CS345 Assignment-3

Tarun Kanodia 190902
Kaustubh Verma 190424

November 14, 2021

1 Problem

While covering the topic on Maximum Flow in the class, we showed a graph with integer capacities on which the Ford-Fulkerson algorithm can be made to run in $\theta(mc_{max})$ time, where c_{max} is the maximum capacity of any edge in G. This running time is not a polynomial in the input size. The objective of this assignment problem is to slightly modify the algorithm so that it runs in polynomial time for all integer capacity graphs.

The modification required in the Ford-Fulkerson algorithm is just the following: In each iteration, pick the path with maximum capacity in the residual network G_f and use it to increase the flow in G during that iteration. The pseudocode of the modification is as follows:

Algorithm 1 Mod-FF(G,s,t)

```
f ← 0
while there exists an s-t path in G_f do
    Pick path P in G_f with maximum capacity c_max
    for each edge (x, y) ∈ P do
        if (x, y) is a forward edge then
            f(x, y) = f(x, y) + c_max
        else
            f(y, x) = f(y, x) − c_max
    Update G_f
return f
```

We want to prove the polynomial time bound on $Mod-FF$ by considering the worst case of the $Poly-FF$.

2 Poly-FF(G,s,t) algorithm

The complete algorithm as mentioned in the hints is as follows: We now prove the following series of claims

Algorithm 2 Poly-FF(G,s,t)

```
1:  f ← 0
2:  k ← maximum capacity of any edge in G
3:  while k ≥ 1 do
4:      while there exists a s-t path in G_f with capacity ≥ k do
5:          Let P be any path in G_f with capacity at least k
6:          c_b ← bottle-neck capacity of P
7:          for each edge (x, y) ∈ P do
8:              if (x, y) is a forward edge then
9:                  f(x, y) ← f(x, y) + c_b
10:             if (x, y) is a backward edge then
11:                 f(y, x) ← f(y, x) − c_b
12:         Update G_f
13:     k ← k/2                                   ▷ using integer division
14: return f
```

to show that $Poly - FF(G, s, t)$ returns the maximum $s - t$ flow in G upon its termination.

2.1 Lemma

Assume we enter the outermost while loop with value k_0 of variable k, then after the termination of inner while loop we have, $f_{max} \leq f + mk_0$, where f_{max} is value of the maximum $s - t$ flow in G and f is the current value of flow in G after inner while loop terminates.

Proof: When the inner while loop of the algorithm terminates, there does not exist any $s - t$ path in G_f with bottleneck capacity at least k_0. Let A denote the set of vertices reachable from s in G_f using only the edges with capacity at least k_0. Now let $A' = V \setminus A$. Note that $t \in A'$, because on termination of the inner while loop there is no path between s and t with bottleneck capacity at least k_0. So $C = (A, A')$ is a valid cut.

Across C, flow $f = f_{out}(A) - f_{in}(A)$ as defined in the lectures. Consider the following observations:

- **For every edge $(x, y) \in E, x \in A, y \in A'$, we have $f(x, y) > c(x, y) - k_0$**: This can be proven as follows: Assume for contradiction that $f(x, y) \leq c(x, y) - k_0$. Using construction of G_f and our assumption, we have (x, y) as forward edge in G_f and it crosses C with capacity atleast k_0. Now since $x \in A$, there is a path from s to x using only the edges with capacity at least k_0. Thus appending (x, y) to that path results in y being reachable from s, thus implying $y \in A$, which is a contradiction.

- **For every edge $(x, y) \in E$, $x \in A', y \in A$, we have $f(x, y) < k_0$:** This can be proven as follows: Assume for contradiction that $f(x, y) \geq k_0$. Using construction of G_f and our assumption, we have (y, x) as a backward edge in G_f crossing C with capacity $\geq k_0$. Also, since $y \in A$, there is a path from s to y using only the edges with capacity at least k_0. Appending edge (y, x) to this path makes x reachable from s, thus implying $x \in A$, which is a contradiction.

From the above observations, we have:

$$\begin{aligned} f &= f_{out}(A) - f_{in}(A) \\ &= \sum_{(x,y)\in E, x\in A, y\in A'} f(x,y) - \sum_{(x,y)\in E, x\in A', y\in A} f(x,y) \\ &> \sum_{(x,y)\in E, x\in A, y\in A'} (c(x,y) - k_0) - \sum_{(x,y)\in E, x\in A', y\in A} k_0 \\ &> \sum_{(x,y)\in E, x\in A, y\in A'} c(x,y) - (\sum_{(x,y)\in E, x\in A, y\in A'} k_0 + \sum_{(x,y)\in E, x\in A', y\in A} k_0) \\ &\geq c(A, A') - mk_0 \end{aligned}$$

where $c(A, A')$ is the capacity of the cut and number of terms in the second summation is bounded by the number of edges of the graph.

