Course-Design And Analysis Of Algorithms

January 3, 2024

lab10 assignment kruskal's algorithm Problem code in c

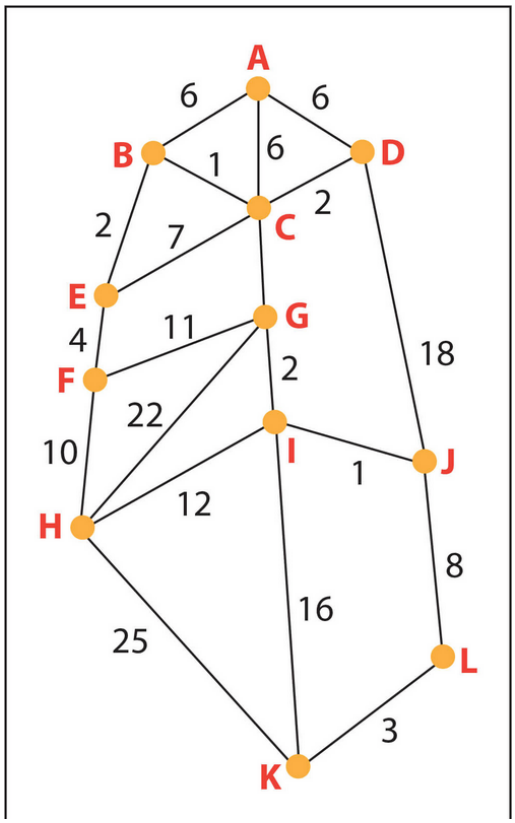
```c
/* Copyright Notice: Confidential - For Evaluation Only */

// This work is protected by copyright and is submitted for evaluation in
//  [DAA]. It is intended solely for
// the purpose of assessment and is not to be shared with
// other students or third parties.

// ------------------------------------------------------------------------


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

static int flag=1;

typedef struct connectset pivot; //connect set has three parts--
// from node,weight of edge between, and to node

struct connectset{
    int fromnode;
    int edgeweight;
    int tonode;
};
int partition(struct connectset* arr,int low,int high);
void qusort(struct connectset* arr,int low,int high);

int partition(struct connectset* arr,int low,int high)//a=0 reward sort,a=1 tonode sort
{
    int i=low;
    int j=low;
    struct connectset pivot=arr[high];
    struct connectset temp =arr[low];//just initialization
        for(int j=low;j<=high-1;j++)
        {
            if(arr[j].edgeweight>=pivot.edgeweight)
            {
                temp=arr[j];
                arr[j]=arr[i];
                arr[i]=temp;
                i=i+1;
            }
        }
        arr[high]=arr[i];
        arr[i]=pivot;
    return i;
}

void qusort(struct connectset* arr,int low,int high)
{
    if((low<high)&&(low>=0)&&(high>=0))
    {
        int j=partition(arr,low,high);
        qusort(arr,low,j-1);
        qusort(arr,j+1,high);
    }
}
int num_node(int** arr)
{
    int* numnode=(int*)calloc(19,sizeof(int));
    for(int i=0;i<19;i++)
```

```
    {
        for(int j=0;j<20;j++)
        {
            if((i==arr[j][0])||(i==arr[j][2]))
            {
                numnode[i]=i;
                break;
            }
        }
    }
    for(int u=1;u<19;u++)
    {
        if(numnode[u]==0)//first place where u-1 is the last node number,
        {                        //(numnode was calloced),so there are u number of nodes
            return u;
        }
    }
}
int stfull(int** st,int sizst)//checks whether all nodes in st or not
{
    int* boolarr=(int*)calloc(sizst+1,sizeof(int));//to see if the nodes are in st, an
    indicator array
    //sizst is no.nodes -1.so number of nodes =sizst+1
    for(int h=0;h<=sizst;h++)//h is node number
    {
        for(int r=0;r<sizst;r++)//to find in st[r]
        {
            if(st[r][1]!=0)//edge wt not 0
            {
                if((st[r][0]==h)||(st[r][2]==h))
                {
                    boolarr[h]=1;
                }
            }
            else
            {
                break;
            }
        }
    }
    for(int p =0;p<sizst+1;p++)
    {
        if(boolarr[p]==0)
        {
            return 0;//there is a place in boolarr that is empty so all nodes are not in st
        }
    }
    return 1;
}
int belongtost(int** st,int val,int sizst)
{
    for(int w=0;w<sizst;w++)
    {
        if((val==st[w][0])||(val==st[w][2]))
        {
            return 1;
        }
    }
    return 0;
}
int* getfromset(int** set,int elem,int setc,int* pair)
{
    for(int e=0;e<=setc;e++)
    {
        if((elem==set[e][0])||(elem==set[e][2]))
        {
            if(elem==set[e][0])
            {
                pair[0]=elem;
                pair[2]=set[e][2];
            }
            else if(elem==set[e][2])
            {
                pair[0]=elem;
                pair[2]=set[e][0];
            }
```

```
