Solution background

Flow Network: a directed graph, in which every edge has a certain capacity *c* associated with it, a starting vertex (the *source*), and an ending vertex (*the sink*).
For every edge (u, v) we define a capacity = c(u, v).

- If there is an edge (u, v) then there won't be an edge (v, u) (in the reverse direction).
- For every vertex other than the source and the sink, the sum of the flow entering it must be equal to the flow leaving it
- "Flow in = Flow out"

Flow
A function f(u, v) that takes two vertices and return the "flow" between them. f(u, v) = 0 if there is no edge (u, v) in the graph.

WARNING:
Sometimes people say "flow" and they mean "the flow in the edge (u, v)". Sometimes people say "flow" meaning the "Flow network" which is the whole directed graph. And sometimes people say flow as flow reaching the sink.

the sink. A ***flow*** in G is a real-valued function $f : V \times V \to \mathbb{R}$ that satisfies the following two properties:

Capacity constraint: For all $u, v \in V$, we require $0 \le f(u,v) \le c(u,v)$.

Flow conservation: For all $u \in V - \{s,t\}$, we require

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

When $(u,v) \notin E$, there can be no flow from u to v, and $f(u,v) = 0$.

Maxflow problem:

Given a network in a graph **G = (V, E)**, find the maximum "flow" that can reach the sink, that is:

Find a **f_i** for every **e in E** such that **f_i <= c_i** where **c_i** is the flow capacity of the i'th edge. Also the sum of the flow entering each vertex must be equal to the sum of the flow leaving that vertex, except the source and the sink, that is, every flow used reaches the sink.

Graphs with parallel edges

Parallel edges: a Graph with edges in both directions: (u, v) and (v, u) at the same time. You can convert this graph to a graph without parallel edges by creating an intermediary vertex Y such that:
Remove edge (u, v)
Add edges (u, X), (X, v),
Keep (v, u)

Max flow problem with multiple sources and sink

This problem is also the same problem as the max flow problem with only one source and only one sink. You can just create a **super source S** with edges (S, si) for every source vertex, with capacity = INFINITY.
Also create a **super sink T** with edges (ti, T) for every sink vertex, with capacity = INFINITY.

Residual network: has the same vertices as the original network, and one or two edges for each edge in the original.

If the flow along the edge x-y is less than the capacity there is a forward edge x-y with a capacity equal to the difference between the capacity and the flow (this is called the residual capacity), and if the flow is positive there is a backward edge y-x with a capacity equal to the flow on x-y

TL:DR
Let **(u, v, f, c)** such that there is a flow **f** in a edge **(u, v)** with capacity **c**, in a given network. In the *residual network* there will be an udge (u, v) or (v, u) according to the cases:

- **(u, v, c - f, c)**
- **(v, u, f, c)** if **f > 0**

Formally:

and sink t. Let f be a flow in G, and consider a pair of vertices $u, v \in V$. We define the ***residual capacity*** $c_f(u, v)$ by

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E\ , \\ f(v,u) & \text{if } (v,u) \in E\ , \\ 0 & \text{otherwise}\ . \end{cases} \tag{26.2}$$

Example:

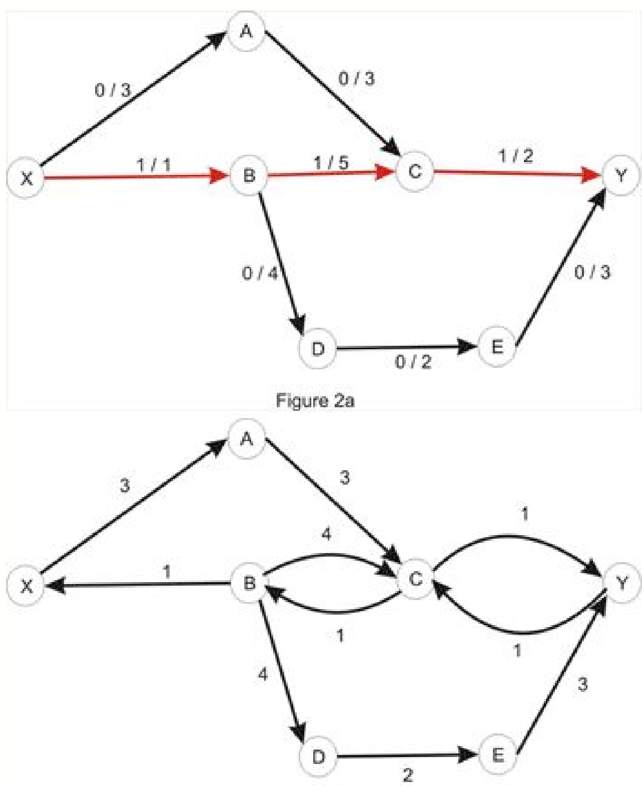

Figure 2a

Figure 2b - The residual network of the network in 2a

Augmenting path: A path from the source to the sink in the residual network, whose purpose is to increase the flow in the original one.

The path capacity of an augmenting path is the minimum capacity of an edge along that path. This is called **residual capacity of an augmenting path.** This is the minimum edge "weight" on the path **p.**

smallest residual capacity on this path is $c_f(v_2, v_3) = 4$. We call the maximum amount by which we can increase the flow on each edge in an augmenting path p the ***residual capacity*** of p, given by

$$c_f(p) = \min\{c_f(u, v) : (u, v) \text{ is on } p\} .$$

Ideia to solve the problem:

1. Start with the original graph
2. Set the current residual network
3. Find any augmenting path from the source to the sink, in the residual network. If there isn't such a path, the algorithm is ended.
4. Add the found augmenting path to the current flow, update de forward and backward edges.
5. Go back to step 2

What is the interpretation of taking a "reverse edge" in the augmenting path?

