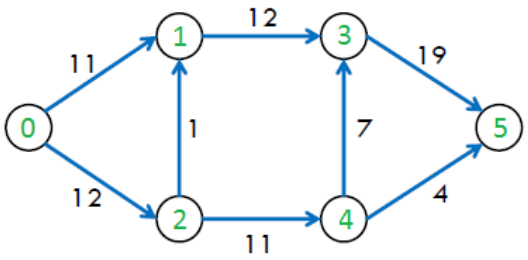
1. Máximo flujo.

Consideremos un grafo dirigido $G = (V, E)$ donde cada arista (u, v) tiene asociada una capacidad $c(u, v) > 0$. Un flujo de s a t es una función que a cada arista le asigna un número $f(u, v)$ que satisface

- $f(u, v) \leq c(u, v)$.
- Para cualquier vértice $v \neq s, t$, el flujo que entra es igual al flujo que sale; s solo tiene flujo saliente y t solo tiene flujo entrante.

El flujo total es el flujo que sale de s.



1.1. Algoritmo de Edmonds-Karp

Complejidad: $O(\text{mín}\{|V||E|^2, |E| \max f\})$.

```
1  #include <iostream>
2  #include <algorithm>
3  #include <vector>
4  #include <queue>
5  using namespace std;
6
7  #define maxn 100000 //Maximo numero de vertices.
8  typedef int T;      //Tipo de dato del flujo.
9
10 struct edge {
11         int to;           //Destino.
12         T capacity, flow; //Capacidad, flujo.
13         edge *rev;        //Arista invertida.
14
15         edge(int _to, T _capacity, T _flow, edge *_rev) {
16                 to = _to; capacity = _capacity; flow = _flow; rev = _rev;
17         }
18 };
19
20 int V, E;                   //Numero de vertices y aristas.
21 vector<edge*> graph[maxn]; //Aristas.
22
23 //Calcula el flujo maximo de s a t.
24 T EdmondsKarp(int s, int t) {
25     T flow = 0;
26     edge *pred[maxn];
27
28     do {
29         //Realiza una BFS desde s hasta t.
```

```
30          queue<int> Q;
31          Q.push(s);
32          fill(pred, pred + V, nullptr);
33
34          while (!Q.empty()) {
35              int curr = Q.front();
36              Q.pop();
37              for (edge *e : graph[curr])
38                  if (pred[e->to] == nullptr && e->to != s && e->capacity > e->
                        flow) {
39                      pred[e->to] = e;
40                      Q.push(e->to);
41                  }
42          }
43
44          //Encontramos un camino de aumento.
45          if (pred[t] != nullptr) {
46              T df = 1e9;
47              for (edge *e = pred[t]; e != nullptr; e = pred[e->rev->to])
48                  df = min(df, e->capacity - e->flow);
49              for (edge *e = pred[t]; e != nullptr; e = pred[e->rev->to]) {
50                  e->flow += df;
51                  e->rev->flow -= df;
52              }
53              flow += df;
54          }
55      }
56      while (pred[t] != nullptr);
57
58      return flow;
59  }
60
61  int main() {
62      ios_base::sync_with_stdio(0); cin.tie();
63      cin >> V >> E;
64
65      //Lee la informacion de las aristas.
66      for (int i = 0; i < E; i++) {
67          int from, to;
68          T capacity;
69          cin >> from >> to >> capacity;
70
71          graph[from].push_back(new edge(to, capacity, 0, nullptr));
72          graph[to].push_back(new edge(from, 0, 0, graph[from].back()));
73          graph[from].back()->rev = graph[to].back();
74      }
75
76      cout << "Flujo maximo: " << EdmondsKarp(0, V - 1) << '\n';
77
78      //Imprime la configuracion del flujo.
79      for (int i = 0; i < V; i++)
80          for (edge *e : graph[i]) {
81                  if (e->capacity > 0)
82                          cout << i << ' ' << e->to << ": " << e->flow << '/' <<
                                e->capacity <<'\n';
83                  delete e;
```

```
84            }
85
86        return 0;
87    }
```

Entrada	**Salida**
6 8 0 1 11 0 2 12 1 3 12 2 1 1 2 4 11 4 3 7 3 5 19 4 5 5	Flujo maximo: 23 0 1: 11/11 0 2: 12/12 1 3: 12/12 2 1: 1/1 2 4: 11/11 3 5: 18/19 4 3: 6/7 4 5: 5/5