CS345 Assignment-3

Ayush Shakya 180178
Soham Ghosal 180771

November 13, 2021

1 Aim

In this problem, we have to come up with an algorithm for a modified version of Ford-Fulkerson Algorithm with integer capacities. Our aim is to come up with a polynomial time algorithm since the already existing FF is runs in $\Theta(mc)$.

Using the hints and template provided in the assignment, we are able to devise an algorithm for Poly-FF which acts as an upper bound for time complexity and maximum number of augmenting paths in the graph G. We would then state and prove a couple of lemmas to finally come up with the correlation among the two algorithms.

2 Pseudocode

Algorithm 1 Ford-Fulkerson Algorithm in polynomial time for integer capacities

```
1:  procedure POLY-FF(G, s, t)
2:      f ← 0
3:      k ← maximum capacity of any edge in G
4:      while k ≥ 1 do
5:          while there exists a path of capacity ≥ k in G_f do
6:              Let P be any path in G_f with capacity at least k
7:              c ← bottleneck(P)                ▷ This function returns the min-capacity in path P
8:              for all edge(x, y) ∈ P do
9:                  if (x,y) is a forward edge then
10:                     f(x, y) ← f(x, y) + c
11:                 end if
12:                 if (x,y) is a backward edge then
13:                     f(y, x) ← f(y, x) − c
14:                 end if
15:             end for
16:         end while
17:         k ← k/2
18:     end while
19: end procedure
```

3 Time Complexity Analysis

To do an overall time complexity analysis of this algorithm, we will approach in a step by step fashion using the hints provided in the assignment. The methodical approach is summarized in the following subsections.

3.1 Lemma 1

If f is the current value of the (s,t)-flow in G, then, $f \geq f_{max} - 2mk_0$***, where*** f_{max} ***is the maximum (s,t)-flow in the original graph G.***

Proof : Let us assume the current flow to be f and value of k to be k_0 at the beginning of an iteration of the outermost while loop. We note that there is no change in the flow after the termination of the inner while loop in the previous iteration. Additionally, since k decreases to half of its previous value in every iteration, the previous value of k must have been $2k_0$. Now we provide another claim, using which we will formalise the proof for lemma 1.

3.1.1 Claim

If k_0 ***is the value of k with which we had entered the outer while loop, then for some flow f at the end of the inner while loop, the maximum flow in the network G is upper bounded by*** $f + mk_0$

Proof: Let f_{max} be the value of the maximum s-t flow in the given graph G. We aim to prove that $f + mk_0 \geq f_{max}$. This proof is quite similar to the proof described in class slides for max-flow min-cut theorem. There does not exist a s-t path bottlenecked by atleast k_0 in the graph G_f at the end of the inner while loop because of the loop condition. This directly translates to the fact that all the s-t paths will have bottleneck capacity less than k_0 by the end of inner while loop. To formalize our proof, we take a cut (A,B) such that A comprises of all the the nodes u in G that can be reached from s in the graph G_f through some path of bottleneck capacity atleast k_0. We define B to the the set of the nodes left (B becomes $V \setminus A$) and it consists of t since it wasn't a part of A. Thus our cut (A,B) is a valid s-t cut.

Claim: $c(u,v) < f(u,v) + k_0$ *for every edge (u,v) in the graph G, s.t.* $u \in A$ *and* $v \in B$. We prove this by contradiction. Assuming, this wasn't the case, then, $c(u,v) - f(u,v) \geq k_0$ and (u,v) would appear as a forward edge in G_f. Since there is a path of bottleneck capacity atleast k_0 from s to u, we may append this edge (u,v) to this path to obtain a path from s to v satisfying the bottleneck capacity constraints. By the way we have defined the set A, we see that v ∈ A which gives us a contradiction. Thus our claim holds true.

Claim: $f(u,v) < k_0$ *for every edge (u,v) in the graph G, s.t.* $u \in B$ *and* $v \in A$. We prove this by contradiction. Assuming, this wasn't the case, then, $f(u,v) \geq k_0$ and (v,u) would appear as a backward edge in G_f. Since there is a path of bottleneck capacity atleast k_0 from s to v, we may append this edge (v, u) to this path to obtain a path from s to u satisfying the bottleneck capacity constraints. By the way we have defined the set A, we see that u ∈ A which gives us a contradiction. Thus our claim holds true.

