PROJECT REPORT

Algorithm

COURSE CODE: CSE246

COURSE INSTRUCTOR:

JESAN AHMED OVI SIR

DECEMBER 6, 2020

ADRI SAHA

Student ID: 2019-1-60-024

Problem Statement:

Travelling Salesman Problem- A traveler needs to visit all the cities from a list, where distances between all the cities are known and each city should be visited just once. In this project, I have written a code in C++ that solves the travelling salesman problem and shows the best path along with the minimal cost. I used recursive algorithm based on Backtracking approach in this regard as they can give same result in far fewer attempts than Brute Force method trials and thus handles the complexity better.

System Requirements:

Processor, RAM, Operating system, IDE, Code blocks.

System Design:

Algorithm & simulation pictures are given below-

I have used Backtracking approach using recursion in this code.

Depth-first search is an algorithm for tree or graph data structures to traverse or search. The algorithm starts at the root node (in the case of a graph, selecting some arbitrary node as the root node) and explores as far as possible before backtracking along each branch. The fundamental idea is therefore to start from the root or some arbitrary node and mark the node and shift a node to the neighboring unmarked node and continue this loop until there is no unmarked adjacent node. Then, backtrack and search for and traverse other unmarked nodes. The nodes in the path are eventually written. In this code,

1. I create a recursive function that takes the index of node and a visited array.
2. Mark the current node as visited and print the node.
3. Traverse all the adjacent and unmarked nodes and call the recursive function with index of adjacent node.
4. And finally print the path by using minipath array.
   Here, minipath is the shortest path for travelling & travel is the minimum cost for this problem.

Implementation:

```
for(i=0;i<n;i++)
  {
    if(!marked[i]&&graph[city][i])
    {
      marked[i]=true;
      tempPath[count]=i;
      minimal_cost(graph,marked,i,n,count+1,cost+graph[city][i],travel);
      marked[i]=false;
    }
```

```
    }
```
Here, visited nodes are true or 1 in mark array. And non-visited are false. This is the backtracking step. The for loop is traversing for the adjacency list. The city node is increasing the count. If marked[i]!=0 and graph[city][i] is false which means it has edges then will enter this loop and make marked true. Minimal_cost recursion will continue <n. When it is equal to na the node marked will make it 0 or false which is backtracking approach.

```
if(count==n && graph[city][0])
   {
      if(travel>cost+graph[city][0])
      {
         for(i=0;i<n;i++)
         {
            minipath[i]=tempPath[i];
         }
      }
      travel=min(travel,cost+graph[city][0]);
   }
```
Minimal_cost is the recursive function which is recursively finding the minimum cost by travel variable and minimum path by minipath array. It is finding minimum path without last node+ every time and create another cycle to find out the possible minimum path. It works like depth first search. It will back if there is no edge or end the path.

```
int i,j,n,graph[100][100];
   cout<<"Enter number of cities: ";
   cin>>n;
   cout<<"\nEnter the cost matrix "<<endl;
   for(i=0; i<n; i++)
   {
      cout<<"\nEnter elements of row "<<i+1<<" : "<<endl;
      for(j=0; j<n; j++)
      {
         cin>>graph[i][j];
      }
   }
```
This is the initial part to create the graph in 2D matrix.

```
cout<<"\n\nThe cost list is:"<<endl;
```

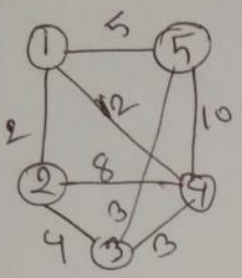
```
for(i=0;i<n;i++)
{
   for(j=0;j<n;j++)
      cout<<"   "<<graph[i][j];

   cout<<endl;
}
```

It is the printing part.

```
. vector<bool>marked(n);
   for(i=0;i<n;i++)
   {
   marked[i]=false;
   }
   marked[0]=true;
   int travel=INT_MAX;
   tempPath[0]=0;
   minimal_cost(graph,marked,0,n,1,0,travel);
```

Initializing and passing into the function part.

