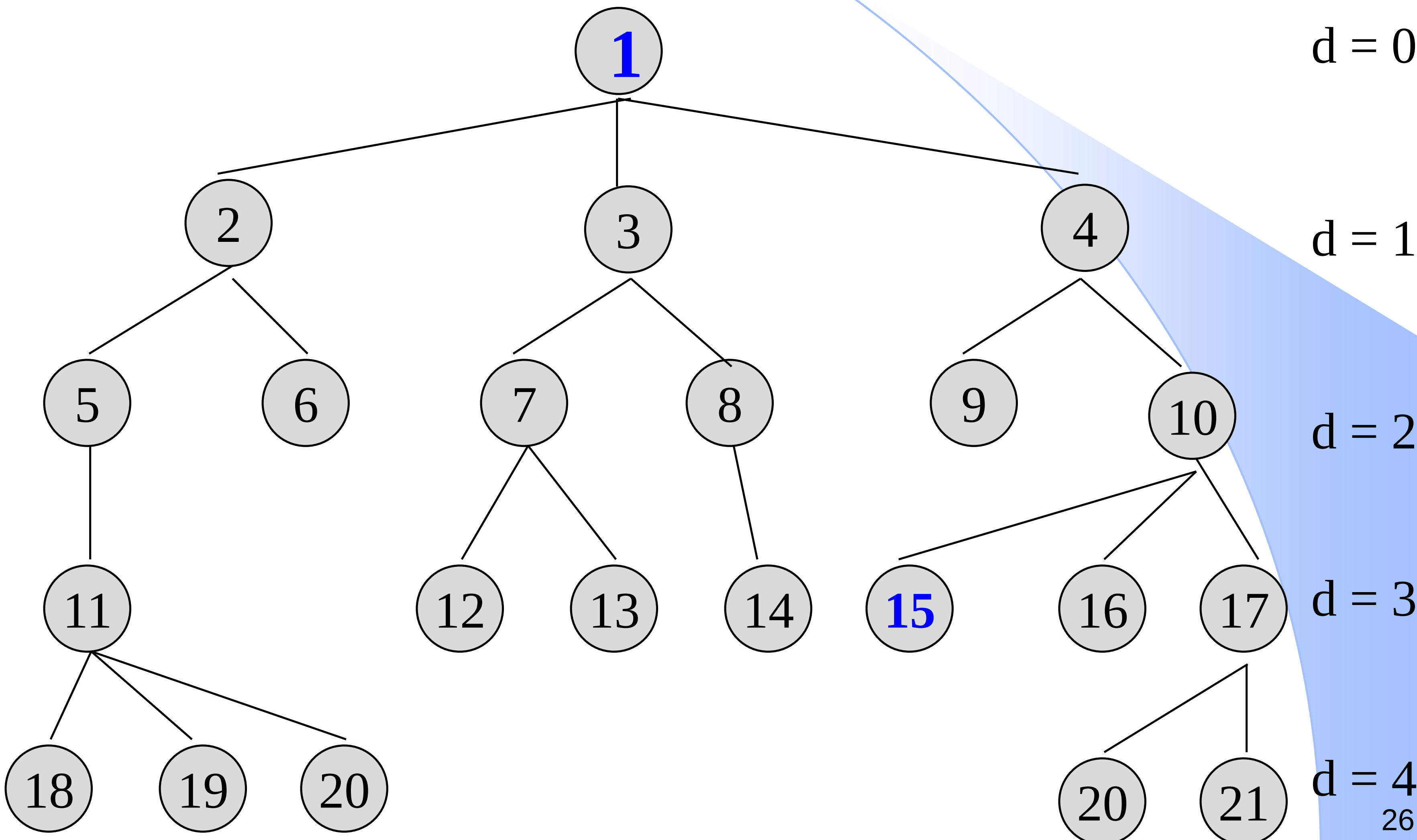
Adicionar ao final de L todos os filhos de n, rotulando cada um com o seu caminho até o nó inicial;

Voltar ao passo 2;

