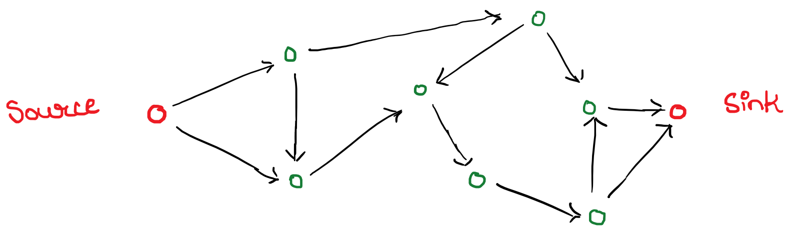
Flow Networks

Suppose that a Network consists of a Source, a Sink, and these two nodes are connected via switches. The bandwidth of each connection isn't the same. At the same time, each of the switch does cut-through switching without any Queue capacity.



This system can be modeled using a directed graph with two special nodes, the source and the sink.

Let the graph be $G(V, E)$, source be s and sink be t.

- By definition, no edge terminates at s and no edge begins at t.

Each edge has a capacity $c(e)$, which is the maximum possible capacity of that edge. We define $f(e)$, as the capacity of the edge in use currently.

$$\forall e \in E, \quad 0 \leq f(e) \leq c(e) \qquad \text{Capacity Constraint}$$

* Flow

Following the example, each bridge has a data stream flowing through it. As no data can be accumulated, Inward flow = Outward flow

(Similar to Kirchoff's law from electricity) flow constraint

- The outwards flow for a node is represented by $f^{\rightarrow}(v)$, and is given by:-

$$f^{\rightarrow}(v) = \sum_{\substack{u \in V \\ (u,v) \in E}} f((u,v))$$

The value of inwards flow is similarly defined, and is represented as $f^{\leftarrow}(v)$.

- Value of the flow $|f|$

The total amount of data "flowing" through the network. It is given by $|f|$, and is calculated as shown.

Lemma

$$|f| = \sum_{\substack{v \in V \\ (s,v) \in E}} f(s,v) = f^{\rightarrow}(s) = \sum_{\substack{v \in V \\ (v,t) \in E}} f(v,t) = f^{\leftarrow}(t)$$

Proof

From above, we can see that

$$|f| = f^{\rightarrow}(s) + 0$$

$$= f^{\rightarrow}(s) + \sum_{v \in V \setminus \{s,t\}} \left(f^{\rightarrow}(v) - f^{\leftarrow}(v)\right) \rightarrow 0 \text{ by flow constraint}$$

$$= \sum_{v \in V \setminus \{t\}} \left(f^{\rightarrow}(v) - f^{\leftarrow}(v)\right) \qquad f^{\leftarrow}(s) = 0 \text{ by definition}$$

$$= \sum_{v \in V \setminus \{t\}} f^{\rightarrow}(v) - \sum_{v \in V \setminus \{t\}} f^{\leftarrow}(v)$$

$$= \underset{1}{f^{\rightarrow}(V \setminus \{t\})} - \underset{2}{f^{\leftarrow}(V \setminus \{t\})}$$

New notation!

Notice that all edges appear in 1 as t has no flow leaving it. However, the edges "supplying" t would not be present in 2. These edges are left after subtraction.

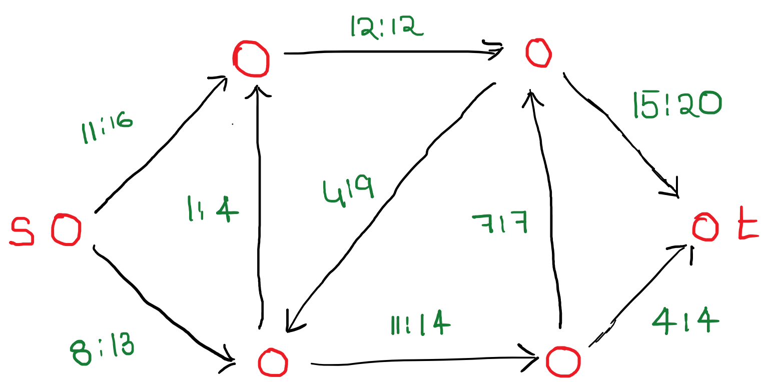
$$\rightarrow |f| = f^{\rightarrow}(V \setminus \{t\}) - f^{\leftarrow}(V \setminus \{t\}) = \underline{f^{\leftarrow}(t)}$$

QED

* Flow Network notation

The network is represented as a directed acyclic graph, with designated source and sink nodes. Each edge is labelled as $f(e) : c(e)$.

Capacity constraints and flow constraints have to be satisfied.



We would like to know what the maximum possible flow in this network is. The following concept is introduced for this.

