Data Structures and Algorithms

Lab Report

Lab13

Group Members Name & Reg #:	**Muhammad Haris Irfan** **(FA18-BCE-090)**
Class	Data Structures and Algorithms CSC211 (**BCE-3B**)
Instructor's Name	Dilshad Sabir

In Lab Tasks

Task:1

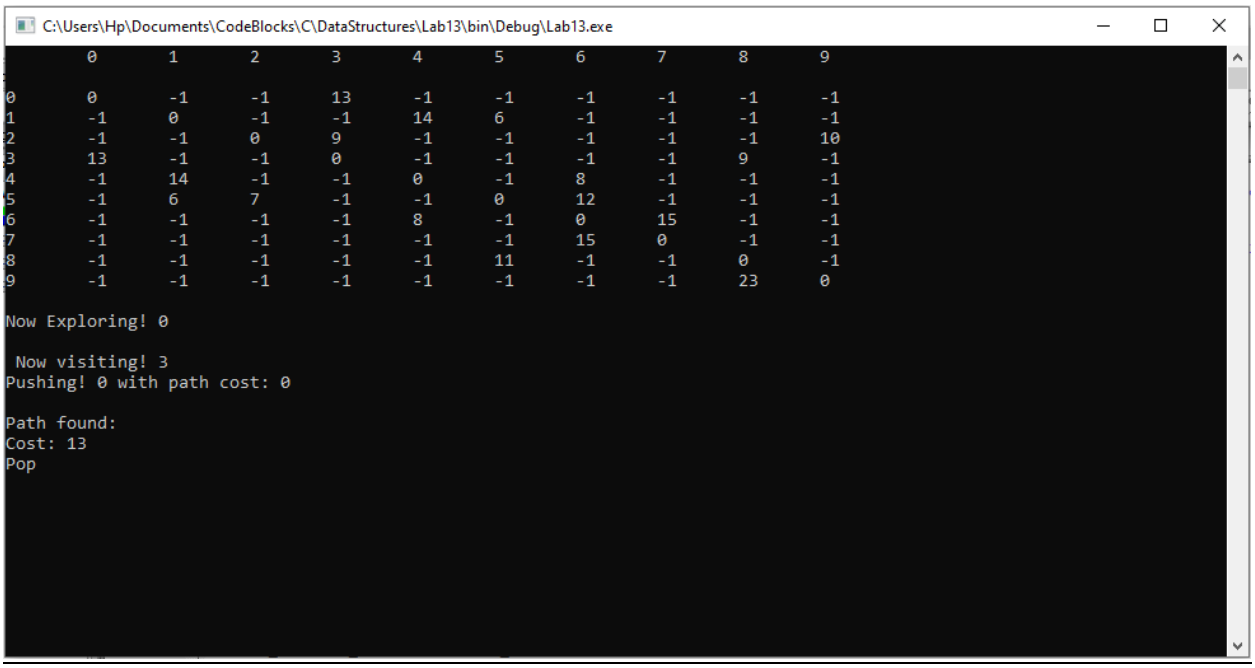
Complete the Adjacency Matrix given as 'my_graph [ ] [ ]' in the main function of the skeleton code provided.

Solution:

The code is shown below,

```
  /** Task 1: Complete the rest of the Adjacency Matrix according to Figure 8 **/
add_edge(&my_graph[0][0], NUM_VERTICES, 0, 3, 13);
  add_edge(&my_graph[0][0], NUM_VERTICES, 1, 4, 14);
  add_edge(&my_graph[0][0], NUM_VERTICES, 1, 5, 6);
  add_edge(&my_graph[0][0], NUM_VERTICES, 5, 2, 7);
  add_edge(&my_graph[0][0], NUM_VERTICES, 2, 3, 9);
  add_edge(&my_graph[0][0], NUM_VERTICES, 2, 9, 10);
  add_edge(&my_graph[0][0], NUM_VERTICES, 6, 7, 15);
  add_edge(&my_graph[0][0], NUM_VERTICES, 7, 6, 15);
  add_edge(&my_graph[0][0], NUM_VERTICES, 6, 4, 8);
  add_edge(&my_graph[0][0], NUM_VERTICES, 4, 6, 8);
  add_edge(&my_graph[0][0], NUM_VERTICES, 5, 6, 12);
  add_edge(&my_graph[0][0], NUM_VERTICES, 8, 5, 11);
  add_edge(&my_graph[0][0], NUM_VERTICES, 3, 8, 9);
  add_edge(&my_graph[0][0], NUM_VERTICES, 3, 0, 13);
  add_edge(&my_graph[0][0], NUM_VERTICES, 5, 1, 6);
  add_edge(&my_graph[0][0], NUM_VERTICES, 9, 8, 23);
  add_edge(&my_graph[0][0], NUM_VERTICES, 4, 1, 14);
```

The Result of the following code is attached below:



It can be seen in the result above that the graph is filled correctly.

Task:2

For this part you will have to complete the function 'find_path_dfs()' which finds the cost of going from the source vertex (*src*) to destination vertex (*dst*) using Depth First Search (DFS) algorithm. You will use a stack for implementing DFS algorithm.

