Man muss immer generalisieren. [One must always generalize.]

— attributed to Carl Gustav Jacob Jacobi (c. 1830) by
Philip J. Davis and Reuben Hersh, *The Mathematical Experience* (1981)

Life is like riding a bicycle. To keep your balance you must keep moving.

— Albert Einstein, in a letter to his son Eduard (February 5, 1930)

A process cannot be understood by stopping it. Understanding must move with the flow of the process, must join it and flow with it.

— The First Law of Mentat, from Frank Herbert's *Dune* (1965)

Scarcely pausing for breath, Vroomfondel shouted, "We don't demand solid facts! What we demand is a total absence of solid facts. I demand that I may or may not be Vroomfondel!"

— Douglas Adams, *The Hitchhiker's Guide to the Galaxy* (1979)

CHAPTER **F**

Balances and Pseudoflows

[Read Chapters 10 and 11 first.]

F.1 Unbalanced Flows

In this chapter, we consider a generalization of flows that allows "stuff" to be injected or extracted from the flow network at the vertices. For the moment, consider a flow network $G = (V, E)$ without any specific source and target vertices. Let $b : V \to \mathbb{R}$ be a ***balance*** function describing how much flow should be injected (if the value is positive) or extracted (if the value is negative) at each vertex. We interpret positive balances as ***demand*** or ***deficit*** and negative balances as (negated) ***supply*** or ***excess***.

We now redefine the word ***flow*** to mean a function $f : E \to \mathbb{R}$ that satisfies the modified balance condition

$$\sum_{u \in V} f(u \to v) - \sum_{w \in V} f(v \to w) = b(v)$$

at *every* node v. A flow f is ***feasible*** if it satisfies the usual capacity constraints $0 \le f(e) \le c(e)$ at every edge e. Our problem now is to compute, given a flow network with edge capacities and vertex balances, either a *feasible* flow in or a proof that no such flow exists.

Note two significant differences from the standard maximum flow problem. First, there are no special source and target vertices that are exempt from the balance constraint. Second, we are not trying to optimize the value of the flow; in fact, the "value" of a flow is no longer even *defined*, because the network has no source and target vertices.

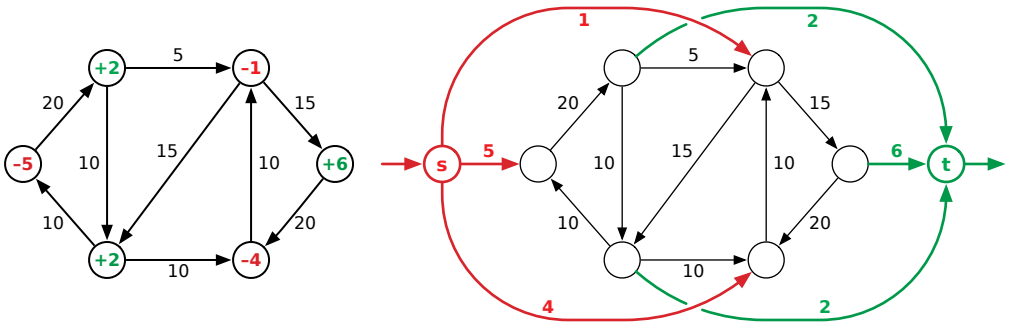
One easy necessary condition is that all the vertex balances must sum to zero; intuitively, every edge $u \to v$ adds the same amount to v's balance that it subtracts from u's balance. More formally, for every feasible flow f, we have

$$\begin{aligned}\sum_v b(v) &= \sum_v \left(\sum_{u \in V} f(u \to v) - \sum_{w \in V} f(v \to w) \right) \\ &= \sum_{u \to v \in E} f(u \to v) - \sum_{v \to w \in E} f(v \to w) = 0.\end{aligned}$$

F.2 Reduction to Maximum Flow

We can reduce the problem of finding a feasible flow in a network with non-zero balance constraints to the standard maximum-flow problem, by adding new vertices and edges to the input graph as follows.

