Project Report
Maximum Flow Algorithms

Introduction

Imagine a factory that is located at Mapusa and has a warehouse in Ponda. There is no direct road from Mapusa to Ponda, the route travels through various cities and small towns. And there are many ways one can reach from Mapusa to Ponda. The entire setup of routes and towns can be thought of as a directed graph with a special vertex called source (Mapusa) and sink (Ponda). Lets us say that a shipping company employs one truck between each city. So, the trucks have different capacities to carry goods. Suppose there is a route that goes from Mapusa - Porvori - Panaji - Korli - Banastari - Farmagudi - Ponda. The truck between Mapusa and Porvori has a capacity to carry 50 crates, trucks between Porvori and Panaji can carry 70 crates, trucks between the Panaji can carry 20 crates, and so on.

Now, the factory manager (source) has to decide how many crates of goods to produce each day so that all the goods produced reach the warehouse (sink) and none of them piles up at any of the intermediate stops. This is a Maximum Flow problem. The flow here is the number of cates that travel from one town/ city to another (from one vertex to another) and each road between towns has a truck with a capacity (edge has a capacity that it can bear).

Let us define the Maximum Flow problem more formally. Let us take a directed graph. We can think of each edge as a conduit for the material. Each conduit has stated capacity, given as the maximum rate at which the material can flow through the conduit. Vertices are conduit junctions, and other than source and sink, materials can flow through them without collecting in them. In other words, the rate at which materials enter the vertex must equal the rate at which it leaves the vertex. We call this property the “flow conservation”.

In the maximum flow problem, we wish to compute the greatest rate at which we can ship the material from the source to the sink without violating any capacity constraints.

1. Flow network

We will take a look at the graph-theoretic definition of a flow network. A **flow network** is directed graph G = (V, E) in which each edge $(u, v) \in E$ has a **capacity** $c(u, v) \geq 0$. If $(u, v) \in E$ then $(v, u) \notin E$, i.e there is no edge in the reverse direction. If $(u, v) \notin E$ then simply $c(u, v) = 0$, and e disallow self-loops. Two special vertices are there called **source** s and **sink** t such that s has no entering edge $(v, s) \notin E$, and t has no leaving edge $(t, v) \notin E$. Each vertex v lies on some path which goes from source s to sink t, i.e $\forall v \in V$ there is a path from $s \rightarrow v \rightarrow t$. The graph is therefore connected and, since every vertex other than s has at least one entering edge $|E| \geq |V| - 1$

A **flow** is a real valued function $f : V \times V \rightarrow \mathbb{R}$ that satisfies the following two properties:

Capacity constraint: For all u, v ∈ V, we require $0 \leq f(u, v) \leq c(u, v)$.
Flow conservation: For all u ∈ V - { s, t }, we require

$$\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v),$$

that is, flow in equals flow out for any vertex u. For (u, v) ∉ E, $f(u, v) = 0$

The value of | f | of a flow is defined as

$$\sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s),$$

that is, the total flow out of the source minus the flow into the source.

Example of the Lucky Puck Company and its Flow network (the shipping route from factory s to warehouse t)

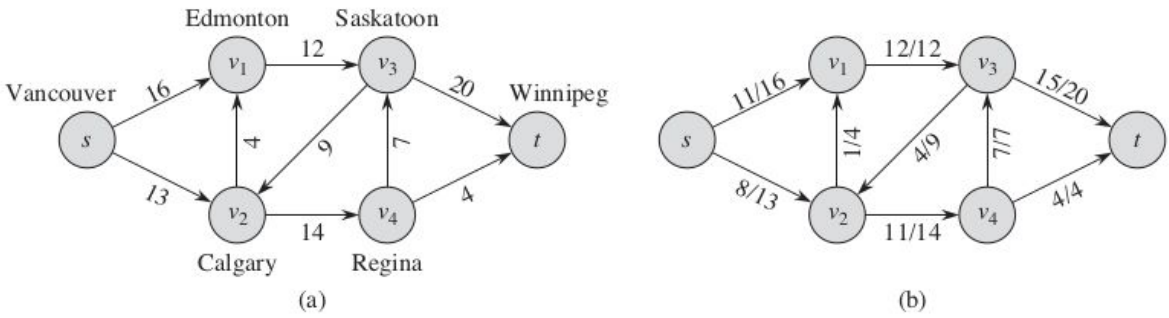


Figure 1 (a)A flow network G = (V, E) for the Lucky Puck Company's trucking problem. The Vancouver factory is the source s, and the Winnipeg warehouse is the sink t. The company ships pucks through intermediate cities, but only c(u, v) crates per day can go from city u to city v. Each edge is labeled with its capacity.
(b) A flow f in G with value | f | = 19. Each edge (u, v) is labeled by f(u, v) / c(u, v). The slash notation merely separates the flow and capacity; it does not indicate division.

Modeling problems with antiparallel edges

In the real-life problems modeling the system as a flow network may introduce antiparallel or reverse edges. But, our definition of a flow network does not allow the existence of an antiparallel edge. To solve the problem we pick one of the two antiparallel edges and split it by adding a vertex in between. For example in the figure 1 a) suppose there is an edge (v1, v2) with cost c(v1, v2) = 10. This edge is antiparallel with edge (v2, v1) with c(v2, v1) = 4. Suppose we take the edge (v1, v2), now we split by adding a new vertex v' and replacing the edge (v1,

v2) with the pair of edges (v1, v') and (v', v2). We set the capacity of both these new edges to c(v1, v2) = 10.

Networks with multiple sources and sinks

A maximum-flow problem may have several sources and sinks, rather than just one of each. The Lucky Puck Company, for example, might actually have a set of m factories {s1, s2, . . . , sm} and a set of n warehouses {t1, t2, . . . , tm}, as shown in figure 2(a). We can reduce this to an ordinary flow network with one source and one sink.

We add a **supersource** s and add a directed edge (s, si) with capacity c(s, si) = ∞ for each i = 1, 2, . . . , m. We also create a new **supersink** t and add a directed edge (ti, t) with capacity c(ti, t) = ∞ for each i = 1, 2, . . . , n. The results are shown in figure 2(b)

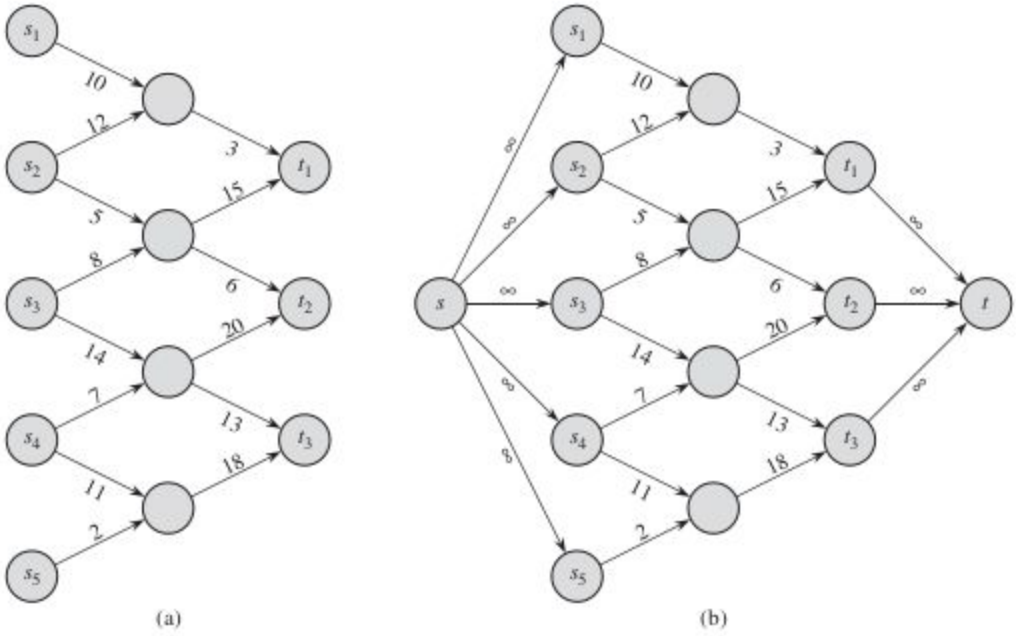


Figure 2

After getting acquainted with the basic requirements of the flow network we will now move on to one of the maximum-flow-algorithms called the Ford-Fulkerson Method.

