**KRUSKAL**

#include <stdio.h>

#include <stdlib.h>

typedef struct head

{

  struct object \*head;

  struct object \*tail;

  int size;

} head;

typedef struct object

{

  int data;

  struct object \*next;

  head \*prev;

} object;

typedef struct Edge

{

  int src, dest, weight;

} Edge;

head \*makeSet(int data)

{

  head \*Header = (head \*)malloc(sizeof(head));

  Header->head = (object \*)malloc(sizeof(object));

  Header->tail = (object \*)malloc(sizeof(object));

  Header->size = 1;

  Header->head->data = data;

  Header->head->next = NULL;

  Header->head->prev = Header;

  Header->tail = Header->head;

  return Header;

}

void Union(head \*x, head \*y)

{

  if (x->head->prev == y->head->prev)

  {

    return;

  }

  head \*X = x->head->prev;

  head \*Y = y->head->prev;

  if (X->size > Y->size)

  {

    object \*temp = Y->head, \*prev;

    while (temp)

    {

      prev = temp;

      temp->prev = X;

      temp = temp->next;

    }

    X->tail->next = Y->head;

    X->tail = Y->tail;

    X->size += Y->size;

    return;

  }

  else

  {

    object \*temp = X->head, \*prev;

    while (temp)

    {

      prev = temp;

      temp->prev = Y;

      temp = temp->next;

    }

    Y->tail->next = X->head;

    Y->tail = X->tail;

    Y->size += X->size;

  }

}

void printGraph(Edge \*edge, int e)

{

  int i;

  printf("Following are the edges in the constructed MST\n");

  for (i = 0; i < e; i++)

  {

    printf("%c -- %c == %d\n", ((char)(edge[i].src + 64)), ((char)(edge[i].dest + 64)), edge[i].weight);

  }

  int totalWeight = 0;

  for (i = 0; i < e; i++)

  {

    totalWeight += edge[i].weight;

  }

  printf("Total weight of the MST = %d\n", totalWeight);

}

int partition(Edge \*edge, int low, int high)

{

  int pivot = edge[high].weight;

  int i = (low - 1), j;

  for (j = low; j <= high - 1; j++)

  {

    if (edge[j].weight < pivot)

    {

      i++;

      Edge temp = edge[i];

      edge[i] = edge[j];

      edge[j] = temp;

    }

  }

  Edge temp = edge[i + 1];

  edge[i + 1] = edge[high];

  edge[high] = temp;

  return (i + 1);

}

void sortEdges(Edge \*edge, int low, int high)

{

  if (low < high)

  {

    int pivot = partition(edge, low, high);

    sortEdges(edge, low, pivot - 1);

    sortEdges(edge, pivot + 1, high);

  }

}

void KruskalMST(head \*\*arr, Edge \*edgeArr, int e, int v)

{

  int treeSize = 0;

  Edge \*result = (Edge \*)malloc(v \* sizeof(Edge));

  int j = 0;

  while (treeSize < e - 1 && j < e)

  {

    if (arr[edgeArr[j].src - 1]->head->prev != arr[edgeArr[j].dest - 1]->head->prev)

    {

      result[treeSize] = edgeArr[j];

      treeSize++;

      Union(arr[edgeArr[j].src - 1], arr[edgeArr[j].dest - 1]);

    }

    j++;

  }

  printGraph(result, v);

}

int main()

{

  int i;

  FILE \*file = fopen("graph.txt", "r");

  if (file == NULL)

  {

    printf("Error opening file");

    exit(1);

  }

  int edges = 14, vertices = 9;

  Edge \*edgeArr = (Edge \*)malloc(edges \* sizeof(Edge));

  for (i = 0; i < edges; i++)

  {

    fscanf(file, "%d %d %d", &edgeArr[i].src, &edgeArr[i].dest, &edgeArr[i].weight);

  }

  fclose(file);

  sortEdges(edgeArr, 0, edges - 1);

  printf("Edges after sorting\n");

  printf("src\tdest\tweight\n");

  for (i = 0; i < edges; i++)

  {

    printf("%c\t%c\t%d\n", ((char)(edgeArr[i].src + 64)), ((char)(edgeArr[i].dest + 64)), edgeArr[i].weight);

  }

  head \*\*arr = (head \*\*)malloc(edges \* sizeof(head \*));

  for (i = 0; i < edges; i++)

  {

    arr[i] = makeSet(i + 1);

  }

  KruskalMST(arr, edgeArr, edges, vertices - 1);

  return 0;

}

**graph.txt**

1 2 4

1 3 8

2 3 11

2 4 8

3 5 7

3 6 1

4 5 2

4 8 4

4 7 7

5 6 6

6 8 2

7 8 14

7 9 9

8 9 10

**OUTPUT**

Edges after sorting  
src dest weight  
C F 1  
F H 2  
D E 2  
D H 4  
A B 4  
E F 6  
C E 7  
D G 7  
A C 8  
B D 8  
G I 9  
H I 10  
B C 11  
G H 14

Following are the edges in the constructed MST

C -- F == 1

F -- H == 2

D -- E == 2

D -- H == 4

A -- B == 4

D -- G == 7

A -- C == 8

G -- I == 9

Total weight of the MST = 37