1. Networks

A **flow network** (also known as a transportation network) is a directed graph where each edge has a capacity, and each edge receives a flow.

A network can be used to model traffic in a computer network, mobility in a city, fluids in pipes, currents in an electrical circuit etc.

A network $N(G,s,t,c)$ consists of the following data:

- A directed graph $G(V, E)$.
- Two vertices s and t are specified; s is called the **source**; and t, the **sink**.
- The **capacity** function, c: $V \times V \rightarrow \mathbb{R}^+$. The positive real number, $c(v_1, v_2)$, is called the capacity of edge (v_1,v_2).
  - If $(v_1, v_2) \in E$ then $c(v_1, v_2) > 0$.
  - If $(v_1, v_2) \notin E$ then $c(v_1, v_2) = 0$.

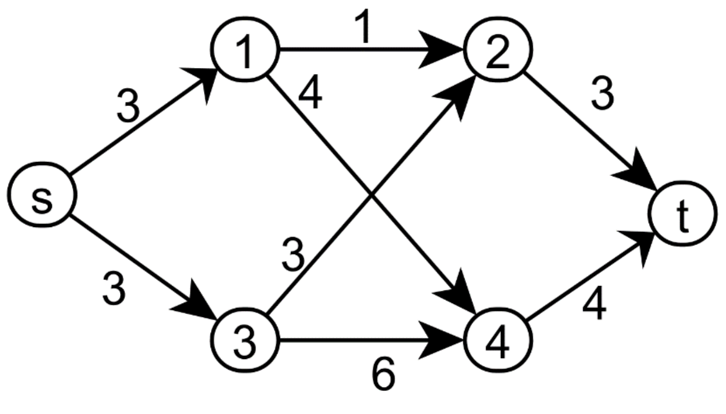


Figure 1

Consider the graph in Figure 1. It is a weighted graph, but it can represent a flow network, where the source is s, the sink is t and the capacities are the weights of the edges.

In real world applications you may expect to find multiple sources and multiple sinks. In this case, many flow problems can actually be solved with algorithms made for one source and one sink by creating a dummy source, with the capacity the total capacities of other sources then connect this dummy source to the real sources with edges with the capacity of each real source capacity. The same for the sink, you create a dummy sink and connect it with edges equal to the capacity of each real sink. So, then you can use algorithms for one-sink/one-source problems even for multi-sink/multi-source problems.

Note: this does not always work if there are other constraints on the problem, for example: "maximizing the minimum flow of any source or sink" or "distributing flow equally between sources/sinks". But the algorithms can generally be adapted to work for these constraints. We will not consider these particular cases.

2. Flows

A **flow** in a network $N=(G=(V,E), s, t, c)$ is a function $f: V\times V\rightarrow\mathbb{R}^{+}$. A flow $f(v_1, v_2)$ through the edge (v_1, v_2) in this network satisfies the following properties:

1) Capacity constraint:

$\forall v_1, v_2 \in V, 0 \leq f(v_1, v_2) \leq c(v_1, v_2)$

which means that a flow through an edge may not exceed the capacity of the edge.

2) Flow Conservation: $\forall v_2 \in \{V - \{s, t\}\}$,

$$\sum_{v_1\in V} f(v_1, v_2) = \sum_{v_1\in V} f(v_2, v_1)$$

In other words, the flow into the vertex v_2 equals the flow out of v_2 for any vertex v_2 except the source vertex s and the sink vertex t.

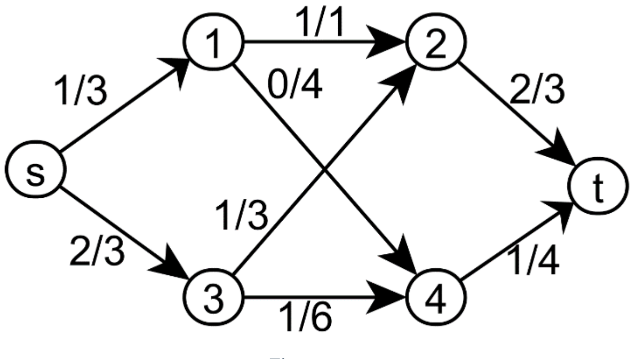


Figure 2

In Figure 2 you can see the network from Figure 1 to which some example flow values have been added. Observe that the 2 properties of the flow are respected.