2. The Ford-Fulkerson Method

The Ford Fulkerson method depends on three important ideas: residual network, augmenting paths, and cuts. The Ford-Fulkerson method iteratively increases the value of the flow. We start with f (u, v) = 0 for all u, v $\in$ V, giving an initial flow of value 0. At each iteration, we increase the flow value in G by finding an "augmenting path" in an associated "residual network" G_f .

Once we know the edges of an augmenting path in G_f, we can easily identify specific edges in G for which we can change the flow so that we increase the value of the flow. Although each iteration of the Ford-Fulkerson method increases the value of the flow, we shall see that the flow on any particular edge of G may increase or decrease. We repeatedly augment the flow until the residual network has no more augmenting paths. The max-flow min-cut theorem will show that upon termination, this process yields a maximum flow.

```
FORD-FULKERSON-METHOD( G, s, t )
1 initialize flow f to 0
2 while there exists an augmenting path p in the residual network G_f
3       augment flow f along p
4 return f
```

In order to implement and analyze the Ford-Fulkerson method, we need to introduce several additional concepts or some tools.

Residual Networks

Intuitively, given a flow network G and a flow f , the residual network G_f consists of edges with capacities that represent how we can change the flow on edges of G. An edge of the flow network can admit an amount of additional flow equal to the edge's capacity minus the flow on that edge. If that value is positive, we place that edge into G_f with a "residual capacity" of $c_f(u, v) = c(u, v) - f(u, v)$. The only edges of G that are in G_f are those that can admit more flow; those edges (u, v) whose flow equals their capacity have $c_f(u, v) = 0$, and they are not in G_f.

As an algorithm manipulates the flow, with the goal of increasing the total flow, it might need to decrease the flow on a particular edge. In order to represent a possible decrease of a positive flow f(u, v) on an edge in G, we place an edge (v, u) into G_f with residual capacity $c_f(v, u) = f(u, v)$ that is, an edge that can admit flow in the opposite direction to (u, v), at most canceling out the flow on (u, v).

More formally suppose we have the flow network G = (E, V) with source s and sink t. Let f be the flow in G, and consider a pair of vertices u, v $\in$ V. We define the **residual capacity** $c_f(u, v)$ by

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E, \\ f(v,u) & \text{if } (v,u) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Because of our assumption that (u, v) ∈ E and (v, u) ∉ E, exactly one case applies to each ordered pair. As an example, if c(u, v) = 16 and f(u, v) = 11, then we can increase $f(u, v)$ by up to $c_f(u, v) = 5$ units before we exceed the capacity constraint on edge (u, v). We also wish to allow an algorithm to return up to 11 units of flow from v to u, and hence $c_f(v, u) = 11$
Given a flow network G = (V, E) and a flow f, the residual network of G induced by f is $G_f = (V, E_f)$, where $E_f = \{(u, v) \in V \times V : c_f(v, u) > 0\}$

That is, as promised above, each edge of the residual network, or residual edge, can admit a flow that is greater than 0. Figure 3(b) shows the corresponding residual network G_f of Figure 3(a).

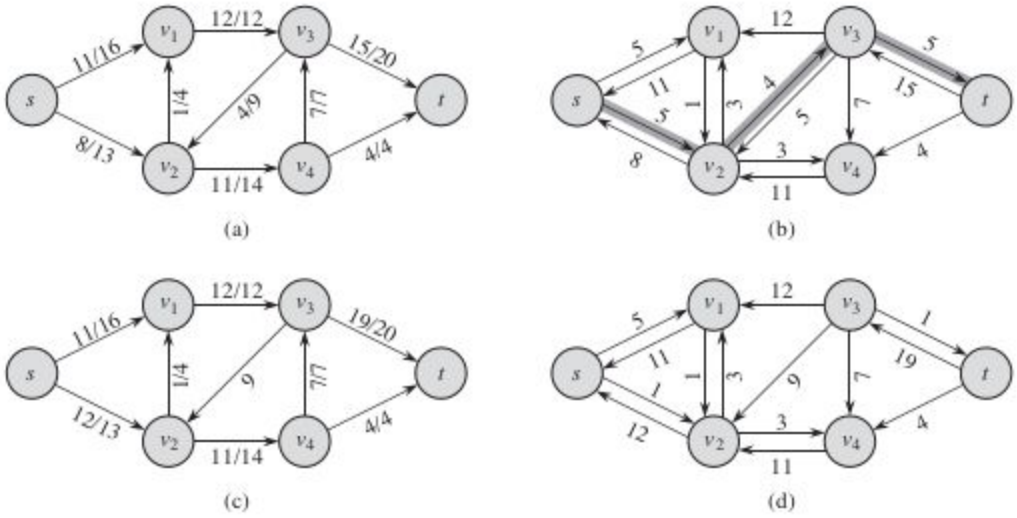


Figure 3

A flow in a residual network provides a roadmap for adding flow to the original flow network. If f is a flow in G and f' is a flow in the corresponding residual network G_f, we define $f \uparrow f'$, the **augmentation** of flow f by f', to be a function from V × V to ℝ, defined by

$$(f \uparrow f')(u,v) = \begin{cases} f(u,v) + f'(u,v) - f'(v,u) & \text{if } (u,v) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

We increase the flow on (u, v) by $f'(u, v)$ but decrease it by $f'(v, u)$ because pushing flow on the reverse edge in the residual network signifies decreasing the flow in the original network.

Lemma 2.1
Let G = (V, E)be a flow network with source s and sink t, and let f be a flow in G. Let G f be the residual network of G induced by f, and let f' be a flow in G_f. Then the function $f \uparrow f'$ is a flow in G with value $|f \uparrow f'| = |f| + |f'|$.

proof

We first verify that $f \uparrow f'$ obeys the capacity constraint for each edge in E and flow conservation at each vertex in $V - \{s, t\}$.
For the capacity constraint, first observe that if $(u, v) \in$ E, then $c_f(u, v) = f(u, v)$. Therefore, we have $f'(u, v) \leq c_f(v, u) = f(u, v)$, and hence

$$(f \uparrow f')(u, v) = f(u, v) + f'(u, v) - f'(v, u)$$
$$\geq f(u, v) + f'(u, v) - f'(u, v) = f'(u, v) \geq 0.$$

In addition,

$$(f \uparrow f')(u, v) = f(u, v) + f'(u, v) - f'(v, u)$$
$$\leq f(u, v) + f'(u, v) \leq f(u, v) + c_f(u, v) = f(u, v) + c(u, v) - f(u, v) = c(u, v)$$

