```cpp
#include "../../bits/stdc++.h"
#include "../../Structure/union_find.hpp"
#include "../graph.hpp"

// verified: http://judge.u-aizu.ac.jp/onlinejudge/review.jsp?rid=3560507

// ブルーフカ法による最小全域木
// 各頂点を1頂点からなる木として開始
// 各ステップでは各木について他の木へ伸びる辺の内，コストが最小であるものを選択し連結する
// O(ElogV)
int boruvka(const Graph &g)
{
    using T = int;
    int n = g.size();
    // index[i] := i が所属する連結成分のインデックス
    std::vector<int> index(n);
    // index の逆引き
    std::vector<int> rev(n);
    UnionFind uf(n);
    T ret = T();
    // f := 連結成分の個数を引数とし，各連結成分について他の連結成分へ伸びる辺の内コスト最小のものを (cost, dst) ペアで返す
    // とりあえず verify 用の関数を書いておく
    // ここから
    std::vector<Edge> es;
    for (int i = 0; i < n; i++)
        for (const auto &e : g[i])
            es.push_back(e);
    auto f = [&](int sz) {
        std::vector<std::pair<int, int>> v(sz, {INF, -1});
        for (const auto &e : es)
        {
            if (index[e.from] == index[e.to])
                continue;
            v[index[e.from]] = std::min(v[index[e.from]], std::make_pair(e.cost, index[e.to]));
            v[index[e.to]] = std::min(v[index[e.to]], std::make_pair(e.cost, index[e.from]));
        }
        return v;
    };
    // ここまで
    while (uf.size(0) != n)
    {
        int idx = 0;
        for (int i = 0; i < n; i++)
        {
            if (uf.root(i) == i)
            {
                index[i] = idx;
                rev[idx] = i;
                idx++;
            }
        }
        for (int i = 0; i < n; i++)
        {
            index[i] = index[uf.root(i)];
        }
        auto v = f(idx);
        bool check = false;
        for (int i = 0; i < idx; i++)
        {
            if (v[i].second != -1 && !uf.find(rev[i], rev[v[i].second]))
            {
                ret += v[i].first;
                uf.unite(rev[i], rev[v[i].second]);
                check = true;
            }
        }
        assert(check);
    }
    return ret;
}

```