# الجامعة التكنولوجية

## قسم علوم الحاسوب / فرع البرمجيات

## المرحلة الرابعة - الكورس الثاني 2024- 2025

## مادة طرق البحث الذكي / منهاج المادة العملي

## أ.د. حسنين سمير عبدالله

**% Forward Search Program.**

```
predicates
run(char,char).
find_rout (char,char).
path(char,char).
write_rout.
database
rout(char,char).
clauses
run(_,_):-retractall(_),fail.
run(S,E):-find_rout(S,E),fail.
run(_,_):-write_rout.
find_rout(S,E):-path(S,E),asserta(rout(S,E)),!.
find_rout(S,E):-path(S,M), % Here the cut(!) should be active in case If you
want only one solution.
find_rout(M,E), asserta(rout(S,M)).
write_rout:-rout(S,E),
write("\nSearching from ", S," t o   ",E), nl, fail.
write_rout.
path('a','b').
path('a','c').
path('a','d').
path('b','g').
path('c','g').
path('d','g').
/*goal:
run('a','g').
Searching from a t o  d
Searching from d t o g
Searching from a t o c
Searching from c t o g
Searching from a t o b
Searching from b t o g
yes*/
```

**% Backward Search Program.**

```
predicates
run(char,char).
find_rout (char,char).
path(char,char).
write_rout.
database
rout (char,char).
clauses
run(_,_):- retractall(_),fail.
run(S,E):-find_rout(S,E),fail.
run(_,_):- write_rout.
find_rout(S,E):-path(S,E),asserta(rout(S,E)).
find_rout(S,E):-path(M,E), %Here the cut(!) should be active in case If you
want only one solution.
find_rout(S,M), asserta(rout(M,E)).
write_rout:-rout(S,E),
write(" \nSearching from ", E," t o  ",S), nl, fail.
write_rout.
path('a','b').
path('a','c').
path('a','d').
path('b','g').
path('c','g').
path('d','g').
/*goal:
run('a','g').
Searching from g t o d
Searching from d t o a
Searching from g t o c
Searching from c t o a
Searching from g t o b
Searching from b t o a
Yes */
```

## Depth First Search

## Algorithm of Depth-First Search

```
Begin
   Open: = [Initial state];                                    %initialize
   Closed: = [ ];
   While open <> [ ] do                                        %state remain
     Begin
       Remove leftmost state from open, call it X;
       If X is a goal then return SUCCESS                      %goal found
         Else
         Begin
           Generate children of X;
           Put X on closed;
           Discard children of X if already on open or closed;   %loop check
           Put remaining children on left end of open           %stack
         End;
     End;
   Return FAIL                                                 %no states left
End.
```

**%Depth first search program**

```
domains
c=char.
l=c*.
predicates
depth(l,l,c).
difference(l,l,l).
append(l,l,l).
member(c,l).
print(l,l).
path(c,c).
clauses
depth([],_,_):-!,write("Goal is not found ").
depth([G|T_Open],Closed,G):-!,print([G|T_Open],Closed),write("Goal is found"), nl.
depth([H|T_Open],Closed,G):-print([H|T_Open],Closed), %Print Open & Closed.
findall(X,path(H,X),Children),%Find children of H.
append(Closed,[H],Closed1),%Put H in Closed.
difference(Children,T_Open,Children1), %Ignore children of H if already on Open or
difference(Children1,Closed1,Children2),%Closed
append(Children2,T_Open,Open1),%Put remaining children on left of Open.
depth(Open1,Closed1,G).
difference([],_,[]):- !.
difference([H|T],Z,[H|T1]):-not(member(H,Z)),!,
difference(T,Z,T1).
difference([_|T],Z,T1):-
difference(T,Z,T1).
member(H,[H|_]):-!.
member(H,[_|T]):-member(H,T).
append([],L,L):-!.
append([H|T],L,[H|M]):-append(T,L,M).
print(Open,Closed):-write("Open=",Open,"  ","Closed=",Closed),nl.
path('a','b').
path('a','c').
path('a','d').
path('b','e').
path('b','c').
path('d','c').
path('d','f').
path('c','g').
/*
goal:depth(['a'],[],'f').
Open=['a']  Closed=[]
Open=['b','c','d']  Closed=['a']
Open=['e','c','d']  Closed=['a','b']
Open=['c','d']  Closed=['a','b','e']
Open=['g','d']  Closed=['a','b','e','c']
Open=['d']  Closed=['a','b','e','c','g']
Open=['f']  Closed=['a','b','e','c','g','d']
Goal is found  */
```

