Maximum flow

ESTE CÓDIGO NO LO ESCRIBÍ YO.

Dada una gráfica dirigida con capacidades, una fuente y un pozo, encuentra el máximo flujo. Puede usarse para resolver mínimo corte, con el teorema de mínimo corte y máximo flujo, simplemente considerando todas las parejas de flujo $(0, v)$ con $v > 0$.

```
// This program was written by jaehyunp and taken from:
// https://github.com/jaehyunp/stanfordacm/blob/master/code/

#include <algorithm>
#include <iostream>
#include <queue>
#include <vector>

using namespace std;

struct Edge
{
    long from, to, cap, flow, index;
    Edge(long from, long to, long cap, long flow, long index)
        : from(from), to(to), cap(cap), flow(flow), index(index)
    {}
};

class PushRelabel
{
public:
    PushRelabel(long N)
        : N(N), G(N), excess(N), dist(N), active(N), count(2 * N)
    {}

    void AddEdge(long from, long to, long cap)
    {
        G[from].push_back(Edge(from, to, cap, 0, G[to].size()));

        if (from == to)
            G[from].back().index++;

        G[to].push_back(Edge(to, from, 0, 0, G[from].size() - 1));
    }

    long GetMaxFlow(long s, long t)
    {
        count[0] = N - 1;
        count[N] = 1;
        dist[s] = N;
        active[s] = active[t] = true;

        for (long i = 0; i < G[s].size(); i++)
        {
            excess[s] += G[s][i].cap;
            Push(G[s][i]);
        }

        while (!Q.empty())
        {
            long v = Q.front();
            Q.pop();
            active[v] = false;
            Discharge(v);
        }

        long totflow = 0;

        for (long i = 0; i < G[s].size(); i++)
            totflow += G[s][i].flow;

        return totflow;
    }

private:
    long N;
    vector<vector<Edge>> G;
    vector<long> excess;
    vector<long> dist, active, count;
    queue<long> Q;

    void Enqueue(long v)
    {
        if (!active[v] && excess[v] > 0)
        {
            active[v] = true;
            Q.push(v);
        }
    }

    void Push(Edge& e)
    {
        long amt = long(min(excess[e.from], long(e.cap - e.flow)));

        if (dist[e.from] <= dist[e.to] || amt == 0)
            return;

        e.flow += amt;
        G[e.to][e.index].flow -= amt;
        excess[e.to] += amt;
        excess[e.from] -= amt;
        Enqueue(e.to);
    }

    void Gap(long k)
    {
        for (long v = 0; v < N; v++)
        {
            if (dist[v] < k)
                continue;

            count[dist[v]]--;
            dist[v] = max(dist[v], N + 1);
            count[dist[v]]++;
            Enqueue(v);
        }
    }

    void Relabel(long v)
    {
        count[dist[v]]--;
        dist[v] = 2 * N;

        for (long i = 0; i < G[v].size(); i++)
            if (G[v][i].cap - G[v][i].flow > 0)
                dist[v] = min(dist[v], dist[G[v][i].to] + 1);

        count[dist[v]]++;
        Enqueue(v);
    }

    void Discharge(long v)
    {
        for (long i = 0; excess[v] > 0 && i < G[v].size(); i++)
            Push(G[v][i]);

        if (excess[v] > 0)
        {
            if (count[dist[v]] == 1)
                Gap(dist[v]);
            else
                Relabel(v);
        }
    }
};

int main()
{
    PushRelabel G(5);

    G.AddEdge(0, 1, 8);
    G.AddEdge(0, 2, 3);
    G.AddEdge(1, 2, 2);
    G.AddEdge(1, 4, 4);
    G.AddEdge(1, 3, 1);
    G.AddEdge(3, 4, 4);

    cout << "Max flow: " << G.GetMaxFlow(0, 4) << endl;

    return 0;
}
```

Output:

```
    Max flow: 5
```