Solution:

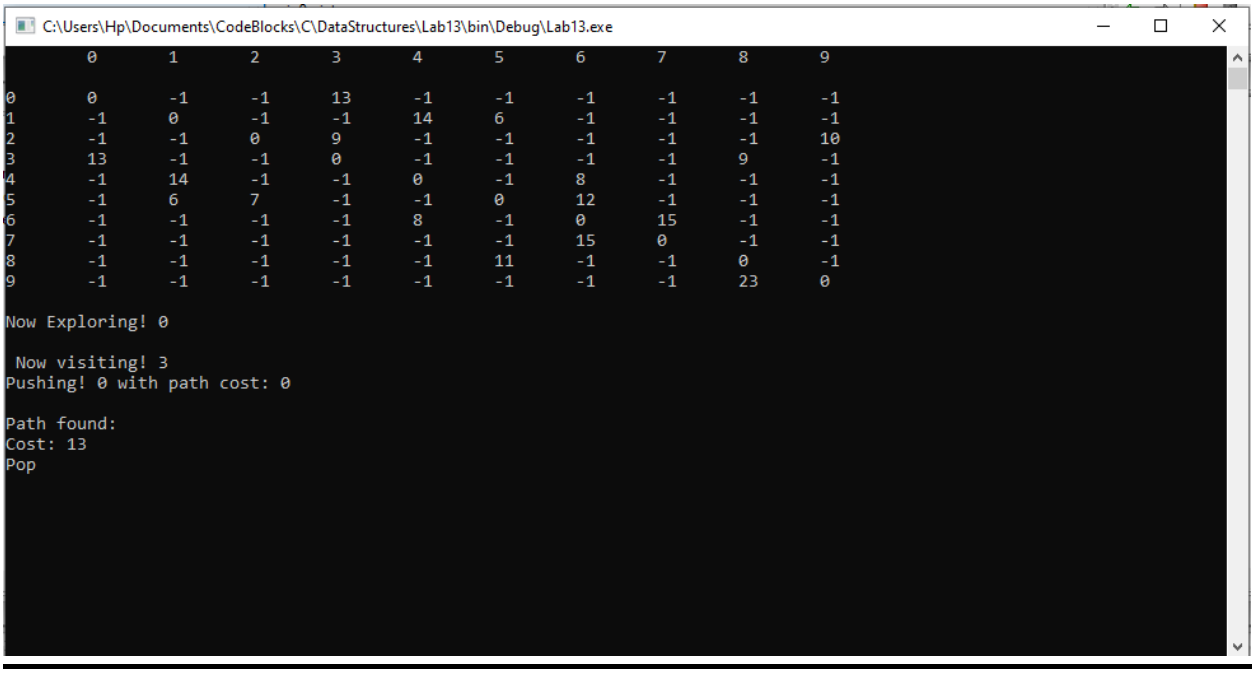
The code is shown below,

```
        else    /// Explore this vertex
        {
        /**     Complete this function. You are free to make any changes to this function.
            But make sure that path cost is correctly found./

        */
        printf("\nNow Exploring! %d\n",crnt_exploring);
        crnt_visiting=0;
        while(crnt_visiting < size)
        {
            if((*(graph+(crnt_exploring*size+crnt_visiting))==0)|| (visited[crnt_visiting]== 1)|| *(graph+(crnt_exploring*size+crnt_visiting))== -1)
                crnt_visiting ++;
            else
                break;
        }

        if (crnt_visiting==size)
        {
            temp=pop(&top);
            crnt_exploring=temp.vertex_num;
            path_cost=temp.cost_to_visit;
        }
else
{
        printf("\n Now visiting! %d\n", crnt_visiting);
        printf("Pushing! %d with path cost: %d\n",crnt_exploring,path_cost);
        temp.vertex_num=crnt_exploring;
        temp.cost_to_visit=path_cost;
        push(&top, temp);
        path_cost += *(graph+(crnt_exploring*size+crnt_visiting));
        crnt_exploring=crnt_visiting;
}
}
}
```

The Result of the following code is attached below:



Post Lab Task.

Task 3:

Complete the code for Task 2 and submit along with a report explaining your implementation.

Solution

Depth First Search

The DFS algorithm is a recursive algorithm that uses the idea of backtracking. It involves exhaustive searches of all the nodes by going ahead, if possible, else by backtracking.

Implementation

The Algorithm I used to complete this function is as follows:

1- Pick a starting node and push all its adjacent nodes into a stack.
2- Pop a node from stack to select the next node to visit and push all its adjacent nodes into a stack.
3- Repeat this process until the stack is empty. However, ensure that the nodes that are visited are marked. This will prevent you from visiting the same node more than once. If you do not mark the nodes that are visited and you visit the same node more than once, you may end up in an infinite loop.

Conclusion:

In this lab, we implemented the insertion function of a graph by giving source node, destination node and path cost, we also completed the Depth First Search function which finds the cost of going from source to destination, furthermore in post lab I have briefly explained my implementation of Depth First Search function.

THE END