CS345 : Assignment 4

Dhruv Gupta (220361)

October 2024

Dhruv Gupta: 220361

Question 1

Consider the following graph G (Figure 1) from the lectures.

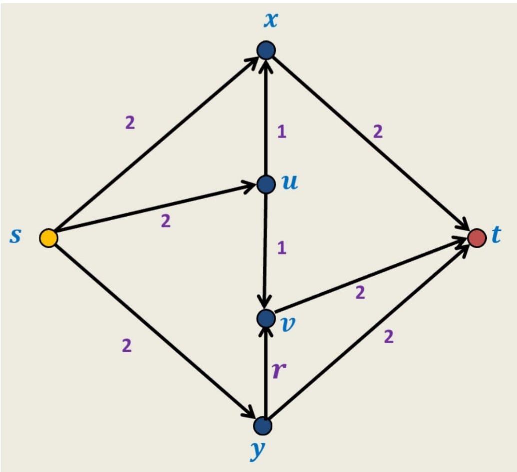

Figure 1: Graph G *(Credits : Lecture Slides)*

Here $r = \frac{\sqrt{5}-1}{2}$, which is the root of the equation $1 - r = r^2$. Note that

- $0 < r < 1 \implies r^k > r^{k+1}\ \forall k \geq 0$.
- $1 - r = r^2 \implies r^k - r^{k+1} = r^{k+2} \iff r^{k+1} + r^{k+2} = r^k \iff r^k - r^{k+2} = r^{k+1}$.

. We will heavily use these properties in our analysis later. Now consider the following paths -

- $p_0 = (s, u, v, t)$
- $p_1 = (s, y, v, u, x, t)$
- $p_2 = (s, u, v, y, t)$
- $p_3 = (s, x, u, v, t)$

While applying the Ford Fulkerson algorithm we will chose the paths in the following order - $p_0, p_1, p_2, p_1, p_3, p_1, p_2, p_1, p_3, ...$ and so on. Note that p_0 is taken only first time. Then p_1, p_2, p_1, p_3, this sequence is repeated indefinitely until the algorithm terminates. We will prove that this will never happen. Let P_i be the i^{th} path in the sequence $(i = 0, 1, 2, ...)$ and let G_i denote the residual graph after we have used the sequence upto P_i (included) and let f_i be the flow, and F_i be the value of the flow at this time. First we make the graph G_0. Note that edge $c(u, v)$ decreases to 0 and a back edge (v, u) with capacity 1 appears. Also $c(s, u)$ and $c(v, t)$ reduce to 1 and back edges (u, s) and (t, v) with capacity 1 appear. Also, $F_0 = 1$. Now we will make the following claim -

Claim : For all $i \geq 0$, after using paths upto P_{4i} (included) in the residual graph G_{4i}, edges will have following capacities [1] -

- $c(u, x) = r^{2i}$
- $c(y, v) = r^{2i+1}$
- $c(u, v) = 0$
- $c(x, u) = 1 - r^{2i}$
- $c(v, y) = r - r^{2i+1}$
- $c(v, u) = 1$
- $c(s, y) = r^{2i-1} + r^2$
- $c(s, u) = r^{2i}$
- $c(s, x) = r^{2i+1} + r^2 + 1$
- $c(x, t) = r^{2i-1} + r^2$
- $c(v, t) = r^{2i+1} + r^2$
- $c(y, t) = r^{2i} + 1$

Also the flow f_{4i} has value $F_{4i} = 3 + 2r - 2r^{2i-1}$.

Proof : We will prove this by induction. For base case $(i = 0)$ we can verify from G_0 the capacities for all the edges and $F_0 = 3 + 2r - \frac{2}{r} = 3 + 2r - 2(1 + r) = 1$. Thus the base case is satisfied. For the induction step we will assume that the claim is true for i. Now we will utilize the paths $P_{4i+1} = p_1, P_{4i+2} = p_2, P_{4i+3} = p_1, P_{4i+4} = p_3$ to get the graph $G_{4(i+1)}$.

- We can observe that the bottleneck capacity of path P_{4i+1} is $c(y, v) = r^{2i+1}$. We can verify this by comparing capacity of (y, v) with the capacity of edges $(s, y), (v, u), (u, x), (x, t)$). Therefore we can send a flow of r^{2i+1} along this path. The capacities of these edges in G_{4i+1} now become
  - $c(s, y) = r^{2i-1} + r^2 - r^{2i+1} = r^{2i} + r^2$
  - $c(y, v) = 0, c(v, y) = r$
  - $c(v, u) = 1 - r^{2i+1}, c(u, v) = r^{2i+1}$

[1] Note that we haven't mentioned all the backedges here. This is because those edges are not used in any $s - t$ path that we take. Only the edges which are present in atleast one $s - t$ path in some residual network are needed here.

