CS345: Assignment 4

Pragati Agrawal (220779)

November 2024

Pragati Agrawal: 220779 (Question 1)

Below is the graph discussed in the lectures for which the Ford-Fulkerson algorithm may never terminate, ($r = (\sqrt{5} - 1)/2$). We can see that the maximum flow possible in the network is 5. (Choose the set of paths (s, x, t), (s, y, t) and (s, u, v, t). This gives a flow of $2 + 2 + 1 = 5$ in the network, and this is the maximum possible, as there is no more $s - t$ path left in the residual network)

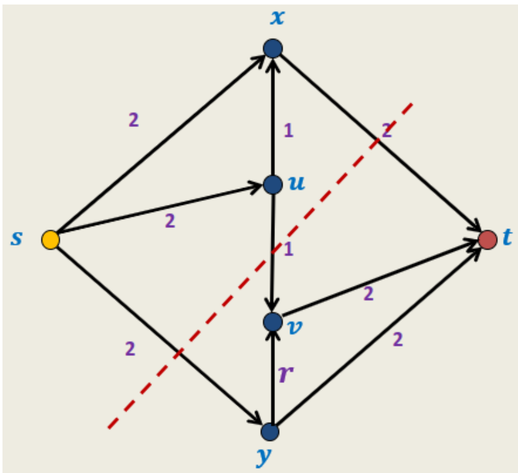

Figure 1: Graph G of real edge weights

The Ford-Fulkerson algorithm may never terminate if it chooses the following sequence of paths: We first choose the path p_0 and then repeatedly choose the same sequence of paths (p_1, p_2, p_3, p_4, p_1, p_2, p_3, p_4...) as mentioned below:

- $p_0 : (s, u, v, t)$
- $p_1 : (s, y, v, u, x, t)$
- $p_2 : (s, u, v, y, t)$
- $p_3 : (s, y, v, u, x, t)$
- $p_4 : (s, x, u, v, t)$

Let $P' = (p_1, p_2, p_3, p_4)$ be the sequence of paths which will be repeated. p_0 will be performed only once. Let F be the flow passes through the network at any point in time, initially $F = 0$. We also define a tuple $T = \{c(u, x), c(y, v), c(u, v))\} = \{r^0, r^1, r^0\}$ initially.

When we run p_0, the bottleneck capacity edge would be (u, v) and the residual capacities of the edges would become:

- $c(s, x) = c(x, t) = c(s, y) = c(y, t) = 2$
- $c(s, u) = c(v, t) = c(v, u) = c(u, x) = 1$
- $c(y, v) = r$
- $c(x, u) = c(u, v) = c(v, y) = 0$

Since the bottle-neck capacity of the $s - t$ path p_0 is 1, therefore now $F = 1$, and $T = \{r^0, r^1, 0\}$. This is the graph we start with before any iteration of P'.

Claim: After k iterations of the sequence P', the tuple $T = \{r^{2k}(r^0, r^1, 0)\}$ and $F = 1 + 2\sum_{i=1}^{2k} r^i$. The residual capacities of the following edges will be:

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

Proof:
Base Case (k=0): For $k = 0$, we have already executed p_0 and the capacities all edges became:

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

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

These capacities satisfy the claim for $k = 0$. Also the flow becomes $F = 1$ and $T = \{r^0, r^1, 0\}$ which clearly show satisfaction of our claim.

Induction Case: We assume that it holds true for some $k > 0$, and we will show that it also holds for $k + 1$.

We run p_1 in the $(k + 1)^{th}$ iteration, which has the bottleneck capacity edge (y, v) among the edges of $p_1 : \{(s, y), (y, v), (v, u), (u, x), (x, t)\}$. The residual capacities become:

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

Since the bottle-neck capacity of the $s-t$ path p_1 is $c(y, v) = r^{2k+1}$, so now $F = 1+2\sum_{i=1}^{2k} r^i + r^{2k+1}$, and $T = \{r^{2k+2}, 0, r^{2k+1}\}$.

Now we run p_2, which has the bottleneck capacity edge (u, v) among the edges of p_2: $\{(s, u), (u, v), (v, y), (y, t)\}$ and the residual capacities would become:

- $c(s, x) = 1 + r^2 + r^{2k+1}$

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

Since the bottle-neck capacity of the s−t path p_2 is $c(u,v) = r^{2k+1}$, so now $F = 1+2\sum_{i=1}^{2k} r^i + 2r^{2k+1}$, and $T = \{r^{2k+2}, r^{2k+1}, 0\}$.

