Diss/xx/25/187

M. Tech. (Computer Science) Dissertation Series

A MODIFICATION OF THE DYNAMIC PROGRAMMING SEARCH WITH HEURISTIC INFORMATION

A dissertation submitted in partial fulfillment of the requirements for the M. Tech. (Computer Science) degree of the Indian Statistical Institute

By

Sk. Baruzzaman

Under the supervision of

Dr. K.S. Ray

INDIAN STATISTICAL INSTITUTE
203, B.T. Road, Cal -108

ACKNOWLEDGEMENT

I wish to express my deep sense of gratitude to Dr. K.S. Ray for his encouragement and guidance throughout this dissertation work.

Calcutta
Date - 27.7.88

Sk. Badruzzaman.
(Sk. Badruzzaman)

INTRODUCTION

The problem of finding an optimal solution from a large set of feasible solutions has always been a difficult and tricky problem to solve. Most problems of real life interest have large number of feasible solutions which usually increase in problem size. Various methods have been developed to tackle the situation. The methods covered in this study are

i) Dynamic Programming Search

ii) A* Search.

Every search process can be viewed as a traversar of a directed graph in which each node represents a problem state and each arc represents a relationship between the states represented by the nodes it connects. The search process must find a path through the graph, starting at an initial state and ending in one or more final states.

The object of a search procedure is to discover a path from initial state to a goal state. There are two directions in which a search could proceed -

Forward, from the start states.

Backward, from the goal states.

CHAPTER - 1

REVIEW OF THE EXISTING DYNAMIC PROGRAMMING SEARCH -

Both the algorithms, dynamic programming and A* are based on stagewise search method. It will be worthwhile to mention at this juncture that the problem is considered a general network problem with a start node and a goal node and a large number of possible paths from the start node to goal node. Associative with each path, is a cost and the objective is to find the path with minimum associative cost. At any iteration it is possible to move to a finite number of nodes from a given node with some associative cost.

For backward search procedure, it starts from gool node and goes in a backward direction to the start node. At every iteration the optimal state within a stage is found. Thus successive steps were taken on the currently selected node thereby preserving the optimality throughout the procedure. This method of search is basically a variation of the branch and bound technique.

Similarly the forward search procedure starts from start node and goes in a forward direction to the goal node.

CHAPTER - II

MODIFIED DYNAMIC PROGRAMMING SEARCH FOR PRUNING -

According to Nilsson (1) Dynamic Programming Search is basically a breadth-first search.

The uniform search methods, whether breadth-first or depth-first, are exhaustive methods for finding path to a goal node. In principle these methods provide a solution to the path-finding problem, but they are often infeasible to use because the search expands too many nodes before a path is found. Since there are always practical limits on the amount of time and storage available to expand on the search, more efficient alternatives to uniform search are based on pruning technique to help reduce search.

This pruning scheme is based on some upper bound of the critical path and this gives rise to a simple scheme of cutting down on search space. If a state is pruned at any particular stage it is pruned forever. Thus it offers a much reduction in the search space for all other subsequent stages.

Pruning Criteria - Let G be a directed graph. S be the starting node of the graph. Let $f^*(n)$ at any node $n \in G$ is the actual cost of an optimal path from node S to node n plus the cost of an optimal path from node n to goal node, that is,

$$f^*(n) = g^*(n) + h^*(n)$$

Thus the value $f^*(n)$ is the cost of an optimal path from S to goal node constrained to go through node n. Let f, the evaluation function, be an estimate of f^*. Then f can be written as

$$f(n) = g^*(n) + h(n)$$

where $h(n)$ is the reducing estimate of $h^*(n)$. For estimate $h(n)$ of $h^*(n)$ we rely on heuristic information from the problem domain. h is the lower bound of h^* (ie. $h(n) \leq h^*(n)$ for all nodes n) for admissible search. Let W be the upper bound of the optimal cost path of the graph G. The numerical value of W can be obtained either from the heuristic information of the problem domain or from any feasible solution, obtained by some means (e.g. by a depth-first search) of the directed graph G. Now we stipulate the following definitions.

<u>Definition 1</u> - A path through a particular state n is said to be nonpromising for optimal search if the $f(n)$ value is higher than W.

<u>Definition 2</u> - A nonterminal sate is said to be an isolated state, if the state has no successor through which a path can be generated upto the terminal state.

If for any state n, $f(n) > W$, that state is marked as a possible candidate state for permanent pruning. For isolated state it is assumed that $h(n) = \infty$. Hence search through isolated state never terminate. So we always ignore the isolated state from the list of search.

The following are some important results for permanent pruning.

<u>Lemma 1</u> - If for any node n, $g^*(n) + h(n) > W$ that state is permanently pruned from graph G. Otherwise that unpruned state may be found by other routes and retained unnecessarily in the subsequent list of search.

<u>Lemma 2</u> - Any succeding node $(n+1)$ of n can also be pruned permanently from the list of search provided the optimal path constrained through $(n+1)$ is also constrained through n.

<u>Proof</u> -

<u>Case 1</u> - When the optimal path constrained through n is same as the optimal path constrained through $(n+1)$

Consider $f^*(n+1) = g^*(n+1) + h^*(n+1)$

$$= g^*(n)+k^*(n,n+1) + h^*(n+1)$$

$$= g^*(n) + h^*(n)$$

where $k^*(n,n+1)$ is the cost of the optimal path between n and $n+1$.

Since $h(n) \leq h^*(n)$, $f^*(n+1) > W$ (by lemma 1).

Case 2 - When the optimal path constrained through n is not same as the optimal path constrained through n and n+1

Consider $f^*(n+1) = g^*(n+1) + h^*(n+1)$

$$= g^*(n) + k^*(n,n+1) + h^*(n+1)$$

$$= g^*(n) + h'(n)$$

where $h'(n)$ is the cost of an non-optimal path from n to any goal node and hence either $h'(n) > h^*(n)$ (if there exists a single optimal path from state n to any goal state)or $h'(n) \geq h^*(n)$ (if there exists more than one optimal path from state n to goal state).

Since $h(n) \leq h^*(n) \leq h'(n)$, therefore $f^*(n+1) > W$ (by lemma 1).

Corollary - Any succeding node (n+1) of n can be pruned permanently provided the optimal path from S to (n+1) is constrained through n.

For final decision of pruning the following steps are considered .

<u>Step 1</u> - Start from the start node S. The successors of S are kept in OPEN set.

<u>Step 2</u> - Those steps of the set OPEN are marked non-promising and pruned permanently whose $f(n) > W$.

<u>Step 3</u> - Among the promising states of the set OPEN, a particular state, whose $f(n)$ value is smallest, is selected for the next iteration. In case of a tie situation an arbitrary decision can be taken.

<u>Step 4</u> - A pointer is appropriately directed towards the father node.

<u>Step 5</u> - The set OPEN (ignoring the permanently pruned states) is reestablished for further iteration.

The result of lemma 1 gives the reduction in search space at any particular iteration based upon the decision of permanent pruning at all other previous iterations and the result of lemma 2 gives the facility of pruning a state before it is searched.

CHAPTER - III

HEURISTIC DYNAMIC PROGRAMMING SEARCH AND COMPARISON WITH A* TECHNIQUE -

Heuristic Function - A heuristic function is a function that maps from problem state descriptions to measure of desirability, usually represented as numbers. Which aspects of the problem state are considered, how those aspects are evaluated, and the weights given to individual aspects are chosen in such a way the value of the heuristic function at a given node in the search process gives as good an estimate as possible of whether that node is on the desired path to a solution.

The following shows some simple heuristic functions for a few problems.

TRAVELLING SALESMAN	the sum of the distances so far.
CHESS	the material advantage of our side over the opponent.
8-PUZZLE	the number of tiles that are in the place the belong.

