Maximum flow - Ford-Fulkerson and Edmonds-Karp

Table of Contents

The Edmonds-Karp algorithm is an implementation of the Ford-Fulkerson method for computing a maximal flow in a flow network.

Flow network

First let's define what a **flow network**, a **flow**, and a **maximum flow** is.

A **network** is a directed graph G with vertices V and edges E combined with a function c, which assigns each edge $e \in E$ a non-negative integer value, the **capacity** of e. Such a network is called a **flow network**, if we

additionally label two vertices, one as **source** and one as **sink**.

A **flow** in a flow network is function f, that again assigns each edge e a non-negative integer value, namely the flow. The function has to fulfill the following two conditions:

The flow of an edge cannot exceed the capacity.

$$f(e) \leq c(e)$$

And the sum of the incoming flow of a vertex u has to be equal to the sum of the outgoing flow of u except in the source and sink vertices.

$$\sum_{(v,u) \in E} f((v,u)) = \sum_{(u,v) \in E} f((u,v))$$

The source vertex s only has an outgoing flow, and the sink vertex t has only incoming flow.

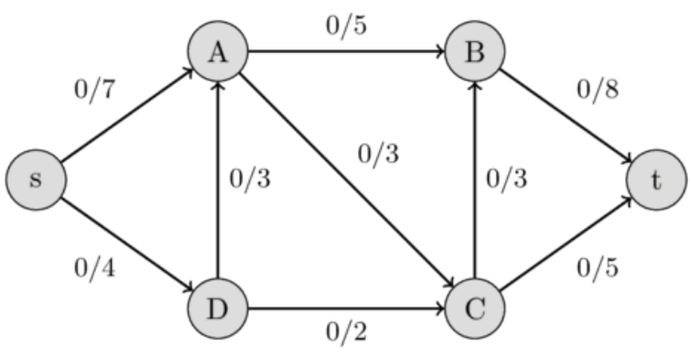
It is easy to see that the following equation holds:

$$\sum_{(s,u) \in E} f((s,u)) = \sum_{(u,t) \in E} f((u,t))$$

A good analogy for a flow network is the following visualization: We represent edges as water pipes, the capacity of an edge is the maximal amount of water that can flow through the pipe per second, and the flow of an edge is the amount of water that currently flows though the pipe per second. This motivates the first flow

condition. There cannot flow more water through a pipe than its capacity. The vertices act as junctions, where water comes out of some pipes, and distributes it in some way to other pipes. This also motivates the second flow condition. In each junction all the incoming water has to be distributed to the other pipes. It cannot magically disappear or appear. The source s is origin of all the water, and the water can only drain in the sink t.

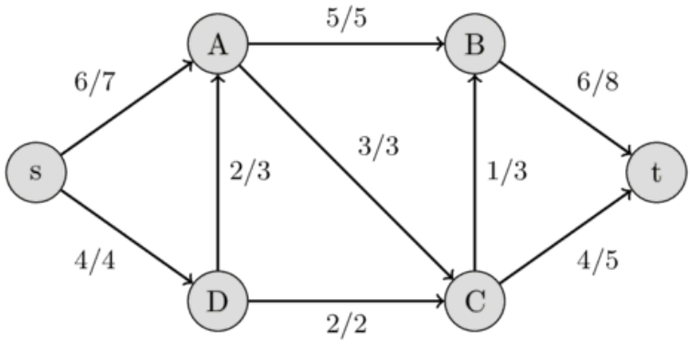
The following image show a flow network. The first value of each edge represents the flow, which is initially 0, and the second value represents the capacity.



The value of a flow of a network is the sum of all flows that gets produced in source s, or equivalently of the flows that are consumed in the sink t. A **maximal flow** is a flow with the maximal possible value. Finding this maximal flow of a flow network is the problem that we want to solve.

In the visualization with water pipes, the problem can be formulated in the following way: how much water can we push through the pipes from the source to the sink.

The following image show the maximal flow in the flow network.



Ford-Fulkerson method

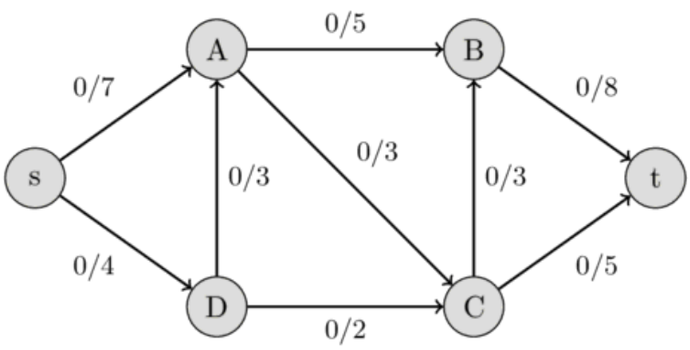
Let's define one more thing. A **residual capacity** of an directed edge is the capacity minus the flow. It should be noted that if there is a flow along some directed edge (u, v), than the reversed edge has capacity 0 and we can define the flow of it as $f((v, u)) = -f((u, v))$. This also defines the residual capacity for all reversed edges. From all these edges we can create a **residual network**, which is just a network with the same vertices and same edges, but we use the residual capacities as capacities.