## Breadth-First Search

## Algorithm of Breadth-First Search

```
Begin
  Open: = [Initial state];                                   %initialize
  Closed: = [ ];
  While open <> [ ] do                                       %state remain
    Begin
      Remove leftmost state from open, call it X;
      If X is a goal then return SUCCESS                     %goal found
        Else
        Begin
          Generate children of X;
          Put X on closed;
          Discard children of X if already on open or closed;   %loop check
          Put remaining children on right end of open           %queue
        End;
    End;
  Return FAIL                                                %no states left
End.
```

**%Breadth first search program**

```
domains
c=char.
l=c*.
predicates
breadth(l,l,c).
difference(l,l,l).
append(l,l,l).
member(c,l).
print(l,l).
path(c,c).
clauses
breadth([],_,_):-!,write("Goal is not found ").
breadth([G|T_Open],Closed,G):-!,print([G|T_Open],Closed),write("Goal is found "),nl.
breadth([H|T_Open],Closed,G):-print([H|T_Open],Closed),
findall(X,path(H,X),Children),
append(Closed,[H],Closed1),
difference(Children,T_Open,Children1),
difference(Children1,Closed1,Children2),
append(T_Open,Children2,Open1),%Put remaining children on rigth of Open.
breadth(Open1,Closed1,G).
difference([],_,[]):- !.
difference([H|T],Z,[H|T1]):-not(member(H,Z)),!,
difference(T,Z,T1).
difference([_|T],Z,T1):-difference(T,Z,T1).
member(H,[H|_]):-!.
member(H,[_|T]):-member(H,T).
append([],L,L):-!.
append([H|T],L,[H|M]):-append(T,L,M).
print(Open,Closed):-write("Open=",Open,"  ","Closed=",Closed),nl.
path('a','b').
path('a','c').
path('a','d').
path('b','e').
path('b','c').
path('d','c').
path('d','f').
path('c','g').
/*goal:breadth(['a'],[],'f').
Open=['a']     Closed=[]
Open=['b','c','d']  Closed=['a']
Open=['c','d','e']  Closed=['a','b']
Open=['d','e','g']   Closed=['a','b','c']
Open=['e','g','f']  Closed=['a','b','c','d']
Open=['g','f']    Closed=['a','b','c','d','e']
Open=['f']    Closed=['a','b','c','d','e','g']
```

Goal is found */

## Hill Climbing Search

## Algorithm of Hill Climbing Search

```
Begin
 Open: = [Initial state];                          %initialize
 Closed: = [ ];
 CS= initial state;
 Path= [initial state];
 Stop= FALSE;
 While open <> [ ] do                              %states remain
  Begin
   If CS=goal then return path
   Generate all children of CS and put them into open;
    If open= [ ] then
    Stop= TRUE
     Else
      Begin
       X= CS;
       For each state Y in open do
        Begin
         Compute the heuristic value of y (h(Y));
         If Y is better than X then
         X=Y
        End;
        If X is better than CS then
        CS=X
         Else
         Stop= TRUE;
      End;
  End;
  Return (FAIL);                                  %open is empty
End.
```

