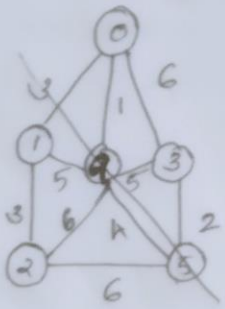
DATA STRUCTURE LAB EXAMINATION

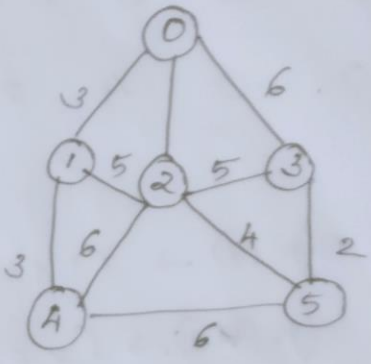
SUBMITTED BY
SREELEKSHMI PRATHAPAN
ROLL NO 40
MCA TKMCE KOLLAM

Data Structures Lab.

Sreelakshmi prathapan
Roll No: 40
TKMCE kollam.

Develop a program to generate a minimum spanning tree using kruscals algorithm for the given graph. and Compute total cost





Algorithm

KRUSKAL(G):

Step 1: A = ϕ

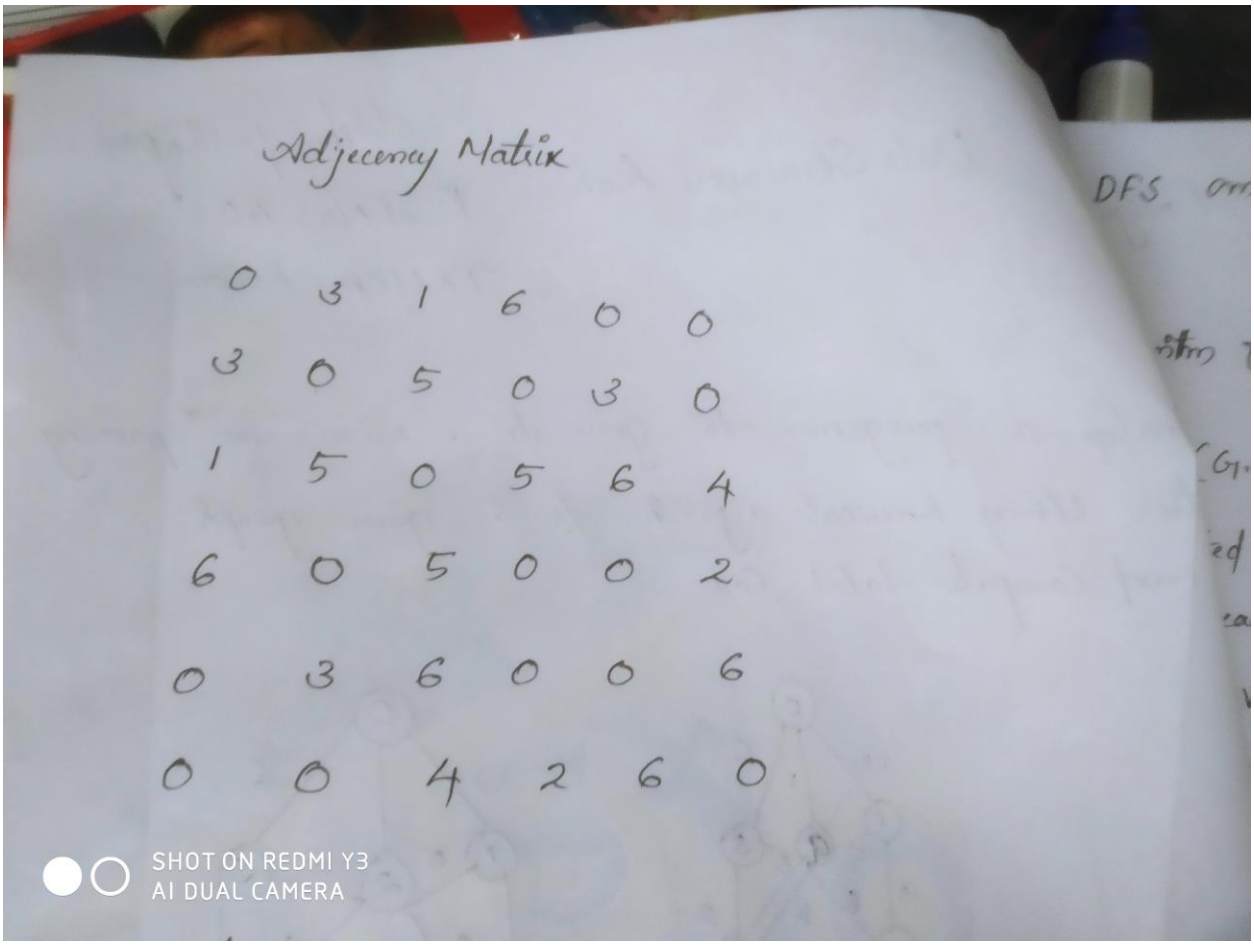
Step 2: for each vertex v ∈ G.V:
MAKE-SET(v)