The Ford-Fulkerson method works as follows. First we set the flow of each edge to zero. Then we look for an **augmenting path** from s to t. An augmenting path is simple path in the residual graph, i.e. along the edges whose residual capacity is positive. Is such a path is found, then we can add increase the flow along these edges. We keep on searching for augmenting paths and increasing the flow. Once there doesn't exists an augmenting path any more, the flow is maximal.

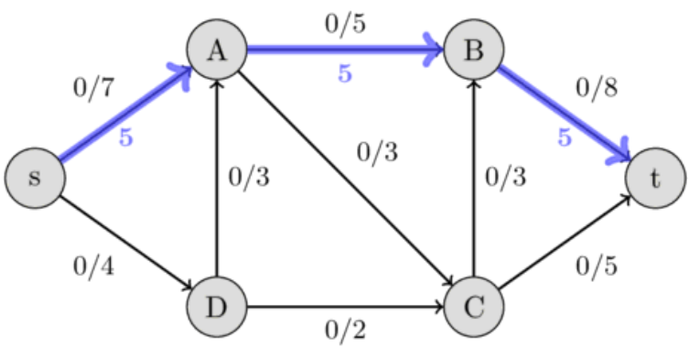
Let us specify in more detail, what increasing the flow along an augmenting path means. Let C be the smallest

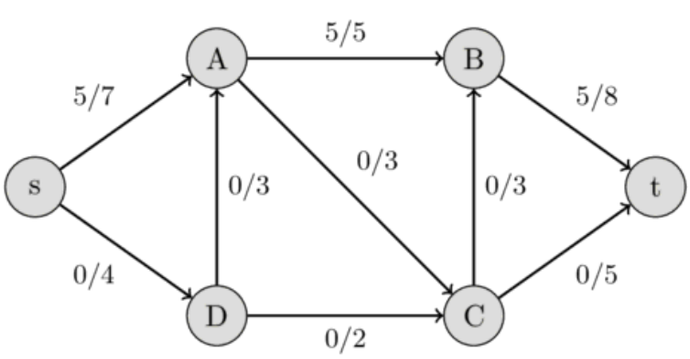
residual capacity of the edges in the path. Then we increase the flow in the following way: we update $f((u, v)) \mathrel{+}= C$ and $f((v, u)) \mathrel{-}= C$ for every edge (u, v) in the path.

Here is an example to demonstrate the method. We use the same flow network as above. Initially we start with a flow of 0.

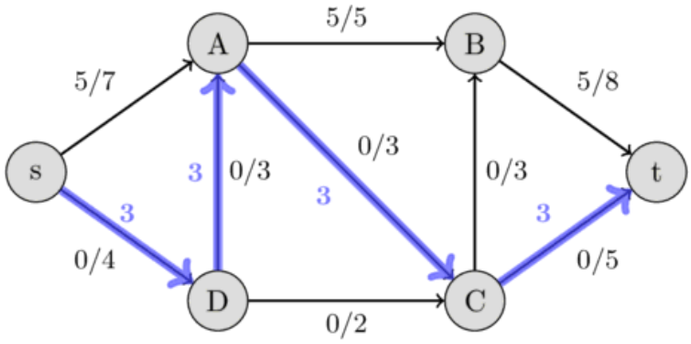


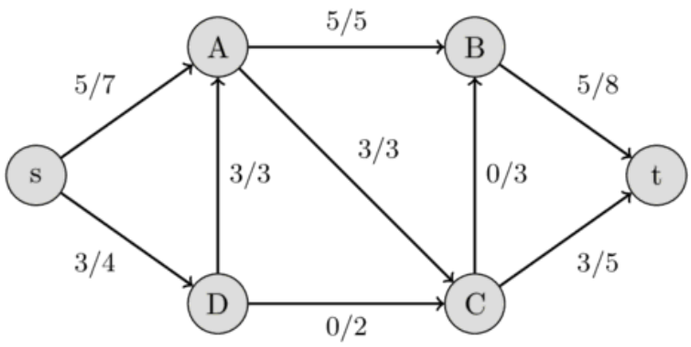
We can find the path $s - A - B - t$ with the residual capacities 7, 5 and 8. Their minimum is 5, therefore we can increase the flow along this path by 5. This gives a flow of 5 for the network.



