الجامعة التكنولوجية

قسم علوم الحاسوب / فرع الذكاء الاصطناعي

المرحلة الثانية - الكورس الثاني2024- 2025

مادة طرق بحث موجه / منهاج المادة العملي

م..د. ندى حسين علي

ت	اسم البرنامج	اسم الخوارزمية
1	Hill Climbing Search	Hill Climbing Search
2	Best First Search	Best First Search
3	A-algorithm Search	A-algorithm Search
4	A star algorithm search program	A star algorithm search program
5	Text Recognition System (An AI Program for Psychology Counseling)	
6	The Chemical Synthesis System	The Chemical Synthesis System
7	Student Advisor System	Student Advisor System

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
Algorithm of A star Search Method

```
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
         Begin
          Add the distance between current state to goal state to the heuristic value
for each child                                              %make the g(n)
        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.
```

% 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  */
```

Text Recognition System (An AI Program for Psychology Counseling)

Pattern in Sentence

From human	**Machine (Computer) Responds**
…..I fell…..	How long have you felt that way?
….. father …..	Tell me more about your family.
…..friendly…..	Better friends help you when you need.
…..hell…..	Please do not use words like that.
…..yes or no….	Please be more explicit.
…..high performance…..	This gives you additional respecting.
…..sadness or anger…..	How are you feeling right now?

…..complex or fixation….. Too many mind games

……..father……. Earlier you mentioned your father.

…….. …….

```
clauses
helpme :-
        write("Speak up, what is your problem?"), nl,
        repeat,
        getclause(L),
        makeans(L),
        fail.

getclause(L) :-
                readln(S),
                str_to_list(S,L).

makeans(L) :-
                recognize(L,1),
                write ("How long have you felt that way?"), nl, !.
makeans(L) :-
                recognize(L,2),
                write ("Tell me more about your family"), nl, !.
makeans(L) :-
                recognize(L,3),
                write ("better friends help you when you need."), nl, !.
makeans(L) :-
                recognize(L,4),
```

```
        write ("Please do not use words like that."), nl, !.
makeans(L) :-
        recognize(L,5),
        write ("Please be more explicit."), nl, !.
makeans(L) :-
        recognize(L,6),
        write ("This gives you additional respecting."), nl, !.
makeans(L) :-
        recognize(L, 7),
        write ("How are you felling right now?"), nl, !.
makeans(L) :-
        recognize(L,8),
        write ("Too many mind games."), nl, !.
makeans(L) :-
        recognize(L,9),
        write ("Earlier you mentioned your father."), nl, !.


makeans(L) :-
        recognize(L,10), write ("Tell me more."), nl, !.

recognize(L, 1) :- contains([i, feel], L).
recognize(L, 2) :- contains([father], L) assert(father).
recognize(L, 3) :- contains([friendly], L)
recognize(L, 4) :- contains([hell], L).
recognize(L, 5) :- L= [yes]; L=[no].
recognize(L, 6) :- contains([high, performance], L).
```

```
recognize(L, 7) :- contains([sadness], L); contains([anger], L).
recognize(L, 8) :- contains([complex], L) ; contains([fixation], L).
recognize(L, 9) :- father.
recognize(_, 10).
```

Systems Based on Heuristic Search and Pattern Recognition (2)

The Chemical Synthesis System

```
domains
rxnlist = reactions*.
reactions = rxn(symbol, ls, integer, integer).
ls = symbol*.
chemicalList= chemicalForm*.
chemicalForm= chemical(symbol, rxnList, integer, integer).
Li= integer*.

predicates
rxn(symbol, ls, integer, integer).
rawmaterial(symbol, integer, integer).
chemical(symbol, rxnlist, integer, integer).
all_chemical(symbol, chemicalList).
best_chemical(symbol, chemicalForm).
one_chemical(symbol, chemicalForm).
append(rxnlist, rxnlist, rxnlist).
min(chemicalList, chemicalForm).
run(symbol).
```

```
clauses
rxn(a, [b1, c1], 12, 60).
rxn(b1, [d1, e1], 5, 45).
rxn(c1, [f1, g1], 3, 15).


rxn(a, [b2, c2], 10, 50).
rxn(b2, [d2, e2], 2, 20).
rxn(c2, [f2, g2], 6, 30).

rawmaterial(d1, 2, 0).
rawmaterial(e1, 0, 0).
rawmaterial(f1, 2, 0).
rawmaterial(g1, 0, 0).