Starting with with original graph G, we construct a new graph $G' = (V', E')$ by adding a new source vertex s with edges to every supply vertex, a new target vertex t with edges from every demand vertex. Specifically, for each vertex v in G, if $b(v) > 0$, we add an edge $v \to t$ with capacity $b(v)$, and if $b(v) < 0$, we add a new edge $s \to v$ with capacity $-b(v)$. Let $c' : E' \to \mathbb{R}$ be the resulting capacity function; by construction, $c'|_E = c$.



A flow network G with non-zero balance constraints, and the transformed network G'.

Now call an (s, t)-flow in G' ***saturating*** if every edge leaving s (or equivalently, every edge entering t) is saturated. Every saturating flow is a maximum flow; conversely, either all maximum flows in G' are saturating, or G' has no saturating flow.

Lemma 1. *G has a feasible flow if and only if G' has a saturating (s, t)-flow.*

Proof: Let $f : E \to \mathbb{R}$ be any feasible flow in G, and consider the function $f' : E' \to \mathbb{R}$ defined as follows:

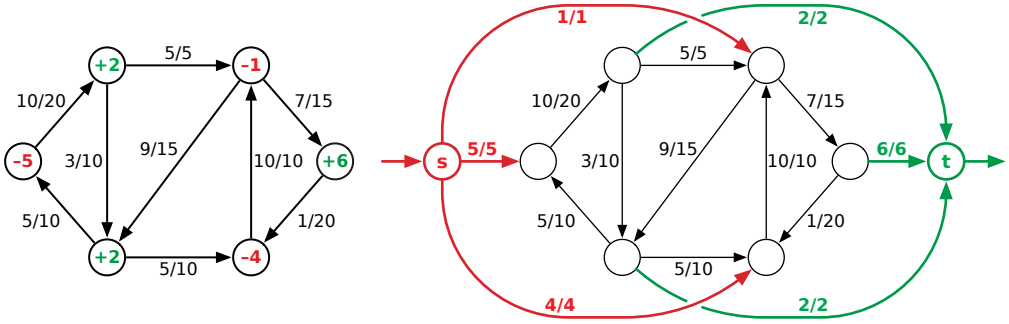
$$f'(e') = \begin{cases} f(e') & \text{if } e' \in E \\ c'(e') & \text{otherwise} \end{cases}$$

Every edge incident to s or t is saturated, and every edge in E satisfies the capacity constraint $0 \le f'(e) = f(e) \le c(e) = c'(e)$. For each vertex v except s and t, we immediately have

$$\begin{aligned} \sum_{u \in V'} f'(u \to v) - \sum_{w \in V'} f'(v \to w) &= \left(\sum_{u \in V} f(u \to v) - \sum_{w \in V} f(v \to w) \right) - b(v) \\ &= b(v) - b(v) \\ &= 0. \end{aligned}$$

We conclude that f' is a feasible (s,t)-flow in G'. Every edge out of s or into t is saturated by definition, so f' is a saturating flow in G'.

Similarly tedious algebra implies that for any saturating (s,t)-flow $f' : E' \to \mathbb{R}$ through G', the restriction $f = f'|_E$ is a feasible flow in G. □



A feasible flow in G and the corresponding saturating flow in G'.

We emphasize that there are flow networks with no feasible flow, even when the sum of the balances is zero. Suppose we partition the vertices of G into two arbitrary subsets S and T. As usual, let $\|S,T\|$ be the total capacity of the cut (S,T):

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

Let $b(S)$ and $b(T)$ denote the sum of the balances of vertices in S and T, respectively:

$$b(S) := \sum_{u \in S} b(u) \qquad \text{and} \qquad b(T) := \sum_{v \in T} b(v).$$

We call the cut (S,T) ***infeasible*** if $\|S,T\| < b(T)$; that is, if T has more demand than can be moved across the cut. The following theorem is a straightforward consequence of the maxflow/mincut theorem (hint, hint):

