Accountability

- People that do not know maximum flows: essentially everybody.
- Average salary on earth \( \$5,000 \)
- People that know maximum flow – most of them work in programming related jobs and make at least \( \$10,000 \) a year.
- Salary of people that learned maximum flows: \( \geq \$10,000 \)
- Salary of people that did not learn maximum flows: \( < \$5,000 \).
- Salary of people that know Latin: \( 0 \) (unemployed).

Conclusion

Thus, by just learning maximum flows (and not knowing Latin) you can double your future salary!

Ford Fulkerson

\[
\text{algFordFulkerson}(G, s, t) \\
\text{Initialize flow } f \text{ to zero} \\
\text{while } \exists \text{ path } \pi \text{ from } s \text{ to } t \text{ in } G_f \text{ do} \\
\quad c_f(\pi) \leftarrow \min \{ c_f(u, v) \mid (u \rightarrow v) \in \pi \} \\
\quad \text{for } \forall (u \rightarrow v) \in \pi \text{ do} \\
\quad \quad f(u, v) \leftarrow f(u, v) + c_f(\pi) \\
\quad \quad f(v, u) \leftarrow f(v, u) - c_f(\pi)
\]

Lemma

If the capacities on the edges of \( G \) are integers, then \( \text{algFordFulkerson} \) runs in \( O(m|f^*|) \) time, where \( |f^*| \) is the amount of flow in the maximum flow and \( m = |E(G)| \).
Proof of Lemma...

Proof.
Observe that the \texttt{algFordFulkerson} method performs only subtraction, addition and \texttt{min} operations. Thus, if it finds an augmenting path \( \pi \), then \( c_f(\pi) \) must be a \textit{positive} integer number. Namely, \( c_f(\pi) \geq 1 \). Thus, \( |f^*| \) must be an integer number (by induction), and each iteration of the algorithm improves the flow by at least 1. It follows that after \( |f^*| \) iterations the algorithm stops. Each iteration takes \( O(m + n) = O(m) \) time, as can be easily verified.

Integrality theorem

Observation (Integrality theorem)

If the capacity function \( c \) takes on only integral values, then the maximum flow \( f \) produced by the \texttt{algFordFulkerson} method has the property that \( |f| \) is integer-valued. Moreover, for all vertices \( u \) and \( v \), the value of \( f(u, v) \) is also an integer.

Edmonds-Karp algorithm

\textbf{Edmonds-Karp}: modify \texttt{algFordFulkerson} so it always returns the shortest augmenting path in \( G_f \).

Definition

For a flow \( f \), let \( \delta_f(v) \) be the length of the shortest path from the source \( s \) to \( v \) in the residual graph \( G_f \). Each edge is considered to be of length 1.

Assume the following key lemma:

Lemma

\( \forall v \in V \setminus \{s, t\} \) the function \( \delta_f(v) \) increases.

The disappearing/reappearing lemma

\textbf{Lemma}

During execution \texttt{Edmonds-Karp}, edge \((u \rightarrow v)\) might disappear/reappear from \( G_f \) at most \( n/2 \) times, \( n = |V(G)| \).

Proof.

- iteration when edge \((u \rightarrow v)\) disappears.
- \((u \rightarrow v)\) appeared in augmenting path \( \pi \).
- Fully utilized: \( c_f(\pi) = c_f(uv) \). \( f \) flow in beginning of iter.
- till \((u \rightarrow v)\) “magically” reappears.
- ... augmenting path \( \sigma \) that contained the edge \((v \rightarrow u)\).
- \( g \): flow used to compute \( \sigma \).
- We have: \( \delta_g(u) = \delta_g(v) + 1 \geq \delta_f(v) + 1 = \delta_f(u) + 2 \)
- distance of \( s \) to \( u \) had increased by 2. QED.
**Comments...**

- $\delta_f(u)$ might become infinity.
- $u$ is no longer reachable from $s$.
- By monotonicity, the edge $(u \rightarrow v)$ would never appear again.