<u>Heuristic Search</u> - For many tasks it is possible to use task-dependant information to help reduce search. Information of this sort is usually called heuristic information, and search procedures using it are called heuristic search methods. It is often possible to specify heuristics that reduce search effort without sacrificing the guarantee of finding the minimal cost paths. In most practical problems, we are interested in minimising some combination of the cost of the path and the cost of the search required to obtained the path. Further more, we are interested in search methods that minimize this combination. If the averaged combination cost of search method 1 is lower than the averaged combination cost of search method 2, then search method 1 is said to have more heuristic power than search method 2.

Averaged combination costs are never actually computed, both because it is difficult to decide on the way to combine path cost and search effort cost and because it would be difficult to define a probability distribution over the set of problems to be encountered.

Therefore the matter of deciding whether one search method has more heuristic power than another is usually left to informed intuition, gained from actual experience with the methods.

Staged Search With Heuristic Information -

The use of heuristic information as discussed so far can substantially reduce the amount of search effort required to find acceptable paths. Its use, therefore, also allows much larger graphs to be searched than would be the case otherwise.

The search process continues in stages, punchuating by pruning operations. At the end of each stage, some of the nodes in OPEN, for example these having the smallest values of f, are marked for retention. The best path to these nodes are remembered. The process continues until either a goal node is found or until resources are exhausted.

Comparison of Heuristic Dynamic Programming Search With A* Technique -

The basic difference between the A* search and modified dynamic programming search (modification due to the introduction of the reducing heuristic) is that at any particular node n of the given graph G the evaluation function is represented as

i) in case of A* search

$$f(n) = g(n) + h(n)$$

where $g(n) > g^*(n)$ and $h(n) \leq h^*(n)$

ii) in case of modified dynamic programming search,

$$f(n) = g^*(n) + h(n)$$

where $h(n) \leq h^*(n)$

It can easily be proved (1) that both the search techniques ensure convergence and admissibility.

CHAPTER - IV

RESULTS AND DISCUSSION -

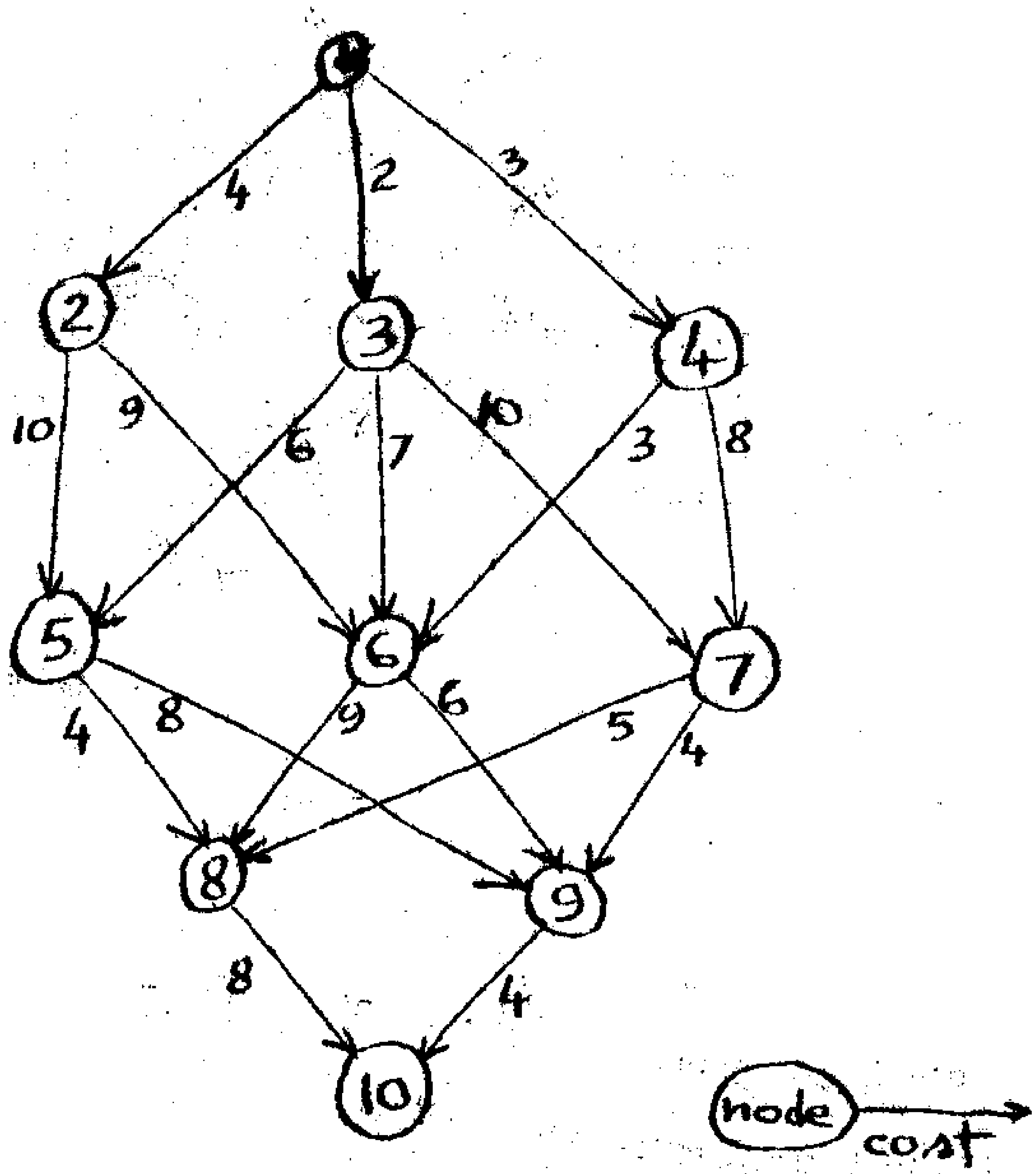
Dynamic programming is a very useful technique for making an appropriate recursive relationship for each individual problem. However, it provides a great computational savings over using exhaustive enumeration to find the best combination of decisions, especially for large problems, for example, if a problem has 10 stages and 10 possible decesions at each stage, then exhaustive enumeration must consider upto 10^{10} combinations, whereas dynamic programming need make no more than 10^3 calculations. Moreover the pruning criteria which we have discussed earlier reduces the search space as well as the computations involved in solving a problem. The result of pruning can be seen in the output section.

It should be clear that the time to solve a problem with dynamic programming is linear with respect to the number of stages, since the amount of time to solve the subproblem at each stage is relatively constant for all stages. If it takes 0.5 sec. of to do the calculations at each stage I and if there are 10 stages, then the total computation time will be approximately 5 sec. On the other hand, 30 stages would take approximately15 sec. Thus the computation time using the dynamic programming is linear with respect to the number of stages, while the computation time for enumeration increases exponentially with the number of stages.

Although dynamic programming can be used to determine the optimal solution for a large variety of problems, it is by no means the most efficient method to use in all cases. Experimence and ingenuity are the guiding factors in determing when to use dynamic programming.