**%Hill climbing program**

```
domains
f=s(char,integer).
l=f*.
c=char.
i=integer.
predicates
hill(l,l,c).
append(l,l,l).
sort(l,l).
```

```
sum(l,i).
min(l,f).
del(f,l,l).
print(l,l).
dead_end(l,f).
equal_cost(l).
path(f,f).
clauses
hill([],_,_):-!,write("The goal is not found").
hill([s(G,H)|T_Open],Closed,G):-!,
print([s(G,H)|T_Open],Closed),
append(Closed,[s(G,H)],Path),
write("Goal is found & the resulted path is ",Path),nl,
sum(Path,N),write("Total cost=",N).
hill([H|T_Open],Closed,G):-print([H|T_Open],Closed),
findall(X,path(H,X),Children),not(dead_end(Children,H)),
append(Closed,[H],Closed1),
sort(Children,S_children),not(equal_cost(S_children)),
append(S_children,T_Open,Open1),
hill(Open1,Closed1,G).
append([],L,L):-!.
append([H|T],L,[H|T1]):-append(T,L,T1).
sum([],0).
sum([s(_,H)|T],N):-sum(T,N1),N=N1+H.
sort([],[]):-!.
sort(L,[M|T]):-min(L,M),
del(M,L,X),
sort(X,T).
min([M],M):-!.
min([s(A,X),s(_,Y)|T],M):-X<=Y,!,
min([s(A,X)|T],M).
min([_|T],M):-min(T,M).
del(X,[X|T],T):-!.
del(X,[H|T],[H|T1]):-del(X,T,T1).
dead_end([],H):-write("Serch is stopped because there is dead end= ",H),nl.
equal_cost([s(A,X),s(B,X)|_]):-S1=s(A,X),S2=s(B,X),
write("Serch is stopped because there are equal costs for the states= ",S1,"&",S2),nl.
print(Open,Closed):-write("Open=",Open," ","Closed=",Closed),nl.
path(s('a',0),s('b',4)).
path(s('a',0),s('c',5)).
path(s('a',0),s('d',3)).
path(s('b',4),s('e',3)).
path(s('b',4),s('c',1)).
path(s('d',3),s('c',2)).
path(s('d',3),s('f',5)). %path(s('d',3),s('f',1)).path(s('d',3),s('f',2)).
path(s('c',1),s('g',3)).
path(s('c',5),s('g',3)).
path(s('c',2),s('g',3)).
/* goal:hill([s('a',0)],[],'g').
Open=[s('a',0)] Closed=[]
Open=[s('d',3),s('b',4),s('c',5)] Closed=[s('a',0)]
Open=[s('c',2),s('f',5),s('b',4),s('c',5)] Closed=[s('a',0),s('d',3)]
Open=[s('g',3),s('f',5),s('b',4),s('c',5)] Closed=[s('a',0),s('d',3),s('c',2)]
Goal is found & the resulted path is [s('a',0),s('d',3),s('c',2),s('g',3)]
Total cost=8 */
goal: hill([s('a',0)],[],'g').%path(s('d',3),s('f',1)).
/* Open=[s('a',0)] Closed=[]
Open=[s('d',3),s('b',4),s('c',5)] Closed=[s('a',0)]
Open=[s('f',1),s('c',2),s('b',4),s('c',5)] Closed=[s('a',0),s('d',3)]
Search is stopped because there is dead end= s('f',1) */
/*
goal:hill([s('a',0)],[],'g').%path(s('d',3),s('f',2)).
Open=[s('a',0)] Closed=[]
Open=[s('d',3),s('b',4),s('c',5)] Closed=[s('a',0)]
Search is stopped because there are equal costs for the states= s('c',2)&s('f',2) */
```

## Best First Search

## Algorithm of Best-First Search

```
Begin
  Open: = [Initial state];                                  %initialize
  Closed: = [ ];
  While open <> [ ] do                                      %states remain
    Begin
      Remove leftmost state from open, call it X;
      If X = goal then return the path from initial to X
      Else
      Begin
        Generate children of X;
        For each child of X do
        Case
          The child is not on open or closed;
            Begin
              Assign the child a heuristic value;
              Add the child to open
            End;
          The child is already on open;
            If the child was reached by a shorter path
            Then give the state on open the shorter path
          The child is already on closed;
            If the child was reached by a shorter path then
              Begin
            Remove the state from closed;
            Add the child to open
          End;
      End;                                                    %case
      Put X on closed;
      Re-order states on open by heuristic merit (best leftmost)
   End;
Return FAIL                                                   %open is empty
End.
```