Simulation:



	1	2	3	4	5
1	0	2	0	12	5
2	2	0	4	8	0
3	0	4	0	3	3
4	12	8	3	0	10
5	5	0	3	10	0

Paths:

① | 1 | 2 | 3 | 4 | 5 | stop = 24

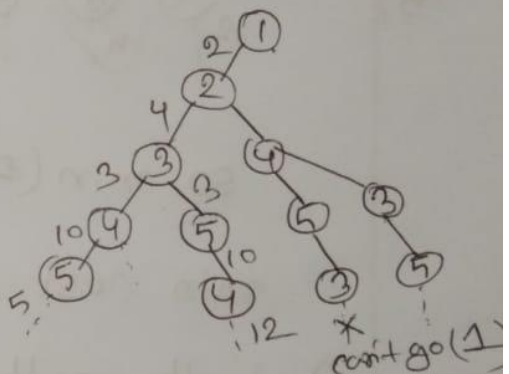
1	2	3	4	0

1	2	3	5	0

full so stop

⑪ | 1 | 2 | 3 | 5 | 4 | = 31

edges



⑪. | 1 | 2 | 4 | 5 | 3 | = 23 ×

⑫ | 1 | 2 | 4 | 3 | 0 |

after some steps

~~| 1 | 4 | 2 | 5 | 3 |~~

1	4	2	0	0

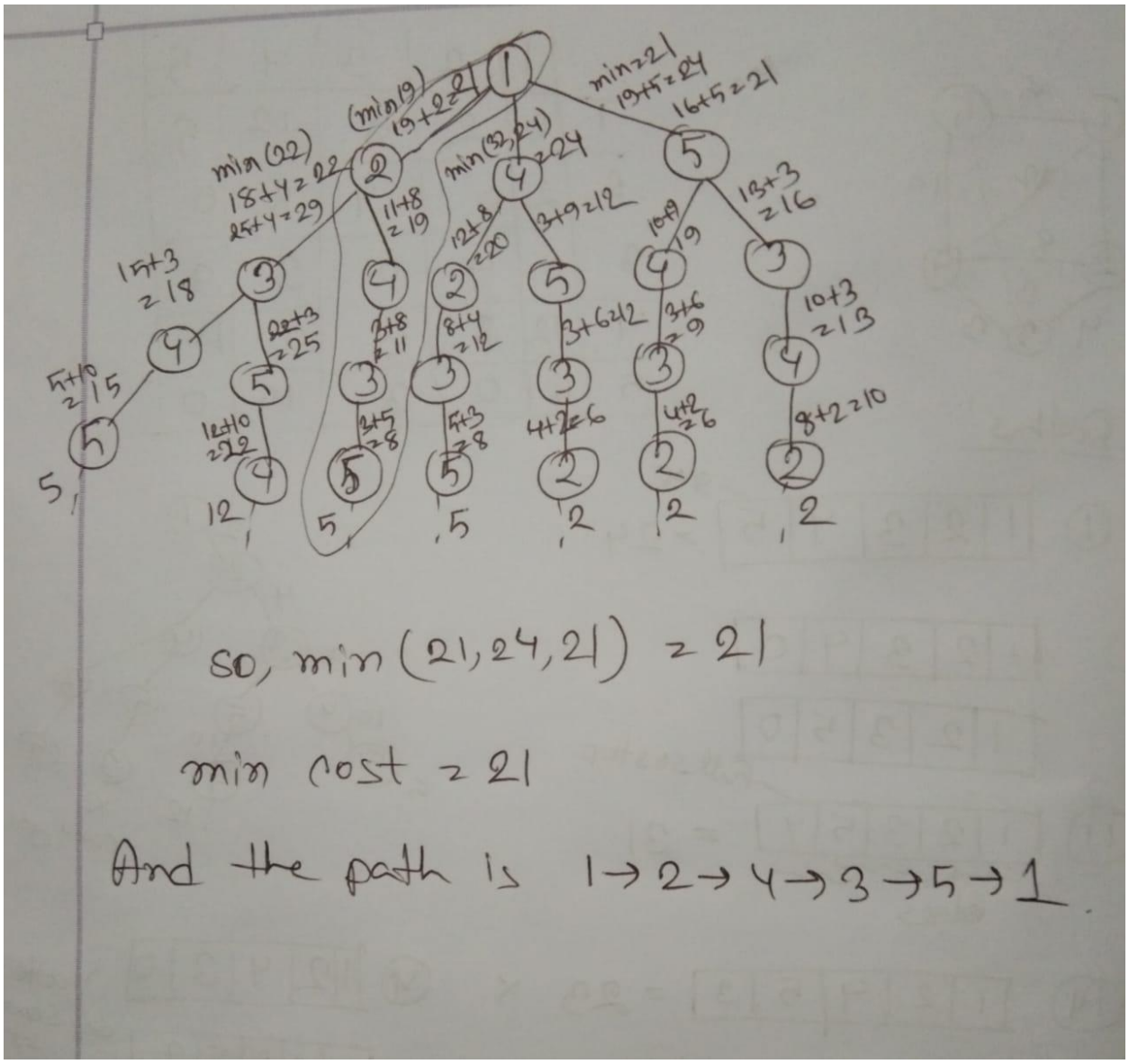
1	4	2	3	5

⑬ | 1 | 2 | 4 | 3 | 5 | = 21

⑭ | 1 | 4 | 5 | 3 | 2 |

