

# 1. Caminos más cortos.

Consideremos un grafo  $G = (V, E)$  donde cada arista  $(u, v)$  tiene una longitud  $d(u, v) > 0$ . El camino más corto entre dos vértices es aquel que minimiza la longitud total del camino.

## 1.1. Algoritmo de Dijkstra

Complejidad:  $O((|E| + |V|) \log |V|)$ .

```

1  #include <iostream>
2  #include <algorithm>
3  #include <vector>
4  #include <queue>
5  #include <utility>
6  using namespace std;
7
8  #define maxn 100000 //Maximo numero de vertices.
9
10 typedef pair<int, int> edge;
11 #define length first
12 #define to second
13
14 int V, E; //Numero de vertices y aristas.
15 vector<edge> graph[maxn]; //Aristas.
16
17 int s; //Vertice inicial.
18 int dist[maxn], pred[maxn]; //Distancia desde s y predecesor en el camino.
19 bool vis[maxn]; //Visitado.
20
21 //Encuentra el camino mas corto desde un vertice a todos los demas.
22 void Dijkstra() {
23     fill_n(dist, V, 1e9);
24     fill_n(pred, V, -1);
25     fill_n(vis, V, false);
26     dist[s] = 0;
27
28     priority_queue<edge> pq;
29     pq.push(edge(dist[s], s));
30
31     while (!pq.empty()) {
32         int curr = pq.top().to;
33         pq.pop();
34         vis[curr] = true;
35
36         for (edge e : graph[curr])
37             if (!vis[e.to] && dist[curr] + e.length < dist[e.to]) {
38                 dist[e.to] = dist[curr] + e.length;
39                 pred[e.to] = curr;
40                 pq.push(edge(-dist[e.to], e.to));
41             }
42     }
43 }
44
45 int main() {
46     ios_base::sync_with_stdio(0); cin.tie();
47     cin >> V >> E;

```

```
48
49 //Lee la informacion de las aristas.
50 for (int i = 0; i < E; ++i) {
51     int u, v, d;
52     cin >> u >> v >> d;
53     graph[u].push_back(edge(d, v));
54     graph[v].push_back(edge(d, u));
55 }
56
57 //Imprime la configuracion.
58 cin >> s;
59 Dijkstra();
60 for (int i = 0; i < V; ++i)
61     cout << i << ": " << pred[i] << " " << dist[i] << "\n";
62
63 return 0;
64 }
```

Entrada	Salida
6 7	0: -1 0
0 1 4	1: 0 4
1 3 10	2: 0 2
3 5 11	3: 4 9
1 2 5	4: 2 5
2 0 2	5: 3 20
2 4 3	
4 3 4	
0	

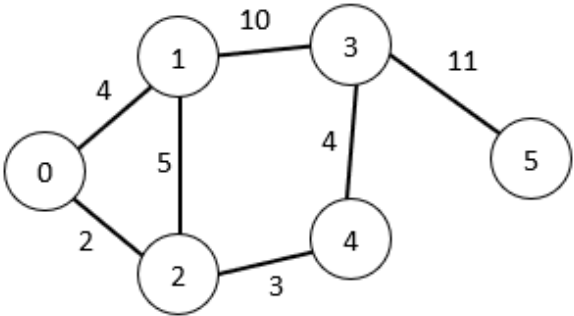


Figura 1: Ejemplo.

## 2. 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  $s$ .

### 2.1. Algoritmo de Edmonds-Karp

Complejidad:  $O(|V||E|^2)$ .

```

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 int s, t;             //Fuente y sumidero.
22 vector<edge*> graph[maxn]; //Aristas.
23
24 //Calcula el flujo maximo de s a t.
25 T EdmondsKarp() {
26     T flow = 0;
27     edge *pred[maxn];
28
29     do {
30         //Realiza una BFS desde s hasta t.
31         queue<int> Q;
32         Q.push(s);
33         fill_n(pred, V, nullptr);
34
35         while (!Q.empty()) {
36             int curr = Q.front();
37             Q.pop();
38             for (edge *e : graph[curr])

```

```

39         if (pred[e->to] == nullptr && e->to != s && e->capacity > e->
40             flow) {
41             pred[e->to] = e;
42             Q.push(e->to);
43         }
44     }
45     //Encontramos un camino de aumento.
46     if (pred[t] != nullptr) {
47         T df = 1e9;
48         for (edge *e = pred[t]; e != nullptr; e = pred[e->rev->to])
49             df = min(df, e->capacity - e->flow);
50         for (edge *e = pred[t]; e != nullptr; e = pred[e->rev->to]) {
51             e->flow += df;
52             e->rev->flow -= df;
53         }
54         flow += df;
55     }
56 }
57 while (pred[t] != nullptr);
58
59 return flow;
60 }
61
62 int main() {
63     ios_base::sync_with_stdio(0); cin.tie();
64     cin >> V >> E;
65
66     //Lee la informacion de las aristas.
67     for (int i = 0; i < E; ++i) {
68         int from, to;
69         T capacity;
70         cin >> from >> to >> capacity;
71
72         graph[from].push_back(new edge(to, capacity, 0, nullptr));
73         graph[to].push_back(new edge(from, 0, 0, graph[from].back()));
74         graph[from].back()->rev = graph[to].back();
75     }
76
77     cin >> s >> t;
78     cout << "Flujo maximo: " << EdmondsKarp() << '\n';
79
80     //Imprime la configuracion del flujo.
81     for (int i = 0; i < V; ++i)
82         for (edge *e : graph[i]) {
83             if (e->capacity > 0)
84                 cout << i << ' ' << e->to << ": " << e->flow << ' / ' << e->
85                     capacity << '\n';
86             delete e;
87         }
88     return 0;
89 }