For flow conservation, because both f and f' obey flow conservation, we have that for all u $\in V - \{s, t\}$,

$$\sum_{v \in V} (f \uparrow f')(u, v) = \sum_{v \in V} (f(u, v) + f'(u, v) - f'(v, u))$$
$$= \sum_{v \in V} f(u, v) + \sum_{v \in V} f'(u, v) - \sum_{v \in V} f'(v, u)$$
$$= \sum_{v \in V} f(v, u) + \sum_{v \in V} f'(v, u) - \sum_{v \in V} f'(u, v)$$
$$= \sum_{v \in V} (f(v, u) + f'(v, u) - f'(u, v))$$
$$= \sum_{v \in V} (f \uparrow f')(v, u)$$

Finally, we compute $|f \uparrow f'|$. We define $V_1 = \{v : (s, v) \in E\}$ to be the set of vertices with edges from s, and $V_2 = \{v : (v, s) \in E\}$ to be the set of vertices with edges to s. We have $V_1 \bigcup V_2 \subseteq V$ and, because we disallow antiparallel edges, $V_1 \bigcap V_2 = \phi$ We now compute

$$|f \uparrow f'| = \sum_{v \in V} (f \uparrow f')(s, v) + \sum_{v \in V} (f \uparrow f')(v, s)$$
$$= \sum_{v \in V_1} (f \uparrow f')(s, v) + \sum_{v \in V_2} (f \uparrow f')(v, s)$$

Where the second line follows because $(f \uparrow f')(w, x) = 0$ if (w, x) ∉ E.

$$
\begin{aligned}
|f \uparrow f'| &= \sum_{v \in V} (f \uparrow f')(s, v) + \sum_{v \in V} (f \uparrow f')(v, s) \\
&= \sum_{v \in V_1} (f \uparrow f')(s, v) + \sum_{v \in V_2} (f \uparrow f')(v, s) \\
&= \sum_{v \in V_1} (f(s, v) + f'(s, v) - f'(v, s)) - \sum_{v \in V_2} (f(v, s) + f'(v, s) - f'(s, v)) \\
&= \sum_{v \in V_1} f(s, v) - \sum_{v \in V_2} f(v, s) + \sum_{v \in V_1} f'(s, v) + \sum_{v \in V_2} f'(s, v) - \sum_{v \in V_1} f'(v, s) - \sum_{v \in V_2} f'(v, s) \\
&= \sum_{v \in V_1} f(s, v) - \sum_{v \in V_2} f(v, s) + \sum_{v \in V_1 \cup V_2} f'(s, v) - \sum_{v \in V_1 \cup V_2} f'(v, s) \\
&= \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s) + \sum_{v \in V} f'(s, v) - \sum_{v \in V} f'(v, s) = |f| + |f'|.
\end{aligned}
$$

Augmenting Paths

Given a flow network G = (V, E) with a flow f, an **augmenting path** p is a simple path from s to t in the residual network G_f.By the definition of the residual network, we may increase the flow on an edge (u, v) of an augmenting path by up to $c_f(u, v)$ without violating the capacity constraint on whichever of (u, v) and (v, u) is in the original flow network G.

The shaded path in Figure 3 (b) is an augmenting path. Treating the residual network G_f in the figure as a flow network, we can increase the flow through each edge of this path by up to 4 units without violating a capacity constraint, since the smallest residual capacity on this path is $c_f(v_1, v_2) = 4$. We call the maximum amount by which we can increase the flow on each edge in an augmenting path p the **residual capacity** of p, given by $c_f(p) = min\{c_f(u, v) : (u, v) \text{ is on } p\}$.

Lemma 2.2

Let G = (V, E) be a flow network, let f be a flow in G, and let p be an augmenting path in G_f. Define a function $f_p : V \times V \to \mathbb{R}$ by

$$
f_p(u, v) = \begin{cases} c_f(p) & \text{if } (u, v) \text{ is on } p, \\ 0 & \text{otherwise}. \end{cases}
$$

Then, f_p is a flow in G_f with value $|f_p| = c_f(p) > 0$.

Proof
Suppose for (u, v) ∈ E $f_p(u, v) \neq c_f(p)$. As we know that the flow through a flow network must follow the capacity constraint, i.e For all u, v ∈ V, we require 0 $\leq f(u, v) \leq c(u, v)$. Therefore, let us take $f_p(u, v) \geq c_f(e)$ where e ∈ E, such that $c_f(e) \neq c_f(p)$.

For all (u, v) such that $c_f(u, v) \geq c_f(e)$ we have that 0 $\leq f_p(u, v) \leq c_f(u, v)$.
But there are edges (u, v) in the flow network such that $c_f(u, v) < c_f(e)$ because $c_f(e) \neq c_f(p)$.
So for such edges $f_p(u, v) = c_f(e) \geq c_f(u, v)$ which violates the capacity constraint. Hence our assumption is wrong.

The following corollary shows that if we augment f by f_p, we get another flow whose value is closer to maximum.

Corollary 2.1
Let G = (V, E) be a flow network, let f be a flow in G, and let p be an augmenting path in G_f. Let f_p be defined as above, and suppose that we augment f by f_p . Then the function $f \uparrow f_p$ is a flow in G with value $|f \uparrow f_p|$ = $|f|$ + $|f_p|$ > $|f|$.

Proof
Immediate from Lemma 2.1 and Lemma 2.2.

Cuts of Flow Networks

A **cut** (S, T) of flow network G = (V, E) is a partition of V into S and T = V - S such that s ∈ S and t ∈ T. If f is a flow, then the net flow across the cut (S, T) is defined to be

$$f(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u).$$

The capacity of the cut (S, T) is

$$c(S, T) = \sum_{u \in S} \sum_{v \in T} c(u, v).$$

A **minimum cut** of the network is the cut whose capacity is minimum over all the cuts.

Consider a cut ({s, v_1, v_2}, {v_3, v_4, t}) in the Figure 3(a). The net flow across this cut is

$$\begin{aligned} f(v_1, v_3) + f(v_2, v_4) - f(v_3, v_2) &= 12 + 11 - 4 \\ &= 19, \end{aligned}$$

and the capacity of this cut is

$$\begin{aligned} c(v_1, v_3) + c(v_2, v_4) &= 12 + 14 \\ &= 26. \end{aligned}$$

Lemma 2.3
Let f be a flow in a flow network G with source s and sink t, and let (S, T) be any cut of G. Then the net flow across (S, T) is $f(S, T) = |f|$.

Proof
The proof can be found in the textbook.

Corollary 2.2
The value of f in a flow network G is bounded by the capacity of any cut of G

Proof
Let (S. T) be any cut of G and let f be any flow. By lemma 2.3 and the capacity constraint,
$|f| = f(S, T)$

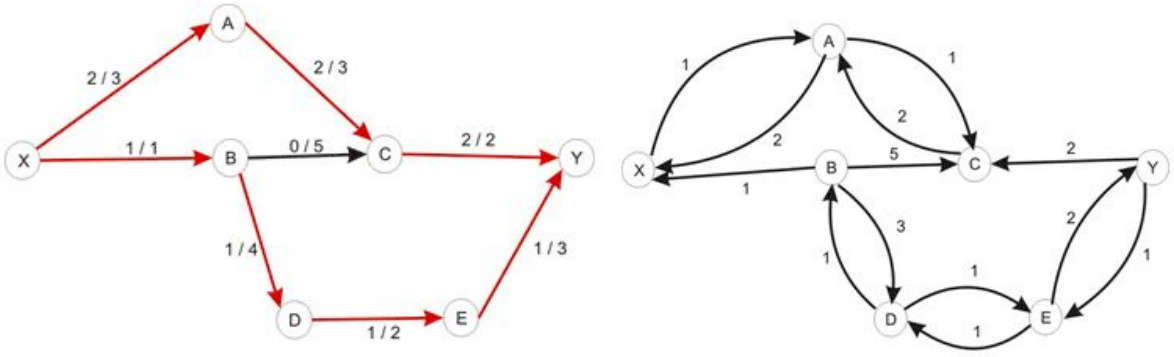
$$= \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u) \leq \sum_{u \in S} \sum_{v \in T} f(u, v) \leq \sum_{u \in S} \sum_{v \in T} c(u, v) = c(S, T).$$

This corollary says that the flow in a network is bounded from above by the capacity of a minimum cut of the network.
Now we will state and prove the most important theorem for the Ford-Fulkerson method, the max-flow min-cut theorem.