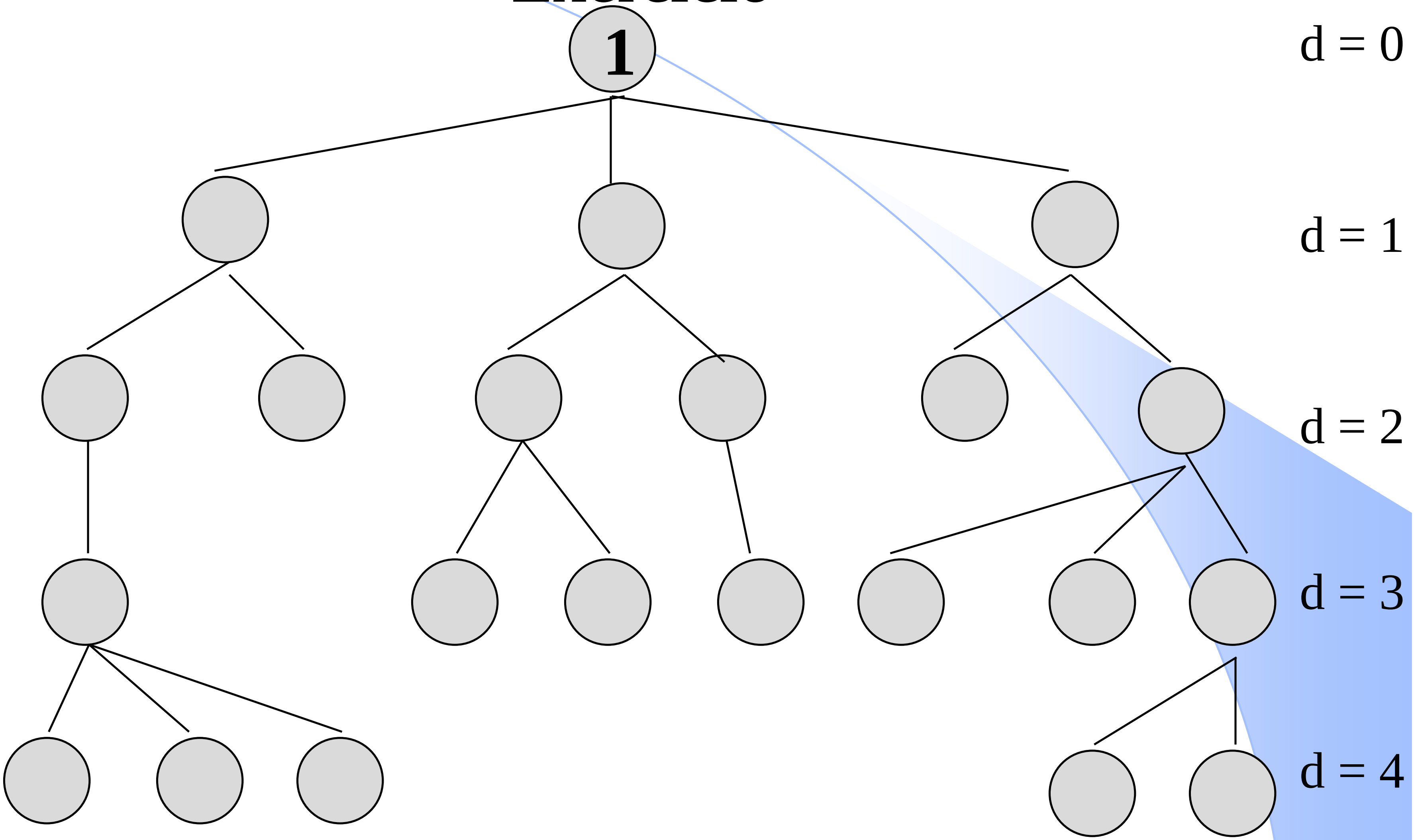
Busca em largura



Exemplo 1: Dada a árvore abaixo, utilizando BP, indique: a) Memória máxima e b) Número mínimo de passos necessários para atingir um dos nós objetivos