Having claimed that all outward edges {(u,v) | u ∈ A and v ∈ B} satisfy $c(u,v) < f(u,v) + k_0$ and all inward edges {(u',v') | v' ∈ A and u' ∈ B} satisfy $f(u',v') < k_0$, we can reduce the flow f as follows:

$$f = f_{out}(A) - f_{in}(A) = \sum_{\substack{(u,v)\in E \\ u\in A \\ v\in B}} f(u,v) - \sum_{\substack{(u',v')\in E \\ v'\in A \\ u'\in B}} f(u',v')$$

$$> \sum_{\substack{(u,v)\in E \\ u\in A \\ v\in B}} (c(u,v) - k_0) - \sum_{\substack{(u',v')\in E \\ v'\in A \\ u'\in B}} k_0$$

$$> \sum_{\substack{(u,v)\in E \\ u\in A \\ v\in B}} c(u,v) - \left(\sum_{\substack{(u,v)\in E \\ u\in A \\ v\in B}} k_0 + \sum_{\substack{(u',v')\in E \\ v'\in A \\ u'\in B}} k_0 \right)$$

$$\geq c(A,B) - mk_0 \text{.....(1)}$$

$$\boxed{\implies f + mk_0 \geq c(A,B)}$$

In the equation (1), c(A,B) represents the capacity of the cut (A,B) while mk_0 upper bounds the bracketted part. Capacity of any s-t cut is an upper bound for the maximum value of flow in the network of graph G. Thus, we can say that, $f_{max} \leq c(A,B) \leq f + mk_0$ $\boxed{\implies f_{max} \leq f + mk_0}$

Using the claim above and the fact that in the previous iteration, the value of k was $2k_0$, we can say that, $f_{max} \leq f + m * (k) = f + 2mk_0$. This proves the result of our lemma. $\boxed{\implies f \geq f_{max} - 2mk_0}$

3.2 Lemma 2

If the current value of k is k_0, the lower bound on the amount by which the flow increases in an iteration of the inner while loop is k_0

Proof : k_0 will either be the maximum capacity of any edge in G, or it will be lower than the maximum capacity. This is clear from the fact that in every iteration of the outer while loop, the value of k is being halved, ie. at any moment, the value of k is less than or equal to the maximum capacity of any edge in G. We choose a path P in G_f such that it has a bottleneck capacity of atleast k_0.
Now, we focus on the inner while loop. In every iteration, we increase the flow along any edge by c. We can directly use the fact that c is atleast k_0. Hence, during every iteration of the inner while loop, $\Delta f \geq k_0$, which is the lower bound.

3.3 Lemma 3

The inner while loop will run for O(m) times only for some specific value of k (say k_0)

Proof : From lemma 1, we know that $f \geq f_{max} - 2mk_0$. Also, in lemma 2, we proved that in every iteration of the inner while loop, the flow increases by atleast k_0. Since the value of flow in the network G, cannot exceed the maximum possible flow, it is a straightforward deduction that the inner while loop runs O(m) times.

3.4 Conclusion

The running time of the algorithm Poly-FF(G,s,t) is $O(m^2 log_2(c_{max}))$

Proof : Initial value of k is c_{max}, where c_{max} is the maximum capacity of any edge in G. In every iteration of the outer while loop, the value of k is being halved. Hence, the outer while loop will run for $O(log_2(c_{max}))$ times. From lemma 3, we know that the inner while loop will run O(m) times. Now, we need to estimate the time complexity of the inner while loop. The maximum number of edges in the path P can be m, where m is the total number of edges in the network G. Hence, finding the bottleneck capacity will take $O(m)$ (we need to iterate on all edges in the path). Also, in the for loop, we iterate over all edges in the path. Thus the for loop will take $O(m)$. So, every iteration of the inner while loop takes $O(m + m) = O(m)$ time.

$$\boxed{\text{Therefore, the final time complexity is } O(m * m * log_2(c_{max})) = O(m^2 log_2(c_{max}))}$$

which is a polynomial in the input size, as desired.

3.5 Correlation

Worst case number of augmenting paths used in the modified Ford-Fulkerson Algorithm is upper bounded by the worst case number of augmenting paths used in Poly-FF(G,s,t)

Proof : We need to devise a correlation between the sequence of augmenting paths used in the modified Ford-Fulkerson Algorithm and the sequence of augmenting paths used in Poly-FF(G,s,t). To do so, we use induction on the index of the paths in the sequence. The inductive sketch is given below:

3.5.1 Inductive Procedure

For every sequence of augmenting paths used in modified Ford-Fulkerson Algorithm, there exists another sequence of augmenting paths used in Poly-FF(G,s,t), and the two sequences are identical.

