```java
class KruskalsMST {
    static int spanningTree(int V, ArrayList<ArrayList<ArrayList<Integer>>> adj) {
        // Add your code here
        HashMap<String,Integer> hmap = new HashMap<String,Integer>();
        int[] set= new int[V];
        Arrays.fill(set,-1);
        //getting unique edges since its a undirected graph
        for(int i=0;i<adj.size();i++){
            for(int k=0;k<adj.get(i).size();k++){
             addToMap(i,adj.get(i).get(k).get(0),adj.get(i).get(k).get(1),hmap);
            }
        }
        //sorting edges based on thier weights
        List<Map.Entry<String,Integer>> list  = new LinkedList<Map.Entry<String,Integer>>(hmap.entrySet());
        Collections.sort(list, new Comparator<Map.Entry<String,Integer>>(){
            public int compare(Map.Entry<String,Integer> e1, Map.Entry<String,Integer> e2){
                return e1.getValue().compareTo(e2.getValue());
            }
        });
        //greedy approch for considering edges in our spanning tree.
        int sumOfWeights=0;
        int edges=0;
        for(Map.Entry<String,Integer> entry : list){
            if(edges==V-1){
                return sumOfWeights;
            }
            //checking wheather considering the edge causes a loop  or not using disjoint sets
            if(considerable(entry.getKey(),set)){
                sumOfWeights+=entry.getValue();
                edges++;
            }
        }
        return sumOfWeights;

    }

    //helper function for adding edges to map
    public static void addToMap(int vertex1,int vertex2,int weight,HashMap<String,Integer> hmap){
        if(vertex1>vertex2){
            hmap.put(vertex2+","+vertex1,weight);
        }
        else{
            hmap.put(vertex1+","+vertex2,weight);
        }
    }

    //disjoint sets
    public static boolean considerable(String str,int[] set){
        String strArr[] = str.split(",");
        int vertex1 =  Integer.parseInt(strArr[0]);
        int vertex2 =  Integer.parseInt(strArr[1]);
        int parent1 = findParent(vertex1,set);
        int parent2 = findParent(vertex2,set);
        if(parent1==parent2){return false;}

        if(set[parent1]<set[parent2]){
            int temp = set[parent2];
            set[parent1]+=temp;
```

```
59              set[parent2]=parent1;
60          }
61          else{
62              int temp=set[parent1];
63              set[parent2]+=temp;
64              set[parent1]=parent2;
65          }
66          return true;
67      }
68
69       //non collapsing
70      // public static int findParent(int vertex,int[] set){
71
72      //   while(set[vertex]>0){
73      //       vertex=set[vertex];
74      //   }
75      //   return vertex;
76      // }
77
78      //collapsing find
79      public static int findParent(int vertex,int[] set){
80          int copy = vertex;
81          while(set[vertex]>0){
82              vertex=set[vertex];
83          }
84          if(copy!=vertex){
85          set[copy]=vertex;}
86          return vertex;
87      }
88
89  }
```