Métodos de Pesquisa

Depth First Search
Breadth First Search
Best First Search
Branch and Bound
A-Star

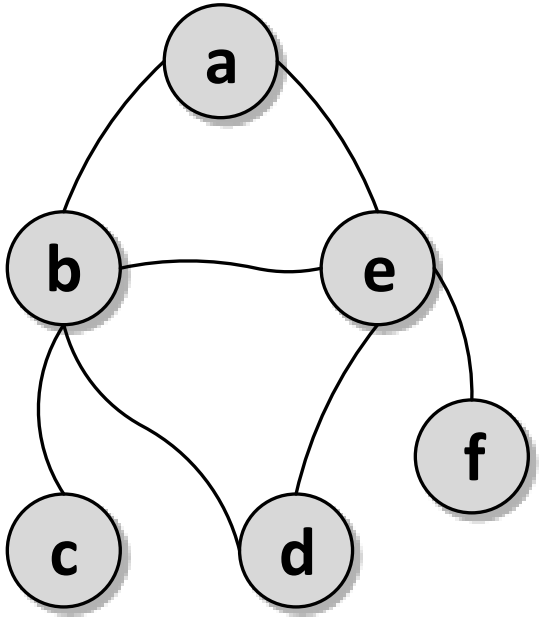
Depth First Search

Depth First Search (DFS)
Caracterização

Método de pesquisa em grafos/árvores não informado

- Ideia base: **Primeiro em profundidade**. Avançar no grafo/árvore (em profundidade) enquanto for possível, se não for, **recuar (backtrack)** e tentar alternativamente outros caminhos
- **Estrutura de dados requerida muito simples e leve** (lista com nodos visitados ou caminho actual)
- **Não garante caminho mais curto ou com menos passos em primeiro lugar**

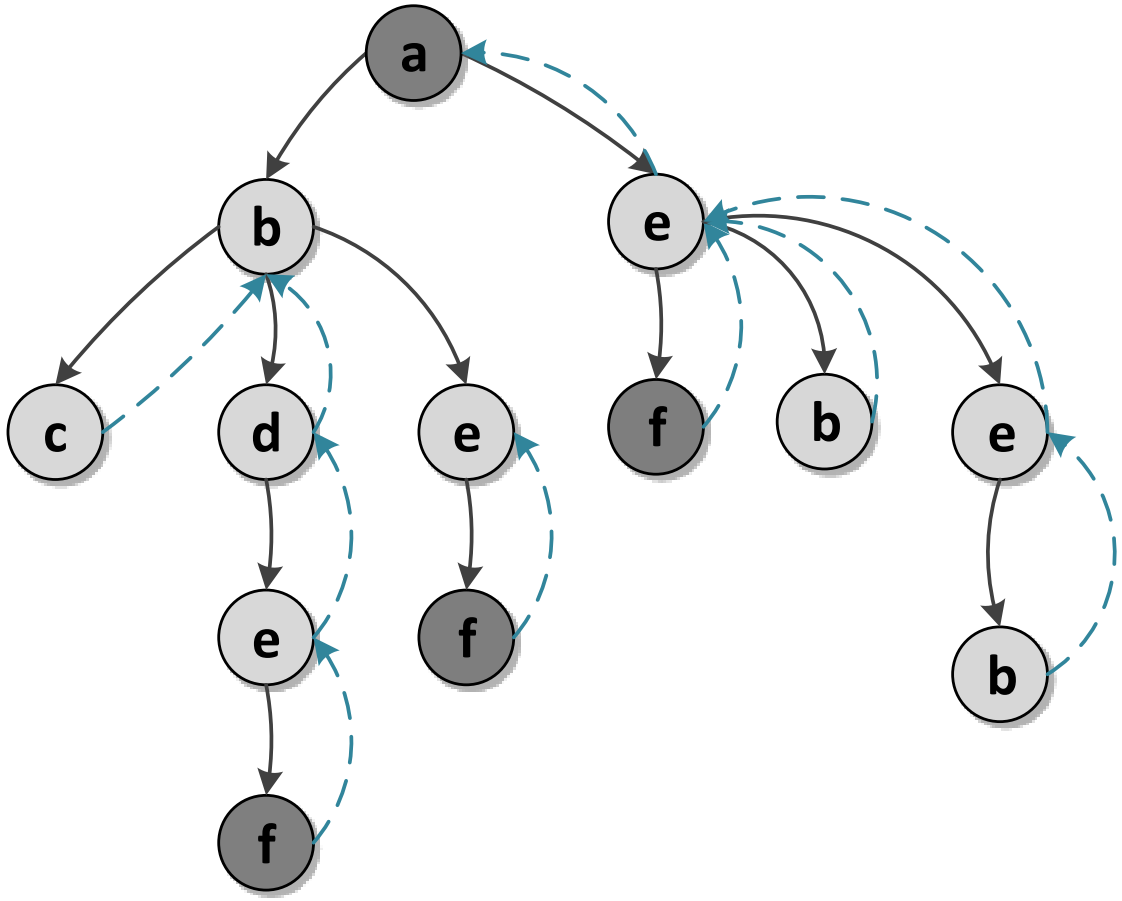
Depth First Search (DFS)
Grafo Exemplo



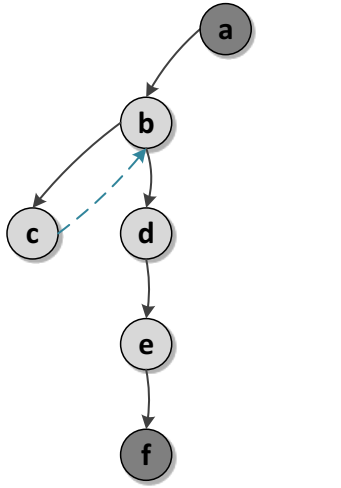

```
%edge(Node1,Node2)
edge(a,b).
edge(a,e).
edge(b,c).
edge(b,d).
edge(b,e).
edge(d,e).
edge(e,f).

% ligacoes bidireccionais
connect(X,Y):-
        edge(X,Y);edge(Y,X).
```