**% Best first search program**

```
domains
f=s(char,integer).
l=f*.
c=char.
i=integer.
predicates
best(l,l,c).
difference(l,l,l).
member(f,l).
append(l,l,l).
best_open(l,l,l).
set_best(f,l,l).
best_closed(l,l,l).
remove_worst(f,l,l).
best_children(l,l,l).
ignore_worst(f,l,l).
check(l,l,l,l,l).
sort(l,l).
min(l,f).
del(f,l,l).
sum(l,i).
print(l,l).
path(f,f).
clauses
best([],_,_):-!,write("The goal is not found").
best([s(G,H)|T],Closed,G):-!,
print([s(G,H)|T],Closed),
append(Closed,[s(G,H)],Path),
write("The goal is found &The resulted path is  ",Path),nl,
sum(Path,N),write("Total cost=",N),nl.
best([H|T_Open],Closed,G):-print([H|T_Open],Closed),
findall(X,path(H,X),Children),
check(Children,T_Open,Closed,Open1,Closed1),
sort(Open1,Open2),
append(Closed1,[H],Closed2),
best(Open2,Closed2,G).
check(Children,Open,Closed,New_Open,Closed):-
difference(Children,Open,X),
difference(X,Closed,Y),
Children=Y,!,%Chidren aren't in Open or in the Closed.
append(Children,Open,New_Open).%add children to the Open.
check(Children,Open,Closed,New_Open,Closed):-
difference(Children,Open,X),not(Children=X),!,%there is a Child or more in the Open.
best_open(Children,Open,Open1),%make the Open is the best by replace the
state by the best.
append(X,Open1,New_Open).%add dissimilar child to the Open.
check(Children,Open,Closed,New_Open,New_closed):-%there is Child or
more in the Closed.
best_closed(Children,Closed,New_closed),%make the Closed is the best by
delete the worst.
best_children(Closed,Children,Best_child),%make the Children is the best by
ignore the not best
append(Best_child,Open,New_Open).%add the pure children to the Open.
difference([],_,[]):- !.
difference([H|T],Z,[H|T1]):-not(member(H,Z)),!,
difference(T,Z,T1).
difference([_|T],Z,T1):-difference(T,Z,T1).
member(s(A,_),[s(A,_)|_]):-!.
member(H,[_|T]):-member(H,T).
append([],L,L):-!.
append([H|T],X,[H|T1]):-append(T,X,T1).
best_open([],Z,Z):-!.
best_open([X|T],Y,Z):-set_best(X,Y,Z1),
best_open(T,Z1,Z).
set_best(_,[],[]):-!.
set_best(s(A,X),[s(A,Y)|T],[s(A,X)|T]):-X<Y,!.
set_best(X,[H|T],[H|Z]):-set_best(X,T,Z).
best_closed([],Z,Z):-!.
best_closed([X|T],Y,Z):-remove_worst(X,Y,Z1),
best_closed(T,Z1,Z).
remove_worst(_,[],[]):-!.
remove_worst(s(A,X),[s(A,Y)|T],T):-Y>X,!.
remove_worst(X,[H|T],[H|Z]):-remove_worst(X,T,Z).
best_children([],Z,Z):-!.
best_children([X|T],Y,Z):-ignore_worst(X,Y,Z1),
best_children(T,Z1,Z).
ignore_worst(_,[],[]):-!.
ignore_worst(s(A,X),[s(A,Y)|T],T):-Y>=X,!.
ignore_worst(X,[H|T],[H|Z]):-ignore_worst(X,T,Z).
sort([],[]):-!.
sort(L,[M|T]):-min(L,M),
del(M,L,X),
sort(X,T).
min([M],M):-!.
min([s(A,X),s(_,Y)|T],M):-X<=Y,!,
min([s(A,X)|T],M).
min([_|T],M):-min(T,M).
del(X,[X|T],T):-!.
del(X,[H|T],[H|T1]):-del(X,T,T1).
sum([],0).
sum([s(_,H)|T],N):-sum(T,N1),N=N1+H.
print(Open,Closed):-write("Open=",Open,"  ","Closed=",Closed), nl.
path(s('a',0),s('b',4)).
path(s('a',0),s('c',5)).
path(s('a',0),s('d',3)).
path(s('b',4),s('e',3)).
path(s('b',4),s('c',1)).
path(s('d',3),s('c',2)).
path(s('d',3),s('f',5)).
path(s('c',1),s('g',3)).
path(s('c',5),s('g',3)).
path(s('c',2),s('g',3)).
/*
goal:best([s('a',0)],[],'e').
Open=[s('a',0)]    Closed=[]
Open=[s('d',3),s('b',4),s('c',5)] Closed=[s('a',0)]
Open=[s('c',2),s('b',4),s('f',5)] Closed=[s('a',0),s('d',3)]
Open=[s('g',3),s('b',4),s('f',5)] Closed=[s('a',0),s('d',3),s('c',2)]
Open=[s('b',4),s('f',5)]     Closed=[s('a',0),s('d',3),s('c',2),s('g',3)]
Open=[s('c',1),s('e',3),s('f',5)]   Closed=[s('a',0),s('d',3),s('g',3),s('b',4)]
Open=[s('e',3),s('f',5)]    Closed=[s('a',0),s('d',3),s('g',3),s('b',4),s('c',1)]
The goal is found &The resulted path is
[s('a',0),s('d',3),s('g',3),s('b',4),s('c',1),s('e',3)]
Total cost=14
yes */
```

