CSS.201.1 Algorithms

Instructor: Umang Bhaskar

TIFR 2024, Aug-Dec

Scribe: Soham Chatterjee

sohamchatterjee999@gmail.com
Website: sohamch08.github.io

Contents

Chapter 1

Maximum Flow

1.1 Flow

Suppose we are given a directed graph $G = (V, E)$ with a source vertex s and a target vertex t. And additionally for every edge $e \in E$ we are given a number $c_e \in \mathbb{Z}_0$ which is called the capacity of the edge.

Definition 1.1.1: Flow

An $s - t$ flow is a function $f : E \longrightarrow \mathbb{R}_0$ which satisfies the following:

1. $\forall\, e \in E, f(e) \leq c_e$
2. $\forall\, v \in V \setminus \{s, t\}, \sum\limits_{e \in in(v)} f(e) = \sum\limits_{e \in out(v)} f(e)$

Also the value of a flow f is denoted by $|f| \coloneqq \sum\limits_{e \in out(s)} f(e)$.

Before proceeding into the setup and the problem first we will assume some things

Assumption.

- *$in(s) = \emptyset$ i.e. there is no edge into s.*
- *$out(t) = \emptyset$ i.e. there is no edge out of t.*
- *There are no parallel edges*

Lemma 1.1.1

For any flow f, $|f| = \sum\limits_{e \in in(t)} f(e)$

Proof: We have for every edge $e \in E, \exists\, v \in V$ such that $e \in in(v)$ and $\exists\, u \in V$ such that $e \in out(u)$. Hence we get

$$\sum_{e \in E} f(e) = \sum_{v \in V} \sum_{e \in in(v)} f(e) = \sum_{v \in V} \sum_{e \in out(v)} f(e) \implies \sum_{v \in V} \left[\sum_{e \in in(v)} f(e) - \sum_{e \in out(v)} f(e) \right] = 0$$

Now we know $\forall\, v \in V \setminus \{s, t\}$. $\sum\limits_{e \in in(v)} f(e) = \sum\limits_{e \in out(v)} f(e)$. Therefore we get

$$\sum_{v \in V} \left[\sum_{e \in in(v)} f(e) - \sum_{e \in out(v)} f(e) \right] = 0 \implies \sum_{v \in \{s,t\}} \left[\sum_{e \in in(v)} f(e) - \sum_{e \in out(v)} f(e) \right] = 0 \implies \sum_{e \in out(s)} f(e) - \sum_{e \in in(t)} f(e)$$

Hence we have $|f| = \sum\limits_{e \in in(t)} f(e)$. ■

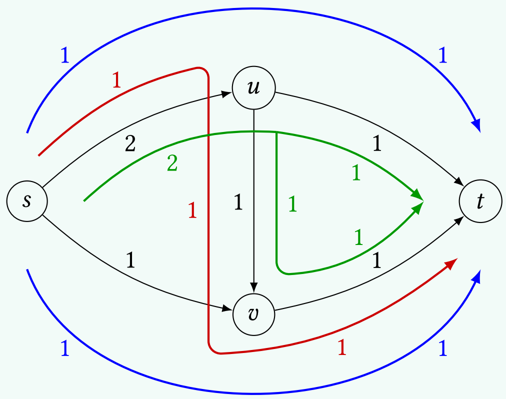
MAX FLOW

Input: A directed graph $G = (V, E)$ with source vertex s and target vertex t and for all edge $e \in E$ capacity of the edge $c_e \in \mathbb{Z}_+$

Question: Given such a graph and its capacities find an $s - t$ flow which has the maximum value

Example 1.1.1

Consider the following directed graph with capacities: $V = \{s, t, u, v\}$, $c_{s,u} = 2, c_{s,v} = c_{u,t} = c_{v,t} = c_{u,v} = 1$. Firstly the following function: $f' : f'(s, u) = 2 = f(u, t)$. It is not a flow since $f(u, t) = 2 > 1 = c_{u,t}$. Now we define three different flow functions:



- f: $f(s, u) = f(u, v) = f(v, t) = 1$ and otherwise 0. Therefore $|f| = 1$
- g: $g(s, u) = g(u, t) = 1$, $g(s, v) = g(v, t) = 1$ and otherwise 0. Therefore $|g| = 2$
- h: $h(s, u) = 2$, $h(u, t) = h(u, v) = h(v, t) = 1$ and otherwise 0. Therefore $|h| = 2$

Notice here g and h has the maximum flow value.

1.2 Ford-Fulkerson Algorithm

Definition 1.2.1: Residual Graph

Given a directed graph $G = (V, E)$ and capacities C_e for all $e \in E$ and an $s - t$ flow f the residual graph $G_f = (V, E_f)$ has the edges with the following properties:

1. If $(u, v) \in E$ and $f(u, v) > 0$ then $(v, u) \in E_f$ and $c^f_{v,u} = f(u, v)$. Such an edge is called a *backward* edge.
2. If $(u, v) \in E$ and $f(u, v) < c_{u,v}$ then $(u, v) \in E_f$ and $c^f_{u,v} = c_{u,v} - f(u, v)$. It is called *forward* edge.