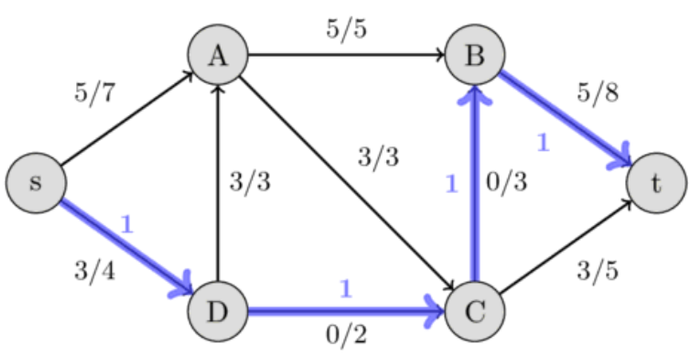


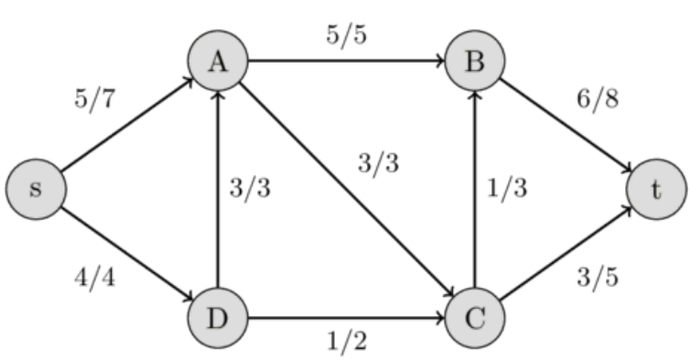
Again we look for an augmenting path, this time we find $s - D - A - C - t$ with the residual capacities 4, 3, 3 and 5. Therefore we can increase the flow by 3 and we get a flow of 8 for the network.



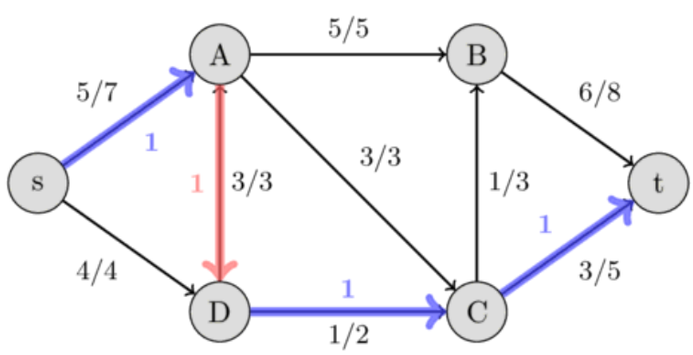


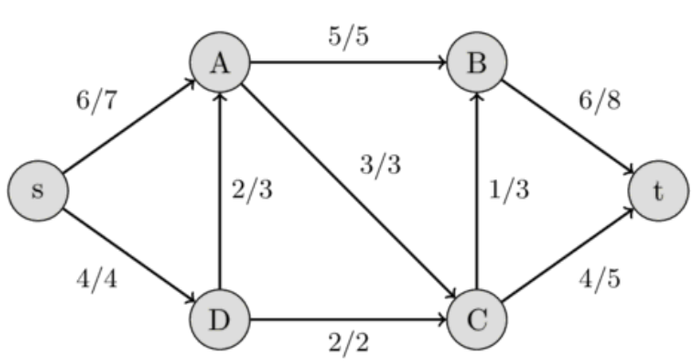
This time we find the path $s - D - C - B - t$ with the residual capacities 1, 2, 3 and 3, and we increase by 1.





This time we find the augmenting path $s - A - D - C - t$ with the residual capacities 2, 3, 1 and 2. We can increase by 1. But this path is very interesting. It includes the reversed edge (A, D). In the original flow network we are not allowed to send any flow from A to D. But because we already have a flow of 3 from D to A this is possible. The intuition of it is the following: Instead of sending a flow of 3 from D to A, we only send 2 and compensate this by sending an additional flow of 1 from s to A, which allows us to send an additional flow of 1 along the path $D - C - t$.





Now it is impossible to find an augmenting path between s and t, therefore this flow of 10 is the maximal possible. We have found the maximal flow.