**A-algorithm Search**

**%A-algorithm search program**

```
domains
f=s(char,integer,integer).
l=f*.
c=char.
i=integer.
predicates
a_algo(l,l,c).
difference(l,l,l).
member(f,l).
```

```
append(l,l,l).
best_open(l,l,l).
set_best(f,l,l).
best_closed(l,l,l).
remove_worst(f,l,l).
best_children(l,l,l).
ignore_worst(f,l,l).
check(l,l,l,l,l).
sort(l,l).
min(l,f).
del(f,l,l).
a_sum(l,l).
original_cost(l,l).
sum(l,i).
print(l,l).
path(f,f).
clauses
a_algo([],_,_):-!,write("The goal is not found").
a_algo([s(G,B,C)|T],Closed,G):-!,
print([s(G,B,C)|T],Closed),
append(Closed,[s(G,B,C)],Path),
original_cost(Path,Path1),  %represent the resulted path with the heuristic as it is in the path
write("The goal is found &The resulted path=",Path1),nl,
sum(Path1,N),write("Total cost=",N),nl.
a_algo([s(A,B,C)|T_Open],Closed,G):-%{B}represent the sum for each of the
heuristic value and
print([s(A,B,C)|T_Open],Closed),Q=B-C, %the generation value.{C}represent
the generation value.
findall(X,path(s(A,Q,C),X),Children),%{Q}represent the heuristic as it is in the path.
a_sum(Children,Children1),
check(Children1,T_Open,Closed,Open1,Closed1),
sort(Open1,Open2),
append(Closed1,[s(A,B,C)],Closed2),
a_algo(Open2,Closed2,G).
check(Children,Open,Closed,New_Open,Closed):-
difference(Children,Open,X),
difference(X,Closed,Y), Children=Y,!,
append(Children,Open,New_Open).
check(Children,Open,Closed,New_Open,Closed):-
difference(Children,Open,X),not(Children=X),!,
best_open(Children,Open,Open1),
append(X,Open1,New_Open).
check(Children,Open,Closed,New_Open,New_closed):-
best_closed(Children,Closed,New_closed),
best_children(Closed,Children,Best_children),
append(Best_children,Open,New_Open).
```

```
difference([],_,[]):- !.
difference([H|T],Z,[H|T1]):-not(member(H,Z)),!,
difference(T,Z,T1).
difference([_|T],Z,T1):-difference(T,Z,T1).
member(s(A,_,_),[s(A,_,_)|_]):-!.
member(H,[_|T]):-member(H,T).
append([],L,L):-!.
append([H|T],X,[H|T1]):-append(T,X,T1).
best_open([],Z,Z):-!.
