Intelligent Search Techniques
3rd class / العمـــــــلي
Net, IS, CS Branches

أ.م. سُـرى محمـود عبـدالله

Email: - Sura.M.abdullah@uotechnology.edu.iq

Google Site: **Google scholar**: https://scholar.google.com/citations?user=ep015F0AAAAJ&hl=en

Research gate: https://www.researchgate.net/profile/Sura_Abdullah

Example6: *write a prolog program to append two lists in one list.*

Domains

L= integer*

Predicates

app(L,L,L)

Clauses

app([],L2,L2).

app([H|T1],L2,[H|T3]):-app(T1,L2,T3).

H	
3,	app([4,5],[6,7,5],[3]).
4,	app([5],[6,7,5],[3,4]).
5,	app([],[6,7,5],[3,4,5]).
	app([3,4,5],[6,7,5],[3,4,5,6,7,5]).

Goal

app([3,4,5],[6,7,5],L).

Output: L=[3,4,5,6,7,5]. 1 solution

Membership Function in List

- Member is possibly the most used user-defined predicate (i.e. you have to define it every time you want to use it).
- It checks to see if an element is in the list or not.

- it returns **yes** if it is.

- and returns **no** if it isn't.

Membership formula:

```
member(X,[X|_]).
member(X,[_|T]) :- member(X,T).
```

- It 1st checks if the first argument unifies with the Head of the list.

If yes then succeed.

If no then fail the first clause.

- The 2nd clause ignores the head of the list (which we know doesn't match) and recourses on the Tail.

write a prolog program to determine if a specific element in the list or not.

domains
i=integer
l=i*
predicates
member(i,l).
clauses
member(X,[X|_]):-!.
member(X,[_|T]):- member(X,T).

goal
member(4,[1,2,4,6,8]).
Output: yes

x
4=1 no, else member(4,[2,4,6,8]).
4=2 no, else member(4,[4,6,8]).
4=4 yes.

goal:
member(4,[1,2,6]).
Output: no

x
4=1 no, else member(4,[2,6]).
4=2 no, else member(4,[6]).
4=6 no, else member(4,[]).

write prolog program to find the difference between two list such as diff([1,2,3],[3,4,5],L), L=[1,2].

```
domains
i=integer
l=i*
predicates
diff(l,l,l).
member(i,l).
clauses
diff([],_,[]):-!.
diff([H|T],Y,[H|T1]):-not(member(H,Y)),diff(T,Y,T1),!.
diff([_|T],Y,Z):-diff(T,Y,Z).
member(X,[X|_]):-!.
member(X,[_|T]):-member(X,T).
```

goal: diff([1,2,3],[3,4,5],L).
output: L=[1,2]

Breadth-First Search Algorithm

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

path('a','b').

path('a','c').

path('a','d').

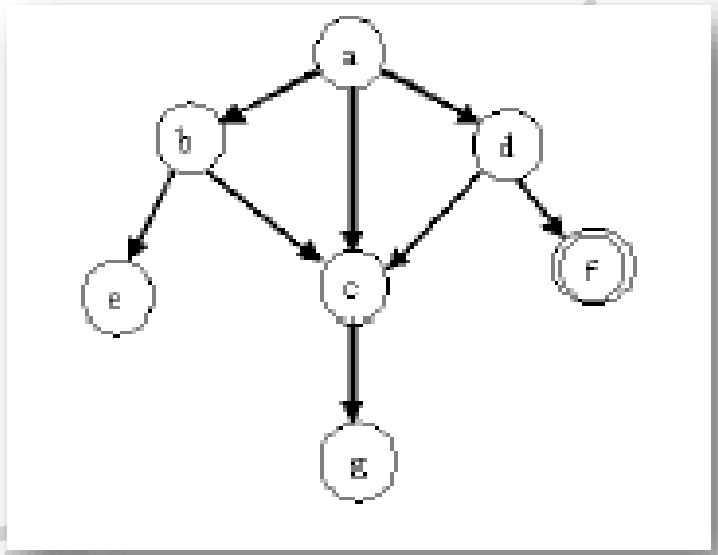
path('b','e').

path('b','c').

path('d','c').

path('d','f').

path('c','g').



```
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),
                               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.
```

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

Depth-First Search Algorithm

Domains

c=char.

l=c*.

