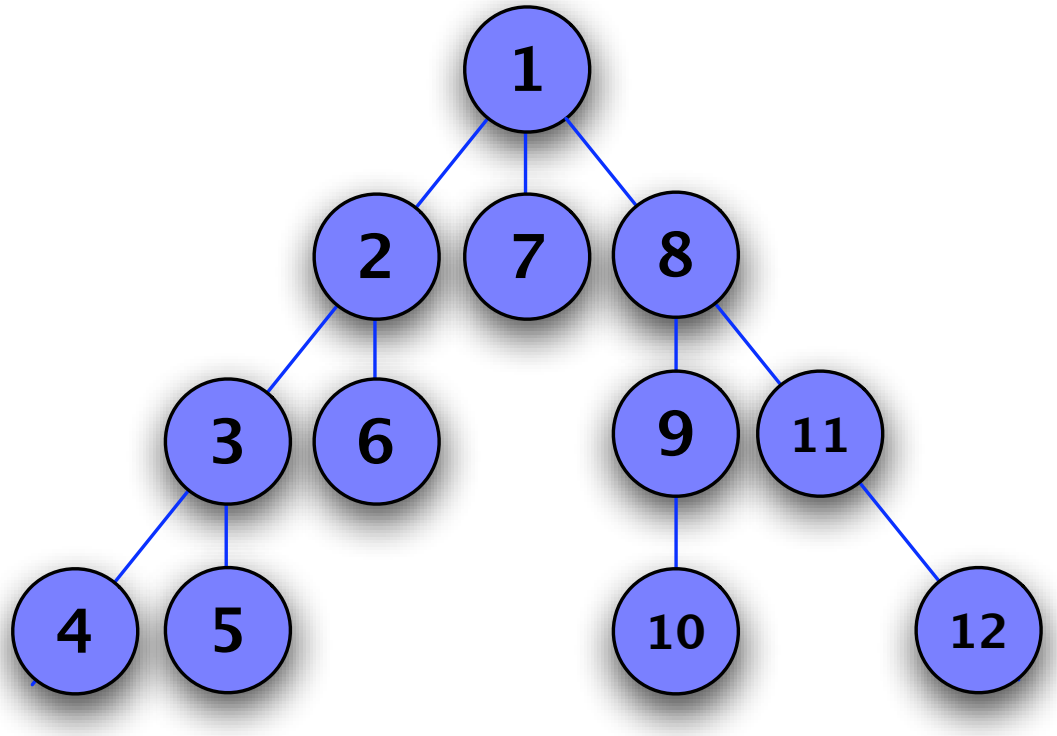
Métodos de Pesquisa

Aula 1

Primeiro em Profundidade

- A ideia inerente a este método é a de tentar avançar de estado para estado até se encontrar a solução.
- Método adequado se as opções tomadas forem na direcção correcta; pode não o ser se a direcção escolhida não for a melhor.
- O método é também adequado para problemas que tenham várias soluções - nesses casos aumenta a probabilidade de se optar por um bom caminho.
- Vantagem - requisitos de memória limitados.

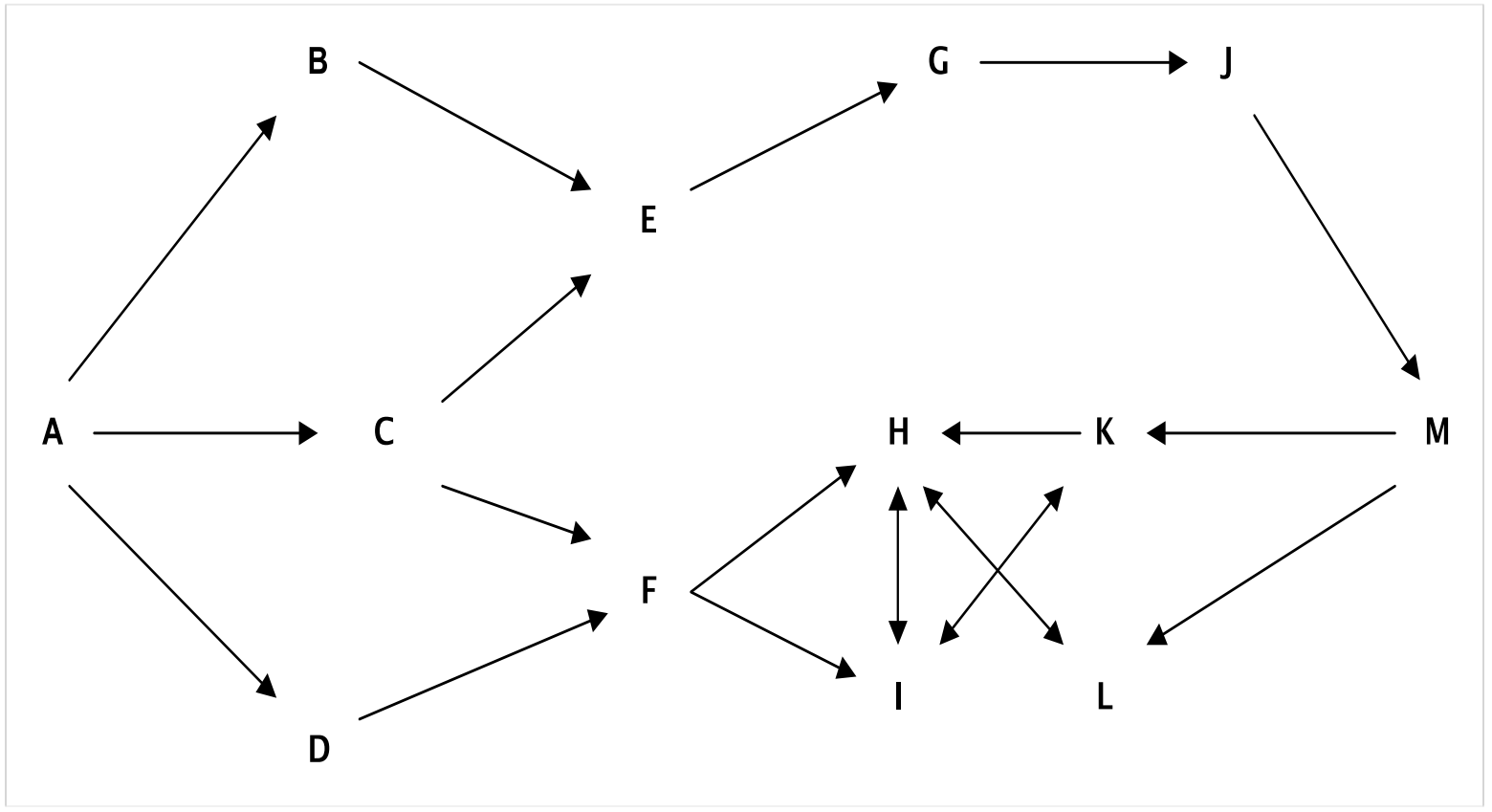
Primeiro em Profundidade



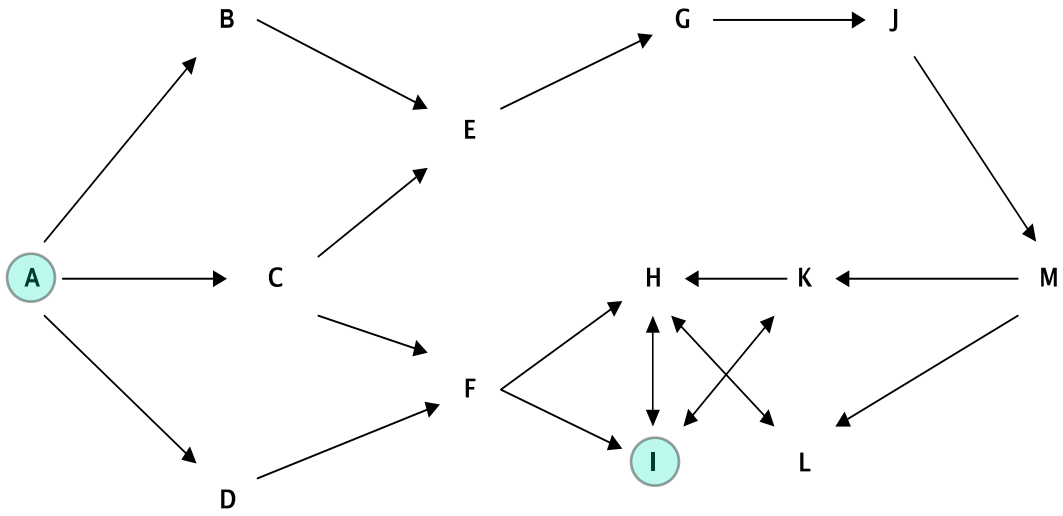
Ordem pela qual os nós são expandidos

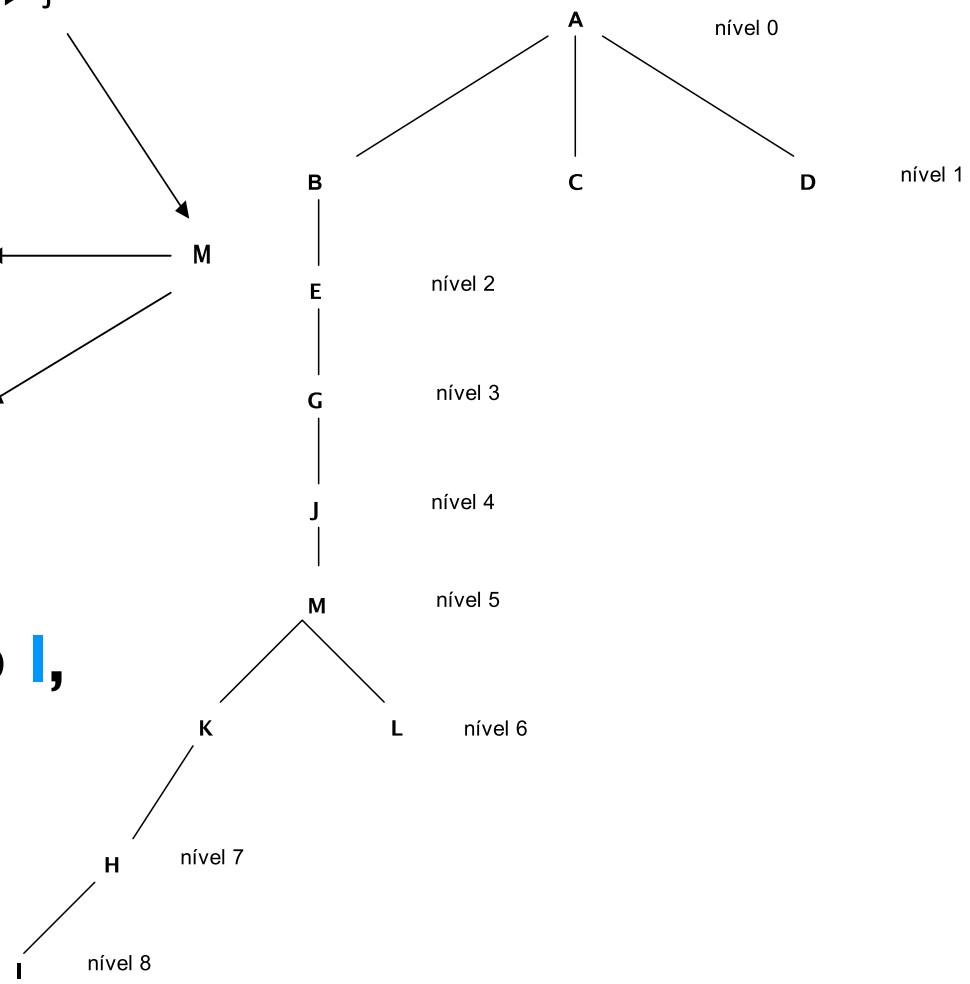
Primeiro em Profundidade

Consideremos o seguinte grafo:



Primeiro em Profundidade

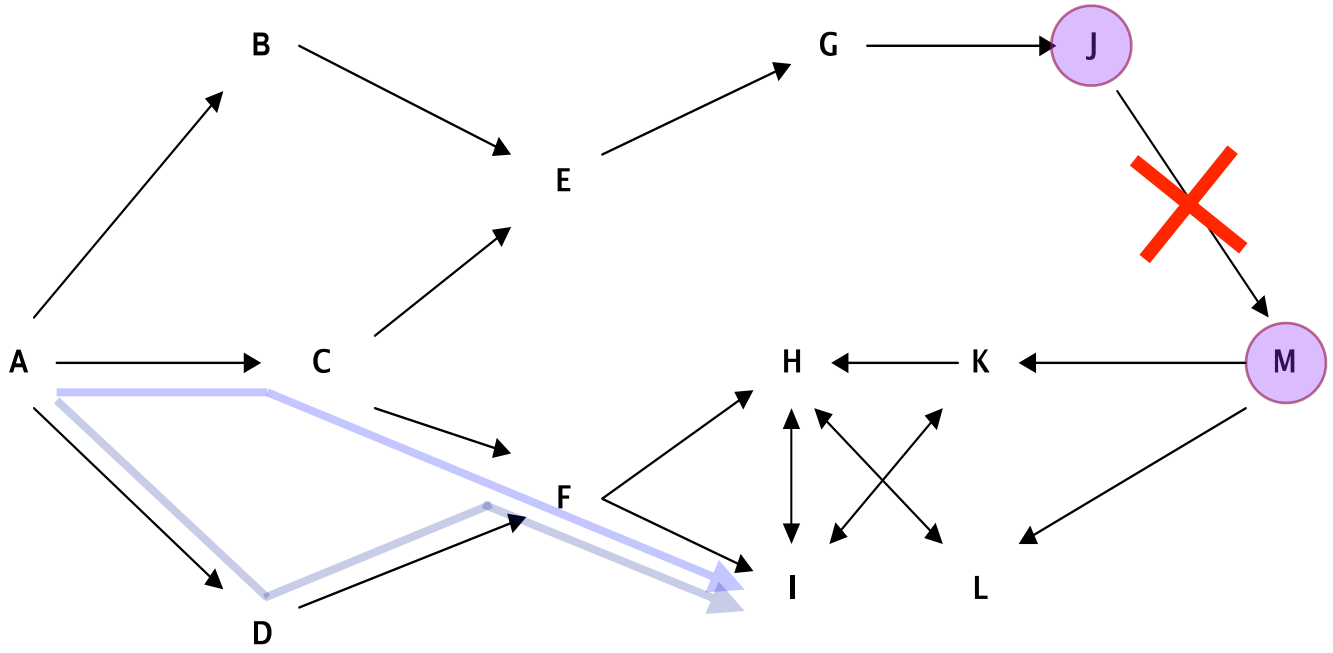




Para ir do nó A até ao nó I, a árvore de pesquisa gerada poderia ser:

Primeiro em Profundidade

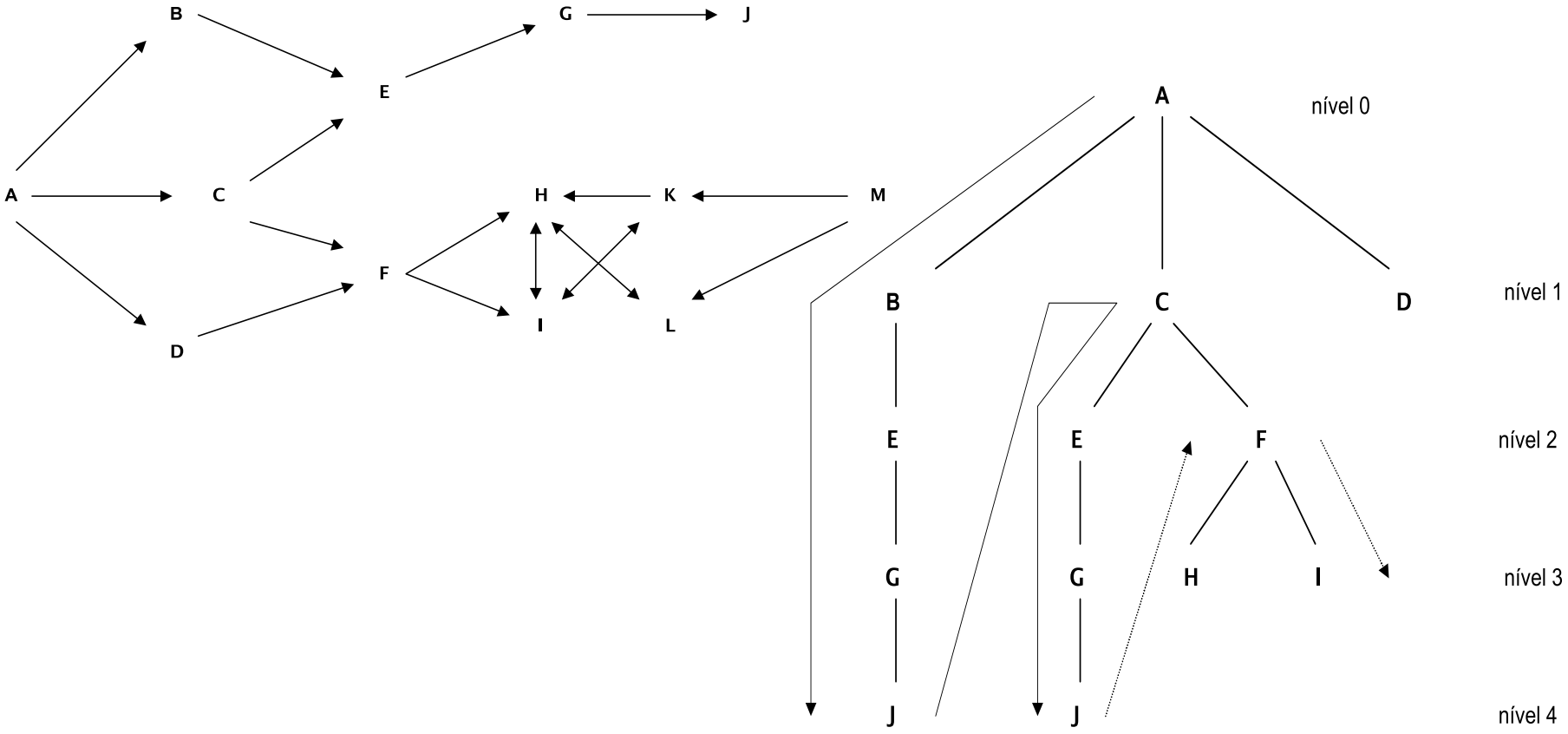
- Não há garantia de que o método permita obter a melhor solução, ou a solução ao nível mais próximo da raiz.
- Existem soluções ao nível 3 como o caminho A-C-F-I ou A-D-F-I.



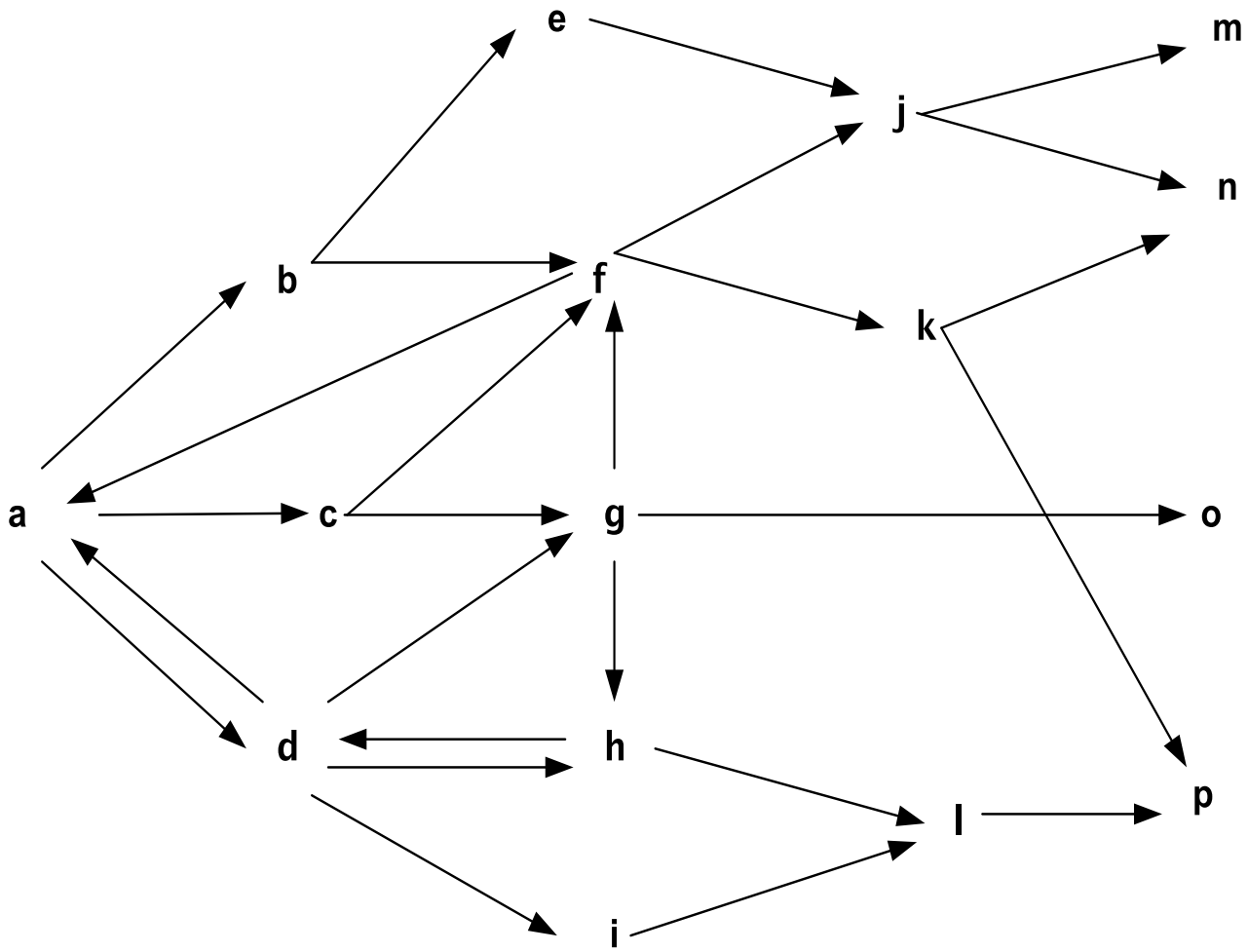
E se fosse retirada a ligação entre a cidade J e M ?

