Faster Flow Algorithms[1]

The FORDFULKERSON algorithm is a good algorithm when U is small.But when the time bound is not satisfactory when U is large. In fact, the algorithm is not a *polynomial time* algorithm. Let's visit this again (we had this nag in the case of SUBSET SUM). What is the number of bits required to describe the input to FORDFULKERSON. Note that for every edge, we need to write down its $u(e)$, and since $u(e)$ is an integer between $\{1, \ldots, U\}$, this requires $O(\log U)$. Thus, the number of bits required to describe the input is $O((n+m)\log U)$. And, U is not upper bounded by any polynomial of $\log U$. So, as stated, FORDFULKERSON is *not* a polynomial time algorithm.

Now let me make another observation: FORDFULKERSON is not a completely described algorithm. This is because in Line 4, we don't quite specify *how* the path from s to t is chosen. We could use BFS, we could use DFS, or perhaps some "oracle" is handing down the path. The example below in Figure 1 shows that if we are not careful with how we choose paths, FORDFULKERSON can take $\Omega(U)$ time.

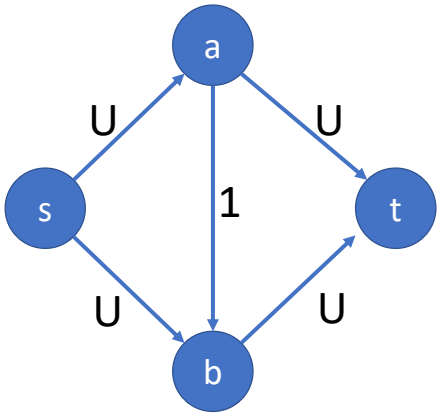

Figure 1: Suppose the first path we augment on is (s, a, b, t); we can augment only 1 unit of flow. Then in the residual graph the edge (b, a) has $u_f = 1$. Now suppose we augment on (s, b, a, t); we can augment only 1 unit of flow. Then to get the final flow of $2U$ units, we need to repeat this $\Theta(U)$ times.

Of course the paths we choose in the above algorithm are silly. There are two glaring ways to improve upon the above algorithm. The first one: instead of choosing any path from s to t in G_f, choose the path where the $\min_{e \in p} u_f(e)$ is maximized. That is, find the path p with the maximum *capacity*; recall, this was a problem in the problem set which could be solved in DIJKSTRA time (which, if you recall, was $O(m + n \log n)$ time). The second one is a little more subtler and not clear at first glance why it is a good algorithm. It says, run BFS to find the *shortest* length path from s to t in G_f. As we see, in some regimes this algorithm is even better than picking the maximum capacity one greedily.

[1] *Lecture notes by Deeparnab Chakrabarty. Last modified : 19th Mar, 2022*
These have not gone through scrutiny and may contain errors. If you find any, or have any other comments, please email me at deeparnab@dartmouth.edu. Highly appreciated!

1 Augmenting on Maximum Capacity Path

For completeness, we state this algorithm in pseudocode.

```
1:  procedure MaxCapFF(G, s, t, u):
2:      Initialize f ≡ 0 and u_f ≡ u and G_f ≡ G.
3:      while true do:
4:          Find the maximum capacity path p from s to t in G_f.
5:          ▷ Recall, the capacity of path p is min_{e∈p} u_f(e).
6:          ▷ This can be found a la Dijkstra in O(m + n log n) time.
7:          Let δ* := min_{e∈p} u_f(e).
8:          if δ* = 0 then:
9:              break
10:         else:
11:             Augment(G_f, s, t, p).
12:     return (f, G_f).
```

Theorem 1. If all the $u(e)$'s positive reals in the range $[1, U]$, then MaxCapFF finds the maximum flow in $O(m \log(nU) \cdot (m + n \log n))$ time.

Remark: *The above algorithm is better than* FordFulkerson *whenever* $U/\log(nU) \gg m/n$. *More importantly, it is a polynomial time algorithm.*

Proof. The MaxCapFF algorithm falls in the FordFulkerson suite and is therefore going to return a maximum flow algorithm. What is interesting is the running time analysis. We show that the **while** loop runs for $O(m \log(nU))$ iterations. Since in each iteration we find the maximum capacity path and run Augment, each iteration takes $O(m + n \log n)$ time.

To argue about the number of iterations, let us introduce some notation. Let T denote the number of iterations of the **while** loop which we index using small τ. We wish to show $T \leq m \log(nU)$. Let δ_τ denote the value of δ^* in the tth **while** loop. Let F_τ be the value of the flow just before the τth while loop. So, $F_1 = 0$ and $F_{T+1} = F^*$.