Theorem (Max-flow min-cut theorem)
If flow f in a flow network G = (V, E) with source s and sink t, then the following conditions are equivalent:

1. f is a maximum flow in G,
2. The residual network G_f contains no augmenting paths.
3. $|f| = c(S, T)$ for some cut (S, T) of G.

First let us validate the theorem. Consider this example with edges marked with f/c



The graph on the left is a maximum flow network with source X and sink Y and the one on the right is it's residual network. The | f | = 3. Let's see if 1 ⇒ 2, and indeed there is no augmenting path in the residual network, i.e there is no path from source X to sink Y in the residual network.

Check if 2 ⇒ 3, consider the cut S = {X, A, C} and T = {B, D, E, Y}. now c(S, T) = c(X, B) + c(C, Y) = 1 + 2 = 3 = | f | = 3.

Check if 3 ⇒ 1, from corollary 2.2 we know that | f | $\leq$ c(S, T), so if | f | is equal to c(S, T) it means that | f | attains its maximum value, and in this example indeed it does.

Let's prove the theorem formally.

Proof

1 ⇒ 2: Suppose for the sake of contradiction that f is a maximum flow in G but that G_f has an augmenting path p. Then, by Corollary 2.1, the flow found by augmenting f by f_p, is a flow in G with value strictly greater than $|f|$, contradicting the assumption that f is a maximum flow.

2 ⇒ 3: Suppose that G_f has no augmenting path, that is, that G_f contains no path from s to t. Define $S = \{v \in V : \textit{there exists a path from } s \textit{ to } v \textit{ in } G_f\}$ and T = V - S. The partition (S, T) is a cut: we have s $\in$ S trivially and t $\notin$ S because there is no path from s to t in G_f. Now consider a pair of vertices u $\in$ S and v $\in$ T. If (u, v) $\in$ E, we must have $f(u, v) = c(u, v)$, since otherwise (u, v) $\in E_f$, which would place v in set S. If (v, u) $\in$ E, we must have $f(u, v) = 0$, because otherwise $c_f(u, v) = f(v, u)$ would be positive and we would have (u, v) $\in E_f$, which would place v in S. Of course, if neither (u, v) nor (v, u) is in E, then $f(u, v) = f(v, u) = 0$. We thus have

$$f(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u)$$

$$f(S, T) = \sum_{u \in S} \sum_{v \in T} c(u, v) - \sum_{u \in S} \sum_{v \in T} 0$$

$$f(S, T) = c(S, T)$$

By lemma 2.3, $|f| = f(S, T) = c(S, T)$.

3 ⇒ 1: By Corollary 2.2, $|f| \leq c(S, T)$ for all cuts (S, T). The condition $|f| = c(S, T)$ thus implies that f is a maximum flow.

The Ford Fulkerson Algorithm

Ford-Fulkerson (G, s, t)
1 **for** each edge (u, v) $\in$ E
2 f(u, v) = 0
3 **while** there exists a path p from s to t in the residual network G_f
4 $c_f(p) = min\{c_f(u, v) : (u, v) \textit{ is in } p\}$
5 **for** each edge (u, v) in p
6 **if** (u, v) $\in$ E
7 f(u, v) = f(u, v) + $c_f(p)$

```
8               else f(u, v) = f(v, u) - c_f(p)
9               end
10      end
11 end
end function
```

Lines 1–2 initialize the flow f to 0. The while loop of lines 3–8 repeatedly finds an augmenting path p in G_f and augments flow f along p by the residual capacity $c_f(p)$. Each residual edge in path p is either an edge in the original network or the reversal of an edge in the original network. Lines 6–8 update the flow in each case appropriately, adding flow when the residual edge is an original edge and subtracting it otherwise. When no augmenting paths exist, the flow f is a maximum flow.

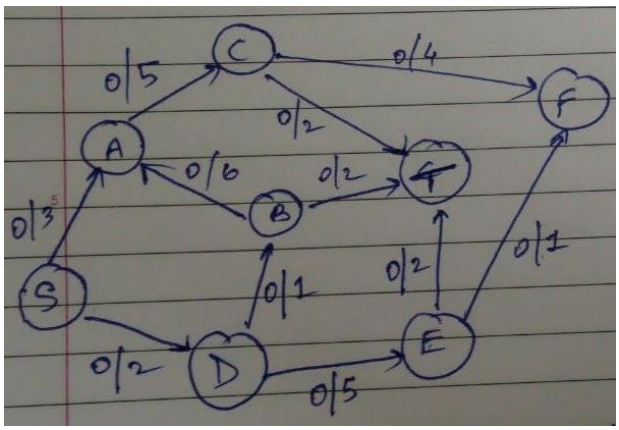
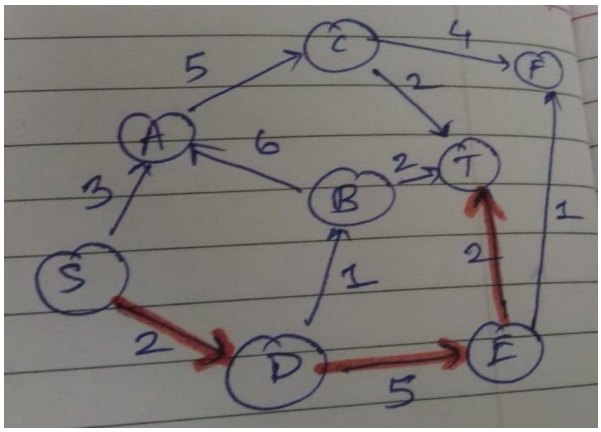
To mention that the augmenting path is found by using bfs. So all the augmenting paths are the shortest paths from s to t. So the time for finding an augmenting path is O(E). That is, each iteration of the while loop will take O(E) time assuming that updating the flow and finding $c_f(p)$ takes constant time. Assignment of f(u, v) = 0 takes O(E) time. Now assuming that each flow augmentation increases the flow by one unit. Let maximum flow be | f* |. Therefore at max there will be | f* | augmentations or while loop iterations. Hence, the total running time for the Ford Fulkerson algorithm will be O(E| f* |).

The book gives another algorithm called the Edmond-Karp algorithm which gives a better bound on the running time of the Ford-Fulkerson Algorithm. It shows that the running time can be bounded by O(E^2V).

