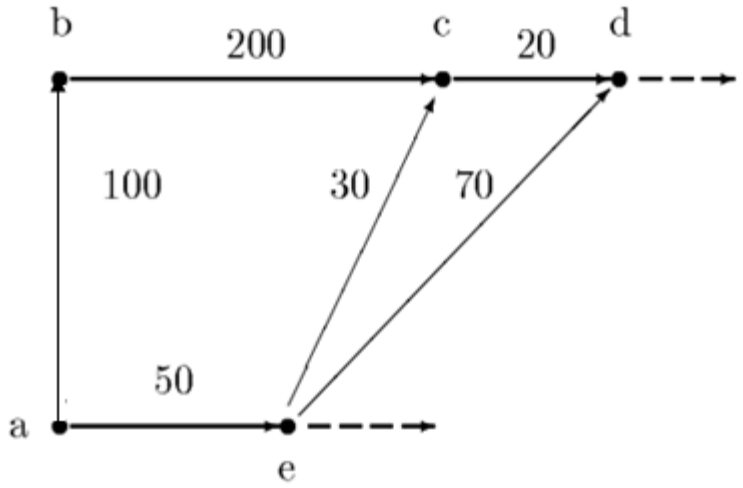
Busca em Grafos

Esta aula trata da Busca em Grafos e alguns problemas clássicos relacionados a esse tema

Inteligência Artificial

Busca em Grafos

```
va_para(a,b,100).
va_para(a,e,50).
va_para(b,c,200).
va_para(c,d,20).
va_para(e,c,30).
va_para(e,d,70).
```

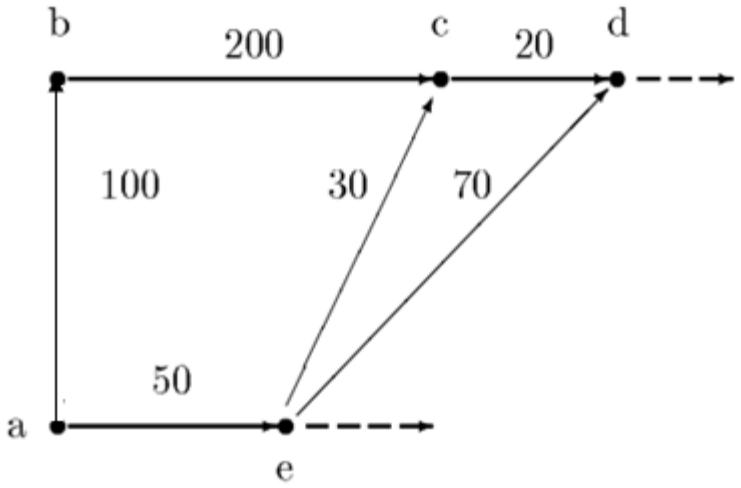


Como encontrar todas as cidades que estão conectadas e a distância total entre elas?

```
%passo 1: calcular distância total entre duas cidades
pode_ir(X,X,0).
pode_ir(Ini, Fim, Dist) :-
            va_para(Ini, Adj, D1),
            pode_ir(Adj, Fim, D2),
            Dist is D1 + D2.
```

Busca em Grafos

```
va_para(a,b,100).
va_para(a,e,50).
va_para(b,c,200).
va_para(c,d,20).
va_para(e,c,30).
va_para(e,d,70).
```




```
?- pode_ir(a,c,X).
X = 300;
X = 80;
no
```

```
pode_ir(X,X,0).
pode_ir(Ini, Fim, Dist) :-
                va_para(Ini, Adj, D1),
                pode_ir(Adj, Fim, D2),
                Dist is D1 + D2.
```

Busca em Grafos

```
%passo 2: encontrar um caminho Cam entre as cidades Ini
e Fim com distância total Dist se é possível um caminho
Cam, partindo de Ini, com lista de cidades [Fim],
distância restante até final 0 e distância total Dist
caminho(Ini,Fim,Dist,Cam):-
      caminho1(Ini,[Fim],0,Dist,Cam).

%passo 3: concluir montagem do caminho quando cidade
inicial Interm for cabeça da lista de cidades
caminho1(Cid,[Cid|Cids],Dist, Dist, [Cid|Cids]).
caminho1(Ini,[Adj|Cids],Dist, DistF, CamF):-
      va_para(Interm,Adj,D1),
      D2 is Dist+D1,
      caminho1(Ini,[Interm,Adj|Cids],D2,DistF,CamF).
```