Predicates

depth(l,l,c).

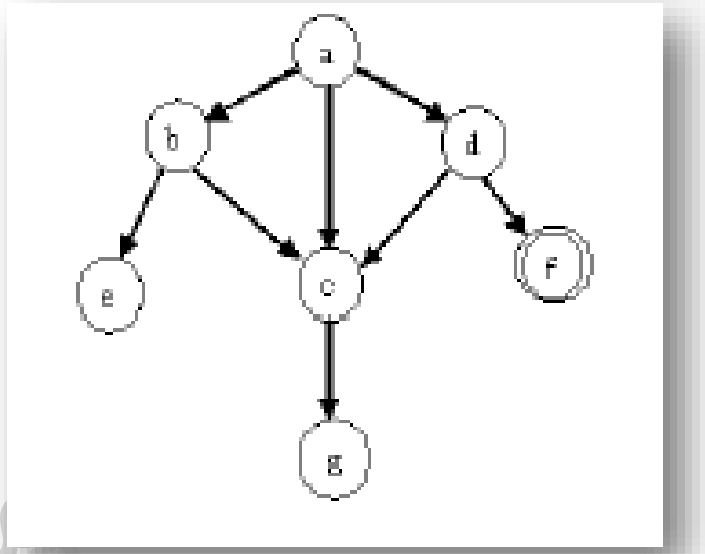
difference(l,l,l).

append(l,l,l).

member(c,l).

print(l,l).

path(c,c).



Clauses

path('a','b').

path('a','c').

path('a','d').

path('b','e').

path('b','c').

path('d','c').

path('d','f').

path('c','g').

```
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),
                              findall(X,path(H,X),Children),
                              append(Closed,[H],Closed1),
                             difference(Children,T_Open,Children1),
                             difference(Children1,Closed1,Children2),
                             append(Children2,T_Open,Open1),
                             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.
```

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

Example: *write a prolog program to find the minimum number in the list.*

Domains

i=integer

l= i*

Predicates

min(l,i,i)

Clauses

min([],M,M).

min([H|T],X,M):-H<X,!, min(T,H,M).

min([_|T],X,M):- min(T,X,M).

H		X			
3	<	15,	yes		max([9,4,5],3,M).
9	<	3,	no		max([4,5],3,M).
4	<	3,	no	else	max([5],3,M).
5	<	3	no	else	max([],3,M).

Goal

min([3,9,4,5],15,M).

Output: M=3. 1 solution

Example: *write a prolog program to delete a specific element from list.*

domains
i=integer
l=i*
predicates
del(i,l,l).
clauses
del(_,[],[]):-!.
del(X,[H|T],L):- X=H, !,del(X,T,L).
del(X,[H|T],[H|T1]):-del(X,T,T1).

X		H	
3	=	1 no, else	del(3,[3,4,3],[1]).
3	=	3 yes,	del(3,[4,3],[1]).
3	=	4 no, else	del(3,[3],[1,4]).
3	=	3 yes,	del(3,[],[1,4]).

Goal
del(3,[1,3,4,3],L).
output: L=[1,4]. 1 solution

Example: *write a prolog program to sort the list elements in ascending order.*

```
domains
i=integer
l=i*
predicates
min(l, i, i)
del(i, l, l)
sort(l, l)
clauses
sort([],[]).
sort(L,[M|T]):-min(L,100,M),del(M,L,Z),sort(Z,T),!.
min([],M,M).
min([H|T],X,M):-H<X, min(T,H,M),!.
min([_|T],X,M):-min(T,X,M).
del(_,[],[]).
del(X,[H|T],L):-X=H ,del(X,T,L).
del(X,[H|T],[H|T1]):-del(X,T,T1),!.
```

Goal: sort ([5,4,9,7,3],L).

