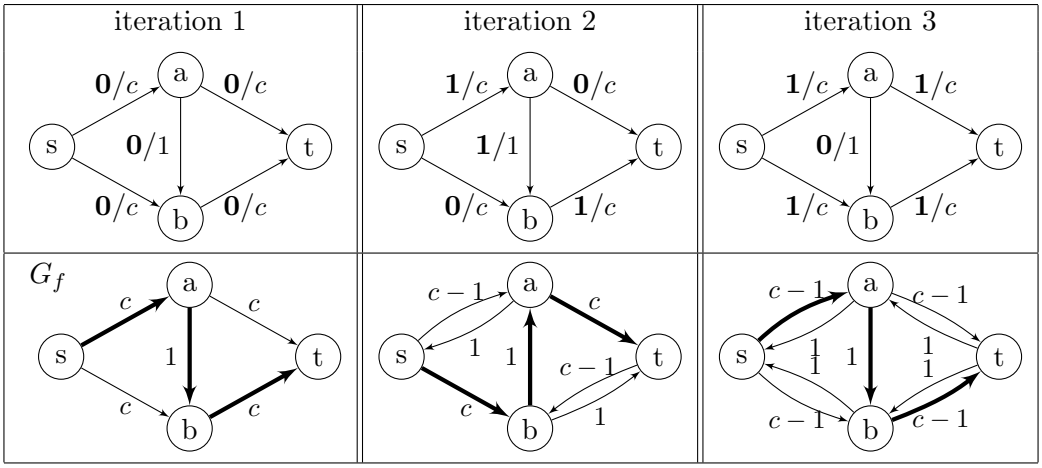
The previous lecture covered the MAX FLOW problem, and described the Ford-Fulkerson augmenting-paths algorithm. We showed that *if* the algorithm terminates, then it returns a max flow. But we did not bound the running time. We observed that each iteration can be implemented in linear time, but we did not bound the number of iterations. In fact, the number of iterations can be exponential. Fix some large integer c and consider the following example:



In each iteration, the algorithm chooses the augmenting path through edge (a, b) or (b, a). Each time, that path has residual capacity 1, so the flow value increases by 1 unit. The number of iterations will be $2c$, which is the value of the maximum flow. This example is bad, but it's not the worst. In fact, there are flow networks (with irrational capacities) on which the algorithm *never terminates*.[1]

To improve the worst-case running time, Edmonds and Karp (and separately Dinitz) suggested modifying the algorithm to choose each augmenting path p in a particular way — to be a *shortest s-t path* (with a minimum number of edges) in G_f:

Edmonds-Karp$(G = (V, E), \mathsf{cap}, s, t)$
1. initialize $f[u, w] = 0$ for all $(u, w) \in E$
2. for any path p in G_f, define $\delta(p) = \min\{\mathsf{cap}_f(u, w) : (u, w) \in p\}$ *(p's* residual capacity*)*
3. while there is an s-t path in G_f:
4.1. let p be a shortest s-t path (with a minimum number of edges, $|p|$) in G_f *(use BFS to find p)*
4.2. for each edge $(u, w) \in p$: *(augment flow along augmenting path p)*
4.3.1. if $(u, w) \in E$: increase $f[u, w]$ by $\delta(p)$
4.3.2. else: decrease $f[w, u]$ by $\delta(p)$ *(cancel $\delta(p)$ units of flow on (w, u))*
5. return f

Each iteration can be implemented in linear time, by using breadth-first search from s to find the augmenting path p in G_f. Recall that the residual graph $G_f = (V, E_f)$ for f has edges and residual edge capacities

$$E_f = \{(u, w) \in E : f(u, w) < \mathsf{cap}(u, w)\} \cup \{(u, w) : (w, u) \in E,\ f(w, u) > 0\}, \text{ where}$$

$$\mathsf{cap}_f(u, w) = \begin{cases} \mathsf{cap}(u, w) - f(u, w) & \text{if } (u, w) \in E \text{ and } f(u, w) < \mathsf{cap}(u, w) \\ f(w, u) & \text{if } (w, u) \in E \text{ and } f(w, u) > 0. \end{cases}$$

[1] https://en.wikipedia.org/wiki/Ford-Fulkerson_algorithm#Non-terminating_example

The algorithm is correct because it is a special case of the Ford-Fulkerson algorithm. Next we bound the run time. The bound uses the following key property. For any vertex v and flow f, define $d_i(v)$ to be the shortest-path distance from s to v in the residual graph G_f after iteration i of the algorithm. (By distance, we mean the minimum number of edges on any s-v path in G_f.)