It should be noted, that the Ford-Fulkerson method doesn't specify a method of finding the augmenting path. Possible approaches are using DFS or BFS which both work in $O(E)$. If all capacities of the network are

integers, then for each augmenting path the flow of the network increases by at least 1 (for more details see Integral flow theorem). Therefore the complexity of Ford-Fulkerson is $O(EF)$, where F is the maximal flow of the network. In case of rational capacities, the algorithm will also terminate, but the complexity is not bounded. In case of irrational capacities, the algorithm might never terminate, and might not even converge to the maximal flow.

Edmonds-Karp algorithm

Edmonds-Karp algorithm is just an implementation of the Ford-Fulkerson method that uses BFS for finding augmenting paths. The algorithm was first published by Yefim Dinitz in 1970, and later independently published by Jack Edmonds and Richard Karp in 1972.

The complexity can be given independently of the maximal flow. The algorithm runs in $O(VE^2)$ time, even for irrational capacities. The intuition is, that every time we find an augmenting path one of the edges becomes saturated, and the distance from the edge to s will be longer, if it appears later again in an augmenting path. And the length of a simple paths is bounded by V.

Implementation

The matrix `capacity` stores the capacity for every pair of vertices. `adj` is the adjacency list of the **undirected graph**, since we have also to use the reversed of

directed edges when we are looking for augmenting paths.

The function `maxflow` will return the value of the maximal flow. During the algorithm the matrix `capacity` will actually store the residual capacity of the network. The value of the flow in each edge will actually no stored, but it is easy to extent the implementation - by using an additional matrix - to also store the flow and return it.

```
int n;
vector<vector<int>> capacity;
vector<vector<int>> adj;

int bfs(int s, int t, vector<int>& parent) {
    fill(parent.begin(), parent.end(), -1);
    parent[s] = -2;
    queue<pair<int, int>> q;
    q.push({s, INF});

    while (!q.empty()) {
        int cur = q.front().first;
        int flow = q.front().second;
        q.pop();

        for (int next : adj[cur]) {
            if (parent[next] == -1 && capacity
                parent[next] = cur;
                int new_flow = min(flow, capac
                if (next == t)
                    return new_flow;
                q.push({next, new_flow});
```

```
            }
        }
    }

    return 0;
}

int maxflow(int s, int t) {
    int flow = 0;
    vector<int> parent(n);
    int new_flow;

    while (new_flow = bfs(s, t, parent)) {
        flow += new_flow;
        int cur = t;
        while (cur != s) {
            int prev = parent[cur];
            capacity[prev][cur] -= new_flow;
            capacity[cur][prev] += new_flow;
            cur = prev;
        }
    }

    return flow;
}
```

Integral flow theorem

The theorem simply says, that if every capacity in the network is integer, than also the flow in each edge will be integer in the maximal flow.

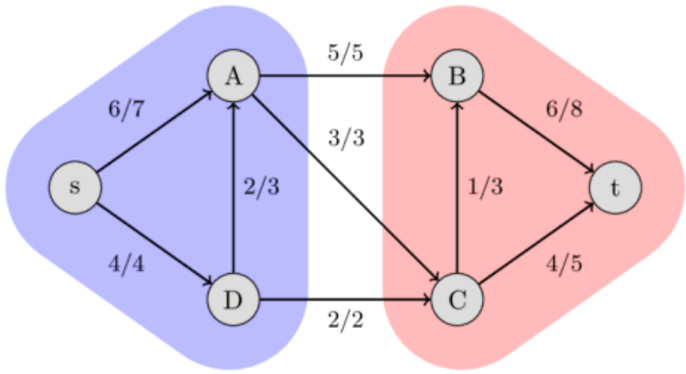
Max-flow min-cut theorem

A s-t-**cut** is a partition of the vertices of a flow network into two sets, such that a set includes the source s and the other one includes the sink t. The capacity of a s-t-cut is defined as the sum of capacities of the edges from the source side to the sink side.

Obviously we cannot send more flow from s to t than the capacity of any s-t-cut. Therefore the maximum flow is bounded by the minimum cut capacity.

The max-flow min-cut theorem goes even further. It says that the capacity of the maximum flow has to be equal to the capacity of the minimum cut.

In the following image you can see the minimum cut of the flow network we used earlier. It shows that the capacity of the cut $\{s, A, D\}$ and $\{B, C, t\}$ is $5 + 3 + 2 = 10$, which is equal to the maximum flow that we found. Other cuts will have a bigger capacity, like the capacity between $\{s, A\}$ and $\{B, C, D, t\}$ is $4 + 3 + 5 = 12$.



A minimum cut can be found after performing a maximum flow computation using the Ford-Fulkerson method. One possible minimum cut is the following: the set of all vertices that can be reached from s in the residual graph (using edges with positive residual capacity), and the set of all the other vertices. This partition can be easily found using DFS starting at s.