rawmaterial(d2, 0, 0).
rawmaterial(e2, 1, 0).
rawmaterial(f2, 1, 0).
rawmaterial(g2, 0, 0).

chemical(Y, [], Cost, Time):- rawmaterial(Y, Cost, Time).
chemical(Y, L, Ct, T):-
        rxn(Y, [X1, X2], C, T1),
        chemical(X1, L1, C1, T2),
        chemical(X2, L2, C2, T3),
        append(L1, L2, Q),
        Ct = C+C1+C2,
```

```
        T = T+T2+T3,
        append([rxn(Y, [X1, X2], C, T1)], Q, L).

best_chemical(Y, M):- all_chemical(Y, X), min(X, M).

all_ chemical(Y, X):- findall(S, one_chemical(Y, S), X).

one_chemical(Y, chemical(Y, L, Ct, T)):- chemical(Y, L, Ct, T).

append([], L, L):-!.
append([H|T], L, [H|T1]):- append(T, L, T1).

min([chemical(Y, L, Ct, T)], chemical(Y, L, Ct, T)).
min([chemical(Y, L, Ct, Time)|T], chemical(Y, L, Ct, Time)):-
                min(T, chemical(Y1, L1, C1, Time1)), Ct <= C1.
min([chemical(Y, L, Ct, Time)|T], chemical(Y, L2, Ct2, Time2)):-
                min(T, chemical(Y, L2, Ct2, Time2)), Ct2 <= Ct.

run(X):- write(" chemical synthesis is:"), nl, chemical(X, L, Cost, Time),
        write(L, "\n with total cost =", Cost, " Time =", Time), nl, fail.
run(X):- write("\n Best chemical synthesis:"), nl, best_chemical(X, Y),
write(Y), nl.

Goal: run(a).
chemical synthesis:
[rxn("a", ["b1", "c1"], 12, 60), rxn("b1", ["d1", "e1"], 5, 45), rxn("c1",
["f1", "g1"], 3, 15)]
```

with total cost = 24 time = 120

[rxn(“a”, [“b2”, “c2”], 10, 50), rxn(“b2”, [“d2”, “e2”], 2, 20), rxn(“c2”, [“f2”, “g2”], 6, 30)]

with total cost = 20 time = 100

best chemical synthesis :

chemical(“a”, [rxn(“a”, [“b2”, “c2”], 10, 50) rxn(“b2”, [“d2”, “e2”], 2, 20), rxn(“c2”, [“f2”, “g2”], 6, 30)], 20, 100

Search with Heuristic Embedded in Rule

Student Advisor System

/* Set of Facts */

given_now(logic design).

given_now(Mathematics).

given_now(prolog language).

given_now(computation theory).

given_now(data structure).

given_now(artificial intelligence).

given_now(expert systems).

given_now(computation theory).

given_now(computer architecture).

.

.

.

required(prolog).
required(logic design).
required(artificial intelligence).
required(expert systems).
required(machine learning).
required(data structure).
required(c++).
.
.
.

elective(computer graphics).
elective(object oriented programming).
elective(data security).
elective(web programming).
elective(operations researches).
.
.
.

waived(digital signal processing).
waived(image processing).
waived(information systems principles).
waived(software engineering).
waived(data hiding).
.
.
.

impreq(object oriented programming, c++).

```
impreq(prolog language, logic design).
impreq(artificial intelligence, prolog language).
impreq(expert systems, artificial intelligence).
impreq(computer architecture, logic design).
impreq(data structure, c++).
.
.
.

passed(logic design).
passed(prolog language).
passed(artificial intelligence).
passed(mathematics).
passed(data structure).
passed(c++).
passed(computation theory).
passed(computer organization).
.
.
.


pos_req_course(X) :-
                        required(X),
                        given_now(X),
                        not(done_with(X)),
                        have_preq_for(X).
```

```
pos_elec_course(X) :-
                    elective(X),
                    given_now(X),
                    not(done_with(X)),
                    have_preq_for(X).


done_with(X) :- waived(X).
done_with(X) :- passed(X).

all_done_with(L) :- findall(X, done_with(X), L).

have_preq_for(X) :-
                    all_preq_for(X, Z),
                    all_done_with(Q),
                    subset(Z, Q).

all_preq_for(X, Z):-
                    findall(Y, preq(X, Y), Z).

preq(X,Y):- impreq(X, Y).
preq(X, Y):- impreq(X, W), preq(W, Y).
```