Algorithm 1: FORD-FULKERSON

Input: Directed graph $G = (V, E)$, source s, target t and edge capacities C_e for all $e \in E$
Output: Flow f with maximum value

```
1 begin
2     for e ∈ E do
3         f(e) = 0
4     while ∃ s ⇝ t path P in G_f do
5         δ ⟵ min_{e∈P}{c_e^f} for e = (u, v) ∈ P do
6             if e is Forward Edge then
7                 f(u, v) ⟵ f(u, v) + δ
8             else
9                 f(u, v) ⟵ f(v, u) − δ
```

We call one iteration of the While loop at line 4 *Flow Augmentation*.

Lemma 1.2.1

At any iteration the f' obtained after the flow augmentation of the flow f is a valid flow

Proof: At any iteration let P be the path from $s \rightsquigarrow t$ and $\delta = \min_{e \in P} c_f(e)$. Let f' be the new function such that for each $(u,v) \in P$ if (u,v) is forward edge in G_f then $f'(u,v) = f(u,v) + \delta$ and if (u,v) is backward edge in G_f then $f'(v,u) = f(v,u) - \delta$ and for other edges $e \in E \setminus P$, $f'(e) = f(e)$.

Now since $\delta = \min_{e \in P} c_f(e)$, $c_f(e) \geq \delta$ for all $e \in P$. Hence if (u,v) is backward edge then $(v,u) \in E$ and $c_f(u,v) = f(u,v)$. Hence $f'(v,u) = f(v,u) - \delta \geq 0$. Therefore for all $e \in E$, $f'(e) \geq 0$.

Now first we will show $f'(e) \leq c_e$ for all $e \in E$. If $(u,v) \in P$ is a forward edge then $(u,v) \in E$ and $c_f(u,v) = c_{u,v} f(u,v)$. Therefore $f'(u,v) = f(u,v) + \delta \leq f(u,v) + c_{u,v} - f(u,v) = c_{u,v}$. Now if $(u,v) \in P$ is a backward edge then $(v,u) \in E$ and $c_f(u,v) = f(u,v)$. Therefore $f'(u,v) = f(u,v) - \delta \leq f(u,v) \leq c_{u,v}$. For other edges $e \in E \setminus P$, $f'(e) = f(u) \leq c_e$. Therefore $f'(e) \leq c_e$ for all $e \in E$

Now we will prove for all $v \in V \setminus \{s,t\}$, $\sum_{e \in in(v)} f'(e) = \sum_{e \in out(v)} f'(e)$. If v is not in the path P in G_f then, $f'(e) = f(e)$ for all edges $e \in in(v) \cup out(v)$. Hence the condition is satisfied for such vertices. Suppose v is in the path P. Then there are two edges e_1 and e_2 in P which are incident on e. If both are forward edges or both are backward edges then one of them is in $in(v)$ and other one is in $out(v)$. WLOG suppose $e_1 \in in(v)$ and $e_2 \in out(v)$ we have

$$\sum_{e \in in(v)} f'(e) = \sum_{e \in in(v) \setminus \{e_1\}} f(e) + f(e_1) \pm \delta = \sum_{e \in out(v) \setminus \{e_2\}} f(e) + f(e_2) \pm \delta = \sum_{e \in out(v)} f'(e)$$

If one of e_1, e_2 forward edge and other one is backward edge then either $e_1, e_2 \in in(v)$ (when e_1 is forward and e_2 is backward) or $e_1, e_2 \in out(v)$ (when e_1 is backward and e_2 is forward). Now if $e_1, e_2 \in in(v)$, $f'(e_1) + f'(e_2) = f(e_1) + \delta + f(e_2) - \delta = f(e_1) + f(e_2)$ and if $e_1, e_2 \in out(v)$ then $f'(e_1) + f'(e_2) = f(e_1) - \delta + f'(e_2) + \delta = f(e_1) + f(e_2)$. Hence

$$\sum_{e \in in(v)} f'(e) = \sum_{e \in in(v)} f(e) = \sum_{e \in out(v)} f(e) = \sum_{e \in out(v)} f'(e)$$

Hence f' is a valid flow. ■

Lemma 1.2.2

At any iteration Given G_f if the flow, f' obtained after flow augmentation of f by δ then

$$|f'| = |f| + \delta$$

Proof: Since we augment flow along an $s \rightsquigarrow t$ path, the first edge of the path is always in $out(s)$. Let the first edge is $e = (s,u)$. Now e has to be a forward edge because otherwise $(u,s) \in E$ and then there is an incoming edge in G which is not possible. Hence

$$|f'| = \sum_{e \in out(s)} f'(e) = \sum_{e \in out(s) \setminus \{e\}} f(e) + f'(e) = \sum_{e \in out(s) \setminus \{e\}} f(e) + f(e) + \delta = \sum_{e \in out(s)} f(e) + \delta = |f| + \delta$$

Hence we have the lemma. ■

Lemma 1.2.3

At every iteration of the Ford-Fulkerson Algorithm the flow values and the residual capacities of the residual graph are non-negative integers.

