**Collage : Vishwakarma Institute of Technology**

**Course Name : Data Structure in C**

**Name : Vedika Vikas Sontakke**

**Roll no : 37**

**PRN NO 12220206**

Assignment : Implement Spanning tree using krushal’s algorithm and prim’s algorithm .

Program : Krushal’s Algo

// Kruskal's algorithm in C

#include <stdio.h>

#define MAX 30

typedef struct edge {

  int u, v, w;

} edge;

typedef struct edge\_list {

  edge data[MAX];

  int n;

} edge\_list;

edge\_list elist;

int Graph[MAX][MAX], n;

edge\_list spanlist;

void kruskalAlgo();

int find(int belongs[], int vertexno);

void applyUnion(int belongs[], int c1, int c2);

void sort();

void print();

// Applying Krushkal Algo

void kruskalAlgo() {

  int belongs[MAX], i, j, cno1, cno2;

  elist.n = 0;

  for (i = 1; i < n; i++)

    for (j = 0; j < i; j++) {

      if (Graph[i][j] != 0) {

        elist.data[elist.n].u = i;

        elist.data[elist.n].v = j;

        elist.data[elist.n].w = Graph[i][j];

        elist.n++;

      }

    }

  sort();

  for (i = 0; i < n; i++)

    belongs[i] = i;

  spanlist.n = 0;

  for (i = 0; i < elist.n; i++) {

    cno1 = find(belongs, elist.data[i].u);

    cno2 = find(belongs, elist.data[i].v);

    if (cno1 != cno2) {

      spanlist.data[spanlist.n] = elist.data[i];

      spanlist.n = spanlist.n + 1;

      applyUnion(belongs, cno1, cno2);

    }

  }

}

int find(int belongs[], int vertexno) {

  return (belongs[vertexno]);

}

void applyUnion(int belongs[], int c1, int c2) {

  int i;

  for (i = 0; i < n; i++)

    if (belongs[i] == c2)

      belongs[i] = c1;

}

// Sorting algo

void sort() {

  int i, j;

  edge temp;

  for (i = 1; i < elist.n; i++)

    for (j = 0; j < elist.n - 1; j++)

      if (elist.data[j].w > elist.data[j + 1].w) {

        temp = elist.data[j];

        elist.data[j] = elist.data[j + 1];

        elist.data[j + 1] = temp;

      }

}

// Printing the result

void print() {

  int i, cost = 0;

  for (i = 0; i < spanlist.n; i++) {

    printf("\n%d - %d : %d", spanlist.data[i].u, spanlist.data[i].v, spanlist.data[i].w);

    cost = cost + spanlist.data[i].w;

  }

  printf("\nSpanning tree cost: %d", cost);

}

int main() {

  int i, j, total\_cost;

  n = 6;

  Graph[0][0] = 0;

  Graph[0][1] = 4;

  Graph[0][2] = 4;

  Graph[0][3] = 0;

  Graph[0][4] = 0;

  Graph[0][5] = 0;

  Graph[0][6] = 0;

  Graph[1][0] = 4;

  Graph[1][1] = 0;

  Graph[1][2] = 2;

  Graph[1][3] = 0;

  Graph[1][4] = 0;

  Graph[1][5] = 0;

  Graph[1][6] = 0;

  Graph[2][0] = 4;

  Graph[2][1] = 2;

  Graph[2][2] = 0;

  Graph[2][3] = 3;

  Graph[2][4] = 4;

  Graph[2][5] = 0;

  Graph[2][6] = 0;

  Graph[3][0] = 0;

  Graph[3][1] = 0;

  Graph[3][2] = 3;

  Graph[3][3] = 0;

  Graph[3][4] = 3;

  Graph[3][5] = 0;

  Graph[3][6] = 0;

  Graph[4][0] = 0;

  Graph[4][1] = 0;

  Graph[4][2] = 4;

  Graph[4][3] = 3;

  Graph[4][4] = 0;

  Graph[4][5] = 0;

  Graph[4][6] = 0;

  Graph[5][0] = 0;

  Graph[5][1] = 0;

  Graph[5][2] = 2;

  Graph[5][3] = 0;

  Graph[5][4] = 3;

  Graph[5][5] = 0;

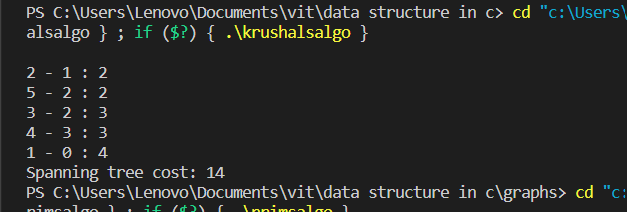
  Graph[5][6] = 0;

  kruskalAlgo();

  print();

}

Output :



Porgam : Prim’s Algo

// Prim's Algorithm in C

#include<stdio.h>

#include<stdbool.h>

#include <string.h>

#define INF 9999999

// number of vertices in graph

#define V 5

// create a 2d array of size 5x5

//for adjacency matrix to represent graph

int G[V][V] = {

  {0, 9, 75, 0, 0},

  {9, 0, 95, 19, 42},

  {75, 95, 0, 51, 66},

  {0, 19, 51, 0, 31},

  {0, 42, 66, 31, 0}};

int main() {

  int no\_edge;  // number of edge

  // create a array to track selected vertex

  // selected will become true otherwise false

  int selected[V];

  // set selected false initially

  memset(selected, false, sizeof(selected));

  // set number of edge to 0

  no\_edge = 0;

  // the number of egde in minimum spanning tree will be

  // always less than (V -1), where V is number of vertices in

  //graph

  // choose 0th vertex and make it true

  selected[0] = true;

  int x;  //  row number

  int y;  //  col number

  // print for edge and weight

  printf("Edge : Weight\n");

  while (no\_edge < V - 1) {

    //For every vertex in the set S, find the all adjacent vertices

    // , calculate the distance from the vertex selected at step 1.

    // if the vertex is already in the set S, discard it otherwise

    //choose another vertex nearest to selected vertex  at step 1.

    int min = INF;

    x = 0;

    y = 0;

    for (int i = 0; i < V; i++) {

      if (selected[i]) {

        for (int j = 0; j < V; j++) {

          if (!selected[j] && G[i][j]) {  // not in selected and there is an edge

            if (min > G[i][j]) {

              min = G[i][j];

              x = i;

              y = j;

            }

          }

        }

      }

    }

    printf("%d - %d : %d\n", x, y, G[x][y]);

    selected[y] = true;

    no\_edge++;

  }

  return 0;

}

Output :

