Push Relabel

We have studied some algorithms to find the maximum flow in a flow network. These algorithms are *augmenting path* algorithms, meaning they maintain a valid flow and augment it iteratively to increase its value.

Here, we consider a new approach for the maximum flow problem, which is known as the push-relabel paradigm. Unlike the augmenting path algorithms, where we maintain validity and progress towards optimality, in this approach, we try to **maintain an optimal flow, and progress towards making the flow valid**.

Let's use the water-pipes analogy for a flow network. Then, the maximum flow problem asks for the maximum amount of water that can be pushed from the source to the sink. Let us allow junctions (vertices) to be able to store (any amount of) excess incoming water. Then, we start by pushing water from the source, then continue pushing water from vertices with excess water to their neighbors, until we reach a state where no vertex has excess water. Observe that at an intermediate stage of the algorithm, some vertices may hold excess water, so the flow is not valid. But since we terminate when no vertex has excess water, we end up with a valid flow.

We also make sure to maintain optimality throughout the algorithm. In this case, by optimality, we mean that there is no path from the source to the sink in the residual graph. Recall that by the max-flow min-cut theorem, if we have a valid flow, then this condition ensures that the flow is a maximum flow.

Some Definitions

Let us define some terms that we will use in the push-relabel algorithm.

Define a **preflow** as follows:

A **preflow** in $G = (V, E)$ is a real valued function $f : E \to \mathbb{R}$ that satisfies the following properties:

- **Capacity constraint**: For all edges $(u, v) \in E$, we have $0 \leq f(u, v) \leq c(u, v)$.
- **Relaxed flow conservation**: For all vertices $v \in V - \{s, t\}$, we have

$$\sum_{(u,v)\in E} f(u, v) \geq \sum_{(v,w)\in E} f(v, w)$$

  i.e. the total flow entering $v \geq$ the total flow exiting v.

Any valid flow is also a preflow.

For a preflow f in a flow network $G = (V, E)$, we define the **excess flow** α_f at a vertex $v \in V - \{s, t\}$ as the difference between the total flow entering v and the total flow exiting v:

$$\alpha_f(v) = \sum_{(u,v)\in E} f(u, v) - \sum_{(v,w)\in E} f(v, w)$$

Lemma 1: For a preflow f, if $\alpha_f(v) = 0$ for all vertices $v \in V - \{s, t\}$, then f is a valid flow.

In our algorithm, we will maintain a preflow f and perform some operations on f as long as there are vertices with excess flow. Let us define these operations.

PUSH

Define the PUSH procedure on an edge (v, w) as follows:

Given a preflow f, consider G_f. Let (v, w) be a residual edge in G_f with residual capacity $c_f(v, w)$. If $\alpha_f(v) > 0$, we can PUSH Δ flow along the edge (v, w), where

$$\Delta = \min\{\alpha_f(v), c_f(v, w)\}$$

We call a push operation to be a **saturating** push, if $\Delta = c_f(v, w)$ $(c_f(v, w) \leq \alpha(v))$. Otherwise, we call it a **non-saturating** push $(\Delta = \alpha_f(v))$. After a saturating push, the edge (v, w) gets saturated.

A push operation along a residual edge (v, w) can be called a **push** on vertex v.

After pushing Δ flow along (v, w), we update the preflow f as follows:
- If $(v, w) \in E$: $f(v, w) \leftarrow f(v, w) + \Delta$.
- If $(w, v) \in E$: $f(w, v) \leftarrow f(w, v) - \Delta$.

Update the excess flow α_f as follows:

$$\alpha_f(v) = \alpha_f(v) - \Delta, \quad \alpha_f(w) = \alpha_f(w) + \Delta$$

Lemma 2: If we perform a non-saturating push on vertex v, then after the push, the excess flow on v, $\alpha_f(v) = 0$.

Height function

On a flow network $G = (V, E)$ and a preflow f with residual graph G_f, we maintain a **height function** $h : V \to \mathbb{Z}_{\geq 0}$ that satisfies the following properties (invariants):

- $h(s) = n$
- $h(t) = 0$
- For all residual edges $(v, w) \in E_f$, $h(v) \leq h(w) + 1$.

The height function is called so, because intuitively, we may imagine each vertex to sit on a platform at some height. As the algorithm progresses, the height of the vertex increases gradually. These height values determine how the flow is pushed: We push flow only downhill. We fix the height of the source to n and the sink to 0, and then push flow from the source to its neighbours, and so on. When flow first enters a vertex, it is stored in its reservoir (as excess flow). From there, we eventually push the excess flow downhill.