Now we run p_3, which has the bottleneck capacity edge (u,x) among the edges of $p_3 : \{(s,y),(y,v),(v,u),(u,x),(x,t)\}$ and the residual capacities would become:

- $c(s,x) = 1 + r^2 + r^{2k+1}$
- $c(s,u) = r^{2k+2}$
- $c(s,y) = r^2 + r^{2k} - r^{2k+2} = r^2 + r^{2k+1}$
- $c(x,u) = 1 - r^{2k+2} + r^{2k+2} = 1$
- $c(u,x) = r^{2k+2} - r^{2k+2} = 0$
- $c(u,v) = 0 + r^{2k+2} = r^{2k+2}$
- $c(v,u) = 1 - r^{2k+2}$
- $c(v,y) = r - r^{2k+1} + r^{2k+2} = r - r^{2k+3}$
- $c(y,v) = r^{2k+1} - r^{2k+2} = r^{2k+3}$
- $c(x,t) = r^2 + r^{2k} - r^{2k+2} = r^2 + r^{2k+1}$
- $c(v,t) = r^2 + r^{2k+1}$
- $c(y,t) = 1 + r^{2k+2}$

Since the bottle-neck capacity of the $s - t$ path p_3 is $c(u,x) = r^{2k+2}$, so now $F = 1 + 2\sum_{i=1}^{2k} r^i + 2r^{2k+1} + r^{2k+2}$, and $T = \{0, r^{2k+3}, r^{2k+2}\}$.

Now we run p_4, which has the bottleneck capacity edge (u,v) among the edges of $p_4 : \{(s,x),(x,u),(u,v),(v,t)\}$ and the residual capacities would become:

- $c(s,x) = 1 + r^2 + r^{2k+1} - r^{2k+2} = 1 + r^2 + r^{2k+3} = 1 + r^2 + r^{2(k+1)+1}$
- $c(s,u) = r^{2k+2} = r^{2(k+1)}$
- $c(s,y) = r^2 + r^{2k+1} = r^2 + r^{(2(k+1)-1}$
- $c(x,u) = 1 - r^{2k+2} = 1 - r^{2(k+1)}$
- $c(u,x) = 0 + r^{2k+2} = r^{2k+2} = r^{2(k+1)}$
- $c(u,v) = r^{2k+2} - r^{2k+2} = 0$
- $c(v,u) = 1 - r^{2k+2} + r^{2k+2} = 1$
- $c(v,y) = r - r^{2k+3} = r - r^{2(k+1)+1}$
- $c(y,v) = r^{2k+3} = r^{2(k+1)+1}$
- $c(x,t) = r^2 + r^{2k+1} = r^2 + r^{2(k+1)-1}$
- $c(v,t) = r^2 + r^{2k+1} - r^{2k+2} = r^2 + r^{2k+3} = r^2 + r^{2(k+1)+1}$
- $c(y,t) = 1 + r^{2k+2} = 1 + r^{2(k+1)}$

Since the bottle-neck capacity of the $s-t$ path p_4 is $c(u,v) = r^{2k+2}$, so now $F = 1+2\sum_{i=1}^{2k} r^i + 2r^{2k+1} + 2r^{2k+2} = 1+2\sum_{i=1}^{2(k+1)} r^i$, and $T = \{r^{2k+2}, r^{2k+3}, 0\} = \{r^{2(k+1)}(r^0, r^1, 0)\}$.

We have shown above that all capacities, flow F and tuple T satisfy the claim for $k+1$. Thus our claim is proved.

Using our claim, we can say that the Ford-Fulkerson algorithm will never terminate, because we can run the sequence of paths P' infinitely and after each iteration, the flow value will increase by some powers of r. On infinite iterations the flow will tend to $F \to 1+2\sum_{i=1}^{\infty} r^i \to= 1+2(r/(1-r)) = 1+2(1+r) = 3+2r = 2+\sqrt{5} < 5$. But we had already shown that we have a sequence of paths that will give us a max-flow of 5 in just 3 iterations of the Ford-Fulkerson algorithm.