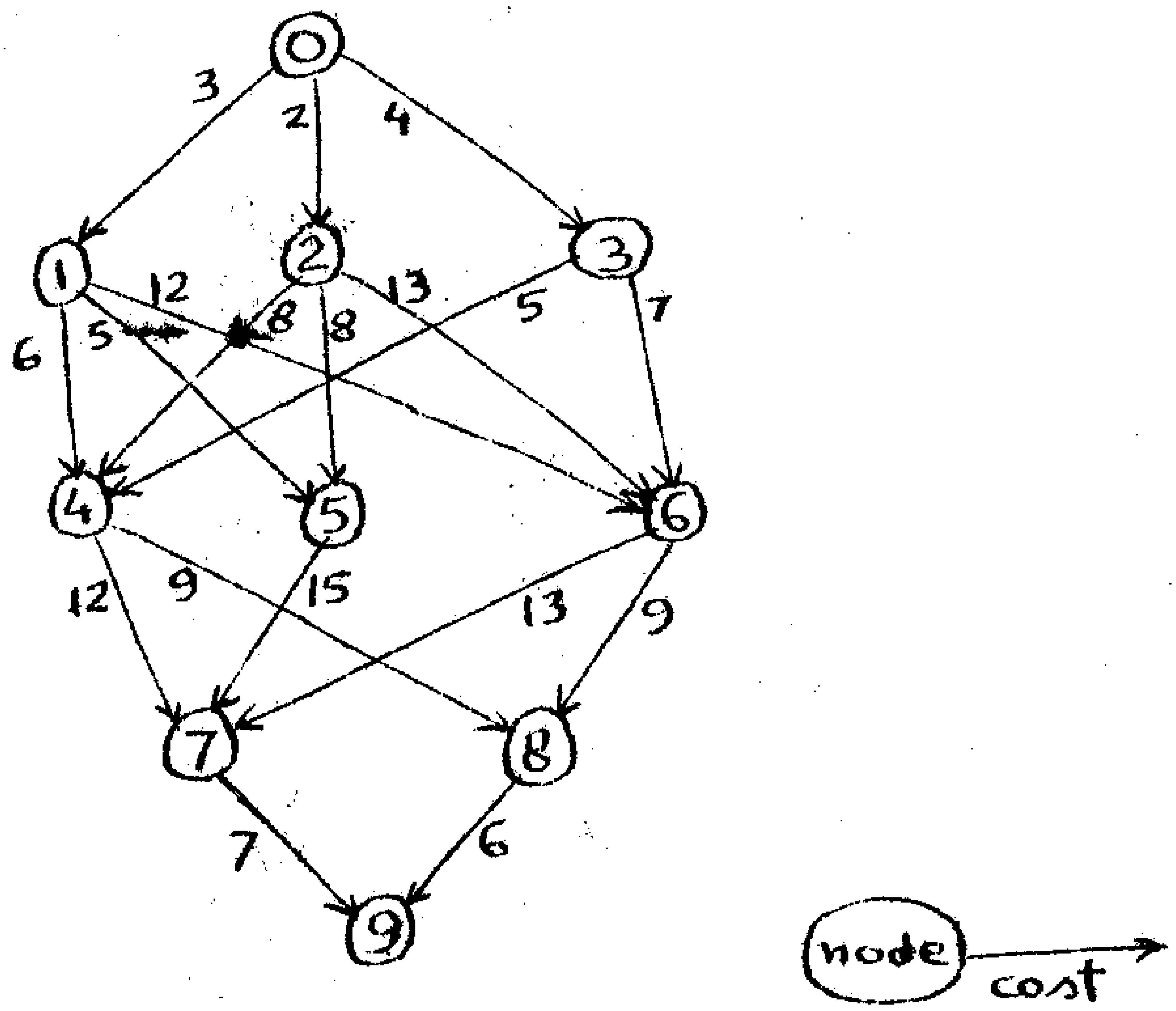
Some Example Problems -

Example 1 -



The dynamic programming procedure used to solve the above stage coach problem was the backward approach. That is we started at stage N and backed through the various stages until finally stage 1 was reached. As we backed through the stages, a number of possible decisions we examined at each stage. The solution can be seen in the out put section.

Example 2 -



This problem is solved by modified dynamic programming technique with pruning. The solution steps can be seen in the output section.

REFERENCES

1. Nilsson, N.J., Principles of Artificial Intelligence.

2. Rich, Elaine, Artificial Intelligence.

3. Gillett, B.E., Introduction to Operations Research.

```
 STAGECOACH PROBLEM WITH  5 STAGES

FOLLOWING ARE THE STATES IN DIFFERENT STAGES
S(i,j)  means  j th state in stage i

    J-->
S(1,j)=    1

S(2,j)=    2    3    4

S(3,j)=    5    6    7

S(4,j)=    8    9

S(5,j)=   10


FOLLOWING ARE THE COST
C(i,j,k)  means the cost from  S(i,j)  to  S(i+1,k)


      K-->
C(1,1,k)=      3      2      4

C(2,1,k)=      6      5     12

C(2,2,k)=      8      8     13

C(2,3,k)=      5     ---     7

C(3,1,k)=     12      9

C(3,2,k)=     15     ---

C(3,3,k)=     13      9

C(4,1,k)=      7

C(4,2,k)=      6

FOLLOWING ARE THE HEURISTICS ASSOCIATED WITH EACH NODE

h(1) = 25
h(2) = 23
h(3) = 25
h(4) = 22
h(5) = 13
h(6) = 20
h(7) = 13
h(8) = 5
h(9) = 5
h(10) = 0


             state
```

h(8) = 5
h(9) = 5
h(10) = 0

	state developed	g(n)	f(n)	father(n)	OPEN set : n(g(n),f(n))
iteration- 1)	1	0	32767	---	2(3,26) 3(2,27) 4(4,26)
iteration- 2)	2	3	26	1	3(2,27) 4(4,26) 5(9,22) 6(8,28) 7(15,28)
iteration- 3)	5	9	22	2	3(2,27) 4(4,26) 6(8,28) 7(15,28) 8(21,26) 9(18,23)
iteration- 4)	9	18	23	5	3(2,27) 4(4,26) 6(8,28) 7(15,28) 8(21,26) 10(24,24)

In stage 2 state 2 i.e. state number 3 is pruned

In stage 2 state 3 i.e. state number 4 is pruned

In stage 3 state 2 i.e. state number 6 is pruned

In stage 3 state 3 i.e. state number 7 is pruned

In stage 4 state 1 i.e. state number 8 is pruned

THE OPTIMAL PATH OBTAINED IS AS FOLLOWS

1-2-5-9-10-

THE OPTIMAL COST IS =24

START NODE = 0

TOTAL NUMBER OF GOAL NODE = 2
GOAL NODE(1) =15
GOAL NODE(2) =16

	state developed	g(n)	f(n)	father(n)	OPEN set : n(g(n),f(n))
iteration- 1)	0	0	17	---	1(4,19) 3(5,20) 5(2,17)
iteration- 2)	5	2	17	0	1(4,19) 3(5,20) 7(4,23) 8(7,10006)
iteration- 3)	1	4	19	0	2(5,23) 3(5,20) 4(9,19) 7(4,23) 8(7,10006)
iteration- 4)	4	9	19	1	2(5,23) 3(5,20) 6(10,19) 7(4,23) 8(7,10006) 9(13,23)
iteration- 5)	6	10	19	4	2(5,23) 3(5,20) 7(4,23) 8(7,10006) 9(13,23) 10(38,44) 11(12,26) 13(20,22)
iteration- 6)	3	5	20	0	2(5,23) 6(7,16) 7(4,23) 8(7,10006) 9(13,23) 10(38,44) 11(12,26) 12(18,24) 13(20,22)
iteration- 7)	6	7	16	3	2(5,23) 7(4,23) 8(7,10006) 9(13,23) 10(35,41) 11(9,23) 12(18,24) 13(17,19)
iteration- 8)	13	17	19	6	2(5,23) 7(4,23) 8(7,10006) 9(13,23) 10(35,41) 11(9,23) 12(18,24) 15(25,25) 16(21,21)

THE OPTIMAL PATH OBTAINED IS AS FOLLOWS

0-3-6-13-16-

THE OPTIMAL COST IS =21

TOTAL NUMBER OF NODES IN THE NETWORK = 17

THE FOLLOWING IS THE PATH MATRIX OF THE NETWORK WITH ASSOCIATED COSTS

0	4	0	5	0	2	0	0	0	0	0	0	0	0	0	0	0
0	0	1	3	5	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	8	0	2	0	0	0	0	0	20	0	0	0
0	0	16	0	0	0	2	15	8	14	0	10	13	0	0	0	0
0	0	0	0	0	0	1	0	0	4	0	0	0	0	0	0	0
0	0	0	4	0	0	0	2	5	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	9	6	28	2	0	10	0	0	0
3	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	2	5	0	0	12	0
0	0	0	0	0	2	0	0	0	0	0	0	1	6	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	9
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	4
0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	9	10
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