Theorem 2. *Every flow network has either a feasible flow or an infeasible cut.*

F.3 Pseudoflows

Instead of reducing our new unbalanced flow problem to the classical maximum flow problem, it is instructive to project the behavior of Ford-Fulkerson on the transformed flow network G' back to the original flow network G. The resulting algorithm no longer maintains and incrementally improves a feasible *flow* in G, but a more general function called a ***pseudoflow***. Formally, a pseudoflow in G is *any* function $\psi : E \to \mathbb{R}$. We say that a pseudoflow ψ is ***feasible*** if $0 \le \psi(e) \le c(e)$ for every edge $e \in E$. A ***flow*** is any pseudoflow that also satisfies the balance constraints at every vertex.

For any pseudoflow ψ in G, we define the residual capacity of any edge $u \to v$ as usual:

$$c_\psi(u \to v) := \begin{cases} c(u \to v) - \psi(u \to v) & \text{if } u \to v \in E \\ \psi(v \to u) & \text{if } v \to u \in E \end{cases}$$

We also define the ***residual balance*** of any node to be its original balance minus its net incoming pseudoflow:

$$b_\psi(v) := b(v) - \sum_u \psi(u \to v) + \sum_w \psi(v \to w).$$

A pseudoflow ψ is a flow if and only if $b_\psi(v) = 0$ for every vertex v. Finally, we define the ***residual network*** G_ψ to be the graph of all edges with positive residual capacity, where the balance of each node v is $b_\psi(v)$. As usual, if ψ is zero everywhere, then G_ψ is the original network G, with its original capacities and balances.

Now we redefine an ***augmenting path*** to be any path in the residual graph G_ψ from any vertex with negative residual balance (supply or excess) to any vertex with positive residual balance (demand or deficit). Pushing flow along an augmenting path decreases the total residual supply

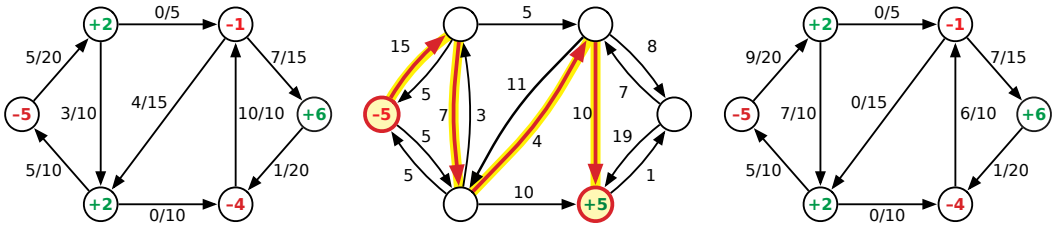
$$B_\psi := \sum_v \left| b_\psi(v) \right|$$

and therefore moves ψ closer to being a feasible flow. The largest amount of flow that we can push along an augmenting path from u to v, before it ceases to be an augmenting path, is the minimum of three quantities:

- The residual supply $-b_\psi(u)$ at the beginning of the path,
- The residual demand $b_\psi(v)$ at the end of the path, and
- The minimum residual capacity among the edges in the path.

On the other hand, if G_ψ contains a vertex v with non-zero residual balance, but does not contain an augmenting path starting or ending at v, then G has no feasible flow. Specifically, if $b_\psi(v) > 0$, then the set S of vertices reachable from v and its complement $T = V \setminus S$ define an infeasible cut; symmetrically, if $b_\psi(v) < 0$, then the set T of vertices that can reach v and its complement $S = V \setminus T$ define an infeasible cut.