best_open([X|T],Y,Z):-set_best(X,Y,Z1),
best_open(T,Z1,Z).
set_best(_,[],[]):-!.
set_best(s(A,B1,C),[s(A,B2,_)|T],[s(A,B1,C)|T]):-B1<B2,!.
set_best(X,[H|T],[H|Z]):-set_best(X,T,Z).
best_closed([],Z,Z):-!.
best_closed([X|T],Y,Z):-remove_worst(X,Y,Z1),
best_closed(T,Z1,Z).
remove_worst(_,[],[]):-!.
remove_worst(s(A,B1,_),[s(A,B2,_)|T],T):-B2>B1,!.
remove_worst(X,[H|T],[H|Z]):-remove_worst(X,T,Z).
best_children([],Z,Z):-!.
best_children([X|T],Y,Z):-ignore_worst(X,Y,Z1),
best_children(T,Z1,Z).
ignore_worst(_,[],[]):-!.
ignore_worst(s(A,B1,_),[s(A,B2,_)|T],T):-B2>=B1,!.
ignore_worst(X,[H|T],[H|Z]):-ignore_worst(X,T,Z).
sort([],[]):-!.
sort(L,[M|T]):-min(L,M),
del(M,L,X),
sort(X,T).
min([M],M):-!.
min([s(A,B1,C),s(_,B2,_)|T],M):-B1<=B2,!,
min([s(A,B1,C)|T],M).
min([_|T],M):-min(T,M).
del(X,[X|T],T):-!.
del(X,[H|T],[H|T1]):-del(X,T,T1).
a_sum([],[]):-!.
a_sum([s(A,B1,C)|T],[s(A,B2,C)|T1]):-B2=B1+C,  a_sum(T,T1).
original_cost([],[]):-!.
original_cost([s(A,B1,C)|T],[s(A,B2,C)|T1]):-  B2=B1-C,   original_cost(T,T1).
sum([],0).
sum([s(_,B,_)|T],N):-sum(T,N1),N=N1+B.
print(Open,Closed):-write("Open=",Open,"  ","Closed=",Closed),nl.
path(s('a',0,0),s('b',4,1)).
path(s('a',0,0),s('c',5,1)).
path(s('a',0,0),s('d',3,1)).
path(s('b',4,1),s('e',3,2)).
path(s('b',4,1),s('c',1,2)).
path(s('d',3,1),s('c',2,2)).
path(s('d',3,1),s('f',5,2)).
path(s('c',1,2),s('g',3,3)).
path(s('c',5,1),s('g',3,2)).
path(s('c',2,2),s('g',3,3)).
/*
goal:  a_algo([s('a',0,0)],[],'e').
Open=[s('a',0,0)] Closed=[]
Open=[s('d',4,1),s('b',5,1),s('c',6,1)] Closed=[s('a',0,0)]
Open=[s('c',4,2),s('b',5,1),s('f',7,2)] Closed=[s('a',0,0),s('d',4,1)]
Open=[s('b',5,1),s('g',6,3),s('f',7,2)] Closed=[s('a',0,0),s('d',4,1),s('c',4,2)]
Open=[s('c',3,2),s('e',5,2),s('g',6,3),s('f',7,2)]
Closed=[s('a',0,0),s('d',4,1),s('b',5,1)]
Open=[s('e',5,2),s('g',6,3),s('f',7,2)]
Closed=[s('a',0,0),s('d',4,1),s('b',5,1),s('c',3,2)]
The goal is found &The resulted
path=[s('a',0,0),s('d',3,1),s('b',4,1),s('c',1,2),s('e',3,2)]
Total cost=11
Yes */
```