Thus we have $c(A, A') \leq f + mk_0$. We know from the lectures that value of max-flow is less than or equal to capacity of any $s - t$ cut. Thus we get, $f_{max} \leq f + mk_0$. Also from this equation, we get that before entering the inner while loop, we have $f_{max} \leq f + 2mk_0$, as at the end of previous iteration the value would have been $k' = 2k_0$, and $f_{max} \leq f + mk' = f + 2mk_0$.

2.2 Time Complexity for Poly-FF(G,s,t)

Now we are in position to do complete analysis of Poly-FF(G,s,t) Algorithm. We start as follow:-

1. Number of iterations of outermost while loop can be calculated as follows:
   - The outermost loop runs till $k \geq 1$.
   - After i number of iterations, $k = \frac{c_{max}}{2^i}$. As loop will only run until $k \geq 1$

$$\begin{aligned} k &\geq 1 \\ \frac{c_{max}}{2^i} &\geq 1 \\ c_{max} &\geq 2^i \\ log_2(c_{max}) &\geq i \end{aligned}$$

   - So total number of iterations are $1 + \log_2(c_{max})$.

   $\implies$ Number of iterations of outermost loop is $O(log_2(c_{max}))$.

2. Number of iterations of the inner loop can be calculated as follows:
   - Let value of variable k for the current iteration be k_0
   - We know P considered by algorithm is a s-t path with capacity $\geq k_0$.
   - So $c_b \geq k_0$ as all edges in path P has capacity $\geq k_0$
   - Hence each iteration increases the current total flow (f) by c_b, as s has only an outgoing edge $\implies$ current total flow is increased by atleast k_0 in a single iteration of inner while loop.
   - By Lemma, $f_{max} \leq f + 2mk_0$, before entering the inner while loop.
   - So, the number of iterations of middle loop is bounded by $2m$.
     Proof: Assume on the contrary that number of iterations of middle loop $= x > 2m$. we just proved each iteration of middle loop increases the flow f by atleast k_0. So after x iterations $f' = f + xk_0 > f + 2mk_0 \geq f_{max}$.
     $\implies$ f'>f_{max} which is a contradiction as f_{max} is the maximum flow possible.

   $\implies$ Number of iterations of the inner while loop is $O(m)$.

3. Number of iterations of the innermost for loop can be calculated as follows:
   - Innermost loop iterates over path P.
   - A path contains each edge at most once.

   $\implies$ The innermost for loop runs for $O(m)$ times as it can iterate over each edge atmost once.

Therefore, the overall time complexity:

$$\begin{aligned}\text{Time complexity} &= O(\log_2(c_{max})) \times O(m) \times O(m) \\ &= O(m^2 \log_2(c_{max}))\end{aligned}$$

$\implies$ **Time complexity for Poly-FF(G,s,t) is** $O(m^2 log_2(c_{max}))$
Also, it is clear from above that the total number of augmenting paths used in Algorithm Poly-FF(G,s,t) in the worst case is upper bounded by $\log_2(c_{max}) \times 2m = 2m \log_2(c_{max}) = O(m \log_2(c_{max}))$.

2.3 Proof of Correctness

The Algorithm Poly-FF(G,s,t) returns the maximum s-t flow in G upon termination
The slight modification from the original Ford-Fulkerson algorithm in the inner while loop is with the intention of improving time complexity, by considering paths with a higher bottleneck capacity, so that flow reaches to max-flow value early, leading to a polynomial time algorithm. However all the edges are still integers which implies c_b is also an integer. So c_b must be atleast 1. In the last iteration when $k = 1$, we consider all $s - t$ paths like Ford Fulkerson algorithm. Since the original Ford-Fulkerson algorithm returns max flow, so should Poly-FF(G,s,t).

Proof: Assume to the contrary that when $Poly - FF(G, s, t)$ exits, it returns a flow $f < f_{max}$. Now let us consider the residual network at exit of $Poly - FF(G, s, t)$ and feed it to the original Ford-Fulkerson algorithm. As $f < f_{max}$ as it is impossible to return $f > f_{max}$ (otherwise it wont be max flow) so Ford-Fulkerson algorithm finds a $s - t$ path P in this residual network with bottleneck capacity $c_b \geq 1$. But this means the residual network has a $s - t$ path with capacity $c_b \geq 1$. We also know last iteration of $Poly - FF(G, s, t)$ has $k = 1$ (as we are considering integer divisions and all integer divisions reach 1 eventually) and $Poly - FF(G, s, t)$ has exited so there should have been no $s - t$ path in residual network with capacity ≥ 1, thus leading to a contradiction. Hence Poly-FF(G,s,t) returns max s-t flow in graph G.