```

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

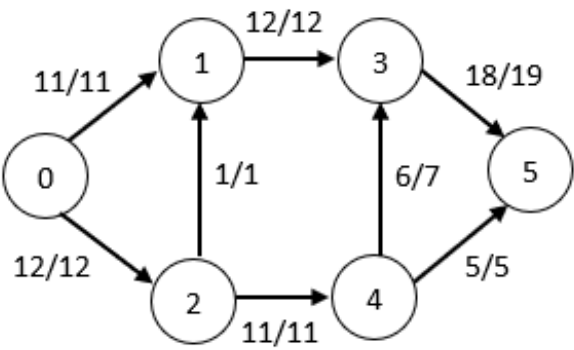


Figura 2: Ejemplo.

### 3. Emparejamiento máximo.

Consideremos un grafo bipartito  $G = (U \cup V, E)$ . Un emparejamiento de  $G$  es un subgrafo en donde cada vértice pertenece a lo más a una arista.

#### 3.1. Algoritmo de Hopcroft-Karp

Complejidad:  $O(|E|\sqrt{|V|})$ .

```
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
9  int U, V, E;          //Numero de vertices en cada lado y de aristas.
10 vector<int> graph[maxn]; //Aristas que van de U a V.
11
12 int pairU[maxn], pairV[maxn], dist[maxn]; //Pares de vertices en el
    emparejamiento.
13
14 //Verifica si existe un camino de aumento.
15 bool BFS() {
16     queue<int> Q;
17     for (int u = 1; u <= U; ++u) {
18         if (!pairU[u]) {
19             dist[u] = 0;
20             Q.push(u);
21         }
22         else
23             dist[u] = 1e9;
24     }
25     dist[0] = 1e9;
26
27     while (!Q.empty()) {
28         int u = Q.front();
29         Q.pop();
30         if (dist[u] < dist[0])
31             for (int v : graph[u])
32                 if (dist[pairV[v]] == 1e9) {
33                     dist[pairV[v]] = dist[u] + 1;
34                     Q.push(pairV[v]);
35                 }
36     }
37     return (dist[0] != 1e9);
38 }
39
40 //Verifica si existe un camino de aumento que comience en u.
41 bool DFS(int u) {
42     if (u) {
43         for (int v : graph[u])
44             if (dist[pairV[v]] == dist[u] + 1 && DFS(pairV[v])) {
45                 pairV[v] = u;
46                 pairU[u] = v;
```

```

47         return true;
48     }
49
50     dist[u] = 1e9;
51     return false;
52 }
53 return true;
54 }
55
56 //Busca un emparejamiento maximo.
57 int HopcroftKarp() {
58     int size = 0;
59     fill_n(pairU, U, 0);
60     fill_n(pairV, V, 0);
61     while (BFS())
62         for (int u = 1; u <= U; ++u)
63             if (!pairU[u] && DFS(u))
64                 size++;
65     return size;
66 }
67
68 int main() {
69     ios_base::sync_with_stdio(0); cin.tie();
70     cin >> U >> V >> E;
71
72     //Lee las aristas. Los vertices estan indexados en 1.
73     for (int i = 0; i < E; ++i) {
74         int u, v;
75         cin >> u >> v;
76         graph[u].push_back(v);
77     }
78
79     //Imprime la configuracion del emparejamiento.
80     cout << "Emparejamiento: " << HopcroftKarp() << '\n';
81     for (int u = 1; u <= U; ++u)
82         if (pairU[u])
83             cout << u << " - " << pairU[u] << '\n';
84
85     return 0;
86 }

```

Entrada	Salida
5 4 8	Emparejamiento: 3
1 1	1 - 1
2 1	2 - 3
2 3	3 - 2
3 2	
3 3	
3 4	
4 3	
5 3	

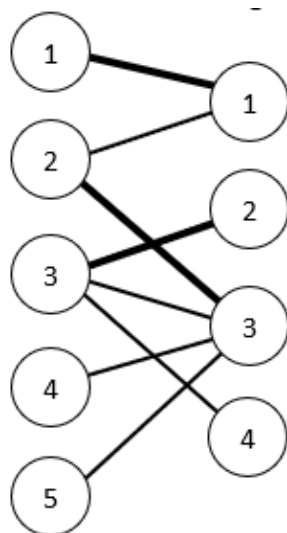


Figura 3: Ejemplo.