THE HEURISTIC ASSOCIATED WITH EACH NODE ARE AS FOLLOWS

h(0) =17
h(1) =15
h(2) =18
h(3) =15
h(4) =10
h(5) =15
h(6) =9
h(7) =19
h(8) =9999
h(9) =10
h(10) =6
h(11) =14
h(12) =6
h(13) =2
h(14) =7
h(15) =0
h(16) =0

```
          if g[s,i] > 0
            then begin
                    OPEN:=OPEN + [i];
                    new_g:=latest_g[s] + g[s,i];
                    new_f:=new_g + h[i];
                    if new_f < f[i]
                      then begin
                               f[i]:=new_f;
                               latest_g[i]:=new_g;
                               father[i] :=s;
                           end;
                 end;
       end;
       if iteration = 1
         then begin
                 write(out,'iteration-',iteration:2,')',s:5,latest_g[s]:10,f[s]:8);
                 write(out,'---':9);
                 write(out,'       ');
              end
         else begin
                 write(out,'iteration-',iteration:2,')',s:5,latest_g[s]:10,f[s]:8,father[s]:9);
                 write(out,'       ');
              end;
       count  := 0;
       for i:= 0 to n-1 do
       if i in OPEN
          then begin
                  count := count +1;
                  write(out,i:2,'(',latest_g[i],',',f[i],') ');
                  if count > 5 then
                    begin
                       count :=0;
                       writeln(out);
                       write(out,' ':52);
                    end;
               end;
       writeln(out);  writeln(out);  writeln(out);
       finding_minimum_of_fn;
       s:=index;
       OPEN:=OPEN -[s];
    until s in GOAL_NODE_SET;
    writeln(out);
    writeln(out);
    writeln(out,'______________________________________________________________________');
    writeln(out);
    writeln(out);
    writeln(out);
    {   FINDING OPTIMAL PATH    }
    x[1] :=s;
    count :=1;
    repeat
      count := count +1;
      x[count] :=father[x[count-1]];
    until  x[count] = start_node;
    writeln(out,' THE OPTIMAL PATH OBTAINED IS AS FOLLOWS  ');
    writeln(out);
    for i:= count downto 1 do write(out,x[i],'-');
    writeln(out);
    writeln(out);
    writeln(out,'THE OPTIMAL COST IS =',f[s]);

END.(*--------MAIN PROGRAM-----------*)
```

```
{$u+}
{$R+}
PROGRAM ASTAR_SEARCH;
var
   GOAL_NODE_SET              :set of 0..50;
   OPEN                       :set of 0..50;
   cost_file,out,hn           :text;
   str,ss                     :string[20];
   i,j,index,n,s,start_node,
   new_g,new_f,
   iteration,count,
   total_goal_node            :integer;
   goal_node                  :array[1..10] of integer;
   latest_g                   :array[0..50] of integer;
   g                          :array[0..50,0..50] of integer;
   h                          :array[0..50] of integer;
   f                          :array[0..50] of integer;
   father                     :array[0..50] of integer;
   x                          :array[0..50] of integer;
(*************************************************************)
procedure finding_minimum_of_fn;
{ THIS PROCEDURE FINDS MINIMUM  OF f(n)'s FOR THOSE NODES n, WHICH
  ARE IN OPEN SET, WHERE f(n)  IS AN OPTIMAL COST FROM START NODE
  TO GOAL NODE CONSTRAINED THROUGH NODE  n. }
var
   min :integer;
begin
   i:=0;
   while not(i in open) do i:=i+1;
   min:=f[i];
   index:=i;
   for j:= i+1 to n-1 do
     begin
        if j in OPEN
          then begin
                 if f[j] < min
                   then begin
                          min:=f[j];
                          index:=j;
                        end;
               end;
      end;
end;{finding_minimum_of_fn}
{****************************************************************               **
          MAIN PROGRAM
*****************************************************************************
THIS PROGRAM FINDS AN OPTIMAL PATH FROM START NODE TO GOAL NODE. THE ALGORITHM
FOLLOWED HERE IS  A* ALGORITHM.
}
BEGIN
   GOAL_NODE_SET:=[];
   OPEN:=[];
   write('WHAT IS THE OUT FILE ?.............');
   readln(str);
   assign(out,str);
   rewrite(out);
   write('GIVE THE TOTAL NUMBER OF NODES......');
   readln(n);
   write('GIVE THE START NODE               ');
```

```
readln(start_node);
write('GIVE THE TOTAL NUMBER OF GOAL NODE..');
readln(total_goal_node);
for i:=  1 to total_goal_node do
  begin
        write('GIVE THE GOAL NODES................');
        readln(goal_node[i]);
        GOAL_NODE_SET:=GOAL_NODE_SET + [goal_node[i]];
  end;
write('GIVE h(n) FILE NAME................');
readln(ss);
assign(hn,ss);
reset(hn);
for i:= 0 to n-1 do
begin
     readln(hn,h[i]);
     f[i] := maxint
end;
write('GIVE THE COST FILE NAME............');
readln(str);
assign(cost_file,str);
reset(cost_file);
for i:= 0 to n-1 do
  for j:= 0 to n-1 do
    read(cost_file,g[i,j]);
writeln(out,'TOTAL NUMBER OF NODES IN THE NETWORK =',n:4);
writeln(out);
writeln(out);
writeln(out,' THE FOLLOWING IS THE PATH MATRIX OF THE NETWORK WITH ASSOCIATED COSTS');
writeln(out);
writeln(out);
for i:= 0 to n-1 do
  BEGIN
    for j:= 0 to n-1 do
      if g[i,j] > 0
        then  write(out,g[i,j]:4)
        else write(out,'0':4);
    writeln(out);writeln;
  END;
writeln(out);
writeln(out);
writeln(out);
writeln(out,'THE HEURISTIC ASSOCIATED WITH EACH NODE ARE AS FOLLOWS');
writeln(out);
writeln(out);
for i:= 0 to n-1 do
  writeln(out,'h(',i,') =',h[i]);
writeln(out);
writeln(out);
writeln(out,'START NODE =',start_node:3);
writeln(out);
writeln(out);
writeln(out,'TOTAL NUMBER OF GOAL NODE =',total_goal_node:3);
for i:= 1 to total_goal_node do
  writeln(out,'GOAL NODE(',i,') =',goal_node[i]);
writeln(out);
writeln(out);
writeln(out);
writeln(out);
s:=start_node;
latest_g[s]:=0;
f[s]:=latest_g[s] +h[s];
iteration:=0;
writeln(out,'                        state');
writeln(out,'                      developed','g(n)':8,'f(n)':7,'father(n)':12,'OPEN set : n(g(n),f(n))':29);
writeln(out,'__________________________________________________________________________________');
writeln(out);
repeat
   iteration:=iteration +1;
   for i:= 0 to n-1 do
     begin
       if f[i] > 0
```

```
 STAGECOACH PROBLEM WITH  5 STAGES

FOLLOWING ARE THE STATES IN DIFFERENT STAGES
S(i,j)  means  j th state in stage i

    J-->
S(1,j)=    1

S(2,j)=    2     3     4

S(3,j)=    5     6     7

S(4,j)=    8     9

S(5,j)=   10


FOLLOWING ARE THE COST
C(i,j,k)  means the cost from  S(i,j)  to  S(i+1,k)


      K-->
C(1,1,k)=      4       2       3

C(2,1,k)=     10       9      ---

C(2,2,k)=      6       7      10

C(2,3,k)=    ---       3       8

C(3,1,k)=      4       8

C(3,2,k)=      9       6