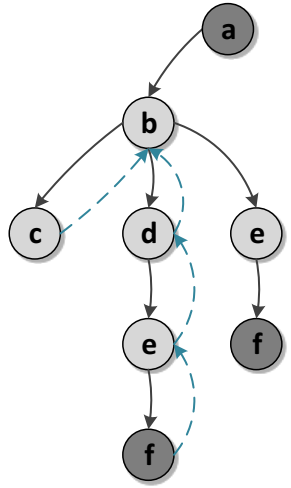
Depth First Search (DFS)
Implementação Prolog / Soluções

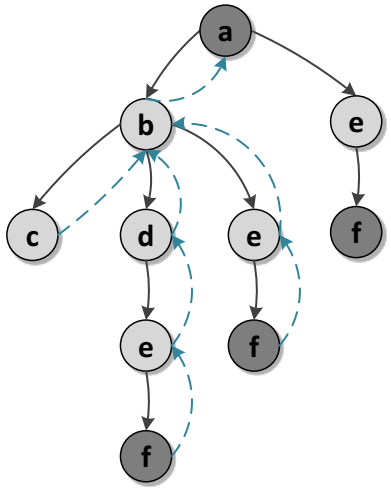
```
dfs(Orig,Dest,Cam):-
        dfs2(Orig,Dest,[Orig],Cam).

%condicao final: nodo actual = destino
dfs2(Dest,Dest,LA,Cam):-
        %caminho actual esta invertido
        reverse(LA,Cam).

dfs2(Act,Dest,LA,Cam):-
        %testar ligacao entre ponto
        %actual e um qualquer X
        connect(Act,X),

        %testar nao circularidade p/ nao
        %visitar nodos ja visitados
        \+ member(X,LA),
        %chamada recursiva
        dfs2(X,Dest,[X|LA],Cam).
```

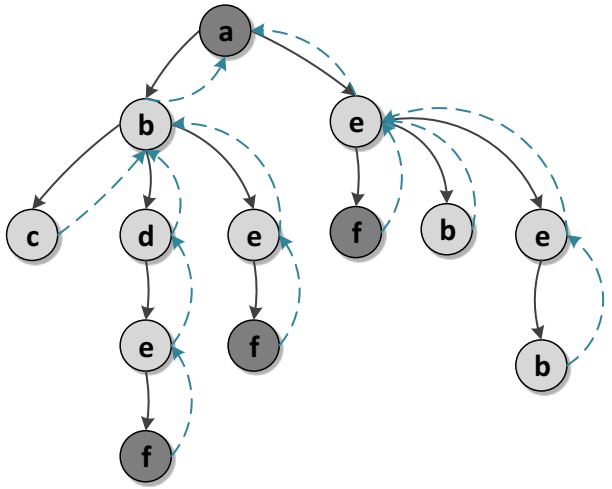


Depth First Search (DFS)
Espaço do problema / Soluções









Depth First Search (DFS)
Implementação Prolog / Soluções

```
dfs(Orig,Dest,Cam):-
       dfs(Orig,Dest,[Orig],Cam).

dfs(Dest,Dest,LA,Cam):-
       reverse(LA,Cam).

dfs(Act,Dest,LA,Cam):-
       connect(Act,X),
       \+ member(X,LA),
       dfs(X,Dest,[X|LA],Cam).
```

```
?- dfs(a,f,Caminho).
[a]
[b,a]
[b,a]
[d,b,a]
[e,d,b,a]
[f,e,d,b,a]
Caminho = [a, b, d, e, f] ;
[b,a]
[e,b,a]
[f,e,b,a]
Caminho = [a, b, e, f] ;
[e,b,a]
[a]
[e,a]
[f,e,a]
Caminho = [a, e, f] ;
[e,a]
[b,e,a]
[b,e,a]
[e,a]
[d,e,a]
[b,d,e,a]
false.
```

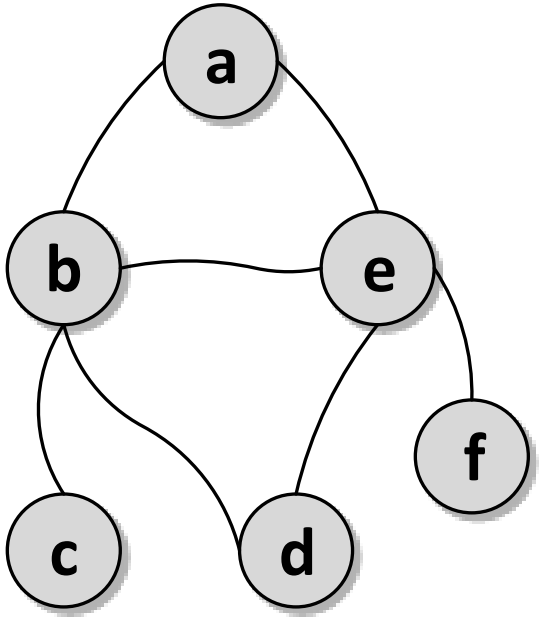
Breadth First Search

Breadth First Search (BFS)
Caracterização

Método de pesquisa em grafos/árvores não informado

- Ideia base: **Primeiro em largura.** A partir de um nodo são explorados todos os nodos adjacentes e posteriormente são explorados os nodos acessiveis através dos adjacentes (nível seguinte) e assim sucessivamente
- **Estrutura de dados requerida pesada** pois é requerido armazenar todos caminhos que ainda podem ser expandidos (fila de caminhos)
- **Garante que caminho com menos passos** em encontrado em primeiro lugar