Note that the third invariant says that an edge in the residual graph can point from v to w only if $h(v) \leq h(w) + 1$, or $h(w) \geq h(v) - 1$. So, edges in the residual graph from some vertex v can only point to vertices that are at least as high as one less than the height of v.

In other words, any residual edge (v, w) can either point uphill $(h(w) > h(v))$, to the same level $(h(w) = h(v))$, or exactly one level downhill $(h(w) = h(v) - 1)$. No edge can point downhill where the difference in heights is more than 1.

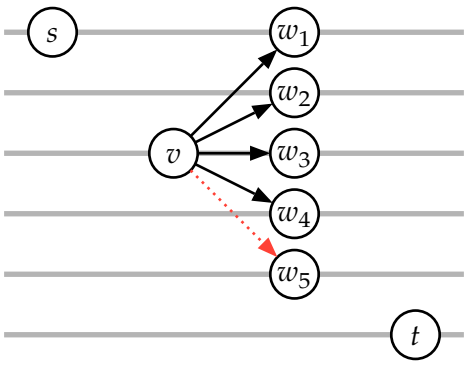

Figure 1: Depiction of vertices of a network at their respective heights. Any edge in the residual network may not point to a vertex more than one level below the current level.

Lemma 3: For a preflow f with residual graph G_f, if h is a valid height function, then there is no path from s to t in G_f.

Proof: Suppose there is a path p from s to t in G_f: $p = \langle s = v_0, v_1, v_2, ..., v_k = t \rangle$. For each edge (v_i, v_{i+1}), we have $h(v_i) - h(v_{i+1}) \leq 1$. Adding this over all i, we get

$$\begin{aligned} h(v_0) - h(v_k) &\leq k \\ \Rightarrow h(s) - h(t) &\leq k \\ \Rightarrow k &\geq n \end{aligned}$$

This is a contradiction, as any path from s to t can have at most $n - 1$ edges. Hence, there is no path from s to t in G_f. ■

Thus, by maintaining a valid height function throughout the algorithm, we maintain our optimality condition, i.e. there does not exist any $s - t$ path in the residual graph.

RELABEL

We may at some point have a vertex v with excess water in its reservoir, but no outgoing residual edges which go downhill. Then, to rid v of its excess flow, we increase the height of v; we call this operation RELABEL. Here, we simply increase the height by 1 (the operation may be needed to be called multiple times until we have some downhill residual edge).

Define the RELABEL procedure on a vertex v as follows:

- Increment the height of v by 1: $h(v) \leftarrow h(v) + 1$.

The Push-Relabel algorithm

In the algorithm, we maintain a preflow f and a valid height function h. At each iteration we perform a push operation, or a relabel operation on some vertex v.

Initialization

What preflow can we initialize the algorithm with? We could think about taking the zero flow as the initial preflow. Clearly, it is a valid flow and thus a valid preflow. But then, it turns out, we cannot define any corresponding height function, because f does not follow our optimality invariant, which we necessarily need to maintain throughout the algorithm.

Instead, we initialize our algorithm with the following parameters:

Define the initial preflow f as follows:

- For all $v \in V$, if $(s, v) \in E, f(s, v) = c(s, v)$. (Saturate all edges leaving the source.)
- For all $(u, v) \in E, u \neq s, f(u, v) = 0$. (Give zero flow to all other edges.)

This is a valid preflow, since on any non-source vertex, we have the total flow exiting the vertex $= 0$, and the total flow entering the vertex ≥ 0.

The excess flow α_f for the above preflow is:
- For all $v \in V$, if $(s, v) \in E, \alpha_f(v) = c(s, v)$. Else, $\alpha_f(v) = 0$.

Define the initial height function h as follows:
- $h(s) = n$.
- For all $v \in V - s, h(v) = 0$.

This height function is valid, since the only problematic residual edges (pointing downhill) can be (s, v) for some $v \in V - s$, but these edges don't exist in G_f since they are saturated in f.

Main loop

PUSH-RELABEL(G)

```
1  initialize preflow f and height function h
2  while there exists a vertex with excess flow α_f(v) > 0
3      choose such a vertex v with maximum height h(v)
4      if there exists an outgoing edge (v, w) such that h(v) = h(w) + 1
5          PUSH Δ = min(α_f(v), c_f(v, w)) flow along (v, w)
6      else
7          RELABEL v
```

Here's an intuitive explanation of what we are doing in the algorithm:

We initially start with the source node at a height n, and every other node at height 0. Also, we have already pushed flow in all the outgoing edges of s to saturation. So, each neighbour v of s holds excess flow equal to the capacity of the edge (s, v). All (s, v) edges are saturated, so in the residual graph, these edges don't exist. Instead, there are (v, s) edges (representing the ability to push back flow to the source, to cancel the some flow in the edge (s, v)).