Primeiro em Profundidade

O método 1º em Profundidade com retrocesso é o algoritmo usado internamente pela linguagem Prolog



Um exemplo



liga(a,b).
liga(a,c).
liga(a,d).
liga(b,e).
liga(b,f).
liga(c,f).
liga(c,g).
liga(d,a)
liga(d,g).
liga(d,h).
liga(d,i).
liga(e,j).
liga(f,a).
liga(f,j).
liga(f,k).
liga(g,f).
liga(g,o).
liga(g,h).
liga(h,d).
liga(h,l).
liga(i,l).
liga(j,m).
liga(j,n).
liga(k,n).
liga(k,p).
liga(l,p).

Primeiro em Profundidade

Implementação

Lista auxiliar c/ os nós visitados até ao momento

```
go(Orig,Dest,L) :- go(Orig,Dest,[Orig],L).

go(Dest,Dest,_,[Dest]).
go(Orig,Dest,LV,[Orig|L]) :-
    liga(Orig,X),
    \+ member(X,LV),
    go(X,Dest,[X|LV],L).
```

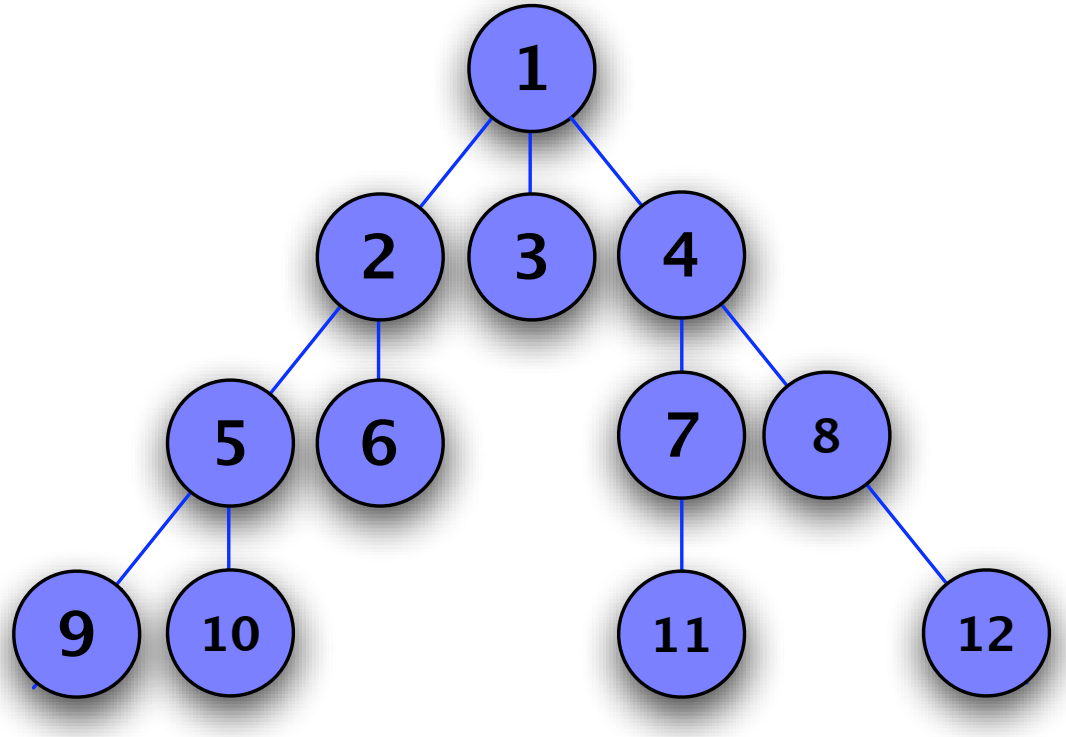
Para evitar visitar nós já visitados

```
?- go(a, m, L).
L=[a,b,e,j,m]
```

Primeiro em **Largura**

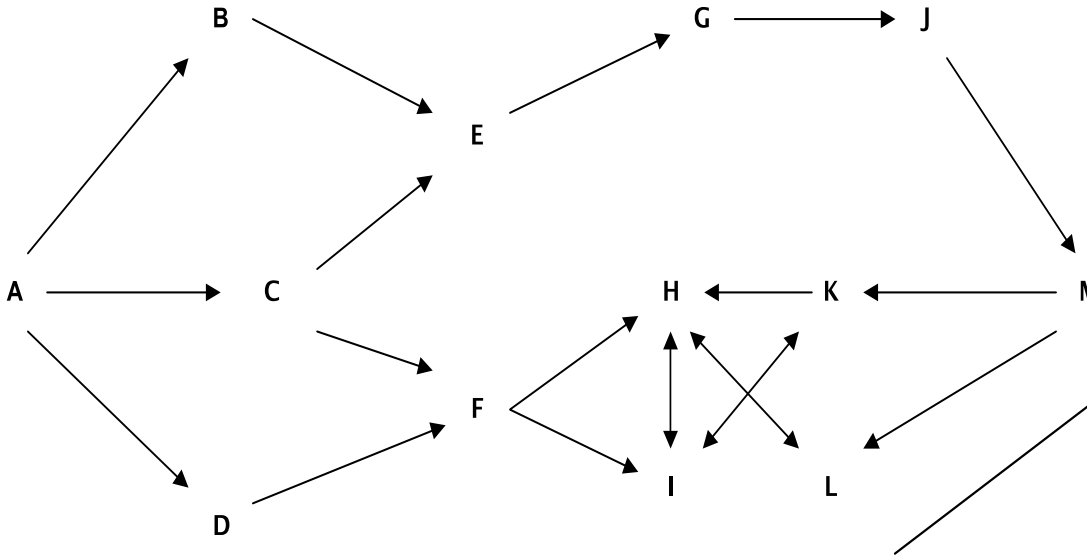
- Neste método só é possível efectuar a pesquisa no nível N da árvore se todos os nós do nível N-1 tiverem já sido explorados
- A árvore é explorada **transversalmente**, derivando daí o nome do método
- Garante a obtenção da solução ao nível mais próximo da raiz – não quer dizer que seja a melhor solução
- Pode requerer muita memória e tempo para resolver problemas mais complexos

Primeiro em Largura

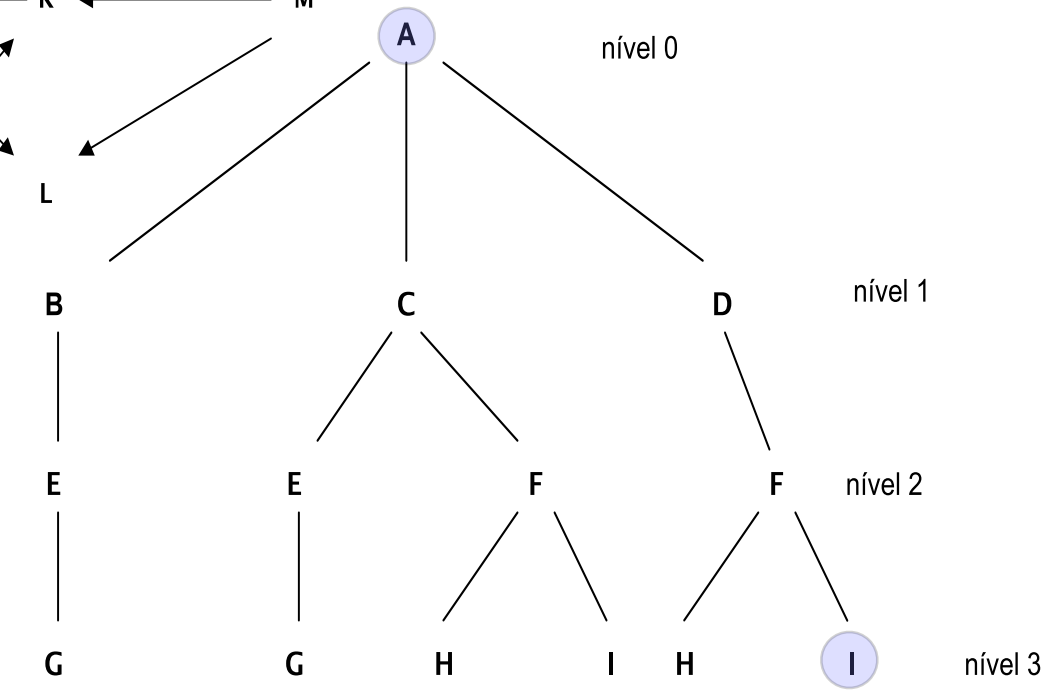


Ordem pela qual os nós são expandidos

Primeiro em Largura

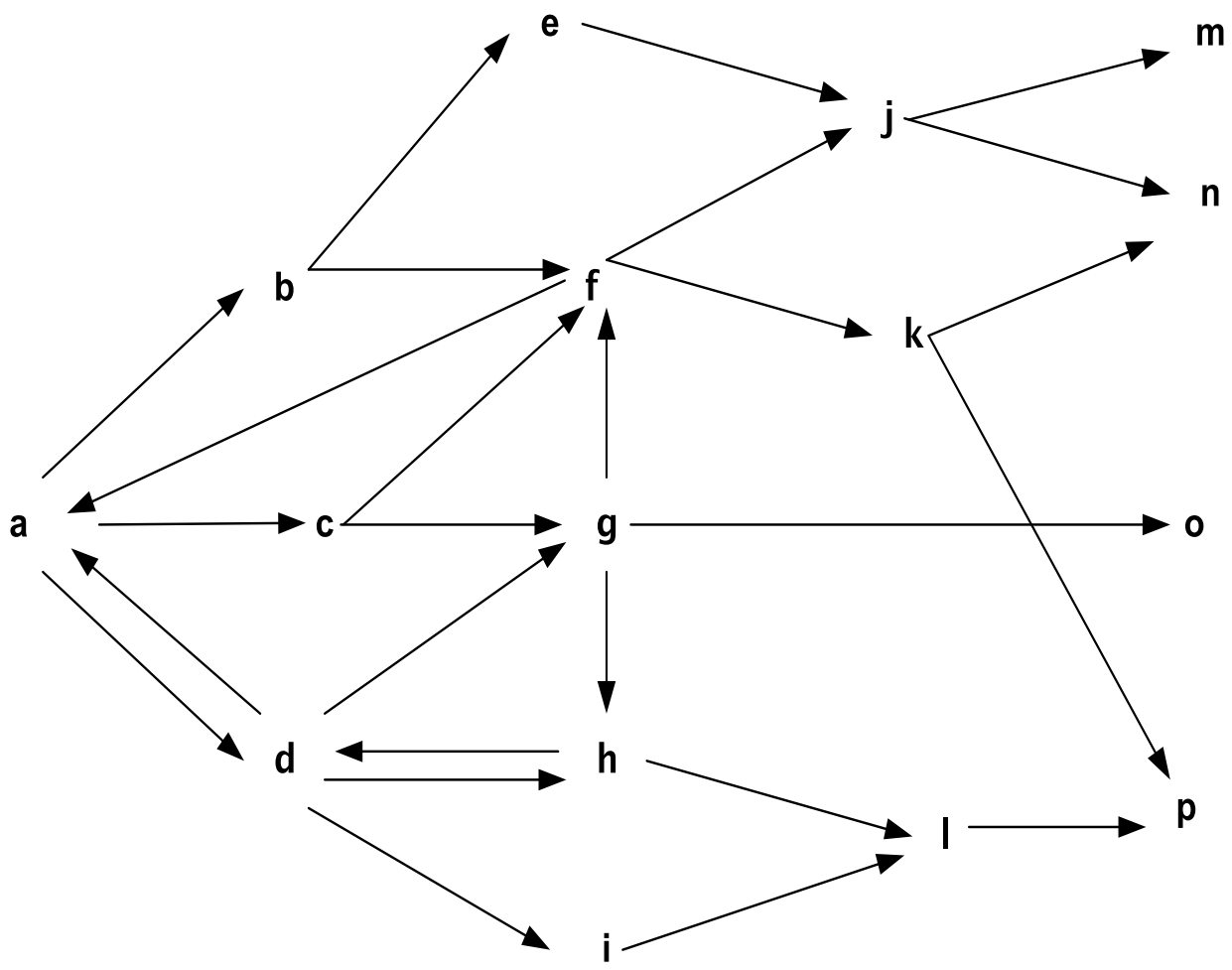


Vejamos como se desenvolve o método para o mesmo exemplo dado anteriormente:



Percurso de A até I

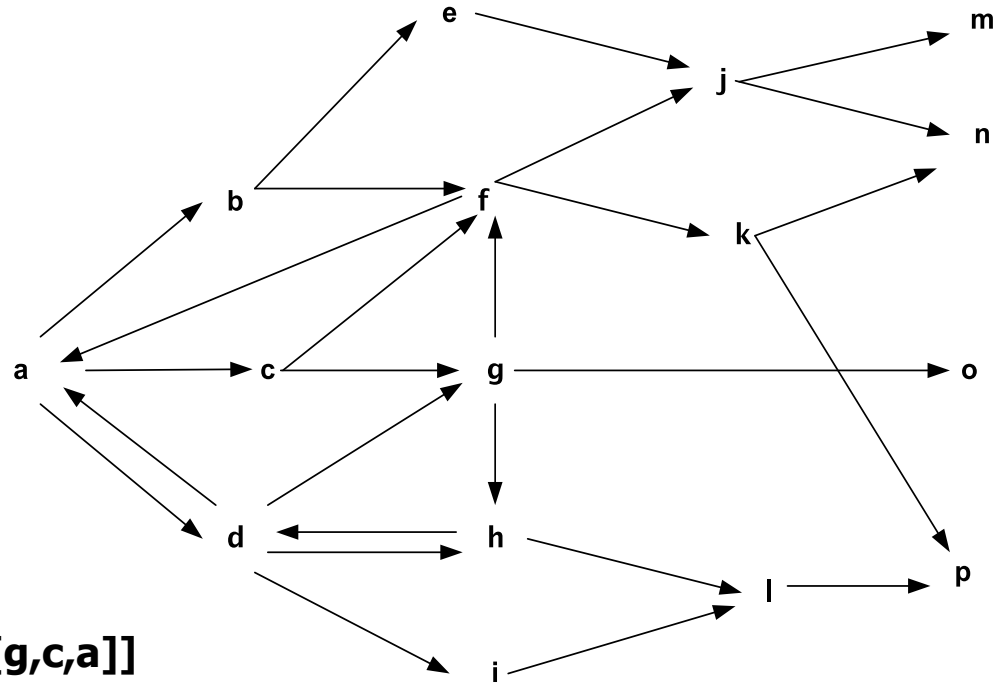
Um exemplo




```
liga(a,b).
liga(a,c).
liga(a,d).
liga(b,e).
liga(b,f).
liga(c,f).
liga(c,g).
liga(d,a).
liga(d,g).
liga(d,h).
liga(d,i).
liga(e,j).
liga(l,p).
liga(f,a).
liga(f,j).
liga(f,k).
liga(g,f).
liga(g,o).
liga(g,h).
liga(h,d).
liga(h,l).
liga(i,l).
liga(j,m).
liga(j,n).
liga(k,n).
liga(k,p).
```

Lista de Percursos

Lista de Percursos após expansão do nó **a**



?- go(a,j,L).

NPerc[[b,a],[c,a],[d,a]]

NPerc[[c,a],[d,a],[e,b,a],[f,b,a]]

NPerc[[d,a],[e,b,a],[f,b,a],[f,c,a],[g,c,a]]

...

NPerc[[i,d,a],[j,e,b,a],[j,f,b,a],[k,f,b,a],[j,f,c,a],[k,f,c,a],[f,g,c,a],[o,g,c,a], ...]

NPerc[[j,e,b,a],[j,f,b,a],[k,f,b,a],[j,f,c,a],[k,f,c,a],[f,g,c,a],[o,g,c,a],[h,g,c,a],[f,g,d,a], ...]

L = [a,b,e,j]

Percursos resultantes da expansão do nó **b**

Um exemplo

```
go(Orig,Dest,Perc) :-
   go1([[Orig]],Dest,P), inverte(P,Perc).

go1([Prim|Resto],Dest,Prim) :- Prim=[Dest|_].
go1([[Dest|T]|Resto],Dest,Perc) :-
     !, go1(Resto,Dest,Perc).
go1([[Ult|T]|Outros],Dest,Perc):-
    findall([Z,Ult|T], proximo_no(Ult,T,Z),Lista),
    append(Outros,Lista,NPerc),
    go1(NPerc,Dest,Perc).

proximo_no(X,T,Z) :- liga(X,Z), \+ member(Z,T).
```

Próximo nó

Nó em expansão

```
[[b,a],[c,a],[d,a]]
```

Exemplo (cont.)

go1([[a|[ ]]|[ ]]

findall([Z, Ult|T], proximo_no(Ult,T,Z), Lista)

b a [ ]
c
d

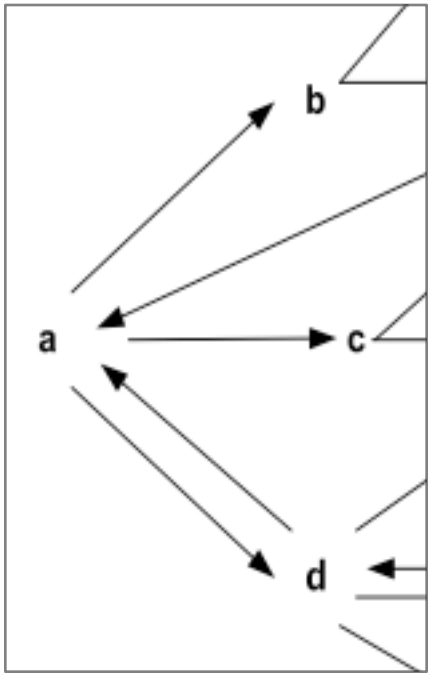
proximo_no(X,T,Z) :- liga(X,Z), \+ member(Z,T).

proximo_no(a, [ ], **b**):-liga(a,b), \+ member(b,[]).
proximo_no(a, [ ], **c**):-liga(a,c), \+ member(c,[]).
proximo_no(a, [ ], **d**):-liga(a,d), \+ member(d,[]).