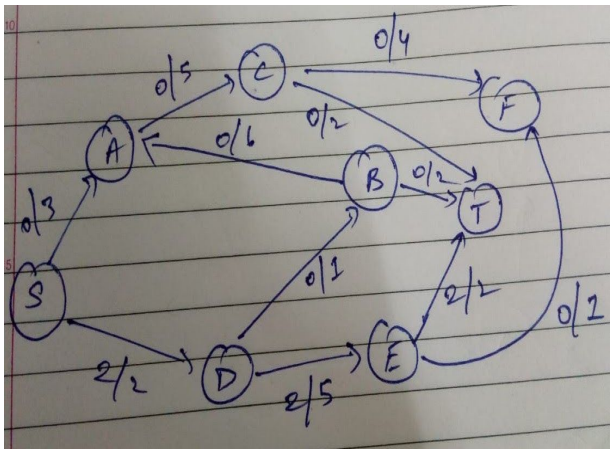
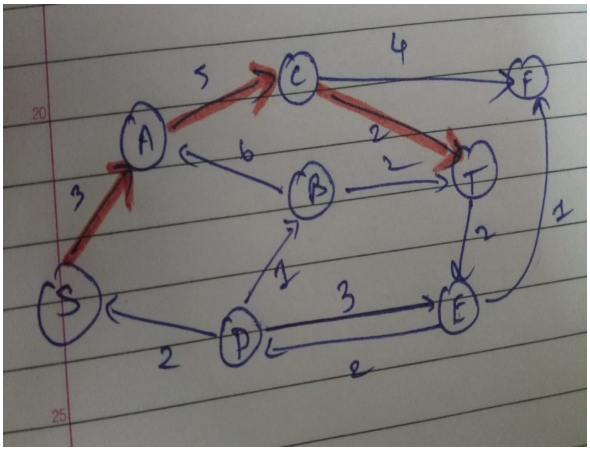
Let us solve the following example using our favoruite Ford-Fulkerson. Let me tell here that the shortest path is found assuming that length of each edge is of one unit. So do not confuse the cost of the edge with the capacity of the edge. The augmenting path chosen is marked with red.

Flow Network Residual Network

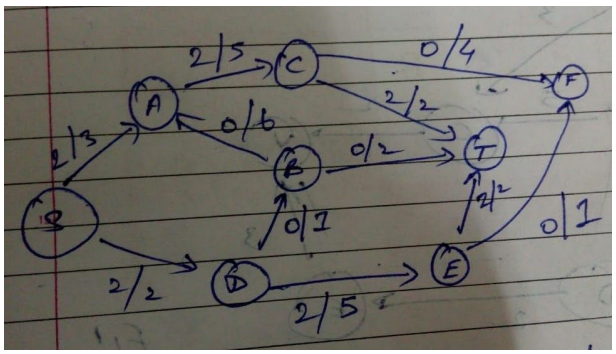
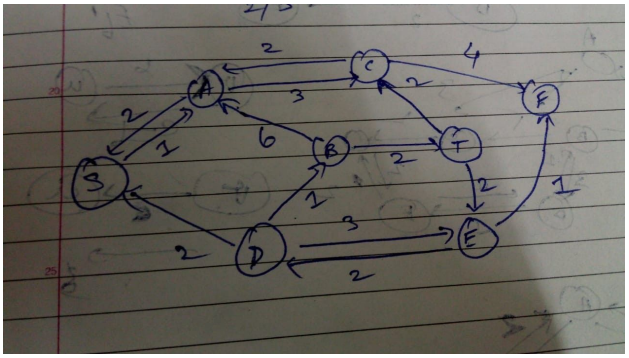
Iteration 1

Iteration 2

Iteration 3

So in the 3rd Iteration, there is no more augmenting path left hence the algorithm halts. The maximum flow calculated by the algorithm is 4 units.

Now we will have a look at another algorithm for finding maximum flow which gives better time bound than the Ford-Fulkerson. The method is called the **push-relabel** algorithm.

3. The Push - relabel Algorithms

The Push-relabel algorithms use similar tools as the Ford-Fulkerson. Rather than examining the entire network at a time push-relabel works one vertex at a time, it takes information only from the neighbor vertices. Most importantly the push-relabel does not maintain the flow conservation property. The push-relabel have a concept of **perflow** instead of a **flow.** The perflow is a function $f : V \times V \rightarrow \mathbb{R}$ that satisfies the capacity constraint and the following relaxed flow conservation:

$$\sum_{v \in V} f(v, u) - \sum_{v \in V} f(u, v) \geq 0$$

For all vertices u ∈ V - { s }. That is the flow into the vertex may exceed the flow out the vertex. Let us call the quantity

$$e(u) = \sum_{v \in V} f(v, u) - \sum_{v \in V} f(u, v)$$

The **excess** flow into the vertex u. We can say that the vertex u ∈ V - { s, t } is **overflowing** if $e(u) > 0$.

Let us understand the push-relabel algorithms in terms of fluid flows. We consider a flow network G = (V, E) to be a system of interconnected pipes of given capacities. Applying this analogy to the Ford-Fulkerson method, we might say that each augmenting path in the network gives rise to an additional stream of fluid, with no branch points, flowing from the source to the sink. The Ford-Fulkerson method iteratively adds more streams of flow until no more can be added.

All the vertices have an attribute called “height”. The height can be thought of as the distance of the vertices from the ground. As we know water can flow on its own only from a higher point to a lower point. So the algorithm can “push” flow along the edge (u, v) if and only if height(u) > height(v). Initially, the height of the source is assigned to |V| and the height of sink to 0 and all other vertices begin from 0 and increase gradually as the algorithm proceeds. Each vertex is assumed to have an infinite reservoir to store the excess flow.

Eventually, we may find that there are some vertices who have unsaturated (flow in the pipe is less than the capacity of the pipe)pipes connecting vertices with the same height or with vertices having height greater than them. We perform “relabeling” on such vertices. We increase its height to one unit more than the height of the lowest of its neighbors to which it has an unsaturated pipe. After a vertex is relabeled, therefore, it has at least one outgoing pipe through which we can push more flow.

The algorithm first sends as much flow as possible downhill from the source toward the sink. The amount it sends is exactly enough to fill each outgoing pipe from the source to capacity; that is, it sends the capacity of the cut (s, V - { s }). When flow first enters an intermediate vertex, it collects in the vertex’s reservoir. From there, we eventually push it downhill.

The basic operation

Let G = (V, E) be a flow network with source s and sink t, and let f be a preflow in G. A function $h : V \to \mathbb{N}$ is a **height function** if $h(s) = |V|$ and $h(t) = 0$, and
$h(u) \leq h(v) + 1$
For every residual edge (u, v) ∈ E_f.

Lemma 3.1
Let G = (V, E) be a flow network, let f be a preflow in G, and let h be a height function on V . For any two vertices u, v ∈ V , if h(u) > h(v) + 1, then (u, v) is not an edge in the residual network.

Proof
The proof lies in the way we increase the height of a vertex u. For the base case when heights are 0 for every edge (u, v) ∈ E_f we have $h(u) \leq h(v) + 1.$
At an intermidiate stage if there are neighbors v with same height as the vertex u then after increasing h(u) = minimum(h(v)) + 1 = h(u) + 1. If the only neighbors are with height greater than then u then we h(u) = minimum(h(v)) + 1. Therefore $h(u) \leq h(v) + 1$.

The push operation
The basic operation PUSH(u, v) applies if u is an overflowing vertex, $c_f(u, v) > 0$ and $h(u) = h(v) + 1$. The pseudocode below updates the preflow f and the excess flows for u and . It assumes that we can compute residual capacity $c_f(u, v)$ in constant time given c and f . We maintain the excess flow stored at a vertex u as the attribute u,e and the height of u as the attribute u.h. The expression $\Delta f(u, v)$ is a temporary variable that stores the amount of flow that we can push from u to v

PUSH(u, v)
1 $\Delta f(u, v) = min(u.e, c_f(u, v))$
2 **if** (u, v) ∈ E
3 $f(u, v) = f(u, v) + \Delta f(u, v)$
4 **else** $f(u, v) = f(u, v) - \Delta f(u, v)$
5 $u.e = u.e - \Delta f(u, v)$
6 $v.e = v.e + \Delta f(u, v)$

Line 1 computes the value $\Delta f(u, v)$, and lines 4–6 update f . Line 5 increases the flow on edge (u, v) because we are pushing flow over a residual edge that is also an original edge. Line 6 decreases the flow on edge (u, v), because the residual edge is actually the reverse of an edge in the original network. Finally, lines 7–8 update the excess flows into vertices u and v. Thus, if f is a preflow before PUSH is called, it remains a preflow afterward.