Lemma 1. For any $1 \leq \tau \leq T$, $\quad \delta_\tau \geq \frac{F^* - F_\tau}{m}$

We prove this lemma shortly. For now, let us assume this and complete the proof of the theorem. Note that $F_{\tau+1} = F_\tau + \delta_\tau$. Therefore, assuming the lemma we get

$$(F_{\tau+1} - F_\tau) \geq \frac{(F^* - F_\tau)}{m}$$

Here's a trick now which is commonly used with inequalities as above. Define $G_\tau := F^* - F_\tau$, the G being for gap. Note that the LHS above is precisely $G_\tau - G_{\tau+1}$. Therefore, we get

$$(G_\tau - G_{\tau+1}) \geq \frac{G_\tau}{m} \quad \Rightarrow \quad G_{\tau+1} \leq G_\tau \cdot \left(1 - \frac{1}{m}\right)$$

Thus, the gap between the current flow and the maximum flow value drops multiplicatively. Applying the above inequality inductively we get

$$G_T \leq G_0 \left(1 - \frac{1}{m}\right)^T \leq nU \cdot e^{-T/m}$$

Here we have used $G_0 = F^* \leq nU$ and the ubiquitious math inequality $(1 + z) \leq e^z$ for all z (we used it for $z = -1/m$).

Since we run the Tth while loop (that is, we haven't reached max-flow yet), we get that $G_T = F^* - F_T \geq 1$. Putting together everything, we get

$$1 \leq nU \cdot e^{-T/m} \quad \Rightarrow \quad T \leq m \log(nU)$$

This ends the proof of the theorem and all that remains is to prove Lemma 1. We do so next.

Proof of Lemma 1. It is not hard to show that the maximum flow in the residual graph before the τth iteration is precisely $F^* - F_\tau$ (can you do it?). Let G_{f_τ} be this residual graph. Since δ_τ is the capacity of the maximum capacity path, it means that if we delete all edges of capacity $\leq \delta_\tau$, then there is no path from s to t. In particular, there is an s, t cut in G_{f_τ} which consists of edges of capacity $\leq \delta_\tau$. The value of this cut can be at most $m\delta_\tau$ (worst case the cut has m different edges). The max-flow-min-cut theorem implies $F^* - F_\tau \leq m\delta_\tau$, which is precisely the lemma. □

□

2 Augmenting on a shortest path: Edmonds-Karp Algorithm

The second idea for augmentation asks to just run BFS from s to t. That is, whenever we are searching for a path from s to t in G_f, take the shortest length path. More precisely, ignore all the edges with $u_f(e) = 0$ in G_f, and then find the shortest *length* path from s to t in G_f. This gives an algorithm whose running time doesn't depend on U at all[2]. For completeness, we state this algorithm in pseudocode.

```
1:  procedure EdmondsKarp(G, s, t, u):
2:      Initialize f ≡ 0 and u_f ≡ u and G_f ≡ G.
3:      while true do:
4:          Find shortest length s,t path p in G_f after removing all u_f(e) = 0 edges.
5:          ▷ This can be found using BFS in O(m + n) time.
6:          Let δ* := min_{e∈p} u_f(e).
7:          if δ* = 0 then:
8:              break
9:          else:
10:             Augment(G_f, s, t, p).
11:     return (f, G_f).
```

[2]Of course it does; what we mean is that it doesn't depend if U fits in the word and adding, multiplying, comparing, etc can be done in $\Theta(1)$ time. A more precise statement would be that the number of iterations doesn't depend on U at all.

Theorem 2. EDMONDSKARP finds the maximum flow in $O(nm)$ iterations, and thus takes $O(nm^2)$ time in all.

Proof. The proof technique of this theorem is quite different from that of Theorem 1; while that proof argued that the total increase in flow in each step (the quantity δ_t was "large"), this proof argues about the "structure" of the residual graph as time marches on. The main idea is this: we will show that by $O(nm)$ iterations of EDMONDSKARP, there cannot be ***any*** path from s to t in the residual graph. And therefore our algorithm will have terminated with the maximum flow.

Once again we use small τ to index the while loops and use f_τ to denote the flow just before while loop τ is run. Let $G_\tau := G_{f_\tau}$ be the residual graph with respect to f_τ with all 0 residual capacity edges removed.

Let $d_\tau(u, v)$ denote the shortest length path from u to v in G_τ. Note that if $d_\tau(s, t) \geq n$ then it must be that s and t are disconnected in G_τ; this is because paths can't have length more than $(n - 1)$. The proof strategy is two pronged: in Lemma 2 we show that the distances can't *decrease* over time, and in Lemma 3 we show that in every $O(m)$ iterations the distance must *strictly* increase. This would prove that in $O(nm)$ iterations, we would have $d_\tau(s, t) \geq n$ implying termination.

In fact, Lemma 2 shows something stronger.