Breadth First Search (BFS)
Grafo Exemplo

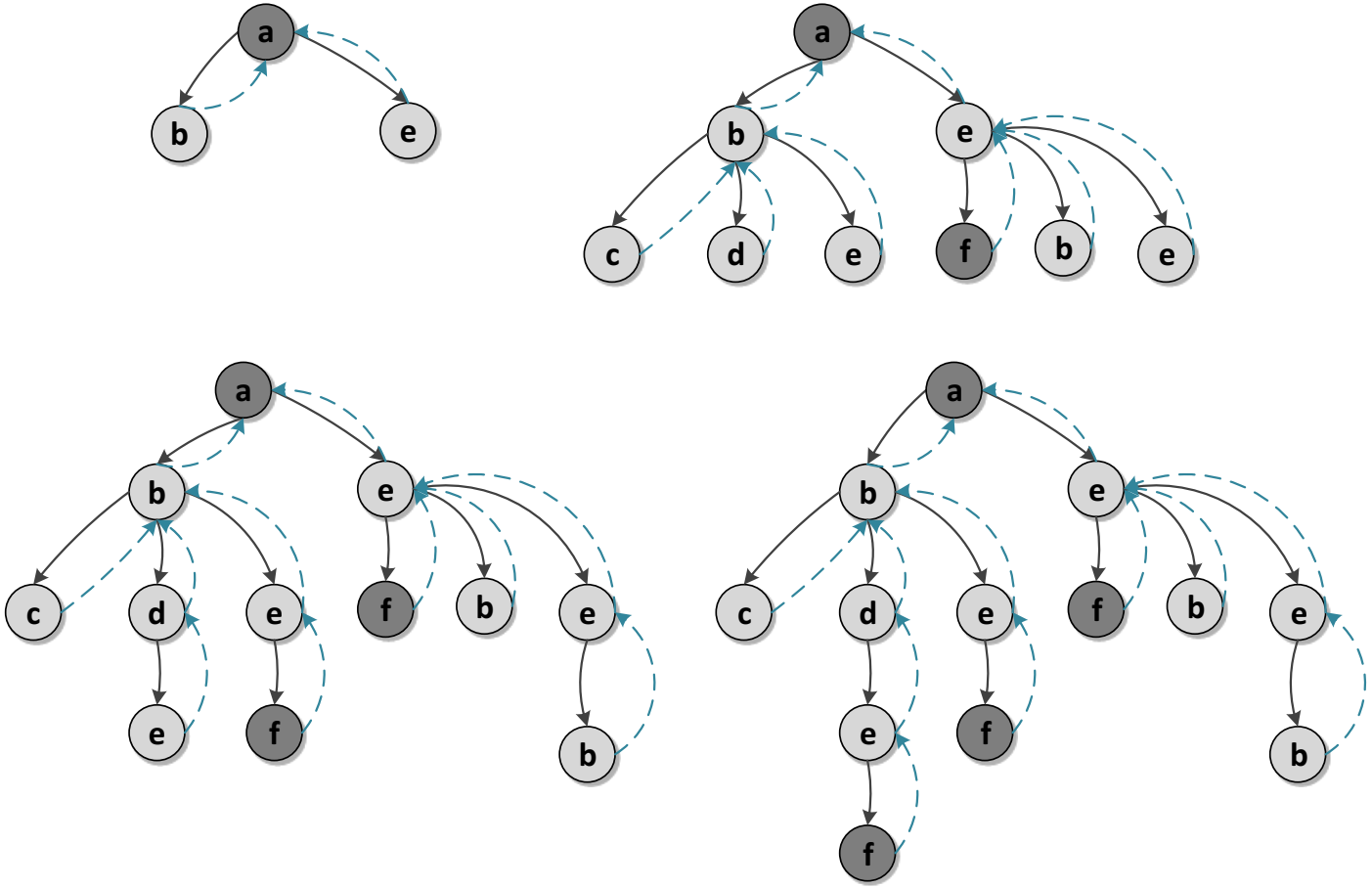



```
%edge(Node1,Node2)
edge(a,b).
edge(a,e).
edge(b,c).
edge(b,d).
edge(b,e).
edge(d,e).
edge(e,f).

% ligacoes bidireccionais
connect(X,Y):-
        edge(X,Y);edge(Y,X).
```

Breadth First Search (BFS) Implementação Prolog

```
bfs(Orig,Dest,Cam):-bfs2(Dest,[[Orig]],Cam).

%condicao final: destino = nó à cabeça do caminho actual
bfs2(Dest,[[Dest|T]|_],Cam):-
        %caminho actual está invertido
        reverse([Dest|T],Cam).

bfs2(Dest,[LA|Outros],Cam):-
        LA=[Act|_],
        %calcular todos os nós adjacentes nao visitados e
        %gerar um caminho novo c/ cada nó e caminho actual
        findall([X|LA],
                (Dest\==Act,connect(Act,X),\+ member(X,LA)),
                Novos),
        %novos caminhos são colocados no final da lista
        %p/ posterior exploracao
        append(Outros,Novos,Todos),
        %chamada recursiva
        bfs2(Dest,Todos,Cam).
```