**A Star Algorithm Search**

```
% A star algorithm search program
domains
f=s(char,integer,integer).
l=f*.
c=char.
i=integer.
predicates
a_star(l,l,c).
difference(l,l,l).
member(f,l).
append(l,l,l).
best_open(l,l,l).
set_best(f,l,l).
best_closed(l,l,l).
remove_worst(f,l,l).
best_children(l,l,l).
ignore_worst(f,l,l).
check(l,l,l,l,l).
sort(l,l).
min(l,f).
del(f,l,l).
a_sum(l,l).
```

```
original_cost(l,l).
sum(l,i).
print(l,l).
path(f,f).
clauses
a_star([],_,_):-!,write("The goal is not found").
a_star([s(G,B,C)|T],Closed,G):-!,
print([s(G,B,C)|T],Closed),
append(Closed,[s(G,B,C)],Path),
original_cost(Path,Path1),
write("The goal is found &The resulted path= ",Path1),nl,
sum(Path1,N),write("Total cost=",N),nl.
a_star([s(A,B,C)|T_Open],Closed,G):-print([s(A,B,C)|T_Open],Closed), Q=B-C,
findall(X,path(s(A,Q,C),X),Children),
a_sum(Children,Children1),
check(Children1,T_Open,Closed,Open1,Closed1),
sort(Open1,Open2),
append(Closed1,[s(A,B,C)],Closed2),
a_star(Open2,Closed2,G).
check(Children,Open,Closed,New_Open,Closed):-
difference(Children,Open,X),
difference(X,Closed,Y), Children=Y,!,
append(Children,Open,New_Open).
check(Children,Open,Closed,New_Open,Closed):-
difference(Children,Open,X),not(Children=X),!,
best_open(Children,Open,Open1),
append(X,Open1,New_Open).
check(Children,Open,Closed,New_Open,New_closed):-
best_closed(Children,Closed,New_closed),
best_children(Closed,Children,Best_children),
append(Best_children,Open,New_Open).
difference([],_,[]):- !.
difference([H|T],Z,[H|T1]):-not(member(H,Z)),!,
difference(T,Z,T1).
difference([_|T],Z,T1):-difference(T,Z,T1).
member(s(A,_,_),[s(A,_,_)|_]):-!.
member(H,[_|T]):-member(H,T).
append([],L,L):-!.
append([H|T],X,[H|T1]):-append(T,X,T1).
best_open([],Z,Z):-!.
best_open([X|T],Y,Z):-set_best(X,Y,Z1),
best_open(T,Z1,Z).
set_best(_,[],[]):-!.
set_best(s(A,B1,C),[s(A,B2,_)|T],[s(A,B1,C)|T]):-B1<B2,!.
set_best(X,[H|T],[H|Z]):-set_best(X,T,Z).
best_closed([],Z,Z):-!.
best_closed([X|T],Y,Z):-remove_worst(X,Y,Z1),
best_closed(T,Z1,Z).
remove_worst(_,[],[]):-!.
remove_worst(s(A,B1,_),[s(A,B2,_)|T],T):-B2>B1,!.
remove_worst(X,[H|T],[H|Z]):-remove_worst(X,T,Z).
best_children([],Z,Z):-!.
best_children([X|T],Y,Z):-ignore_worst(X,Y,Z1),
best_children(T,Z1,Z).
ignore_worst(_,[],[]):-!.
ignore_worst(s(A,B1,_),[s(A,B2,_)|T],T):-B2>=B1,!.
ignore_worst(X,[H|T],[H|Z]):-ignore_worst(X,T,Z).
sort([],[]):-!.
sort(L,[M|T]):-min(L,M),
del(M,L,X),   sort(X,T).
min([M],M):-!.
min([s(A,B1,C),s(_,B2,_)|T],M):-B1<=B2,!,
min([s(A,B1,C)|T],M).
min([_|T],M):-min(T,M).
del(X,[X|T],T):-!.
del(X,[H|T],[H|T1]):-del(X,T,T1).
a_sum([],[]):-!.
a_sum([s(A,B1,C)|T],[s(A,B2,C)|T1]):-B2=B1+C,
a_sum(T,T1).
original_cost([],[]):-!.
original_cost([s(A,B1,C)|T],[s(A,B2,C)|T1]):-B2=B1-C,
original_cost(T,T1).
sum([],0).
sum([s(_,B,_)|T],N):-sum(T,N1),N=N1+B.
print(Open,Closed):-write("Open=",Open,"  ","Closed=",Closed),nl.
path(s('a',0,2),s('b',4,1)).
path(s('a',0,2),s('c',5,3)).
path(s('a',0,2),s('d',3,3)).
path(s('b',4,1),s('e',3,0)).
path(s('b',4,1),s('c',1,2)).
path(s('d',3,3),s('c',2,4)).
path(s('d',3,3),s('f',5,4)).
path(s('c',1,2),s('g',3,3)).
path(s('c',5,3),s('g',3,4)).
path(s('c',2,4),s('g',3,5)).
/*
goal:a_star([s('a',2,2)],[],'e').
Open=[s('a',2,2)]  Closed=[]
Open=[s('b',5,1),s('d',6,3),s('c',8,3)]  Closed=[s('a',2,2)]
Open=[s('e',3,0),s('c',3,2),s('d',6,3)]  Closed=[s('a',2,2),s('b',5,1)]
The goal is found &The resulted path= [s('a',0,2),s('b',4,1),s('e',3,0)]
Total cost=7  */
```