Disjoint set

```
#include<stdio.h>
#include<stdlib.h>
struct node
        {
        int data;
        struct node *next;
        struct node *repr;
        };
struct sets
        {
        struct node *head;
        struct node *tail;
        };
struct Edge
        {
        int i,j;
        };
/*          0 ---1    4---5    7
            | \ |       | /    |
            2---3    6     8                    */
struct Edge E[14]={       {0,1},{0,3},{1,3},{0,2},{2,3},{4,5},{4,6},{5,6},{7,8}};
int m=9;
struct sets set[20];
int nset=0;
int V[]={0,1,2,3,4,5,6,7,8},n=9;
void printsets(struct sets set[],int n);
struct node* makeset(int v);
int findset(int v);
void uunion(int u,int v);
////////////////////////////////////////
int main()
        {
        struct Edge S[10];
        int k,x,y;
        for(k=0;k<n;k++)
                {
                set[k].head = set[k].tail= makeset(k);
                }
        printsets(set,n);
        for(k=0;k<m;k++)
                {
                x=findset(E[k].i);
                y=findset(E[k].j);
                printf("\n\n%d,%d",E[k].i,E[k].j);
                if(findset(E[k].i)!=findset(E[k].j))
                        {
                        uunion(E[k].i,E[k].j);
```

```
                        printsets(set,n);
                        }
                else
                        printf("\tunion not possibe.Same set ");
                }
        }


struct node* makeset(int v)
        {
        struct node *curr;
        curr= (struct node *)malloc(sizeof(struct node));
        curr->  data=v;
        curr->next=NULL;
        curr->repr=curr;
        nset++;
        return curr;
        }

int findset(int v)
        {
        int i;
        struct node *curr;

        for(i=0;i<n;i++)
                {
                curr= set[i].head;
                while(curr!=NULL)
                        {
                        if(curr->data==v)
                                return i;
                        curr=curr->next;
                        }
                }
        return -1;
        }

void uunion(int u,int v)
        {
        struct node *t,*p;
        int i,j;
        i = findset(u);
        j = findset(v);
        t = set[i].tail;
        t->     next =set[j].head;
        p=set[j].head;
        while(p!=NULL)
                {
```

```
                p->repr = set[i].head;
                p=p->next;
                }
        set[i].tail = set[j].tail;
        set[j].head=set[j].tail=NULL;
        nset--;
        }

void printsets(struct sets set[],int n)
        {
        int i;
                struct node *p;
        for(i=0;i<n;i++)
                {
                if(set[i].head!=NULL)
                        {
                        p=set[i].head;
                        printf("\t{");
                        while(p!=NULL)
                                {
                                printf(" %d",p->data);
                                p=p->next;
                                }
                        printf("}");
                        }
                }
        }
-------------------- ------------------------------------------------

#include<stdio.h>
#include<stdlib.h>
struct node
        {
        int data;
        struct node *next;
        struct node *repr;
        };
struct sets
        {
        struct node *head;
        struct node *tail;
        };

struct Edge
        {
        int start,end,weight;
        };
```

```
/*  1 ---2 ----3
            / |      / \    | \
           0         |  8   \   | 4
            \ | / \   \   |/
              7 -- 6 ---5
*/
struct Edge E[14]={       {0,1,4},{0,7,8},{1,2,8},{1,7,11},{2,3,7},{2,8,2},{2,5,4},
                                    {3,4,9},{3,5,14},{4,5,10},{5,6,2},{6,7,1},{6,8,6},{7,8,7}};
int m=14;
struct sets set[20];
struct Edge mst[14];
int mste =0;
int V[]={0,1,2,3,4,5,6,7,8},n=9;
/////Function prototype/////////
void MST_Kruskal(int V[],struct Edge[],int n,int m);
void sort(struct Edge a[],int n);
void printedge(struct Edge [],int n)         ;
struct node* makeset(int v);
int findset(int v);
void uunion(int u,int v);
//////////////////////////////////////////
int main()
        {
        struct Edge S[10];
        int i;
        MST_Kruskal(V,E,n,m);
        printedge(mst,mste);
        }

void MST_Kruskal(int V[],struct Edge E[],int n,int m)
        {
        int i;
        for(i=0;i<n;i++)
                {
                set[i].head = set[i].tail= makeset(i);
                }
        sort(E,m);
        for(i=0;i<m;i++)
                {
                if(findset(E[i].start)!=findset(E[i].end))
                        {
                        uunion(E[i].start,E[i].end);
                        mst[mste++]=E[i];
                        }
                }
        }
struct node* makeset(int v)
        {
```

```
        struct node *curr;
        curr= (struct node *)malloc(sizeof(struct node));
        curr->   data=v;
        curr->next=NULL;
        curr->repr=curr;
        return curr;
        }

int findset(int v)
        {
        int i;
        struct node *curr;
        for(i=0;i<n;i++)
                {
                curr= set[i].head;
                while(curr!=NULL)
                        {
                        if(curr->data==v)
                                return i;
                        curr=curr->next;
                        }
                }
        return -1;
        }

void uunion(int u,int v)
        {
        struct node *t,*p;
        int i,j;
        i = findset(u);
        j = findset(v);
        t = set[i].tail;
        t->     next =set[j].head;
        p=set[j].head;
        while(p!=NULL)
                {
                p->repr = set[i].head;
                p=p->next;
                }
        set[i].tail = set[j].tail;

        }
void sort(struct Edge a[],int n)
        {
        int i,j;
        struct Edge key;
        for(j=1;j<n;j++)
                {
```

```
                i=j-1;
                key=a[j];
                while(i>=0 && key.weight<a[i].weight)
                        {
                        a[i+1]=a[i];
                        i--;
                        }
                a[i+1]=key;
                }

        }
void printedge(struct Edge e[],int n)
        {
        int i;
        for(i=0;i<n;i++)
                {
                printf("\n%d %d %d",e[i].start,e[i].end,e[i].weight);
                }
        }
```