In general, for a flow problem we are given the graph with fixed capacities, and we want to find some correct values for the flow – in other words the capacity for each edge is fixed, but the flow value on each edge is not. The flow is the variable while the capacity is the constant.

Usually, what we care about is maximizing the total flow – maximizing the amount of flow that is generated by the source and consumed by the sink. The value of these must be equal.

To properly define total flow, remember that a source and a sink can be chosen as any of the graph's vertices, so the source may have inbound edges and the sink may have outbound edges. If so, the value of the flow on those edges can be higher than 0. So, we must consider this when defining the total flow.

Let f be a flow on the network $N = (G=(V, E), s, t, c)$. We call $|f|$ the total flow of f.

$$|f| = \sum_{v\in V} f(s,v) - \sum_{v\in V} f(v,s) = \sum_{v\in V} f(v,t) - \sum_{v\in V} f(t,s)$$

That is, the total flow is the flow that exits the source without the flow that enters (i.e. the flow that is created by the source) which must be equal to the flow that enters the sink without the flow that exists the sink (i.e. the flow that is consumed by the sink)

In the example in Figure 4, the total flow is 3.
Consider also the flow network and its flow from Figure 3. Here the total flow is 8:

- if we look at the source, we have outbound edges (s, 1) and (s, 3) with flows 5 and 6 respectively. Thus total outgoing flow 11. But, if we look at the inbound edges, we have (2, s) and (4, s), with flows 1 and 2 respectively. Thus, total incoming flow is 3. Then the total flow created by the source is 11 – 3 = 8.
- if we look at the sink, we have inbound edges (4, t) and (2,t) with flows 4 and 7 respectively. Then total incoming flow is 11. But we also have the outbound edges (t, 1) and (t, 3) with flows 2 and 1 respectively. Thus, total outgoing flow of 3. So total flow consumed by t is 11 – 3 = 8, the same as the flow generated by the source.

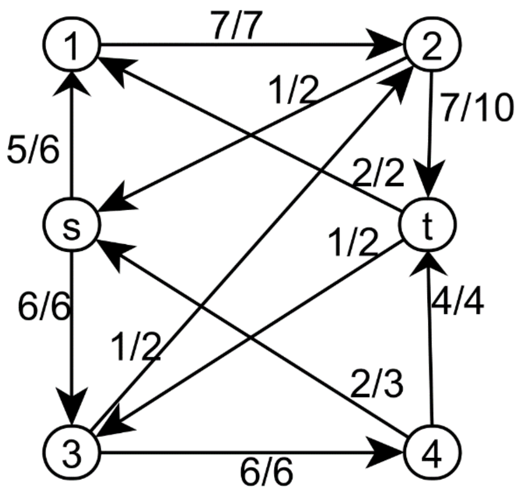


Figure 3

A flow is called *maximum* if it has the highest possible total flow value of all possible flows.

3. Residual network

Given a flow f on a network $N = (G=(V, E), s, t, c)$, the **residual network** with respect to flow f is the network $N_f = (G_f = (V, E_f), s, t, c_f)$ that has the same vertices, source and sink as N with the edges in E_f and the **residual capacities** (values given by c_f) defined as follows:

$\forall\ (v_1, v_2) \in E$:

a) if $f(v_1, v_2) < c(v_1,v_2)$ then $(v_1, v_2) \in E_f$ and $c_f(v_1, v_2) = c(v_1, v_2) - f(v_1, v_2)$.
These are called **forward edges**.

b) if $f(v_1, v_2) > 0$ then $\underline{(v_2, v_1)} \in E_f$ and $c_f(v_2, v_1) = f(v_1, v_2)$
These are called **backward edges**.

c_f is called the **residual capacity** of an edge.
In short:

- a forward edges represents the unused capacity of the edge from the original network – the amount of flow that could be added on the edge.
- the backward edges represent the used capacity of the network –