Busca em Grafos

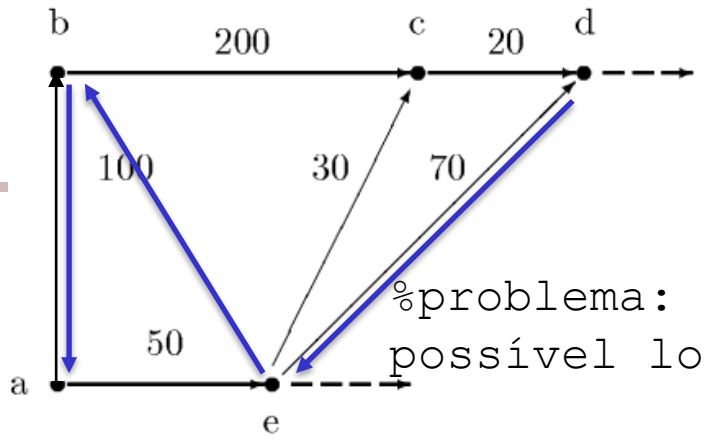
```
caminho(Ini,Fim,Dist,Cam):-
            caminho1(Ini,[Fim],0,Dist,Cam).

caminho1(Cid,[Cid|Cids],Dist, Dist, [Cid|Cids]).
caminho1(Ini,[Adj|Cids],Dist,DistF,CamF):-
            va_para(Interm,Adj,D1),
            D2 is Dist+D1,
            caminho1(Ini,[Interm,Adj|Cids],D2,DistF,CamF).
```

```
?- caminho(a,c,Dist,Cam).
Dist = 300
Cam = [a,b,c]
...
```

```
va_para(a,b,100).
va_para(a,e,50).
va_para(b,c,200).
va_para(c,d,20).
va_para(e,c,30).
va_para(e,d,70).
```

Busca em Grafos




```
va_para(a,b,100).   va_para(d,e,70).
va_para(a,e,50).    va_para(b,a,100).
va_para(b,c,200).   va_para(e,b,150).
va_para(c,d,20).
va_para(e,c,30).    pertence1(X,[X|_]):-!.
va_para(e,d,70).    pertence1(X,[_|C]):-
                              pertence1(X,C).
```

%problema:
possível loop

```
caminho(Ini,Fim,Dist,Cam):-
            caminho1(Ini,[Fim],0,Dist,Cam).

caminho1(Cid,[Cid|Cids],Dist, Dist, [Cid|Cids]).
caminho1(Ini,[Adj|Cids],Dist,DistF,CamF):-
            va_para(Interm,Adj,D1),
            not pertence(Interm,[Adj|Cids]),
            D2 is Dist+D1,
            caminho1(Ini,[Interm,Adj|Cids],D2,DistF,CamF).
```

Busca em Profundidade (Prolog)

```
% resolva(No,Solucao) Solucao é um caminho acíclico (na
  ordem reversa) entre nó inicial No e nó final
resolva(No,Solucao) :-
      depthFirst([],No,Solucao).

% depthFirst(Caminho,No,Solucao) estende o caminho
  [No|Caminho], com nó atual No, até um nó final obtendo
  Solucao
depthFirst(Caminho,No,[No|Caminho]) :-
      final(No). %fato da base: No é final
depthFirst(Caminho,No,S) :-
      s(No,No1), %fato da base: transição de No para No1
      \+ pertence(No1,Caminho),          % evita um ciclo
      depthFirst([No|Caminho],No1,S).

pertence(E,[E|_]).
pertence(E,[_|T]) :-
      pertence(E,T).
```


7

Caixas D'água

- Dada a base de dados a seguir, a qual contém o estoque de caixas d'água de uma loja

```
caixa(Modelo, [Altura, Largura, Profundidade]).
```

- Quais os predicados para:
  - Determinar o volume da caixa.
  - O modelo da caixa com menor capacidade em volume.
  - O volume total que poderia ser armazenado caso todas as caixas disponíveis na loja fossem preenchidas.

Caixas D'água

```
%Declaração de fatos
%caixa(Modelo, [Altura, Largura, Profundidade])
caixa(m1, [1,1,1]).
caixa(m2, [2,2,2]).
caixa(m21, [3,3,2]).
caixa(m22, [3,2,2]).
caixa(m3, [3,3,3]).
caixa(m5, [5,5,5]).
```

Caixas D'água

```
caixa(m1, [1,1,1]).
caixa(m2, [2,2,2]).
caixa(m21, [3,3,2]).
caixa(m22, [3,2,2]).
caixa(m3, [3,3,3]).
caixa(m5, [5,5,5]).
```

```
% Predicado para determinar volume
vol(Mod, Vol):-
        caixa(Mod, [A,L,P]),
        Vol is A*L*P.

% Predicado para determinar caixa com menor capacidade
menor_caixa(M):-
        vol(M, Vol),
        not (vol(_, Vol2), Vol2 < Vol).

%Predicado para determinar volume total
vol_total(VT):-
        findall(Vol, vol(M, Vol), LVol),
        soma(LVol, VT).

% Predicado auxiliar
soma([],0).
soma([Elem|Cauda],S):-
        soma(Cauda, S1),
        S is S1+Elem.
```

Slides baseados em:

Bratko, I.;
Prolog Programming for Artificial Intelligence,
3rd Edition, Pearson Education, 2001.

Clocksin, W.F.; Mellish, C.S.;
Programming in Prolog,
5th Edition, Springer-Verlag, 2003.

Programas Prolog para o
Processamento de Listas e Aplicações,
Monard, M.C & Nicoletti, M.C., ICMC-USP, 1993

Adaptado por Huei Diana Lee