NAME: Aditya Somani

PRN: 71901204L ROLL:

T1851061

SL-5(Group'B')

Assignment No. (B)4

AIM: Write a program to solve the travelling salesman problem and to print the path and the

cost using Dynamic Programming.

OBJECTIVE:

1. To understand the concept of Hamiltonian path.

2. To understand the concept of non-recursive backtracking.

3. How to find Time Complexity of algorithms and verify the same.

THEORY:

Backtracking:

Backtracking is a rather typical recursive algorithm, and any recursive algorithm can

be rewritten as a stack algorithm. In fact, that is how your recursive algorithms are translated

into machine or assembly language.

boolean solve(Node n) {

put node n on the stack;

while the stack is not empty {

if the node at the top of the stack is a leaf {

if it is a goal node, return true

else pop it off the stack

}

else {

if the node at the top of the stack has untried children

push the next untried child onto the stack

else pop the node off the stack

}

return false

}

Starting from the root, the only nodes that can be pushed onto the stack are the children

of the node currently on the top of the stack, and these are only pushed on one child at a time;

hence, the nodes on the stack at all times describe a valid path in the tree. Nodes are removed

from the stack only when it is known that they have no goal nodes among their descendents.

Therefore, if the root node gets removed (making the stack empty), there must have been no

goal nodes at all, and no solution to the problem.

When the stack algorithm terminates successfully, the nodes on the stack form (in reverse

order) a path from the root to a goal node.

Algorithm:

•

```
Function Bound(X[1:n],l)
begin
/* X[1:l-1] are assigned C-compliant
values. This function checks to
see if X[l] is C-compliant */
for i=1 to l-1 do
if X[l] = X[i] then
return(false);
endif
end for
if (l > 1 and (X[l-1],X[l]) is not an edge) then
return(false);
endif
if (l=n and (X[n],X[1]) is not an edge) then
return(false);
endif
return(true);
end
```

Time Complexity of Hamiltonian path :

In Hamiltonian cycle, in each recursive call one of the remaining vertices is selected in the worst case. In each recursive call the branch factor decreases by 1. Recursion in this case can be thought of as n nested loops where in each loop the number of iterations decreases by one. Hence the time complexity is given by:

T(N) = N*(T(N-1) + O(1))

T(N) = N*(N-1)*(N-2).. = O(N!)

Conclusion:

We successfully solved the travelling salesman problem and printed the path and the cost using

Dynamic Programming.

PROGRAM:

```
#include <stdio.h>

int matrix[25][25], visited_cities[10], limit, cost = 0;


int tsp(int c)

{

 int count, nearest_city = 999;

 int minimum = 999, temp;

 for(count = 0; count < limit; count++)

        {

                if((matrix[c][count] != 0) && (visited_cities[count] == 0))

                        {

                                if(matrix[c][count] < minimum)

                                        {

                                                minimum = matrix[count][0] + matrix[c][count];

                                        }
```

```
                                        temp = matrix[c][count];
                                        nearest_city = count;
                                }
        }
 if(minimum != 999)
        {
                cost = cost + temp;
        }
 return nearest_city;
}

void minimum_cost(int city)
{
 int nearest_city;
 visited_cities[city] = 1;
 printf("%d ", city + 1);
 nearest_city = tsp(city);
 if(nearest_city == 999)
        {
                nearest_city = 0;
                printf("%d", nearest_city + 1);
               cost = cost + matrix[city][nearest_city];
 return;
        }
 minimum_cost(nearest_city);
```

```
}

int main()
{
 int i, j;
 printf("Enter Total Number of Cities:\t");
 scanf("%d", &limit);
 printf("\nEnter Cost Matrix\n");
 for(i = 0; i < limit; i++)
        {
                printf("\nEnter %d Elements in Row[%d]\n", limit, i + 1);
                        for(j = 0; j < limit; j++)
                                {
                                        scanf("%d", &matrix[i][j]);
                                }
                visited_cities[i] = 0;
        }
 printf("\nEntered Cost Matrix\n");
 for(i = 0; i < limit; i++)
        {
                printf("\n");
                        for(j = 0; j < limit; j++)
                                {
                                        printf("%d ", matrix[i][j]);
                                }
```

```
        }
 printf("\n\nPath:\t");
 minimum_cost(0);
 printf("\n\nMinimum Cost: \t");
 printf("%d\n", cost);
 return 0;
}
```

OUTPUT:-