Lista - [[b,a|[ ]], [c,a|[ ]], [d,a|[ ]]]
Lista - [[b,a], [c,a], [d,a]]

Exemplo (cont.)

```
go1([[Ult|T]|Outros],Dest,Perc) :-
        findall( [Z,Ult|T] ,
                 proximo_no(Ult,T,Z),
                 Lista ),
    append(Outros,Lista,NPerc),
    go1(NPerc,Dest,Perc).
```

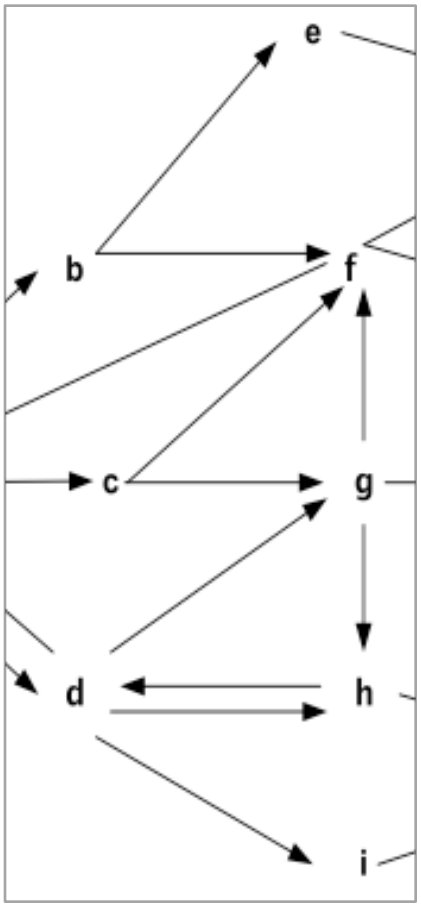


Após o findall:

```
Ult = a
T = []
Outros = []
Dest = p
Lista = [[b,a],[c,a],[d,a]]
NPerc = [[b,a],[c,a],[d,a]]
```

Nó a ser expandido

Exemplo (cont.)




```
NPerc = [[b,a],[c,a],[d,a]]
```

```
go1([[Ult|T]|Outros],Dest,Perc):-
    findall( [Z,Ult|T] ,
        proximo_no(Ult,T,Z), Lista ),
        append(Outros,Lista,NPerc),
        go1(NPerc,Dest,Perc).
```

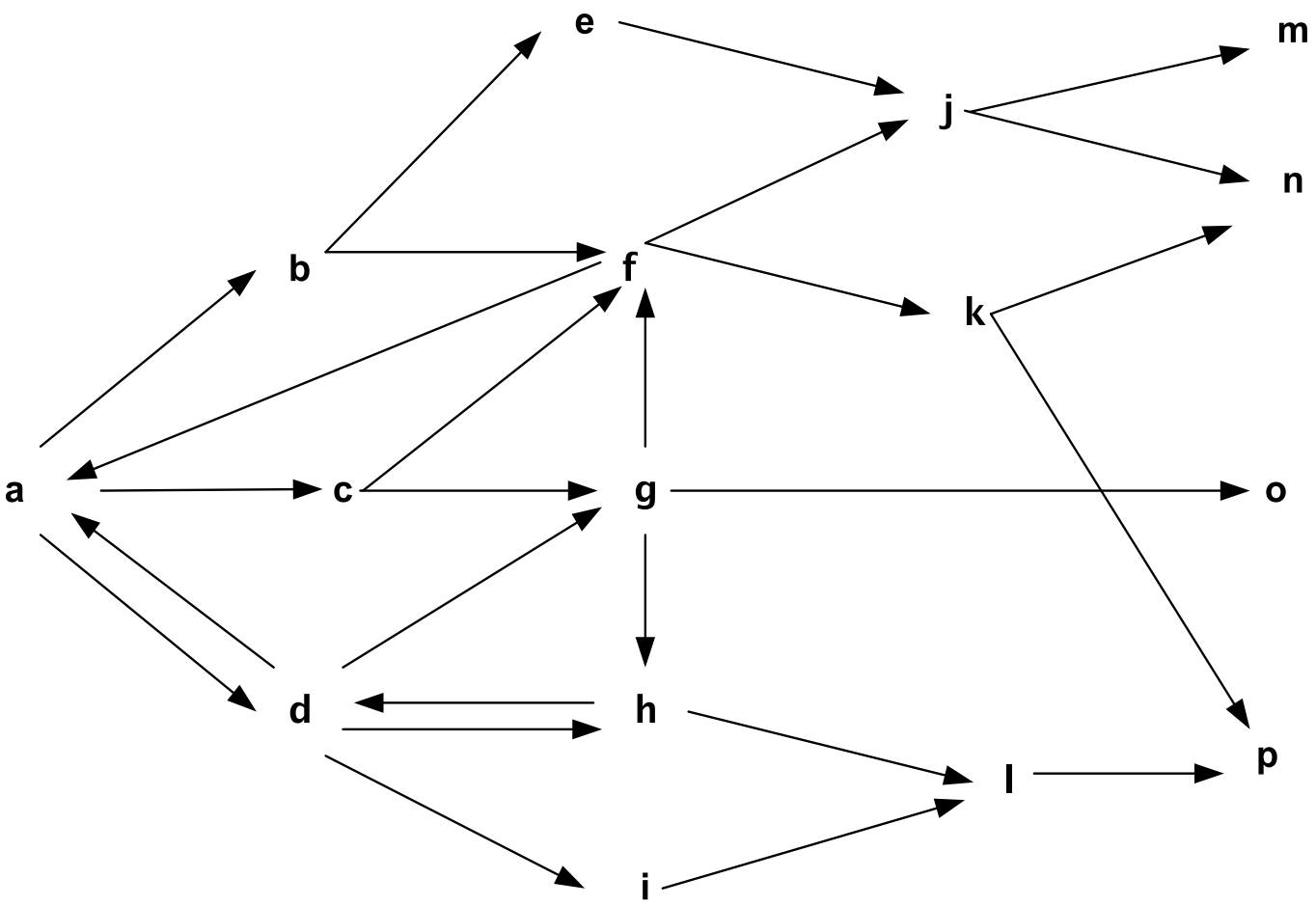
```
Ult = b
T = [a]
Outros = [[c,a],[d,a]]
Dest = p
Lista = [[e,b,a],[f,b,a]]
NPerc = [[c,a],[d,a],[e,b,a],[f,b,a]]
```

Nó a ser expandido

Um exemplo: considerando uma pequena alteração

```
go(Orig,Dest,Perc):- go1([[Orig]],Dest,P), inverte(P,Perc).

go1([Prim|Resto],Dest,Prim):- Prim=[Dest|_].
go1([[Dest|T]|Resto],Dest,Perc):- !, go1(Resto,Dest,Perc).
go1([[Ult|T]|Outros],Dest,Perc):-
        findall([Z,Ult|T],proximo_no(Ult,T,Z),Lista),
        % append(Outros,Lista,NPerc), pesquisa 1º em largura
        append(Lista,Outros,NPerc),
        go1(NPerc,Dest,Perc).

proximo_no(X,T,Z):- liga(X,Z), not member(Z,T).

inverte(L,LI):-inverte(L,[],LI).
inverte([],LI,LI).
inverte([H|T],L,LI):-inverte(T,[H|L],LI).
```

Um exemplo: soluções obtidas



?- go(a,j,L).

L = [a,b,e,j] ;

L = [a,b,f,j] ;

L = [a,c,f,j] ;

L = [a,c,g,f,j] ;

L = [a,d,g,f,j]

Um exemplo

```
go(Orig,Dest,Perc):- go1([[Orig]],Dest,P), inverte(P,Perc).

go1([Prim|Resto],Dest,Prim):- Prim=[Dest|_].
go1([[Dest|T]|Resto],Dest,Perc):- !, go1(Resto,Dest,Perc).
go1([[Ult|T]|Outros],Dest,Perc):-
        findall([Z,Ult|T],proximo_no(Ult,T,Z),Lista),
        append(Lista,Outros,NPerc), % pesquisa 1º em profundidade
        % append(Outros,Lista,NPerc), pesquisa 1º em largura
        write('NPerc:'), write(NPerc),nl,
        go1(NPerc,Dest,Perc).

proximo_no(X,T,Z):- liga(X,Z), not member(Z,T).

inverte(L,LI):-inverte(L,[],LI).
inverte([],LI,LI).
inverte([H|T],L,LI):-inverte(T,[H|L],LI).
```

Um exemplo – 1º em **Profundidade** a->j

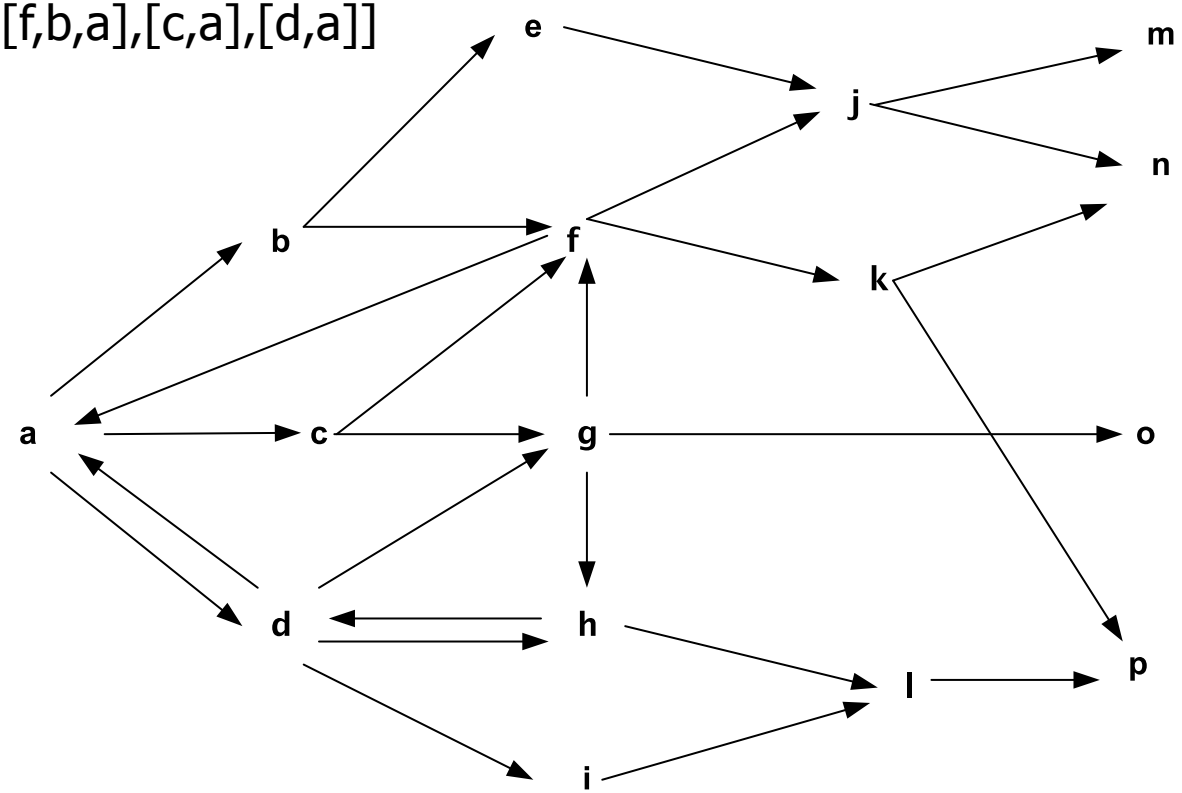
?- go(a,j,L).

NPerc[[b,a],[c,a],[d,a]]

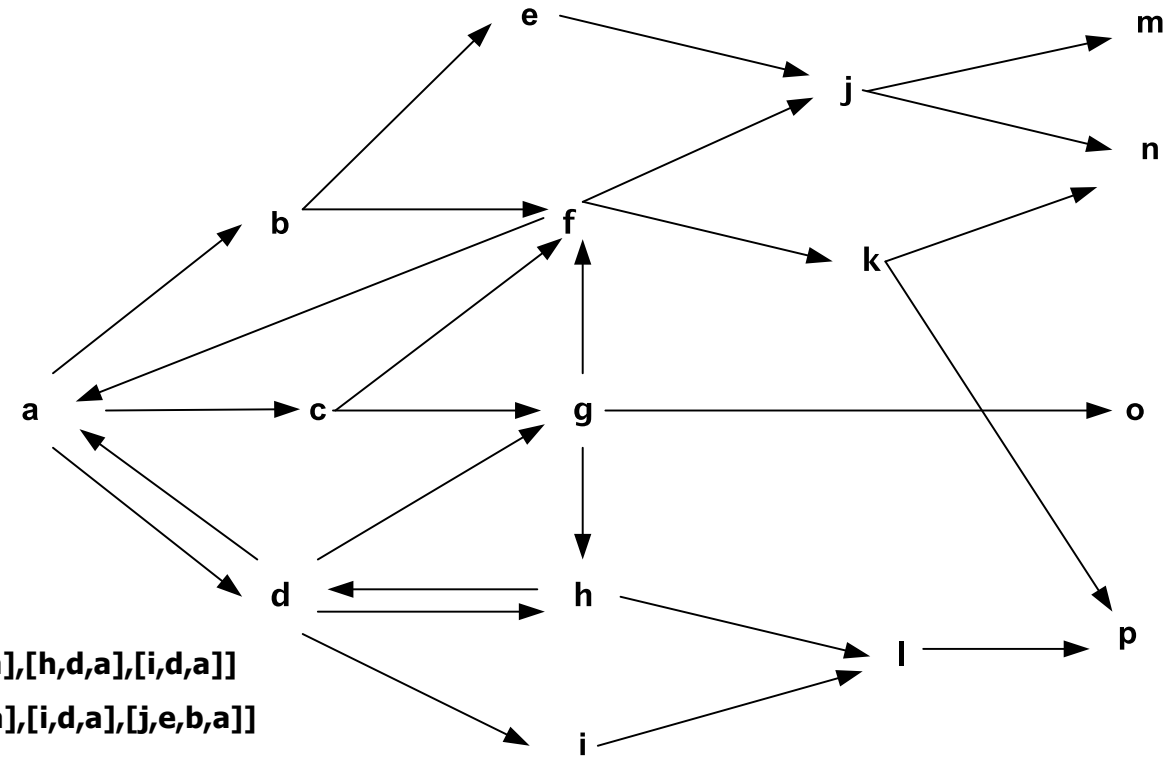
NPerc[[e,b,a],[f,b,a],[c,a],[d,a]]

NPerc[[j,e,b,a],[f,b,a],[c,a],[d,a]]

L = [a,b,e,j]



Um exemplo – Primeiro em **Largura** a->j



?- go(a,j,L).

NPerc[[b,a],[c,a],[d,a]]

NPerc[[c,a],[d,a],[e,b,a],[f,b,a]]

NPerc[[d,a],[e,b,a],[f,b,a],[f,c,a],[g,c,a]]

NPerc[[e,b,a],[f,b,a],[f,c,a],[g,c,a],[g,d,a],[h,d,a],[i,d,a]]

NPerc[[f,b,a],[f,c,a],[g,c,a],[g,d,a],[h,d,a],[i,d,a],[j,e,b,a]]

NPerc[[f,c,a],[g,c,a],[g,d,a],[h,d,a],[i,d,a],[j,e,b,a],[j,f,b,a],[k,f,b,a]]

NPerc[[g,c,a],[g,d,a],[h,d,a],[i,d,a],[j,e,b,a],[j,f,b,a],[k,f,b,a],[j,f,c,a],[k,f,c,a]]

NPerc[[g,d,a],[h,d,a],[i,d,a],[j,e,b,a],[j,f,b,a],[k,f,b,a],[j,f,c,a],[k,f,c,a],[f,g,c,a],[o,g,c,a],[h,g,c,a]]

NPerc[[h,d,a],[i,d,a],[j,e,b,a],[j,f,b,a],[k,f,b,a],[j,f,c,a],[k,f,c,a],[f,g,c,a],[o,g,c,a],[h,g,c,a],[f,g,d,a],[o,g,d,a],[h,g,d,a]]

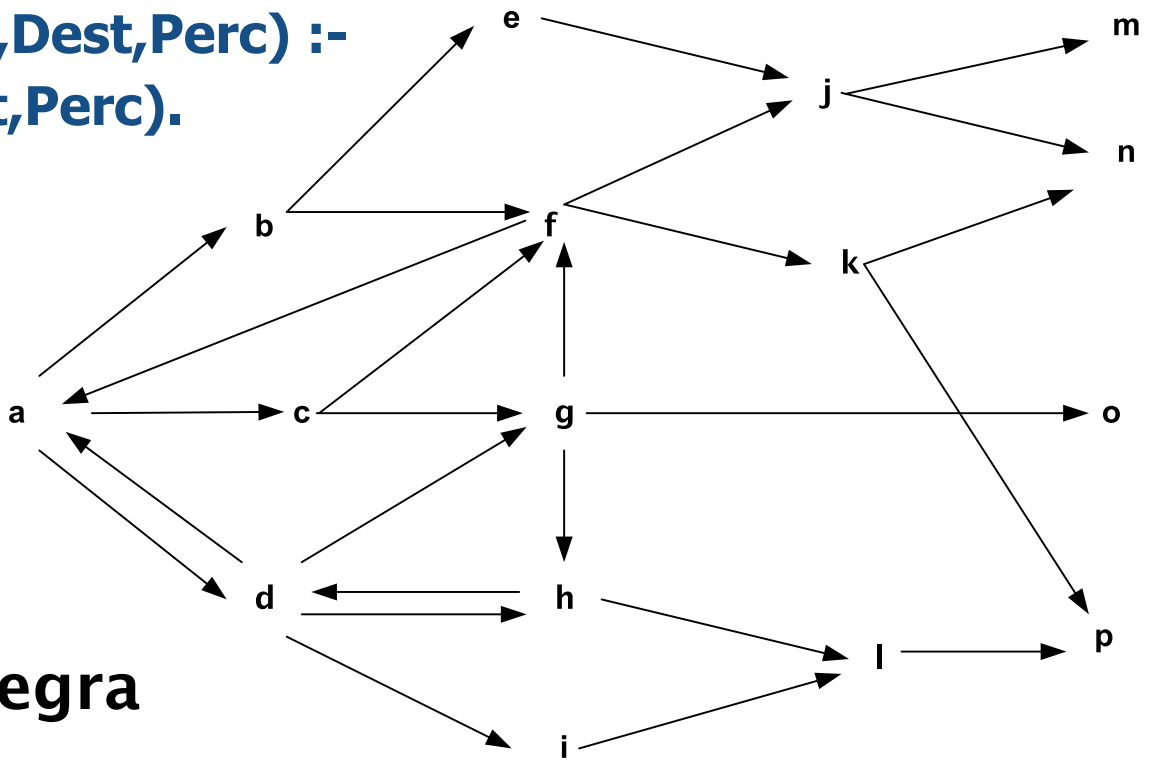
NPerc[[i,d,a],[j,e,b,a],[j,f,b,a],[k,f,b,a],[j,f,c,a],[k,f,c,a],[f,g,c,a],[o,g,c,a],[h,g,c,a],[f,g,d,a],[o,g,d,a],[h,g,d,a],[l,h,d,a]]

NPerc[[j,e,b,a],[j,f,b,a],[k,f,b,a],[j,f,c,a],[k,f,c,a],[f,g,c,a],[o,g,c,a],[h,g,c,a],[f,g,d,a],[o,g,d,a],[h,g,d,a],[l,h,d,a],[l,i,d,a]]

L = [a,b,e,j]

Um exemplo – Primeiro em Largura a->j

```
go1([Prim|Resto],Dest,Prim):- Prim=[Dest|_].
go1([[Dest|T]|Resto],Dest,Perc) :-
        !, go1(Resto,Dest,Perc).
```



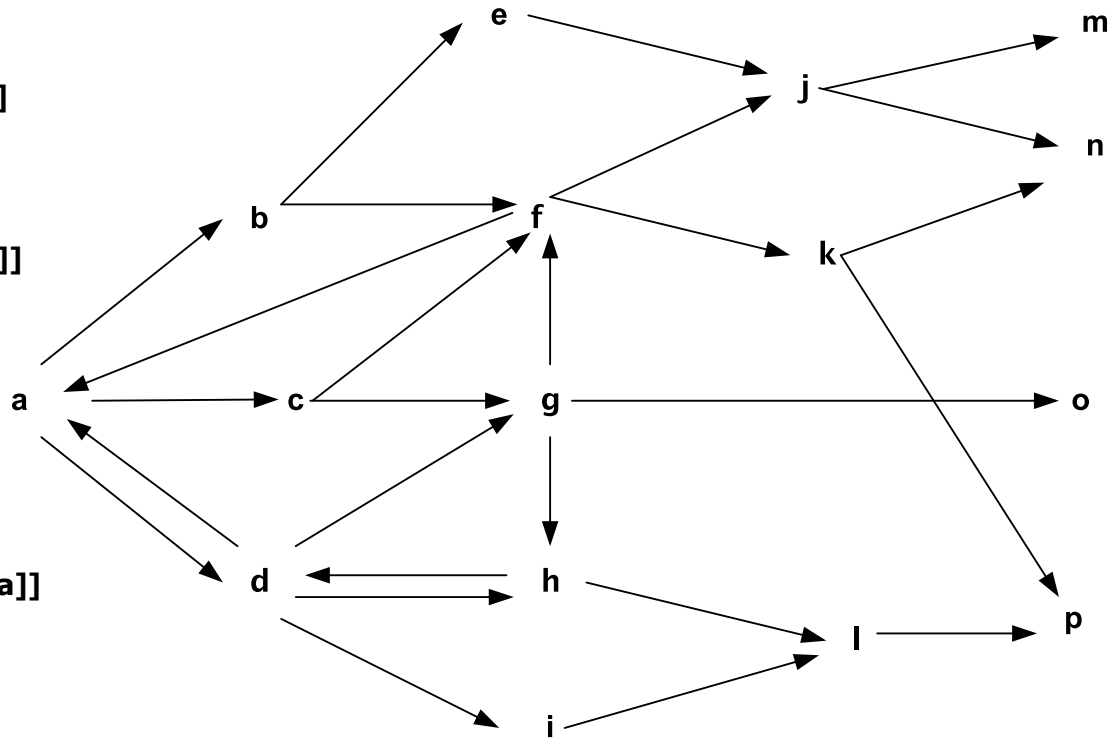
Explicação da 2ª regra

NPerc[[j,e,b,a],[j,f,b,a],[k,f,b,a],[j,f,c,a],[k,f,c,a],[f,g,c,a],[o,g,c,a],[h,g,c,a],[f,g,d,a],[o,g,d,a],[h,g,d,a],[l,h,d,a],[l,i,d,a]]