Step 3: For each (u,v) ∈ G.E ordered by increasing order of the weight (u,v):

if FIND-SET(u) ≠ FIND SET(v):
A = A ∪ {(u,v)}.

UNION(u, v)

Step 4: Return A.



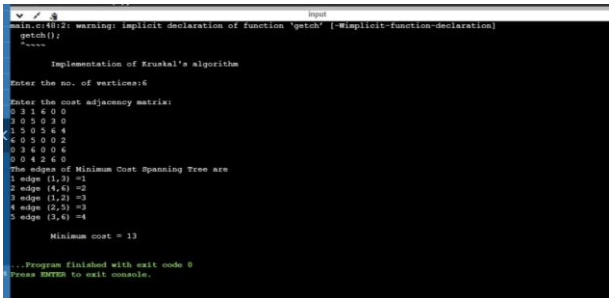
SOURCE CODE

```
#include<stdio.h>
#include<stdlib.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
void main()
{

        printf("\n\tImplementation of Kruskal's algorithm\n");
        printf("\nEnter the no. of vertices:");
        scanf("%d",&n);
        printf("\nEnter the cost adjacency matrix:\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
```

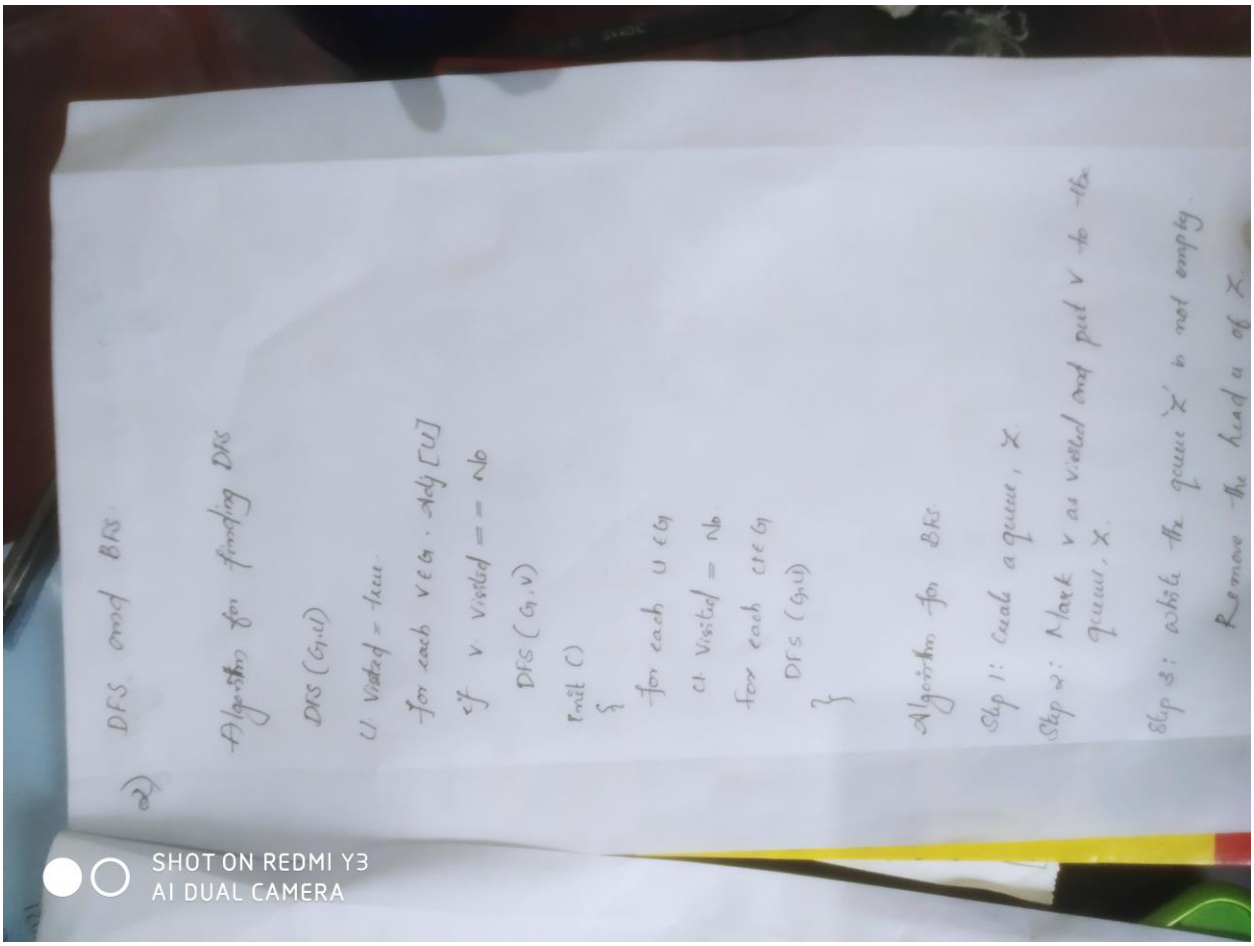
```
                scanf("%d",&cost[i][j]);
                if(cost[i][j]==0)
                        cost[i][j]=999;
        }
    }
    printf("The edges of Minimum Cost Spanning Tree are\n");
    while(ne < n)
    {
        for(i=1,min=999;i<=n;i++)
        {
            for(j=1;j <= n;j++)
            {
                if(cost[i][j] < min)
                {
                    min=cost[i][j];
                    a=u=i;
                    b=v=j;
                }
            }
        }
        u=find(u);
        v=find(v);
        if(uni(u,v))
        {
            printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
            mincost +=min;
        }
        cost[a][b]=cost[b][a]=999;
    }
    printf("\n\tMinimum cost = %d\n",mincost);
    getch();
}
int find(int i)
{
    while(parent[i])
    i=parent[i];
    return i;
}
int uni(int i,int j)
{
    if(i!=j)
    {
        parent[j]=i;
        return 1;
```