Exercício



Resposta ao exemplo 1

a) 1 L = {1}

2 L = $\{2_1, 3_1, 4_1\}$

3 L = $\{5_{21}, 6_{21}, 3_1, 4_1\}$

4 L = $\{11_{521}, 6_{21}, 3_1, 4_1\}$

5 L = $\{18_{11521}, 19_{11521}, 20_{11521}\ 6_{21}, 3_1, 4_1\}$

6 L = $\{19_{11521}, 20_{11521}\ 6_{21}, 3_1, 4_1\}$

7 L = $\{20_{11521}, 6_{21}, 3_1, 4_1\}$

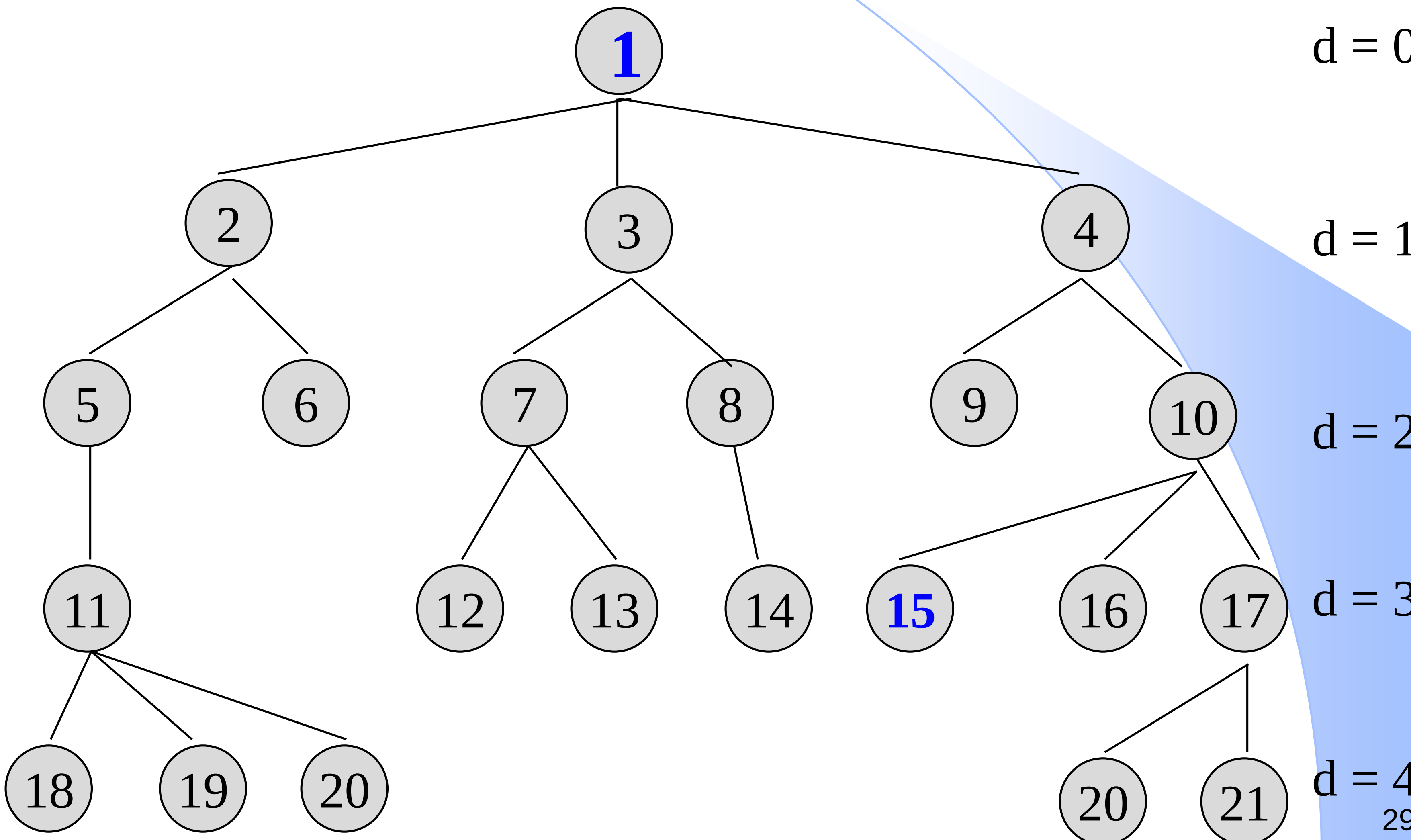
8 L = $\{6_{21}, 3_1, 4_1\}$

9 L = $\{3_1, 4_1\}$

....

18 L = $\{15_{1041}, 16_{1041}, 17_{1041}\}$

Exemplo 2: Dada a árvore abaixo, utilizando BL, indique: a) Memória máxima e b) Número mínimo de passos necessários para atingir um dos nós objetivos



Observações

- BP e BL não precisam ser realizadas em uma ordem específica
- Memória utilizada pelas duas técnicas
  - BP: precisa armazenar todos os filhos não visitados de cada nó entre nó atual e nó inicial
  - BL: antes de examinar nó a uma profundidade d, é necessário examinar e armazenar todos os nós a uma profundidade d - 1
  - BP utiliza menos memória

Observações

- Quanto ao tempo
  - BP é geralmente mais rápida
  - Métodos de busca cega não examinam a árvore de forma ótima, o que poderia minimizar o tempo gasto para resolver o problema

Técnicas exaustivas de busca

- Completude: a solução sempre será encontrada?
- "Optimalidade": o caminho mais curto será encontrado antes dos caminhos mais longos?
- Eficiência: quais são os requisitos de memória e tempo de execução?

Busca Baseada em Agenda

1. pegue o próximo nó na agenda;
2. Se é um nó meta, pare;
3. Senão
   1. Obtenha seus filhos;
   2. Coloque-os na agenda;
   3. Vá para o passo 1.

Agenda: lista de nós a serem avaliados (backtracking)

Pesquisa baseada em Agenda (Agenda-based search)

```
% search(Agenda,Goal) <- Goal é um nó meta, e um
%                        nó filho de um dos nós em
%                        Agenda
search(Agenda,Goal):-
   next(Agenda,Goal,Rest),
   goal(Goal).
search(Agenda,Goal):-
   next(Agenda,Atual,Rest),
   children(Atual,Filhos),
   add(Filhos,Rest,NovaAgenda),
   search(NovaAgenda,Goal).
```

Busca em profundidade x busca em largura (Depth-first vs. breadth-first search)

- Busca em profundidade (*Depth-first)*
  - agenda = pilha (last-in first-out)
  - incompleta: pode ficar preso em um ramo infinito
  - Nenhuma propriedade de menor caminho
  - Requisito de memória: $O(B \times n)$
    n= n.médio de filhos
    B=profundidade
- Busca em largura (*Breadth-first)*
  - agenda = fila (first-in first-out)
  - completa: guarante encontrar todas as soluções
  - Primeira solução encontrada no menor caminho
  - Requisito de memória: $O(B^n)$

Busca informada não exaustiva

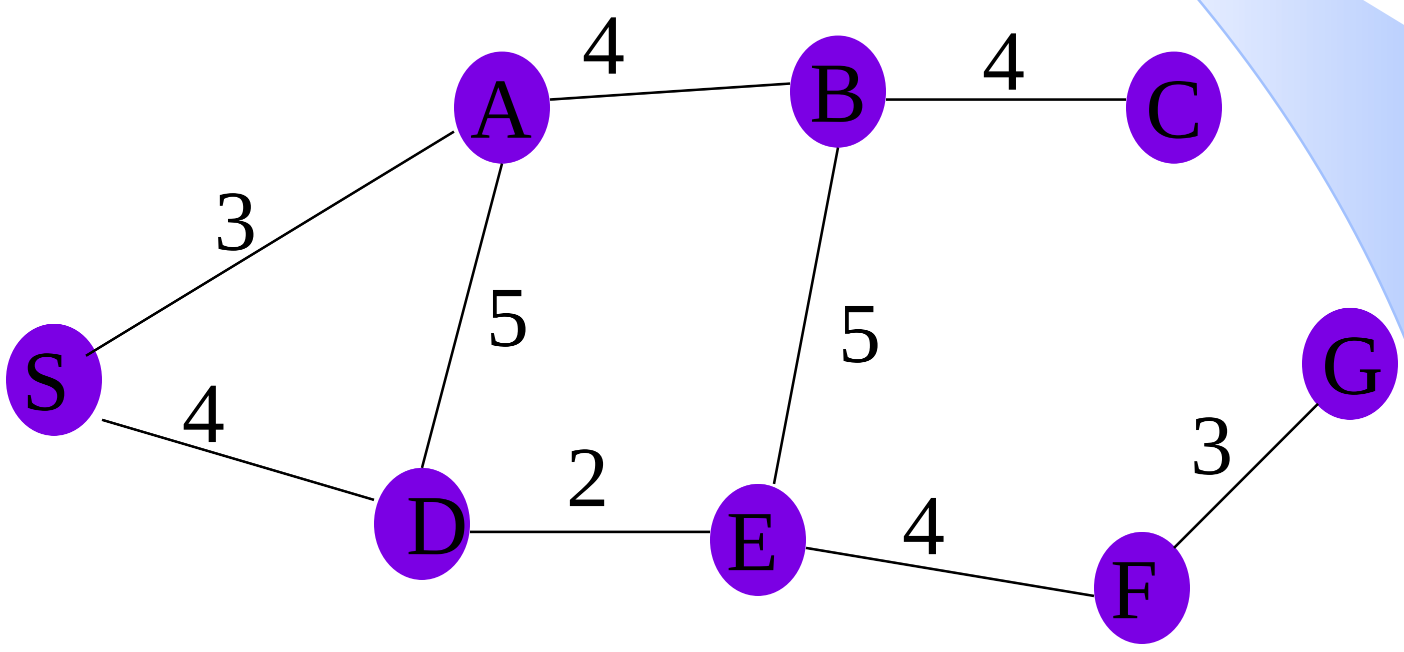
- A busca exaustiva (anteriores) no pior caso examinam todos os nós;
- Isto porque todos os filhos de um nó são adicionados a agenda.
- Adicionando só os nós “mais promissores” tem-se um busca não exaustiva (não necessariamente completa).

