Nithin S
221IT085

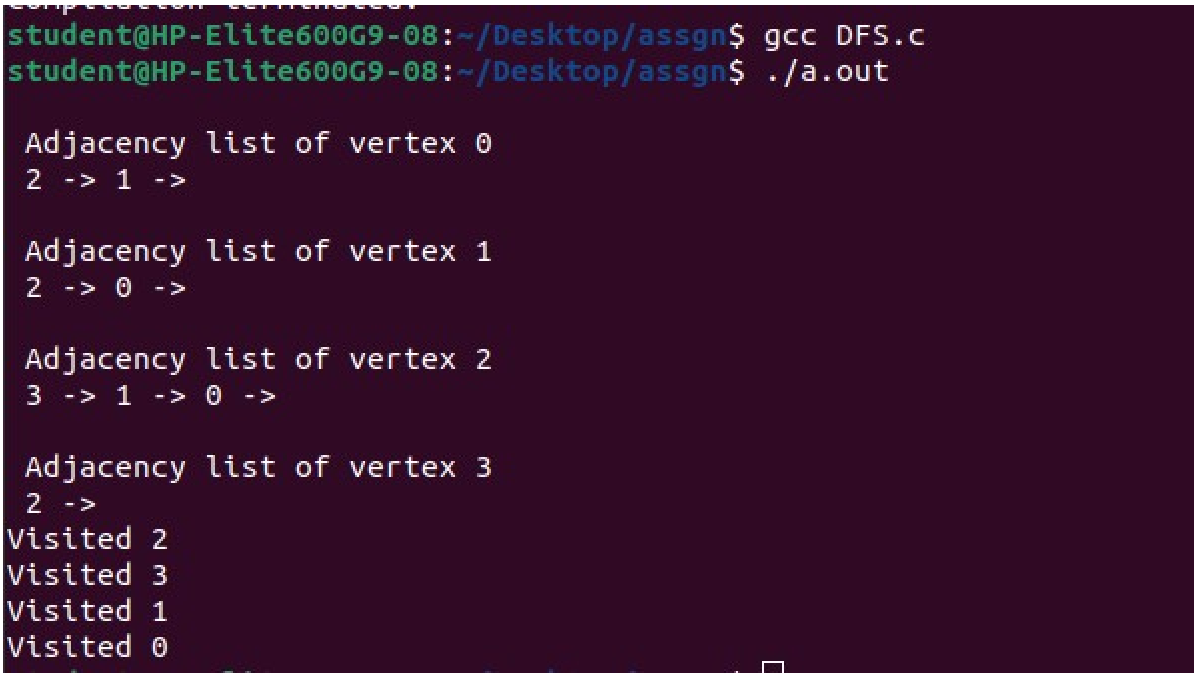
IT206 Lab Assignment

Q1) Implement DFS

CODE

```
#include <stdio.h>
#include <stdlib.h>

struct node
{
  int vertex;
  struct node* next;
};

struct node* createNode(int v);

struct Graph
{
  int numVertices;
  int* visited;
  struct node** adjLists;
};

void DFS(struct Graph* graph, int vertex) {
  struct node* adjList = graph->adjLists[vertex];
  struct node* temp = adjList;

  graph->visited[vertex] = 1;
  printf("Visited %d \n", vertex);
  while (temp != NULL) {
    int connectedVertex = temp->vertex;

    if (graph->visited[connectedVertex] == 0) {
      DFS(graph, connectedVertex);
    }
    temp = temp->next;
  }
}

struct node* createNode(int v) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->vertex = v;
  newNode->next = NULL;
  return newNode;
}


struct Graph* createGraph(int vertices) {
  struct Graph* graph = malloc(sizeof(struct Graph));
  graph->numVertices = vertices;

  graph->adjLists = malloc(vertices * sizeof(struct
node*));

  graph->visited = malloc(vertices * sizeof(int));

  int i;
  for (i = 0; i < vertices; i++) {
    graph->adjLists[i] = NULL;
    graph->visited[i] = 0;
  }
  return graph;
}


void addEdge(struct Graph* graph, int src, int dest)
{
  struct node* newNode = createNode(dest);
  newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;
  newNode = createNode(src);
  newNode->next = graph->adjLists[dest];
  graph->adjLists[dest] = newNode;
}

void printGraph(struct Graph* graph) {
  int v;
  for (v = 0; v < graph->numVertices; v++) {
    struct node* temp = graph->adjLists[v];
    printf("\n Adjacency list of vertex %d\n ", v);
    while (temp) {
      printf("%d -> ", temp->vertex);
      temp = temp->next;
    }
    printf("\n");
  }
}

int main() {
  struct Graph* graph = createGraph(4);
  addEdge(graph, 0, 1);
  addEdge(graph, 0, 2);
  addEdge(graph, 1, 2);
  addEdge(graph, 2, 3);

  printGraph(graph);

  DFS(graph, 2);

  return 0;
}
```

