The Dinitz Algorithm

Reminder: Ford-Fulkerson Algorithm

Recall that the Ford-Fulkerson algorithm can be used to find a maximum flow in the network.

`Input` : Network $\mathcal{N}(G, s, t, c)$

`Output`: Flow function $f$

1. for $e \in E$ do
2. \hspace{1cm} $f(e) \leftarrow 0$
3. end
4. while there exists an augmenting path from $s$ to $t$ of value $\Delta$ do
5. \hspace{1cm} push $\Delta$ units of flow from $s$ to $t$
6. end

Algorithm 1: Ford-Fulkerson Algorithm

Clearly, if Ford-Fulkerson algorithm stops then there is no augmenting path from $s$ to $t$. Let $S \subseteq V$ be a collection of vertices $v$ such that there is an augmenting path from $s$ to $v$. Then $s \in S$, $t \notin S$. We have:

$$F \equiv (1) \sum_{e \in (S,S)} f(e) - \sum_{e \in (S,S)} f(e)$$

$$\equiv (2) \sum_{e \in (S,S)} c(e) - \sum_{e \in (S,S)} 0$$

$$\equiv (3) c(S)$$

Here, transition (1) gives the total flow in the cut $(S : \bar{S})$. Transition (2) holds because any edge in the cut $(S : \bar{S})$ must be saturated (otherwise the edge is useful, and both its endpoints are in $S$). Similarly, any edge in the cut $(\bar{S} : S)$ must have zero flow. Transition (3) is due to definition of the capacity of the cut. We obtain the following statement.

**Theorem 1** Every network has a maximum flow, which is equal to the minimum capacity of any cut between $s$ and $t$ ($s \in S$, $t \notin S$).

---

$^1$Additional reading: Sections 5.2 and 5.3 in the book of S. Even “Graph Algorithms”.
Example

Consider the following network, where $M$ is a large positive number.

![Network Diagram]

In Ford-Fulkerson the choice of an augmenting path is arbitrary. Assume that in this network, the following two augmenting paths are chosen interchangeably: $s \to a \to b \to t$ and $s \to b \to a \to t$. In that case, each augmenting path improves the total flow by one unit. Therefore, $2M$ iterations are required to find the maximum flow. It is not efficient.

Some (more efficient) algorithms for finding maximum flows:

- **Edmonds-Karp (1972).** Complexity: $O(|V||E|^2)$.
- **Dinitz (1970).** Complexity: $O(|V|^2|E|)$.
- **Goldberg-Tarjan (1986).** Complexity: $O(|V|^2\sqrt{|E|})$ or $O(|V||E| \log (|V|^2/|E|))$.
- **Orlin (2013).** Complexity: $O(|V||E|)$ under some weak assumption on the graph.

Edmonds-Karp Algorithm

Edmonds-Karp algorithm (1972) is a modification of Ford-Fulkerson algorithm. It is also used to find a maximum flow in the network.

```
Input : Network $N(G, s, t, c)$
Output: Flow function $f$
1 for $e \in E$ do
2 \hspace{1em} $f(e) \leftarrow 0$;
3 end
4 while there exists an augmenting path from $s$ to $t$ do
5 \hspace{1em} push flow through the shortest augmenting path from $s$ to $t$;
6 end
```

Algorithm 2: Edmonds-Karp Algorithm

The difference of this algorithm with Ford-Fulkerson algorithm is that each time the shortest augmenting path is used. In the original Ford-Fulkerson algorithm it is not specified what
augmenting path should be taken. Since Edmonds-Karp is a special case of Ford-Fulkerson (the choice of augmenting paths in Edmonds-Karp is a legal choice also in Ford-Fulkerson, and the stopping condition is the same), the correctness of Edmonds-Karp follows from the correctness of Ford-Fulkerson.

**Time complexity.** By using the BFS, finding the shortest (augmenting) path is done in $O(|E|)$. Each edge can be a bottleneck (of an augmenting path) in $N$ only $|V|/2$ times (the proof of this fact is omitted). Therefore, the total time complexity of Edmonds-Karp is $O(|V| \cdot |E|^2)$.

**High Level Overview of Dinitz Algorithm**

The algorithm starts with some legitimate flow (for example, zero flow in all edges) and gradually improves it. When no improvement is possible, the algorithm stops.

**Definition 1** Given the network $N(G(V, E), s, t, c)$ and the flow function $f$, we define the residual network $N'(G'(V, E'), s, t, c')$ as follows:

- For every $u \xrightarrow{e} v$ in $E$ such that $f(e) < c(e)$ we also have $u \xleftarrow{e} v$ in $E'$. We define its residual capacity $c'(e)$ to be $c(e) - f(e)$.

- For every $u \xrightarrow{e} v$ in $E$ such that $f(e) > 0$ we have $u \xleftarrow{e'} v$ in $E$. We define $c'(e') = f(e)$.

If $0 < f(e) < c(e)$ then every edge $e \in E$ gives a rise to two antiparallel edges in $E'$. Otherwise, if $f(e) = 0$ or $f(e) = c(e)$, then there is only one edge in $E'$ corresponding to $e$. Therefore, $|E| \leq |E'| \leq 2|E|$.

Dinitz Algorithm proceeds in phases. In each phase, the current $f$ is used to produce the corresponding residue network $N'$. A layered network $N''$ is then produced by the BFS Algorithm applied to $N'$ starting from $s$. If $t$ is not reached in this process – the algorithm stops. If $t$ is reached then a maximal flow $f''$ is found in $N''$. This flow is added to $N$, and the new phase is launched.
Example

Consider the following network:

We illustrate the run of Dinitz algorithm on this network.
Phase 1

Network $N$:

Network $N'$:

Network $N''$ after some maximal flow is found (the maximal flow is not unique):
Phase 2

Network $\mathcal{N}$:

Network $\mathcal{N}'$:

Network $\mathcal{N}''$ after some maximal flow is found:
Phase 3

Network $\mathcal{N}$:

Network $\mathcal{N}'$:

Network $\mathcal{N}''$:

The vertex $t$ cannot be reached, and therefore the algorithm stops. The current flow in the network $\mathcal{N}$ is maximum.