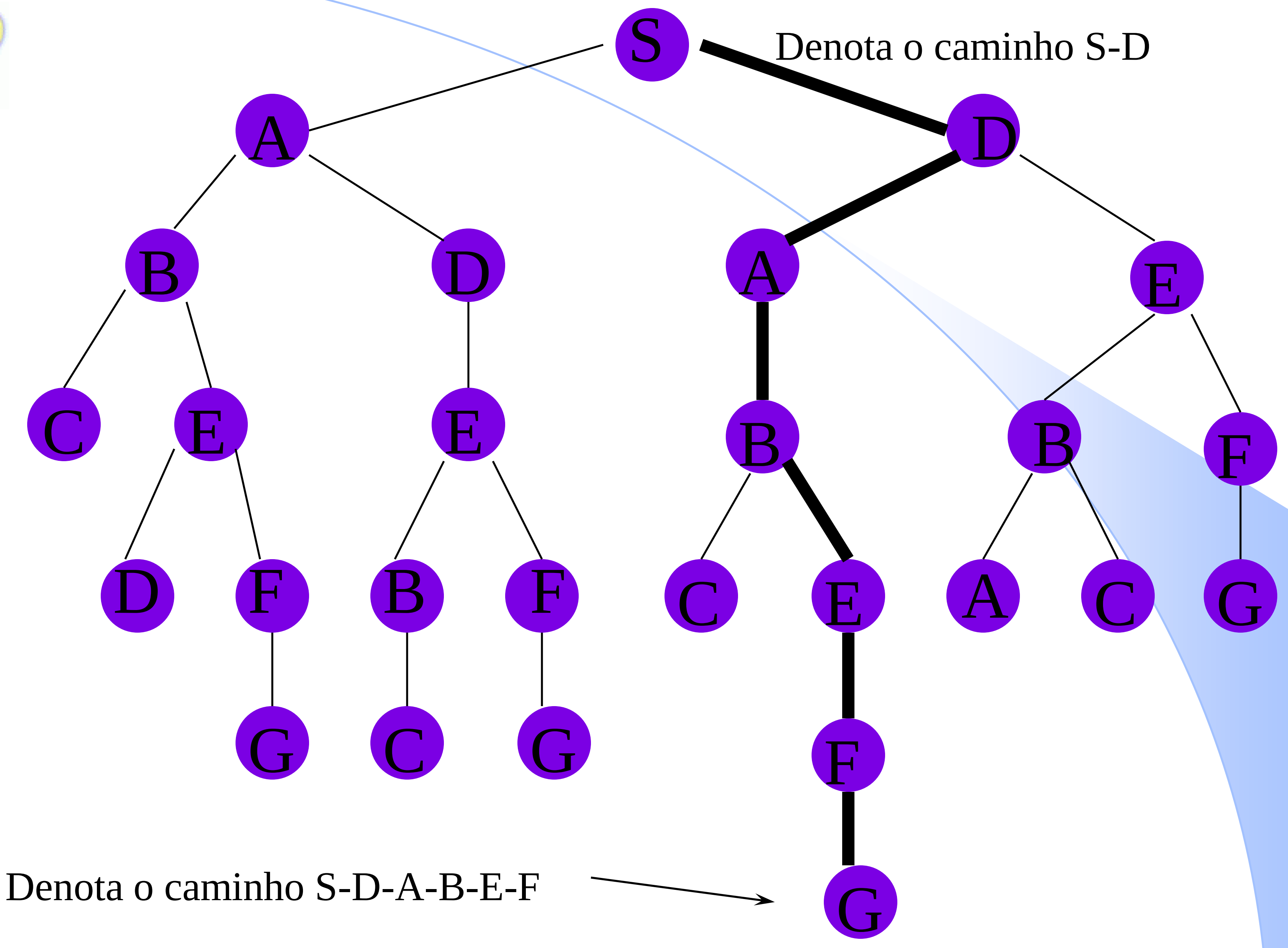
Resumo Principais Tópicos

- Busca: paradigma de resolução de problemas
- Métodos de busca
  - Busca cega
    - Busca em profundidade
    - Busca em largura
  - Busca heurística
    - Algoritmo A (discutido adiante)
    - Hill Climbing (Anexo)
    - Busca em feixe (Anexo)

Exemplo

- Dado o grafo abaixo, encontrar a menor distância de S a G




S
Denota o caminho S-D
A
D
B
D
A
E
C
E
E
B
B
F
D
F
B
F
C
E
A
C
G
G
C
G
F
Denota o caminho S-D-A-B-E-F
G

