DATA STRUCTURE LAB EXAM

SINDOORI B

REG NO : TKM20MCA-2038

S1 -136

QUESTION 2

Develop a program to generate a minimum spanning tree using Kruskal's Algorithm for the given graph and computes the total cost.

ALGORITHM

Step

1: Start

2: Create a graph

3: Make an adjacency matrix for that graph

4: Sort all the edges from low weight to high

5: Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.

6: Add the weigth ti mincost.

7:Keep adding edges until we reach all vertices.

8: Stop

PROGRAM CODE

```
#include<stdio.h>

#define MAX 30


typedef struct edge

{

int u,v,w;

}edge;


typedef struct edgelist

{

edge data[MAX];

int n;

}edgelist;


edgelist elist;


int G[MAX][MAX],n;

edgelist spanlist;


void kruskal();

int find(int belongs[],int vertexno);

void union1(int belongs[],int c1,int c2);

void sort();

void print();
```

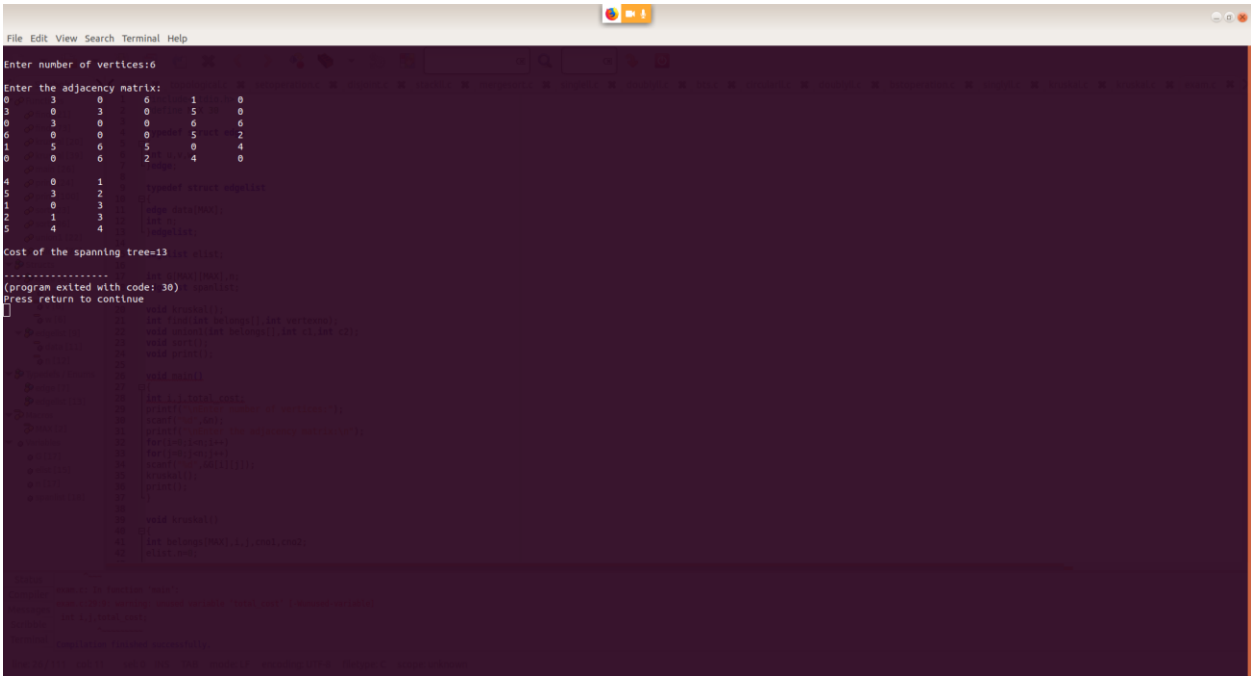
```
void main()
{
int i,j,total_cost;
printf("\nEnter number of vertices ");
scanf("%d",&n);
printf("\nEnter the adjacency matrix \n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
kruskal();
print();
}

void kruskal()
{
int belongs[MAX],i,j,cno1,cno2;
elist.n=0;

for(i=1;i<n;i++)
for(j=0;j<i;j++)
{
if(G[i][j]!=0)
{
elist.data[elist.n].u=i;
```

```
elist.data[elist.n].v=j;
elist.data[elist.n].w=G[i][j];
elist.n++;
}
}

sort();
for(i=0;i<n;i++)
belongs[i]=i;
spanlist.n=0;
for(i=0;i<elist.n;i++)
{
cno1=find(belongs,elist.data[i].u);
cno2=find(belongs,elist.data[i].v);
if(cno1!=cno2)
{
spanlist.data[spanlist.n]=elist.data[i];
spanlist.n=spanlist.n+1;
union1(belongs,cno1,cno2);
}
}
}

int find(int belongs[],int vertexno)
{
```

```
return(belongs[vertexno]);
}

void union1(int belongs[],int c1,int c2)
{
int i;
for(i=0;i<n;i++)
if(belongs[i]==c2)
belongs[i]=c1;
}

void sort()
{
int i,j;
edge temp;
for(i=1;i<elist.n;i++)
for(j=0;j<elist.n-1;j++)
if(elist.data[j].w>elist.data[j+1].w)
{
temp=elist.data[j];
elist.data[j]=elist.data[j+1];
elist.data[j+1]=temp;
}
}
```

```
void print()
{
int i,cost=0;
for(i=0;i<spanlist.n;i++)
{
printf("\n%d\t%d\t%d",spanlist.data[i].u,spanlist.data[i].v,spanlist.data[i].w);
cost=cost+spanlist.data[i].w;
}

printf("\n\nCost of the spanning tree=%d",cost);
}
```