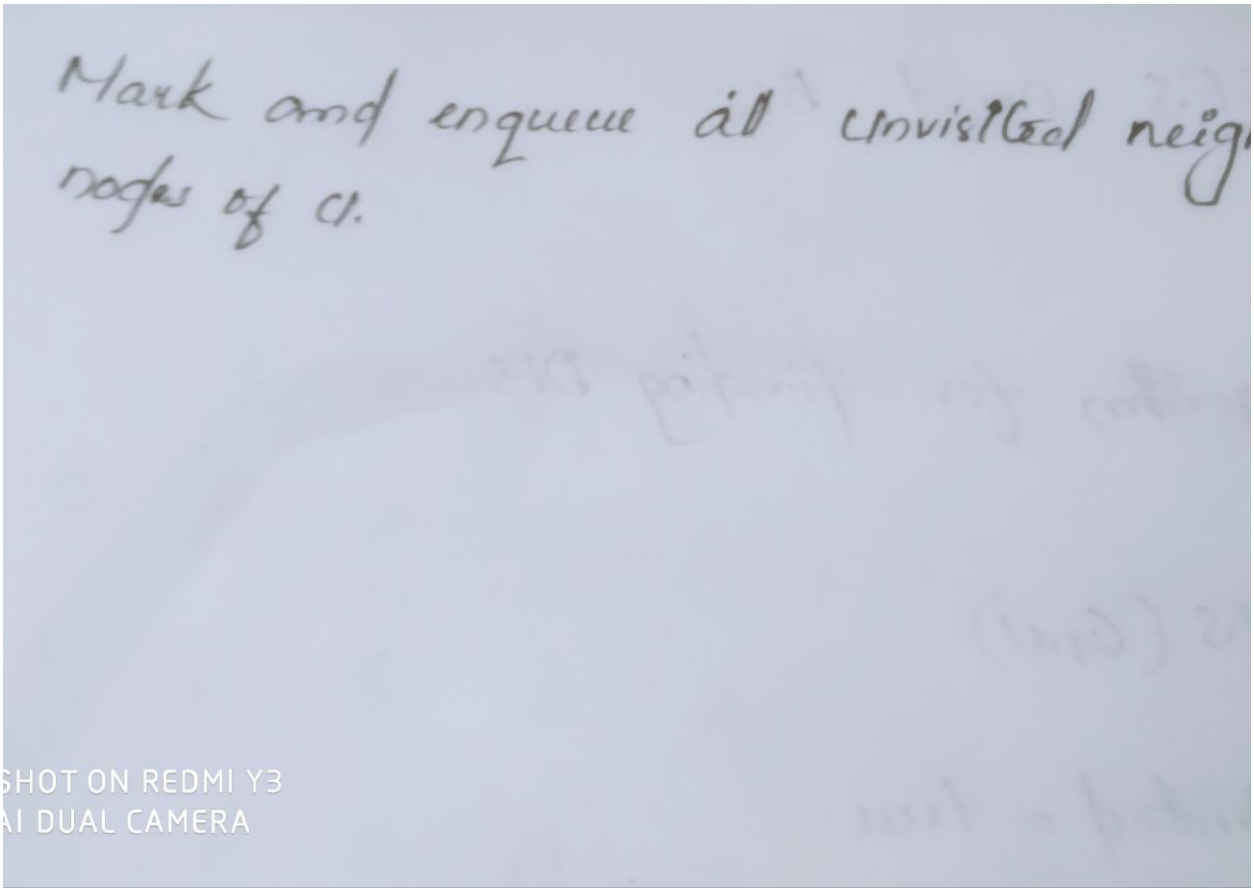
```
        }
        return 0;
}
```

Out put



Second question





Source code for BFS

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

#define MAX 10

//vertex count
int vertex_count =0;

// vertex definitions
struct vertex{
        char data;
        bool visited;
};

//array of vertices
struct vertex *graph[MAX];
```

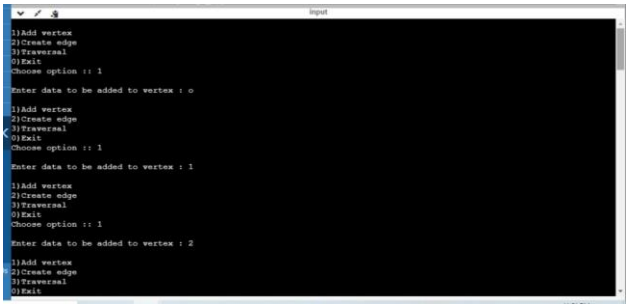
```
//adjacency matrix
int adj_matrix[MAX][MAX];

//Queuee

int queue[MAX];
int rear=-1;
int front=0;
int queue_count=0;

void enqueue(int data){
	queue[++rear]=data;
	queue_count++;
}

int dequeue(){
	queue_count--;
	return queue[front++];
}

bool is_queue_empty(){
	return queue_count == 0;
}

//add vertex to the vertex list
void add_vertex(char data){
	struct vertex *new = (struct vertex*)malloc(sizeof(struct vertex));
	new->data = data;
	new->visited = false;
	graph[vertex_count]=new;
	vertex_count++;
}

//add edge to edge array
void add_edge(int start,int end){
	adj_matrix[start][end]=1;
	adj_matrix[end][start]=1;
}

// to return adjacent vertex
int adj_vertex(int vertex_get){
	int i;
	for(i=0;i<vertex_count;i++){