Putting all these pieces together, we obtain a simple algorithm for computing either a feasible flow or an infeasible cut. Initialize $\psi(e) = 0$ at every edge e. Then as long



From left to right: A pseudoflow ψ in a flow network G; the residual graph G_ψ with one augmenting path highlighted; and the updated pseudoflow after pushing 4 units along the augmenting path.

as the residual graph G_ψ has a vertex with non-zero balance, update ψ by pushing as much flow as possible along an arbitrary augmenting path. When all residual balances are zero, the current pseudoflow ψ is actual a feasible flow.

```
FeasibleFlow(V, E, c, b):
  for every edge e ∈ E
      ψ(e) ← 0
  B ← Σ_v |b(v)|/2
  while B > 0
      construct G_ψ
      ⟨⟨Find augmenting path π⟩⟩
      s ← any vertex with b_ψ(s) < 0
      if s cannot reach a vertex t in G_ψ with b_ψ(t) > 0
          return Infeasible
      t ← any vertex reachable from s with b_ψ(t) > 0
      π ← any path in G_ψ from s to t
      ⟨⟨Push as much flow as possible along π⟩⟩
      R ← min{−b_ψ(s), b_ψ(t), min_{e∈π} c_ψ(e)}
      B ← B − R
      for every directed edge e ∈ π
          if e ∈ E
              ψ(e) ← ψ(e) + R
          else ⟨⟨rev(e) ∈ E⟩⟩
              ψ(e) ← ψ(e) − R
  return ψ
```

Naturally this algorithm comes with both the same power and the same limitations as Ford-Fulkerson. We can find a single augmenting path in $O(V+E)$ time via whatever-first search. If all capacities and balances are integers, the basic algorithm halts after at most B iterations, where $B = \sum_v |b(v)|/2$, but if we allow irrational capacities **⟨⟨or irrational balances?⟩⟩**, the algorithm could run forever without converging to a flow. Choosing the augmenting path with maximum residual capacity or with the fewest edges leads to faster algorithms; in particular, if we always choose the shortest augmenting path, the algorithm runs in $O(VE^2)$ time.

F.4 Variations on a Theme

There are several variations on the standard maximum-flow problem, with additional or modified constraints, that can be solved quickly using the pseudoflow formulation. These variations are all solved using a two-stage process:

- First find a *feasible* flow f in the original input graph G.
- Then find a *maximum* flow f' in the residual graph G_f.
- Finally, return the flow $f + f'$.

In each variation, the residual graph G_f we use in the second stage will be a textbook-standard flow network. Notice that Ford-Fulkerson itself can be seen as an example of this two-stage algorithm, where the flow f found in the first stage is zero everywhere, or more subtly, where f is obtained by interrupting Ford-Fulkerson after *any* augmentation step.

Maximum Flows with Non-Zero Balances

Suppose we are given a flow network $G = (V, E)$ with edge capacities $c : E \to \mathbb{R}^+$, non-trivial vertex balances $b : V \to \mathbb{R}$, and two special vertices s and t, and we are asked to compute the maximum (s, t)-flow in this network. In this context, an (s, t)-flow is a function $f : E \to \mathbb{R}^+$ that satisfies the modified balance conditions

$$\sum_w f(v \to w) - \sum_u f(u \to v) = b(v)$$

at every vertex v ***except s and t***. As usual, our goal is to find an (s, t)-flow that maximizes the net flow out of s:

$$|f| = \sum_w f(s \to w) - \sum_u f(u \to s)$$

The algorithms in the previous sections *almost* solve the first stage directly, except for two issues: (1) the terminal vertices s and t are not subject to balance constraints, and (2) the sum of the vertex balances need not be zero. In fact, we can handle both of these issues at once by modifying the graph as follows. First, to avoid any ambiguity, we (re)define $b(s) = b(t) = 0$. Then we add one new vertex z with balance $b(z) = -\sum_v b(v)$, and two new infinite-capacity edges $t \to z$ and $z \to s$. Call the resulting modified flow network G'. Straightforward definition-chasing implies that any feasible flow in G' restricts to a feasible (s, t)-flow in G, and conversely, any feasible (s, t)-flow in G can be extended to a feasible flow in G'.

★★★ Simpler to assign balances to s and t. Figure!