OUTPUT



QUESTION 2

Develop a programs to implement DFS and BFS

ALGORITHM BFS

STEPS

1: Start

2: Create a graph

3: Create adjacency matrix

4: Start by putting any one of the graph's vertices at the back of a queue.

5: Take the front item of the queue and add it to the visited list.

6: Create a list of that vertex's adjacent nodes.

7:Add the ones which aren't in the visited list to the back of the queue.

8: Keep repeating steps 5,6 and 7until the queue is empty.

9: Stop

ALGORITHM DFS

STEPS

1: Start

2: Create a graph

3: Create adjacency matrix

4: Start by putting any one of the graph's vertices on top of a stack.

5: Take the top item of the stack and add it to the visited list.

6: Create a list of that vertex's adjacent nodes.

7 Add the ones which aren't in the visited list to the top of the stack.

8: Keep repeating steps 5,5 and 7 until the stack is empty.

PROGRAM CODE

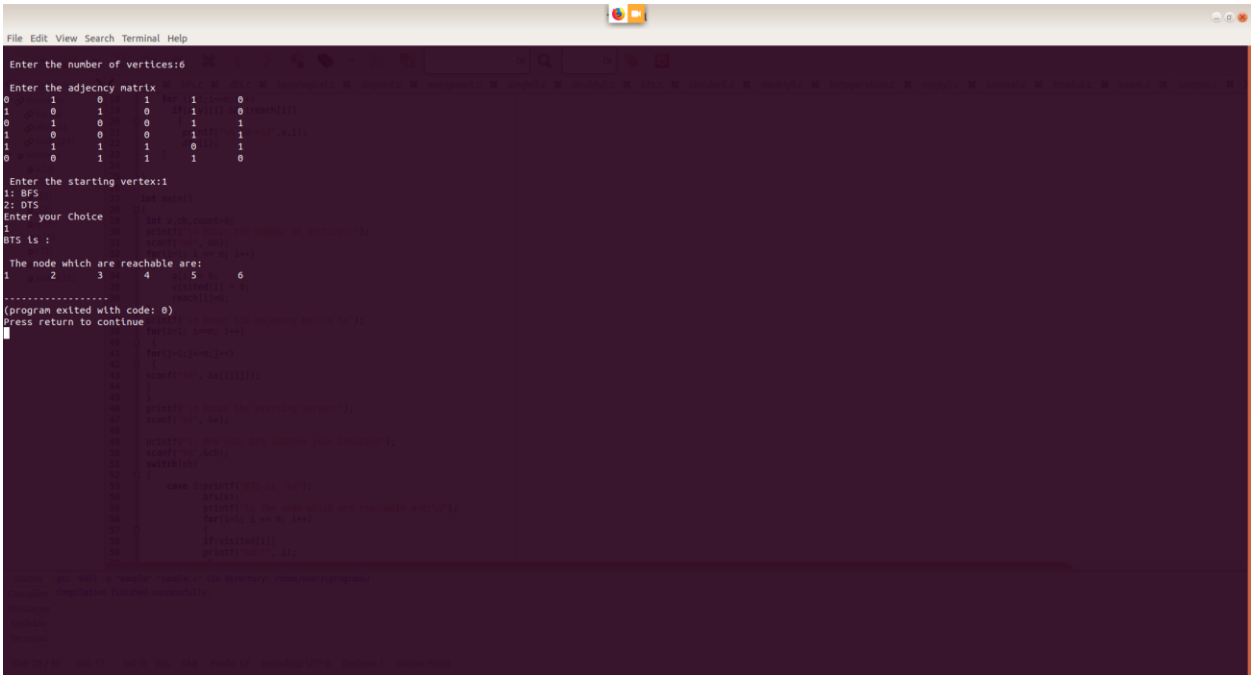
```
#include<stdio.h>
int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1,reach[20];


void bfs(int v)
 {
 for(i = 1; i <= n; i++)
 if(a[v][i] && !visited[i])
 q[++r] = i;
 if(f <= r) {
 visited[q[f]] = 1;
 bfs(q[f++]);
 }
}
void dfs(int v)
{
        int i;
        reach[v]=1;
        for (i=1;i<=n;i++)
          if(a[v][i] && !reach[i])
            {
                printf("\n %d->%d",v,i);
                dfs(i);
        }

}
```