Breadth First Search (BFS)
Espaço do Problema / Soluções



Depth First Search (DFS) Implementação Prolog / Soluções

```
bfs(Orig,Dest,Cam):-
bfs2(Dest,[[Orig]],Cam).

bfs2(Dest,[[Dest|T]|_],Cam):-
        reverse([Dest|T],Cam).


bfs2(Dest,[LA|Outros],Cam):-
           LA=[Act|_],

         findall([X|LA],
         Dest\==Act,connect(Act,X),
        \+member(X,LA)),Novos),

         append(Outros,Novos,Todos),
         bfs2(Dest,Todos,Cam).
```

```
?- bfs(a,f,Caminho).
[a]
[b,a]
[e,a]
[c,b,a]
[d,b,a]
[e,b,a]
[f,e,a]
Caminho = [a, e, f] ;
[f,e,a]
[b,e,a]
[d,e,a]
[e,d,b,a]
[f,e,b,a]
Caminho = [a, b, e, f] ;
[f,e,b,a]
[d,e,b,a]
[c,b,e,a]
[d,b,e,a]
[b,d,e,a]
[f,e,d,b,a]
Caminho = [a, b, d, e, f] ;
[f,e,d,b,a]
[c,b,d,e,a]
false.
```

Depth First Search (DFS)
Usando Implementação do BFS com alteração

```
bfs(Orig,Dest,Cam):-bfs2(Dest,[[Orig]],Cam).

%condicao final: destino = nó à cabeça do caminho actual
bfs2(Dest,[[Dest|T]|_],Cam):-
        %caminho actual está invertido
        reverse([Dest|T],Cam).

bfs2(Dest,[LA|Outros],Cam):-
        LA=[Act|_],
        %calcular todos os nós adjacentes nao visitados e
        %gerar um caminho novo c/ cada nó e caminho actual
        findall([X|LA],
                (Dest\==Act,connect(Act,X),\+ member(X,LA)),
                Novos),
        %novos caminhos são colocados no inicio da lista
        %p/ exploracao imediata
        append(Novos, Outros, Todos),
        %append(Outros, Novos, Todos),

        %chamada recursiva
        bfs2(Dest,Todos,Cam).
```

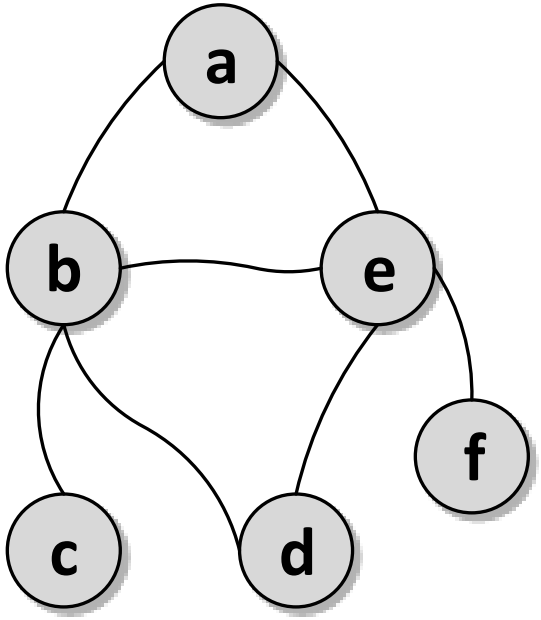
Best First Search

Best First Search (BestFS)
Caracterização

Método de pesquisa em grafos/árvores informado

- Ideia base: Este método é similar ao DFS com uma diferença, a decisão sobre qual o nodo a explorar de seguida ser feita com base **num critério de decisão local**
- **Solução final é resultado da soma dos máximos locais** e como tal pode não ser o máximo global
- **No caso de produzir solução, esta é obtida muito rapidamente** pois não são exploradas múltiplas soluções; não existe backtracking
- **Não garante que caminho com menos passos ou mais barato** seja encontrado em primeiro lugar, de facto pode não gerar qualquer solução

Best First Search (BestFS)
Grafo Exemplo



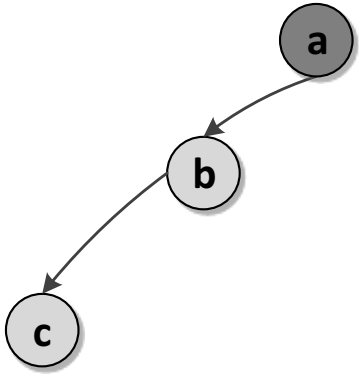

```
%edge(Node1,Node2)
edge(a,b).
edge(a,e).
edge(b,c).
edge(b,d).
edge(b,e).
edge(d,e).
edge(e,f).

% ligacoes bidireccionais
connect(X,Y):-
        edge(X,Y);edge(Y,X).
```

Best First Search (BestFS) Implementação Prolog

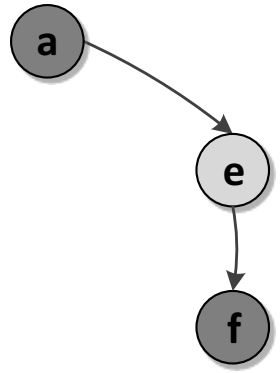
```
bestfs(Orig,Dest,Cam):-
        bestfs2(Dest,[Orig],Cam).

%condicao final: destino = nó à cabeça do caminho actual
bestfs2(Dest,[Dest|T],Cam):- !,
        %caminho actual está invertido
        reverse([Dest|T],Cam).

bestfs2(Dest,LA,Cam):-
        LA=[Act|_],
        %calcular todos os nodos adjacentes nao visitados e
        % guardar um tuplo com estimativa e novo caminho
        findall((EstX,[X|LA]),
                (connect(Act,X),\+ member(X,LA), estimativa(X,Dest,EstX)),Novos),
        %ordenar pela estimativa
        sort(Novos,NovosOrd),
        %extrair o melhor que está à cabeça
        NovosOrd = [(_,Melhor)|_],
        %chamada recursiva
        bestfs2(Dest,Melhor,Cam).
```

Best First Search (BestFS)
Espaço do Problema / Soluções

Estimativa nula: método fica identico ao DFS mas sem backtracking pelo que não produz soluções