A push is a **saturating push** if edge (u, v) in the residual network becomes saturated ($c_f(u, v) = 0$ afterward); otherwise, it is a **nonsaturating push**. If an edge becomes saturated, it disappears from the residual network. After a nonsaturating push from u to , the vertex u is no longer overflowing. Since the push was nonsaturating, the amount of flow $\Delta f(u, v)$ actually pushed must equal u.e prior to the push. Since u.e is reduced by this amount, it becomes 0 after the push.

The relabel operation

We can relabel an overflowing vertex u if for every vertex for which there is residual capacity from u to, flow cannot be pushed from u to because v is not downhill from u.

RELABEL(u)
1 //Applies when u is overflowing and for all v ∈ V such that (u, v) ∈ E_f, we have u.h $\leq$ v.h
2 $u.h = 1 + min\{v.h : (u, v) \in E_f\}$

When we call the operation RELABEL(u), we say that vertex u is relabeled. Note that when u is relabeled, E_f must contain at least one edge that leaves u, so that the minimization in the code is over a nonempty set. This property follows from the assumption that u is overflowing, which in turn tells us that

$$u.e = \sum_{v \in V} f(v, u) - \sum_{v \in V} f(u, v) > 0 .$$

Since all flows are nonnegative, we must therefore have at least one vertex v such that $f(u, v) > 0$. But then, $c_f(u, v) > 0$, which implies that (u, v) ∈ E_f.

The generic algorithm

The generic algorithm first initializes parameters in the following way.

```
INITIALIZE-PERFLOW(G, s)
1 for each vertex v in V
2         v.h = 0
3         v.e = 0
4 for each edge (u, v) in E
5         f(u, v) = 0
6 s.h = | V |
7 for each vertex v adjacent to s
8         f(s, v) = c(s, v)
9         v.e = c(s, v)
10        s.e = s.e − c(s, v)
```

After initialization the only edges (u, v) which do not satisfy the condition for height function, i.e u.h $\leq$ v.h + 1 is u = s. But when the algorithm is initialized such edges (s, v) are saturated hence they are not in the residual network.

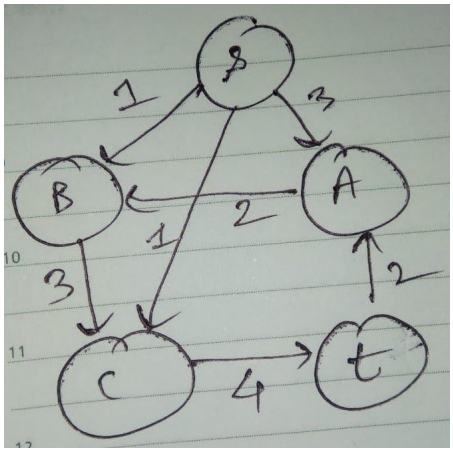
Initialization, followed by a sequence of push and relabel operations, executed in no particular order, yields the GENERIC-PUSH-RELABEL algorithm:

GENERIC-PUSH-RELABEL(G)
1 INITIALIZE-PERFLOW(G, s)
2 **while** there exists an applicable push or relabel operation
3 select an applicable push or relabel operation and perform on it.

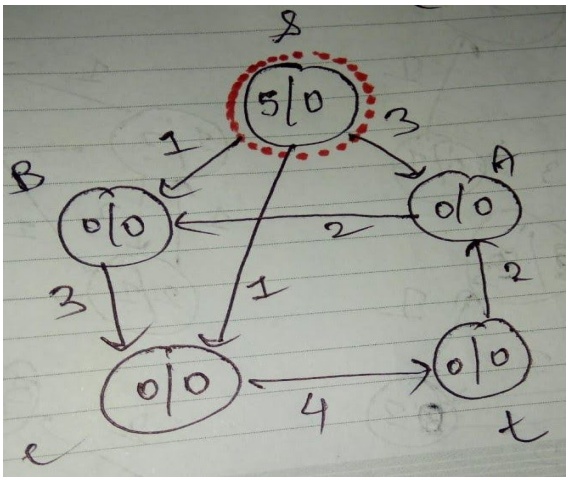
Lemma 3.2
Let G = (V, E) be a flow network with source s and sink t, let f be a preflow, and let h be any height function for f . If u is any overflowing vertex, then either a push or relabel operation applies to it.

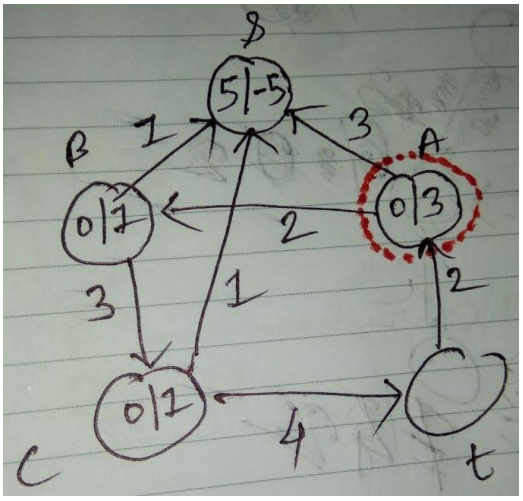
Poof The proof is very simple and can be referred from the textbook

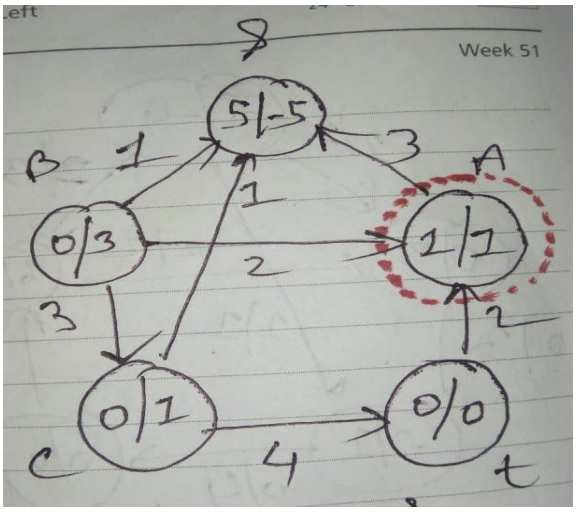
Let us do an example of the push-relabel algorithm. The Graph given below demonstrates a flow network with source s and sink t.