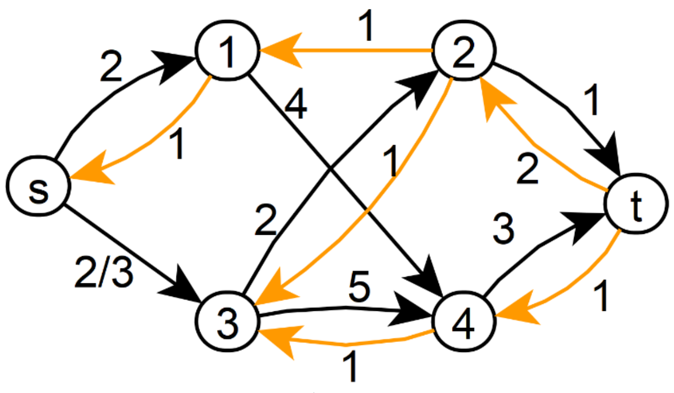


Figure 4

In Figure 4 is presented the residual network of the flow from Figure 2. Backward edges are highlighted in orange.

A path from the source *s* to destination *t* in a residual network N_f is called an **augmenting path** with respect to flow f.

The residual capacity of an augmenting path is the minimum residual capacity of any edge along the path. That is, if P is an augmenting path, then:

$$c_f(P) = min_{(v_1,v_2)\in P}\ \{c_f(v_1, v_2)\}$$

A flow is maximum if there is no augmenting path with respect to that flow.

Note: the residual network is a conceptual construction to build the algorithms upon, to understand what is going on. But when searching for an augmenting path, from a programming perspective, we do not actually need to create a new graph for our residual network. We can just simulate the edges of a residual network by going over our original graph and just checking the flow and capacity for every edge (to see if it would be a backward or forward edge inthe residual graph).

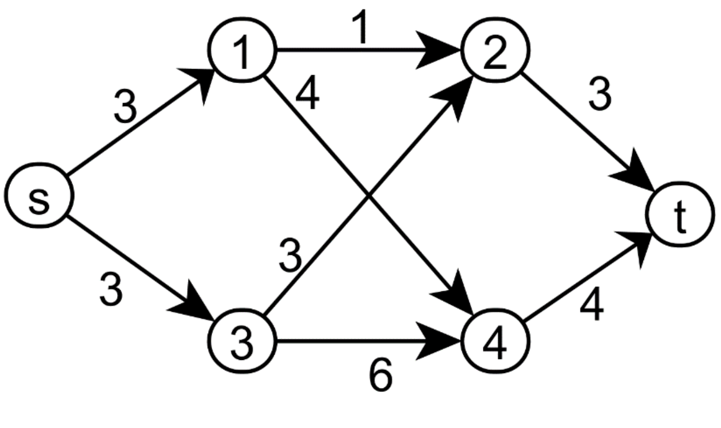
4. **Ford-Fulkerson algorithm** – to find the maximum flow in a graph

The idea of the algorithm is simple: from an empty flow find an augmenting path and augment the flow, repeat until there are no augmenting paths left.

Generic overview of the algorithm:

1. Flow starts at 0 for all edges.
2. Find an augmenting path for the network.
3. If found:
   a. add the residual capacity of the path to the flow of all edges in the path where the path goes through the forward edge.
   b. subtract the residual capacity of the path from the flow of all the edges where the path goes through the backward edge.
   c. go to step 2.
4. Else end algorithm, you found the maximum flow.

Let's try to find the maximum flow for the example given in Figure 1 with the Ford-Fulkerson algorithm. The first residual graph, with flow values 0:



Notice that this is the same as the original graph. Why?

Since all flow values are 0 that means we have only forward edges, for which the residual capacity is the unused flow on that edge – the actual capacity of the edge.
An augmenting path is any path from s to t. For example we might find P=(s, 3, 4, t). $c_f(P) = 3$.