Iteratively, we choose the highest vertex v having excess flow.
- If it has an outgoing downhill edge (v, w) in the residual graph, then we **PUSH** flow through that edge.
  - After pushing this flow, the reverse edge (w, v) gains residual capacity. (If it was not already in G_f, it now is.)
  - If this was a **saturating push**, then (v, w) gets saturated. It no longer is in G_f.
  - If this was a **non-saturating push**, then all the excess flow at v gets moved to w. The excess flow at v is 0 now. We will not choose this vertex in the next iteration.
- Otherwise, we **RELABEL** (increase its height), till one of its outgoing edges becomes downhill. (v definitely has *some* outgoing edge. (Why?))

Eventually, all flow that can possibly get through to the sink will arrive. Because of the capacity constraints, flow is restricted by cut capacities. At this point, if there are still any vertices with excess flow, the algorithm will send back all the excess flow to the source. In this case, the intermediate vertices will have been RELABEL-ed to be at a higher level than the source itself.

Correctness

Here, we need to show that we always maintain a valid preflow, and we always keep the three invariants related to the height function.

The former is easy to show; in a relabel operation there are no flow value changes, so it is irrelevant. In a push operation, $\alpha_f(v)$ can reduce by $\Delta \leq \alpha_f(v)$. So, after this operation, $\alpha_f(v) \geq 0$, thus, f is a valid preflow.

For the height function, we never relabel s or t, so their heights remain unchanged. For other vertices,

- When we call RELABEL(v), height of v increases by 1.
  - Before this call, say there was some $(u, v) \in E_f$, $h(u) \leq h(v) + 1$. Then after this call, this inequality is not broken.
  - Before this call, say there was some $(v, w) \in E_f$, $h(v) \leq h(w) + 1$. But, we know that $h(v) \neq h(w) + 1$, otherwise a PUSH operation would have occured. So, before the call, $h(v) < h(w) + 1$. Now, after the increment, we will have $h'(v) \leq h(w) + 1$, retaining the invariant.
- When we call PUSH(v, w), we have $h(v) = h(w) + 1$.
  - After the push, we might introduce a new edge (w, v) in the residual graph. We have, $h(w) = h(v) - 1 \Rightarrow h(w) \leq h(v) + 1$. So the invariant holds for this new edge.
  - For all other edges, the invariant is retained since no heights were changed.

Thus throughout our algorithm, we maintain a valid preflow, and satisfy our invariants regarding the height function.

Termination

Here, we show that the algorithm must terminate, and also show that the final result is optimal.

> **Theorem 1 (Termination of the Push-Relabel algorithm)**
>
> PUSH-RELABEL terminates with a maximum flow f.

Theorem 1.1: If PUSH-RELABEL terminates, then the final preflow f is a maximum flow.

Proof: When the algorithm terminates, we have a preflow f and a valid height function h. At termination, there does not exist any vertex v with excess flow $\alpha_f(v) > 0$. Thus, f is a valid flow (Lemma 1). Furthermore, in G_f, there does not exist any path from s to t (Lemma 3). Thus, by the max-flow min-cut theorem, f is a maximum flow.

Theorem 1.2: PUSH-RELABEL terminates after at most $O(n^2)$ RELABEL operations and $O(n^3)$ PUSH operations.

We will prove this theorem by proving some intermediate results first.

Lemma 4: For any vertex v such that $\alpha_f(v) > 0$, there exists a path from v to s in G_f.

(Intuitively, this must be true, because if v has excess flow, it must have gotten it from s, so it must be able to route it back to s in G_f somehow.)

Proof: Let $A = \{v \in V \mid \text{there exists a path from } v \rightsquigarrow s \text{ in } G_f\}$, and $B = V \setminus A$.

For some vertex $v \in V$,

$$\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) \leq 0$$

by definition of a preflow; where $\delta^+(v)$ denotes the outgoing edges of v and $\delta^-(v)$ denotes the incoming edges to v.