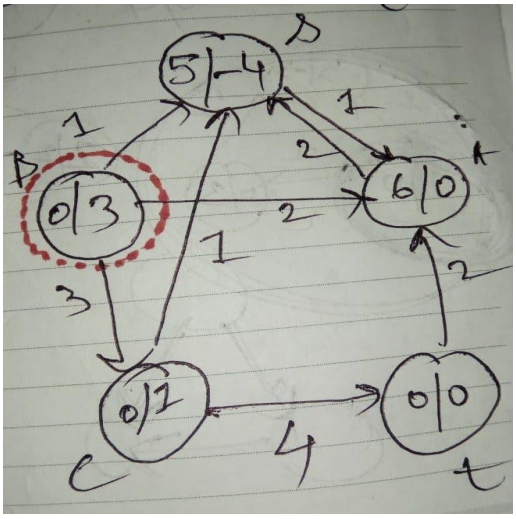


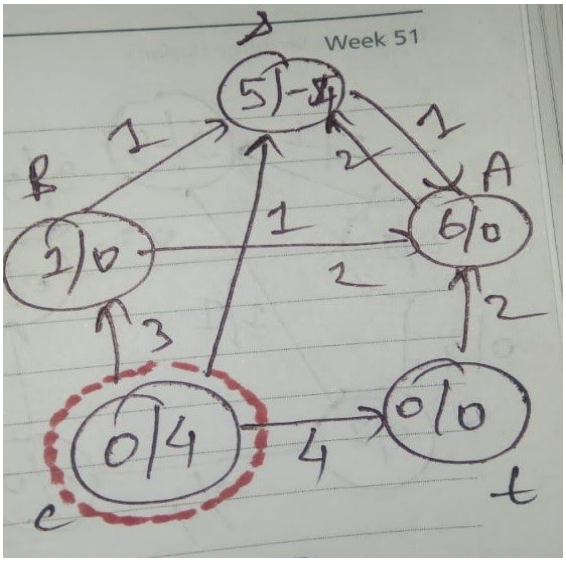
We will use the push relabel algorithm to find the maximum flow in the graph. Each picture is a residual network. Each picture has marked the vertex which the algorithm selects. So the algorithm will work until there is any pushing or relabelling left in the graph or until there is any excess flow in any of the vertices. Each vertex has excess_flow/ vertex_height written inside them. The vertices are ordered alphabetical wise.

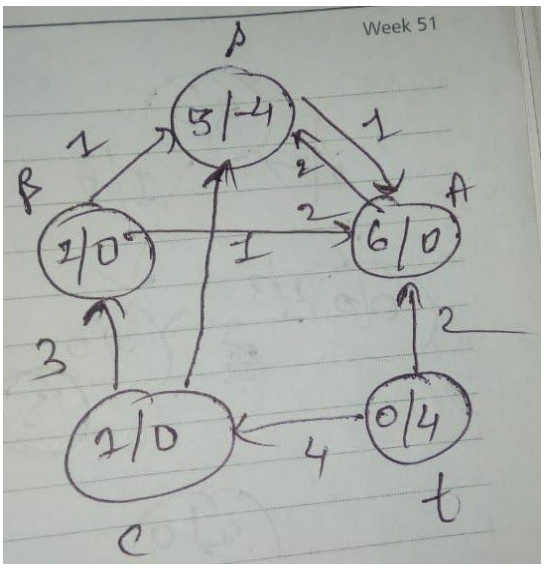
1. 

2. 

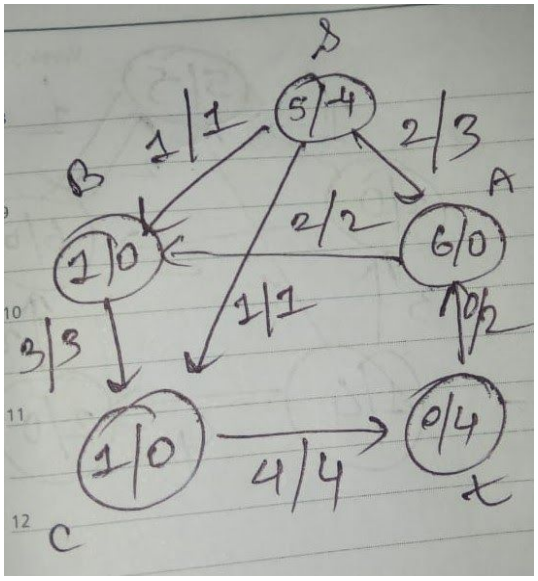
3. 

4. 

5. 

6. 

Since the excess flow in each vertex is 0 therefore the algorithm terminates. The maximum flow is 4. Each vertex is denoted by flow/ capacity.



Analysis of the push-relabel method
To find the running time of the algorithm we have to first look at some results.

Lemma 3.3
Let G = (V, E) be a flow network with source s and sink t, and let f be a preflow in G. Then, for any overflowing vertex x, there is a simple path from x to s in the residual network G_f.

Proof
The proof can refered to from the textbook.

This Lemma can be verified in our example. All six residual network has a simple path from all v to s.

Lemma 3.4
Let G=(V, E) be a flow network with source s and sink t. At any time during the execution of GENERIC-PUSH-RELABEL on G, we have u.h $\leq$2|V| $-$ 1 for all vertices u $\in$ V.

Lemma 3.5
Let G = (V, E) be a flow network with source s and sink t. Then, during the execution of GENERIC -PUSH -RELABEL on G, the number of relabel operations is at most 2|V| $-$ 1 per vertex and at most (2|V| $-$ 1)(|V| $-$ 2)< 2$|V^2|$ overall.

Proofs for both the lemmas are very simple.

Lemma 3.6
During the execution of GENERIC-PUSH-RELABEL on any flow network G = (V, E) the number of saturating pushes is less than 2|V||E|.

The book has given a subtle proof for this lemma given below.

Proof
For any pair of vertices u, v $\in$ V, we will count the saturating pushes from u to v and from v to u together, calling them the saturating pushes between u and v. If there are any such pushes, at least one of (u, v) and (v, u) is actually an edge in E. Now, suppose that a saturating push from u to v has occurred. At that time v.h = u.h - 1. In order for another push from u to v to occur later, the algorithm must first push flow from v to u, which cannot happen until v.h = u.h + 1. Since u.h never decreases, in order for v.h = u.h + 1, the value of v.h must increase by at least 2. Likewise, u.h must increase by at least 2 between saturating pushes from v to u. Heights start at 0 and, by Lemma 3.4, never exceed 2|V| − 1, which implies that the number of times any vertex can have its height increase by 2 is less than |V|. Since at least one of u.h and v.h must increase by 2 between any two saturating pushes between u and, there are fewer than 2|V| saturating pushes between u and v. Multiplying by the number of edges gives a bound of less than 2|V||E| on the total number of saturating pushes.

Lemma 3.7
During the execution of GENERIC-PUSH-RELABEL on any flow network G = (V, E), the number of nonsaturating pushes is less than $4|V|^2(|V| + |E|)$.

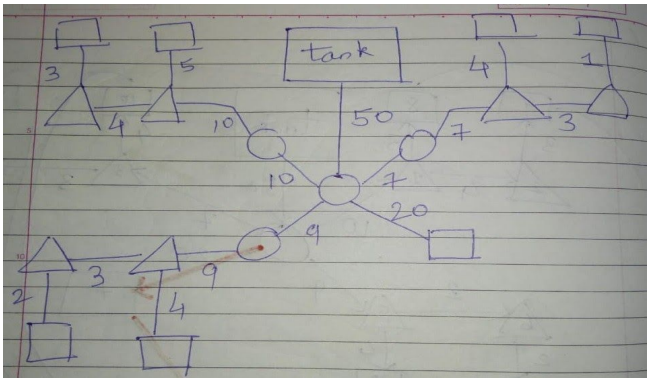
Proof of this lemma is a bit lengthy but very simple.

From lemmas 3.5, 3.6, 3.7 we can say that because the only operations involved are relabel and push, therefore the running time of the GENRIC-PUSH-RELABEL algorithm is

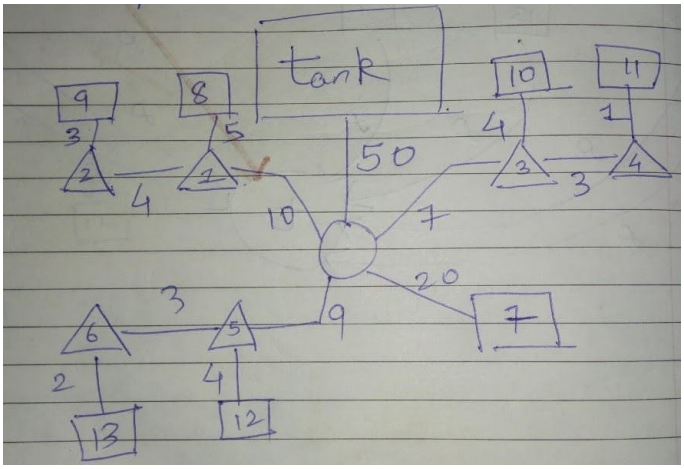
O($2|V|^2$) + O(2|V||E|) + O($4|V|^2$ (|V| + |E|)) = O($|V|^2E$)

Therefore the running time is O($|V|^2E$).