Thus, we can find a feasible (s, t)-flow f in G in $O(VE^2)$ time by repeatedly pushing flow along shortest augmenting paths, or in $O(VE)$ time using Orlin's maximum-flow algorithm. In the resulting residual graph G_f, every vertex (except at s and t) has residual balance zero, so we can find a maximum flow in G_f using any standard algorithm.

Lower Bounds on Edges

In another standard variant, the input includes a ***lower bound $\ell(e)$*** on the flow value of each edge e, in addition to the capacity $c(e)$. In this context, a flow f is feasible if and only if $\ell(e) \le f(e) \le c(e)$ for every edge e. In the standard flow problem, we have $\ell(e) = 0$ for every edge e.

Although it is natural to require the lower bounds $\ell(e)$ to be non-negative, we can in fact allow negative lower bounds, and therefore negative flow values $f(e)$, if we interpret negative flow along an edge $u \to v$ as positive flow along its reversal $v \to u$. More formally, we define a pseudoflow as a function $\psi : E \to \mathbb{R}$ such that

$$\psi(v \to u) = -\psi(u \to v)$$

for every edge $u \to v$; more simply, a pseudoflow is an *antisymmetric* function over the edges. The antisymmetry is reflected in the upper and lower bounds on flow:

$$\ell(v \to u) = -c(u \to v) \qquad c(v \to u) = -\ell(u \to v)$$

Then for any pseudoflow ψ, each edge $u \to v$ has both a residual capacity and a residual lower bound, defined as follows to maintain antisymmetry:

$$\ell_\psi(u \to v) := \begin{cases} \ell(u \to v) - \psi(u \to v) & \text{if } u \to v \in E \\ \psi(u \to v) - c(v \to u) & \text{if } v \to u \in E \end{cases}$$

$$c_\psi(u \to v) := \begin{cases} c(u \to v) - \psi(u \to v) & \text{if } u \to v \in E \\ \psi(u \to v) - \ell(v \to u) & \text{if } v \to u \in E \end{cases}$$

Now the residual network G_ψ consists of all edges $u \to v$ with non-negative residual capacity $c_\psi(u \to v) \ge 0$. We can easily verify (hint, hint) that this antisymmetric formulation of (pseudo)flows and residual graphs is completely consistent with our usual formulation of flows as *non-negative* functions.

Given a flow network with both lower bounds and capacities on the edges, we can compute a maximum (s, t)-flow as follows. If all lower bounds are zero or negative, we can apply any standard maxflow algorithm, after replacing each edge $u \to v$ with a negative lower bound with a pair of opposing edges $u \to v$ and $v \to u$, each with positive capacity.[1] Otherwise, we first define an initial feasible pseudoflow ψ that meets every positive lower bound:

$$\psi(u \to v) = \max\{\ell(u \to v), 0\}$$

Vertices in the resulting residual graph G_ψ may have non-zero residual balance, but every edge has a lower bound that is either zero or negative. Thus, we can compute a maximum (s, t)-flow in G_ψ using the previous two-stage approach, in $O(VE)$ time.

Figure!

★★★

[1] Wait, didn't we forbid opposing edges two chapters ago? Okay, fine: Replace each edge $u \to v$ with a pair of opposing *paths* $u \to x \to v$ and $v \to y \to u$, where x and y are new vertices.

♥F.5 Push-Relabel

★★★

This section needs figures and an editing sweep. Cite Alexander Karzanov (preflows)?

The pseudoflow formulation is the foundation of another family of efficient maximum-flow algorithms that is *not* based on path-augmentation, called ***push-relabel*** or ***preflow-push*** algorithms, discovered by Andrew Goldberg and then refined in collaboration with Robert Tarjan before formal publication in 1986, while Goldberg was still a PhD student.