Problemas da busca

- Com o aumento da árvore de decisão e do número de possíveis caminhos, o tempo de busca aumenta
- Existem várias formas de reduzir o tempo de busca, alguns dos quais serão discutidos mais adiante

Busca Heurística

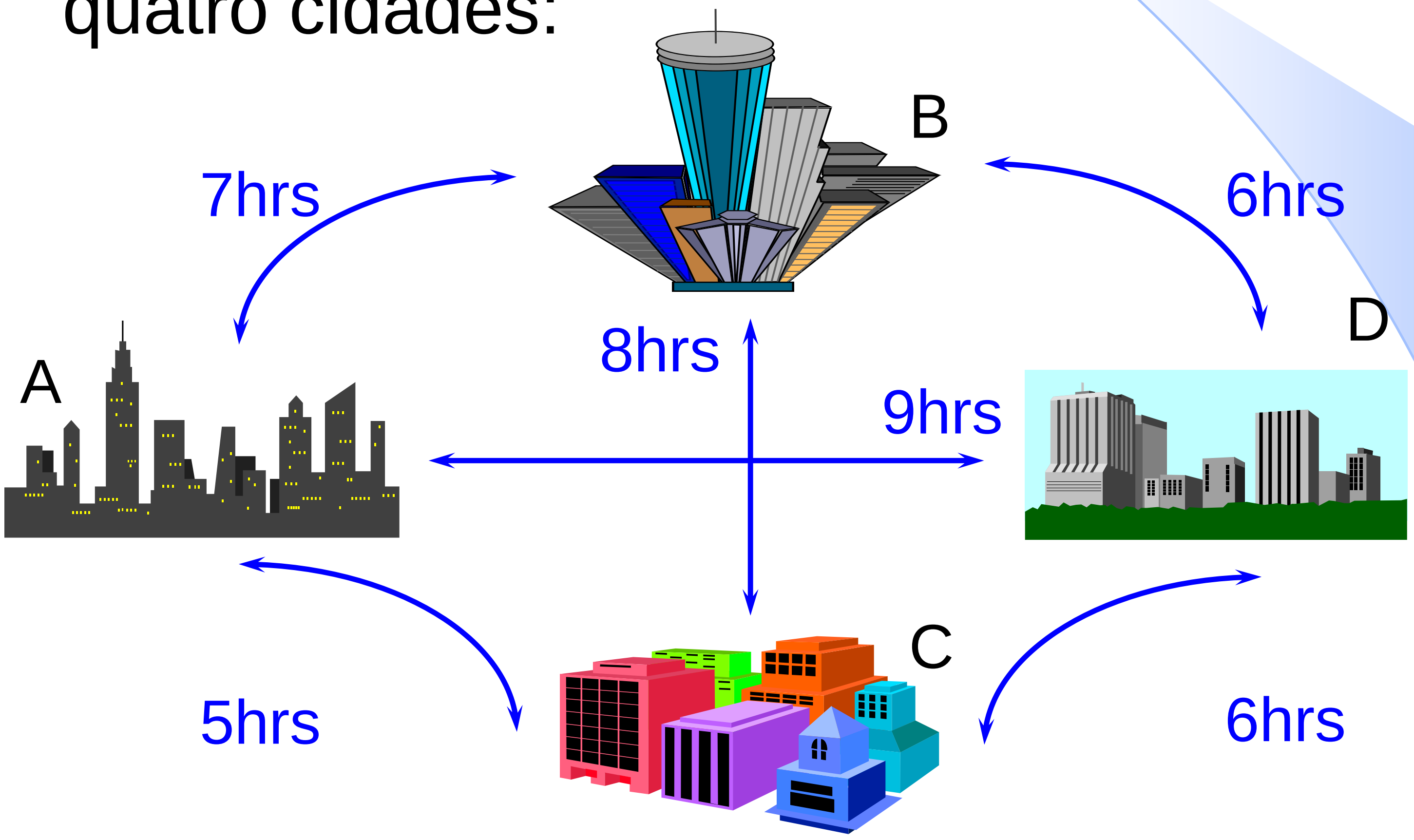
- Digamos que você está numa Cidade, e quer pegar um trem para casa, mas não sabe qual deve pegar.
- Se você morasse na zona Norte, naturalmente ignoraria todos os trens que fossem para o sul.
- Se você morasse na zona Sul, naturalmente ignoraria todos os trens que fossem para o Norte.

Exemplo: problema do caixeiro viajante (TSP)

- Um caixeiro viajante deve visitar N cidades em sua área de vendas
- O caixeiro começa de uma base, visita cada cidade uma única vez e retorna à sua cidade no final
- A cada viagem esta associado um custo
  - O caixeiro deve percorrer a rota mais curta