OUTPUT



```
compilation terminated.
student@HP-Elite600G9-08:~/Desktop/assgn$ gcc DFS.c
student@HP-Elite600G9-08:~/Desktop/assgn$ ./a.out

 Adjacency list of vertex 0
 2 -> 1 ->

 Adjacency list of vertex 1
 2 -> 0 ->

 Adjacency list of vertex 2
 3 -> 1 -> 0 ->

 Adjacency list of vertex 3
 2 ->
Visited 2
Visited 3
Visited 1
Visited 0
```

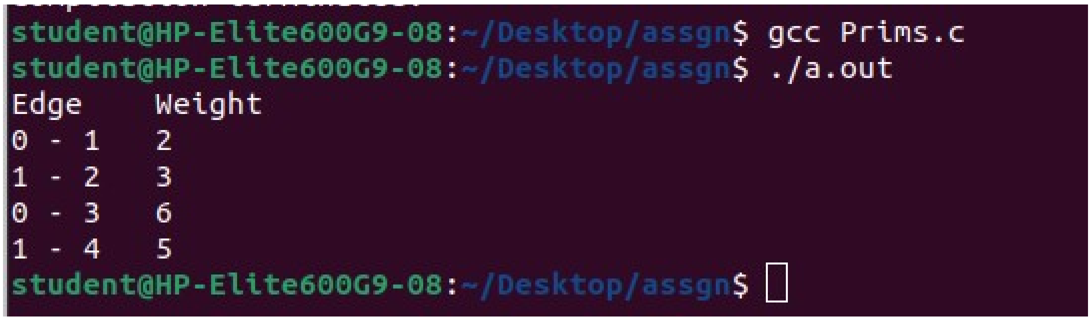
Q2) Implement Prims

CODE

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>

#define V 5


int minKey(int key[], bool mstSet[])
{

    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;

    return min_index;
}
```

```
int printMST(int parent[], int graph[V][V])
{
    printf("Edge \tWeight\n");
    for (int i = 1; i < V; i++)
        printf("%d - %d \t%d \n", parent[i], i,
            graph[i][parent[i]]);
}


void primMST(int graph[V][V])
{

    int parent[V];
    int key[V];
    bool mstSet[V];

    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;


    key[0] = 0;


    parent[0] = -1;


    for (int count = 0; count < V - 1; count++) {


        int u = minKey(key, mstSet);

        mstSet[u] = true;


        for (int v = 0; v < V; v++)


            if (graph[u][v] && mstSet[v] == false
                && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }
    printMST(parent, graph);
}

int main()
{
    int graph[V][V] = { { 0, 2, 0, 6, 0 },
                        { 2, 0, 3, 8, 5 },
                        { 0, 3, 0, 0, 7 },
                        { 6, 8, 0, 0, 9 },
                        { 0, 5, 7, 9, 0 } };


    primMST(graph);