            pair[1]=set[e][1];

            return pair;//as it would give the edge weight also,,,pair[2] would give the to node
        }
        //belongtost does the same thing only this returns a pair (triplet)
    }
    return pair;
}
int main()
{
    int n=20;
    // scanf("%d",&n);
    // printf("%d\n",n);

    pivot g[20];
    int arr_[20][3]={{0,6,1},{0,6,3},{0,6,2},{1,1,2},{2,2,3},{4,2,1},{4,7,2},{3,18,9},{4,4,5},{5,11,6},{5,10,7},{6

    int edgecount=0;

    int** arr=(int**)calloc(20,sizeof(int*));
    for (int u =0;u<20;u++)
    {
        arr[u]=(int*)calloc(3,sizeof(int));
        memcpy(arr[u],arr_[u],3*sizeof(int));
    }
    int nn=num_node(arr);


    int sizset=20-(nn-1)+1;
    int** set=(int**)calloc(sizset,sizeof(int*));
    int m1[3]={-1,-1,-1};
    for(int j=0;j<sizset;j++)
    {
        set[j]=(int*)calloc(3,sizeof(int));//to keep the disjoint sets
        memcpy(set[j],m1,3*sizeof(int));
    }

    //spanning tree will have edges ,1 less than the number of nodes
    int** st=(int**)calloc((nn-1),sizeof(int*));
    for(int t=0;t<nn-1;t++)
    {
        st[t]=(int*)calloc(3,sizeof(int));
        memcpy(st[t],m1,3*sizeof(int));
    }
    // calloc a row pointer means to null it for initialization
    //calloc ing a value means to make it zero--chat gpt
    for(int i=0;i<n;i++)
    {
        g[i].fromnode=arr[i][0];
        g[i].edgeweight=arr[i][1];
        g[i].tonode=arr[i][2];
    }
    qusort(g,0,n-1);
    for(int t=0;t<n;t++)
    {
        printf("%d %d %d sorted based on edge weight\n",g[t].fromnode,g[t].edgeweight,g[t].tonode);
    }//sorted descendingly
    st[0][0]=g[19].fromnode;
    st[0][1]=g[19].edgeweight;
    st[0][2]=g[19].tonode;
    edgecount=1;
    int c=18;
    int setc=0;//conuter for num of elements in set
    while(!stfull(st,nn-1))
    {
        int a=g[c].fromnode;
        int e=g[c].edgeweight;
        int b=g[c].tonode;

        if((!belongtost(st,a,nn-1))&&(!belongtost(st,b,nn-1)))
        {//both don't belong

            set[setc][0]=a;
```

```
            set[setc][1]=e;
            set[setc][2]=b;
            setc+=1;
        }
        else if((belongtost(st,a,nn-1))&&(!belongtost(st,b,nn-1)))//add the pair to st
        {
            st[edgecount][0]=a;
            st[edgecount][1]=e;
            st[edgecount][2]=b;
            edgecount+=1;
            int* pair=(int*)calloc(3,sizeof(int));//calloc this
            memcpy(pair,m1,3*sizeof(int));
            memcpy(pair,getfromset(set,b,setc,pair),3*sizeof(int));
            if(pair[1]!=-1)
            {
                st[edgecount][0]=pair[0];
                st[edgecount][1]=pair[1];
                st[edgecount][2]=pair[2];
                edgecount+=1;
            }
            free(pair);
        }
        else if((!belongtost(st,a,nn-1))&&(belongtost(st,b,nn-1)))//add the pair to st
        {
            st[edgecount][0]=a;
            st[edgecount][1]=e;
            st[edgecount][2]=b;
            edgecount+=1;
            int* pair=(int*)calloc(3,sizeof(int));//calloc this
            memcpy(pair,m1,3*sizeof(int));
            memcpy(pair,getfromset(set,a,setc,pair),3*sizeof(int));
            if(pair[1]!=-1)
            {
                st[edgecount][0]=pair[0];
                st[edgecount][1]=pair[1];
                st[edgecount][2]=pair[2];
                edgecount+=1;
            }
            free(pair);
        }
        c--;
    }
    printf("\n\n");
    for(int w=0;w<edgecount;w++)
    {
        printf("%d %d %d the from ,edge weight and to-node\n",st[w][0],st[w][1],st[w][2]);
    }
    return 0;
```