Pragati Agrawal: 220779 (Question 2)

We maintain **a doubly linked list** L to solve the problem. Additionally, we maintain the current maximum value in the multiset as cur_max. So, $S = (L, cur\_max)$. We assume $Head$ pointer stores the head of the doubly linked list L and the following operations can be performed:

- **Insert(L, x):** Inserts a node with value x at the beginning of the list L in $O(1)$ time.
- **Delete(L, p):** Deletes the node in the list L pointer to by pointer p, in $O(1)$ time.
- **FindSize(L):** Traverses the list L and finds the size of the list, in $O(size\ of\ list)$ time.
- **CreateNewArray(n):** Given a number n, creates an empty array of size n.
- **FormArray(L, V):** Traverses the list L and forms an array V consisting of all the elements in the list, in $O(size\ of\ list)$ time.
- **FindMedian(V):** Given an array V of size n, returns its median M in $O(n)$ time.
- **PrintList(L):** Prints the values of each node in the list in $O(size\ of\ list)$ time.

Pseudo-code for the Algorithm:

$S = (L, cur\_max)$.

Algorithm 1 DO-ALL-OPERATIONS(S, x)

INSERT$(S, x)\{$
    Insert(L, x);
    $cur\_max = max(cur\_max, x)$;
$\}$
REPORT-MAX$(S)\{$
    return cur_max;
$\}$
DELETE-LARGER-HALF$(S)\{$
    $n \leftarrow$ **FindSize**(L); $V \leftarrow$ **CreateNewArray**(n);
    FormArray(L, V); $M \leftarrow$ **FindMedian**(V);
    $p \leftarrow$**Head**(L); $cur\_max = -\infty$;
    while$(p \neq NULL)\{$
        if$(p \rightarrow val \geq M)$ **Delete**(L, p);
        else $cur\_max = max(cur\_max, p \rightarrow val)$;
        $p = p \rightarrow next$;
    $\}$
$\}$
PRINT-LIST$(S)\{$
    $p \leftarrow$**Head**(L);
    while$(p \neq NULL)\{$
        print$(p \rightarrow val)$; $p = p \rightarrow next$;
    $\}$
$\}$

Description of the Algorithm:

We initialise cur_max to some very low number, say $-\infty$. Now to implement the operations of the multiset S, we do the following:

- **INSERT**(S, x)**:** Simply perform **Insert(**L, x**)**. Also, set $cur\_max = max(cur\_max, x)$.
- **REPORT-MAX**(S)**:** Simply return cur_max.
- **DELETE-LARGER-HALF**(S)**:** We first find the median M of the list and also reset cur_max to $-\infty$. Now we iterate over all nodes p in the list, and if its value is greater than or equal to M, we perform **Delete(**L, p**)**. This ensures that the elements lying in the larger half i.e. $\lceil |S|/2 \rceil$ get removed from the list. Otherwise, we update $cur\_max = max(cur\_max, p \rightarrow val)$, so that cur_max stores the maximum of only the elements now present in the list.

Amortized Analysis of Time Complexity:

Let c be the time taken for any constant time operation.

$$\phi = 8c * \textit{number of elements in the list}$$

We can see that $\phi(0) = 0$ and $\phi(i) \geq 0 \; \forall \; i \geq 0$.

Operation	**Actual Cost**	$\Delta\phi$	**Amortized Cost**
INSERT(S, x)	c	$8c$	$9c$
REPORT-MAX(S)	c	0	c
DELETE-LARGER-HALF(S)	$4cn + c$	$-4cn$	c

INSERT and REPORT-MAX are constant time operations as per our implementation. So their actual cost is c. In INSERT, the number of elements increases by 1, so $\Delta\phi = 8c(n+1) - 8cn = 8c$. In REPORT-MAX, the number of elements remains the same.
DELETE-LARGER-HALF involves iterating over the list 4 times: FindSize, FormArray, FindMedian and the while loop, all take $O(n)$ time if the list size is n. So total $4cn$ time and some constant time operations form the actual cost $4cn + c$. Now if $n = 2m + 1$ (odd) then final size of list will become m and $\Delta\phi = 8cm - 8cn = 8c(n-1)/2 - 8cn = 4cn - 4c - 8cn = -4cn - 4c < -4cn$. Similarly, if $n = 2m$ (even) then too final size of list will become m and $\Delta\phi = 8cm - 8cn = 8c(n)/2 - 8cn = 4cn - 8cn = -4cn$. So in both the cases, $\Delta\phi$ is atmost $-4cn$.
Hence we can see that each operation takes some constant time (amortized). The overall time complexity of a sequence of m operations is $O(9cm) = O(m)$. The PRINT-LIST will take $O(n)$ time, if list size is n.