Lemma 1. *For any vertex $v \in V$, during the execution of the algorithm, the distance from s to v in the residual graph never decreases. That is, for each iteration i, $d_i(v) \leq d_{i+1}(v)$.*

Proof. As the algorithm proceeds, edges enter and leave the residual graph. Removing an edge cannot decrease the distance from s to v, because it does not create any new paths. Adding an edge creates new paths, but we show (for this algorithm) that these paths cannot be shortest paths.

As the algorithm proceeds, imagine partitioning the vertices into levels according to their current distance from s in G_f: the source s is in level 0, out-neighbors of s in G_f are in level 1, and so on, so level ℓ contains the vertices at distance ℓ from s in G_f. So, a path from s is a shortest path in G_f if and only if each edge (u, w) on the path goes from one level to the next.

Consider iteration i. Augmenting the flow along the augmenting path p adds some edges to the residual graph and removes others. Let A and R be the sets of edges added and removed. The path p is a shortest path in G_f, so each edge (u, w) in p goes from one level ($d_i(u)$) to the next ($d_i(w) = d_i(u) + 1$). Each edge in A is the reverse of some edge in p, so each $(w, u) \in A$ goes from some level $d_i(w)$ to the preceding level $d_i(u) = d_i(w) - 1$.

Now imagine obtaining the next residual graph (at the start of iteration $i + 1$) in two steps: first add the edges in A, then remove the edges in R. The second step can't create any new shortest path. To finish, we argue that the first step, adding the edges in A, neither creates nor destroys any shortest path. First consider adding just the first edge (w, u) from A. As discussed above, w is in some level $d_i(w)$, and u is in the previous level $d_i(u) = d_i(w) - 1$. Any path p' that is created by adding (w, u) must use (w, u). But such a path p' cannot be a shortest path, because it goes through (w, u), so the prefix of p' to u has at least $d_i(w) + 1 = d_i(u) + 2$ edges, whereas there is a path from s to u with $d_i(u)$ edges. Applying the same argument inductively to each edge in A, shows that adding the edges in A does not create or destroy any shortest path in the residual graph. □

Theorem 1 (Edmonds-Karp, Dinitz)**.** *In any execution of the algorithm, there are $O(|V| \cdot |E|)$ iterations.*

Proof. In each iteration of the algorithm, when the algorithm augments flow along the augmenting path p, at least one edge (u, w) on p becomes saturated and so leaves the residual graph in that iteration. We argue that each edge leaves the residual graph $O(|V|)$ times. This implies that there are at $O(|V| \cdot |E|)$ iterations.

Consider any possible edge (u, w) in the residual graph. Consider an iteration i in which (u, w) leaves the residual graph. Let i' be the iteration when it first returns (if any). Following the reasoning in the proof of the lemma, in iteration i, because (u, w) is in the augmenting path, we have $d_i(w) = d_i(u)+1$. Then, because (u, w) enters the residual graph in iteration i', it must be that the reverse edge (w, u) is in the augmenting path in iteration i'. That means $d_{i'}(u) = d_{i'}(w) + 1$.

By the lemma, $d_{i'}(w) \geq d_i(w)$. Combining these inequalities gives $d_{i'}(u) = d_{i'}(w) + 1 \geq d_i(w) + 1 = d_i(u) + 2$.

That is, each time (u, w) leaves then re-enters the residual graph, the distance from s to u in the residual graph increases by at least 2. Since the distance to u is at least 0 initially, and at most $|V| - 1$ at the end, it follows that (u, w) leaves the residual graph $O(|V|)$ times. □

External resources on the Edmonds-Karp algorithm

- CLRS Chapter 26.
- Erickson's Lectures 23.6.2:
  http://jeffe.cs.illinois.edu/teaching/algorithms/notes/23-maxflow.pdf