L = [a,b,e,j]

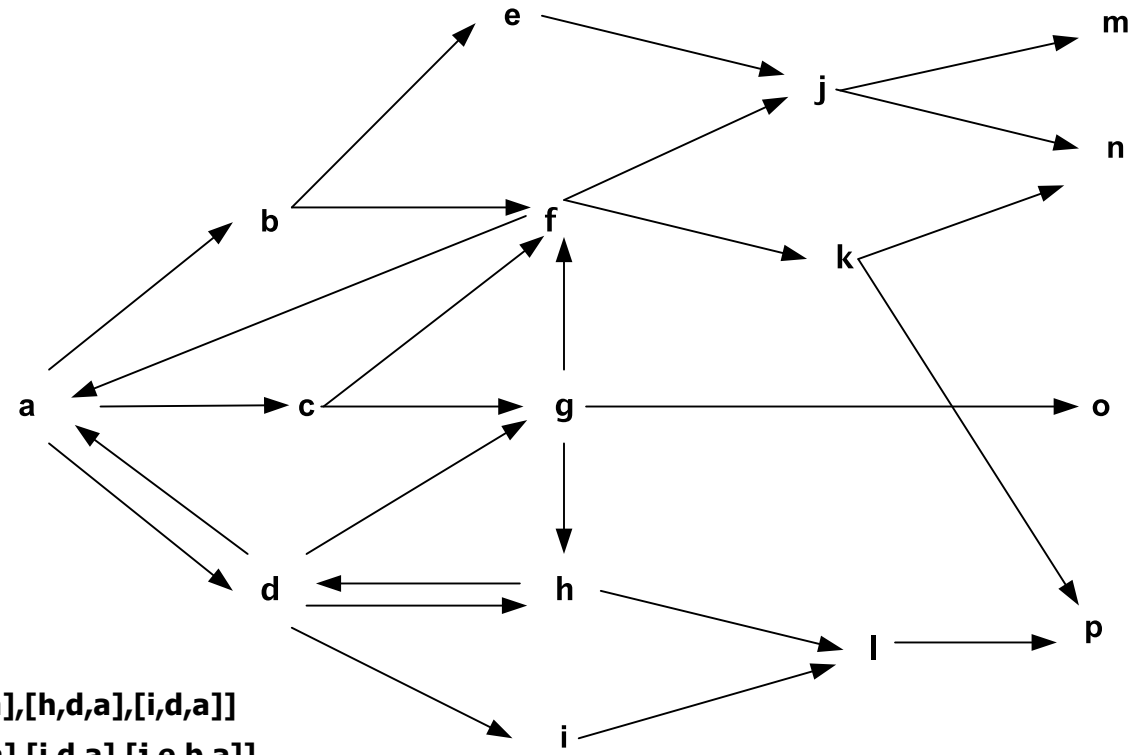
Um exemplo – 1º em Profundidade a->i

?- go(a,i,L).
NPerc[[b,a],[c,a],[d,a]]
NPerc[[e,b,a],[f,b,a],[c,a],[d,a]]
NPerc[[j,e,b,a],[f,b,a],[c,a],[d,a]]
NPerc[[m,j,e,b,a],[n,j,e,b,a],[f,b,a],[c,a],[d,a]]
NPerc[[n,j,e,b,a],[f,b,a],[c,a],[d,a]]
NPerc[[f,b,a],[c,a],[d,a]]
NPerc[[j,f,b,a],[k,f,b,a],[c,a],[d,a]]
NPerc[[m,j,f,b,a],[n,j,f,b,a],[k,f,b,a],[c,a],[d,a]]
NPerc[[n,j,f,b,a],[k,f,b,a],[c,a],[d,a]]
NPerc[[k,f,b,a],[c,a],[d,a]]
NPerc[[n,k,f,b,a],[p,k,f,b,a],[c,a],[d,a]]
NPerc[[p,k,f,b,a],[c,a],[d,a]]
NPerc[[c,a],[d,a]]
NPerc[[f,c,a],[g,c,a],[d,a]]
NPerc[[j,f,c,a],[k,f,c,a],[g,c,a],[d,a]]
NPerc[[m,j,f,c,a],[n,j,f,c,a],[k,f,c,a],[g,c,a],[d,a]]
NPerc[[n,j,f,c,a],[k,f,c,a],[g,c,a],[d,a]]
NPerc[[k,f,c,a],[g,c,a],[d,a]]
NPerc[[n,k,f,c,a],[p,k,f,c,a],[g,c,a],[d,a]]
NPerc[[p,k,f,c,a],[g,c,a],[d,a]]
NPerc[[g,c,a],[d,a]]
NPerc[[f,g,c,a],[o,g,c,a],[h,g,c,a],[d,a]]
NPerc[[j,f,g,c,a],[k,f,g,c,a],[o,g,c,a],[h,g,c,a],[d,a]]
NPerc[[m,j,f,g,c,a],[n,j,f,g,c,a],[k,f,g,c,a],[o,g,c,a],[h,g,c,a],[d,a]]
NPerc[[n,j,f,g,c,a],[k,f,g,c,a],[o,g,c,a],[h,g,c,a],[d,a]]
NPerc[[k,f,g,c,a],[o,g,c,a],[h,g,c,a],[d,a]]
NPerc[[n,k,f,g,c,a],[p,k,f,g,c,a],[o,g,c,a],[h,g,c,a],[d,a]]



NPerc[[p,k,f,g,c,a],[o,g,c,a],[h,g,c,a],[d,a]]
NPerc[[o,g,c,a],[h,g,c,a],[d,a]]
NPerc[[h,g,c,a],[d,a]]
NPerc[[d,h,g,c,a],[l,h,g,c,a],[d,a]]
NPerc[[i,d,h,g,c,a],[l,h,g,c,a],[d,a]]
L = [a,c,g,h,d,i]

Um exemplo – Primeiro em Largura a->i



?- go(a,i,L).

NPerc[[b,a],[c,a],[d,a]]

NPerc[[c,a],[d,a],[e,b,a],[f,b,a]]

NPerc[[d,a],[e,b,a],[f,b,a],[f,c,a],[g,c,a]]

NPerc[[e,b,a],[f,b,a],[f,c,a],[g,c,a],[g,d,a],[h,d,a],[i,d,a]]

NPerc[[f,b,a],[f,c,a],[g,c,a],[g,d,a],[h,d,a],[i,d,a],[j,e,b,a]]

NPerc[[f,c,a],[g,c,a],[g,d,a],[h,d,a],[i,d,a],[j,e,b,a],[j,f,b,a],[k,f,b,a]]

NPerc[[g,c,a],[g,d,a],[h,d,a],[i,d,a],[j,e,b,a],[j,f,b,a],[k,f,b,a],[j,f,c,a],[k,f,c,a]]

NPerc[[g,d,a],[h,d,a],[i,d,a],[j,e,b,a],[j,f,b,a],[k,f,b,a],[j,f,c,a],[k,f,c,a],[f,g,c,a],[o,g,c,a],[h,g,c,a]]

NPerc[[h,d,a],[i,d,a],[j,e,b,a],[j,f,b,a],[k,f,b,a],[j,f,c,a],[k,f,c,a],[f,g,c,a],[o,g,c,a],[h,g,c,a],[f,g,d,a],[o,g,d,a],[h,g,d,a]]

NPerc[[i,d,a],[j,e,b,a],[j,f,b,a],[k,f,b,a],[j,f,c,a],[k,f,c,a],[f,g,c,a],[o,g,c,a],[h,g,c,a],[f,g,d,a],[o,g,d,a],[h,g,d,a],[l,h,d,a]]

L = [a,d,i]

Métodos de Pesquisa

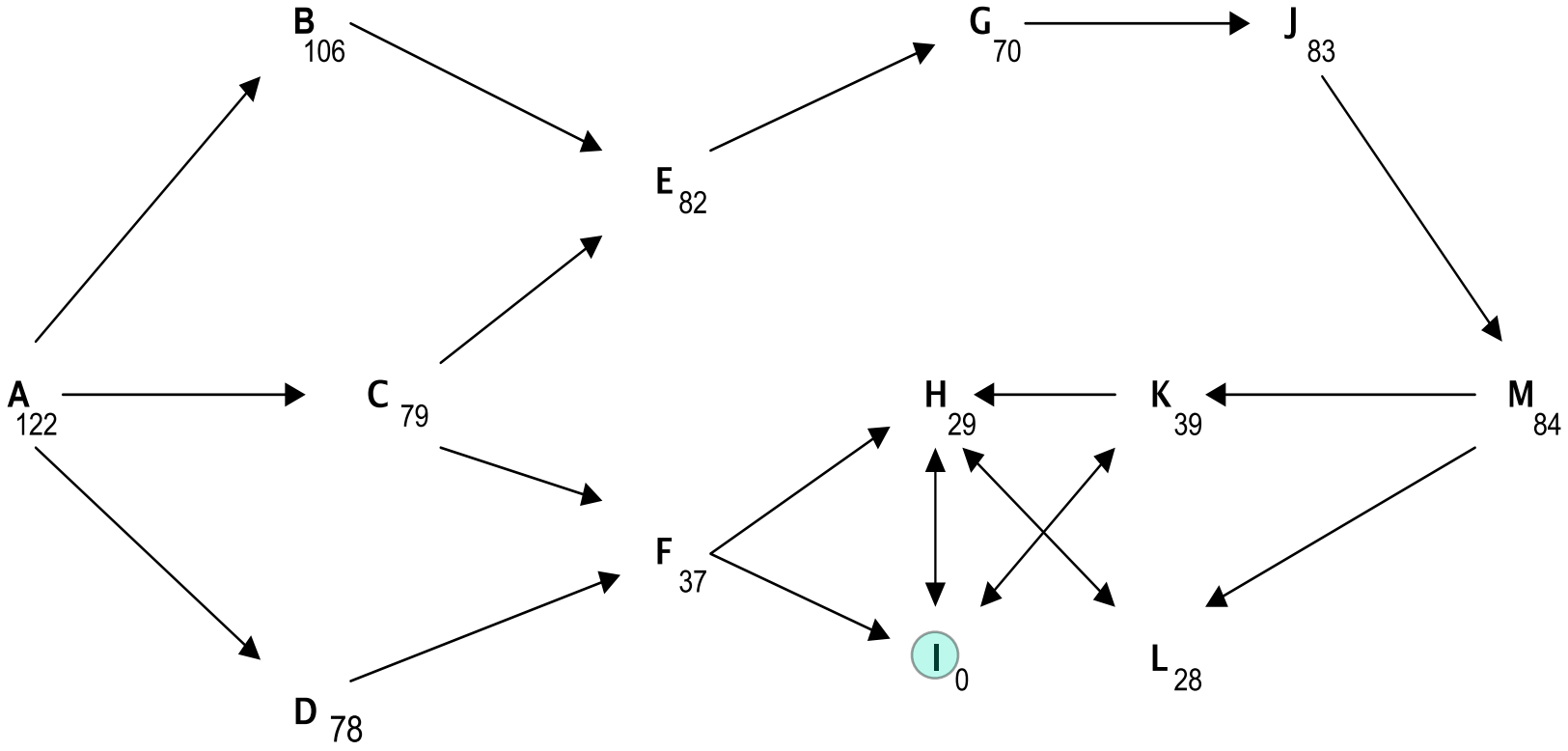
Aula 2

Primeiro o Melhor

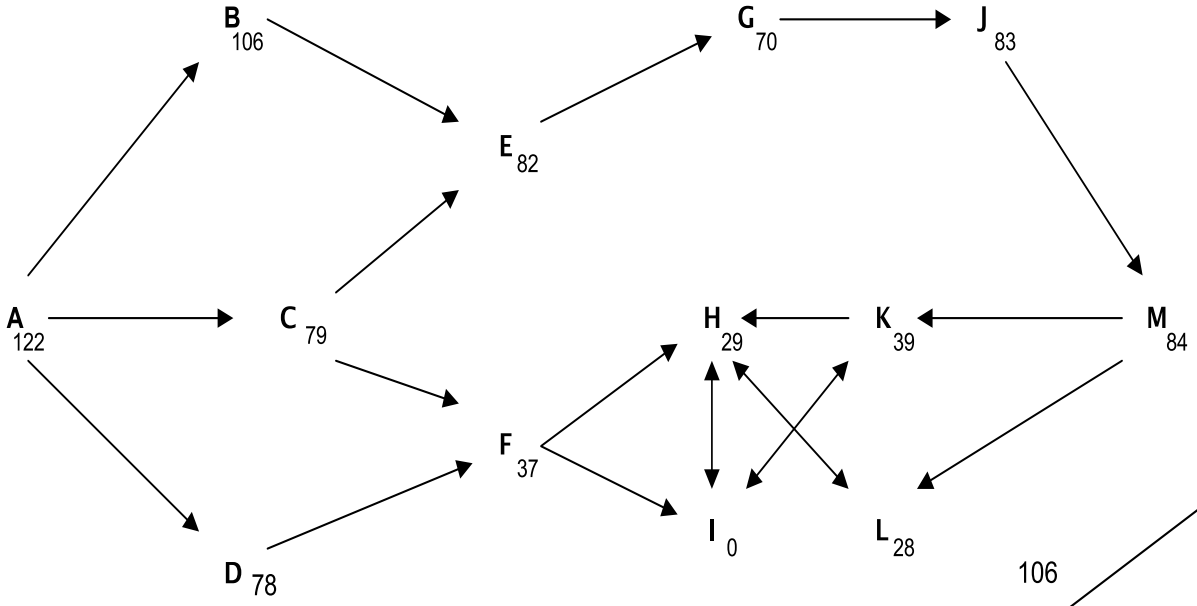
- Este é o primeiro método dito "**informado**" de pesquisa. Assemelha-se ao Primeiro em Profundidade. A diferença reside no facto da decisão sobre qual o ramo a seguir ser feita com base num critério de **decisão local**
- No caso de haver mais que um caminho a seguir a partir de um dado nó, opta-se localmente pelo que pareça mais promissor.
- É necessário dispor de um **valor numérico** que avalie o **custo** ou o **ganho** inerentes a escolher um determinado caminho

Primeiro o Melhor

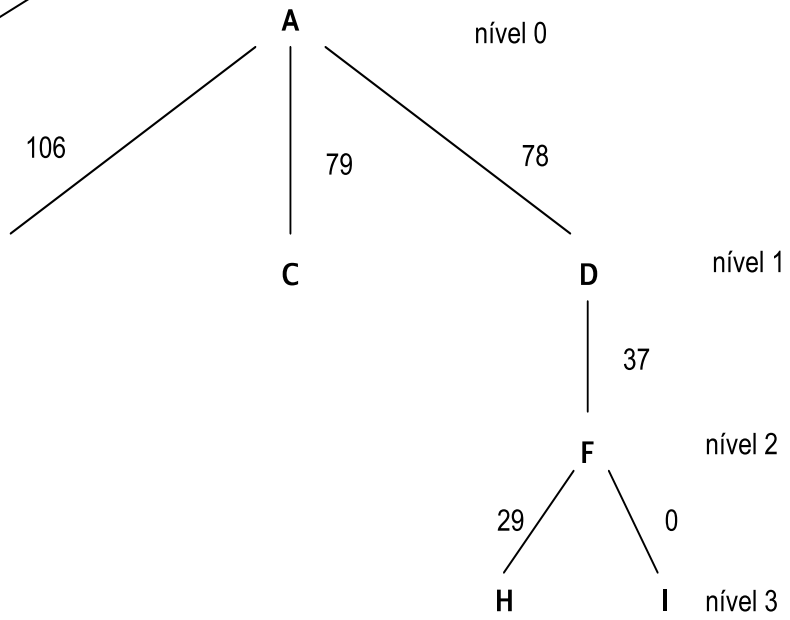
Na figura repete-se o grafo usado anteriormente, mas agora usando **estimativas** do custo para ir de cada uma das cidades até a cidade I (considerada o destino).



Primeiro o Melhor



Árvore gerada:



Nada garante que a melhor solução seja encontrada!

Primeiro em Profundidade

Implementação

Lista auxiliar c/ os nós visitados até ao momento

```
go(Orig,Dest,L) :- go(Orig,Dest,[Orig],L).

go(Dest,Dest,_,[Dest]).
go(Orig,Dest,LV,[Orig|L]) :-
    liga(Orig,X),
    \+ member(X,LV),
    go(X,Dest,[X|LV],L).
```

Para evitar visitar nós já visitados

```
?- go(a, m, L).
L=[a,b,e,j,m]
```

Primeiro o Melhor

```
estimativa(a,122).
estimativa(b,75).
estimativa(c,79).
estimativa(d,78).
estimativa(e,82).
estimativa(f,37).
estimativa(g,70).
estimativa(h,29).
estimativa(i,0).
estimativa(j,83).
estimativa(m,84).
estimativa(k,39).
estimativa(l,28).
```

```
go(Orig,Dest,L):-go(Orig,Dest,[Orig],L).

go(Dest,Dest,_,[Dest]).
go(Orig,Dest,LA,[Orig|L]):-
    findall((X,EstX),(liga(Orig,X),estimativa(X,EstX)),LX),
    melhor(LX,MX,_),
    \+ member(MX,LA),
    go(MX,Dest,[MX|LA],L).

melhor([(_,EstX)|T],M,EstM):-
    melhor(T,M,EstM),
    EstM =< EstX, !.
melhor([(X,EstX)|_],X,EstX).
```

```
[ (b, 75), (c, 79), (d, 78)]
```

Traçagem de melhor/3

```
Call: (9) melhor([ (b, 75), (c, 79), (d, 78)], _G2060, _G2061)
Call: (10) melhor([ (c, 79), (d, 78)], _G2060, _G2061)
Call: (11) melhor([ (d, 78)], _G2060, _G2061)
Call: (12) melhor([], _G2060, _G2061)
Fail: (12) melhor([], _G2060, _G2061)
Redo: (11) melhor([ (d, 78)], _G2060, _G2061)
Exit: (11) melhor([ (d, 78)], d, 78)
Call: (11) 78=<79
Exit: (11) 78=<79
Exit: (10) melhor([ (c, 79), (d, 78)], d, 78)
Call: (10) 78=<75
Fail: (10) 78=<75
Redo: (9) melhor([ (b, 75), (c, 79), (d, 78)], _G2060, _G2061)
Exit: (9) melhor([ (b, 75), (c, 79), (d, 78)], b, 75)
```