C(3,3,k)=      5       4

C(4,1,k)=      8

C(4,2,k)=      4
THE OPTIMAL STATES ARE FOLLOWS
1
4
6
9
10

 THE MINIMUM POLICY COST =16
```

```
      begin
         mi:=m[i];
         found:=false;
         j:=0;
         repeat
            j:=j+1;
            if s[i,j] = next_optimal_state[i-1,kk] then found:=true;
         until ( (j=mi) or (found=true) );
         x[i+1]:=next_optimal_state[i,j];
         kk:=j;
      end;
      writeln; writeln; writeln;
      writeln(out, 'THE OPTIMAL STATES ARE FOLLOWS');
      for i:= 1 to n do writeln(out,x[i]);
   writeln(out); writeln(out);
   writeln(out,' THE MINIMUM POLICY COST =',minimum_policy_cost[1,1]);
end;
(************************************************************************
                    MAIN PROGRAM STARTS HERE
************************************************************************)
BEGIN

   write('GIVE THE TOTAL NUMBER OF STAGES   ');
   readln(n);
   writeln (' GIVE NUMBER OF STATES IN EACH STAGE  ');
   for i:= 1 to n do readln(m[i]);
   write(' GIVE THE COST FILE NAME  ');
   readln(str);
   assign(cost_file,str);
 {$I-}
   reset(cost_file);
 {$I+}
   if ioresult <> 0 then writeln('ERROR IN READING COST FILE');
   nm1:=n-1;
   for i:= 1 to nm1 do
     begin
        mi:=m[i];
        mip1:=m[i+1];
        for j:= 1 to mi do
          begin
             for k:= 1 to mip1 do read(cost_file, policy_cost[i,j,k]);
          end;
     end;
     nm1:=n-1;
   loop := 0;
   for i:= 1 to n do
     begin
        mi:=m[i];
        for j:= 1 to mi do
          begin
             loop := loop + 1;
             s[i,j] := loop;
          end;
     end;
   write(' SHALL I CALCULATE THE OPTIMAL PATH ?  ');
   readln(ch);
   if upcase(ch) = 'Y'
      then begin
              write(' WHAT IS THE OUTPUT FILE ?   ');
              readln(str);
              assign(out,str);
              rewrite(out);
              finding_optimal_path;
           end;
   write('SHALL I DRAW THE NETWORK ?  ');
   readln(ch);
   if upcase(ch) = 'Y'
     then  draw_network;
END.
```

```
    begin
       minimum_policy_cost[i,j]:=policy_cost[i,j,1];
       next_optimal_state[i,j]:=s[n,1];
    end;
  nm2:=n-2;
  for ii:= 1 to nm2 do
    begin
       i:=n-ii-1;
       mi:=m[i];
       mip1:=m[i+1];
       for j:= 1 to mi do
         begin
            minimum_policy_cost[i,j]:=policy_cost[i,j,1] + minimum_policy_cost[i+1,1];
            next_optimal_state[i,j]:=s[i+1,1];
            if mip1 <> 1
              then begin
                      for k:= 2 to mip1 do
                        begin
                           if minimum_policy_cost[i,j] > policy_cost[i,j,k] + minimum_policy_cost[i+1,k]
                             then begin
                                     minimum_policy_cost[i,j]:=policy_cost[i,j,k]+minimum_policy_cost[i+1,k]
                                     next_optimal_state[i,j]:=s[i+1,k];
                                  end;
                        end;
                   end;
         end;
    end;
   writeln(out,'  STAGECOACH PROBLEM WITH  ',n,' STAGES');
   writeln(out); writeln(out);
   writeln(out,' FOLLOWING ARE THE STATES IN DIFFERENT STAGES');
   writeln(out,' S(i,j)  means  j th state in stage i ');
   writeln(out);
   writeln(out,' J-->':9);
   for i:= 1 to n do
     begin
        write(out,' S(',i,',j)=');
        mi:=m[i];
        for j:= 1 to mi do write(out, s[i,j]:5);
        writeln(out);  writeln(out);
     end;
   writeln(out); writeln(out);
   writeln(out,' FOLLOWING ARE THE COST');
   writeln(out,' C(i,j,k)  means the cost from  S(i,j)  to  S(i+1,k)');
   writeln(out);  writeln(out);
   writeln(out,' K-->':11);
   for i:= 1 to nm1 do
     begin
        mi:=m[i];
        mip1:=m[i+1];
        for j:= 1 to mi do
          begin
             write(out,' C(',i,',',j,',','k)=');
             for k:= 1 to mip1 do
               begin
                  if policy_cost[i,j,k] < big_number
                    then write(out, policy_cost[i,j,k]:7)
                    else write(out,'    ','---')
               end;
             writeln(out);  writeln(out);
          end
     end;
{********************************************}
   x[1]:=s[1,1];
   x[2]:=next_optimal_state[1,1];
   kk:=1;
   for i:= 2 to nm1 do
     begin
        mi:=m[i];
        found:=false;
        j:=0;
        repeat
           j:=j+1;
```

```
(*$u+*)
PROGRAM STAGECOACH;
const
   big_number = 9999;
var
   ch                                   :char;
   str                                  :string[20];
   cost_file ,out                       :text;
   s                                    :array[1..10,1..8] of integer;
   policy_cost                          :array[1..8,1..7,1..7] of integer;
   minimum_policy_cost                  :array[1..8,1..8] of integer;
   next_optimal_state                   :array[1..8,1..8] of integer;
   x                                    :array[1..10] of integer;
   m                                    :array[1..10] of integer;
   n,i,j,kk,ii,nm1,nm2,mi,
   mip1,k,depth,startx,starty,
   endx,endy,y,dn,loop                  :integer;
   found                                :boolean;
{*********************************************************************}
procedure draw_network;
{ THIS PROCEDURE DRAW THE NETWORK OF THE GIVEN STAGE_COACH PROBLEM. }
begin
   write('GIVE THE DEPTH OF EACH LEVEL_________ ');
   readln(depth);
   graphmode;
   nm1:=n-1;
   for i:= 1 to nm1 do
     begin
        mi:=m[i];
        mip1:=m[i+1];
        for j:= 1 to mi do
          begin
             for k:= 1 to mip1 do
               begin
                  if policy_cost[i,j,k] < big_number
                    then begin
                            startx:=10*j*8 -depth;
                            starty:=((i-1)*depth+2)*8 -depth;
                            endx:=10*k*8 -depth;
                            endy:=i*depth*8 -depth;
                            draw(startx,starty,endx,endy,1);
                         end;
               end;
          end;
     end;
   highvideo;
   dn:= depth*n;
   for i := 1 to n do
     begin
        mi:=m[i];
        for j:= 1 to mi do write(s[i,j]:10);
        y:=depth*i+1;
        gotoxy(1,y);
     end;
end;
{*********************************************************************}
procedure finding_optimal_path;
{ THIS PROCEDURE FINDS THE OPTIMAL PATH FROM START NODE TO GOAL NODE.
  IT STARTS FROM GOAL NODE OF  STAGE  n AND THROUGH  BACKWARD SEARCH IT
  REACHES THE START NODE AT FIRST  STAGE. FOR EACH NODE  j IN  STAGE
  i IT FINDS THE NEXT OPTIMAL  STATE  IN STAGE  j+1. }
begin
   i:=n-1;
   mi:=m[i];
   for j:= 1 to mi do
```