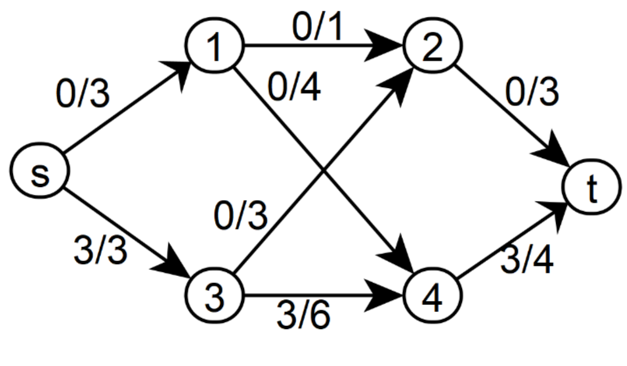


Figure 5

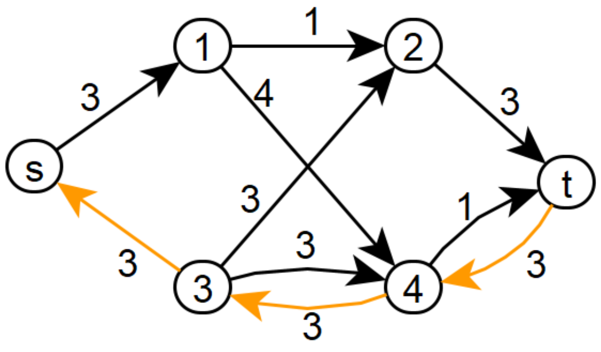
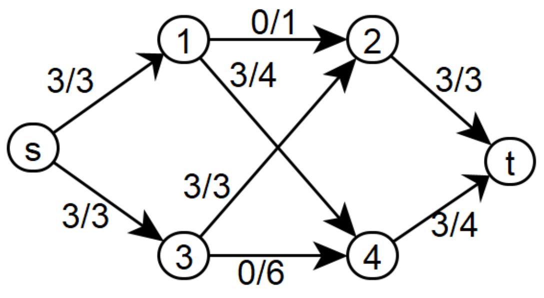


Figure 7

After augmenting the flow we have the network in Figure 5, with the residual graph in Figure 6. In this residual graph we might find the augmenting path P = (s, 1, 4, 3, 2, t). $c_f(P) = 3$.



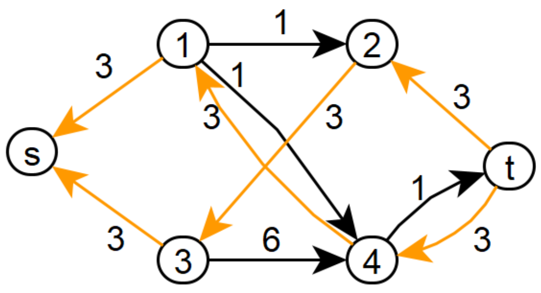
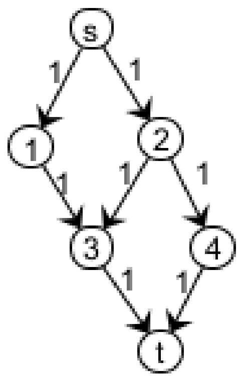


Figure 7

After augmenting the flow again we have the flow and its residual network presented in Figure 7. At this point, as you can see, there are no more augmenting paths (s does not have any outbound edges in the residual graph => impossible to have a path from s to t).

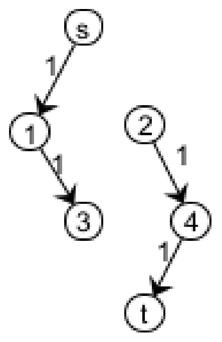
So then the resulting flow is a maximum flow, with total flow being 6.

At this point you may ask yourself: but why must it be so complicated? Do we really need the backward edges?
Consider the following example (left):



If, by chance, the first augmenting path we choose is (s,2,3,t) then, without backward edges we will have the following residual network (right):

In that residual network there are no augmenting paths. But the maximum flow has a total flow value of 2, if the flow goes (s,1,3,t) and (s,2,4,t). Thus, the algorithm would not work correctly.



With backward edges ((2, s), (3, 2) and (t, 3)) we would still have an augmenting path from s to t – (s,1,3,2,4,t), therefore the algorithm finds the correct maximum flow.

Time complexity of Ford Fulkerson algorithm: Finding a path from s to t with a traversal is O(n+m). In worst case we only increase the total flow by one with every augmenting path => meaning in the worst case we have to do a traversal max_flow number of times => O(max_flow * (m+n)).