**Observation**

For every iteration/augmenting path of Edmonds-Karp algorithm, at least one edge disappears from the residual graph $G_f$.

**Edmonds-Karp # of iterations**

**Lemma**

Edmonds-Karp handles $O(nm)$ augmenting paths before it stops. Its running time is $O(nm^2)$, where $n = |V(G)|$ and $m = |E(G)|$.

**Proof.**

- Every edge might disappear at most $n/2$ times.
- At most $nm/2$ edge disappearances during execution Edmonds-Karp.
- In each iteration, by path augmentation, at least one edge disappears.
- Edmonds-Karp algorithm perform at most $O(mn)$ iterations.
- Computing augmenting path takes $O(m)$ time.
- Overall running time is $O(nm^2)$.

**Proof continued...**

- $\pi = s \rightarrow \cdots \rightarrow u \rightarrow v$: shortest path in $G_g$ from $s$ to $v$.
- $(u \rightarrow v) \in E(G_g)$, and thus $\delta_g(u) = \delta_g(v) - 1$.
- By choice of $v$: $\delta_g(u) \geq \delta_f(u)$.
  - (i) If $(u \rightarrow v) \in E(G_f)$ then
    \[ \delta_f(v) \leq \delta_f(u) + 1 \leq \delta_g(u) + 1 = \delta_g(v) - 1 + 1 = \delta_g(v). \]
    This contradicts our assumptions that $\delta_f(v) > \delta_g(v)$.
Proof continued II

(ii) \( f(u \rightarrow v) \notin E(G_f) \):
1. \( \pi \) used in computing \( g \) from \( f \) contains \( (v \rightarrow u) \).
2. \( (u \rightarrow v) \) reappeared in the residual graph \( G_g \) (while not being present in \( G_f \)).
3. \( \implies \pi \) pushed a flow in the other direction on the edge \((u \rightarrow v)\). Namely, \((v \rightarrow u) \in \pi\).
4. Algorithm always augment along the shortest path. By assumption \( \delta_g(v) < \delta_f(v) \), and definition of \( u \):
   \[ \delta_f(u) = \delta_f(v) + 1 > \delta_g(v) = \delta_g(u) + 1. \]
5. \( \implies \delta_f(u) > \delta_g(u) \)
   \( \implies \) monotonicity property fails for \( u \).
   But: \( \delta_g(u) < \delta_g(v) \). A contradiction.

---

Bipartite matching

Definition

\( G = (V, E) \): undirected graph.
\( M \subseteq E \): matching if all vertices \( v \in V \), at most one edge of \( M \) is incident on \( v \).
\( M \) is maximum matching if for any matching \( M' \): \( |M| \geq |M'| \).
\( M \) is perfect if it involves all vertices.

Computing bipartite matching

Theorem

Compute maximum bipartite matching in \( O(nm) \) time.

Proof.
1. \( G \): bipartite graph \( G \). \( (n \) vertices and \( m \) edges)
2. Create new graph \( H \) with source on left and sink right.
3. Direct all edges from left to right. Set all capacities to one.
4. By Integrality theorem, flow in \( H \) is \( 0/1 \) on edges.
5. A flow of value \( k \) in \( H \) \( \implies \) a collection of \( k \) vertex disjoint \( s - t \) paths \( \implies \) matching in \( G \) of size \( k \).
6. \( M \): matching of \( k \) edge in \( G \), \( \implies \) flow of value \( k \) in \( H \).
7. Running time of the algorithm is \( O(nm) \). Max flow is \( n \), and as such, at most \( n \) augmenting paths.
Extension: Multiple Sources and Sinks

Question
Given a flow network with several sources and sinks, how can we compute maximum flow on such a network?

Solution
The idea is to create a super source, that send all its flow to the old sources and similarly create a super sink that receives all the flow. Clearly, computing flow in both networks in equivalent.