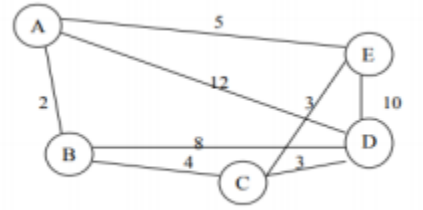
(vii) | 1 | 5 | 4 | 3 | 2 | = 24

viii) | 1 | 5 | 3 | 4 | 2 | = 21

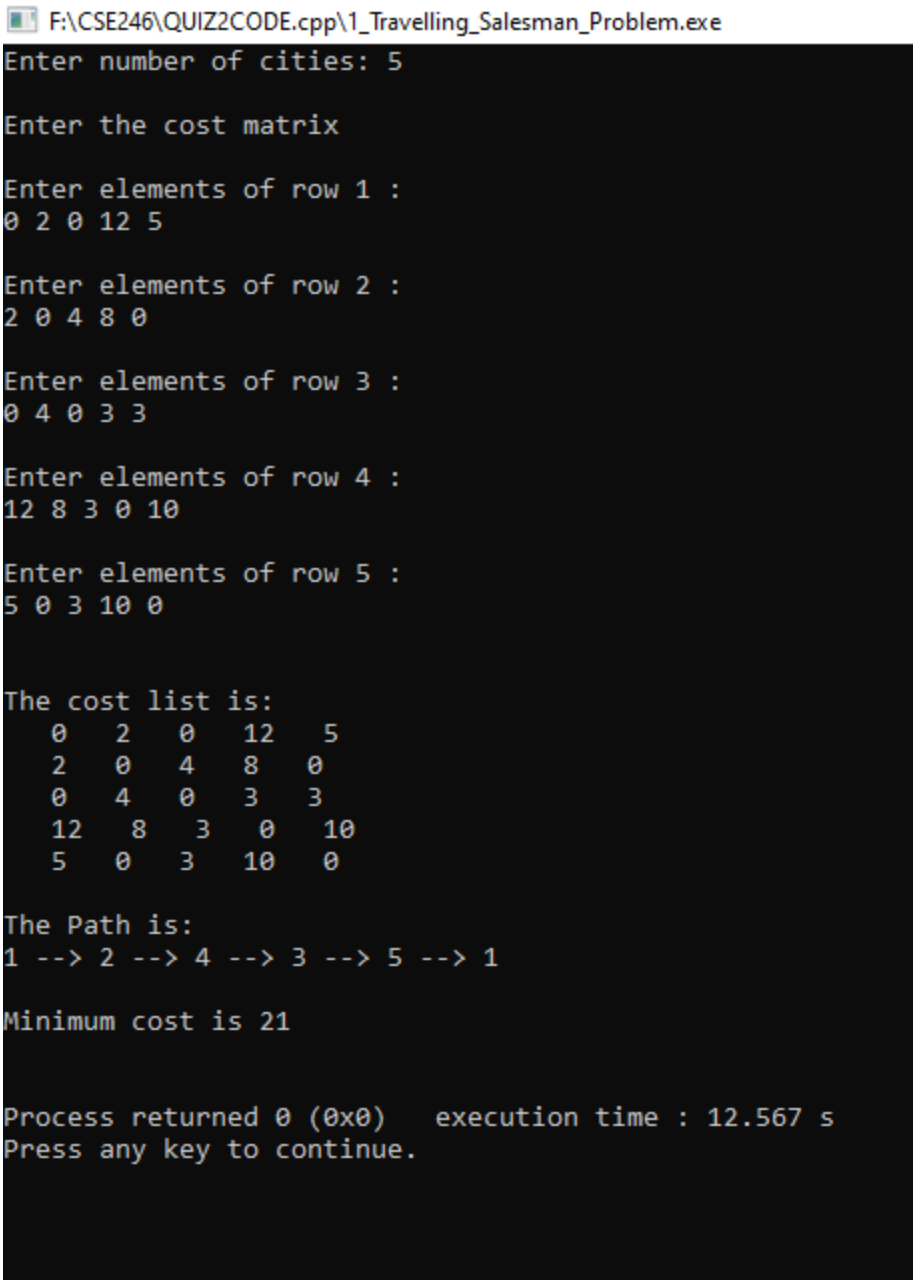


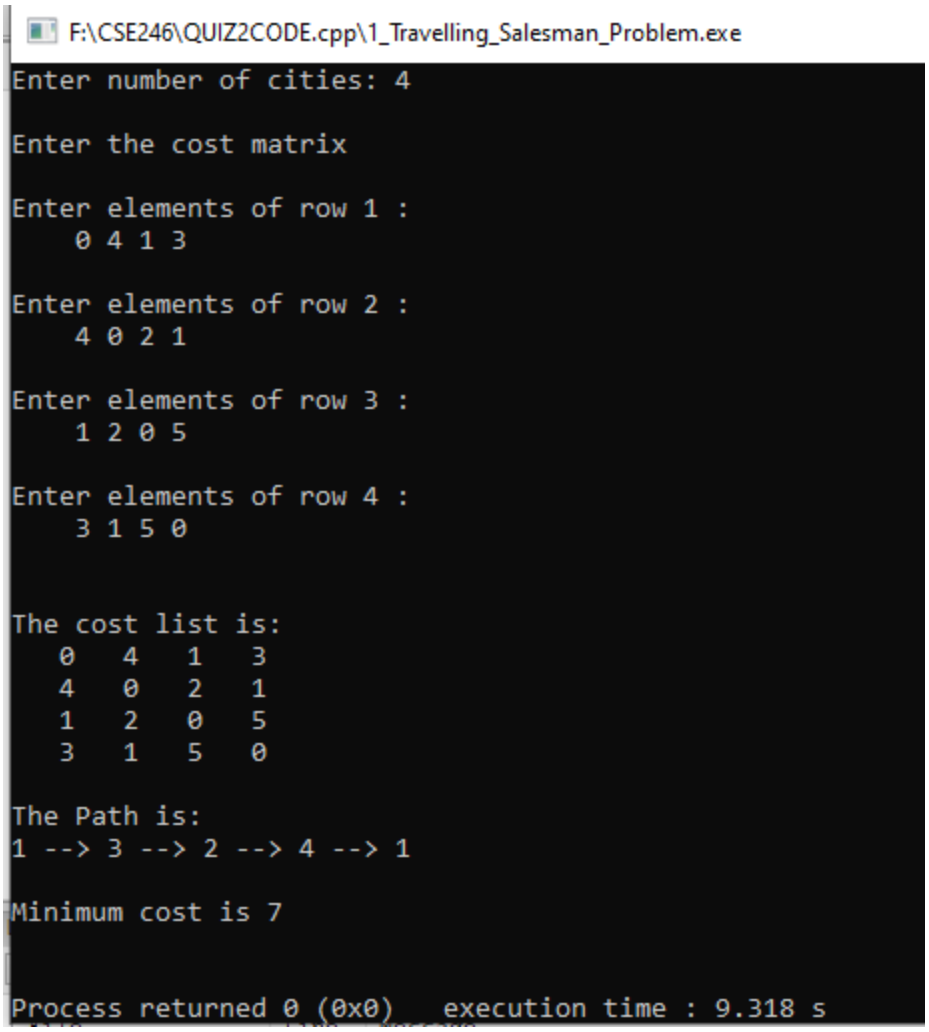
Testing Results:

There are some output of this code.



Input graph


F:\CSE246\QUIZ2CODE.cpp\1_Travelling_Salesman_Problem.exe
Enter number of cities: 5
Enter the cost matrix
Enter elements of row 1 :
0 2 0 12 5
Enter elements of row 2 :
2 0 4 8 0
Enter elements of row 3 :
0 4 0 3 3
Enter elements of row 4 :
12 8 3 0 10
Enter elements of row 5 :
5 0 3 10 0
The cost list is:
0 2 0 12 5
2 0 4 8 0
0 4 0 3 3
12 8 3 0 10
5 0 3 10 0
The Path is:
1 --> 2 --> 4 --> 3 --> 5 --> 1
Minimum cost is 21
Process returned 0 (0x0) execution time : 12.567 s
Press any key to continue.
Output 1



output 2

The input is in 2D array matrix with cost. And the output is of the minimum cost for travelling and this minimum path.