Problem 1

(Max flow using the bottleneck path) [REDACTED]

Problem 2

(Push-Relabel algorithm) The algorithms explored up to this point share a fundamental attribute: they aim to preserve a conservation of flow constraints while working towards severing the connection between s and t (aka a "cut"). But in the 1980s, some researchers began thinking about max-flow another way: what if, instead of maintaining a connection between s and t, we start with them already disconnected and then "relax" our flow to maintain conservation? This paradigm shift led to the discovery of the Push-Relabel algorithm, which we now present below.[1]

Before we do so, we need to introduce a few definitions.

- A **preflow** is a nonnegative vector $\{f_e\}_{e \in E}$ that satisfies two constraints:
  1. **Capacity constraints:** $f_e \leq c_e$ for every edge $e \in E$;
  2. **Relaxed conservation constraints:** for every vertex v other than s,

     $$\text{amount of flow entering } v \geq \text{amount of flow exiting } v.$$

     We call the difference between the LHS and the RHS the **excess** of a vertex.
- The **height** of a vertex $h(v)$ is defined by a function $h\colon V \to \mathbb{N}$. Intuitively, you can think of h as a potential function that flows downhill from s to t.

Now we can introduce our algorithm, defined on a graph $G = (V, E, c_e, s, t)$. Also denote the residual graph as $G^f = (V, E^f, c_e^f, s, t)$.

[1] For additional motivation and context behind this algorithm, please consult this lecture by Tim Roughgarden.

Algorithm 1 PUSHRELABELMAXFLOW(G)

Initialize $h(s) = n$
Initialize $h(v) = 0$ for all $v \neq s$
Initialize $f_e = c_e$ for all $e \in$ outgoing edges from s
Initialize $f_e = 0$ for all $e \notin$ outgoing edges from s
while exists vertices $\mathcal{V} \in V \setminus \{s, t\}$ with excess flow > 0
    Choose vertex v such that $v = \arg\max_{\nu \in \mathcal{V}} h(\nu)$.
    if exists outgoing edge (v, w) in E^f such that $h(v) = h(w) + 1$ **then**
        PUSH(v, w)
    else
        RELABEL(v)

▷ Subroutine for Push operation
procedure PUSH(v, w)
    $\Delta = \min \left\{ \text{excess flow}(v), c^f_{(v,w)} \right\}$
    Push Δ units of flow along (v, w)

▷ Subroutine for Relabel operation
procedure RELABEL(v)
    Increment $h(v)$ by 1

(a) The network below represents the (hypothetical) connections between a number of international airports, where the capacity of an edge represents the maximum number of flights that can be scheduled between two airports on a given day.

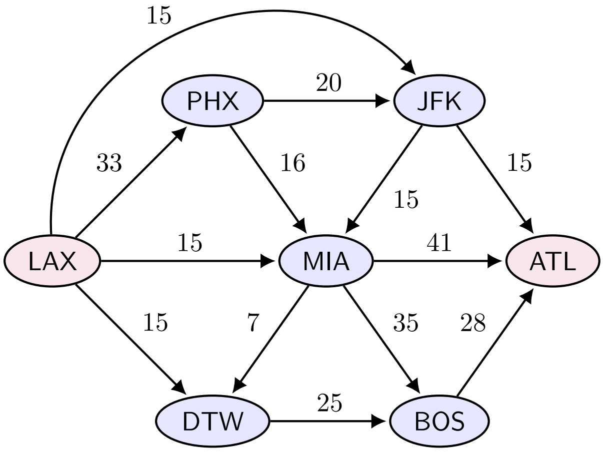
Imagine an urgent situation, like a massive concert or a sports event suddenly announced in Atlanta, has just caused a surge of travelers needing to fly from Los Angeles (LAX) to Atlanta (ATL) ASAP. The big question is: how many flights can actually make this journey in a single day without overstretching the system?

Run the Push-Relabel Algorithm to answer this question. Along with your final numerical answer, include the following:

i. A sketch of the final residual graph with the accompanying capacities (indicate edges in $E^f \setminus E$ with dashed lines), as well as the final heights of each node.

ii. The number of times the `Push` subroutine has been called, as well as the number of times the `Relabel` subroutine was called.

In the case of ties, choose the node with the smallest lexicographical order.



(b) A basic implementation of the algorithm in the exposition gives us a runtime of $\mathcal{O}(mn^2)$. Give an implementation of this algorithm that runs in $\mathcal{O}(n^3)$ time, and prove the runtime of your algorithm.

Hints: first prove the running time bound assuming that, in each iteration, you can identify the highest vertex with positive excess in O(1) time. The hard part is to maintain the vertices with positive excess in a data structure such that, summed over all of the iterations of the algorithm, only $\mathcal{O}(n^3)$ *total time is used to identify these vertices.*

(c) *(Optional)* It can be shown (with a little more work) that a tight bound for this algorithm is $\Theta(n^2\sqrt{m})$[2]. Although this bound is nothing special in the worst-case (for example, Dinic's algorithm runs in time $\mathcal{O}(nm^2)$, the PR algorithm is often the one used.[3] What are some reasons why this algorithm may perform better in practice?

[2]Asymptotically, a $\sqrt{m}$ term is better than a n term.

[3]This dichotomy with theoretical bounds and practical application is oftentimes the case with flow-based networks; for example, in computer vision.