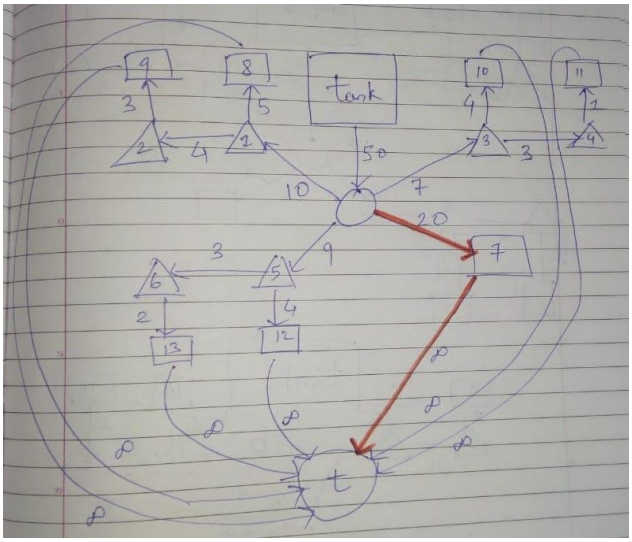
<u>Implementation of the Ford-Fulkerson algorithm.</u>
The Following graph is the real flow network in my house. The capacities are approxmately equal to real capacities.



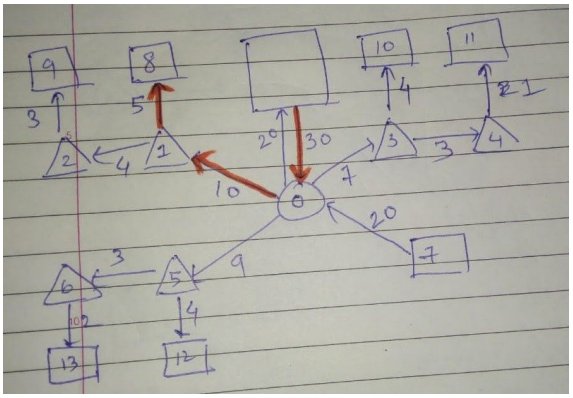
The tank is the source, the circles are the primary control knobs, triangles are secondary control knobs and squares are sinks. Since the circles have the same incoming and outgoing capacities hence the flow network can be reduced to the following network.

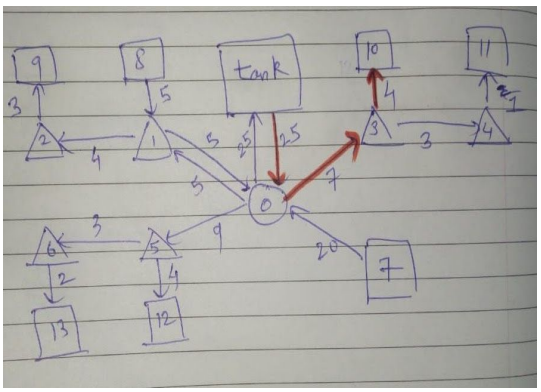


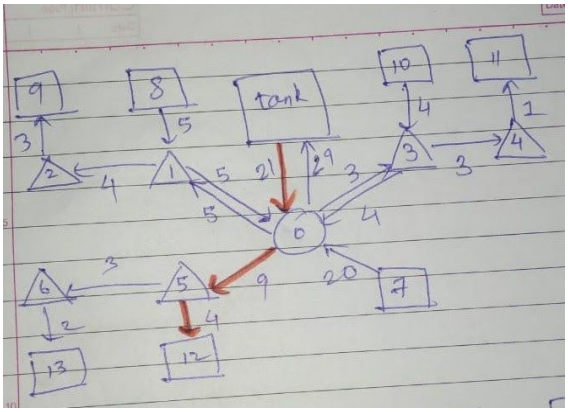
Since the Ford-Fulkerson works with a single source and a single sink, hence all we add one more link with an infinite capacity to a virtual sink.

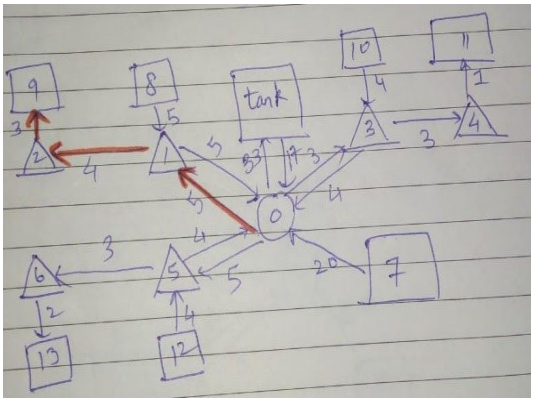


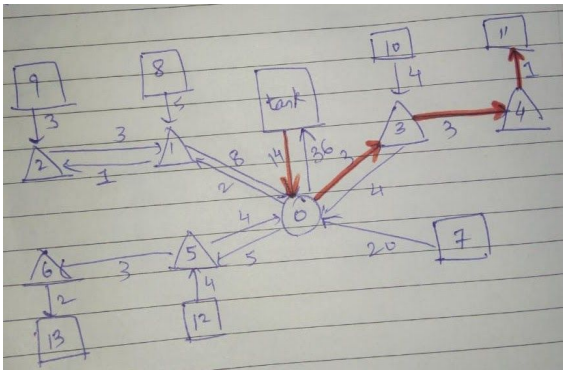
Now the following pictures have the residual networks which are the result of each iteration of the Ford-Fulkerson. And the path selected is marked with red. The above picture is the initial residual network and the path selected is marked with red.

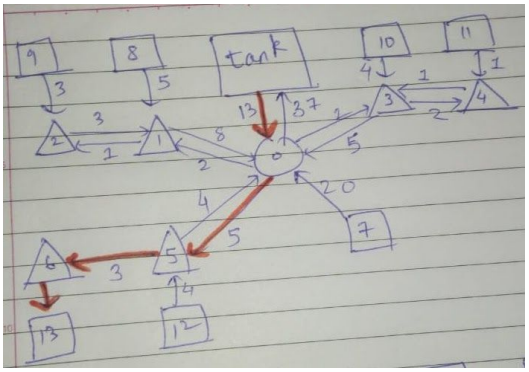
1. 

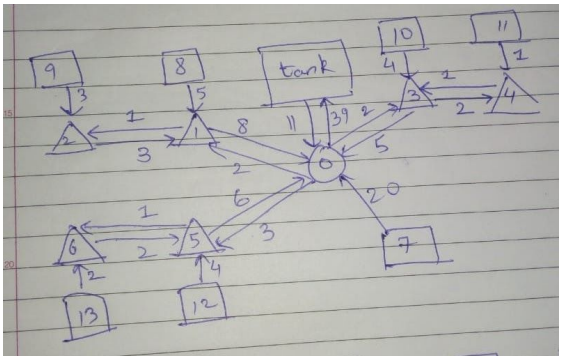
2. 

3. 

4. 

5. 

6. 

7. 

Since there is no more augmenting path left the algorithm halts. Hence, the maximum flow is 39 units.