Summing this over all vertices $v \in B$, we get

$$\sum_{v \in B} \left(\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) \right) \leq 0$$
$$\sum_{v \in B} \sum_{e \in \delta^+(v)} f(e) - \sum_{v \in B} \sum_{e \in \delta^-(v)} f(e) \leq 0$$

The LHS of this inequality is the difference between the flows of edges outgoing from some vertex in B, and edges incoming to some vertex in B. Note that terms corresponding to the edges which are both incoming and outgoing to vertices in B cancel out in this sum. So we are only left with edges going from A to B, or from B to A.

$$= \sum_{\substack{v \in B \\ w \in A \\ (v,w) \in E}} f(v,w) - \sum_{\substack{u \in A \\ v \in B \\ (u,v) \in E}} f(u,v)$$

Here, we claim that each term in the second summation must be zero. Suppose it isn't: There is some $(u,v) \in E, u \in A, v \in B$, and $f(u,v) > 0$. Then, the residual edge (v,u) has positive residual capacity, so it exists in G_f. Also, $u \rightsquigarrow s$ is a path in G_f (since $u \in A$). But then, $v \to u \rightsquigarrow s$ is also a path in G_f, so v is connected to s, thus $v \in A$, which is a contradiction. So the second summation results to 0.

This leaves the LHS to just be

$$\sum_{\substack{v \in B \\ w \in A \\ (v,w) \in E}} f(v,w) \geq 0$$

Since we have shown the LHS to be both ≥ 0 and ≤ 0, the only possibility is that it is equal to 0. In the original summation (over $v \in B$), each term of the summation was ≤ 0. Since their sum is zero, all of them must be individually 0. So, for all $v \in B$,

$$\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = 0$$
$$\Rightarrow \alpha_f(v) = 0$$

So, we can say, that if a vertex v has $\alpha_f(v) > 0$, it must necessarily belong to A, and thus there must exist a v to s path in G_f. ■

Lemma 5: For any vertex $v \in V$, $h(v) \leq 2n$.

Proof: We only relabel a vertex when it has excess flow $\alpha_f(v) > 0$. By Lemma 4, at this point there must exist a $v \rightsquigarrow s$ path. Let this path be $p = \langle v = v_0, v_1, v_2, ..., v_k = s \rangle$. For each edge (v_i, v_{i+1}), we have $h(v_i) - h(v_{i+1}) \leq 1$. Adding this over all $k \leq n-1$ edges, we get

$$h(v_0) - h(v_k) \leq k$$
$$\Rightarrow h(v) - h(s) \leq k$$
$$\Rightarrow h(v) \leq n + k \leq 2n - 1$$

Upon performing the relabel operation, the height increases by 1. So, after any relabel operation, the height of any vertex can be at most $h(v) \leq 2n$. ■

Corollary of Lemma 5: The total number of RELABEL operations in PUSH-RELABEL is at most $(n-2) \times 2n = O(n^2)$. ■

Lemma 6: Between any two saturating pushes along the edge (v, w), both v and w have been relabeled at least twice.

Proof: Let us follow the sequence of operations needed for two consecutive saturated pushes on the same edge (v, w).

1. For a saturating push to occur, we require $h(v) = h(w) + 1$. After this push, the edge (v, w) will become saturated, so it will no longer exist in G_f.
2. To set up another saturating push across (v, w) we need to first add the (v, w) edge to G_f by performing a push along the reverse edge (w, v). For this push to occur, we need $h(w) = h(v) + 1$. This requires (at minimum) two RELABELs to w to increase its height so that it can push flow down to v.
3. Once we have the edge (v, w) in G_f, we can potentially perform the push along this edge. But we need to have $h(v) = h(w) + 1$ for the push to happen. Again, this requires (at minimum) two RELABELs to v to increase its height more than w so that it can push flow down to it.

Thus, we require two relabel operations on both v and w between two consecutive saturating push operations along an edge (v, w). ■

Corollary of Lemma 6: The total number of saturating pushes in PUSH-RELABEL is $O(mn)$.

Proof: Since the total number of relabel operations on a vertex is bounded by $O(n)$ (Lemma 5), and between two consecutive saturated pushes on a edge $(v, w) \in E_f$, v must be relabelled at least twice; so each edge can have at most $O(n)$ saturating pushes. Therefore, the total number of saturating pushes is $O(mn)$. ■

Lemma 7: Between any two consecutive relabel operations, there can be at most n non-saturating pushes.

Proof: Consider the sequence of operations between any two consecutive RELABEL operations. In this duration, all height values are constant since RELABEL is never called. Say, we have a non-saturating push along (v, w). This means, v is the max-height vertex with excess flow, and it pushes all its excess flow to w (since it is a non-saturating push). Now, v does not have any excess flow.

