//Assignment no 7

#include<iostream>

using namespace std;

class snode

{

public: // data structure for sparse matrix.

char u1,u2;

int wt;

};

class test

{

int n,m,x;

snode arr[10],res[10];

public:

test()

{

n=0;

m=1;

}

void inputsparse();

void displaysparse();

void bsort();

void prims();

void dispmst();

};

// Function to Display result

void test::dispmst()

{

for(int i=0;i<m;i++)

{

cout<<res[i].u1<<" "<<res[i].u2<<" "<<res[i].wt<<endl;

}

}

// Function to find minimum spanning tree.

void test::prims()

{

int cnt=0;

int flag1,flag2,i;

res[0]=arr[0];

m=1;

cnt=1;

do

{

for(i=1; i<n ; i++) //arr

{

flag1=0;

flag2=0;

for(int j=0;j<m;j++)

{

if((arr[i].u1==res[j].u1 || arr[i].u1==res[j].u2 ) && flag1==0)

{

flag1++;

}

if((arr[i].u2==res[j].u1 || arr[i].u2==res[j].u2 ) && flag2==0)

{

flag2++;

}

}

if(flag1!=1 ^ flag2 !=1) // ^ Exore operation.

{

res[m++]=arr[i];

}

}

cnt = x-1;

}while(m!=cnt) ;

}

// Function tio Read input graph.

void test::inputsparse()

{

cout<<"ENTER NO OF EDGES: ";

cin>>n;

cout<<"ENTER NO OF VERTICES: ";

cin>>x;

for(int i=0;i<n;i++)

{

cout<<"ENTER 1ST VERTEX: ";

cin>>arr[i].u1;

cout<<"ENTER 2ND VERTEX: ";

cin>>arr[i].u2;

cout<<"ENTER WEIGHT: ";

cin>>arr[i].wt;

cout<<endl;

}

bsort();

}

void test::displaysparse()

{

for(int i=0;i<n;i++)

{

cout<<arr[i].u1<<" "<<arr[i].u2<<" "<<arr[i].wt<<endl;

}

}

//Sort the given edges of the graph using bubble sort

void test::bsort()

{

snode temp;

for(int i=0;i<n-1;i++)

{

for(int j=0;j<n-1-i;j++)

{

if(arr[j].wt>arr[j+1].wt)

{

temp=arr[j];

arr[j]=arr[j+1];

arr[j+1]=temp;

}

}

}

}

int main()

{

test obj;

obj.inputsparse();

obj.displaysparse();

obj.prims();

cout<<"RESULT:"<<endl;

obj.dispmst();

return 0;

}

output:

escoe@gescoe-OptiPlex-3010:~/Desktop/SE-A-55$ g++ mst\_7.cpp

gescoe@gescoe-OptiPlex-3010:~/Desktop/SE-A-55$ ./a.out

ENTER NO OF EDGES: 4

ENTER NO OF VERTICES: 5

ENTER 1ST VERTEX: 1

ENTER 2ND VERTEX: 2

ENTER WEIGHT: 6

ENTER 1ST VERTEX: 3

ENTER 2ND VERTEX: 4

ENTER WEIGHT: 7

ENTER 1ST VERTEX: 5

ENTER 2ND VERTEX: 6

ENTER WEIGHT: 8

ENTER 1ST VERTEX: 7

ENTER 2ND VERTEX: 8

ENTER WEIGHT: 9

1. 2 6

3 4 7

5 6 8

7 8 9

Sure! Here's a simplified explanation of the **theory** behind **Prim's Algorithm** for finding a **Minimum Spanning Tree (MST)** in a graph.

### ****What is a Minimum Spanning Tree (MST)?****

In graph theory, a **spanning tree** of a graph is a subgraph that includes all the vertices of the original graph, and it is a tree (meaning it has no cycles). The **minimum spanning tree** is the spanning tree with the smallest possible total edge weight.

### ****Prim's Algorithm Explanation****

Prim's algorithm is a **greedy algorithm** that helps in finding the **minimum spanning tree (MST)** for a connected, weighted graph. It starts from any node and grows the MST by adding the smallest edge that connects a node in the tree to a node outside of it.

### ****Steps of Prim’s Algorithm****

1. **Start with an arbitrary node**:
   * Choose any vertex to start the algorithm. This will be the first node in your MST.
2. **Select the smallest edge**:
   * Find the smallest edge that connects the MST (the set of nodes already included in the MST) to a node that is not in the MST yet. This is the edge with the smallest weight.
3. **Add the edge to the MST**:
   * Once the smallest edge is selected, add this edge and the node it connects to the MST.
4. **Repeat the process**:
   * Continue this process until all the nodes are included in the MST. In each iteration, select the smallest edge that connects a node inside the MST to one outside it.

### ****Why is Prim's Algorithm Greedy?****

Prim's algorithm is greedy because at every step, it picks the **best option locally** (the smallest edge weight connecting the MST to another node) with the hope that these local decisions will lead to a global optimum (the minimum spanning tree).

### ****Example to Understand Prim's Algorithm****

Let's say we have a graph with 4 vertices and the following edges:

* **Vertex 1** to **Vertex 2** with weight 6
* **Vertex 2** to **Vertex 3** with weight 5
* **Vertex 1** to **Vertex 3** with weight 9
* **Vertex 3** to **Vertex 4** with weight 7

To apply Prim’s Algorithm:

1. Start with **Vertex 1**.
2. The smallest edge connecting **Vertex 1** to another vertex is **1 -> 2** with weight 6.
3. Now, the MST includes **Vertex 1** and **Vertex 2**. The smallest edge connecting the MST to a new vertex is **2 -> 3** with weight 5.
4. Now, the MST includes **Vertex 1**, **Vertex 2**, and **Vertex 3**. The smallest edge connecting the MST to a new vertex is **3 -> 4** with weight 7.
5. Now, all vertices are included in the MST, and we have the MST edges:
   * **1 -> 2** with weight 6
   * **2 -> 3** with weight 5
   * **3 -> 4** with weight 7

### ****Key Characteristics of Prim’s Algorithm****

* **Efficient for Dense Graphs**: Prim’s algorithm is typically more efficient for dense graphs, where there are many edges.
* **Starts from any vertex**: You can choose any vertex to start building the MST.
* **Greedy approach**: The algorithm makes decisions based on the smallest edge connecting the tree to a new vertex, ensuring that at each step the tree grows with the least possible weight.

### ****Time Complexity****

* The time complexity of Prim’s algorithm depends on how the graph is represented and how the smallest edge is selected. In the worst case, when using an adjacency matrix, the time complexity is O(n2)O(n^2), where nn is the number of vertices.
* If using a priority queue (min-heap), the time complexity can be improved to O((E+V)log⁡V)O((E + V) \log V), where EE is the number of edges and VV is the number of vertices.

### ****Why Use Prim's Algorithm?****

Prim’s algorithm is useful when you need to connect all nodes of a graph with the least possible edge weights. It’s commonly used in **network design** problems, such as connecting computers, cities, or any infrastructure where minimizing the cost of connections is important.

I hope this helps! Let me know if you need further clarification.