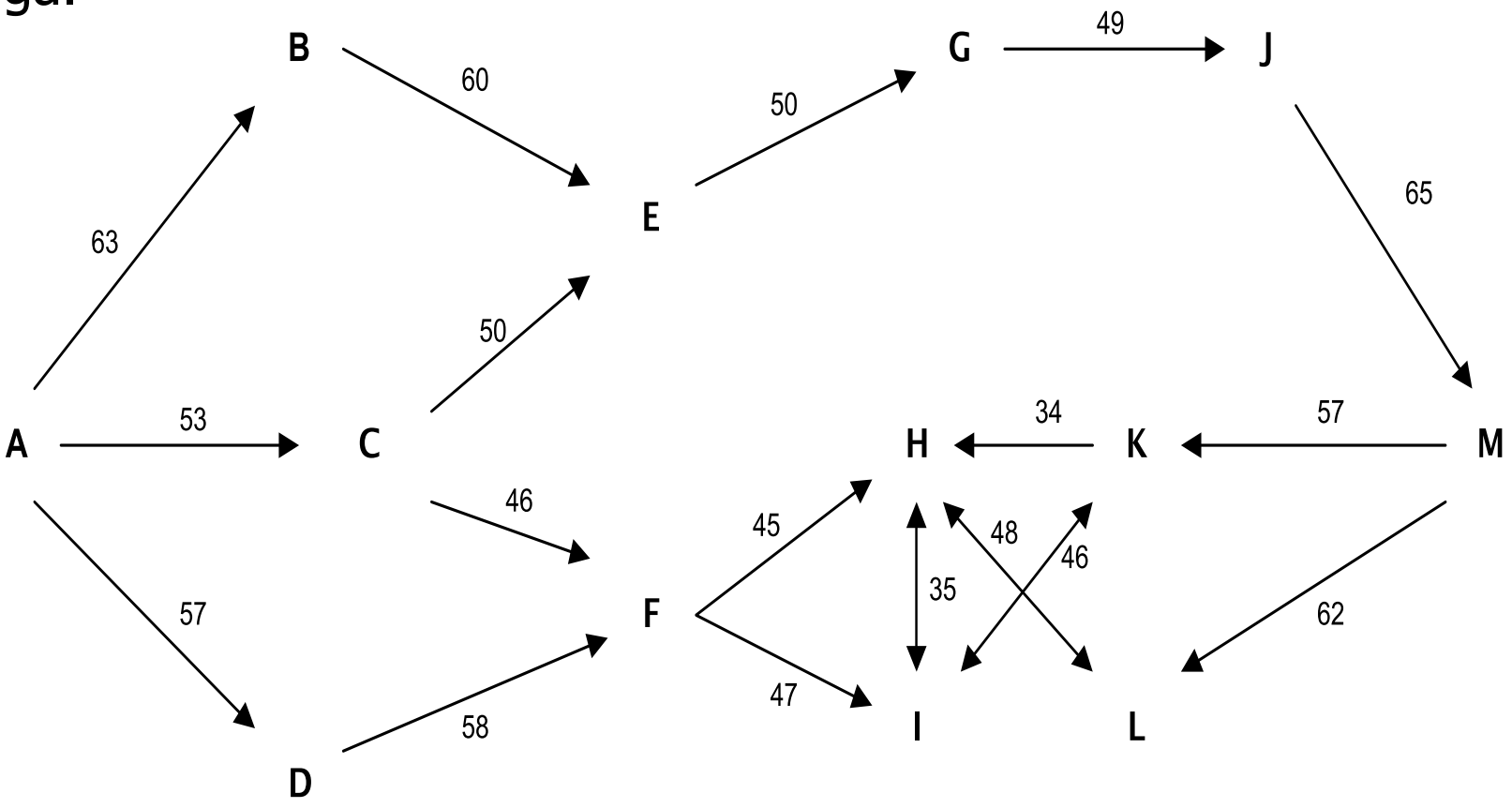
```
melhor([(_,EstX)|T],M,EstM):-
    melhor(T,M,EstM),
    EstM =< EstX, !.
melhor([(X,EstX)|_],X,EstX).
```

Branch and Bound

- O **Branch and Bound** é um método de pesquisa que corresponde ao método "Primeiro o Melhor" com avaliação de transições locais mas com possibilidade de alterar a qualquer momento o nó sobre qual se vai considerar a próxima expansão
- Assim, o próximo nó a expandir não é obrigatoriamente um descendente do último nó expandido
- A comparação entre os nós candidatos à expansão é feita com base no **valor acumulado** do custo ou do ganho desde a raiz até esses nós

Branch and Bound

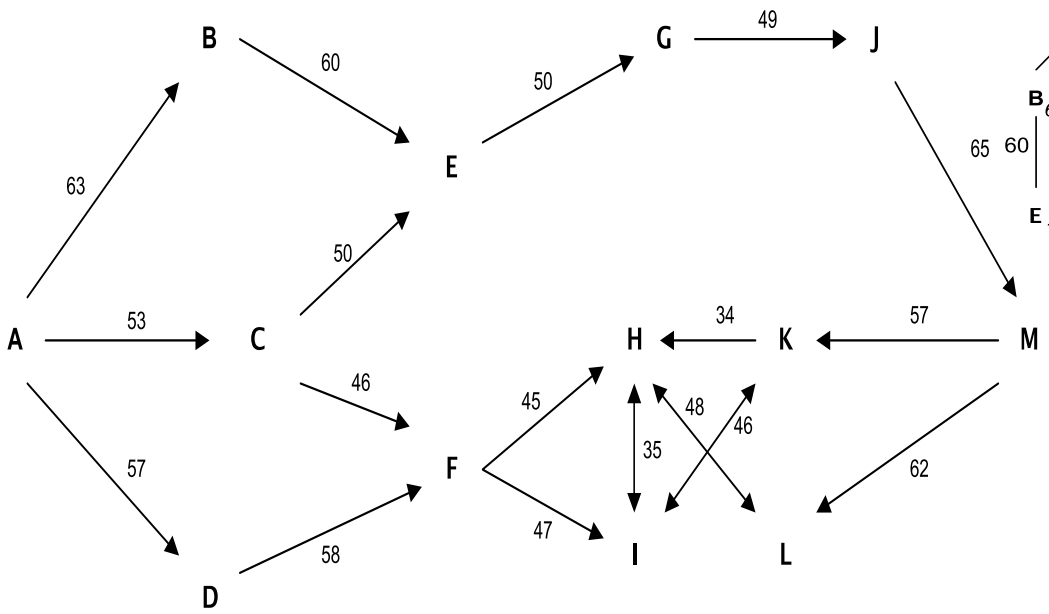
Exemplo de um grafo em que junto cada arco é etiquetado com a **distância** a ser percorrida entre os 2 nós que o arco liga.

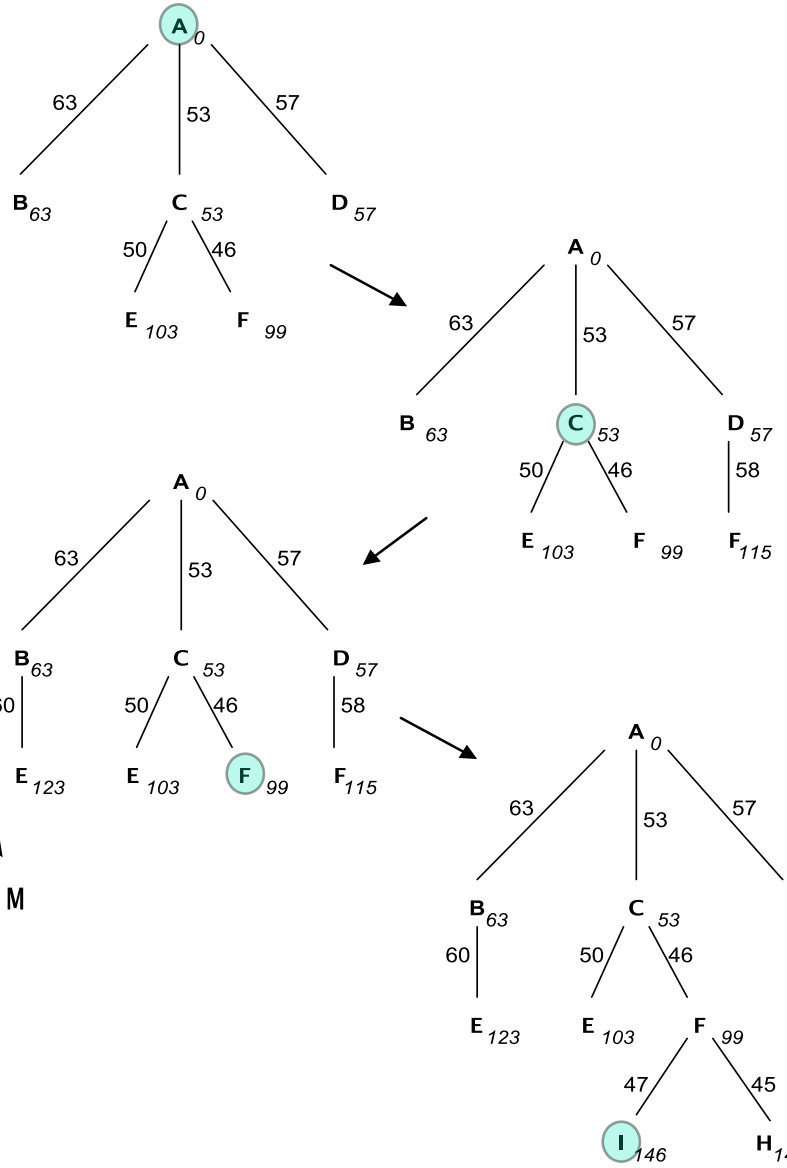


Branch and Bound

Expansão da árvore de pesquisa usando o método Branch and Bound:

- Junto aos ramos estão os custos locais de transição
- junto aos nós estão os custos acumulados desde a raíz até ao nó, resultando do somatório dos custos dos ramos que vão da raiz até ao nó





Análise

- O principal inconveniente do Branch and Bound reside no facto de não ser sensível à distância que um dado nó se encontra da solução
- Será que se pode garantir que a primeira solução que se encontra usando o Branch and Bound é sempre a melhor solução?
- O que acontece se a solução estiver muito afastada da raiz, ou seja, estiver a um nível muito elevado?

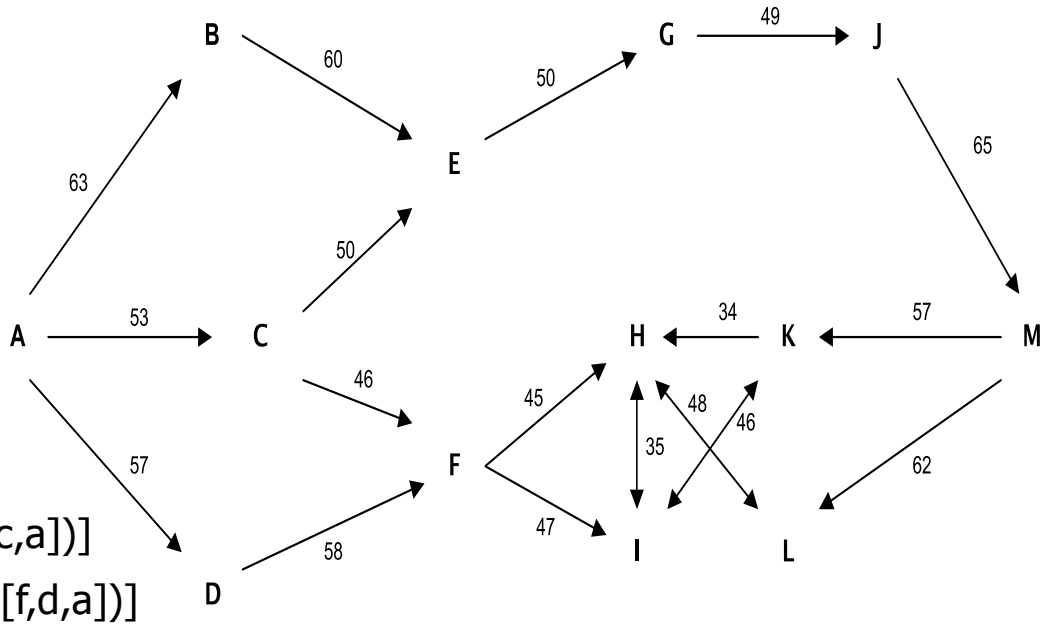
Branch and Bound

```
go(Orig,Dest,Perc):-
    go1([(0,[Orig])],Dest,P),
    reverse(P,Perc).

go1([(_,Prim)|_],Dest,Prim):- Prim=[Dest|_].
go1([(_,[Dest|_])|Resto],Dest,Perc):- !, go1(Resto,Dest,Perc).
go1([(C,[Ult|T])|Outros],Dest,Perc):-
        findall((NC,[Z,Ult|T]),
                (proximo_no(Ult,T,Z,C1),NC is C+C1),Lista),
        append(Outros,Lista,NPerc),
        sort(NPerc,NPerc1),
        write(NPerc1),nl,
        go1(NPerc1,Dest,Perc).

proximo_no(X,T,Z,C):- liga(X,Z,C), not member(Z,T).
```

```
liga(a,b,63).
liga(a,c,53).
liga(a,d,57).
...
```

Perc - [(53,[c,a]),(57,[d,a]),(63,[b,a])]

Branch and Bound



?- go(a,i,L).

[(53,[c,a]),(57,[d,a]),(63,[b,a])]

[(57,[d,a]),(63,[b,a]),(99,[f,c,a]),(103,[e,c,a])]

[(63,[b,a]),(99,[f,c,a]),(103,[e,c,a]),(115,[f,d,a])]

[(99,[f,c,a]),(103,[e,c,a]),(115,[f,d,a]),(123,[e,b,a])]

[(103,[e,c,a]),(115,[f,d,a]),(123,[e,b,a]),(144,[h,f,c,a]),(146,[i,f,c,a])]

[(115,[f,d,a]),(123,[e,b,a]),(144,[h,f,c,a]),(146,[i,f,c,a]),(153,[g,e,c,a])]

[(123,[e,b,a]),(144,[h,f,c,a]),(146,[i,f,c,a]),(153,[g,e,c,a]),(160,[h,f,d,a]),(162,[i,f,d,a])]

[(144,[h,f,c,a]),(146,[i,f,c,a]),(153,[g,e,c,a]),(160,[h,f,d,a]),(162,[i,f,d,a]),(173,[g,e,b,a])]

[(146,[i,f,c,a]),(153,[g,e,c,a]),(160,[h,f,d,a]),(162,[i,f,d,a]),(173,[g,e,b,a]),(179,[i,h,f,c,a]),
(192,[l,h,f,c,a])]

L = [a,c,f,i]

Métodos de Pesquisa

Aula 3

A*

- **A*** - Junção num único método do que há de bom no Primeiro o Melhor (o uso de funções que estimam a distância à solução), com o que há de melhor no Branch and Bound (o uso de custos acumulados conhecidos e a possibilidade de saltar de um ponto para outro na árvore de pesquisa sem que o novo ponto seja um descendente do primeiro).
- É utilizada função heurística **f'** tal que:

  f' = g + h'
  - **g** é o custo conhecido para ir do estado inicial até ao estado do nó que se está a considerar (no momento, uma folha ainda não expandida)
  - **h'** é uma estimativa do custo para ir desse nó até a solução, estando portanto sujeita a erro

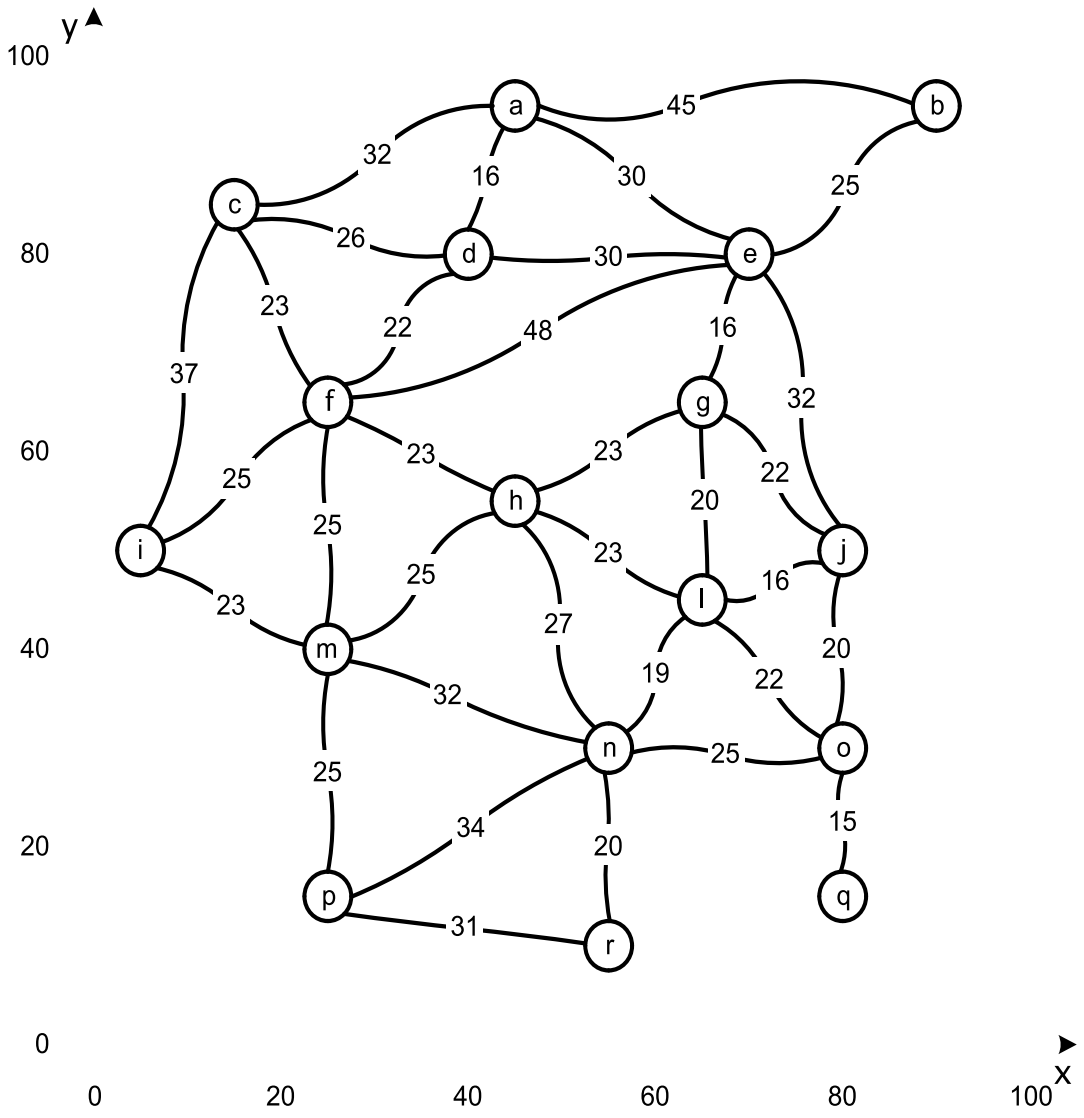
A*

O que deverá ser **h'** relativamente ao valor real?

- Seja h' uma estimativa da distância de um nó à solução no método de pesquisa A*:
  - se estivermos a operar com custos convém que h' seja um **minorante** do custo real
  - ao passo que se estivermos a operar com lucros interessa que h' seja um **majorante**
- O que acontece se o minorante for 0?

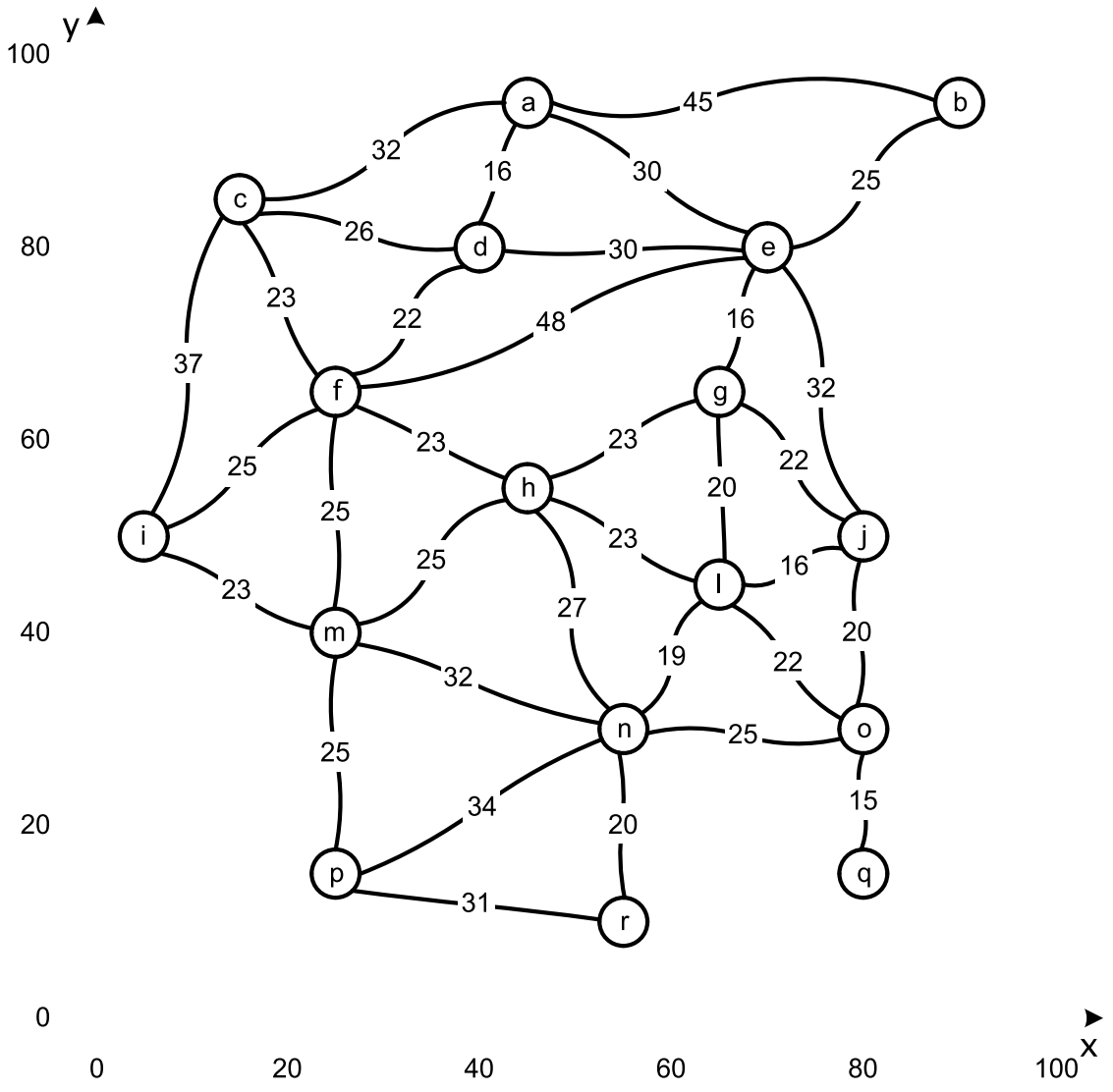
Um exemplo

Grafo representando cidades cotadas nos eixos **x** e **y** e distâncias entre cidades