After this, in this duration, can there ever be a push on vertex v? For it to happen, there must be some excess flow on v, for which some edge (u, v) must have pushed flow to v. If so, then u will have had a larger height than v, and also have had excess flow. But, this is a contradiction, since flow strictly flows downhill, and originally v was the max-height vertex with excess flow; then excess flow cannot reach any vertex higher than v.

Thus, each node can have at most one non-saturating push in the duration. Hence, there can be at most n non-saturating push between two consecutive relabel operations, there can be at most n non-saturating pushes. ■

Corollary of Lemma 7: The total number of non-saturating pushes in PUSH-RELABEL is $n \times O(n^2) = O(n^3)$. ■

Proof of Theorem 1.2: Follows from the corollaries of Lemmas 5, 6 and 7.

Thus we have shown that the algorithm must terminate with a maximum flow, and it will terminate after performing at most $O(n^3)$ pushes and $O(n^2)$ relabels.

Worked Example

Let us work out the steps of this algorithm by running it on a simple flow network:

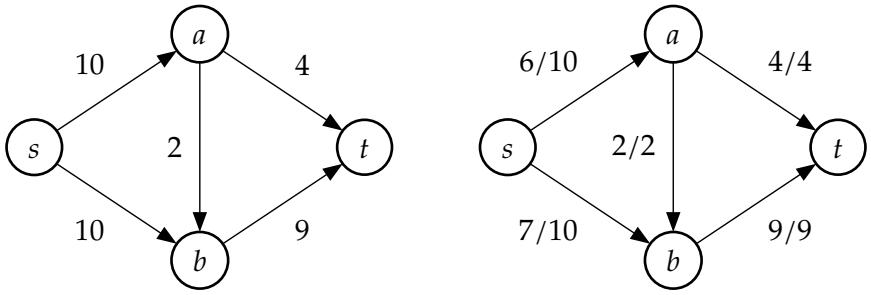

Figure 2: (a) A flow network G. (b) A maximum flow f in G, found by the Push Relabel algorithm.

Figure 3 shows the working out of the algorithm on the flow network G. Here, $n = 4$, so we begin with $h(s) = 4$ and $h(a) = h(b) = h(t) = 0$, having saturated both edges leaving s. This leaves $\alpha_f(a) = 10$ and $\alpha_f(b) = 10$; every other vertex has zero excess. This is the state shown in the first panel below.

Both a and b are tied for the highest vertex with excess flow, so the algorithm may pick either; say it picks a. Since $h(a) = 0$, no outgoing residual edge of a can be downhill, so we RELABEL a once, setting $h(a) \leftarrow 1$. Now (a, b) is a residual downhill edge, so we PUSH $\Delta = \min\big(\alpha_f(a), c_f(a, b)\big) = \min(10, 2) = 2$ along it. This is a **saturating** push: it empties the capacity of (a, b), so this edge leaves G_f and a reverse edge (b, a) of capacity 2 appears in its place. We now have $\alpha_f(a) = 8$ and $\alpha_f(b) = 12$.

a is still the highest vertex with excess flow, so we continue PUSHing from it. The edge (a, t) has residual capacity 4 and goes downhill, so we push $\Delta = \min(8, 4) = 4$. Since $\alpha_f(a) = 8 > 4 = c_f(a, t)$, this saturates (a, t) and gives us $\alpha_f(a) = 4$, $\alpha_f(t) = 4$, the first flow reaching the sink.

Now a has excess but no downhill residual edge. Its only remaining residual edge is (a, s), of capacity 10. So we call RELABEL on a repeatedly until $h(a) = 5$. We can now push along (a, s): $\Delta = \min\big(\alpha_f(a), c_f(a, s)\big) = \min(4, 10) = 4$, a **non-saturating** push, so all of a's excess moves to s and $\alpha_f(a) = 0$.

With a having gotten rid of all its excess flow, b (still at height 0, holding $\alpha_f(b) = 12$) becomes the highest vertex with non zero excess. None of its residual edges are downhill yet, so we RELABEL b once, to $h(b) = 1$, at which point (b, t) becomes downhill. We push $\Delta = \min(12, 9) = 9$, saturating (b, t) and giving $\alpha_f(b) = 3$, $\alpha_f(t) = 9$.