2.4 Termination

Actually the same proof for termination works as given for original Ford-Fulkerson algorithm as that proof is based on fact that at each iteration we increase f by a positive integer and the max flow of any finite graph is a finite value. Both of these results are still valid as a result Poly-FF(G,s,t) is also guaranteed to terminate.

3 Mod-FF Algorithm

3.1 Number of Worst Case Augmentation Paths

The idea is to show that for a given run of Mod-FF(G,s,t), an identical run is possible in Poly-FF(G,s,t) which means the number of augmentation paths used in worst case of Poly-FF(G,s,t) is $\geq$ augmentation paths used in Mod-FF(G,s,t)
Let $P_1, P_2 \ldots P_n$ be the the augmenting paths used by Mod-FF(G,s,t) during its complete execution as the termination is guaranteed as shown later .
We can see that as path with maximum bottleneck capacity in residual network currently can always be chosen by Poly-FF(G,s,t) as the next augmentation path.
Assume on the contrary that the path with maximum capacity(say P') can not be chosen, rather some other path(say P) is chosen. Path P can only be chosen if its bottleneck capacity $\geq$ current value of k. Also path P' can not be chosen as next augmentation path only if capacity of $P' < k$ but as P' is the path with maximum capacity so capacity of $P' \geq$ capacity of $P \geq k$ which leads us to a contradiction. Hence the path

with max bottleneck capacity can always be chosen as next augmentation path.
We claim that sequence $P_1, P_2 \dots P_n$ is also a valid sequence of augmentation paths used by Poly-FF(G,s,t). We do so using strong induction on the sequence $P_1, P_2 \dots P_i$ as follow:
Induction Hypothesis: $P_1, P_2 \dots P_i$ is a valid order of augmentation paths used for both Mod-FF(G,s,t) and Poly-FF(G,s,t) as well as the residual network after using augmentation paths $P_1, P_2 \dots P_i$ is identical in Mod-FF(G,s,t) and Poly-FF(G,s,t)
Base case We see that at start when we have not used any augmentation path both algorithms have identical residual network.
Induction Step By induction hypothesis we have used used augmentation paths $P_1, P_2 \dots P_i$ (possibly none) and we have identical residual networks. Now Mod-FF(G,s,t) uses augmentation path P_{i+1} which must be the $s-t$ path in the residual network with max capacity. As seen above it is always possible to take the path with max capacity as next augmentation path for Poly-FF(G,s,t). So the $P_1, P_2 \dots P_i, P_{i+1}$ is a valid sequence of augmentation paths used for both the algorithms and as we can see that if we have same residual network currently and are using the same path the residual network after augmenting using this new path(P_{i+1}) will be same.
Hence by induction we can see $P_1, P_2 \dots P_n$ is a valid sequence of augmentation paths used for both Algorithms and as at the end of using this sequence of augmentation paths Mod-FF(G,s,t) returns, i.e, has no $s-t$ path in its residual network so the Poly-FF(G,s,t) will also return and have no more $s-t$ paths in its residual network. Now we see that for the worst case run of Mod-FF(G,s,t) their exists a run of Poly-FF(G,s,t) with same number of steps(actually same steps) which implies the worst case number of augmentation paths used in Poly-FF(G,s,t)$\geq$ worst case number of augmentation paths used in Mod-FF(G,s,t).

Therefore,the worst case number of augmentation path used in Mod-FF(G,s,t) is upper bounded by worst case number of augmentation paths used in Poly-FF(G,s,t).

3.2 Proof of Correctness

The Algorithm Mod-FF(G,s,t) returns the maximum s-t flow in G upon termination
The only modification from the original Ford-Fulkerson algorithm is the way of choosing paths. However all the edges all still integers which implies c_b is also an integer. As c_b is an integer and must be >0 . c_b must be atleast 1. Assume to contrary that when Mod-FF(G,s,t) exits it returns a $f < f_{max}$. Now let us take residual network at exit of Mod-FF(G,s,t) and feed it to original Ford-Fulkerson algorithm. As $f < f_{max}$ so Ford-Fulkerson algorithm finds a s-t path P in this residual network with bottleneck capacity $c_b \geq 1$. But this means the residual network has a s-t path with capacity $c_b \geq 1$. We also know Mod-FF(G,s,t) exited which implies there does not exist a s-t path in residual network. A contradiction. Hence Mod-FF(G,s,t) returns max $s-t$ flow in graph G.

3.3 Termination

Actually the same proof for termination works as given for original Ford-Fulkerson algorithm as that proof is based on fact that at each iteration we increase f by a positive integer and the max flow of any finite graph is a finite value. Both of these results are still valid and as a result Mod-FF(G,s,t) is also guaranteed to terminate.