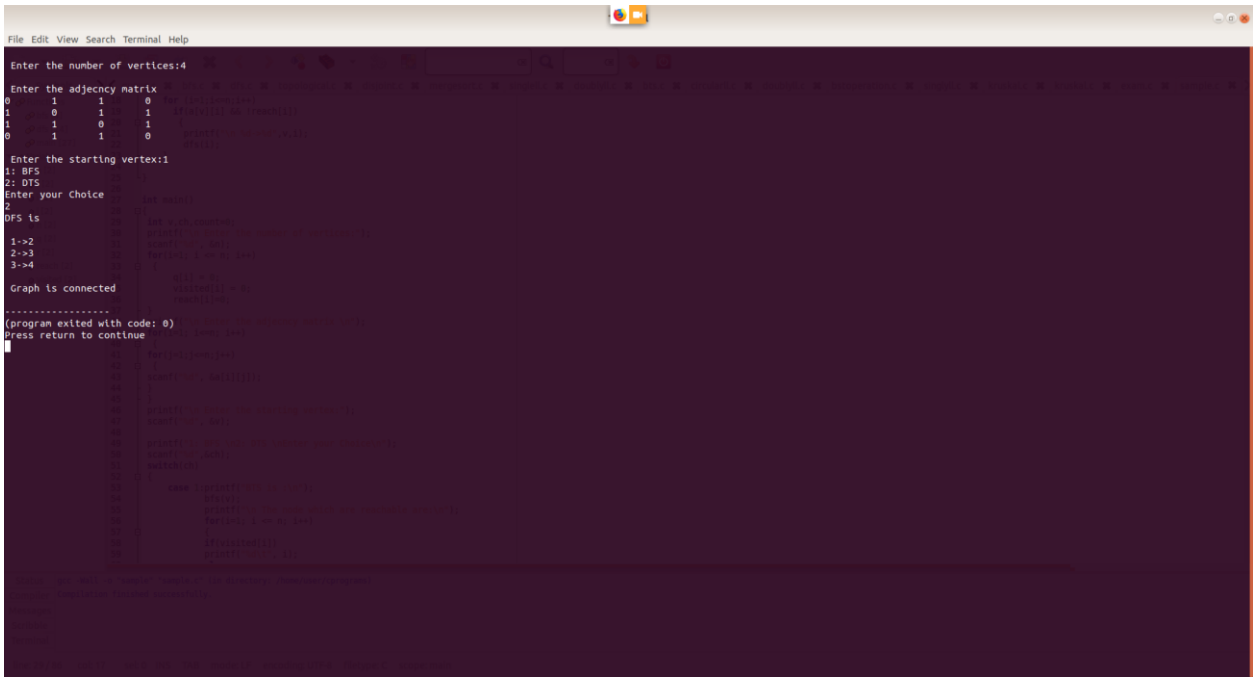
```
int main()
{
 int v,ch,count=0;
 printf("\n Enter the number of vertices:");
 scanf("%d", &n);
 for(i=1; i <= n; i++)
  {
            q[i] = 0;
            visited[i] = 0;
            reach[i]=0;
 }
 printf("\n Enter the adjecncy matrix \n");
 for(i=1; i<=n; i++)
  {
 for(j=1;j<=n;j++)
  {
 scanf("%d", &a[i][j]);
 }
 }
 printf("\n Enter the starting vertex:");
 scanf("%d", &v);

 printf("1: BFS \n2: DTS \nEnter your Choice\n");
 scanf("%d",&ch);
```

```
switch(ch)
{
        case 1:printf("BTS is :\n");
                        bfs(v);
                        printf("\n The node which are reachable are:\n");
                        for(i=1; i <= n; i++)
                        {
                        if(visited[i])
                        printf("%d\t", i);
                        else
                        {
                        printf("\n Bfs is not possible. Not all nodes are reachable");
                        break;
                        }
                        }
                        break;

      case 2: printf("DFS is \n");
                        dfs(1);
                        printf("\n");
                        for (i=1;i<=n;i++)
                                {
                                        if(reach[i])
                                        count++;
                                }
```