Proof: Initial flow and the residual capacities are non-negative integers. Let till i^{th} iteration the flow values and the residual capacities were non-negative integers. Let the flow after i^{th} iteration was f. Hence $\forall\, e \in E$, $f(e) \in \mathbb{Z}_0$. Therefore in the G_f for all $e \in E_f$, $c_f(e) \in \mathbb{Z}_0$. Hence $\delta \in \mathbb{Z}_0$. Therefore $\forall\, e \in E$, $f'(e) \in \mathbb{Z}_0$. And therefore for all $e \in E_{f'}$ where $G_{f'}$ is the residual graph of the flow f', $c_{f'}(e) \in \mathbb{Z}_0$. Hence by mathematical induction the lemma follows. ■

At any iteration let P be the path from $s \rightsquigarrow t$. Then for all $e \in P$, $c_f(e) > 0$. Therefore $\delta = \min_{e \in P} c_f(e) \geq 1$. Therefore the algorithm must stop in at most $\sum_{e \in out(s)} c_e$ since we can have the value of a flow to be at max the value of the sum of capacities of edges in $out(s)$ and therefore we can increase the flow at max that many times.

Lemma 1.2.4

If f is a max flow then there is no $s \rightsquigarrow t$ path in G_f.

Proof: Suppose there is an $s \rightsquigarrow t$ path P in G_f. We will show that then f is not a max flow following the algorithm. Then $\forall\ e \in P$, $c_f(e) > 0$. Hence $\delta = \min_{e \in P} c_f(e) \geq 1$. Now after the flow augmentation process of f by δ we get a new valid flow f' by Lemma 1.2.1 and by Lemma 1.2.2 we have $|f'| = |f| + \delta > ||f|$. Hence f is not a maximum flow. Hence contradiction. Therefore there is no $s \rightsquigarrow t$ path in G_f. ■

1.2.1 Max Flow Min Cut

Definition 1.2.2: Cut Set

For a graph $G = (V, E)$ and a subset $A \subseteq V$, the cut $(A, V \setminus A)$ is a bipartition of V where the edges E_A of the graph $G_A = (A, V \setminus A, E_A)$ is the set $E_A = E \cap (A \times (V \setminus A))$.

Now if s, t are two vertices of G then an $s - t$ *Cut* $(A, V \setminus A)$ is a cut such that $s \in A$ and $t \in V \setminus A$.

Now we define for a cut $(A, V \setminus A)$ the *Capacity of the Cut* $(A, V \setminus A) = \sum_{e \in E_A} c_e$. For an $s - t$ cut $(A, V \setminus A)$ we denote the capacity of the cut by $cap(A)$ A *Min* $s - t$ *Cut* is a $s - t$ cut of minimum capacity. Then we have the following relation between cut and flow.

Lemma 1.2.5

Given a graph $G = (V, E)$, s, t, $c_e \in \mathbb{Z}_0$ for all $e \in E$ for any flow f and a $s - t$ cut $(A, V \setminus A)$

$$|f| \leq cap(A)$$

Proof: Given f and the $s - t$ cut $(A, V \setminus A)$ we have

$$\begin{aligned}
|f| &= \sum_{e \in out(s)} f(e) \\
&= \sum_{v \in A} \left[\sum_{e \in out(v)} f(e) - \sum_{e \in in(v)} f(e) \right] \\
&= \sum_{\substack{e=(u,v),\\ u \in A, v \notin A}} f(e) - \sum_{\substack{e=(u,v),\\ u \notin A, v \in A}} f(e) && \text{[Edges for both endpoints in } A \text{ are canceled out]} \\
&= \sum_{e \in out(A)} f(e) - \sum_{e \in in(A)} f(e) \\
&\leq \sum_{e \in out(A)} f(e) \leq \sum_{e \in out(A)} c_e = cap(A)
\end{aligned}$$

Hence we have the lemma. ■

Having this lemma we have have for any flow f and $s - t$ cut $(A, V \setminus A)$ we have

$$|f| \leq cap(A) \implies \max_f |f| \leq \min_{s-t \text{ cut } (A, V \setminus A)} cap(A)$$

So we have the following theorem that the value of maximum flow is equal to the capacity of minimum cut.

Theorem 1.2.6 Max Flow Min Cut

Given a graph $G = (V, E)$, s, t, $c_e \in \mathbb{Z}_0$ for all $e \in E$. Then the following are equivalent:

(1) f is a maximum flow.

(2) There is no $s \rightsquigarrow t$ path in G_f

(3) There exists an $s - t$ cut of capacity $|f|$

Proof:

(1) $\implies$ (2): This is by Lemma 1.2.4.

(2) $\implies$ (3): We are given a flow f such that there is no $s \rightsquigarrow t$ path in G_f. We will construct a $s - t$ cut which has the capacity $|f|$. Now take A to be all the vertices reachable from s in G_f. This is a valid $s - t$ cut since $s \in A$ and as there is no $s \rightsquigarrow t$ path in G_f, $t \notin A$. Now

$$|f| = \sum_{e \in out(A)} f(e) - \sum_{e \in in(A)} f(e)$$

Now $\forall\ e = (u, v) \in E$ where $u \in A$ and $v \notin A$ we have $c_{u,v} = f(u, v) \implies c_{u,v} - f(u, v) = 0$ since otherwise $c_{u,v} - f(u, v) \neq 0 \implies c_{u,v} > f(u, v) \implies (u, v) \in E_f$ and therefore v is reachable from s but $v \notin A$, contradiction. Therefore (u, v) is a backward edge and hence . Now $\forall\ e = (u, v) \in E$ where $u \notin A$ and $v \in A$ we have $f(u, v) = 0$ since otherwise $f(u, v) > 0 \implies (v, u) \in E_f$ and therefore u is reachable from s but $u \notin A$, contradiction. Hence we have

$$|f| = \sum_{e \in out(A)} f(e) - \sum_{e \in in(A)} f(e) = \sum_{e \in out(A)} c_e = cap(A)$$