The graph this code is for is :



code output:

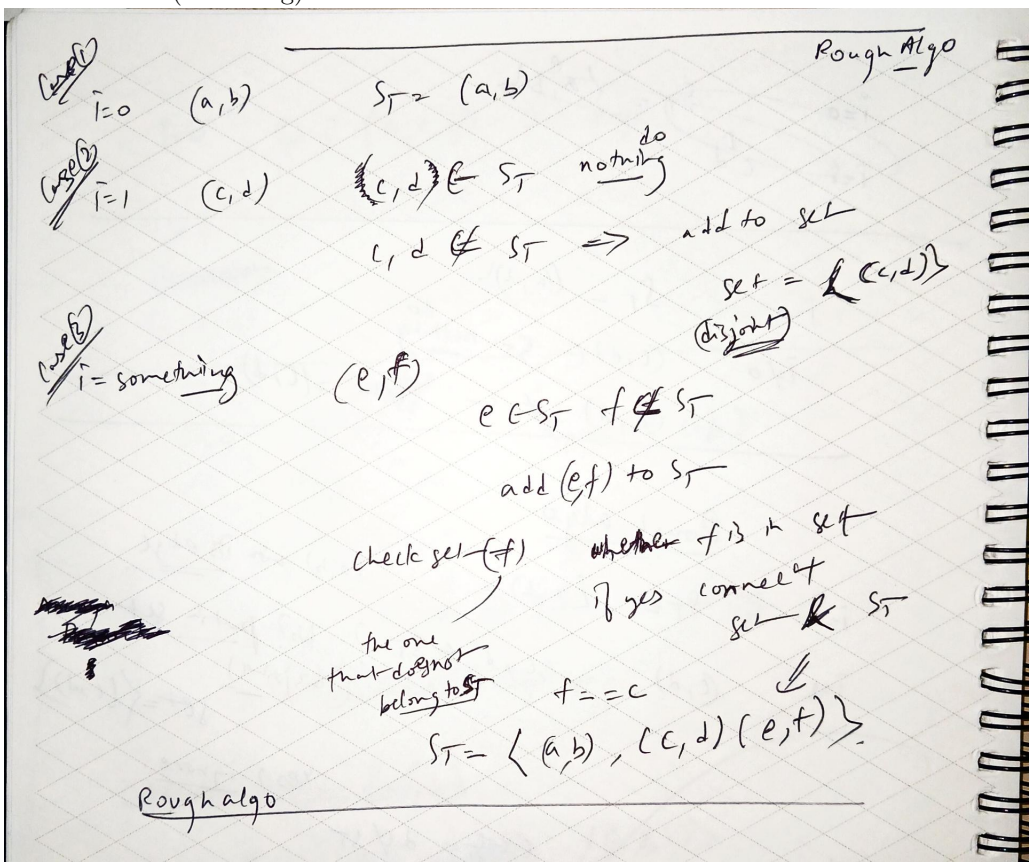
```
/home/asus/Documents/mysem5/daa_design_and_ana
7 25 10 sorted based on edge weight
6 22 7 sorted based on edge weight
3 18 9 sorted based on edge weight
8 16 10 sorted based on edge weight
7 12 8 sorted based on edge weight
5 11 6 sorted based on edge weight
5 10 7 sorted based on edge weight
11 8 9 sorted based on edge weight
4 7 2 sorted based on edge weight
0 6 3 sorted based on edge weight
0 6 2 sorted based on edge weight
0 6 1 sorted based on edge weight
4 4 5 sorted based on edge weight
10 3 11 sorted based on edge weight
4 2 1 sorted based on edge weight
6 2 8 sorted based on edge weight
2 2 3 sorted based on edge weight
1 1 2 sorted based on edge weight
8 1 9 sorted based on edge weight
2 1 6 sorted based on edge weight


2 1 6 the from ,edge weight to-node
1 1 2 the from ,edge weight to-node
2 2 3 the from ,edge weight to-node
6 2 8 the from ,edge weight to-node
8 1 9 the from ,edge weight to-node
4 2 1 the from ,edge weight to-node
4 4 5 the from ,edge weight to-node
0 6 1 the from ,edge weight to-node
11 8 9 the from ,edge weight to-node
11 3 10 the from ,edge weight to-node
5 10 7 the from ,edge weight to-node
```

1 About the hand written algo:

St is the spanning tree ,a,b,c,d,e,f are the nodes arbitraryily picked for showing the cases.Each pair like (a,b) also includes or indicates the presence of the edge weight with it.

1.1 the code's first part

Is to sort the edges structwise based on edge weight in descending order..and edge weights are used in the opposite direction(ascending)



References

none