Suppose you take a path **p** consisting only of "direct edges" (edge (u, v) is in G).

- **p = s -> … -> u - > v -> … > t**

So, in the residual network there will be an edge **(v, u)**, but this edge isn't in the original graph.

Let **q** be an augmenting path from the source to the sink in the residual graph such that

- **q = s -> … > v -> u -> x -> … t**

In this case you took a reverse edge to reach the sink. The interpretation of that is that there is certainly a path from **v** to **t**, so you can use that path too. Since you took the reverse edge **(v, u)**, you reverted that flow, and that flow now will go to the path **s -> … u -> x … -> t** instead of **s -> … u -> v … > t** so you're actually just changing the flow from an edge to another edge.

Why taking augmenting path always increase the flow?

Let p be an augmenting path in the residual network, **p = s -> … -> u -> v > .... > t**

1. If there is no reverse edge in **p** than this certainly increases the current flow because you're just pushing more flow through a path in the original graph which still have capacity for that.
2. If you take a reverse edge in **p** it means that there isn't that edge in the original graph so we must analyse that path more carefully. The interpretation of the reverse

edge **(v, u)** in the augmenting path **p = s -> … -> v -> u -> x -> … -> t** is that you actually found two paths in the original graph:

a. A path **s -> … -> u -> v … -> t** in the original graph, which has some flow going through it.
b. And a path **s -> … -> u -> x … -> t** in the original graph, which is capable of running some flow.

You found path **b** in the residual network. The interpretation of that path is that instead of using vertex **u** to send flow to vertex **v**, you can send flow to vertex **x** from vertex **u**. If you do that, vertex **v** now have some empty space which you can used to send flow through it.

So, you're basically reverting the flow on some edge, to make space for another flow which will increase the total flow reaching the sink.

Corollary 26.3 (CLRS)

Let G = (V, E) be the original graph, and **f** a flow in that graph. Let **f'** be a flow in the residual graph of G.
Yo can augment the original flow **f** by **f'**:
|f ^ f'| = |f| + |f'| > |f|
That is, taking an augmenting path will increase your flow by at least **1.**

Correctness proof background

Cut
A cut **(S, T)** of a flow network **G = (V, E)** is a partition of **V** such that **V = S + T** and the source is in S, and the sink is in T.

Net flow
The flow across the cut is defined as the sum of the flow **reaching T from S** minus the flow **reaching S from T**, i.e is the difference of the flow leaving S from the flow entering S.

The net flow is the flow itself
There are some ways to visualize that, let's make some observations first

- The flow of a network is the flow **reaching the sink**.
- Every flow entering a vertex must also leave it (flow conservation), except for the source and sink vertices

If we sum the flow entering set **T** and subtract the flow leaving **T**, that means that some flow **entered T but didn't leave it.** Since every flow entering a vertex must also leave it, except for the sink, it means that the remaining flow is exactly in the sink, which is the flow of the network.

Capacity of a cut

The ***capacity*** of the cut (S, T) is

$$c(S, T) = \sum_{u \in S} \sum_{v \in T} c(u, v) .$$

Minimum cut

A ***minimum cut*** of a network is a cut whose capacity is minimum over all cuts of the network.

The max-flow is bounded by the minimum capacity of all cuts

MAX-FLOW MIN-CUT Theorem

Theorem 26.6 (Max-flow min-cut theorem)
If f is a flow in a flow network $G = (V, E)$ with source s and sink t, then the following conditions are equivalent:

1. f is a maximum flow in G.
2. The residual network G_f contains no augmenting paths.
3. $|f| = c(S, T)$ for some cut (S, T) of G.

1 implies 2

An augmenting always path increases the flow in the original graph. If **f** is a max flow, it can't have an augmenting path in its residual network.

2 implies 3

Some definitions first:

G = (V, E) is the flow network, **s** is the source, **t** is the sink

Gf = the residual network of G, such that Gf has NO augmenting path
Let's make a cut of **V = S + T**

S = {all vertices reachable from s in Gf}
T = {V - S}

For every vertex **u in S** and vertex **v in T** we have the following cases:

- If **(u, v)** is present in **E**, f(u, v) = c(u, v) otherwise **v** would be reachable from **s**
- If **(v, u)** is present in **E**, f(v, u) = 0, otherwise **v** would be reachable from **s**

We know that the flow of that cut is the flow of **G**. Let's calculate the value of **|f|**.

$$
\begin{aligned}
f(S,T) &= \sum_{u\in S}\sum_{v\in T} f(u,v) - \sum_{v\in T}\sum_{u\in S} f(v,u) \\
&= \sum_{u\in S}\sum_{v\in T} c(u,v) - \sum_{v\in T}\sum_{u\in S} 0 \\
&= c(S,T)\,.
\end{aligned}
$$

By Lemma 26.4, therefore, $|f| = f(S,T) = c(S,T)$.

3 implies 1

The maximum flow of a network is bounded by the minimum capacity of a cut, of any cut. Since **Gf** with no augmenting paths implies that we have a cut whose value is the capacity of the cut, we can't make make a better flow. Of course, that cut is the minimum-capacity-cut.

Edmond karp algorithm

The shortest path from **u** to **v** in **Gf** increases monotonically.
Proof: after finding an augmenting path, we remove one edge of **Gf**. This means that the shortest path can't be shortened since now you have less options to make a shortest path. The edge we remove in the residual network is the **critical edge**

Critical edge: the minimum capacity on an edge in the residual network which is part of an augmenting path from **s** to **t** in the residual network. This edge will be removed from **Gf** since all of it's capacity was used.