Every push-relabel algorithm maintains a special type of pseudoflow, called a ***preflow***, in which every node (except s) has non-negative residual balance:

$$b_\psi(v) := \sum_u \psi(u{\to}v) - \sum_w \psi(v{\to}w) \geq 0.$$

We call a vertex ***active*** if it is not the target vertex t and its residual balance is positive; we immediately observe that ψ is actually a *flow* if and only if no vertex is active. The algorithm also maintains a non-negative integer ***height*** $\boldsymbol{ht(v)}$ for each vertex v. We call any edge $u{\to}v$ with $ht(u) > ht(v)$ a ***downward*** edge.

The push-relabel algorithm begins by setting $ht(s) = |V|$ and $ht(v) = 0$ for every node $v \neq s$, and choosing an initial pseudoflow ψ that saturates all edges leaving s and leaves all other edges empty:

$$\psi(u{\to}v) = \begin{cases} c(u{\to}v) & \text{if } u = s \\ 0 & \text{otherwise} \end{cases}$$

Then for as long as there are active nodes, the algorithm repeatedly performs one of the following operations at an arbitrary active node u:

- ***Push:*** For some downward residual edge $u{\to}v$ out of u, increase $\psi(u{\to}v)$ by the minimum of the excess at u and the residual capacity of $u{\to}v$.
- ***Relabel:*** If u has no downward outgoing residual edges, increase the height of u by 1.

It is not at all obvious (at least, it wasn't obvious to me at first) that the push-relabel algorithm correctly computes a maximum flow, or even that it halts in finite time. To prove that the algorithm is correct, we need a series of relatively simple observations.

First, we say that the height function $ht : V \to \mathbb{N}$ and the pseudoflow $\psi : E \to \mathbb{R}$ are ***compatible*** if $ht(u) \leq ht(v) + 1$ for every edge $u{\to}v$ in the residual graph G_ψ.

Lemma 3. *After each step of the push-relabel algorithm, the height function ht and the pseudoflow ψ are compatible.*

Proof: We prove the lemma by induction. Initially, every residual edge either enters the source vertex s or has both endpoints with height zero, so the initial heights and pseudoflow are compatible. For each later step of the algorithm, there are two cases to consider.

- Just before we push flow along the residual edge $u \to v$, then $u \to v$ must be a downward edge, so $ht(v) < ht(u)$. If the edge $u \to v$ was empty before the push, then this step adds the reversed edge $v \to u$ to the residual graph, and $ht(v) < ht(u) \leq ht(u) + 1$.
- On the other hand, just before we relabel an active vertex v, we must have $ht(v) \leq ht(w)$ for every outgoing residual edge $v \to w$ and (by the induction hypothesis) $ht(u) \leq ht(v) + 1$ for every incoming residual edge $u \to v$. Thus, after relabeling, we have $ht(v) \leq ht(w) + 1$ for every outgoing residual edge $v \to w$ and $ht(u) \leq ht(v) \leq ht(v) + 1$ for every incoming residual edge $u \to v$.

In both cases, compatibility of the new height function and the new pseudoflow follows from the inductive hypothesis. □

Lemma 4. *After each step of the push-relabel algorithm, there is no residual path from s to t.*

Proof: Suppose to the contrary that there is a simple residual path $v_0 \to v_1 \to \cdots \to v_k$, where $v_0 = s$ and $v_k = t$. This path does not repeat vertices, so $k < |V|$. Because the height of s and t never change, we have $ht(v_0) = |V|$ and $ht(v_k) = 0$. Finally, compatibility implies $ht(v_i) \geq ht(v_{i-1}) - 1$, and thus by induction $ht(v_i) \geq ht(v_0) - i$, for each index i. In particular, we have $ht(v_k) \geq |V| - k > 0$, giving us a contradiction. □

Lemma 5. ***If*** *the push-relabel algorithm terminates, it returns a maximum (s, t)-flow.*

Proof: The algorithm terminates only when there are no active vertices, which means that every vertex except s and t has zero residual balance. A pseudoflow with non-zero residual balance is *definition* of a flow! Any flow whose residual graph has no paths from s to t is a maximum (s, t)-flow. □

For a full proof of correctness, we still need to prove that the algorithm terminates, but the easiest way to prove termination is by proving an upper bound on the running time. In the analysis, we distinguish between two types of push operations. A push along edge $u \to v$ is ***saturating*** if we have $\psi(u \to v) = c(u \to v)$ after the push, and ***non-saturating*** otherwise.