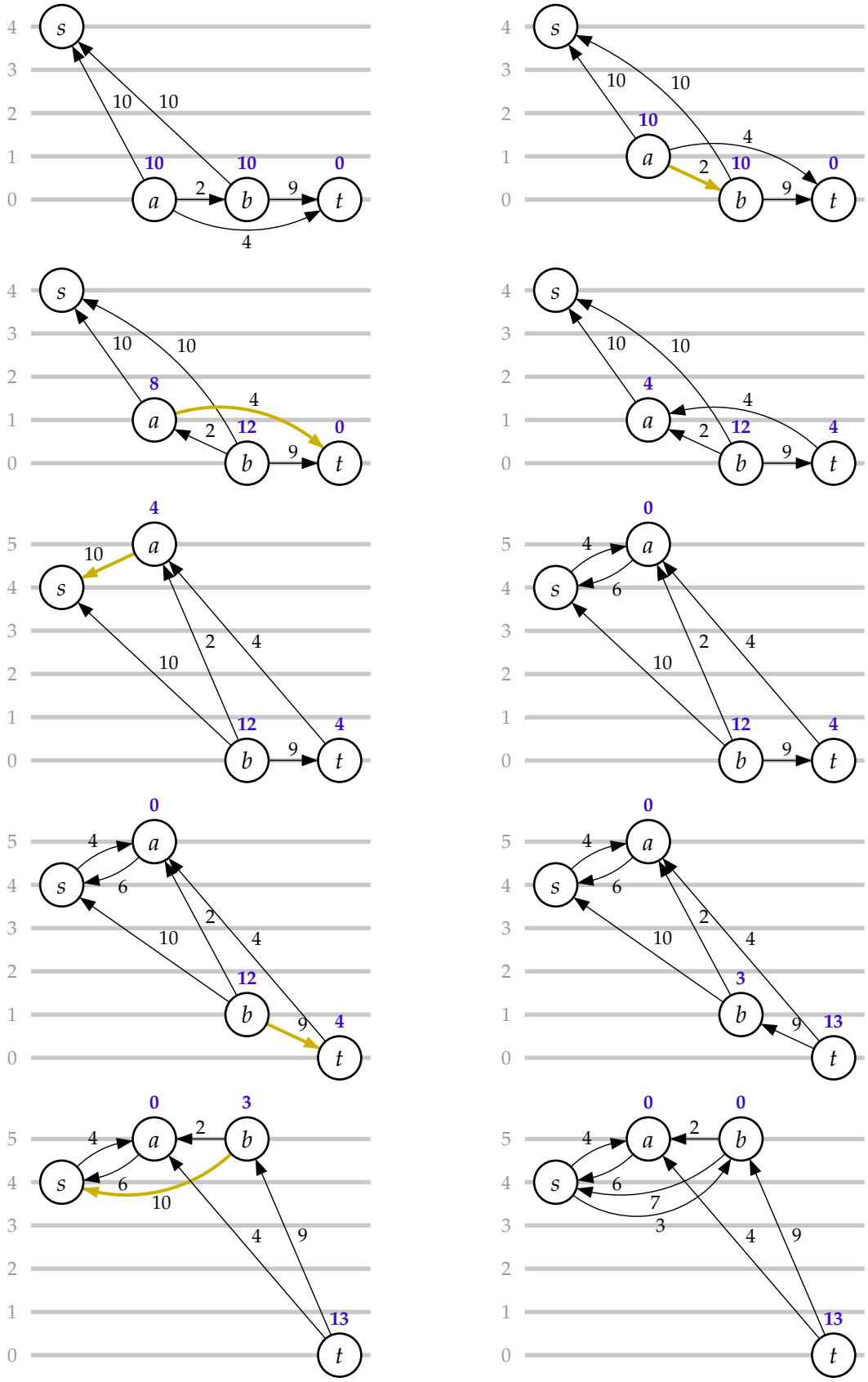

Figure 3: Working of the PUSH-RELABEL algorithm on a flow network G. Consecutive RELABEL operations on the same vertex are shown together. Excess flows are denoted by purple amounts on each vertex. Highlighted edges (in yellow) denote edges which are about to be subject to a PUSH operation.

b still holds excess 3, but its only remaining residual edges now go to a (height 5) and s (height 4). As before, we RELABEL b several times in a row until $h(b) = 5$, then push the remaining $\Delta = \min(3, 10) = 3$ along (b, s), emptying $\alpha_f(b)$.

At this point no vertex in $V - \{s, t\}$ has positive excess flow, so the algorithm terminates. By Lemma 1, the resulting preflow is a valid flow, and by Theorem 1.1 it is a maximum flow. The excess that has accumulated at t, $\alpha_f(t) = 9 + 4 = 13$, is exactly the value of the maximum flow of G. Figure 2 (b) shows the maximum flow itself computed by this algorithm.