STAGECOACH PROBLEM WITH 7 STAGES

FOLLOWING ARE THE STATES IN DIFFERENT STAGES
S(i,j) means j th state in stage i

J-->				
S(1,j)=	1			
S(2,j)=	2	3	4	
S(3,j)=	5	6	7	8
S(4,j)=	9	10	11	
S(5,j)=	12	13	14	
S(6,j)=	15	16		
S(7,j)=	17			

FOLLOWING ARE THE COST
C(i,j,k) means the cost from S(i,j) to S(i+1,k)

K-->				
C(1,1,k)=	8	7	4	
C(2,1,k)=	8	5	7	10
C(2,2,k)=	9	6	---	8
C(2,3,k)=	---	8	9	12
C(3,1,k)=	10	11	10	
C(3,2,k)=	---	7	9	
C(3,3,k)=	---	4	2	
C(3,4,k)=	7	8	13	
C(4,1,k)=	5	6	3	
C(4,2,k)=	11	10	12	
C(4,3,k)=	15	---	10	
C(5,1,k)=	7	6		
C(5,2,k)=	5	4		
C(5,3,k)=	9	8		
C(6,1,k)=	7			
C(6,2,k)=	8			

THE FOLLOWING ARE THE HEURISTICS ASSOCIATED WITH EACH NODE

h(1) = 32

C(5,1,k)=	7	6
C(5,2,k)=	5	4
C(5,3,k)=	9	8
C(6,1,k)=	7	
C(6,2,k)=	8	

THE FOLLOWING ARE THE HEURISTICS ASSOCIATED WITH EACH NODE

h(1) = 32
h(2) = 31
h(3) = 30
h(4) = 30
h(5) = 23
h(6) = 26
h(7) = 25
h(8) = 22
h(9) = 13
h(10) = 15
h(11) = 22
h(12) = 12
h(13) = 10
h(14) = 14
h(15) = 6
h(16) = 6
h(17) = 0

	state developed	g(n)	f(n)	father(n)	OPEN set : n(g(n),f(n))
iteration- 1)	1	0	32767	---	2(8,39) 3(7,37) 4(4,34)
iteration- 2)	4	4	34	1	2(8,39) 3(7,37) 6(12,38) 7(13,38) 8(16,38)
iteration- 3)	3	7	37	1	2(8,39) 5(16,39) 6(12,38) 7(13,38) 8(15,37)
iteration- 4)	8	15	37	3	2(8,39) 5(16,39) 6(12,38) 7(13,38) 9(22,35) 10(23,38) 11(28,50)

In stage 4 state 3 i.e. state number 11 is pruned

	state developed	g(n)	f(n)	father(n)	OPEN set : n(g(n),f(n))
iteration- 5)	9	22	35	8	2(8,39) 5(16,39) 6(12,38) 7(13,38) 10(23,38) 12(27,39) 13(28,38) 14(25,39)
iteration- 6)	6	12	38	4	2(8,39) 5(16,39) 7(13,38) 10(19,34) 12(27,39) 13(28,38) 14(25,39)
iteration- 7)	10	19	34	6	2(8,39) 5(16,39) 7(13,38) 12(27,39) 13(28,38) 14(25,39)

iteration- 9)	10	17	32	7	2(8,39) 5(16,39) 12(27,39) 13(27,37) 14(25,39)
iteration-10)	13	27	37	10	2(8,39) 5(16,39) 12(27,39) 14(25,39) 15(32,38) 16(31,37)
iteration-11)	16	31	37	13	2(8,39) 5(16,39) 12(27,39) 14(25,39) 15(32,38) 17(39,39)
iteration-12)	15	32	38	13	2(8,39) 5(16,39) 12(27,39) 14(25,39) 17(39,39)
iteration-13)	2	8	39	1	5(16,39) 6(12,38) 7(13,38) 8(15,37) 12(27,39) 14(25,39) 17(39,39)
iteration-14)	8	15	37	3	5(16,39) 6(12,38) 7(13,38) 9(22,35) 10(17,32) 12(27,39) 14(25,39) 17(39,39)
iteration-15)	10	17	32	7	5(16,39) 6(12,38) 7(13,38) 9(22,35) 12(27,39) 13(27,37) 14(25,39) 17(39,39)
iteration-16)	9	22	35	8	5(16,39) 6(12,38) 7(13,38) 12(27,39) 13(27,37) 14(25,39) 17(39,39)
iteration-17)	13	27	37	10	5(16,39) 6(12,38) 7(13,38) 12(27,39) 14(25,39) 15(32,38) 16(31,37) 17(39,39)
iteration-18)	16	31	37	13	5(16,39) 6(12,38) 7(13,38) 12(27,39) 14(25,39) 15(32,38) 17(39,39)
iteration-19)	6	12	38	4	5(16,39) 7(13,38) 10(17,32) 12(27,39) 14(25,39) 15(32,38) 17(39,39)
iteration-20)	10	17	32	7	5(16,39) 7(13,38) 12(27,39) 13(27,37) 14(25,39) 15(32,38) 17(39,39)
iteration-21)	13	27	37	10	5(16,39) 7(13,38) 12(27,39) 14(25,39) 15(32,38) 16(31,37) 17(39,39)
iteration-22)	16	31	37	13	5(16,39) 7(13,38) 12(27,39) 14(25,39) 15(32,38) 17(39,39)
iteration-23)	7	13	38	4	5(16,39) 10(17,32) 12(27,39) 14(25,39) 15(32,38) 17(39,39)
iteration-24)	10	17	32	7	5(16,39) 12(27,39) 13(27,37) 14(25,39) 15(32,38) 17(39,39)
iteration-25)	13	27	37	10	5(16,39) 12(27,39) 14(25,39) 15(32,38) 16(31,37) 17(39,39)
iteration-26)	16	31	37	13	5(16,39) 12(27,39) 14(25,39) 15(32,38) 17(39,39)

iteration-27)	15	32	38	13	5(16,39) 12(27,39) 14(25,39) 17(39,39)
iteration-28)	5	16	39	3	9(22,35) 10(17,32) 12(27,39) 14(25,39) 17(39,39)
iteration-29)	10	17	32	7	9(22,35) 12(27,39) 13(27,37) 14(25,39) 17(39,39)
iteration-30)	9	22	35	8	12(27,39) 13(27,37) 14(25,39) 17(39,39)
iteration-31)	13	27	37	10	12(27,39) 14(25,39) 15(32,38) 16(31,37) 17(39,39)
iteration-32)	16	31	37	13	12(27,39) 14(25,39) 15(32,38) 17(39,39)
iteration-33)	15	32	38	13	12(27,39) 14(25,39) 17(39,39)
iteration-34)	12	27	39	9	14(25,39) 15(32,38) 16(31,37) 17(39,39)
iteration-35)	16	31	37	13	14(25,39) 15(32,38) 17(39,39)
iteration-36)	15	32	38	13	14(25,39) 17(39,39)
iteration-37)	14	25	39	9	15(32,38) 16(31,37) 17(39,39)
iteration-38)	16	31	37	13	15(32,38) 17(39,39)
iteration-39)	15	32	38	13	17(39,39)