Lemma 6. *The push-relabel algorithm performs at most $O(V^2)$ relabel operations.*

Proof: At every stage of the algorithm, each active node v has a residual path back to s (because we can follow some unit of flow from s to v). Compatibility now implies that the height of each active node is less than $2|V|$. But we only change the height of a node when it is active, and then only upward, so the height of *every* node is less than $2|V|$. We conclude that each node is relabeled at most $2|V|$ times. □

Lemma 7. *The push-relabel algorithm performs at most $O(VE)$ saturating pushes.*

Proof: Consider an arbitrary edge $u \to v$. We only push along $u \to v$ when $ht(u) > ht(v)$. If this push is saturating, it removes $u \to v$ from the residual graph. This edge reappears in the residual graph only when we push along the reversed edge $v \to u$, which is only possible when $ht(u) < ht(v)$. Thus, between any saturating push through $u \to v$ and the reappearance of $u \to v$ in the residual graph, vertex v must be relabeled at least twice. Similarly, vertex u must be relabeled at least twice before the next push along $u \to v$. By the previous lemma, u and v are each relabeled at most $2|V|$ times. We conclude that there are at most $|V|$ saturating pushes along $u \to v$. □

Lemma 8. *The push-relabel algorithm performs at most $O(V^2E)$ non-saturating pushes.*

Proof: Define the ***potential* Φ** of the current residual graph to be the sum of the heights of all active vertices. This potential is always non-negative, because heights are non-negative. Moreover, we have $\Phi = 0$ when the algorithm starts (because every active node has height zero) and $\Phi = 0$ again when the algorithm terminates (because there are no active vertices).

Every relabel operation increases Φ by 1. Every saturating push along $u \to v$ makes v active (if it wasn't already), and therefore increases Φ by at most $ht(v) \leq 2|V|$. Thus, the total potential increase from all relabels and saturating pushes is at most $O(V^2E)$.

On the other hand, every non-saturating push along $u \to v$ makes the vertex u inactive and makes v active (if it wasn't already) and therefore *decreases* the potential by at least $ht(u) - ht(v) \geq 1$.

Because the potential starts and ends at zero, the total potential *decrease* from all non-saturating pushes must equal the total potential *increase* from the other operations. The lemma follows immediately. □

With appropriate elementary data structures, we can perform each push in $O(1)$ time, and each relabel in time proportional to the degree of the node. It follows that the algorithm runs in ***$O(V^2E)$ time***; in particular, the algorithm always terminates with the correct output!

Like the Ford-Fulkerson algorithm, the push-relabel approach can be made more efficient by carefully choosing *which* push or relabel operation to perform at each step. Two natural choices lead to faster algorithms:

- **FIFO:** The active vertices are kept in a standard queue. At each step, we remove the active vertex from the front of the queue, and then either push from or relabel that vertex until it becomes inactive. Any newly active vertices are inserted at the back of the queue. This rule reduces the number of non-saturating pushes to $O(V^3)$, and so the resulting algorithm runs in ***$O(V^3)$ time***.
- **Highest label:** At each step, we either push from or relabel the vertex with maximum height (breaking ties arbitrarily). This rule reduces the number of non-saturating pushes to $O(V^2\sqrt{E})$, and so the resulting algorithm runs in ***$O(V^2\sqrt{E})$ time***.

With more advanced data structures that support pushing flow along more than one edge at a time, the running time of the push-relabel algorithm can be improved to $\boldsymbol{O(VE \log(V^2/E))}$. (This was one of the theoretically-fastest algorithms known before Orlin's algorithm.) In practice, however, this optimization is usually slower than the more basic algorithm that handles one edge at a time.

Exercises

Need more!