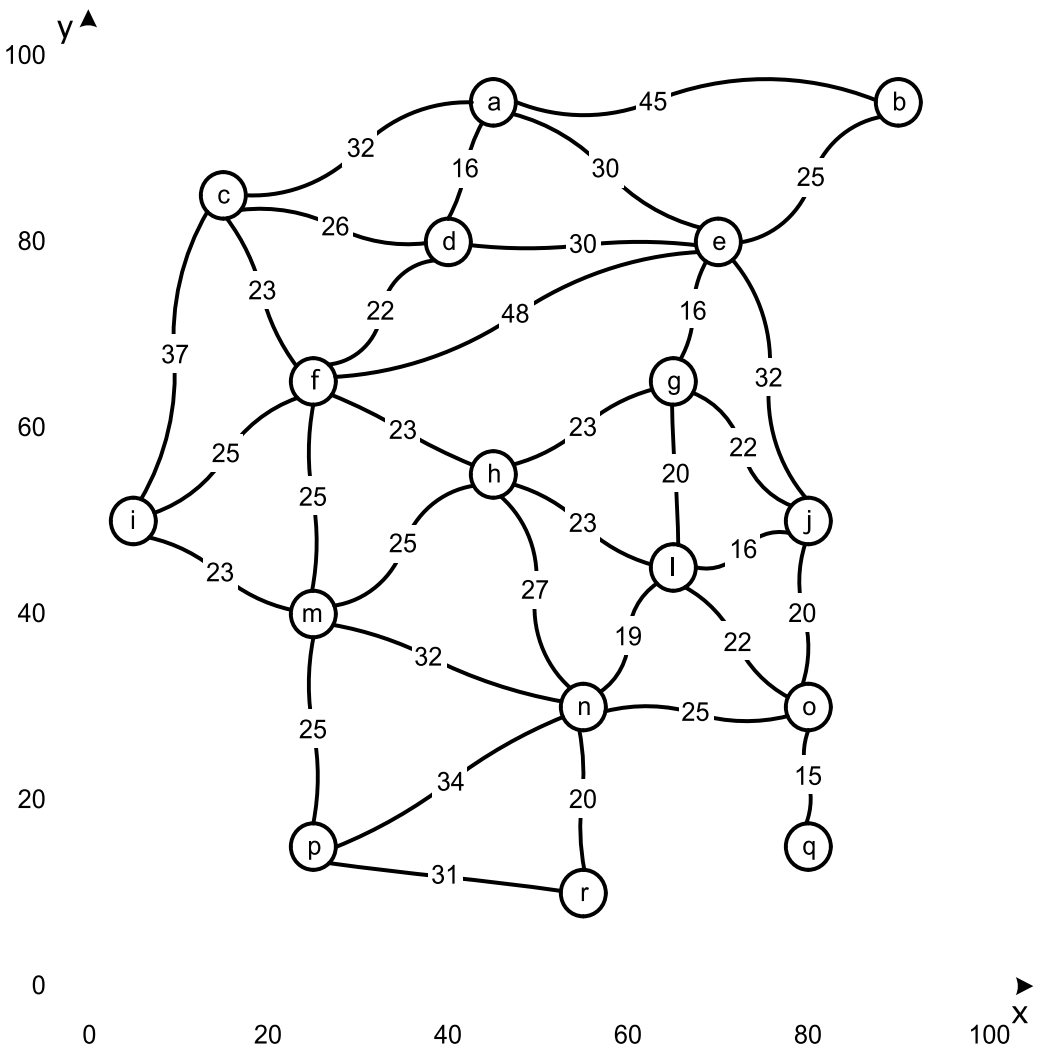
Um exemplo

```
% cidade(Cidade, PosX, PosY).
cidade(a,45,95).
cidade(b,90,95).
cidade(c,15,85).
cidade(d,40,80).
cidade(e,70,80).
cidade(f,25,65).
cidade(g,65,65).
cidade(h,45,55).
cidade(i,5,50).
cidade(j,80,50).
cidade(l,65,45).
cidade(m,25,40).
cidade(n,55,30).
cidade(o,80,30).
cidade(p,25,15).
cidade(q,80,15).
cidade(r,55,10).
```



Um exemplo

```
% estradah(Cidade1,Cidade2, Distância).
estradah(a,b,45).
estradah(a,c,32).
estradah(a,d,16).
estradah(a,e,30).
estradah(b,e,25).
estradah(d,e,30).
estradah(c,d,26).
estradah(c,f,23).
estradah(c,i,37).
estradah(d,f,22).
estradah(f,h,23).
estradah(f,m,25).
estradah(f,i,25).
estradah(i,m,23).
estradah(e,f,48).
estradah(e,g,16).
estradah(e,j,32).
estradah(g,h,23).
estradah(g,l,20).
estradah(g,j,22).
estradah(h,m,25).
estradah(h,n,27).
estradah(h,l,23).
estradah(j,l,16).
estradah(j,o,20).
estradah(l,n,19).
estradah(l,o,22).
estradah(m,n,32).
estradah(m,p,25).
estradah(n,p,34).
estradah(n,r,20).
estradah(o,n,25).
estradah(o,q,15).
estradah(p,r,31).
```



Implementação

```
hbf(Orig,Dest,Perc,Total):-
                estimativa(Orig,Dest,H),
                hbf1([c(H/0,[Orig])],Dest,P,Total), reverse(P,Perc).

hbf1(Percursos,Dest,Percurso,Total):-
                menor_percursoh(Percursos,Menor,Restantes),
                perc_seguintesh(Menor,Dest,Restantes,Percurso,Total).

perc_seguintesh(c(_/Dist,Percurso),Dest,_,Percurso,Dist):-
      Percurso=[Dest|_].
perc_seguintesh(c(_,[Dest|_]),Dest,Restantes,Percurso,Total):-
      !, hbf1(Restantes,Dest,Percurso,Total).
perc_seguintesh(c(_/Dist,[Ult|T]),Dest,Percursos,Percurso,Total):-
    findall(c(H1/D1,[Z,Ult|T]), proxi_noh(Ult,T,Z,Dist,Dest,H1/D1),Lista),
    append(Lista,Percursos,NovosPercursos),
    hbf1(NovosPercursos,Dest,Percurso,Total).
```

c(91.9238815542512 / 45, [b,a])

regra da condição limite

regra p/ produzir alternativas

Implementação

Predicados auxiliares

```
proxi_noh(X,T,Y,Dist,Dest,H/Dist1) :-
                    (estradah(X,Y,Z);estradah(Y,X,Z)),
                     \+ member(Y,T),
                     Dist1 is Dist + Z,
                     estimativa(Y,Dest,H).

estimativa(C1,C2,Est):-
          cidade(C1,X1,Y1),
          cidade(C2,X2,Y2),
          DX is X1-X2,
          DY is Y1-Y2,
          Est is sqrt(DX*DX+DY*DY).
% 'Est is 0' para desprezar a heurística.
```

Implementação

Predicados auxiliares

```
menor_percursoh([Perc|Percursos],Menor,[Perc|Resto]) :-
        menor_percursoh(Percursos,Menor,Resto),
         menorh(Menor,Perc),!.
menor_percursoh([Perc|Resto],Perc,Resto).

menorh(c(H1/D1,_),c(H2/D2,_)) :-
        C1 is H1+D1, C2 is H2+D2, C1<C2.
```

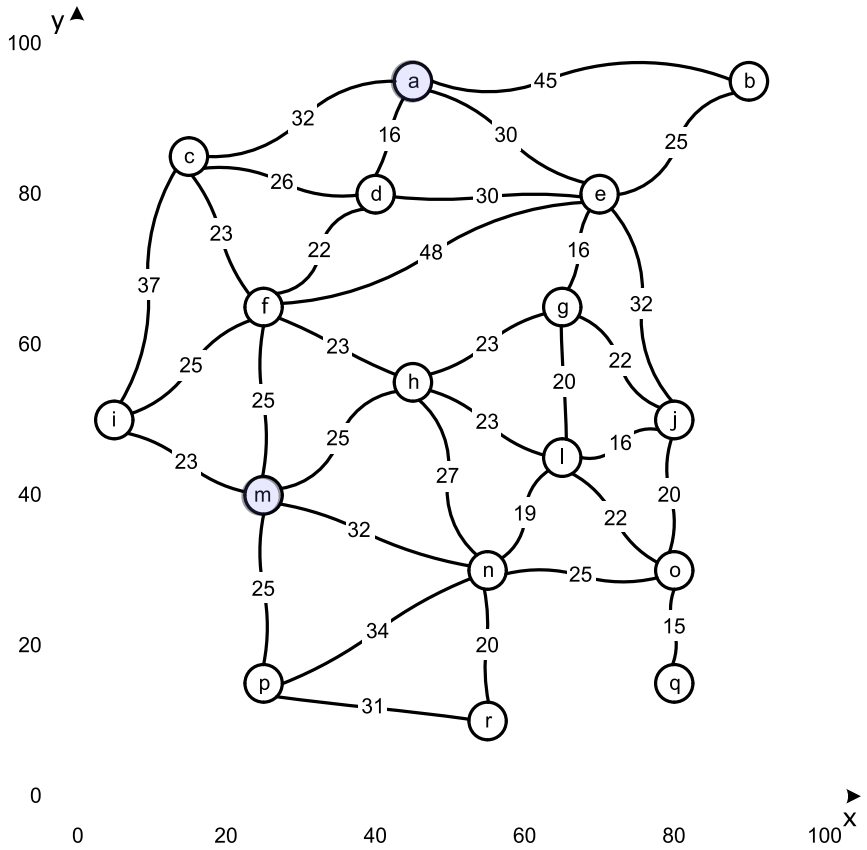
Um exemplo

?- hbf(a,m,P,T).

Nó a expandir (menor h')

C1 is H1+D1, C2 is H2+D2, C1<C2

Nós resultantes da expansão de nó



[c(85 / 45,[**b**,a]),c(46 / 32,[**c**,a]),c(42 / 16,[**d**,a]),c(60 / 30,[**e**,a])]
[c(60 / 46,[**e**,d,a]),c(25 / 38,[**f**,d,a]),c(46 / 42,[**c**,d,a]),c(85 / 45,[**b**,a]),c(46 / 32,[**c**,a]),c(60 / 30,[**e**,a])]
[c(25 / 61,[**h**,f,d,a]),c(0 / 63,[**m**,f,d,a]),c(22 / 63,[**i**,f,d,a]),c(46 / 61,[**c**,f,d,a]),c(60 / 86,[**e**,f,d,a]),c(60 / 46,[**e**,d,a]),
c(46 / 42,[**c**,d,a]),c(85 / 45,[**b**,a]),c(46 / 32,[**c**,a]),c(60 / 30,[**e**,a])]

P = [a,d,f,m] ,
T = 63

Um exemplo

?- hbf(a,r,Percurso,Distância_Total).
Percurso = [a,e,g,l,n,r] ,
Distância_Total = 105 ;

Percurso = [a,d,f,h,n,r] ,
Distância_Total = 108 ;

Percurso = [a,d,f,m,n,r] ,
Distância_Total = 115 ;

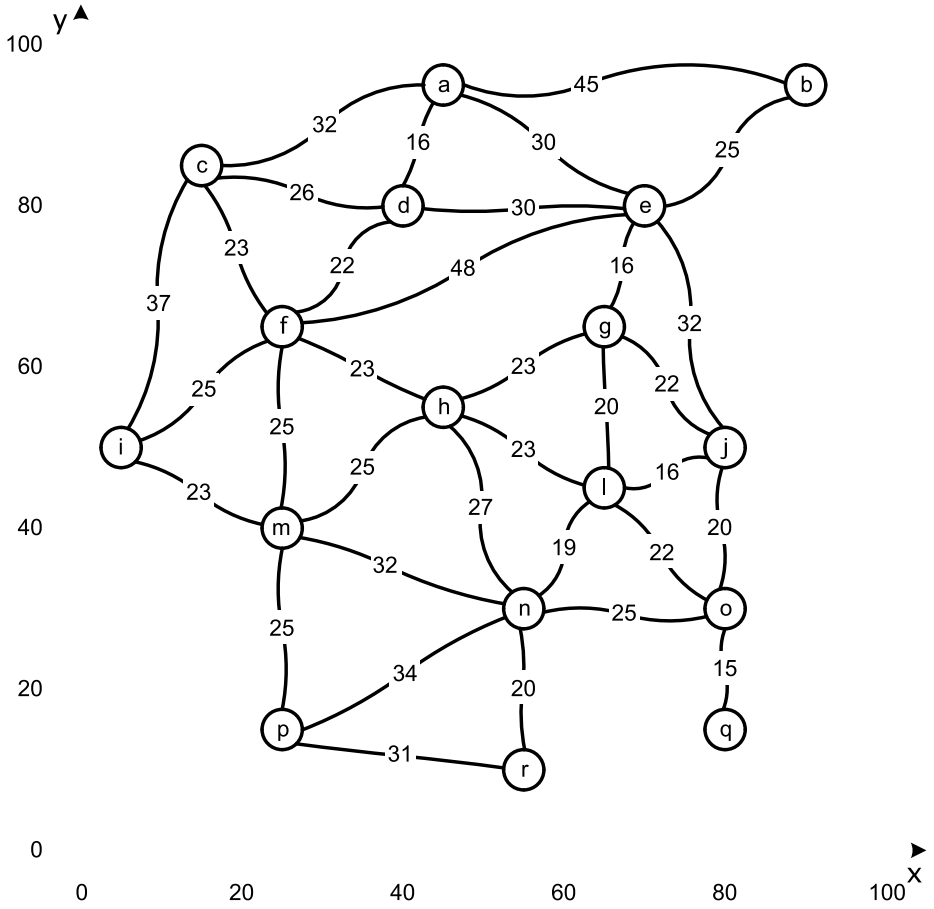
Percurso = [a,e,g,h,n,r] ,
Distância_Total = 116 ;

Percurso = [a,e,j,l,n,r] ,
Distância_Total = 117 ;

Percurso = [a,d,f,m,p,r] ,
Distância_Total = 119 ;

Percurso = [a,d,e,g,l,n,r] ,
Distância_Total = 121 ;

Percurso = [a,d,f,h,l,n,r] ,
Distância_Total = 123 ;
.....

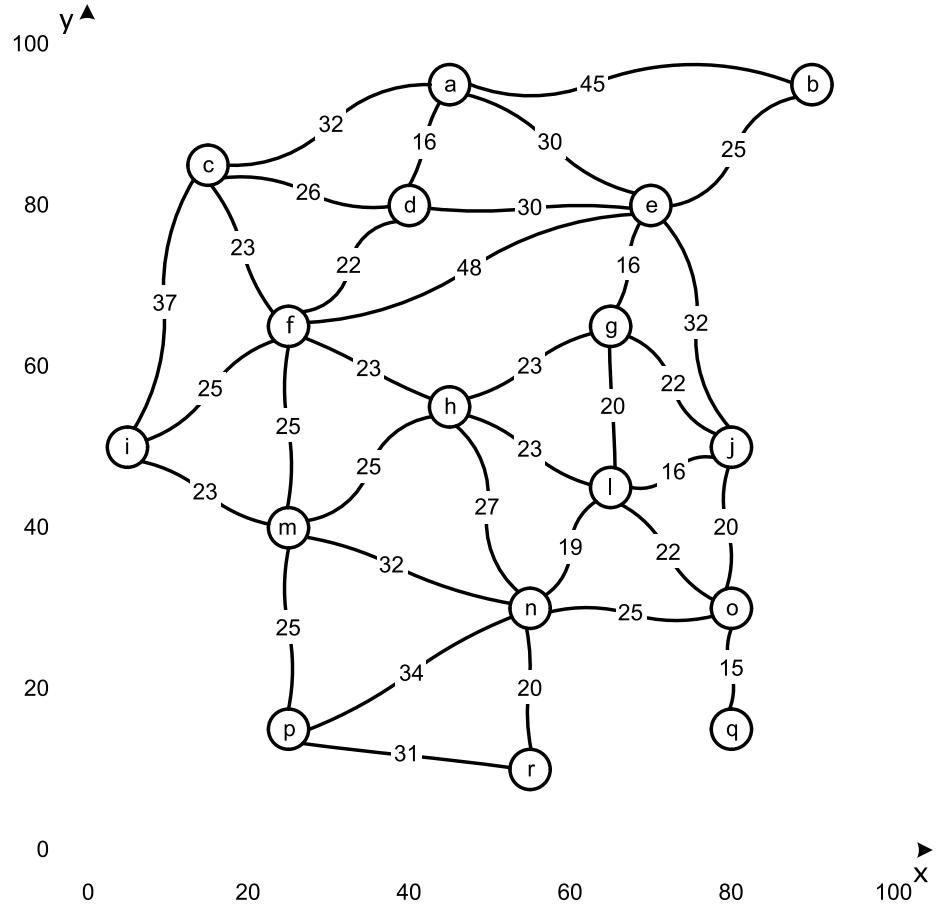


Note-se que como as estradas são bidireccionais há um enorme número de soluções

Um exemplo

?- findall(par(P,D),hbf(a,r,P,D),L).
Error 4, Heap Space Full, Trying menor_percursoh/3

Aborted



Um exemplo

Vamos agora ver o tempo que demora a primeira solução usando o A* e alterando hbf:

```
:- use_module(library(statistics)). % time/1

hbf(Orig,Dest,Perc,Total):-
     estimativa(Orig,Dest,H),
     time(
            ( hbf1([c(H/0,[Orig])],Dest,P,Total), reverse(P,Perc))
          ).

?- hbf(a,r,Percurso,Distância_Total).
% 7,820 inferences, 0,016 CPU in 0,017 seconds (17 ms)
Percurso = [a,e,g,l,n,r] ,
Distância_Total = 105
```

Um exemplo

Para comparar com o Branch and Bound basta fazer a estimativa igual a zero:

```
/*estimativa(C1,C2,Est):-
        cidade(C1,X1,Y1),
        cidade(C2,X2,Y2),
        DX is X1-X2,
        DY is Y1-Y2,
        Est is sqrt(DX*DX+DY*DY).*/
estimativa(_,_,0).
```

```
| ?- hbf(a,r,Percurso,Distância_Total).
% 77,224 inferences, 0,038 CPU in 0,039 seconds (39 ms)
Percurso = [a,e,g,l,n,r] ,
Distância_Total = 105
```

A solução encontrada foi a mesma, mas demorou mais tempo

Algoritmos Genéticos

Aula 4

Algoritmos Genéticos

Enquanto que os métodos de pesquisa estudados anteriormente são métodos **generativos** que percorrem o espaço de pesquisa em busca de uma solução para o problema,

os **Algoritmos Genéticos** trabalham com **populações** de soluções que são combinadas para obter novas soluções.

