Competitive Programming Assignment

(https://github.com/MalavikaJayakumar/Competitive-Programming-Problems)

1. **Minimum Spanning Tree**
   a. **Prims Algorithm**

Code:

```
#include<iostream>
using namespace std;

int V;
int minweight(int key[],bool mst[])
{
	int min = INT_MAX,minindex;
	for(int v=0;v<V;v++)
	{
		if(mst[v] == false && key[v]<min)
		{
			min=key[v];
			minindex=v;
		}
	}
	return minindex;
}

void mstdisplay(int parent[],int *graph)
{
	cout<<"edge \t Weight \n";
	for(int i=1;i<V;i++)
	{
		cout<<parent[i]<<" - "<<i<<"
\t"<<*((graph+(i*V))+parent[i])<<" \n";
	}
}

void mstdisplay(int *graph)
{
	int parent[V],key[V];
	bool mst[V];
	for(int i=0;i<V;i++)
	{
		key[i]=INT_MAX;
		mst[i]=false;
	}
	key[0]=0;
	parent[0]=-1;
	for(int count = 0;count<V-1;count++)
	{
		int u =minweight(key,mst);
		mst[u] = true;
		for(int v=0;v<V;v++)
		{
```

```
                    if(*((graph+(u*V))+v) && mst[v]== false &&
*((graph+(u*V))+v)<key[v])
                    {
                            parent[v]=u;
                key[v]=*((graph+(u*V))+v);
                    }
                }
        }
        mstdisplay(parent,graph);
}

int readgraph(int *m,int v,int e)
{
        int x,y,w;
        cout<<"\n enter start node,end node and weight";
        for(int i=0;i<e;i++)
        {
                cin>>x>>y>>w;
                *((m+(x*v))+y)=w;
                *((m+(y*v))+x)=w;
        }

        return 0;
}

int main()
{
        int e;
        cout<<"Enter no.of vertices and edges: ";
        cin>>V>>e;
        int graph[V][V]={0};
        readgraph((int *)graph,V,e);
        mstdisplay((int *)graph);
        return 0;
}
```

Output

```
Enter no.of vertices and edges: 4 5

 enter start node,end node and weight0 1 1
0 3 4
0 2 5
2 3 3
1 3 2
edge     Weight
0 - 1    1
3 - 2    3
1 - 3    2
```

b. Kruskal's Algorithm

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

#define edge pair<int,int>

class Graph {
private:
    vector<pair<int, edge> > G;
    vector<pair<int, edge> > T;
    int *parent;
    int V;
public:
    Graph(int V);
    void AddWeightedEdge(int e);
    int find_set(int i);
    void union_set(int u, int v);
    void kruskal();
    void print();
};
Graph::Graph(int V) {
    parent = new int[V];
    for (int i = 0; i < V; i++)
        parent[i] = i;

    G.clear();
    T.clear();
}
void Graph::AddWeightedEdge(int e) {
   int i,src,dest,wt;
   cout<<"Enter start node, end node and weight: ";
    for(i=0;i<e;i++)
    {
           cin>>src>>dest>>wt;
           G.push_back(make_pair(wt, edge(src, dest)));
    }

}
int Graph::find_set(int i) {

    if (i == parent[i])
        return i;
    else
        return find_set(parent[i]);
}

void Graph::union_set(int u, int v) {
    parent[u] = parent[v];
}
```

```
void Graph::kruskal() {
    int i, uRep, vRep;
    sort(G.begin(), G.end());
    for (i = 0; i < G.size(); i++) {
        uRep = find_set(G[i].second.first);
        vRep = find_set(G[i].second.second);
        if (uRep != vRep) {
            T.push_back(G[i]);
            union_set(uRep, vRep);
        }
    }
}
void Graph::print() {
    cout<<"Minimum Spanning tree\n";
    cout << "Edge :" << " Weight" << endl;
    for (int i = 0; i < T.size(); i++) {
        cout << T[i].second.first << " - " << T[i].second.second << " : "
                << T[i].first;
        cout << endl;
    }
}
int main() {
    int v,e;
    cout<<"Enter number of vertices and edges: ";
    cin>>v>>e;
    Graph g(v);
    g.AddWeightedEdge(e);
    g.kruskal();
    g.print();
    return 0;
}
```

Output

```
Enter number of vertices and edges: 4 5
Enter start node, end node and weight: 0 1 1
0 2 5
1 2 4
1 3 2
2 3 3
Minimum Spanning tree
Edge : Weight
0 - 1 : 1
1 - 3 : 2
2 - 3 : 3
```