- $c(u, x) = r^{2i} - r^{2i+1} = r^{2i+2}, c(x, u) = 1 - r^{2i} + r^{2i+1} = 1 - r^{2i+2}$
- $c(x, t) = r^{2i-1} + r^2 - r^{2i+1} = r^{2i} + r^2$

Capacities for rest of the edges in the graph (that are mentioned in the claim) remain same as before. Also $F_{4i+1} = 3 + 2r - 2r^{2i-1} + r^{2i+1}$.

- Now for path P_{4i+2}, bottleneck capacity is clearly $c(u, v) = r^{2i+1}$. The capacities of these edges in G_{4i+2} now become
  - $c(s, u) = r^{2i} - r^{2i+1} = r^{2i+2}$
  - $c(y, v) = r^{2i+1}, c(v, y) = r - r^{2i+1}$
  - $c(v, u) = 1, c(u, v) = 0$
  - $c(y, t) = r^{2i} + 1 - r^{2i+1} = r^{2i+2} + 1$

  Also $F_{4i+2} = 3 + 2r - 2r^{2i-1} + 2r^{2i+1} = 3 + 2r - 2r^{2i-1}(1 - r^2) = 3 + 2r - 2r^{2i}$.

- Now for path P_{4i+3}, bottleneck capacity is clearly $c(u, x) = r^{2i+2}$. The capacities of these edges in G_{4i+3} now become
  - $c(s, y) = r^{2i} + r^2 - r^{2i+2} = r^{2i+1} + r^2$
  - $c(y, v) = r^{2i+1} - r^{2i+2} = r^{2i+3}, c(v, y) = r - r^{2i+3}$
  - $c(v, u) = 1 - r^{2i+2}, c(u, v) = r^{2i+2}$
  - $c(u, x) = r^{2i+2} - r^{2i+2} = 0, c(x, u) = 1 - r^{2i+2} + r^{2i+2} = 1$
  - $c(x, t) = r^{2i} + r^2 - r^{2i+2} = r^{2i+1} + r^2$

  Capacities for rest of the edges in the graph (that are mentioned in the claim) remain same as before. Also $F_{4i+3} = 3 + 2r - 2r^{2i} + r^{2i+2}$.

- Finally for path P_{4i+4}, bottleneck capacity is clearly $c(u, v) = r^{2i+2}$. The capacities of these edges in G_{4i+4} now become
  - $c(s, x) = r^{2i+1} + r^2 + 1 - r^{2i+2} = r^{2i+3} + r^2 + 1$
  - $c(v, u) = 1 - r^{2i+2} + r^{2i+2} = 1, c(u, v) = r^{2i+2} - r^{2i+2} = 0$
  - $c(u, x) = r^{2i+2}, c(x, u) = 1 - r^{2i+2}$
  - $c(v, t) = r^{2i+1} + r^2 - r^{2i+2} = r^{2i+3} + r^2$

  Capacities for rest of the edges in the graph (that are mentioned in the claim) remain same as before. Also $F_{4i+4} = 3 + 2r - 2r^{2i} + 2r^{2i+2} = 3 + 2r - 2r^{2i}(1 - r^2) = 3 + 2r - 2r^{2i+1}$.

We can see that in $G_{4(i+1)}$ the capacities of the above mentioned edges in $G_{4(i+1)}$ are as follows

- $c(u, x) = r^{2(i+1)}$
- $c(y, v) = r^{2(i+1)+1}$
- $c(u, v) = 0$
- $c(x, u) = 1 - r^{2(i+1)}$
- $c(v, y) = r - r^{2(i+1)+1}$

- $c(v, u) = 1$
- $c(s, y) = r^{2(i+1)-1} + r^2$
- $c(s, u) = r^{2(i+1)}$
- $c(s, x) = r^{2(i+1)+1} + r^2 + 1$
- $c(x, t) = r^{2(i+1)-1} + r^2$
- $c(v, t) = r^{2(i+1)+1} + r^2$
- $c(y, t) = r^{2(i+1)} + 1$

Also, $F_{4(i+1)} = 3+2r-2r^{2(i+1)-1}$. Thus we have proven our claim for $i+1$. This completes the proof.

