```
namespace DSU {
  class dsu {
    private:

      const int n;
      int* root;

    public:
      dsu(int n) : n(n) {
        root = new int[n+3]();
      }

      int get(int x) { return root[x] ? root[x] = get(root[x]) : x; }

      void merge(int v,int u) {
        int rv = get(v);
        int ru = get(u);
        if(rv==ru) return ;
        root[rv] = ru;
      }

      ~dsu() {
//        delete[] root;
      }

      void clear() {
        for(int i = 0;i<n+2;i++) root[i] = 0;
      }

      void del() {
        delete[] root;
      }
  };
};

namespace MST {
  struct _edge_ {
    int v1,v2;
    int s,ind = -1;
    bool ok = true;

    _edge_(int v1,int v2,int s,int ind)
      : v1(v1),v2(v2),s(s),ind(ind) {};

    int get_other(int v) {
      return v1==v ? v2 : v1;
    }

    bool operator<(_edge_ w) const { return s< w.s; };
  };
```

```
  typedef pair<int,int> pii;
  class mst {
    public :
      vector<_edge_> mst_base;
      mst(int n,vector<_edge_>&es,int wwww = 0) : n(n) {
        tree = new vector<int>[n+2]();
        eds = es;
        create_dsu();
      }

      mst(int n,vector<_edge_>&es,DSU::dsu*d) : n(n),ds(d) {
        tree = new vector<int>[n+2]();
        for(auto w : es)
          if(d->get(w.v1) != d->get(w.v2)) {
            eds.push_back(w);
          }
        create_dsu();
      }

      void add_edge(_edge_ e) {
        e.v1 = ds->get(e.v1);
        e.v2 = ds->get(e.v2);
        for(auto w : tree[e.v1])
          if(eds[w].ok && eds[w].ind == e.ind)
          {
            ans += e.s - eds[w].s;
            eds[w].s = e.s;
            return ;
          }
        pii x = mx_cycle(e.v1,e.v2);
        if(x.first>e.s) {
          ans += e.s - x.first;
          eds.push_back(e);
          add_to_tree(eds.size()-1);
          eds[x.second].ok = false;
        }
      }

      int get_mst() { return ans; };

      ~mst() {
//        eds.clear();
//        delete[] tree;
      }

      void del() {
        delete[] tree;
        eds.clear();
      }

    private:
```

```
      int n;
      int ans ;
      vector<_edge_> eds;
      DSU::dsu*ds;
      vector<int> *tree;

      void create_dsu() {
        ans = 0;
        sort(eds.begin(),eds.end());
        using namespace DSU;
        dsu d (n) ;

        for(int i = 0;i<eds.size();i++) {
          _edge_&w = eds[i];
          if(d.get(ds->get(w.v1)) != d.get(ds->get(w.v2)))
            ans += w.s,d.merge(ds->get(w.v1),ds->get(w.v2)),add_to_tree(i);
//          cout << "\t" << w.v1 << " " << w.v2 << " " << w.s << " " << ans <<
 endl;
        }
        d.del();
      }

      void make_edges() {
      }

      void add_to_tree(int i) {
        tree[ds->get(eds[i].v1)].push_back(i);
        mst_base.push_back(eds[i]) ;
        tree[ds->get(eds[i].v2)].push_back(i);
      }

      pii mx_cycle(int v,int u,int p = -1,int val = 0) {
        if(v==u) return {0,0};
        pii mx = {-1,-1};
        for(auto w : tree[v])
          if(eds[w].ok && ds->get(eds[w].get_other(v)) != p) {
            pii zz = mx_cycle(ds->get(eds[w].get_other(v)),u,v,val + eds[w].s);
            if(~zz.first) mx = max(zz,{eds[w].s,w});
          }
        return mx;
      }
  };
};
```