* An (s,t)-cut :-

Let the flow network be given by $G(V,E)$ with s as the source and t is the sink. An (s,t)-cut is given by partioning V into two sets, S and T which are mutully exclusive and exhaustive.

$$s \in S \qquad S \cup T = V$$
$$t \in T \quad , \quad S \cap T = \phi$$

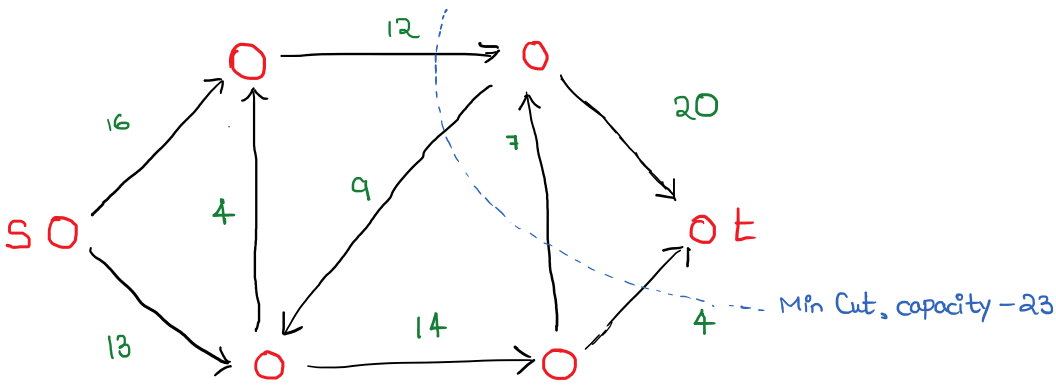
The capacity for an (s,t)-Cut (S,T) is given by:-

$$cap(S,T) = \sum_{\substack{u \in S,\, v \in T \\ (u,v) \in E}} c(u,v)$$

Notice how only one direction is considered, $S \to T$.

- Mincut problem

Given a flow network, we wish to find the cut of the graph which has the least possible capacity. For example,



There is an inherent relationship between max-flow and min-cut classes of problems. The below lemma hints at what this could be.

Lemma
Weak duality

Consider a flow network G. For any (s,t)-Cut (S,T) over this G, the value of $|f| \leq capacity(S,T)$. The equality is acheived when the flow through edges $S \to T$ is saturated and edges $T \to S$ are avoided.

Proof

$$|f| = f^{\rightarrow}(S) = f^{\rightarrow}(S) - f^{\leftarrow}(S)$$

$$\leq f^{\rightarrow}(S)$$

$$\leq \sum_{\substack{u \in S,\, v \in T \\ (u,v) \in E}} f(u,v) \leq \sum_{\substack{u \in S,\, v \in T \\ (u,v) \in E}} c(u,v)$$

as $f(u,v) \leq c(u,v)$ by definition

def of $cap(S,T)$

$$\leq cap(S,T)$$

$\leq \text{Cap}(S,T)$

QED

Also notice that inequalities are obtained by ignoring $f^{\leftarrow}(S)$ and taking $f(u,v) \leq c(u,v)$. Therefore, the equality holds if :-

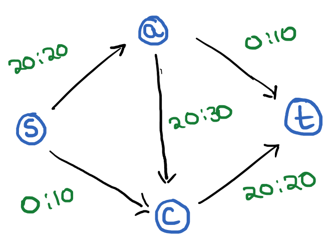
1) $f^{\leftarrow}(S) = 0$ —— T→S edges should be ignored
2) $f(u,v) = c(u,v)$ —— S→T edges should be saturated

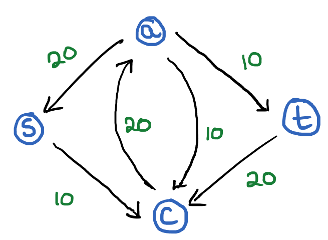
We define a few more concepts and ideas before tackling the max-flow problem. These will feel random at first, but the reasons will become clear as we move ahead with the solution.

Idea 1 – Residual Graph

For a given flow network $G(V,E)$ and $c: E \rightarrow \mathbb{Z}$ defined with a value of flow $|f|$ in the network, the residual Graph G_f is defined as follows :-

- Nodes in G_f are the same as nodes in G
- If an edge e in G has $f(e) < c(e)$, then a similar edge with capacity $c(e) - f(e)$ is present in G_f. This is called the Forward Edge.
- If an edge e in G has $f(e) > 0$, then a reverse edge with capacity $f(e)$ is present in G_f. This is called the Backwards Edge.





Atleast one and atmost two edges are added into the residual graph for every edge in G.

$$\# \text{ of edges in } G_f \leq 2 \times \# \text{ of edges in } G$$

Idea 1.2 **Augmenting paths**

For a residual graph G_f, let π be any path from s to t, with the mimum residual capacity along this path being given by $\theta(\pi, f)$.

Consider the following function :-

if $e = (u,v)$ then $e^{-1} = (v,u)$

```
Aug(π, f):
    b ← θ(π, f)
    for every edge e ∈ π, do
        if e is a forward edge, then
            increase f(e) in G by b
        else
            decrease f(e^-1) in G by b
        endif
    endfor
```

That is, we're modifying the original graph by using the information from the residual graph. The reasoning will be made clear shortly.

* ## Ford-Fulkerson's Algorithm

We state the algorithm below. The proof for the algorithm shall follow.

for Edge $e \in E$, set $f(e) = 0$ —— Initialisation

compute G_f

while path π from s to t exists in G_f

$\quad f \leftarrow Aug(\pi, f)$

end while

output f

Need to prove correctness

Although this algorithm seems very elegant, we have yet to prove the correctness and the time complexity. As a start, we first show that the algorithm terminates.