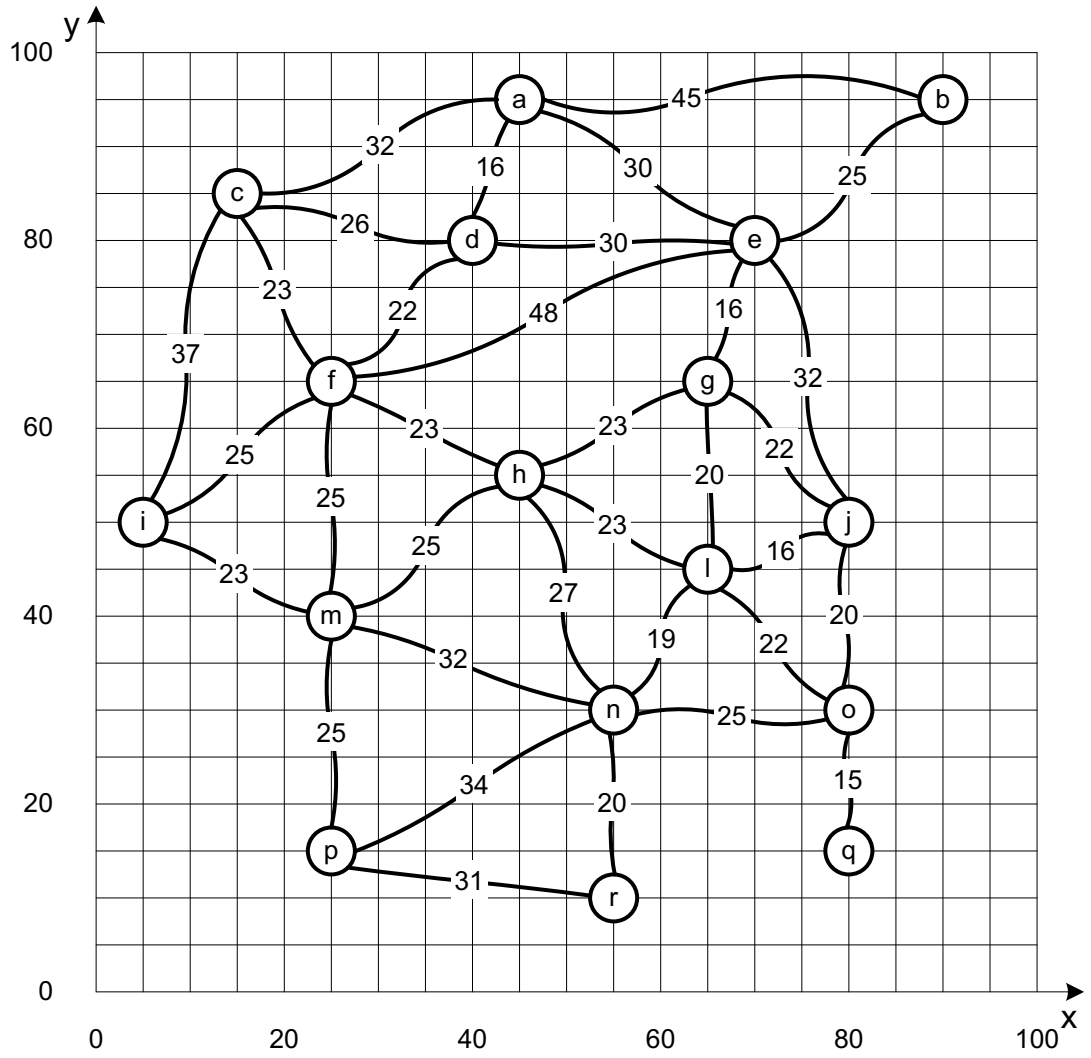


Situação em que estimativa e->f melhor do que b->f



Best First Search (BestFS)
(Outro) Grafo exemplo

```
%node(Nodo,PosX,PosY)
node(a,45,95).
node(b,90,95).
…
%edge(Nodo1,Nodo2,Custo)
edge(a,b,45).
edge(a,c,32).
edge(a,d,16).
…

%ligação bidireccional
connect(X,Y,C):-
        edge(X,Y,C);
        edge(Y,X,C).
…
```



Best First Search (BestFS) – versão c/custos Implementação Prolog / Soluções

```
bestfs(Orig,Dest,Cam,Custo):-
        bestfs2(Dest,(0,[Orig]),Cam,Custo).

bestfs2(Dest,(Custo,[Dest|T]),Cam,Custo):-
        !,
        reverse([Dest|T],Cam).

bestfs2(Dest,(Ca,LA),Cam,Custo):-
        LA=[Act|_],
        findall((EstX,CaX,[X|LA]),
        (connect(Act,X,CX),\+ member(X,LA),
        estimativa(X,Dest,EstX),
        CaX is Ca+CX),Novos),

        sort(Novos,NovosOrd),
        NovosOrd = [(_,CM,Melhor)|_],
        bestfs2(Dest,(CM,Melhor),Cam,Custo).
```

```
?- bestfs(a,r,Cam,Custo).
Cam = [a, d, f, m, n, r],
Custo = 115.
```

Best First Search (BestFS)
Estimativa

```
estimativa(Nodo1,Nodo2,Estimativa):-
        node(Nodo1,X1,Y1),
        node(Nodo2,X2,Y2),
        Estimativa is sqrt((X1-X2)^2+(Y1-Y2)^2).
```

Branch and Bound (BnB)

Branch and Bound (Bnb)
Caracterização

Método de pesquisa em grafos/árvores informado

- Ideia base: Este método **calcula de forma sistemática e exaustiva todos os caminhos possíveis expandindo aquele que tem um custo acumulado** (até o momento) **mais baixo**
- **O primeiro caminho produzido é obrigatoriamente o melhor (custo mais baixo)**
- O próximo nodo a expandir frequentemente não é um descendente directo do último nodo expandido
- Solução pesada visto que considera em cada passo a totalidade dos caminhos parciais calculados até o momento
- **Este algoritmo não é sensível à distância do nodo actual à nodo destino**