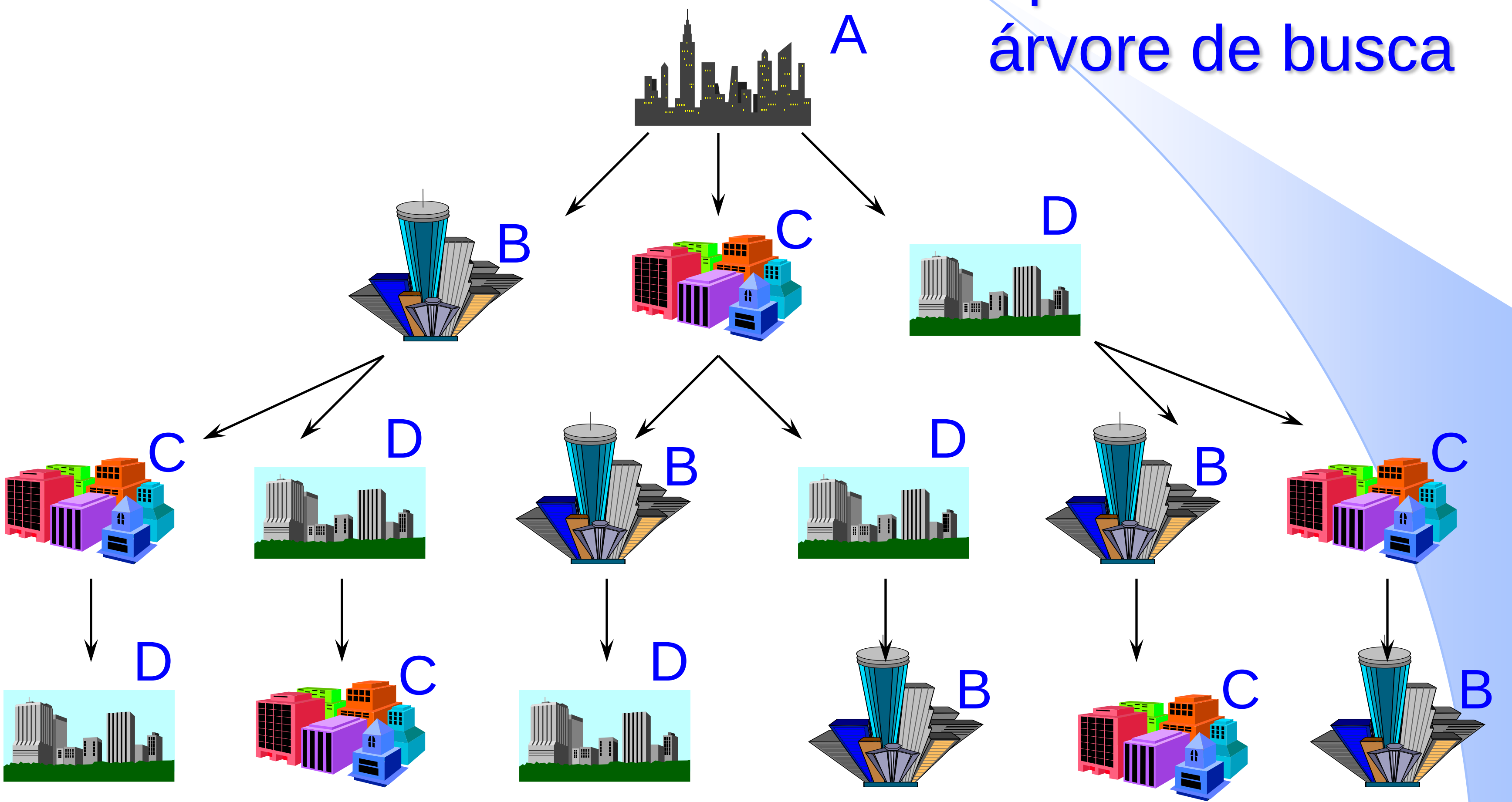
O problema TSP

Considere as rotas definidas entre estas quatro cidades:



O problema TSP

O problema do TSP representado como árvore de busca



Explosão Combinatória

- Com quatro cidades, temos 6 caminhos possíveis.
- Com dez cidades, temos 362.880 caminhos possíveis.
- Quanto mais cidades adicionarmos ao TSP, mais caminhos possíveis há.
- O que nos leva a uma *explosão combinatória*.
- Como prevenir ou pelo menos limitar isto?

Problemas Clássicos

- Encontrar um caminho para um objetivo
- Missionários e canibais
- N-rainhas
- Jogos
- Xadrez
- Gamão
- Torres de Hanói
- Simplesmente encontrar um objetivo
- Problema do tabuleiro de xadrez danificado

Observações

- Perguntas a serem feitas antes de utilizar métodos de busca:
  - Busca é a melhor maneira para resolver o problema?
  - Quais métodos de busca resolvem o problema?
  - Qual deles é o mais eficiente para este problema?

Algoritimo A

- Um ***algoritmo A*** é um algoritmo de busca *best-first* que objetiva minimizar o custo total do caminho do nó início ao nó meta.
  - $f(n) = g(n) + h(n)$

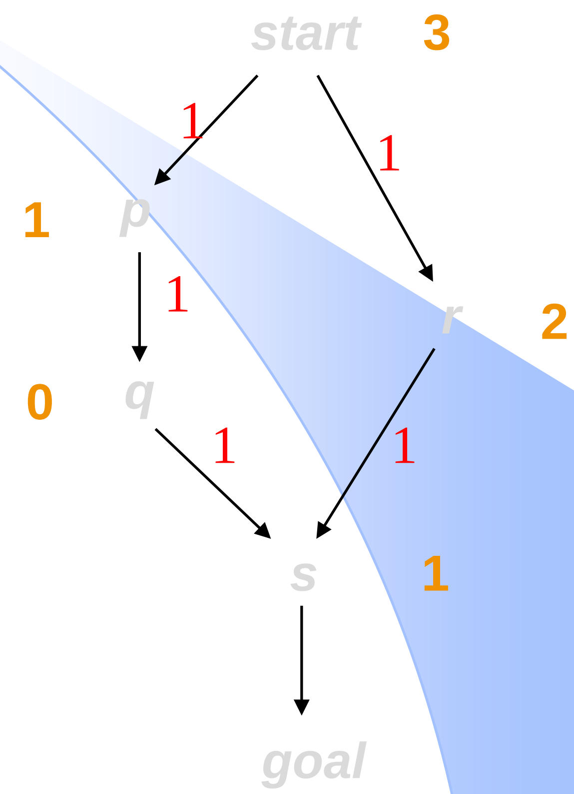
Estimativa do custo total ao longo do caminho através de n

Custo real para alcançar n

Estimativa para alcançar a meta a partir de n

exemplo

- Menor caminho:
  start-r-s-goal
- Não é encontrado imediatamente.
  - h(p) < h(r)
  - Depois de q ser investigado r é colocado na agenda, com
    f(r)=1+2=3
    f(s)=3+1=4



- O algoritmo A é completo: desde que g(n) previne que a busca fique presa em um caminho infinito (comporta-se como na busca *breadth-first*);
- Aliás, *bf search* é um caso especial do algoritmo A, com h(n)=0 para todos os nós.