Analogia entre a evolução natural e os algoritmos genéticos

Analogia com a Natureza		
Evolução Natural	⇔	**Algoritmos Genéticos**
Indivíduo	—	Solução
População	—	Conjunto de Soluções
Genótipo (cromossomas)	—	Representação da Solução
Reprodução Sexual	—	Operador de Recombinação (p.ex. cruzamento)
Mutação	—	Operador Mutação
Gerações	—	Ciclos

Algoritmos Genéticos

Gerar população inicial de indivíduos

Enquanto o critério de paragem não for atendido

- *Avaliar a aptidão de todos os indivíduos*
- *Seleccionar os indivíduos mais aptos para reprodução*
- *Gerar novos indivíduos através do cruzamento e avaliar a respectiva aptidão*
- *Gerar nova população através da inserção de alguns bons indivíduos e eliminação de maus indivíduos*
- *Aplicar a mutação a alguns indivíduos*

Fim

Esqueleto típico de um algoritmo genético

Gerar P_0 ← População inicial

$t \leftarrow 0$ ← iteração

Avaliar P_t

enquanto ! Condição de final (P_t) **fazer**

População de soluções potenciais na iteração **t**

Gerar novos indivíduos

$P'_t \leftarrow$ Seleccionar P_t — Seleccionar os mais aptos

$P''_t \leftarrow$ Aplicar operadores de re-combinação P'_t

$P'''_t \leftarrow$ Aplicar operadores de mutação P''_t

Avaliar P'''_t — Avaliar a sua aptidão

$P_{t+1} \leftarrow$ Seleccionar Sobreviventes de P_t e de P'''_t

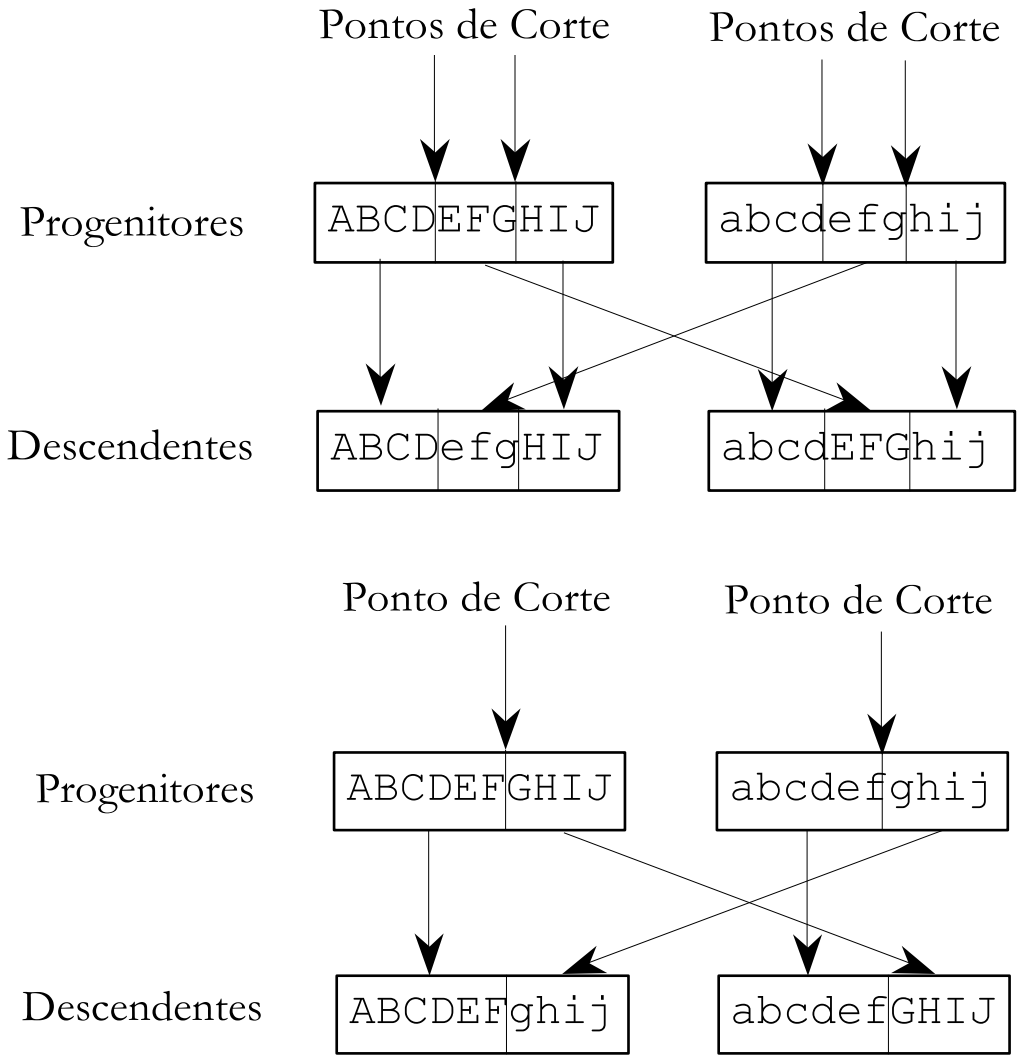
$t \leftarrow t + 1$ ← nova iteração

Fim

retornar Melhor Solução Global

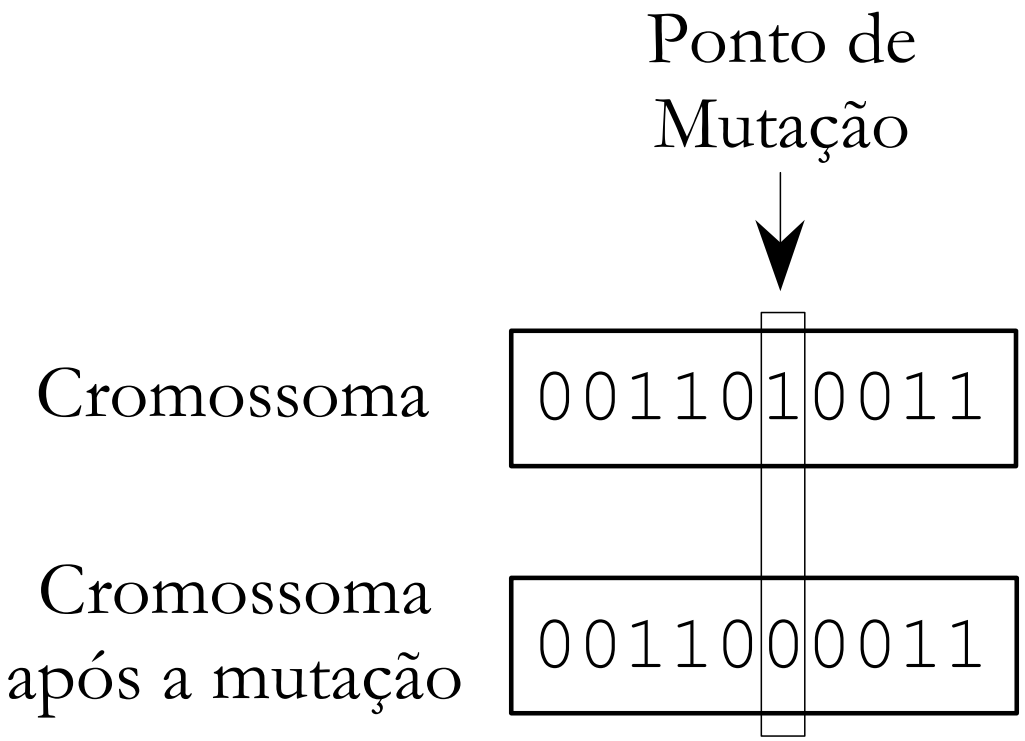
Operações de Cruzamento

Dois Exemplos do operador de cruzamento



Operador de Mutação

Exemplo de Mutação



Problema - Exemplo

- Considere o problema de sequenciamento de 5 tarefas numa única máquina.
- Para cada tarefa **j** (j=1, …, 5), seja $\mathbf{p_j}$ o tempo de processamento, $\mathbf{d_j}$ a data de entrega e $\mathbf{w_j}$ a penalização (por unidade de tempo) no caso da tarefa j se atrasar.
- O objectivo é minimizar a soma pesada dos atrasos $\Sigma \mathbf{w_j T_j}$. (T - atrazos)

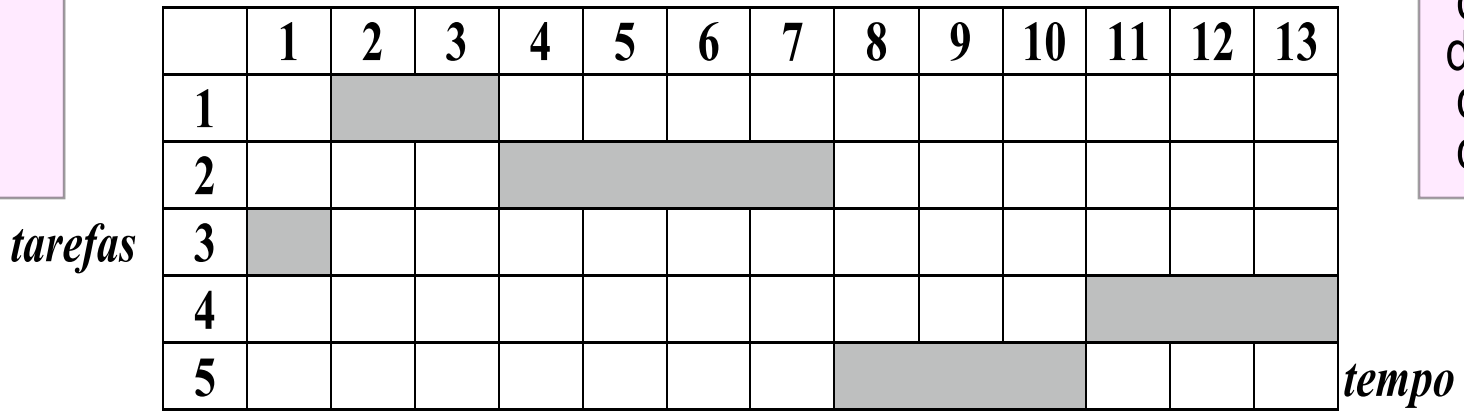
Problema Exemplo (cont)

Para visualizar uma sequência é muitas vezes utilizado um diagrama de Gantt, onde as linhas estão associadas às tarefas e as colunas aos períodos de tempo.

Para a sequência de tarefas [3,1,2,5,4] obtém-se o calendário representado a seguir.

Duração
p1=2
p2=4
p3=1
p4=3
p5=3

Datas
d1=5
d2=7
d3=11
d4=9
d5=8



Problema - Exemplo (cont)

Método de Recombinação usado

1. Supondo o seguinte par de indivíduos, são seleccionados 2 pontos de cruzamento (4º e 7º).

2. Os genes situados entre os dois pontos de cruzamento, inclusive, são copiados para os seus descendentes, sendo as restantes posições preenchidas por um caracter H.

A=123**4567**89 A'=HHH**4567**HH

B=452**1876**93 B'=HHH**1876**HH

Problema - Exemplo (cont)

Método de Recombinação usado (cont.)

3. De seguida, e começando no segundo ponto de cruzamento do pai B define-se uma nova ordem para os genes:

[9 3 4 5 2 1 8 7 6]

B era 452**1876**93

4. Depois de remover os genes 4, 5, 6 e 7 já definidos no filho A', ficamos com os genes [9 3 2 1 8]. As posições em A' contendo H serão preenchidas por essa sequência começando pelo segundo ponto de cruzamento:

A'=218**4567**93

A' era HHH**4567**HH

Problema - Exemplo (cont)

Método de Recombinação usado (cont.)

5. Da mesma forma para gerar o segundo descendente B', define-se uma nova sequência a partir de A:

[8 9 1 2 3 4 5 6 7].

A era 123**4567**89

6. Eliminando os genes já definidos em B', obtém-se a sequência: [9 2 3 4 5]. A seguir substitui-se nas lacunas (H) de B' a sequência obtida, começando no 2º ponto de cruzamento obtendo-se o descendente B'.

B'=345**1876**92

Problema - Exemplo (cont)

```
:- use_module(library(random)).
:- dynamic tarefas/1.

% tarefa(Id,TempoProcessamento,
          DataEntrega,Penalizacao).

tarefa(t1,2,5,1).
tarefa(t2,4,7,6).
tarefa(t3,1,11,2).
tarefa(t4,3,9,3).
tarefa(t5,3,8,2).

% parametrização
geracoes(3).
populacao(4).
prob_cruzamento(0.3).
prob_mutacao(0.01).
```

Critério de paragem

Problema - Exemplo (cont)

```
/** Algoritmo genetico **/

gera :-
   numero_tarefas(N),
   retractall(tarefas(_)),
   assert(tarefas(N)),
   gera_populacao(Pop),
   avalia_populacao(Pop,PopAv),
   ordena_populacao(PopAv,PopOrd),
   geracoes(NG),
   gera_geracao(NG,PopOrd).
```

← Recursivo

Problema - Exemplo (cont)

```
numero_tarefas(NumT) :-
   findall(Tarefa,tarefa(Tarefa,_,_,_),LT),
   length(LT,NumT).
```

Problema - Exemplo (cont)

gera_populacao

```
gera_populacao(Pop) :-
    populacao(TamPop),
    findall(Tarefa,tarefa(Tarefa,_,_,_),ListaTarefas),
    gera_populacao(TamPop,ListaTarefas,Pop).

gera_populacao(0,_,[ ]) :- !.

gera_populacao(TamPop,ListaTarefas,[Ind|Resto]) :-
    TamPop1 is TamPop-1,
    gera_populacao(TamPop1,ListaTarefas,Resto),
    gera_individuo(ListaTarefas,Ind), \+ member(Ind,Resto).
```

gera_individuo

```
gera_individuo(ListaTarefas,Ind) :-
    random_permutation(ListaTarefas,Ind).
```

Pop - [[t1,t5,t2,t3,t4],[t5,t3,t4,t1,t2],[t3,t2,t1,t4,t5],[t5,t1,t4,t3,t2]]

Problema - Exemplo (cont)

gera_geracao

```
gera_geracao(0,Pop) :- !,
    write('Geração '), write(0), write(':'), nl, write(Pop), nl.

gera_geracao(G,Pop) :-
    write('Geração '), write(G), write(':'), nl, write(Pop), nl,
    cruzamento(Pop,NPop1),
    mutacao(NPop1,NPop),
    avalia_populacao(NPop,NPopAv),
    ordena_populacao(NPopAv,NPopOrd),
    G1 is G-1,
    gera_geracao(G1,NPopOrd).
```

Problema Exemplo (cont)

Operador de Crossover

```
cruzamento([],[]).
cruzamento([Ind*_],[Ind]).
cruzamento([Ind1*_,Ind2*_|Resto],[NInd1,NInd2|Resto1]) :-
      gerar_pontos_cruzamento(P1,P2),
      prob_cruzamento(Pcruz), Pc is rand(1),
      ((Pc =< Pcruz, !, cruzar(Ind1,Ind2,P1,P2,NInd1),
      cruzar(Ind2,Ind1,P1,P2,NInd2))
      ;
      (NInd1=Ind1,NInd2=Ind2)),
      cruzamento(Resto,Resto1).

cruzar(Ind1,Ind2,P1,P2,NInd1) :-
     sublista(Ind1,P1,P2,Sub1),
     tarefas(NumT), R is NumT-P2,
     rotate_right(Ind2,R,Ind21),
     elimina(Ind21,Sub1,Sub2),
     insere(Sub2,Sub1,P2,NInd1).
```

P1 e P2 - pontos de cruzamento

cruzamento(Pop,NovaPop)

insere Sub2 em Sub1 a partir de P2, obtendo NInd1

Problema - Exemplo (cont)

Avaliação da população

```
avalia_populacao([ ],[ ]).
avalia_populacao([Ind|Resto],[Ind*V|Resto1]):-
    avalia(Ind,V), avalia_populacao(Resto,Resto1).
```

Avaliação dos indivíduos

```
avalia(Seq,V) :- avalia(Seq,0,V).

avalia([ ],_,0).
avalia([T|Resto],Inst,V) :-
    tarefa(T,Dur,Prazo,Pen),
    InstFim is Inst+Dur,
    avalia(Resto,InstFim,VResto),
    (
        (InstFim =< Prazo,!, VT is 0)
          ;
        (VT is (InstFim-Prazo)*Pen)
    ),
    V is VT+VResto.
```

```
% tarefa(Id, TempoProcessamento,
                DataEntrega, Penalizacao).
```

```
Resto1 = [[t1,t3,t2,t5,t4] * 16,[t5,t3,t1,t4,t2] * 37,[t5,t4,t1,t3,t2] * 39, ...]
```

Problema - Exemplo (cont)

```
ordena_populacao(PopAv,PopAvOrd) :-
    msort(PopAv,PopAvOrd).

    % não remove elementos repetidos
    % ao contrário de sort/2
```

Problema Exemplo (cont)

Operador de Mutação

```
mutacao([ ],[ ]).
mutacao([Ind|Rest],[NInd|Rest1]) :-
    prob_mutacao(Pmut),
    ((maybe(Pmut),!,mutacao1(Ind,NInd)) ; NInd=Ind),
    mutacao(Rest,Rest1).

mutacao1(Ind,NInd) :-
    gerar_pontos_cruzamento(P1,P2),
    mutacao22(Ind,P1,P2,NInd).

mutacao22([G1|Ind],1,P2,[G2|NInd]) :-
    !, P21 is P2-1, mutacao23(G1,P21,Ind,G2,NInd).
mutacao22([G|Ind],P1,P2,[G|NInd]) :-
    P11 is P1-1, P21 is P2-1, mutacao22(Ind,P11,P21,NInd).

mutacao23(G1,1,[G2|Ind],G2,[G1|Ind]) :- !.
mutacao23(G1,P,[G|Ind],G2,[G|NInd]) :-
    P1 is P-1,
    mutacao23(G1,P1,Ind,G2,NInd).
```

Selecciona 2 genes para serem trocados

mutacao(Pop,NovaPop)

Operador de mutação

```
?- mutacao22([t1, t3, t2, t5, t4], 2, 4, L).
   Call: (7) mutacao22([t1, t3, t2, t5, t4], 2, 4, _G4647) ? creep
   Call: (8) _G4745 is 2+ -1 ? creep
   Exit: (8) 1 is 2+ -1 ? creep
   Call: (8) _G4748 is 4+ -1 ? creep
   Exit: (8) 3 is 4+ -1 ? creep
   Call: (8) mutacao22([t3, t2, t5, t4], 1, 3, _G4733) ? creep
   Call: (9) _G4754 is 3+ -1 ? creep
   Exit: (9) 2 is 3+ -1 ? creep
   Call: (9) mutacao23(t3, 2, [t2, t5, t4], _G4741, _G4742) ? creep
   Call: (10) _G4760 is 2+ -1 ? creep
   Exit: (10) 1 is 2+ -1 ? creep
   Call: (10) mutacao23(t3, 1, [t5, t4], _G4741, _G4748) ? creep
   Exit: (10) mutacao23(t3, 1, [t5, t4], t5, [t3, t4]) ? creep
   Exit: (9) mutacao23(t3, 2, [t2, t5, t4], t5, [t2, t3, t4]) ? creep
   Exit: (8) mutacao22([t3, t2, t5, t4], 1, 3, [t5, t2, t3, t4]) ? creep
   Exit: (7) mutacao22([t1, t3, t2, t5, t4], 2, 4, [t1, t5, t2, t3, t4]) ? creep
L = [t1, t5, t2, t3, t4].
```

Método Minimax

Aula 5

Método Minimax

- O método Minimax é o método mais conhecido para lidar com jogos.
- Admite-se que existe um **gerador de estados** e uma função que avalia a vantagem ou desvantagem de um dado estado.
- Considere-se que esta **função de avaliação** é um **estimador heurístico** das hipóteses de vitória do ponto de vista de um dos jogadores:
  - Quanto maior o valor, maiores serão as hipóteses do jogador vencer
  - Quanto menor o valor, maiores serão as hipóteses do oponente vencer
- Pretende-se **maximizar** o valor dado pela função de avaliação.