Branch and Bound (BFS)
Implementação Prolog

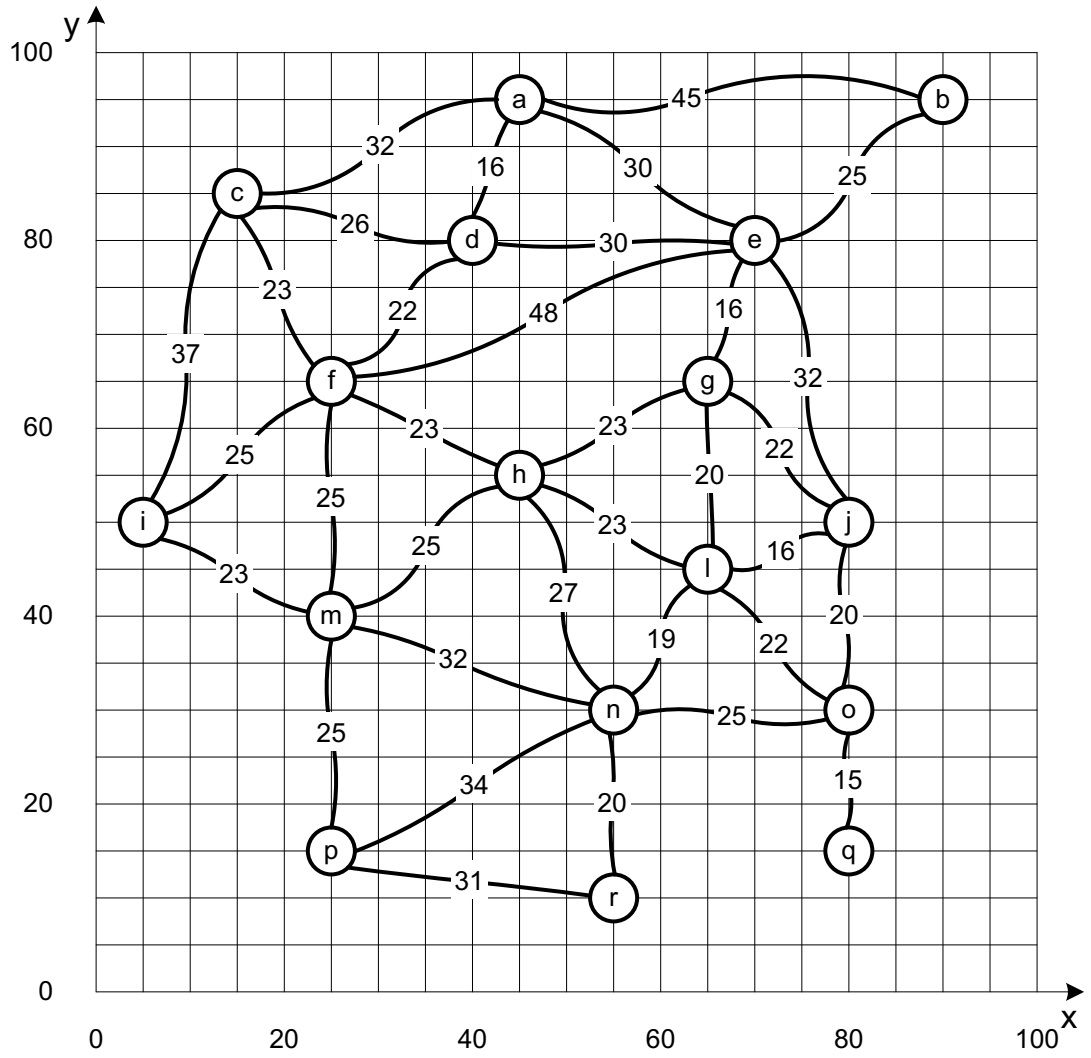
```
bnb(Orig,Dest,Cam,Custo):- bnb2(Dest,[(0,[Orig])],Cam,Custo).

%condicao final: destino = nó à cabeça do caminho actual
bnb2(Dest,[(Custo,[Dest|T])|_],Cam,Custo):-
        %caminho actual está invertido
        reverse([Dest|T],Cam).

bnb2(Dest,[(Ca,LA)|Outros],Cam,Custo):-
        LA=[Act|_],
        %calcular todos os nodos adjacentes nao visitados e
        %gerar tuplos c/ um caminho novo juntado o nodo + caminho actual
        % o custo de cada caminho é o custo acumulado + peso do ramo
        findall((CaX,[X|LA]),
        (Dest\==Act,edge(Act,X,CustoX),\+ member(X,LA),CaX is CustoX + Ca),Novos),
        %os novos caminhos sao adicionados aos caminhos não explorados
        append(Outros,Novos,Todos),
        %a ordenação (não sendo o mais eficiente) garante que
        % o melhor caminho fica na primeira posição
        sort(Todos,TodosOrd),
        %chamada recursiva
        bnb2(Dest,TodosOrd,Cam,Custo).
```

Branch and Bound (BFS)
Implementação Prolog

```
bnb(Orig,Dest,Cam,Custo):-
bnb2(Dest,[(0,[Orig])],Cam,Custo).

bnb2(Dest,[(Custo,[Dest|T])|_],Cam,Custo):-
     reverse([Dest|T],Cam).

bnb2(Dest,[(Ca,LA)|Outros],Cam,Custo):-
     LA=[Act|_],
     findall((CaX,[X|LA]),
     (Dest\==Act,edge(Act,X,CustoX),\+ member(X,LA),
     CaX is CustoX + Ca),Novos),
     append(Outros,Novos,Todos),
     sort(Todos,TodosOrd),
     bnb2(Dest,TodosOrd,Cam,Custo).
```