(3) $\implies$ (1): Now by Lemma 1.2.5 we have for any flow f and $s - t$ cut

$$|f| \leq cap(A) \implies \max_f |f| \leq \min_{s-t \text{ cut } (A, V \setminus A)} cap(A)$$

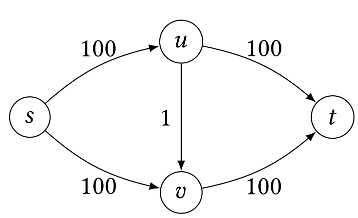
Now given f there exists an $s - t$ cut of capacity $|f|$. Hence f is a max flow.

■

Hence at the end of the Ford-Fulkerson Algorithm let the flow returned by the algorithm is f. The algorithm terminates when there is no $s \rightsquigarrow t$ path in G_f. Hence by Max Flow Min Cut Theorem we have f is a maximum flow. This completes the analysis of the Ford-Fulkerson Algorithm.

Since the capacities of the edges can be very large we want an algorithm return the maximum flow with running time $poly(n, m, \log c_e)$ where n is the number of vertices and m is number of edges and $\log c_e$ basically means number of bits at most needed to represent the capacities.

But Ford-Fulkerson algorithm takes does not run in $poly(n, m, \log c_e)$ instead $poly(n, m, c_e)$ as the while loop in the algorithm takes $poly(c_e)$ many iterations. For example in the following graph: it takes around 100 steps



and in general Ford-Fulkerson takes $O(|f_{\max}|)$ time. For this reason we will now discuss a modification of the Ford-Fulkerson Algorithm which takes $poly(n, m, \log c_e)$ time, Edmonds-Karp Algorithm.

1.2.2 Edmonds-Karp Algorithm

To get a $poly(n, m, \log c_e)$ time algorithm we will always pick the shortest $s \rightsquigarrow t$ path in the residual graph. This algorithm is known as the Edmonds-Karp Algorithm

Suppose f_i be the total flow after i^{th} iteration. And G_{f_i} be the residual graph with respect f_i. Then $f_0(e) = 0$ for all $e \in E$ and $G_{f_0} = G$. Also suppose $dist_i(v) =$ Shortest $s \rightsquigarrow v$ path distance in the residual graph G_{f_i}. Hence $dist_i(s) = 0$ for all i and $dist_i(t) = \infty$ at the end of the algorithm.

Note:-

In i^{th} iteration of the Ford-Fulkerson Algorithm or Edmonds-Karp Algorithm if P is the $s \rightsquigarrow t$ in the residual graph G_{f_i} where $e = (u, v) \in P$ and $c_{f_i}(u, v) = \delta = \min_{e \in P} c_{f_i}(e)$ then the edge (u, v) is not present in the next residual graph $G_{f_{i+1}}$. Thus at least one edge disappears in each iteration of Ford-Fulkerson or Edmonds-Karp Algorithm.

Now we will prove following two lemmas which will help us to prove that the Edmond-Karp algorithm takes $O(mn)$ iterations.

Lemma 1.2.7

At any iteration i, $\forall\, v \in V$, $dist_i(v) \leq dist_{i+1}(v)$

Proof: Suppose this is not true. Then let i be the first iteration in which there exists a vertex $v \in V$ such that $dist_i(v) > dist_{i+1}(v)$. We pick such v which minimizes $dist_{i+1}(v)$. Consider the shortest path P from $s \rightsquigarrow v$ in $G_{f_{i+1}}$. Hence length of P, $|P| = dist_{i+1}(v)$. Let (u, v) be the last edge of P.

$$P: \quad s \rightsquigarrow u \longrightarrow v$$

Then

$$dist_{i+1}(v) = dist_{i+1}(u) + 1 \geq dist_i(u) + 1$$

Here the last inequality follows because v is the vertex which has the minimum $dist_{i+1}(v)$ among all the vertices $w \in V$ which follows $dist_i(w) > dist_{i+1}(w)$. Now we will analyze case wise.

- **Case 1:** $(u, v) \in E_{f_i}$. Then
$$dist_i(v) \leq dist_i(u) + 1 \leq dist_{i+1}(v)$$
But this is not possible since $dist_i(v) > dist_{i+1}(v)$.

- **Case 2:** $(u, v) \notin E_{f_i}$. Then $(v, u) \in E_{f_i}$. Since $(u, v) \in E_{f_{i+1}}$ then we must have sent flow along (v, u). Since we take the shortest $s \rightsquigarrow t$ path in G_{f_i} in the algorithm we have $dist_i(u) = dist_i(v) + 1$. But then
$$dist_i(u) \leq dist_{i+1}(v) - 1 \implies dist_{i+1}(v) \geq dist_i(v) + 2$$
But this is not possible.