Método Minimax

- O objectivo é seleccionar a jogada que garanta a melhor situação ao fim de **n** jogadas
- A melhor situação corresponde ao estado cujo valor da função de avaliação seja o maior (problema de **maximização**)
- O objectivo é alcançado propagando o valor correspondendo ao melhor estado até ao nó raiz; este valor corresponde ao ganho mínimo que se obtém se optarmos pela jogada correcta

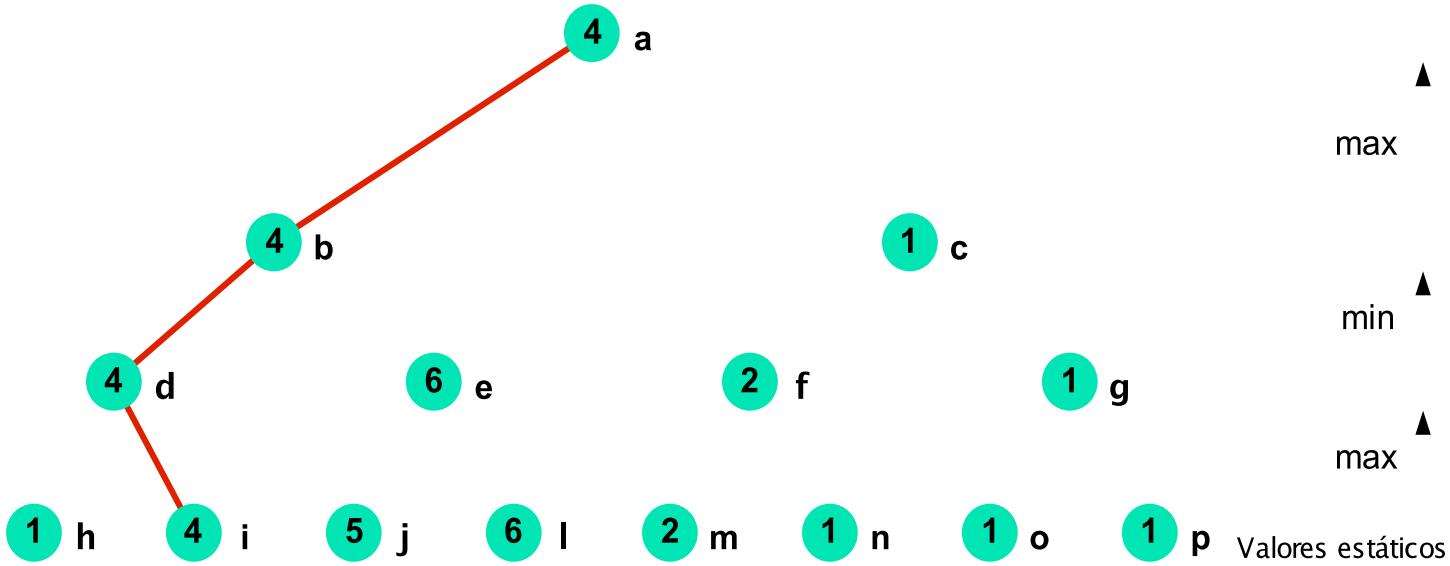
Método Minimax

A propagação de valores através dos nós da árvore é realizada da seguinte forma:

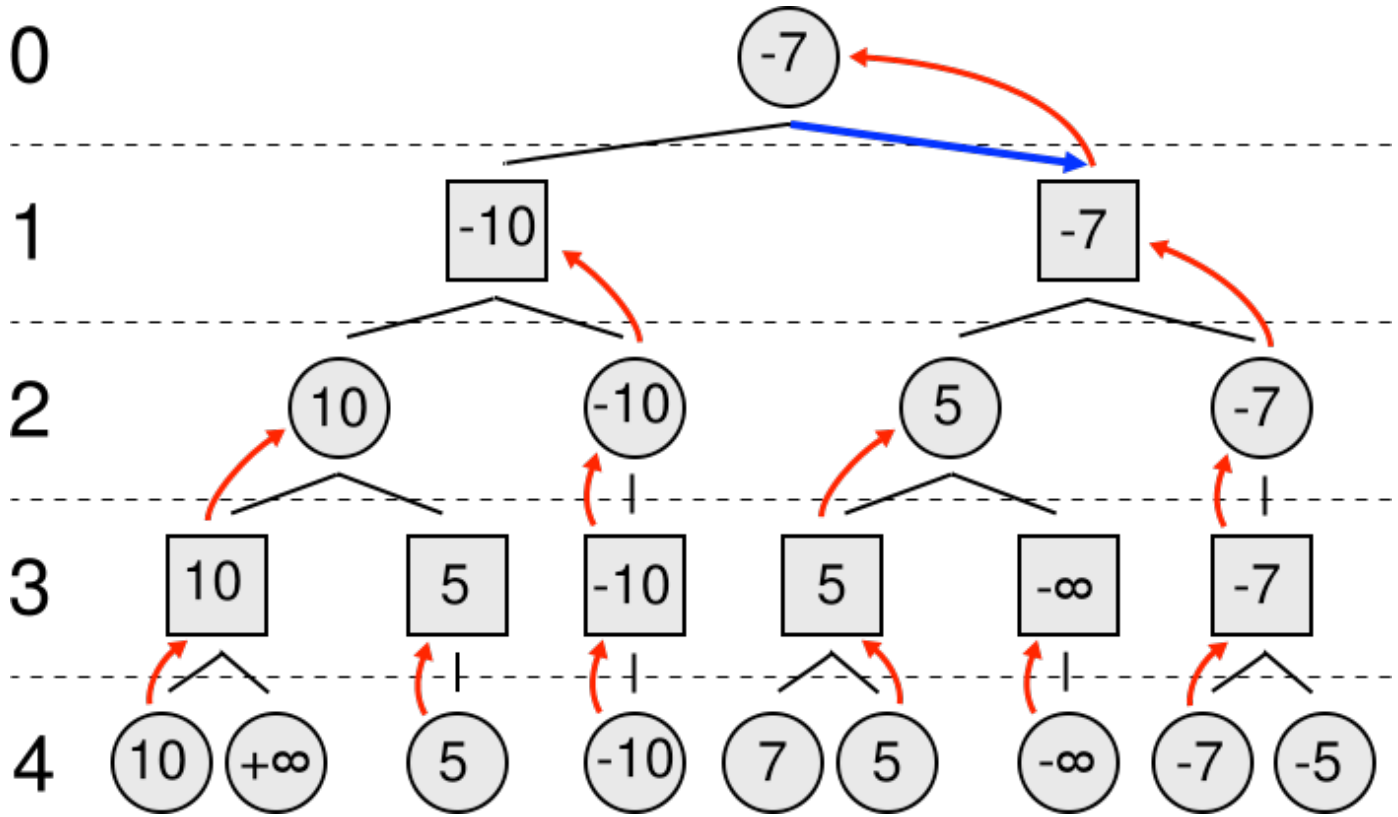
- Nos níveis que correspondem às acções do **jogador**, selecciona-se para propagação o maior valor (nível de **maximização**)
- Nos níveis que correspondem às acções do **oponente**, selecciona-se para propagação o menor valor (nível de **minimização**)
- Os valores associados aos **nós folha** são obtidos pela aplicação da função **heurística** de avaliação do mérito de cada um dos estados, de acordo com o ponto de vista do jogador

Método Minimax

Consideremos o problema genérico representado na figura. Os nós representam estados e os ramos representam as jogadas possíveis a partir de cada estado. Os valores associados aos nós folha são obtidos por uma função de avaliação.



Método Minimax



Círculos – MAX
Quadrados – MIN

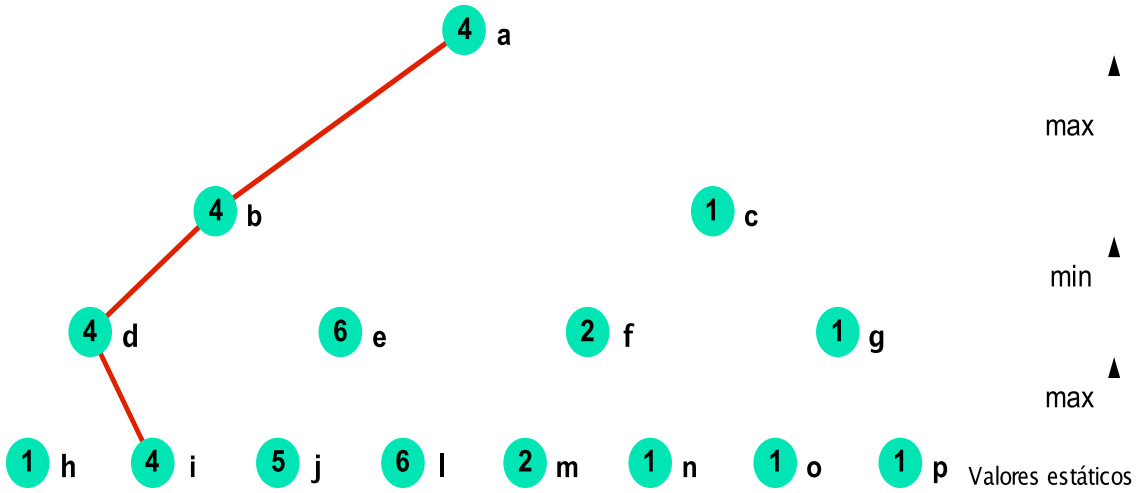
Fonte: Wikipedia

Método Minimax

Relações do problema genérico

```
max_to_move(d).
max_to_move(e).
max_to_move(f).
max_to_move(g).
min_to_move(b).
min_to_move(c).
min_to_move(h).
min_to_move(i).
min_to_move(j).
min_to_move(l).
min_to_move(m).
min_to_move(n).
min_to_move(o).
min_to_move(p).
```

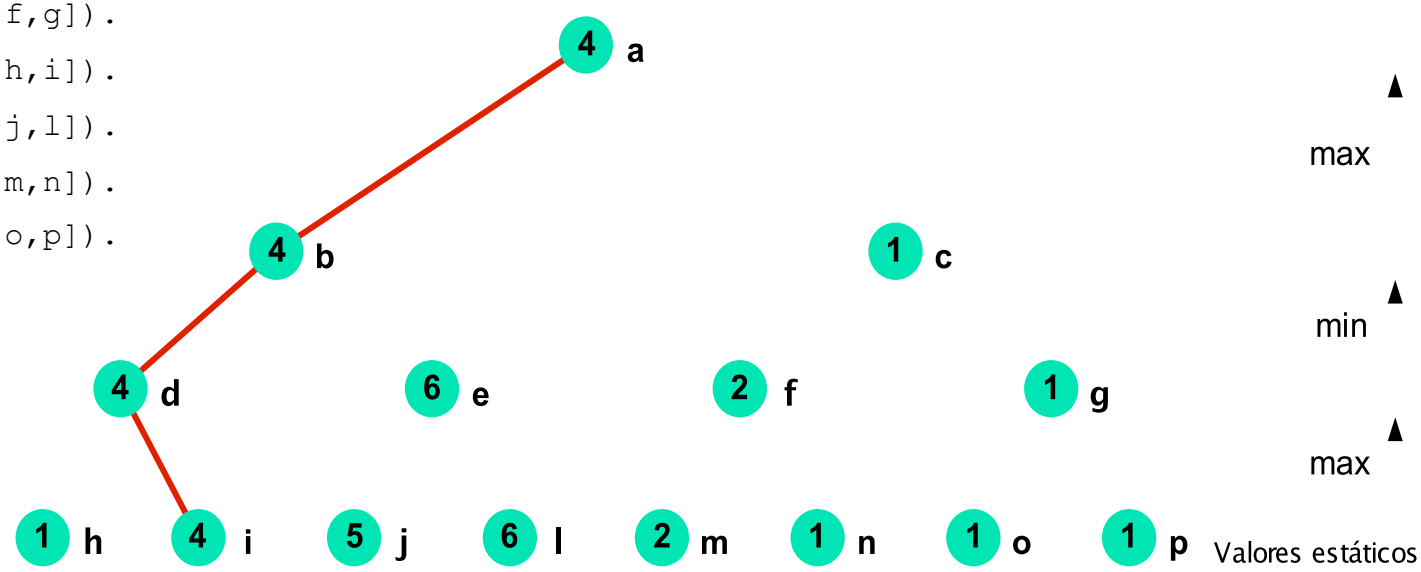
max_to_move -> max to move to



max_to_move/1 e **min_to_move/1** definem para cada nó qual o valor (maior ou menor) dos seus descendentes que será transferido para o próprio nó; o maior no caso de max_to_move/1 ou o menor no caso de min_to_move/1

Relações do problema genérico (cont.)

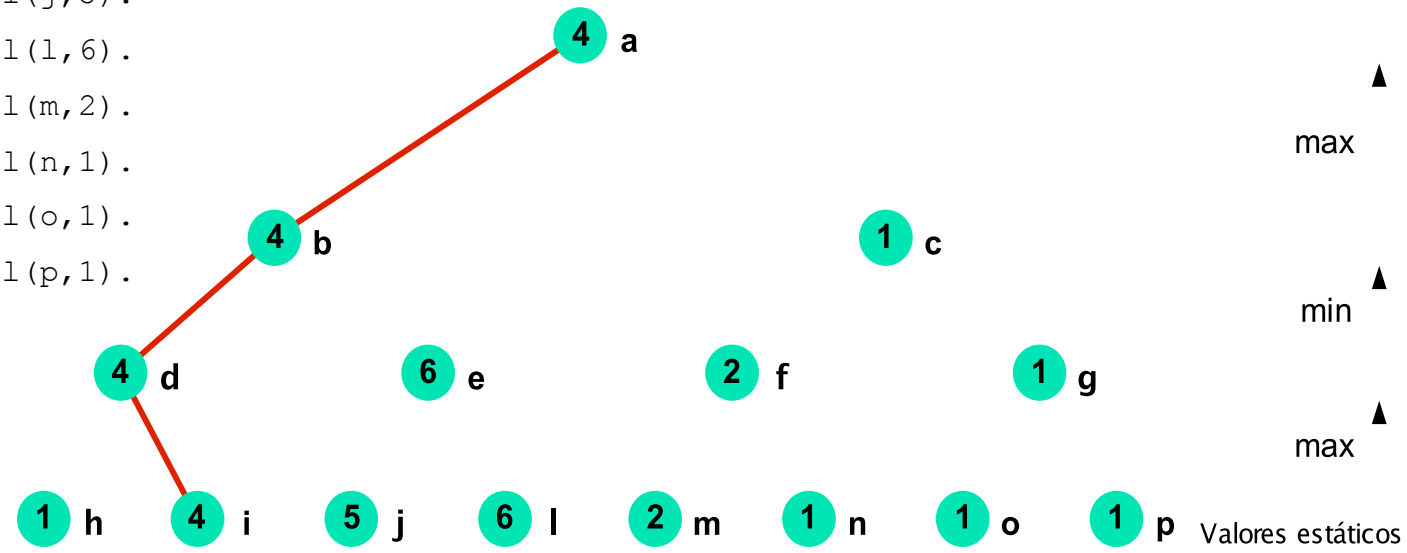
```
moves(a,[b,c]).
moves(b,[d,e]).
moves(c,[f,g]).
moves(d,[h,i]).
moves(e,[j,l]).
moves(f,[m,n]).
moves(g,[o,p]).
```



moves/2 define para cada nó a lista de nós descendentes (correspondem às jogadas válidas a partir de cada nó). **A lista de nós descendentes é gerada em função das regras do jogo**

Relações do problema genérico (cont.)

```
static_eval(h,1).
static_eval(i,4).
static_eval(j,5).
static_eval(l,6).
static_eval(m,2).
static_eval(n,1).
static_eval(o,1).
static_eval(p,1).
```



static_eval/2 define o valor do **mérito**, do ponto de vista do jogador representado pelo algoritmo, de cada **estado terminal** (nós folha da árvore), estimado através de **uma função de avaliação**

Método Minimax

Algoritmo

- O objectivo é **propagar** o valor **minimax** a partir dos nós folha.
- O predicado principal é

  minimax(Pos, BestPos, Val, NodesList)

  em que
  - **Val** é o valor minimax da posição **Pos**
  - **BestPos** é a melhor posição atingível a partir de **Pos** (a jogada a realizar para obter **Val**)
  - **NodesList** contém a sequência de nós usados para propagar o valor duma posição final até **Pos**.

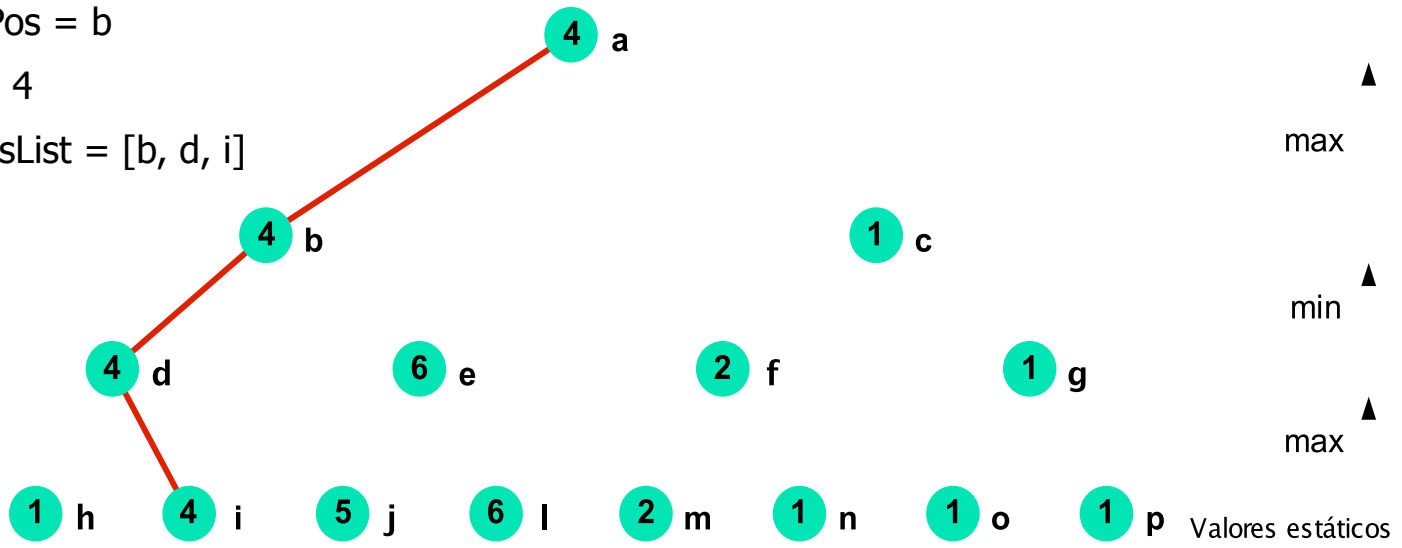
Resultado

?- minimax(a, BestPos, Val, NodesList).

BestPos = b

Val = 4

NodesList = [b, d, i]



Implementação do método Minimax com cortes Alpha-Beta:
Prolog Programming for Artificial Intelligence
Ivan Bratko
Addison-Wesley

Método Minimax

best/4 selecciona a "melhor" posição **BestPos** a partir de uma lista de posições candidatas **PosList** (descendentes de **Pos**). **Val** é o valor de **BestPos** e logo também de **Pos**. A melhor posição pode ser um máximo ou um mínimo, dependendo do nível de **Pos**.

```
minimax(Pos, BestPos, Val, NodesList):-
        moves(Pos, PosList), !,
        best(PosList, BestPos, Val, NodesList).
```

moves/2 falha se **Pos** é uma folha

1ª iteração

```
moves(a,[b,c]).
best([b|[c]],BestPos,BestVal,[BestPos|NodesList]) :-
                        ...
```

```
minimax(Pos, BestPos, Val, []):-
        static_eval(Pos, Val).
```

Pos é uma folha

Método Minimax

Algoritmo (cont.)

```
best([Pos], Pos, Val, NodesList):-
   minimax(Pos, _, Val, NodesList), !.
best([Pos1|PosList], BestPos, BestVal, [BestPos|NodesList]):-
   minimax(Pos1, _, Val1, NodesList1),

   best( PosList, Pos2, Val2, NodesList2),

   betterof(Pos1, Val1, Pos2, Val2, BestPos, BestVal,
        NodesList1, NodesList2, NodesList).
```

Quando a lista de candidatos contém apenas um elemento, **minimax/4** é usado para propagar o melhor valor a partir dos nós folha descendentes de **Pos**

Val1 é o melhor valor para **Pos1**

Val2 é o valor de **Pos2**, sendo esta a melhor posição entre as posições contidas em **PostList**

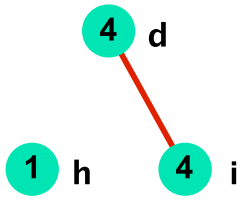
BestPos é a melhor posição entre **Pos1** e **Pos2**

1ª iteração

```
best([b|[c]],BestPos,BestVal,[BestPos|NodesList])
```

Método Minimax

Algoritmo (cont)




```
best([Pos], Pos, Val, NodesList):-
    minimax(Pos, _, Val, NodesList), !.
best([Pos1|PosList],BestPos,BestVal,[BestPos|NodesList]) :-
  minimax(Pos1,_,Val1,NodesList1),
  best(PosList,Pos2,Val2,NodesList2),
  betterof(Pos1,Val1,Pos2,Val2,BestPos,BestVal,NodesList1,NodesList2,NodesList).
```

Pos1 = h
PosList = [i]
Val1 = 1
NodesList1 = [ ]
Pos2 = i
Val2 = 4
NodesList2 = [ ]

Dados do **debugger** quando **best** é invocado após se ter atingido o nó folha **i**

Método Minimax

Algoritmo (cont.)

Regras de propagação dos valores dos nós:

v(P) – valor da função de avaliação aplicada ao estado P (nó folha)

V(P) – valor propagado para o estado P a partir dos estados descendentes de P

$V(P) = v(P)$ se P é um estado que ocupa um nó folha

$V(P) = \max_i V(P_i)$ se P é um estado relativo a um nível de maximização

$V(P) = \min_i V(P_i)$ se P é um estado relativo a um nível de minimização

Método Minimax

Algoritmo (cont.)

BestPos BestVal

```
betterof(Pos0, Val0, Pos1, Val1, Pos0, Val0,
                     NodesLst0, NodesLst1, NodesLst0):-
     min_to_move(Pos0), Val0 > Val1, !.
betterof(Pos0, Val0, Pos1, Val1, Pos0, Val0,
                     NodesLst0, NodesLst1, NodesLst0):-
     max_to_move(Pos0), Val0 < Val1, !.
betterof(Pos0, Val0, Pos1, Val1, Pos1, Val1,
                     NodesLst0, NodesLst1, NodesLst1).
```

```
max_to_move(f).
max_to_move(g).


?- betterof(f, 2, g, 1, BestPos, Val, _, _, _).
BestPos = g
Val = 1
```