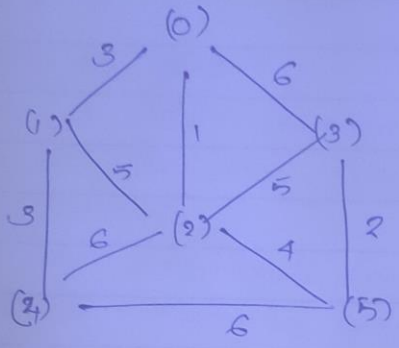
```
                    if(count==n)
                    printf("\n Graph is connected"); else
                    printf("\n Graph is not connected");


                    break;
        default :printf("inavlid entry");
}
}
```

OUTPUT

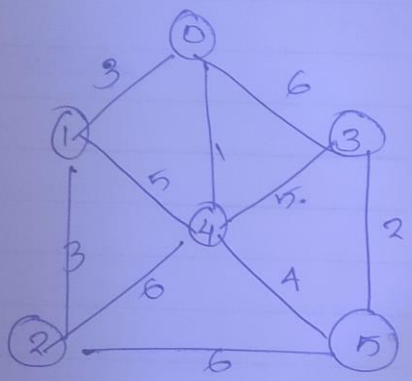



File Edit View Search Terminal Help
Enter the number of vertices:4
Enter the adjecncy matrix
0 1 1 0
1 0 1 1
1 1 0 1
0 1 1 0
Enter the starting vertex:1
1: BFS
2: DTS
Enter your Choice
2
DFS is
1->2
2->3
3->4
Graph is connected
(program exited with code: 0)
Press return to continue