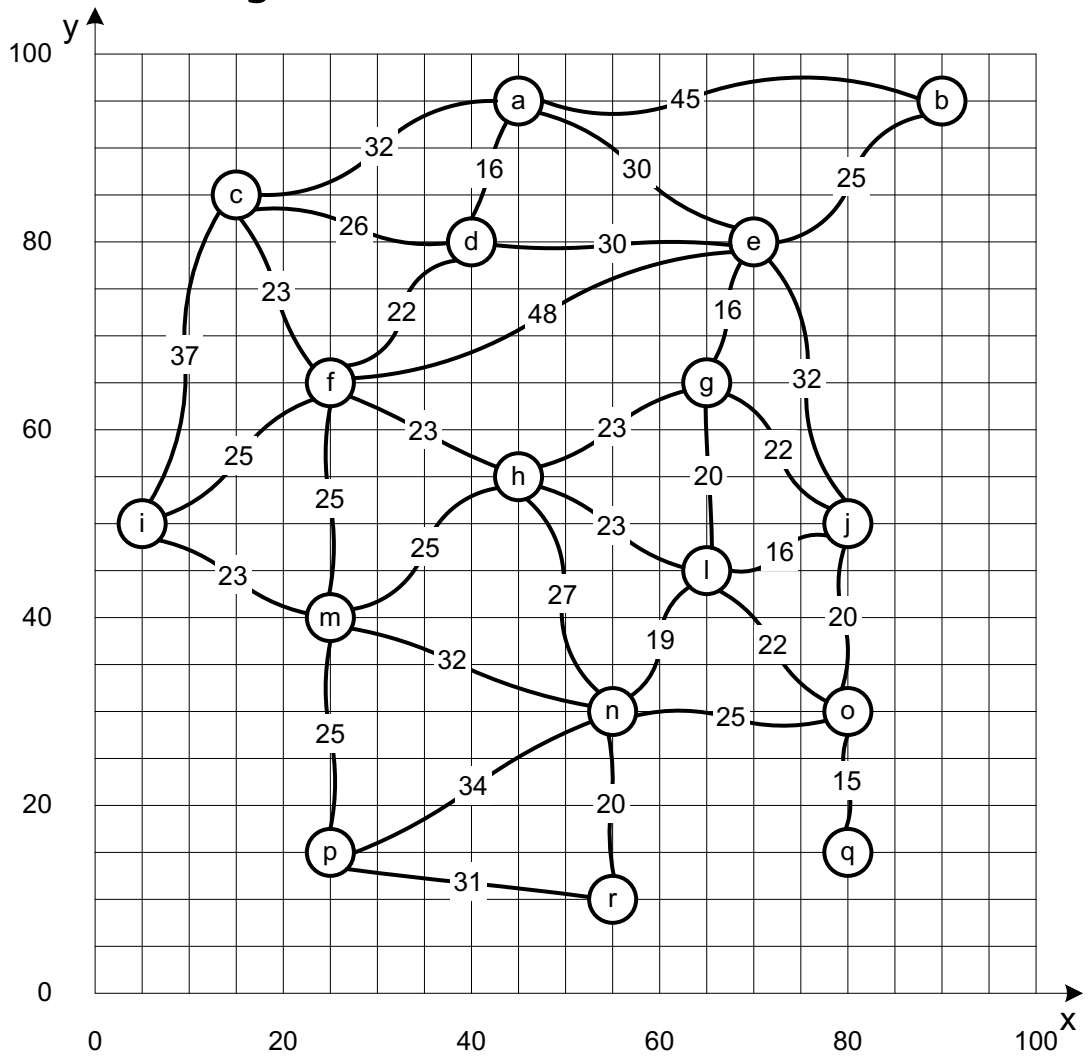
Branch and Bound (Bnb)
Grafo exemplo

```
%node(Nodo,PosX,PosY)
node(a,45,95).
node(b,90,95).
…
%edge(Nodo1,Nodo2,Custo)
edge(a,b,45).
edge(a,c,32).
edge(a,d,16).
…

%ligação bidireccional
connect(X,Y,C):-
        edge(X,Y,C);
        edge(Y,X,C).
…
```



Branch and Bound (BFS)
Espaço do problema / soluções

```
?- bnb(a,r,Cam,Custo).
Cam = [a, e, g, l, n, r],
Custo = 105 ;
Cam = [a, d, f, h, n, r],
Custo = 108 ;
Cam = [a, d, f, m, n, r],
Custo = 115 ;
Cam = [a, e, g, h, n, r],
Custo = 116 ;
Cam = [a, e, j, l, n, r],
Custo = 117 ;
Cam = [a, d, f, m, p, r],
Custo = 119 ;
Cam = [a, d, e, g, l, n, r],
Custo = 121 ;
Cam = [a, d, f, h, l, n, r],
Custo = 123 ;
…
```