		if(adj_matrix[vertex_get][i] == 1 && graph[i]->visited == false){
```

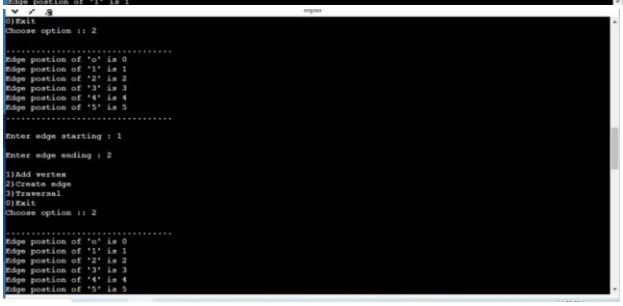
```
                        return i;
                }
        }
        return -1;
}

// to display vertex value
void display_vertex(int pos){
        printf("%c -> ",graph[pos]->data);
}

void bfs(struct vertex *new,int start){
        if(!new){
                printf("\nNothing to display\n");
                return;
        }

        int i;
        int unvisited;

        printf("\n||||||||||||||||||||||||||||||\n");

        new->visited =true;
        display_vertex(start);
        enqueue(start);

        while(!is_queue_empty()){
                int pop_vertex = dequeue();
                //printf("\npoped : %d",pop_vertex);
                while((unvisited = adj_vertex(pop_vertex))!=-1){
                        graph[unvisited]->visited = true;
                        display_vertex(unvisited);
                        enqueue(unvisited);
                }
        }
        printf("\n||||||||||||||||||||||||||||||\n");

        for(i=0;i<vertex_count;i++){
                graph[i]->visited = false;
        }
}