★★★

1. Recall from Chapter 11 that a ***path cover*** of a directed acyclic graph is a collection of directed paths, such that every vertex in G appears in at least one path. We previously saw how to compute *disjoint* path covers (where each vertex lies on *exactly* one path) by reduction to maximum bipartite matching. Your task in this problem is to compute path covers *without* the disjointness constraint.

   (a) Suppose you are given a dag G with a unique source s and a unique sink t. Describe an algorithm to find the smallest path cover of G in which every path starts at s and ends at t.

   (b) Describe an algorithm to find the smallest path cover of an arbitrary dag G, with no additional restrictions on the paths.

2. (a) Prove that any flow f in a network $G = (V, E)$ with non-zero balance constraints (and no source or target) can be expressed as a weighted sum of directed paths and directed cycles, such that
   - each path leads from an excess node to a deficit node;
   - a directed edge $u \to v$ appears in at least one path or cycle if and only if $f(u \to v) > 0$; and
   - the total number of paths and cycles is at most E.

   (b) Describe an algorithm to construct such a decomposition in $O(VE)$ time.

3. Let $G = (V, E)$ be an arbitrary flow network with source s and sink t. Recall that a ***preflow*** is a pseudoflow $\psi : E \to \mathbb{R}$ where every vertex except s has non-negative residual balance; that is, for each vertex v, we have

$$\sum_u \psi(u \to v) > \sum_w \psi(v \to w).$$

   A preflow ψ is *feasible* if $0 \le \psi(u \to v) \le c(u \to v)$ for every edge $u \to v$. A ***maximum*** preflow if is a feasible preflow such that the net flow into t

$$\sum_u \psi(u \to t) - \sum_w \psi(t \to w)$$

is as large as possible. Describe an algorithm to transform an arbitrary maximum pseudoflow into a maximum flow in $O(VE)$ time. *[Hint: Flow decomposition!]*

4. (a) An ***edge cover*** of an undirected graph $G = (V, E)$ is a subset of edges $C \subseteq E$ such that every vertex is the endpoint of *at least* one edge in C. (Thus, edge covers are the "opposite" of matchings.) Describe and analyze an efficient algorithm to compute the *smallest* edge cover of a given bipartite graph.

   (b) Describe and analyze an algorithm for the following more general problem. The input consists of a bipartite graph $G = (V, E)$ and two functions $\ell, u : V \to \mathbb{N}$. Your algorithm should either output a subset of edges $C \subseteq E$ such that each vertex $v \in V$ is incident to at least $\ell(v)$ edges and at most $u(v)$ edges in C, or correctly report that no such subset exists.

5. (a) Suppose we are given a directed graph G, two vertices s and t, and two functions $\ell, c : V \to \mathbb{R}$ over the *vertices*. An (s,t)-flow $f : E \to \mathbb{R}$ in this network is feasible if and only if the total flow into each vertex v (except s and t) lies between $\ell(v)$ and $c(v)$:

$$\ell(v) \le \sum_{u \to v} f(u \to v) \le c(v).$$

   Describe an efficient algorithm to compute a maximum (s,t)-flow in this network.

   (b) Suppose we are given a directed graph G, two vertices s and t, and two functions $b^-, b^+ : V \to \mathbb{R}$ over the vertices. An (s,t)-flow $f : E \to \mathbb{R}$ in this network is feasible if and only if the total ***net*** flow into each vertex v (except s and t) lies between $b^-(v)$ and $b^+(v)$:

$$b^-(v) \le \sum_{u \to v} f(u \to v) - \sum_{v \to w} f(v \to w) \le b^+(v).$$

   Describe an efficient algorithm to compute a maximum (s,t)-flow in this network.

   (c) Describe an efficient algorithm to compute a maximum (s,t)-flow in a network with *all* of the features we've seen so far:
   - upper and lower bounds on the flow through each edge,
   - upper and lower bounds on the flow into each vertex, and
   - upper and lower bounds on the flow balance at each vertex.