Time complexity

Let's recall the algorithm and find its time complexity, to compare it to the other maximum-flow algorithms.

PUSH-RELABEL(G)

```
1  initialize preflow f and height function h
2  while there exists a vertex with excess flow α_f(v) > 0
3      choose such a vertex v with maximum height h(v)
4      if there exists an outgoing edge (v, w) such that h(v) = h(w) + 1
5          PUSH Δ = min(α_f(v), c_f(v, w)) flow along (v, w)
6      else
7          RELABEL v
```

Here, the initialization step runs in linear time. So, the main section affecting the time complexity is the **while** loop. Note that each on each iteration of the while loop, we perform an operation (either PUSH, or RELABEL). But, we have already bounded the total number of operations performed to be $O(n^3 + mn) = O(n^3)$. So, the while loop runs for $O(n^3)$ iterations.

Both the PUSH operation and the RELABEL operation runs in constant time. This is evident from the description of these operations themselves. The only other thing happening in an iteration of the loop, is to find the max-height vertex having positive excess flow (line 3). If we can find this in $O(1)$ time, then we can claim that the algorithm runs in $O(n^3)$ time complexity.

In problem 1, we see that we can simply keep the vertices with non-zero excess into collections by their height, such that we can achieve $O(1)$ amortized selection of the max-height vertex with positive excess flow. Therefore, we get $O(n^3)$ running time of the PUSH-RELABEL algorithm.

As compared to Dinic's algorithm (which was an augmenting path algorithm) which runs in $O(mn^2)$ time, this is an asymptotic improvement to its time complexity, especially for dense graphs.

Problems

1. Design/use a suitable data structure so that the push relabel algorithm takes $O(n^3)$ time. Implement the push relabel algorithm using your data structure and compare its performance against Edmond-Karp's algorithm and Dinic's algorithm.

Call a vertex having positive excess flow 'active'.

We have established, that if we can somehow find the highest active vertex (line 3) in $O(1)$ amortized time, then our algorithm runs in $O(n^3)$.

We will store the **active vertices** in a collection of buckets, indexed by their height. So, create a array $B[]$ of linked lists of size $2n + 1$, representing the heights of the vertices from 0 to $2n$. Each entry $B[h]$ is a doubly linked list of all active vertices currently at height h.

Store a value `max_h` which is the highest index of a bucket which is non-empty (contains at least one vertex).

Operations:
- On PUSH, at most one vertex may become inactive ($O(1)$ deletion from a doubly linked list), at most one vertex may become active ($O(1)$ insertion into a linked list).
- On RELABEL, the height of a vertex changes, which involves deletion from one bucket and insertion to another bucket.
- Whenever we insert a vertex to a linked list, if its $h >$ `max_h`, update `max_h`. Whenever we delete from $h =$ `max_h`, if the list becomes empty, reduce `max_h` to the next non empty bucket.

Choosing the highest active vertex: Choose the vertex at the head of the B[`max_h`] list. ($O(1)$)

In total, on each iteration, there are $O(1)$ insertions and deletions, all of which will run in constant time. The only other thing is decrementing of `max_h` when we delete a vertex from `max_h`, and its list becomes empty. Observe that `max_h` can only increase during a RELABEL operation (and that too by at most 1). Since the total number of relabel operations are $O(n^2)$, the total number of increments to `max_h` is at most $O(n^2)$. So, there can be at most $O(n^2)$ decrements to `max_h` throughout the algorithm. Since there are $O(n^3)$ iterations in total, the cost for decrementing amortizes to $O(1)$ per iteration.

Thus, the total running time of the algorithm is $O(n^3)$.

2. [CLRS] Suppose that all edge capacities in the flow network G are in the set $\{1, 2, ..., k\}$. Analyse the running time of the push relabel algorithm in terms of the number of vertices n, the number of edges m in G as well as k.

- The total number of relabel operations is $O(n^2)$ (Corollary of Lemma 5).
- The total number of saturating push operations is $O(mn)$ (Corollary of Lemma 6).

For the number of non saturating push operations, we can get a better bound here, since each edge capacity is at most k.
- Once an edge (u, v) is saturated, no more flow can be pushed through it, until flow is pushed backward through (v, u), for which v will have to be higher than u. Since the height of a vertex can be $2\,|V|$, an edge can be saturated at most $2\,|V|$ times.
- Between two saturations of the edge (u, v), it can undergo at most k non saturating pushes.

Thus, summing over all edges, the total number of non-saturating pushes is bounded by $O(2k\,|V|\,|E|)$. Since this dominates the complexities of the other operations, the running time of the algorithm is $O(2k\,|V|\,|E|)$.

