**AIM:-**

### **Kruskals algorithm for MST using greedy method**

**THEORY:-**

**Kruskal's algorithm** is a [minimum-spanning-tree algorithm](https://en.wikipedia.org/wiki/Minimum_spanning_tree#Algorithms) which finds an edge of the least possible weight that connects any two trees in the forest.[[1]](https://en.wikipedia.org/wiki/Kruskal%27s_algorithm#cite_note-:0-1) It is a [greedy algorithm](https://en.wikipedia.org/wiki/Greedy_algorithm) in [graph theory](https://en.wikipedia.org/wiki/Graph_theory) as it finds a [minimum spanning tree](https://en.wikipedia.org/wiki/Minimum_spanning_tree) for a [connected](https://en.wikipedia.org/wiki/Connectivity_(graph_theory)) [weighted graph](https://en.wikipedia.org/wiki/Glossary_of_graph_theory#Weighted_graphs_and_networks) adding increasing cost arcs at each step.[[1]](https://en.wikipedia.org/wiki/Kruskal%27s_algorithm#cite_note-:0-1) This means it finds a subset of the [edges](https://en.wikipedia.org/wiki/Edge_(graph_theory)) that forms a tree that includes every [vertex](https://en.wikipedia.org/wiki/Vertex_(graph_theory)), where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it finds a *minimum spanning forest* (a minimum spanning tree for each [connected component](https://en.wikipedia.org/wiki/Connected_component_(graph_theory))).

Pseudocode:-

KRUSKAL(G):

1 A = ∅

2 **foreach** v ∈ G.V:

3 MAKE-SET(v)

4 **foreach** (u, v) in G.E ordered by weight(u, v), increasing:

5 **if** FIND-SET(u) ≠ FIND-SET(v):

6 A = A ∪ {(u, v)}

7 UNION(u, v)

8 **return** A

**Complexity**

Kruskal's algorithm can be shown to run in [*O*](https://en.wikipedia.org/wiki/Big-O_notation)(*E* [log](https://en.wikipedia.org/wiki/Binary_logarithm) *E*) time, or equivalently, *O*(*E* log *V*) time, where *E* is the number of edges in the graph and *V* is the number of vertices, all with simple data structures. These running times are equivalent because:

* *E* is at most V^2 and log ⁡ V 2 = 2 log ⁡ V {\displaystyle \log V^{2}=2\log V} log V^2=2log V is O ( log ⁡ V ) {\displaystyle O(\log V)} O(logV).

We can achieve this bound as follows: first sort the edges by weight using a [comparison sort](https://en.wikipedia.org/wiki/Comparison_sort) in *O*(*E* log *E*) time; this allows the step "remove an edge with minimum weight from *S*" to operate in constant time. Next, we use a [disjoint-set data structure](https://en.wikipedia.org/wiki/Disjoint-set_data_structure) to keep track of which vertices are in which components. We need to perform O(*V*) operations, as in each iteration we connect a vertex to the spanning tree, two 'find' operations and possibly one union for each edge. Even a simple disjoint-set data structure such as disjoint-set forests with union by rank can perform O(*V*) operations in *O*(*V* log *V*) time. Thus the total time is *O*(*E* log *E*) = *O*(*E* log *V*).

**PROGRAM:-**

#include<stdio.h>

#include<stdlib.h>

#define MAX 9999

typedef struct edge{

int x,y,value;

}edge;

typedef struct SET{

int parent;

int value;

struct SET\* next;

}SET;

SET\* create()

{

return (SET\*)malloc(sizeof(SET));

}

int compare(const void \*s1, const void \*s2)

{

struct edge \*e1 = (struct edge \*)s1;

struct edge \*e2 = (struct edge \*)s2;

return e1->value - e2->value;

}

int find(int x,SET\* head)

{

SET \*temp=head;

while(temp!=NULL)

{

if(temp->value==x)

{

if(temp->parent==-1)

return x;

return find(temp->parent,head);

}

temp=temp->next;

}

return -1;

}

void makeunion(int x,int y,SET\* head)

{

int xset=find( x,head);

int yset=find(y,head);

SET \*temp=head;

while(temp!=NULL)

{

if(temp->value==xset)

{

temp->parent=yset;

//printf("parent of %d is now %d\n",xset,yset);

return;

}

temp=temp->next;

}

return;

}

void main()

{

printf("Enter no of nodes\n");

int V;

scanf("%d",&V);

int i=0;

SET\* head=NULL;

for(i=0;i<V;i++)

{

SET\* temp=create();

temp->value=i;

temp->parent=-1;

temp->next=head;

head=temp;

}

printf("enter no of edges\n");

int count;

scanf("%d",&count);

printf("enter each edge x,y,value(undirected graph)\n");

edge array[count];

for(i=0;i<count;i++)

{

//printf("enter value of edge%d\n",i);

scanf("%d",&array[i].x);

scanf("%d",&array[i].y);

scanf("%d",&array[i].value);

}

edge answer[V-1];

qsort(array,count,sizeof(edge),compare);

printf("After sorting edges are\nx\ty\tvalue\n");

for(i=0;i<count;i++)

printf("%d\t%d\t%d\n",array[i].x,array[i].y,array[i].value);

int l=0;

int c=0;

while(l<V&&c!=count)

{

edge current=array[c++];

int x=current.x;

int y=current.y;

if(find(x,head)!=find(y,head))

{

makeunion(x,y,head);

answer[l++]=current;

printf("\n this edge added %d\t%d\t%d\n",current.x,current.y,current.value);

}

}

printf("edges included are\n");

printf("i\tj\tvalue\n");

for(i=0;i<V-1;i++)

{

printf("%d\t%d\t%d\n",answer[i].x,answer[i].y,answer[i].value);

}

}

**OUTPUT:-**

Enter no of nodes

9

enter no of edges

14

enter each edge x,y,value(undirected graph)

0

1

4

1

2

8

2

3

7

3

4

9

4

5

10

5

6

2

6

7

1

7

0

8

1

7

11

7

8

7

8

6

6

8

2

2

2

5

4

3

5

14

After sorting edges are

x y value

6 7 1

8 2 2

5 6 2

0 1 4

2 5 4

8 6 6

2 3 7

7 8 7

7 0 8

1 2 8

3 4 9

4 5 10

1 7 11

3 5 14

this edge added 6 7 1

this edge added 8 2 2

this edge added 5 6 2

this edge added 0 1 4

this edge added 2 5 4

this edge added 2 3 7

this edge added 7 0 8

this edge added 3 4 9

edges included are

i j value

6 7 1

8 2 2

5 6 2

0 1 4

2 5 4

2 3 7

7 0 8

3 4 9

**CONCLUSION:-**

This method works only for undirected connected graph to find MST.