Hence contradiction ↯ Therefore for all iterations i, for all vertices $v \in V$, $dist_i(v) \leq dist_{i+1}(v)$. ■

Lemma 1.2.8

For any edge $e = (u, v) \in E$ the number of iterations where either (u, v) appears or (v, u) appears is at most $O(n)$ i.e.

$$\left|\left\{i \colon (u, v) \notin G_{f_i}, (u, v) \in G_{f_{i+1}}\right\}\right| + \left|\left\{i \colon (v, u) \notin G_{f_i}, (v, u) \in G_{f_{i+1}}\right\}\right| = O(n)$$

Proof: Following the proof of Lemma 1.2.7 in the second case we showed if $(u, v) \notin G_{f_i}$ but $(u, v) \in G_{f_{i+1}}$ then $dist_{i+1}(v) \geq dist_i(v) + 2$. Hence the distance increases by at least 2. Now this can happen at most $O(n)$ many times since $\forall\, i$, $dist_i(v) \leq n - 1$. Hence the number of iterations where either (u, v) appears or (v, u) appears is at most $O(n)$. ■

With this this lemma we will prove that the Edmonds-Karp Algorithm takes $O(mn)$ iterations.

Theorem 1.2.9

Edmonds-Karp Algorithm terminates in $O(mn)$ many iterations.

Proof: For k iterations at least k edges must disappear. Since each edge can reappear $O(n)$ times by Lemma 1.2.8, it can disappear at most $O(n)$ many times. In each iteration at least one edge disappears. Now after $O(mn)$ iterations number of disappearances is at most $O(mn)$. But after $O(mn)$ many disappearances there are no edge remaining and therefore there is no $s \rightsquigarrow t$ path. Hence the algorithm terminates. Therefore the Algorithm terminates in $O(mn)$ iterations. ∎

Hence Edmond-Karp Algorithm takes $O(m^2n)poly(\log c_e) = O\left(m^2 n \log^{O(1)}(c_e)\right)$ time since it takes $O(mn)$ iterations and in each iteration it finds the shortest $s \rightsquigarrow t$ path in G_{f_i} in $O(m)$ time and in each iteration it does addition and subtraction and finds minimum of the capacities which takes polynomial of the bits needed to represent them time.

1.3 Preflow-Push/Push-Relabel Algorithm

In this algorithm we will maintain something called "Preflow" which is not a valid flow. Unlike Ford-Fulkerson, Edmonds-Karp it does not maintain a $s \rightsquigarrow t$ path in the residual graph and the algorithm stops when the preflow is actually a valid flow.

Definition 1.3.1: Preflow

Given a graph $G = (V, E)$ and the edge capacities c_e, a function $f : E \longrightarrow \mathbb{R}_0$ is a preflow if is satisfies:

1. $\forall\, e \in E,\ f(e) \le c_e$.
2. $\forall\, v \in V \setminus \{s\},\ \sum\limits_{e in(v)} f(e) \ge \sum\limits_{e \in out(v)} f(e)$

Notice here unlike the definition of Flow here in the second criteria we need $\sum\limits_{e in(v)} f(e) \ge \sum\limits_{e \in out(v)} f(e)$ instead of $\sum\limits_{e in(v)} f(e) = \sum\limits_{e \in out(v)} f(e)$.

N3121ow define for all $v \in V$ and for all preflow f, $excess_f(v) = \sum\limits_{e \in in(v)} f(e) - \sum\limits_{e \in out(v)} f(e)$. If f is a preflow then $excess_f(s) \le 0$ and $\forall\, v \in V \setminus \{s\}$, $excess_f(v) \ge 0$

Lemma 1.3.1

For all preflow f

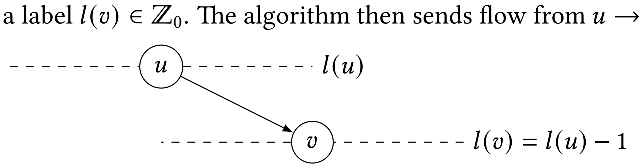
$$\sum_{v \in V} excess_f(v) = 0$$

Proof:

$$\begin{aligned}
\sum_{v \in V} excess_f(v) &= \sum_{v \in V}\left[\sum_{e \in in(v)} f(e) - \sum_{e \in out(v)} f(e)\right] \\
&= \sum_{v \in V}\sum_{e \in in(v)} f(e) - \sum_{v \in V}\sum_{e \in out(v)} f(e) \\
&= \sum_{e \in E} f(e) - \sum_{e \in E} f(e) = 0
\end{aligned}$$

∎

Now for each $v \in V$ we assign a label $l(v) \in \mathbb{Z}_0$. The algorithm then sends flow from $u \longrightarrow v$ if $l(v) = l(u) - 1$.