A-Star Search

A-Star (A* ou a-Star)
Caracterização

Método de pesquisa em grafos/árvores informado

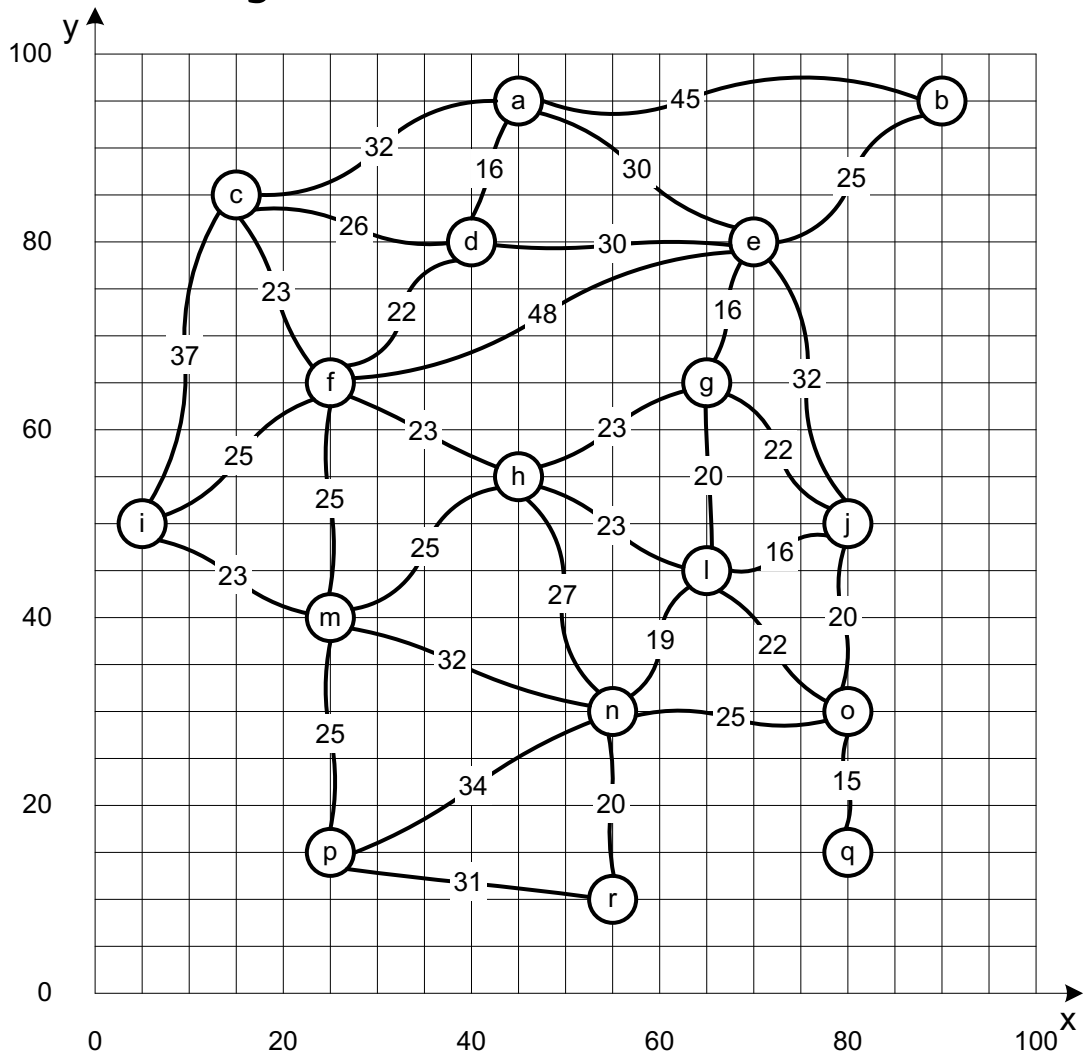
- Ideia base: Este método **conjuga a possibilidade de utilizar uma função estimativa** (utilizada no Best First Search) com a **contabilização dos custos acumulados conhecidos** (utilizada no Branch and Bound)
- **O primeiro caminho produzido é obrigatoriamente o melhor (custo mais baixo)**
- No caso da função estimativa ser eficaz a solução é produzida muito mais rapidamente do que no Branch and Bound; no caso de não existir função estimativa este método degenera no Branch and Bound
- **Este algoritmo é sensível à distância do nodo actual à nodo destino**

A-Star
Implementação Prolog

```
aStar(Orig,Dest,Cam,Custo):-
      aStar2(Dest,[(_,0,[Orig])],Cam,Custo).

aStar2(Dest,[(_,Custo,[Dest|T])|_],Cam,Custo):-
      reverse([Dest|T],Cam).

aStar2(Dest,[(_,Ca,LA)|Outros],Cam,Custo):-
      LA=[Act|_],
      findall((CEX,CaX,[X|LA]),
            (Dest\==Act,edge(Act,X,CustoX),\+ member(X,LA),
            CaX is CustoX + Ca, estimativa(X,Dest,EstX),
            CEX is CaX +EstX),Novos),
      append(Outros,Novos,Todos),
      sort(Todos,TodosOrd),
      aStar2(Dest,TodosOrd,Cam,Custo).
```

Branch and Bound (BFS)
Espaço do problema / soluções

```
?- aStar(a,r,Cam,Custo).
Cam = [a, e, g, l, n, r],
Custo = 105 ;
Cam = [a, d, f, h, n, r],
Custo = 108 ;
Cam = [a, d, f, m, n, r],
Custo = 115 ;
Cam = [a, e, g, h, n, r],
Custo = 116 ;
Cam = [a, e, j, l, n, r],
Custo = 117 ;
Cam = [a, d, f, m, p, r],
Custo = 119 ;
Cam = [a, d, e, g, l, n, r],
Custo = 121 ;
Cam = [a, d, f, h, l, n, r],
Custo = 123 ;
Cam = [a, e, g, j, l, n, r],
Custo = 123 ;
…
```



Comparação
Best First, Branch and Bound e A-Star

```
?- bestfs(a,r,Cam,Custo).
Cam = [a, d, f, m, n, r],
Custo = 115.
(apenas 1 solução)

?- bnb(a,r,Cam,Custo).
Cam = [a, e, g, l, n, r],
Custo = 105 ;

?- aStar(a,r,Cam,Custo).
Cam = [a, e, g, l, n, r],
Custo = 105 ;
```