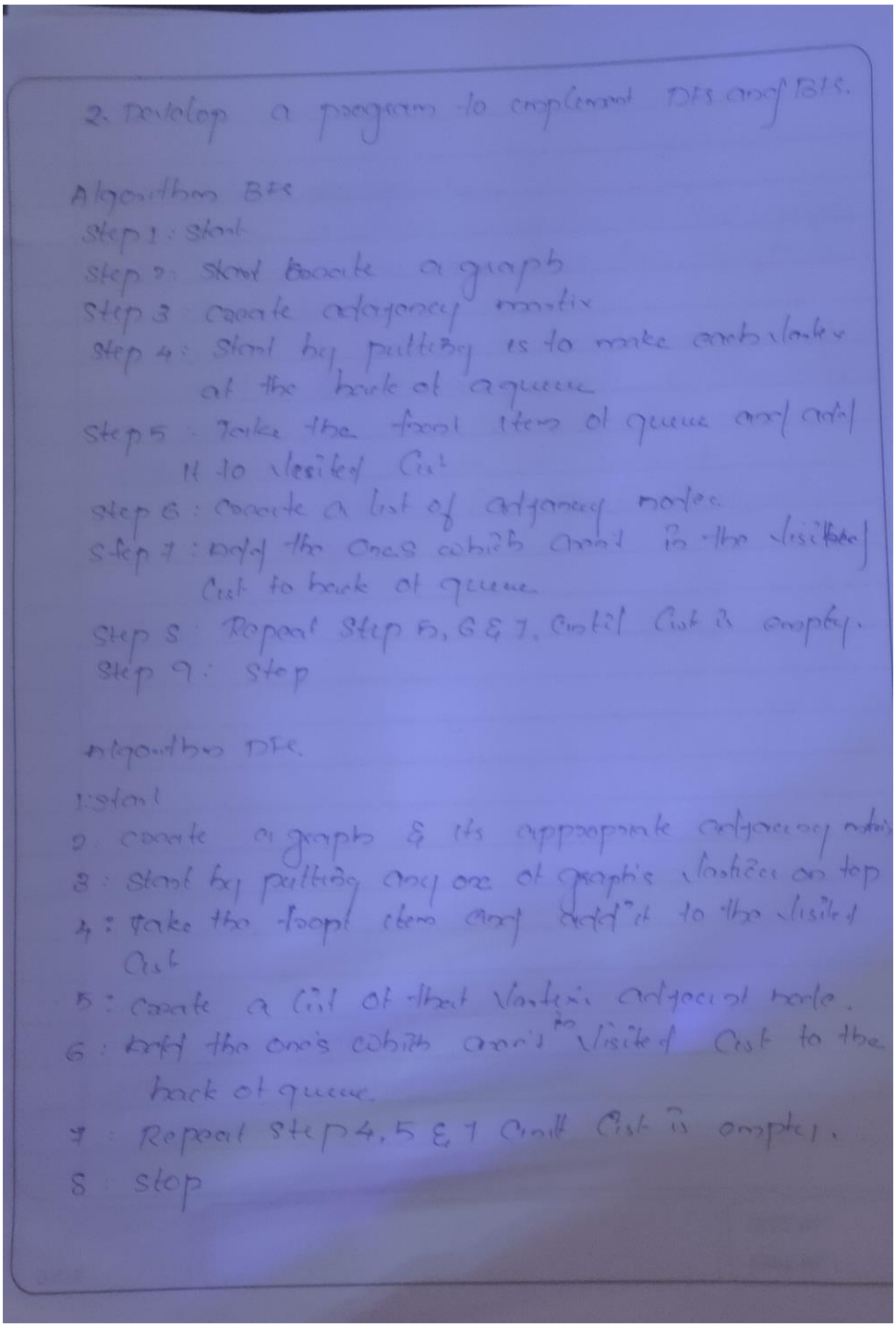
1. Develop a program to generate a minimum spanning tree using kruskal algorithm for the given graph and compute the total cost.