Setup : Let the residual graph of Poly-FF(G,s,t) be G_f and the residual graph of the modified Ford-Fulkerson algorithm be $G_f^{'}$.

Induction Hypothesis : For Poly-FF(G,s,t), there exists a sequence of paths $P_1, P_2..P_i$, such that $G_f = G_f^{'}$

Base Case : Consider $i = 1$. Before augmenting this path, the two residual graphs must have been the same (same as the original network). By the modified FF algorithm, P_1 is one of the maximum bottleneck capacity augmenting paths of $G_f^{'}$. We look at the same path P_1 in the Poly-FF algorithm, and we define k_0 to be the maximum value of k such that $bottleneck(P_1) \geq k_0$.
We claim that in the previous iterations of Poly-FF, no other path could have been chosen. To prove this, note that had some other path been chosen in some previous iteration, its capacity would have been greater than that of P_1. But, we know P_1 is one of the maximum bottleneck capacity augmenting paths of $G_f^{'}$ and since $G_f^{'} = G_f$, it is also one of the maximum bottleneck capacity augmenting paths of G_f. Thus, our claim is correct.
Hence, for i=1, in Poly-FF, we can choose the path to be P_1. So, in this case, we chose the same path in both the algorithms, hence the residual graph updates must be exactly similar, or in other words, $G_f = G_f^{'}$ holds even after augmenting P_1.

Inductive Step : We now consider some case where $i > 1$. By the inductive hypothesis, for Poly-FF, there exists a sequence of paths $P_1, P_2..P_{i-1}$ such that $G_f = G_f^{'}$. Let's say the modified FF algorithm now chooses the path P_i. Therefore, P_i must be one of the maximum bottleneck capacity augmenting paths. Now we have 2 cases:

1. **Case 1:** The current value of k in the Poly-FF algorithm is strictly less than the bottleneck capacity of the path P_i. In this case, we can simply choose the path P_i in Poly-FF, and perform updates in the residual graph. For this case, since the residual graph was the same till after augmenting the first i-1 paths, and we augmented the same path in both the algorithms, the residual graph would still be the same. Thus, the induction hypothesis holds for this case.

2. **Case 2:** Here, the current value of k in the Poly-FF algorithm is not strictly less than the bottleneck capacity of the path P_i. Let the current value of k in the Poly-FF algorithm be $k^{'}$. The outer while loop will run(without choosing any path) till the following condition is met:

   $$k_f \leq bottleneck(P_i) \leq 2k_f$$

   We again claim, that during all the iterations where the value of k was decreased from $k\prime$ to k_f, no path would have been chosen by Poly-FF. Had some path been chosen, its maximum bottleneck capacity would have to be greater than $2k_f$, which is a contradiction.
   Hence, we can choose P_i in the Poly-FF algorithm. Since again, we chose the same path in both

the algorithms, the updates in the residual graph would be the same, and again $G_f = G'_f$. Hence, the induction hypothesis holds.

Termination : If the modified Ford-Fulkerson Algorithm terminates (no path is found in G'_f), the Poly-FF Algorithm is bound to terminate. This is because, since $G_f = G'_f$, if modified FF algorithm cannot find a path from the source to the sink in the residual graph, it is guranteed that there is no path between the source and the sink in the residual graph. Hence, even Poly-FF terminates.

Using the inductive procedure, we can now show that for any sequence of augmenting paths used in the modified Ford-Fulkerson Algorithm, we have an identical sequence of augmenting paths in the Poly-FF Algorithm. This implies that the number of augmenting paths used in the modified FF algorithm will always be bounded by the number of augmenting paths used in the Poly-FF algorithm, which concludes the proof for correlation.

From lemma 3, the inner while loop will run for $O(m)$ times only for some specific value of k (say k_0). This means for some k_0, the inner while loop uses at most $O(m)$ augmenting paths for every iteration of the outer while loop. We have also deduced in the conclusion that the outer while loop runs $O(log_2 c_{max})$ times. So, the maximum number of augmenting paths used by Poly-FF is $O(mlog_2 c_{max})$. Therefore, using the correlation which we just proved in the above section, we can safely conclude that the modified FF Algorithm uses $O(mlog_2 c_{max})$ number of augmenting paths.