THE OPTIMAL PATH OBTAINED IS AS FOLLOWS

1-4-7-10-13-16-17-

THE OPTIMAL COST IS =39

```
{$u+}
PROGRAM STAGECOACH_WITH_PRUNING;
var
    OPEN                                   :set of 0..50;
    f                                      :array[0..50] of integer;
    h                                      :array[0..50] of integer;
    optimal                                :array[0..50]  of integer;
    father                                 :array[0..50] of integer;
    ch                                     :char;
    str                                    :string[20];
    hn,cost_file ,out                      :text;
    s                                      :array[1..10,1..8] of integer;
    policy_cost                            :array[1..8,1..7,1..7] of integer;
    minimum_policy_cost                    :array[1..8,1..8] of integer;
    next_optimal_state                     :array[1..8,1..8] of integer;
    x                                      :array[1..10] of integer;
    m                                      :array[1..10] of integer;
    n,i,j,kk,ii,nm1,nm2,mi,
    mip1,k,depth,startx,starty,
    endx,endy,y,dn,index,
    new_W,W,s_node,goal_node,
    iteration,start_node,new_optimal,
    new_f,count,total_node,
    loop,i1,j1,jj1,position,
    loop1                         :integer;
    found                                  :boolean;
(**********************************************************)
procedure finding_minimum_of_fn;
{ THIS PROGRAM FINDS THE MINIMUM OF f(n)'s FOR THOSE NODES WHICH
  ARE IN OPEN SET, WHERE f(n) IS AN OPTIMAL PATH FROM START NODE
  TO GOAL NODE CONSTRAINED THROUGH NODE n. }
var
    min : integer;
begin
    i1:=2;
    while not(i1 in open) do i1:=i1+1 ;
    min:=f[i1];
    index:=i1;
    for j1:= i1+1 to total_node do
      begin
          if j1 in OPEN
            then begin
                      if f[j1] < min
                        then begin
                                 min:=f[j1];
                                 index:=j1;
                             end;
                 end;
      end;
    position := m[1];
    loop := 1;
    repeat
        loop := loop +1;
        position := position + m[loop];
    until index <= position;
    i := loop;
    loop := 0;
    repeat
        loop := loop +1;
    until s[i,loop] = index;
    j := loop;
end;{finding_minimum_of_fn}
{*************************************************************************}
procedure draw_network;
{ THIS PROCEDURE DRAW THE NETWORK OF THE GIVEN STAGE_COACH PROBLEM }
begin
```

```
   readln(depth);
   graphmode;
   nm1:=n-1;
   for i:= 1 to nm1 do
     begin
        mi:=m[i];
        mip1:=m[i+1];
        for j:= 1 to mi do
          begin
              for k:= 1 to mip1 do
                begin
                    if policy_cost[i,j,k] >0
                      then begin
                                   startx:=10*j*8 -depth;
                                   starty:=((i-1)*depth+2)*8 -depth;
                                   endx:=10*k*8 -depth;
                                   endy:=i*depth*8 -depth;
                                   draw(startx,starty,endx,endy,1);
                               end;
                end;
          end;
     end;
   highvideo;
   dn:= depth*n;
   for i := 1 to n do
     begin
        mi:=m[i];
        for j:= 1 to mi do write(s[i,j]:10);
        y:=depth*i+1;
        gotoxy(1,y);
     end;
end;

{*************************************************************************}
procedure calculate_W;
begin
   new_W :=0;
   i1 := i;
   j1 := j;

   s[i1,j1] := s_node;
   repeat
       k :=0;
       repeat
           k := k+1;
       until policy_cost[i1,j1,k] > 0;
       new_w := new_W + policy_cost[i1,j1,k];
       i1 := i1+1;
       j1 := k;
   until s[i1,j1] = goal_node;
   new_W := optimal[s_node] + new_W;
   if new_W < W then W := new_W;
end;
{*************************************************************************}
procedure finding_optimal_path_with_pruning;
begin
    writeln(out);
    writeln(out);
    writeln(out);
    writeln(out);
    writeln(out,'                    state');
    writeln(out,'                 developed','g(n)':8,'f(n)':7,'father(n)':12,'OPEN set : n(g(n),f(n))':29);
    writeln(out,'----------------------------------------------------------------------------------------');
    writeln(out);
    i := 1;
    j := 1;
    iteration := 0;
    s_node := start_node;
    repeat
        iteration := iteration +1;
        calculate_W;
   writeln('W =',W);
```

```
 i1 := i+1;
 mip1 := m[i1];
 for k:= 1 to mip1 do
   if policy_cost[i,j,k] > 0
     then begin
            OPEN := OPEN + [s[i+1,k]];
            new_optimal := optimal[s[i,j]] + policy_cost[i,j,k];
            new_f := new_optimal + h[s[i+1,k]];
            if new_f < f[s[i+1,k]]
              then begin
                     f[s[i+1,k]] := new_f;
                     optimal [s[i+1,k]] := new_optimal;
                     father[s[i+1,k]] := s_node;
                   end;
          end;

  if  iteration = 1
    then begin
           write(out,'iteration-',iteration:2,')',s_node:5,optimal[s_node]:10,f[s_node]:8);
           write(out,'---':9);
           write(out,'      ');
         end
    else begin
           write(out,'iteration-',iteration:2,')',s_node:5,optimal[s_node]:10,f[s_node]:8,father[s_node]:9);
           write(out,'      ');
         end;
  count := 0;
  for loop:= start_node to total_node do
  if loop in OPEN
     then begin
            count := count +1;
            write(out,loop:2,'(',optimal[loop],',',f[loop],') ');
            if count > 4 then
              begin
                count :=0;
                writeln(out);
                write(out,' ':52);
              end;
          end;
  writeln(out);  writeln(out);  writeln(out);
  finding_minimum_of_fn;

  for loop1 := start_node to total_node do
     begin
        if ((loop1 in OPEN) and (f[loop1] > W))
          then begin
                  position := m[1];
                  loop := 1;
                  repeat
                      loop := loop + 1;
                      position := position + m[loop];
                  until loop1 <= position;
                  i1 := loop;
                  loop := 0;
                  repeat
                      loop := loop + 1;
                  until s[i1,loop] = loop1;
                  j1 := loop;

                  for jj1 := 1 to m[i1+1] do
                     if policy_cost[i,j1,jj1] > 0 then policy_cost[i1,j1,jj1] := 0;
                  OPEN := OPEN -[loop1];
                  for jj1 := 1 to m[i1-1] do
                     if policy_cost[i1-1,jj1,j1] > 0 then policy_cost[i1-1,jj1,j1] := 0;
         writeln(out,'In  stage  ',i1,'  state  ',j1,'  i.e.  state number  ',loop1,'  is pruned');
         writeln(out);
         writeln(out);

               end;
     end;
```

```
        s_node := index;
        OPEN := OPEN - [s_node];
    until s_node = goal_node;
    writeln(out);
    writeln(out);
    writeln(out,'________________________________________________________________________');
    writeln(out);
    writeln(out);

    { FINDING OPTIMAL PATH  }
    x [1] := s_node;
    count := 1;
    repeat
        count := count +1;
        x [count] := father [x[count -1]];
    until x [count] = start_node;
    writeln(out,'THE OPTIMAL PATH OBTAINED IS AS FOLLOWS');
    writeln(out);
    for i:= count downto 1 do
      write(out,x[i],'-');
      writeln(out);
      writeln(out);
      writeln(out,'THE OPTIMAL COST IS =',f[s_node]);
end;

{*********************************************************************
                    MAIN PROGRAM STARTS HERE
*********************************************************************}
BEGIN
   write('GIVE THE TOTAL NUMBER OF STAGES   ');
   readln(n);
   writeln (' GIVE NUMBER OF STATES IN EACH STAGE  ');
   for i:= 1 to n do readln(m[i]);
   for i:= 1 to n do writeln(m[i]);
   write(' GIVE THE COST FILE NAME  ');
   readln(str);
   assign(cost_file,str);
 {$I-}
   reset(cost_file);
 {$I+}
   if ioresult <> 0 then writeln('ERROR IN READING COST FILE');
   nm1:=n-1;
   for i:= 1 to nm1 do
     begin
        mi:=m[i];
        mip1:=m[i+1];
        for j:= 1 to mi do
          begin
             for k:= 1 to mip1 do read(cost_file, policy_cost[i,j,k]);
          end;
     end;
     nm1:=n-1;
   write('GIVE THE HEURISTIC FILE NAME  ');
   readln(str);
   assign(hn,str);
   reset(hn);
   total_node := 0;
   for loop := 1 to n do
     total_node := total_node + m[loop];
   for loop := 1 to total_node do
     readln(hn,h[loop]);
   for loop := 1 to total_node do
     begin
        optimal[loop] := 0;
        f[loop] := maxint;
     end;
   W := maxint;
   start_node := 1;
   goal_node := total_node;
   writeln('GOAL NODE =',goal_node);
```