Using the above claim, we can conclude that we can continue this sequence indefinitely, i.e., if we use this sequence the Ford-Fulkerson algorithm would never terminate, since in every residual network that we get in this case, an $s - t$ path always exists. Also, since $F_{4i} = 1 + 2 * \sum_{k=1}^{2i} r^k = 1 + 2r\frac{(1-r^{2i})}{(1-r)} = 1 + 2r\frac{(1-r^{2i})}{r^2} = 1 + 2\frac{(1-r^{2i})}{r} = 1 + 2(1 - r^{2i})(1 + r) = 1 + 2(1 + r - (r^{2i} + r^{2i+1})) = 1 + 2(1 + r - r^{2i-1}) = 3 + 2r - 2r^{2i-1}$. Therefore, asymptotically, i.e., for $i \to \infty$, $F = 3 + 2r < 5$ (since $r < 1$). Although, by choosing the paths $(s, x, t), (s, u, v, t), (s, y, t)$ in the Ford-Fulkerson Algorithm, the algorithm terminates to give a max flow of value 5.
Thus we have proven that first, there exists a graph with 6 nodes with real capacities, on which the Ford-Fulkerson algorithm may never terminate, and second, the flow that we get in such a case can be smaller than the max flow, even asymptotically.

Dhruv Gupta: 220361

Question 2

We will use a **doubly linked list** L to maintain the elements in the multiset and a variables mx to store the current maximum element. Therefore our multiset will be $S = (L, mx)$. Each node in L contains following data - $val, prev, next$. We will assume following functions available for L –

- **Head**(L) **-** Returns the head of L in $O(1)$ time.
- **Insert**(L, x) **-** Insert the element x at the head of L and updates $mx = max(mx, x)$. Takes $O(1)$ time.
- **Delete**(L, p) **-** Given a pointer p to an element, deletes that element from L in $O(1)$ time.
- **Median**(L) **-** Returns the median of all the elements in L in $O(n)$ time (where n is the number of elements in L). This can be done by copying all the elements in a temporary array, then finding the median using the divide and conquer technique (as discussed in class).
- **Print**(L) **-** Prints all the elements present in L in $O(n)$ (where n is the number of elements in L) time .

We assume that the list is initially empty and $mx = -\infty$. Also all the edge cases like operations in empty list are handled by the above mentioned list functions. We can implement the multiset functions as follows -

```
INSERT(S = (L, mx), x){
    mx ← max(mx, x);
    Insert(L, x);
}
DELETE-LARGER-HALF(S = (L, mx)){
    mid ← Median(L);
    p ← Head(L);
    while(p ≠ NULL) { // Delete all the elements in the larger half
        if(val(p) ≥ mid) { Delete(L, p); }
        p ← next(p);
    }
    p ← Head(L);
    mx ← −∞;
    while(p ≠ NULL) { // Update the max element
        mx ← max(val(p), mx);
        p ← next(p);
    }
}
REPORT-MAX(S = (L, mx)){
    return mx;
}
```

To output all the elements in S, we can simply use the **Print**(L) function. We claim that this implementation will take $O(m)$ time for any sequence of m operations. We will prove this claim using amortized time complexity analysis.

Amortized Analysis :

We will assume each $O(1)$ time operation takes some constant time c. Let n be the number of elements after the i^{th} operation. Let us define the potential function ϕ as

$$\phi(i) = 8c * number\ of\ elements\ after\ i^{th}\ operation = 8cn$$

For Insert, number of elements changes from $n-1$ to n. For Delete-Larger-Half, number of elements changes from $2n+k$ ($k=0$ or 1) to n. For Report-Max, number of elements do not change. We can summarize the actual time, change in potential function and the amortized time taken by each type of operation as follows -

	Actual time	$\Delta\phi$	**Amortized time**
Insert	$2c$	$8c$	$10c$
Delete-Larger-Half	$(3+3k+8n)c$	$(-8n-8k)c$	$(-5k+3)c \leq 3c$
Report-Max	c	0	c

Note that in Delete-Larger-Half, the first while loop iterates for $2n+k$ times ($k=0$ or 1), then the second while loop iterates for n times. Also, the **Median** function takes $c(2n+k)$ time. Thus actual time is $c(1+(2n+k)+2(2n+k)+2+2(n)) = c(3+3k+8n)$ and $\Delta\phi = 8c(n) - 8c(2n+k) = c(-8n-8k)$. Rest of the entries are trivial to fill. Now, we can see that amortized taken by the i^{th} operation is atmost $10c, i.e., O(1) \implies$ amortized time complexity for m operations $= O(m) \implies$ total time taken by any sequence of m operations is $O(m)$.