Lemma 2. For any vertex v and any τ, $d_\tau(s, v) \leq d_{\tau+1}(s, v)$ and $d_\tau(v, t) \leq d_{\tau+1}(v, t)$

Proof. Let us consider how the graph changes from G_τ to $G_{\tau+1}$. In G_τ we find a shortest length path from s to t. Let this path be $(s = x_0, x_1, \ldots, x_k = t)$. We send $\delta := \min_{e \in p} u_\tau(e)$ flow through it; say the edge $e = (x_i, x_{i+1})$ has residual capacity δ. In $G_{\tau+1}$, we see that this edge has residual capacity 0 and is therefore absent from $G_{\tau+1}$. Furthermore, all the "flipped" edges of the form (x_{j+1}, x_j) are present in $G_{\tau+1}$ (and they could've been present in G_τ as well).

Now suppose $d_{\tau+1}(s, v) < d_\tau(s, v)$ for some v. Let q be the shortest path from s to v in $G_{\tau+1}$; observe that q must contain an edge which was absent in G_τ, and therefore is a flipped edge of the form (x_{j+1}, x_j). In fact, let's choose v such that the number of these flipped edges in a shortest path in $G_{\tau+1}$ is as *small as possible* and let (x_{j+1}, x_j) be the *last* flipped edge in q.

Let's divide this q into three pieces: q_1 is the part from s to x_{j+1}, then the edge (x_{j+1}, x_j), and let q_2 be the path from x_j to v. Since (x_{j+1}, x_j) is the last flipped edge in q, we get that q_2 is a path present in G_τ as well. Now, $|q| = |q_1| + 1 + |q_2|$. Since q_1 has fewer flipped edges than q, we get by the choice of v that $|q_1| \geq d_\tau(s, x_{j+1}) = j + 1$. This gives $|q| \geq j + 2 + |q_2|$. On the other hand, there is a walk in G_τ from s to x_j using the shortest path, and then following q_2 of length $j + |q_2| < |q|$. This is a contradiction to our supposition.

The other proof about $d_\tau(v, t)$'s is similar and is left as an exercise. □

Although the previous lemma shows that as τ increases the distances from s to t can't decrease, this distance can remain the same for a while. The next lemma shows that this can't happen for long.

Lemma 3. The distance $d_\tau(s, t)$ strictly increases in any contiguous $m + 1$ iterations.

Proof. Let's chop up the while loops into *epochs*; epoch k for $1 \leq k \leq n - 1$ contains all the τ's such that $\mathsf{dist}_\tau[t] = k$. We need to show that the length of any epoch is $\leq m$.

To do so, we need to show that for two τ and $\tau + 1$ in the same epoch k, some quantity (potential) must be diminishing and thereby prove that epochs can't last forever. This is a common technique in the analysis

of many algorithms, and choosing this potential often requires a deeper understanding of the algorithm's innards.

What could this quantity be? Once again, observe that when we go from G_τ to $G_{\tau+1}$, one of the edges (x, y) on a shortest path from s to t "flips". In particular, the edge (x, y) is absent from $G_{\tau+1}$ (since all 0 residual capacity edges are removed), and the edge (y, x) is present. Now comes the main observation : the edge (y, x) *cannot* be in any shortest path from s to t *for any* $\tau' > \tau$ *in the same epoch*. Why? Well, note that $d_{\tau'}(s, t) = d_\tau(s, t)$ since we are in the same epoch. On the other hand, for any path path containing edge (y, x), its length is at least $d_{\tau'}(s, y) + 1 + d_{\tau'}(x, t)$ which by the previous lemma is at least $d_\tau(s, y) + 1 + d_\tau(x, t)$. But, this quantity is $> k$; to see this note $k = d_\tau(s, x) + 1 + d_\tau(y, t) = d_\tau(s, y) + d_\tau(x, t) - 1$. Therefore, no such path can be a shortest path (since we are in the same epoch).

Remark: *We should assert here that across epochs the edge (y, x) can appear in a shortest path. In fact, you should think of examples where this does occur.*

Why was the above reasoning useful? Well, since the edge (y, x) doesn't appear in any shortest path, the flipped edge (x, y) doesn't come back across iterations in the same epoch. Therefore one edge keeps "vanishing" in each epoch iteration and since there are at most m edges, we get that any epoch lasts at most m iterations.

To formalize, define E_τ to be the set of edges which appear in *some* shortest path from s to t in G_τ. The edge (x, y) was present in E_τ but is absent from $E_{\tau+1}$. On the other hand, every edge in $E_{\tau+1}$ is present in E_τ; this is because all shortest paths from s to t in $G_{\tau+1}$ are also shortest paths in G_τ. Thus, as we go from τ to $\tau + 1$ the size of the set $|E_\tau|$ drops by at least 1. Since this number can be at most m to begin with, any epoch lasts for $\leq m$ iterations. This completes the proof. □

In sum, since $d_0(s, t) \geq 0$ from Lemma 2 and Lemma 3 we get that in $\leq nm$ iterations we would get s and t disconnected leading to the termination of EDMONDSKARP. Since each iteration is a run of BFS, we get the theorem. □