void show(){
        int i;
```

```
	printf("\n.....\n");
	for(i=0;i<vertex_count;i++){
		printf("Edge postion of '%c' is %d\n",graph[i]->data,i);
	}
	printf(".....\n");
}

int main(){
	int opt;
	char data;
	int edge_1,edge_2;
	int i, j;
	int start;

   for(i = 0; i < MAX; i++)    // set adjacency
     for(j = 0; j < MAX; j++) // matrix to 0
       adj_matrix[i][j] = 0;

	do{
		printf("\n1)Add vertex \n2)Create edge \n3)Traversal \n0)Exit \nChoose option ::
");
		scanf("%d",&opt);
		switch(opt){
			case 1:
				printf("\nEnter data to be added to vertex : ");
				scanf(" %c", &data);
				add_vertex(data);
				break;
			case 2:
		show();
				printf("\nEnter edge starting : ");
				scanf("%d",&edge_1);
				printf("\nEnter edge ending : ");
				scanf("%d",&edge_2);
				if(vertex_count-1 < edge_1 || vertex_count-1 < edge_2){
					printf("\nThere is no vertex !!\n");
				}
				else{
					add_edge(edge_1,edge_2);
				}
				break;
			case 3:
				printf("\nEnter starting vertex position : ");
				scanf("%d",&start);
```

```
                                bfs(graph[start],start);
                                break;
                        default:
                                printf("\nInvalid option try again !! ...");
                        }
        }while(opt!=0);

        return 0;
}
```

Out put



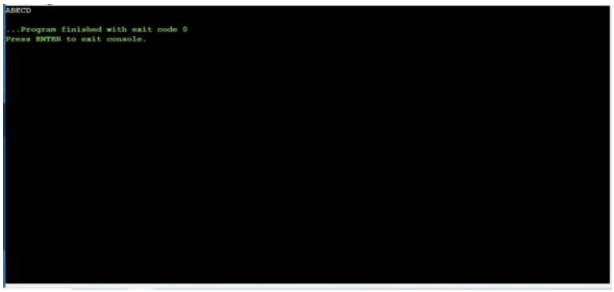


Source code forBFS DFS

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

#define MAX 6

//vertex count
int vertex_count =0;

// vertex definitions
struct vertex{
	char data;
	bool visited;
};

//array of vertices
struct vertex *graph[MAX];

//adjacency matrix
int adj_matrix[MAX][MAX];

//stack

int stack[MAX];
int top = -1;

void push(int data){
	stack[++top]=data;
}

int pop(){
	return stack[top--];
}

int peek(){
	return stack[top];
}

bool is_stack_empty(){
	return top == -1;
```

```
}

//add vertex to the vertex list
void add_vertex(char data){
	struct vertex *new = (struct vertex*)malloc(sizeof(struct vertex));
	new->data = data;
	new->visited = false;
	graph[vertex_count]=new;
	vertex_count++;
}

//add edge to edge array
void add_edge(int start,int end){
	adj_matrix[start][end]=1;
	adj_matrix[end][start]=1;
}

// to return adjacent vertex
int adj_vertex(int vertex_get){
	int i;
	for(i=0;i<vertex_count;i++){
		if(adj_matrix[vertex_get][i] == 1 && graph[i]->visited == false){
			return i;
		}
	}
	return -1;
}

// to display vertex value
void display_vertex(int pos){
	printf("%c",graph[pos]->data);
}

void dfs(){
	int i;
	int unvisited;

	graph[0]->visited =true;
	display_vertex(0);
	push(0);

	while(!is_stack_empty()){
		int unvisited = adj_vertex(peek());
```

```
            if(unvisited == -1){
                  pop();
            }
            else{
                  graph[unvisited]->visited = true;
                        display_vertex(unvisited);
                        push(unvisited);
            }
      }

      for(i=0;i<vertex_count;i++){
            graph[i]->visited = false;
      }
}

int main(){
      int i, j;

   for(i = 0; i < MAX; i++)    // set adjacency
      for(j = 0; j < MAX; j++) // matrix to 0
         adj_matrix[i][j] = 0;


      add_vertex('A');
      add_vertex('B');
      add_vertex('C');
      add_vertex('D');
      add_vertex('E');


      add_edge(0,1);
      add_edge(0,2);
      add_edge(0,3);
      add_edge(1,4);
      add_edge(2,4);
      add_edge(3,4);

      dfs();

      return 0;
}
```


ABECD
...Program finished with exit code 0
Press ENTER to exit console.