```c
// ::::::UNION-FIND IMPLEMENTATION OF KRUSHKAL ALGOITHM---USING POINTER AND PATH COMPRESSION TECHNIQUE:::::::::::


#include<stdio.h>
#include<stdlib.h>
struct uf* ds=NULL;
struct data* dt=NULL;
static int p=0;

struct edge{
    int src,dest,w;
};

struct data{
    int v;
    int e;
    struct edge* ed;
};
struct node{
    int dest;
    struct node* next;
};

struct list{
    struct node* head;
};

struct uf{
    int *size;
    struct list* root;
    struct list* nod;
};

struct data* cdata(int v,int e){
    struct data* dt=(struct data*)malloc(sizeof(struct data));
    dt->v=v;
    dt->e=e;
    dt->ed=(struct edge*)malloc(e*sizeof(struct edge));
    return dt;
}

void addedge(struct data* dt,int src,int dest,int w){
    dt->ed[p].dest=dest;
     dt->ed[p].src=src;
      dt->ed[p++].w=w;

}

struct node* cnode(int d){
    struct node* temp=(struct node*)malloc(sizeof(struct node));
    temp->dest=d;
    temp->next=temp;
    return(temp);
}

struct uf* cuf(int v){
    struct uf* ds=(struct uf*)malloc(sizeof(struct uf));
    ds->size=(int*)malloc(v*sizeof(int));
    ds->root=(struct list*)malloc(v*sizeof(struct list));
    ds->nod=(struct list*)malloc(v*sizeof(struct list));
    for(int i=0;i<v;++i){
        ds->size[i]=1;
        ds->root[i].head=ds->nod[i].head=cnode(i);
    }
    return(ds);
```

```c
}

void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

int partition ( int low, int high)
{
    int pivot = dt->ed[high].w;
    int i = (low - 1);

    for (int j = low; j <= high- 1; j++)
    {
        if (dt->ed[j].w <= pivot)
        {
            i++;
            swap(&dt->ed[i].w, &dt->ed[j].w);
            swap(&dt->ed[i].src,&dt->ed[j].src);

            swap(&dt->ed[i].dest,&dt->ed[j].dest);
        }
    }
    swap(&dt->ed[i+1].w, &dt->ed[high].w);
            swap(&dt->ed[i+1].src,&dt->ed[high].src);

            swap(&dt->ed[i+1].dest,&dt->ed[high].dest);
    return (i + 1);
}

void quickSort(struct edge* ar, int low, int high)
{
    if (low < high)
    {
        int pi = partition( low, high);
        quickSort(ar, low, pi - 1);
        quickSort(ar, pi + 1, high);
    }
}
int find(int k){
    struct node* temp=ds->nod[k].head;
    while(temp->next!=temp){
        temp=temp->next;
    }
    ds->nod[k].head->next=temp;
    return(temp->dest);
}


void Union(struct uf* ds,int u,int v){
        int x=find(u);
        int y=find(v);

    if(ds->size[x]<ds->size[y]){\
         printf(" %d  ----> %d\n",u+1,v+1);
            struct node* temp=ds->nod[x].head;
            temp->next=ds->nod[y].head;
            ds->root[x].head=NULL;
            ds->size[y]+=ds->size[x];
            ds->size[x]=0;
                for(int i=0;i<7;++i){
                if(ds->root[i].head==NULL)
               printf("| size:  %d |  |root:: NULL|        |node:: %d| |parent::%d|\n",ds->size[i],ds->nod[i].head->dest+1,ds->nod[i].head->next->dest+1);
```

```

                else
        printf("| size:  %d |  |root::%d    |       |node:: %d|              |parent::%d| \n",ds->size[i],ds->root[i].head->dest+1,ds->nod[i].head->dest+1,ds->nod[i].head->next->dest+1);


    }

 dt->v=v;

    else{
         printf(" %d  ----> %d\n",u+1,v+1);
            struct node* temp=ds->nod[y].head;
            temp->next=ds->nod[x].head;
            ds->root[y].head=NULL;
            ds->size[x]+=ds->size[y];
            ds->size[y]=0;
                for(int i=0;i<7;++i){
                if(ds->root[i].head==NULL)
               printf("| size:  %d |  |root:: NULL|       |node:: %d|
|parent::%d|\n",ds->size[i],ds->nod[i].head->dest+1,ds->nod[i].head->next->dest+1);

                else
        printf("| size:  %d |  |root::%d    |       |node:: %d|              |parent::%d| \n",ds->size[i],ds->root[i].head->dest+1,ds->nod[i].head->dest+1,ds->nod[i].head->next->dest+1);


    }
         }
}
void krushkal(){
    int v=dt->v;
    int i=0;
    int e=0;
    quickSort(dt->ed,0,dt->e-1);
    while(e<v-1){

        struct edge dy=dt->ed[i++];
        printf("\ncall  %d and %d \n",dy.src+1,dy.dest+1);
        int x=find(dy.src);
        int y=find(dy.dest);
        if(x!=y){

            Union(ds,dy.src,dy.dest);
            e++;
        }

    }


}
int main(){
    ds=cuf(7)
    dt=cdata(7,8);
    addedge(dt,2,3,70);
    addedge(dt,4,6,10);
    addedge(dt,5,6,5);
    addedge(dt,1,2,6);
    addedge(dt,1,4,20);
    addedge(dt,0,1,10);
    addedge(dt,4,5,10);
    addedge(dt,0,2,18);


    printf("\n::::::::::::::::::::INITIAL SETUP::::::::::::::::::\n");
    for(int i=0;i<7;++i){
```

```c
        printf("| size:  %d |  |root::%d|   |node:: %d|  |parent::%d| \n",ds->size[i],ds->root[i].head->dest+1,ds->nod[i].head->dest+1,ds->nod[i].head->next->dest+1);
    }

    printf("::::::::::::::::CALLING KRIUSHKAL'S:::::::::::::::::::::\n");

    krushkal();


             int o=find(6);
             int y=find(4);
             printf("\n \n:::FINAL ANSWER::::\n");
                for(int i=0;i<7;++i){
                if(ds->root[i].head==NULL)
               printf("| size:  %d |  |root:: NULL|       |node:: %d||parent::%d|\n",ds->size[i],ds->nod[i].head->dest+1,ds->nod[i].head->next->dest+1);

                else
        printf("| size:  %d |  |root::%d    |       |node:: %d|              |parent::%d| \n",ds->size[i],ds->root[i].head->dest+1,ds->nod[i].head->dest+1,ds->nod[i].head->next->dest+1);


    }

    return 0;

}




```