- É ótimo: sempre leva a solução de menor caminho (se arestas têm os mesmos custos);
- Um algoritmo A com um heurística otimista é chamado A*.
  - Obs 1 h(n) é uma heurística admissível se nunca superestima o custo para alcançar o objetivo (otmista)
  - Obs 2. Melhor heurística não quer dizer mais otimista. Ao contrário uma boa heurística é tão pessimista quanto possível sem se tornar não admissível.

Exercícios

Caminho do cavalo no tabuleliro de xadrez

- Cavalo numa posição (x,y) pode ir para
- (x+deltax, y+deltay)
- (deltax,deltay) ϵ {(2,1),(2,-1),(-2,1),(-2,-1),
  (1,2),(-1,2),(1,-2),(-1,-2)}

```
pode_ir([X,Y],[X1,Y1]):-
    delta(Dx,Dy),
    X1 is  X+Dx, 0<X, X<9,
    Y1 is Y + Dy, 0<Y,Y<9.
```

```
delta(2,1).
delta(2,-1).
...
```

Caminho cavalo

```
caminho_cav(I,F,C):-
    caminho_cav(F,[I],C).

caminho_cav1(X,[X|L],[X|L]).

caminho_cav1(I,[E|C],Cam):-
    pode_ir(E,E1),
    not(member(E1,C)),
    caminho_cav1(I,[E1,E|C],Cam).
```

8 rainhas

Ex

/* transporta de uma margem de um rio para outra margem os objetos milho, galinha, lobo, sendo que não podem ficar juntos m,g e g,l

m1=[m,g,l] b=[] m2=[]

m1=[m,l] b=[g] m2=[] escolhe um da margem1 para transportar para outra

m1=[m,l] b=[] m2=[g]

m1=[m] b=[l] m2=[g]

m1=[m] b=[g] m2=[l] troca o que está na margem2 com o que está no barco

m1=[g] b=[m] m2=[l] troca o que está na margem1 com o que está no barco

...

m1=[] b=[] m2=[g,l,m]

*/

```
pode([m,l]).
pode([l,m]).
pode([g]).
pode([m]).
Pode([l]).
```

```
% plano([[g,l,m],[],[]],[[],[],[g,l,m]],P)

plano(O,D,Pl) :-
        plano2([[O]],D,P),
        inverte(P,Pl),
        show(Pl).

plano2([[D|C]|_],D,[D|C]).
plano2([[A|R]|Outros],D,C):-
        todos_filhos(A,R,L),
        append(Outros,L,L1),
        plano2(L1,D,C).
```

```
todos_filhos(A,R,L):-
        findall([X,A|R],
        (aresta(A,X),not(member(X,R))),
        L).
append([],L,L).
append([H|T],L,[H|T1]):-
        append(T,L,T1).
inverte(A,B):-
        inverte(A,[],B).
inverte([],B,B).
inverte([H|T],Ac,B):-
        inverte(T,[H|Ac],B).
show([]).
show([A|R]):-
        writeln(A),
        show(R).
```

```
% aresta([[g,l,m],[],[]],[[g,m],[l],[]])

aresta([L,[],[]],[L2,[X],[]]):-
        member(X,L),
        retira(X,L,L2),
        pode(L2).
aresta([[],[],L],[[],[X],L2]):-
        member(X,L),
        retira(X,L,L2),
        pode(L2).
aresta([L,[],[Y]],[L2,[X],[Y]]):-
        member(X,L),
        retira(X,L,L2).
aresta([L,[],[Y]],[L,[Y],[]]).
aresta([[Y],[],L],[[Y],[X],L2]):-
        member(X,L),
        retira(X,L,L2).

aresta([[Y],[],L],[[],[Y],L]).
aresta([[A],[B],[C]],[[A,B],[],[C]]):-
        pode([A,B]).

aresta([[A],[B],[C]],[[B],[A],[C]]).
aresta([[A],[B],[C]],[[A],[],[B,C]]):-
        pode([B,C]).

aresta([[A],[B],[C]],[[A],[C],[B]]).
aresta([[],[Y],L],[[Y],[],L]).
aresta([L,[Y],[]],[L,[],[Y]]).
aresta([[],[A],[B,C]],[[],[],[A,B,C]]).
aresta([[A,B],[C],[]],[[A,B,C],[],[]]).
retira(X,[],[]).
retira(X,[X|R],R).
retira(X,[Y|R],[Y|R1]) :-
        X \= Y,
        retira(X,R,R1).
```

```
/* deseja-se encontrar uma instanciação correta para as cores de
   cada país de tal forma que países com fronteira não tenham a
   mesma cor.
   predicado
   colours(country_colour_list)  % [(brasil,yellow),(bolivia,red), (uruguai,blue),...]
*/
% representação de vizinhança
ngb(brasil,[paraguai,bolivia,uruguai,argentina]).
ngb(paraguai,[bolivia,brasil,chile]).
ngb(bolivia,[paraguai,brasil,peru,chile]).
ngb(argentina,[uruguai,brasil,chile]).
ngb(chile,[argentina,[paraguai,bolivia]).
ngb(peru,[bolivia,brasil]).
```

```
colours([]).

colours([(Country,Colour)|Rest]):-
        colours(Rest),
        member(Colour,[yellow,blue,red,green]),
        not(samecolorNeighbour(Country,Country1,Colour,Rest)).

samecolorNeighbour(Country,Country1,Colour,Rest) :-
        member((Country1,Colour),Rest),
        neighbour(Country,Country1).

neighbour(Country,Country1):-
        ngb(Country,Neighbours),
        member(Country1,Neighbours).
```

```
go :-
        colours([(brasil,A),(argentina,B),(uruguai,C),(chile,D)]),
        write('brasil '),write(A),
        write('; argentina '),write(B),
        write('; uruguai '),write(C),
        write('; chile '),write(D),nl,
        fail.
go.
go1 :-  %último na lista o com maior n. de vizinhos
        colours([(uruguai,C),(chile,D),(argentina,B),(brasil,A)]),
        write('brasil '),write(A),
        write('; argentina '),write(B),
        write('; uruguai '),write(C),
        write('; chile '),write(D),nl,
        fail.
go1.
```