iteration-32)	16	31	37	13	12(27,39) 14(25,39) 15(32,38) 17(39,39)
iteration-33)	15	32	38	13	12(27,39) 14(25,39) 17(39,39)
iteration-34)	12	27	39	9	14(25,39) 15(32,38) 16(31,37) 17(39,39)
iteration-35)	16	31	37	13	14(25,39) 15(32,38) 17(39,39)
iteration-36)	15	32	38	13	14(25,39) 17(39,39)
iteration-37)	14	25	39	9	15(32,38) 16(31,37) 17(39,39)
iteration-38)	16	31	37	13	15(32,38) 17(39,39)
iteration-39)	15	32	38	13	17(39,39)

```
    loop := 0;
    for i:= 1 to n do
      begin
         mi:=m[i];
         for j:= 1 to mi do
           begin
              loop := loop +1;
              s[i,j] := loop;
           end;
      end;


    write(' SHALL I CALCULATE THE OPTIMAL PATH ?  ');
    readln(ch);
    if upcase(ch) = 'Y'
       then begin
                write(' WHAT IS THE OUTPUT FILE ?   ');
                readln(str);
                assign(out,str);
                rewrite(out);
                OPEN :=[];
                   writeln(out,'  STAGECOACH PROBLEM WITH  ',n,' STAGES');
    writeln(out); writeln(out);
    writeln(out,' FOLLOWING ARE THE STATES IN DIFFERENT STAGES');
    writeln(out,' S(i,j)  means  j th state in stage i ');
    writeln(out);
    writeln(out,' J-->':9);
    for i:= 1 to n do
      begin
         write(out,' S(',i,',j)=');
         mi:=m[i];
         for j:= 1 to mi do write(out, s[i,j]:5);
         writeln(out);  writeln(out);
      end;
    writeln(out); writeln(out);
    writeln(out,' FOLLOWING ARE THE COST');
    writeln(out,' C(i,j,k)  means the cost from  S(i,j)  to  S(i+1,k)');
    writeln(out);  writeln(out);
    writeln(out,' K-->':11);
    for i:= 1 to nm1 do
      begin
         mi:=m[i];
         mip1:=m[i+1];
         for j:= 1 to mi do
           begin
              write(out,' C(',i,',',j,',','k)=');
              for k:= 1 to mip1 do
                begin
                   if policy_cost[i,j,k]  > 0
                     then write(out, policy_cost[i,j,k]:7)
                     else write(out,'     ','---')
                end;
              writeln(out);  writeln(out);
           end
      end;
                finding_optimal_path_with_pruning;
             end;
    write('SHALL I DRAW THE NETWORK ?  ');
    readln(ch);
    if upcase(ch) = 'Y'
      then  draw_network;
END.
```

STAGECOACH PROBLEM WITH 7 STAGES

FOLLOWING ARE THE STATES IN DIFFERENT STAGES
S(i,j) means j th state in stage i

J-->				
S(1,j)=	1			
S(2,j)=	2	3	4	
S(3,j)=	5	6	7	8
S(4,j)=	9	10	11	
S(5,j)=	12	13	14	
S(6,j)=	15	16		
S(7,j)=	17			

FOLLOWING ARE THE COST
C(i,j,k) means the cost from S(i,j) to S(i+1,k)

K-->				
C(1,1,k)=	8	7	4	
C(2,1,k)=	8	5	7	10
C(2,2,k)=	9	6	---	8
C(2,3,k)=	---	8	9	12
C(3,1,k)=	10	11	10	
C(3,2,k)=	---	7	9	
C(3,3,k)=	---	4	2	
C(3,4,k)=	7	8	13	
C(4,1,k)=	5	6	3	

C(4,2,k)=	11	10	12
C(4,3,k)=	15	---	10
C(5,1,k)=	7	6	
C(5,2,k)=	5	4	
C(5,3,k)=	9	8	
C(6,1,k)=	7		
C(6,2,k)=	8		

	state developed	g(n)	f(n)	father(n)	OPEN set : n(g(n),f(n))
iteration- 1)	1	0	32767	---	2(8,39) 3(7,37) 4(4,34)
iteration- 2)	4	4	34	1	2(8,39) 3(7,37) 6(12,38) 7(13,38) 8(16,38)
iteration- 3)	3	7	37	1	2(8,39) 5(16,39) 6(12,38) 7(13,38) 8(15,37)
iteration- 4)	8	15	37	3	2(8,39) 5(16,39) 6(12,38) 7(13,38) 9(22,35) 10(23,38) 11(28,50)

In stage 4 state 3 i.e. state number 11 is pruned

iteration- 5)	9	22	35	8	2(8,39) 5(16,39) 6(12,38) 7(13,38) 10(23,38) 12(27,39) 13(28,38) 14(25,39)
iteration- 6)	6	12	38	4	2(8,39) 5(16,39) 7(13,38) 10(19,34) 12(27,39) 13(28,38) 14(25,39)
iteration- 7)	10	19	34	6	2(8,39) 5(16,39) 7(13,38) 12(27,39) 13(28,38) 14(25,39)
iteration- 8)	7	13	38	4	2(8,39) 5(16,39) 10(17,32) 12(27,39) 13(28,38) 14(25,39)
iteration- 9)	10	17	32	7	2(8,39) 5(16,39) 12(27,39) 13(27,37) 14(25,39)
iteration-10)	13	27	37	10	2(8,39) 5(16,39) 12(27,39) 14(25,39) 15(32,38) 16(31,37)
iteration-11)	16	31	37	13	2(8,39) 5(16,39) 12(27,39) 14(25,39) 15(32,38) 17(39,39)
iteration-12)	15	32	38	13	2(8,39) 5(16,39) 12(27,39) 14(25,39) 17(39,39)

iteration-13)	2	8	39	1	5(16,39) 6(12,38) 7(13,38) 8(15,37) 12(27,39) 14(25,39) 17(39,39)
iteration-14)	8	15	37	3	5(16,39) 6(12,38) 7(13,38) 9(22,35) 10(17,32) 12(27,39) 14(25,39) 17(39,39)
iteration-15)	10	17	32	7	5(16,39) 6(12,38) 7(13,38) 9(22,35) 12(27,39) 13(27,37) 14(25,39) 17(39,39)
iteration-16)	9	22	35	8	5(16,39) 6(12,38) 7(13,38) 12(27,39) 13(27,37) 14(25,39) 17(39,39)
iteration-17)	13	27	37	10	5(16,39) 6(12,38) 7(13,38) 12(27,39) 14(25,39) 15(32,38) 16(31,37) 17(39,39)
iteration-18)	16	31	37	13	5(16,39) 6(12,38) 7(13,38) 12(27,39) 14(25,39) 15(32,38) 17(39,39)
iteration-19)	6	12	38	4	5(16,39) 7(13,38) 10(17,32) 12(27,39) 14(25,39) 15(32,38) 17(39,39)
iteration-20)	10	17	32	7	5(16,39) 7(13,38) 12(27,39) 13(27,37) 14(25,39) 15(32,38) 17(39,39)
iteration-21)	13	27	37	10	5(16,39) 7(13,38) 12(27,39) 14(25,39) 15(32,38) 16(31,37) 17(39,39)
iteration-22)	16	31	37	13	5(16,39) 7(13,38) 12(27,39) 14(25,39) 15(32,38) 17(39,39)
iteration-23)	7	13	38	4	5(16,39) 10(17,32) 12(27,39) 14(25,39) 15(32,38) 17(39,39)
iteration-24)	10	17	32	7	5(16,39) 12(27,39) 13(27,37) 14(25,39) 15(32,38) 17(39,39)
iteration-25)	13	27	37	10	5(16,39) 12(27,39) 14(25,39) 15(32,38) 16(31,37) 17(39,39)
iteration-26)	16	31	37	13	5(16,39) 12(27,39) 14(25,39) 15(32,38) 17(39,39)
iteration-27)	15	32	38	13	5(16,39) 12(27,39) 14(25,39) 17(39,39)
iteration-28)	5	16	39	3	9(22,35) 10(17,32) 12(27,39) 14(25,39) 17(39,39)
iteration-29)	10	17	32	7	9(22,35) 12(27,39) 13(27,37) 14(25,39) 17(39,39)
iteration-30)	9	22	35	8	12(27,39) 13(27,37) 14(25,39) 17(39,39)
iteration-31)	13	27	37	10	12(27,39) 14(25,39) 15(32,38) 16(31,37) 17(39,39)

iteration-32)	16	31	37	13	12(27,39) 14(25,39) 15(32,38) 17(39,39)
iteration-33)	15	32	38	13	12(27,39) 14(25,39) 17(39,39)
iteration-34)	12	27	39	9	14(25,39) 15(32,38) 16(31,37) 17(39,39)
iteration-35)	16	31	37	13	14(25,39) 15(32,38) 17(39,39)
iteration-36)	15	32	38	13	14(25,39) 17(39,39)
iteration-37)	14	25	39	9	15(32,38) 16(31,37) 17(39,39)
iteration-38)	16	31	37	13	15(32,38) 17(39,39)
iteration-39)	15	32	38	13	17(39,39)

THE OPTIMAL PATH OBTAINED IS AS FOLLOWS

1-4-7-10-13-16-17-

THE OPTIMAL COST IS =39