	0	1	2	3	4
0	0	3	1	6	0
1	3	0	5	0	3
2	1				
3					
4					
5					



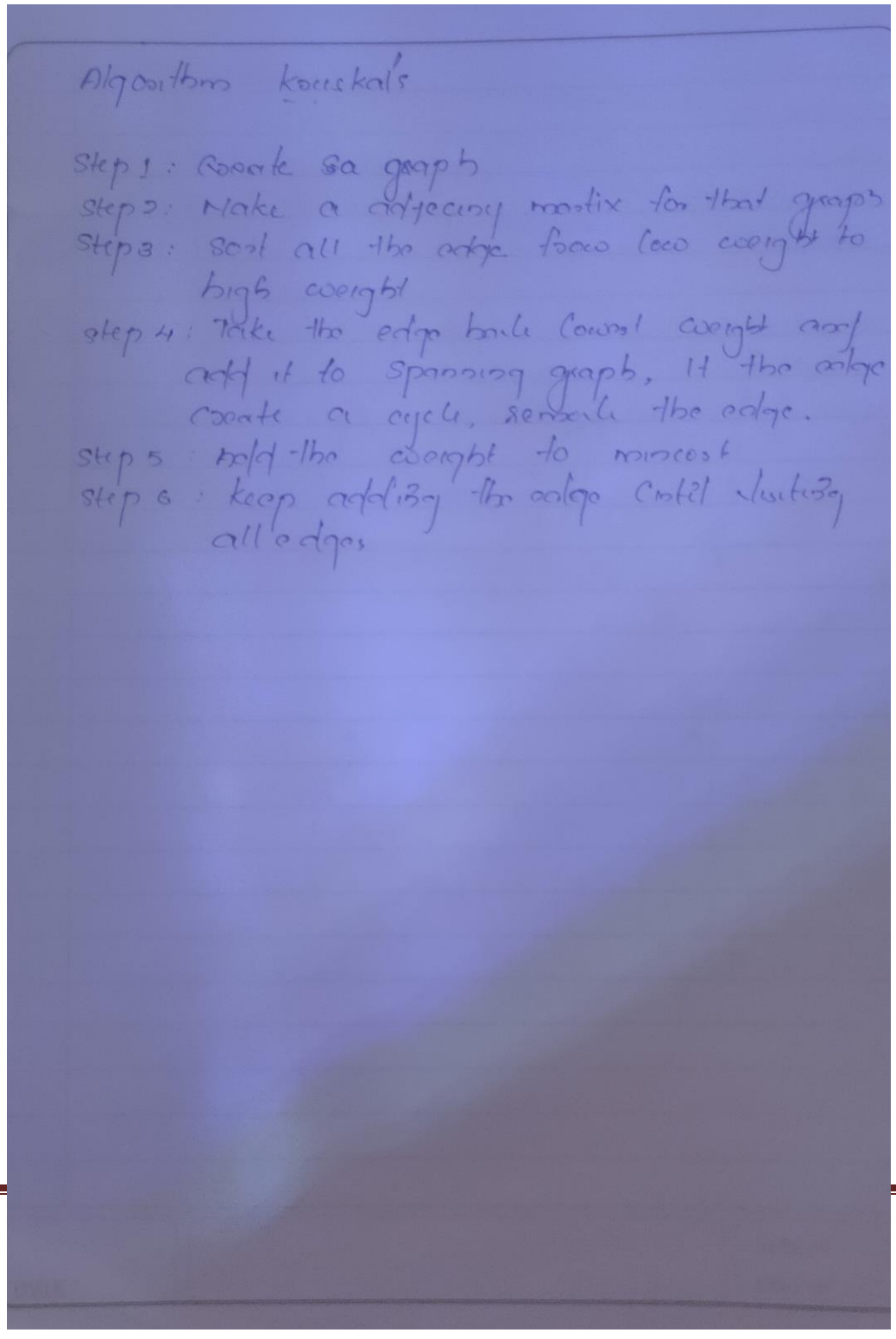
	0	1	2	3
0	0	3	0	6
1	3	0	3	0
2	0	3	0	0
3	6	0	0	0
4	1	5	6	5
5	0	0	6	2



2. Develop a program to implement DFS and BFS.

Algorithm BFS
Step 1: start
Step 2: Start Create a graph
Step 3: Create adjacency matrix
Step 4: Start by putting it to make each vertex at the back of queue
Step 5: Take the front item of queue and add it to visited list
Step 6: Create a list of adjacency nodes.
Step 7: Add the ones which aren't in the visited list to back of queue
Step 8: Repeat Step 5, 6 & 7, Until list is empty.
Step 9: Stop

Algorithm DFS.
1: start
2: create a graph & its appropriate adjacency matrix
3: Start by putting any one of graph's vertices on top
4: Take the front item and add it to the visited list
5: Create a list of that vertex adjacent node.
6: Add the one's which aren't in visited list to the back of queue.
7: Repeat step 4, 5 & 7 until list is empty.
8: stop



Algorithm kruskal's

Step 1 : Create a graph
Step 2 : Make a adjacency matrix for that graph
Step 3 : Sort all the edge from low weight to high weight
Step 4 : Take the edge having lowest weight and add it to spanning graph, if the edge create a cycle, remove the edge.
Step 5 : Add the weight to mincost
Step 6 : Keep adding the edge until visiting all edges