```cpp
/**
    Marlon Fernandes de Alcantara
    Kruskal - Minimum Spanning Forest
**/

#include <iostream>
#include <cstdio>
#include <vector>
#include <algorithm>

using namespace std;

#define all(c)    (c).begin(),  (c).end()
#define rall(c)  (c).rbegin(), (c).rend()

#define _auto(var, x)           typeof(x) var = (x)
#define _auxforeach(it, b, e)   for(_auto(it, b), _itend = e; it != _itend; ++it)
#define foreach(it, r...)       _auxforeach(it, r)

typedef long long LLONG;
typedef pair<int,int> PII;
typedef vector<int> VI;
typedef vector<VI> VVI;

enum { INF = 1<<30 };


struct UnionFind { // Union Find Representation by X-Tree
    VI roots, ranks;   // roots for each element (label) and ranks(tree degree)

    UnionFind(int nElems) : roots(nElems),
            ranks(nElems){
        foreach(i, 0,nElems) roots[i] = i;
    }

    /// The root node of x's tree. // Unique label
    int rootOf(int x) { // "almost" O(1)
        if(roots[x] != x)
            roots[x] = rootOf(roots[x]);
        return roots[x];
    }
    /// Merges the trees of a and b. // O(1) //make root(a) = root(b)
    void mergeRootsOf(int a, int b) {
        a = rootOf(a);
        b = rootOf(b);
        if(a != b) {
            if(ranks[a] > ranks[b])
                swap(a, b);
            else if(ranks[a] == ranks[b])
                ranks[b] ++;
            roots[a] = b;
        }
    }
};

struct Edge{
    int u, v, w;
    Edge(int u=0, int v=0, int w=0) : u(u<v?u:v),v(u<v?v:u),w(w){}
    bool operator<(const Edge &x)const{
        if(w!=x.w) return w<x.w;
        if(u!=x.u) return u<x.u;
        return v<x.v;
    }
};

/// Complexidade O(e.log(v))
vector<Edge> Kruskal(int v, vector<Edge> e){ ///Kruskal em 8 linhas XD
    sort(all(e));                       // Passo1: Ordenar as arestas
    UnionFind subtrees(v);              // Criar uma estrutura para verificar os conjuntos de subarvores(UnionFind) almost O(1)
    vector<Edge> result;                // Vetor para guardar as arestas que serao utilizadas, desnecessario se quiser apenas o peso da arvore
    foreach(it, all(e)){                // Passo2: Para cada aresta, ligar duas sub-arvores distintas com a menor aresta, ate que acabem as arestas
        if(subtrees.rootOf(it->u)!=subtrees.rootOf(it->v)){
            result.push_back(*it);
            subtrees.mergeRootsOf(it->u, it->v);
        }
    }
    return result;
}

int main(){
    int v, e;
    scanf("%d %d", &v, &e);
    vector<Edge> edges;
    for(int i=0; i<e; ++i){
        int u, v, w;
        scanf("%d %d %d", &u, &v, &w);
        edges.push_back(Edge(u,v,w));
    }

    vector<Edge> answer = Kruskal(v, edges);
    puts("Arestas que compoem a Floresta Geradora Minima:\n");
    int weight=0;
    foreach(it, all(answer)){
        printf("%d - %d  [%d]\n", it->u, it->v, it->w);
        weight+=it->w;
    }
    printf("\nO peso da arvore: %d\n", weight);

}
```