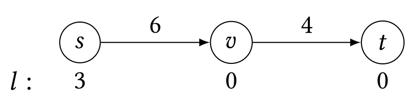
Algorithm 2: PREFLOW-PUSH

Input: Directed graph $G = (V, E)$, source s, target t and edge capacities C_e for all $e \in E$
Output: Flow f with maximum value

1 **begin**
2 Initially $\forall\ e = (s, u) \in E$, $f(e) = c_e$ and $f(e)$ for all other edges.
3 $l(s) \longleftarrow n$
4 **for** $v \in V \setminus \{s\}$ **do**
5 $l(v) \longleftarrow 0$
6 **while** $\exists\ v \neq t$, $excess_f(v) > 0$ **do**
7 **if** $\exists\ u$, *such that* $(v, u) \in E_f$ *and* $l(u) = l(v) - 1$ **then**
8 $\delta \longleftarrow \min\left\{excess_f(v), c_f(v, u)\right\}$
9 **if** (v, u) *is Forward Edge* **then**
10 $f(v, u) \longleftarrow f(v, u) + \delta$
11 **else**
12 $f(u, v) \longleftarrow f(u, v) - \delta$
13 **else**
14 $l(v) \longleftarrow l(v) + 1$ //Relabeling

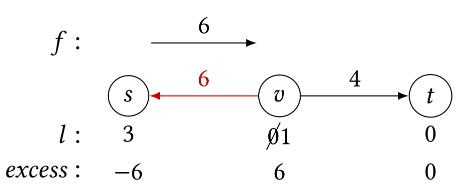
In the algorithm in line 8 if $\delta = c_f(v, u)$ then we call it *saturating push* and if $\delta = excess_f(v)$ then we call it *non-saturating push.*

Now we will show an example of how the algorithm on a graph. We will start the algorithm with the following graph:



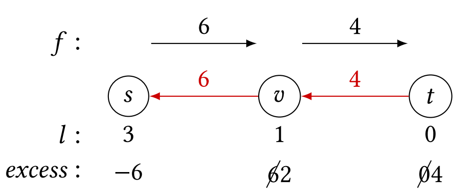
Below we will show change of the residual graph and preflow in each iteration of the WHILE loop:

- Step 1:



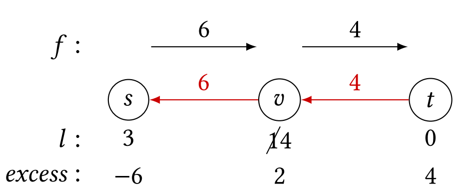
Since $excess_f f(v) = 6 > 0$. So in first iteration v is taken. Since there is no edge (v, u) with $l(u) = l(v) - 1$, label of v got increased

- Step 2:



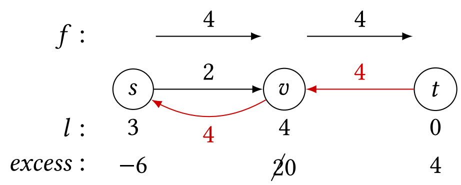
Since $excess_f f(v) = 2 > 0$, in second iteration again v is selected. There is an edge (v, t) with $l(t) = 0 = l(v) - 1 = 1 - 1$. Now $\delta = c_f(v, t) = 4$. Hence saturating push. The preflow gets updated, $f(s, v) = 6$, $f(v, t) = 4$.

- Step 5:



Since $excess_f f(v) = 2 > 0$, in next 3 iterations again v is selected. Since there is no edge (v, u) with $l(u) = l(v) - 1$, label of v gets increased every time. Which becomes 4 after 3 iterations.

- Step 6:



Since $excess_f f(v) = 2 > 0$, in this iteration again v is selected. There is an edge (v,s) with $l(t) = 3 = l(v) - 1 = 4 - 1$. Now $\delta = excess_f(v,s) = 2$. Hence it's non-saturating push. So the preflow gets updated $f(s,v) = 6 - 2 = 4$, $f(v,t) = 4$. Now it's a valid flow. Now there is no vertex with postive *excess*. Hence the algorithm stops.

Observation 1. *Labels are monotone non-decreasing.*

Observation 2. *For every iteration f is always a preflow. The proof is similar to Lemma 1.2.1 but use inequalities.*

Observation 3. $\sum\limits_{v \in V} excess_f(v) = 0$ *and* $\forall\, v \in V \setminus \{s\}$, $excess_f(v) \geq 0$. *Hence* $excess_f(s) \leq 0 \implies l(s)$ *is unchanged.*

Now suppose f^i denote the preflow after the i^{th} iteration of the algorithm. Then

$$f^0(e) = \begin{cases} c_e & \text{when } e = (s,u) \\ 0 & \text{otherwise} \end{cases}$$

Now we will show the correctness of the algorithm.

Lemma 1.3.2

$\forall\, v \in V, \forall\, i$, $excess_{f^i}(v) > 0 \implies \exists\, v \rightsquigarrow s$ in G_{f^i}

Proof: First we fix v and i such that $excess_{f^i} > 0$. Let X be the set of vertices reachable from v in G_{f^i}. Now

$$\sum_{u \in X} excess_{f^i}(u) = \sum_{u \in X}\left[\sum_{e \in in(v)} f^i(e) - \sum_{e \in out(v)} f^i(e)\right] = \sum_{e \in in(X)} f^i(e) - \sum_{e \in out(X)} f^i(e)$$

Now if $\sum\limits_{e \in in(X)} f^i(e) > 0$ then $\exists\, e = (u', u) \in E$ such that $u' \notin X$ and $u \in X$ and $f^i(e) > 0$. Then the backward edge $(u, u') \in E_{f^i}$. Then u' is reachable from v in G_{f^i}. But $u' \notin X$. Contradiction ↯ Therefore $\sum\limits_{e \in in(X)} f^i(e) = 0$. Hence

$$\sum_{u \in X} excess_{f^i}(u) = \cancel{\sum_{e \in in(X)} f^i(e)} - \sum_{e \in out(X)} f^i(e) \leq 0$$

But from Observation 3 we have $\forall\, w \in V \setminus \{s\}$, $excess_{f^i}(w) \geq 0$. But at the same time $\sum\limits_{u \in X} excess_{f^i}(u) \leq 0$ and $excess_{f^i}(v) > 0$. Hence $\exists$ a vertex $u \in X$ such that $excess_{f^i}(u) < 0$. But we know only vertex with negative *excess* is s. Therefore $s \in X$. Hence s is reachable from v. ■

Lemma 1.3.3

$\forall\, i$, if $(u,v) \in G_{f^i}$ then $l(v) \geq l(u) - 1$.

