Practical-7(C-15)

Problem Statement:
You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. Solve the problem by suggesting appropriate data structures.

Code:
```
#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 kruskals();
     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::kruskals()
{
              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.kruskals();
   cout<<"RESULT:"<<endl;
   obj.dispmst();
return 0;
}
```

OUTPUT:

```
ENTER NO OF EDGES: 3
ENTER NO OF VERTICES: 4
ENTER 1ST VERTEX: 1
ENTER 2ND VERTEX: 2
ENTER WEIGHT: 4
ENTER 1ST VERTEX: 2
ENTER 2ND VERTEX: 3
ENTER WEIGHT: 6
ENTER 1ST VERTEX: 3
ENTER 2ND VERTEX: 4
ENTER WEIGHT: 2
3 4 2
1 2 4
2 3 6
RESULT:
3 4 2
2 3 6
1 2 4
```