Referências

- Capítulo 4 (Russel & Norvig)
- Capítulo 5 e 6 (Peter Flach)

Referências para Busca

- Ginsberg, M. *Essentials of Artificial Intelligence* Parte II, cap 3, 4 e 5
- Russel & Norvig *Artificial Intelligence A Modern Approach* Parte II, cap 3 e 4
- Winston

Anexo
técnicas adicionais típicas em Aprendizado de máquina

adptadas de

Simply Logical – Peter Flach

Detecção de loop

```
% busca depth-first com detecção de loop
search_df_loop([Goal|Rest],Visited,Goal):-
  goal(Goal).
search_df_loop([Current|Rest],Visited,Goal):-
  children(Current,Children),
  add_df(Children,Rest,Visited,NewAgenda),
  search_df_loop(NewAgenda,[Current|Visited],Goal).

add_df([],Agenda,Visited,Agenda).
add_df([Child|Rest],OldAgenda,Visited,[Child|NewAgenda]):-
  not element(Child,OldAgenda),
  not element(Child,Visited),
  add_df(Rest,OldAgenda,Visited,NewAgenda).
add_df([Child|Rest],OldAgenda,Visited,NewAgenda):-
  element(Child,OldAgenda),
  add_df(Rest,OldAgenda,Visited,NewAgenda).
add_df([Child|Rest],OldAgenda,Visited,NewAgenda):-
  element(Child,Visited),
  add_df(Rest,OldAgenda,Visited,NewAgenda).
```

Busca com Backtracking

```
% busca depth-first usando backtracking
search_bt(Goal,Goal):-
 goal(Goal).
search_bt(Current,Goal):-
 arc(Current,Child),
 search_bt(Child,Goal).

% Busca depth-first usando backtracking com
 limite de profundidade
search_d(D,Goal,Goal):-
 goal(Goal).
search_d(D,Current,Goal):-
 D>0, D1 is D-1,
 arc(Current,Child),
 search_d(D1,Child,Goal).
```

Iterative deepening

```
search_id(First,Goal):-
  search_id(1,First,Goal).     % start with depth 1

search_id(D,Current,Goal):-
  search_d(D,Current,Goal).
search_id(D,Current,Goal):-
  D1 is D+1,                   % increase depth
  search_id(D1,Current,Goal).
```

- Combina as vantagens da busca em largura (completa, menor caminho) com as da busca em profundidade (eficiência de memória)

Busca Best-first (informada)

```
search_bstf([Goal|Rest],Goal):-
  goal(Goal).
search_bstf([Current|Rest],Goal):-
  children(Current,Children),
  add_bstf(Children,Rest,NewAgenda),
  search_bstf(NewAgenda,Goal).

% add_bstf(A,B,C) <- C contains the elements of
  A and B
%                    (B and C sorted according
  to eval/2)
add_bstf([],Agenda,Agenda).
add_bstf([Child|Children],OldAgenda,NewAgenda):
  -
  add_one(Child,OldAgenda,TmpAgenda),
  add_bstf(Children,TmpAgenda,NewAgenda).

% add_one(S,A,B) <- B is A with S inserted acc.
  to eval/2
add_one(Child,OldAgenda,NewAgenda):-
  eval(Child,Value),
```

```
search_beam(Agenda,Goal):-
    search_beam(1,Agenda,[],Goal).

search_beam(D,[],NextLayer,Goal):-
    D1 is D+1,
    search_beam(D1,NextLayer,[],Goal).
search_beam(D,[Goal|Rest],NextLayer,Goal):-
    goal(Goal).
search_beam(D,[Current|Rest],NextLayer,Goal):-
    children(Current,Children),
    add_beam(D,Children,NextLayer,NewNextLayer),
    search_beam(D,Rest,NewNextLayer,Goal).
```

- Aqui, o número de filhos adicionados ao “feixe” é dependente da profundidade D do nó.
- Para manter a profundidade como uma variável “global”, a busca é camada por camada.

Beam search

Hill-climbing

```
search_hc(Goal,Goal):-
 goal(Goal).
search_hc(Current,Goal):-
 children(Current,Children),
 select_best(Children,Best),
 search_hc(Best,Goal).
```

```
% hill_climbing as a variant of best-first search
search_hc([Goal|_],Goal):-
  goal(Goal).
search_hc([Current|_],Goal):-
  children(Current,Children),
  add_bstf(Children,[],NewAgenda),
  search_hc(NewAgenda,Goal).
```

- Limita o tamanho da Agenda em 1.
- Chamada greedy search ou busca gulosa, uma vez que não usa backtracking, ($f(n)=h(n)$).
- Pode atingir um máximo local, e não alcançar o máximo global.

Simulated annealing

- tenta evitar ficar preso em um máximo local