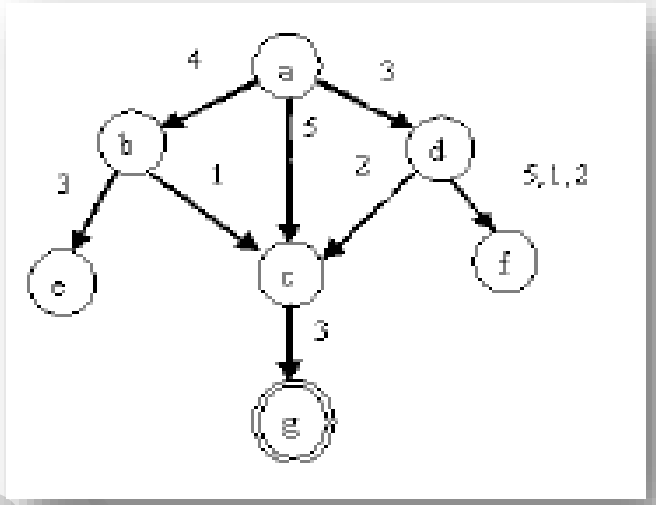
Output: L=[3,4,5,7,9]

Hill Climbing Algorithm

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
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)).



```
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("Search is stopped because there are equal costs for the
states= ",S1,"&",S2), nl.

print(Open,Closed):-  write("Open=",Open," ","Closed=",Closed),nl.
```

/* 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

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).

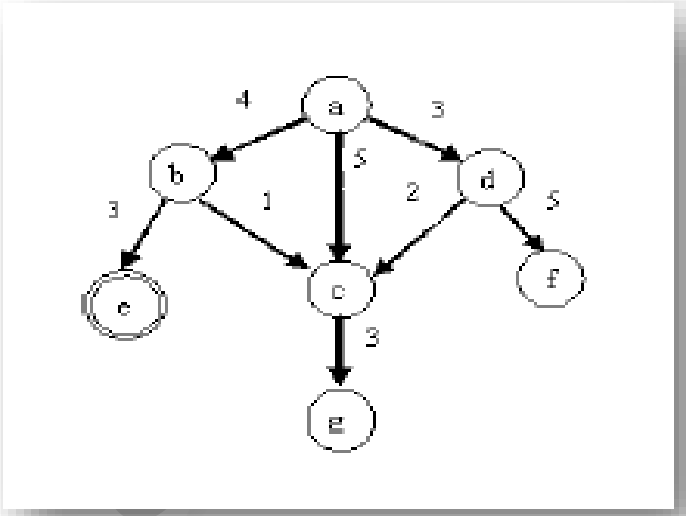
min(l,f).

del(f,l,l).

sum(l,i).

print(l,l).

path(f,f).



clauses

```
 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)).
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,!,
                                               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_child),
                                            append(Best_child,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,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.
```

/* 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 Search Algorithm

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).

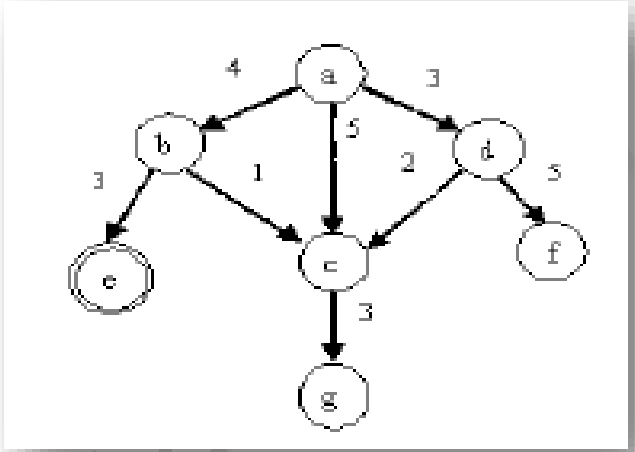
a_sum(l,l).

original_cost(l,l).

sum(l,i).

print(l,l).

path(f,f).



clauses

```
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)).

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),
                                  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):- 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_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.
```

/* 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 ***/**