Proof: We will prove this using induction on i. Initially $l(s) = n$ and $l(v) = 0$ for all $v \in V \setminus \{s\}$. Hence for all edges (u,v) where $u, v \neq s$ this is satisfied. All the other edges incident on s are in $in(s)$ in the residual graph. And $l(s) = n \geq l(u) = 0$. Therefore the base case is followed.

Now suppose the condition is true for f^{i-1}. Now in the i^{th} iteration suppose the selected vertex is $v \in V \setminus \{t\}$ with $excess_{f^{i-1}} > 0$. Now there are two possible cases.

- **Case 1:** If the step is relabeling then $f^{i-1} = f^i$, $G_{f^{i-1}} = G_{f^i}$ but v is relabeled by $l(v) + 1$. Now for any edge $e = (u,v) \in in(v)$ by Inductive Hypothesis $l(v) \geq l(u) - 1 \implies l(v) + 1 \geq l(u) - 1$. Now consider any edge $e = (v,w) \in out(v)$. By Inductive Hypothesis we have $l(w) \geq l(v) - 1$. Now if $l(w) = l(v) - 1$ then we would have pushed flow along the edge (v,w). Since that is not the case we have $l(w) > l(v) - 1$. Therefore $l(w) \geq (l(v) + 1) - 1$. Hence the condition is satisfied.

- **Case 2:** If the step is pushing flow then suppose we push flow along the edge $(v, w) \in E_{f^{i-1}}$ and $l(w) = l(v) - 1$. Now if we push flow along the edge (v, w) we might introduce the reverse edge (w, v) in G_{f^i}. In that case $l(v) = l(w) + 1 \geq l(w) - 1$. Hence the condition is satisfied.

Therefore by mathematical induction $\forall\, i, \forall\, (u, v) \in E_{f^i}, l(v) \geq l(u) - 1$. ■

Corollary 1.3.4

There is no $s \rightsquigarrow t$ path in G_{f^i} in any iteration i. Thus when the algorithm terminates f is a max flow.

Proof: Now $l(s) = n$ and $l(t) = 0$. We fix v and i. If there is a $s \rightsquigarrow v$ path in G_{f^i} then length of the path is at most $n - 1$. For each edge in the path the label decreases by at most 1 by Lemma 1.3.3. Hence $l(v) \geq 1$. Therefore for every vertex $v \in V$, reachable from s we have $l(v) \geq 1$. But $l(t) = 0$. Hence t is not reachable from s. Hence if the algorithm terminates, band if f is a valid flow then y Max Flow Min Cut Theorem it is a max flow. ■

Corollary 1.3.5

$\forall\, v \in V, \forall\, i, l(v) \leq 2n$.

Proof: Suppose $\exists\, v, i$ such that $l(v) = 2n$ and $excess_{f^i}(v) > 0$. By Lemma 1.3.2 there exists an $v \rightsquigarrow s$ path in G_{f^i}. Now by Lemma 1.3.3 for each edge in the path the label decreases by at most 1 and the length of the path is at most $n - 1$. Since $l(v) = 2n$, $l(s) \geq n + 1$. But we know $l(s)$ for all i by Observation 3. Hence contradiction ↯ Therefore for all $v \in V$ and $\forall\, i$, $l(v) \leq 2n$. ■

Corollary 1.3.6

Total number relabeling operations is $\leq 2n^2$

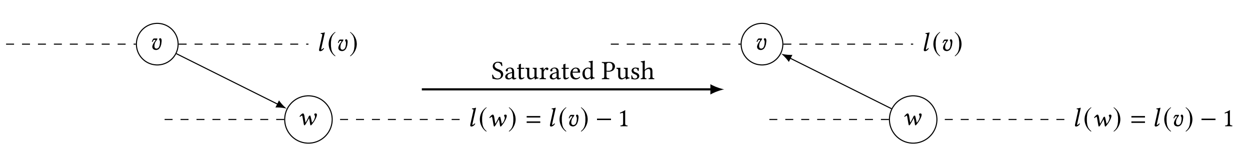
Proof: By Corollary 1.3.5 each vertex label can be at most $2n$. So total number of relabeling operations done in the algorithm is at most $2n^2$ ■

Now we need a bound on the number of push operations. We will count separately the number of Saturating Pushes and number of Non-Saturating Pushes.