3. Given a flow network G and a flow f show that the underlying undirected graph of G is connected if and only if the underlying undirected graph of G_f is connected.

If the underlying undirected graph of G is connected, then it is necessary that the underlying undirected graph of G_f is too, since for every node $(u, v) \in E$, at least one of (u, v) or (v, u) exists in E_f. Thus, if there is a path from v_1 to v_2 in the underlying undirected graph of G, then that same path also exists in the underlying undirected graph of G_f.

Suppose the underlying undirected graph of G_f is connected. Then for any $u, v \in V$ there exists a path from u to v in the underlying undirected graph of G_f. Each edge in this path is of the form (u, v) where either (u, v) or (v, u) belongs to E_f. Since the residual edges are either forward or backward edges of the original graph G, either (u, v) or (v, u) exists in E. Therefore the path from u to v exists in the underlying undirected graph of G.

4. Consider a run of push-relabel algorithm. For a residual graph G_f, let $\delta_f(u, v)$ be the number of edges in the shortest path from u to v.
   - Show that for any vertex u, $\delta_f(u, t) \geq h(u)$ where h denotes the current height function.
   - Is it possible that $h(s) \geq h(u) > \delta_f(u, s)$? Construct examples.

- We can show this by induction on the number of operations.
  Base: Initially $\delta_f(s, t) = \infty$, $h(s) = n$; for other vertices $h(v) = 0$, $\delta_f(v, t) \geq 0$.
  Step:
  - PUSH: Suppose a push happens along the edge (v, w), so $h(v) = 1 + h(w)$. After the push, the changes in the set E_f are, a potential removal of the edge (v, w), and a potential addition of the edge (w, v).
    - Removing an edge can never decrease the length of the shortest path between any two vertices. So, $\delta_f(u, t) \geq h(u)$ is maintained.
    - Suppose addition of (w, v) creates a new shorter path from some x to t, then $\delta_f(x, t) = \delta_f(x, w) + 1 + \delta_f(v, t)$. We have, $\delta_f(v, t) \geq h(v)$ (hypothesis). Let $p = \langle x = v_0, ..., v_k = w \rangle$ be the shortest path from x to w in G_f, then for each edge (v_i, v_{i+1}) we have $h(v_i) - h(v_{i+1}) \leq 1$. Adding over all edges we get $h(x) - h(w) \leq \delta_f(x, w)$.

      So we have

$$\begin{aligned} \delta_f(x, t) &= \delta_f(x, w) + 1 + \delta_f(v, t) \\ &\geq h(x) - h(w) + 1 + h(v) \\ &\geq h(x) + 2 > h(x) \end{aligned}$$

  - RELABEL: Suppose vertex u is relabelled, this means there do not exist any downhill edges from u. If $(u, v) \in E_f$, $h(u) \leq h(v)$ (before relabelling)
    - If u has no path to t, $\delta_f(u, t) = \infty$, so the property holds trivially.
    - If u has a path to t, let the first node on the shortest path be y. Then, $\delta_f(u, t) = 1 + \delta_f(y, t)$. Before relabelling, $h(u) \leq h(y)$, so after relabelling $h'(u) \leq 1 + h(y)$.

$$h'(u) \leq 1 + h(y) \leq 1 + \delta_f(y, t) = \delta_f(u, t)$$

  Alternative Solution: (Credits: AS) There is a simpler solution:

  If there is no path from u to t (in G_f), then $\delta_f(u, t) = \infty$ and the inequality trivially holds. Suppose there is a path from u to t. Let the shortest path from u to t be $p = \langle u = v_0, v_1, ..., v_k = t \rangle$. Here, $k = \delta_f(u, t)$.

  For each edge (v_i, v_{i+1}), we have $h(v_i) - h(v_{i+1}) \leq 1$. Adding this over all k edges, we get:

$$\begin{aligned} h(v_0) - h(v_k) \leq k \quad &\Rightarrow h(u) - h(t) \leq k \\ \Rightarrow k \geq h(u) \quad &\Rightarrow \delta_f(u, t) \geq h(u) \end{aligned}$$

- Consider the network $s \xrightarrow{10} a \xrightarrow{10} b \xrightarrow{1} t$.

  A partial simulation of the algorithm is shown below. At some intermediate stage, we have $h(s) = 4$, $h(a) = 3$, $\delta_f(a, s) = 1$, thus $h(s) \geq h(a) > \delta_f(a, s)$.