mst.cpp

```
/* FILE: mst.cpp   last change: 11-Feb-2015   author: Romeo Rizzi
 * an implementation of Kruskal algorithm for problem mst.
 */

//#define NDEBUG   // NDEBUG definita nella versione che consegno
#include <cassert>
#ifndef NDEBUG
#  include <iostream>  // uso di cin e cout non consentito in versione finale
#endif
#include <fstream>
#include <algorithm>

using namespace std;

const int MAX_Q = 100;  int q;
const int MAX_N = 10000;  int n;
const int MAX_M = 1000000;  int m;

int uStack[MAX_Q], vStack[MAX_Q], weightStack[MAX_Q], valStack[MAX_Q];

struct edge_t { int nodeA, nodeB; long int weight; };
bool compare_edge_t(const edge_t& a, const edge_t& b) { return (a.weight < b.weight); }
edge_t edge[MAX_M]; // edges get numbered from 0, whereas nodes get numbered from 1 as in the text
of the exercise.
edge_t tree[MAX_N-1]; // stores the optimal solution.


// UNION FIND: (path-compression is more than enough since we need to spend m\log m for sorting the
 edges. No rank-fusion needed).
int rep[MAX_N +1]; // rep[v] = the representative of node v. (A node in the same connected componen
t as defined by the edges considered so far).
int find(int v) { if( rep[v] == v ) return v; else return rep[v] = find(rep[v]); }  // finds repres
entative and performs path compression

int main() {
  ifstream fin("input.txt"); assert( fin );
  fin >> n >> m >> q;
  for(int j = 0; j < m; j++) {
      fin >> edge[j].nodeA  >> edge[j].nodeB  >> edge[j].weight;
      if( j < q ) {
        uStack[j] = edge[j].nodeA;
        vStack[j] = edge[j].nodeB;
        weightStack[j] = edge[j].weight;
        valStack[j] = -1;
      }
  }
  sort(edge, edge +m, compare_edge_t);

// Kruskal algorithm
  // in the begginning there is no edge:
  for(int i = 1; i <= n; i++)  rep[i] = i;  // every node is on its own, whence it's its own repres
entative;

  // then the edges appear one by one, in non-decreasing order of weights:
  int opt_val = 0; int opt_card = 0;
  for(int j = 0; j < m; j++) {
    int repA = find(edge[j].nodeA); int repB = find(edge[j].nodeB);
    if( repA != repB ) {
      tree[opt_card++] = edge[j]; opt_val += edge[j].weight;
      rep[repA] = repB;
      for(int p = 0; p < q; p++)
        if( valStack[p] == -1)
          if( find(uStack[p]) == find(vStack[p]) )
             valStack[p] = weightStack[p] -edge[j].weight;
    }
  }

  ofstream fout("output.txt"); assert( fout );
  fout << opt_val << endl;
  for(int p = 0; p < q; p++)
    fout << valStack[p] << endl;

  fout.close();

  return 0;
}
```