Lemma 1.3.7

Total number of saturating pushes is $\leq 2mn$

Proof: We first fix an edge (v, w). Now we will count the number of saturating pushes along (v, w). Then $\delta = c_f(v, w)$. Now consider the scenario of two consecutive saturating pushes along (v, w). When the first saturating push along (v, w) occurred we have $l(w) = l(v) - 1$. Now if (v, w) is forward edge then $\delta = c_f(v, w) = c_{v,w} - f(v, w)$. Then new flow along (v, w) is $f(v, w) + c_f(v, w) = c_{v,w}$. Hence the edge (v, w) vanishes and the flow along (w, v) is $c_{v,w}$. If (v, w) is a backward edge then $\delta = c_f(w, v) = f(w, v)$. Hence then new flow along (w, v) is $f(w, v) - \delta = 0$. Hence again the (w, v) edge vanishes and the flow along (w, v) is $f(w, v)$.



Therefore after a saturated push along (v, w) the edge vanishes and the (w, v) edge is there. Hence in order for another push along (v, w) the algorithm must push flow along (w, v). And this happens when we have the new labels of

v, w follow the condition $l'(w) = l'(v) + 1$. Since by Observation 1 the labels never decreases in order for $l(w) = l(v) + 1$ the label of v must increase by at least 2.

Now starting from $l(v) = 0$ we have by Lemma 1.3.5 $l(v) \leq 2n$ and for each saturating push along (v, w) the $l(v)$ increase by 2. Hence at most n many saturating pushes occurred along (v, w). Now in the original graph since there are m edges the total number of saturating pushes is $\leq 2mn$. ■

Now we will count the number of non-saturating pushes. For such pushes along any edge (v, u) the $excess_f(v)$ goes to 0. We define the potential function for a preflow f,

$$\phi(f) = \sum_{v:\ excess_f(v)>0} l(v)$$

Now $\phi(f) \geq 0$ for all preflow f and initially at the start of the algorithm $\phi(f^0) = 0$.

Observation 4. *For relabeling operation $\phi(f)$ increases by* 1.

Since there are at most $2n^2$ relabeling operations by Corollary 1.3.6, $\phi(f)$ increases by at most $2n^2$ with relabeling operations.

Observation 5. *For each saturating push $excess_f(v, w)$ might not go to* 0 *and therefore ϕ might increase.*

Now by Lemma 1.3.7 total number of saturated pushes is at most $2mn$. And by Corollary 1.3.5 each vertex has label at most $2n$. Hence in total $\phi(f)$ can increase at most $2mn \times 2n = 4mn^2$ by saturated pushes.

Lemma 1.3.8

For each non-saturating push $\phi(f)$ decreases by at least 1.

Proof: Suppose at any iteration i a non-saturating push occur along an edge (v, w). Therefore $l(w) = l(v) - 1$. We will show that $\phi(f^i) \leq \phi(f^{i-1}) - 1$. We have $\delta = excess_{f^{i-1}}(v)$. Now if (v, w) is a forward edge then new flow along (v, w) is $f^i(v, w) = f^{i-1}(v, w) + excess_{f^{i-1}}(v)$. Since $(v, w) \in out(v)$

$$excess_{f^i}(v) = \sum_{e\in in(v)} f^i(e) - \sum_{e\in out(v)} f^i(e) = \sum_{e\in in(v)} f^{i-1}(e) - \sum_{e\in out(v)\setminus\{(v,w)\}} f^{i-1}(e) - f^i(v, w) = excess_{f^{i-1}}(v) - \delta = 0$$

Otherwise if (v, w) is a backward edge. Then ew flow along (w, v) is $f^i(w, v) = f^{i-1}(w, v) - excess_{f^{i-1}}(v)$. Since $(w, v) \in in(v)$

$$excess_{f^i}(v) = \sum_{e\in in(v)} f^i(e) - \sum_{e\in out(v)} f^i(e) = f^i(w, v) + \sum_{e\in in(v)\setminus\{(w,v)\}} f^{i-1}(e) - \sum_{e\in out(v)} f^{i-1}(e) = -\delta + excess_{f^{i-1}}(v) = 0$$

In both cases $excess_{f^i}(v) = 0$. Therefore v goes out of the summation. Now there are two cases depending on the value of $excess_{f^{i-1}}(w)$

- **Case 1:** If $excess_{f^{i-1}}(w) > 0$ i.e. w had excess flow before push operation then $\phi(f^{i-1})$ decreases by $l(v)$ i.e. $\phi(f^i) = \phi(f^{i-1}) - l(v)$. Since $l(w) = l(v) - 1$ and by Observation 1 $l(v) \geq 1$. Therefore $\phi(f^i) = \phi(f^{i-1}) - l(v) \leq \phi(f^{i-1}) - 1$.
- **Case 2:** If $excess_{f^{i-1}}(w) = 0$, then $excess_{f^i}(w) = excess_{f^{i-1}}(w) + \delta > 0$ since $\delta = excess_{f^{i-1}}(v) > 0$ and therefore $\phi(f^i) = \phi(f^{i-1}) - l(v) + l(w) = \phi(f^{i-1}) - 1$

Hence for both the cases $\phi(f^i) \leq \phi(f^{i-1}) - 1$. Therefore $\phi(f^{i-1})$ decreases by at least 1. ■

Lemma 1.3.9

$\phi(f)$ can increase at most

Now

$$\#\text{Non-saturating Pushes} \leq \text{Total decrease in } \phi \leq \text{Total increase in } \phi \leq 2n^2 + 4mn^2 = O(mn^2)$$