    return 0;
}
```

OUTPUT



```
student@HP-Elite600G9-08:~/Desktop/assgn$ gcc Prims.c
student@HP-Elite600G9-08:~/Desktop/assgn$ ./a.out
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
student@HP-Elite600G9-08:~/Desktop/assgn$ 
```

Q3) Implement Kruskals

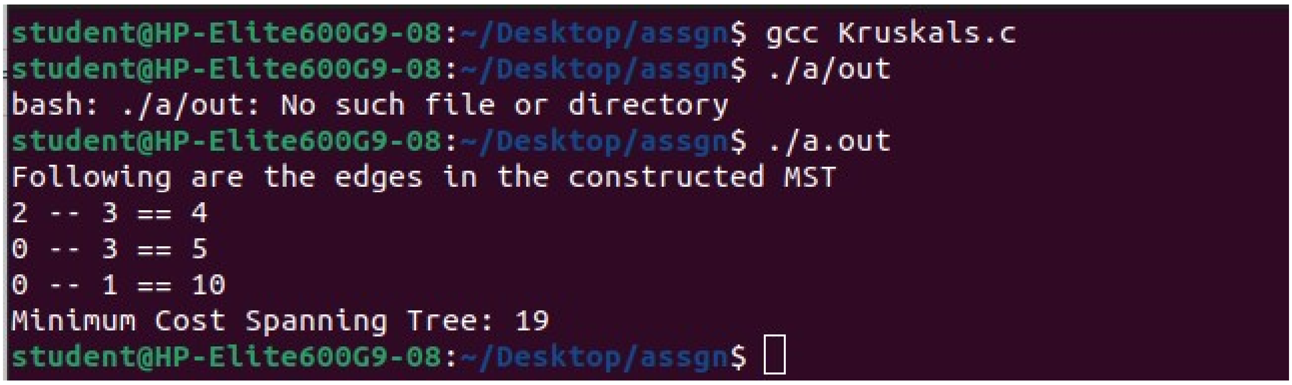
CODE

```
#include <stdio.h>
#include <stdlib.h>
```

```
int comparator(const void* p1, const void* p2)
{
    const int(*x)[3] = p1;
    const int(*y)[3] = p2;

    return (*x)[2] - (*y)[2];
}

void makeSet(int parent[], int rank[], int n)
{
    for (int i = 0; i < n; i++) {
        parent[i] = i;
        rank[i] = 0;
    }
}

int findParent(int parent[], int component)
{
    if (parent[component] == component)
        return component;

    return parent[component]
        = findParent(parent, parent[component]);
}

void unionSet(int u, int v, int parent[], int rank[],
int n)
{

    u = findParent(parent, u);
    v = findParent(parent, v);

    if (rank[u] < rank[v]) {
        parent[u] = v;
    }
    else if (rank[u] > rank[v]) {
        parent[v] = u;
    }
    else {
        parent[v] = u;
        rank[u]++;
    }
}

void kruskalAlgo(int n, int edge[n][3])
{

    qsort(edge, n, sizeof(edge[0]), comparator);

    int parent[n];
    int rank[n];

    makeSet(parent, rank, n);

    int minCost = 0;

    printf(
        "Following are the edges in the constructed
MST\n");
    for (int i = 0; i < n; i++) {
        int v1 = findParent(parent, edge[i][0]);
        int v2 = findParent(parent, edge[i][1]);
        int wt = edge[i][2];


        if (v1 != v2) {
            unionSet(v1, v2, parent, rank, n);
            minCost += wt;
            printf("%d -- %d == %d\n", edge[i][0],
                edge[i][1], wt);
        }
    }

    printf("Minimum Cost Spanning Tree: %d\n",
minCost);
}


int main()
```

```
{
    int edge[5][3] = { { 0, 1, 10 },
                       { 0, 2, 6 },
                       { 0, 3, 5 },
                       { 1, 3, 15 },
                       { 2, 3, 4 } };

    kruskalAlgo(5, edge);

    return 0;
}
```

OUTPUT



```
student@HP-Elite600G9-08:~/Desktop/assgn$ gcc Kruskals.c
student@HP-Elite600G9-08:~/Desktop/assgn$ ./a/out
bash: ./a/out: No such file or directory
student@HP-Elite600G9-08:~/Desktop/assgn$ ./a.out
Following are the edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Cost Spanning Tree: 19
student@HP-Elite600G9-08:~/Desktop/assgn$ 
```