Precept Outline

- DP algorithms for NP-complete problems
  - Knapsack
  - Subset Sum
- Push-relabel maxflow algorithms

A. Dynamic Programming for NP-Complete Problems

It is fairly common for well-known hard (i.e., NP-complete) problems to admit dynamic programming solutions; these solutions aren't efficient in general, of course (otherwise they would imply P $=$ NP), but can still be efficient in special cases. These are known as *pseudo-polynomial time* algorithms. (The general coin changing problem is one such example.)

Part 1: Subset Sum

(One version of) the SUBSET-SUM problem is defined by a set $S \subset \mathbb{N}$ and a target value $v \in \mathbb{Z}$. A solution is a subset $\varnothing \neq T \subseteq S$ whose numbers add to v, i.e., such that $\sum_{x \in T} x = v$.

Suppose, without loss of generality, that S is given by the sequence $(x_1, x_2, \ldots, x_n)$. Find a dynamic programming recurrence considering subproblems on $S_i := \{x_1, \ldots, x_i\}$. Conclude, from it, a $O(nv)$-time algorithm for SUBSET-SUM.

Part 2: Knapsack

(One version of) the KNAPSACK is defined by a set $P \subset \mathbb{N}^2$ (of *weight* and *value* pairs) of the same size along with a (maximum weight) $m \in \mathbb{N}$. Writing $P = \{(w_1, v_1), \ldots, (w_n, v_n)\}$, a solution is a subset $S \subseteq [n]$ that maximizes the value $\sum_{i \in [n]} v_i$ under the maximum weight constraint $\sum_{i \in S} w_i \leq m$. (Note that this problem generalizes SUBSET-SUM, by checking if the optimal solution for instance $P = \{(x_i, x_i) : i \in [n]\}$ with maximum weight v has value v.)

Find a dynamic programming recurrence for KNAPSACK, and conclude from it a $\Theta(mn)$-time algorithm for the problem.

B. Push-Relabel Maxflow Algorithms

Push-relabel algorithms are a "second generation" of maxflow algorithms developed after Ford-Fulkerson that improve on its $O(VE^2)$ complexity: simpler implementations run in $O(V^2E)$ time, and can achieve $O(VE\log(V^2/E))$ with advanced data structures (namely, link-cut trees).

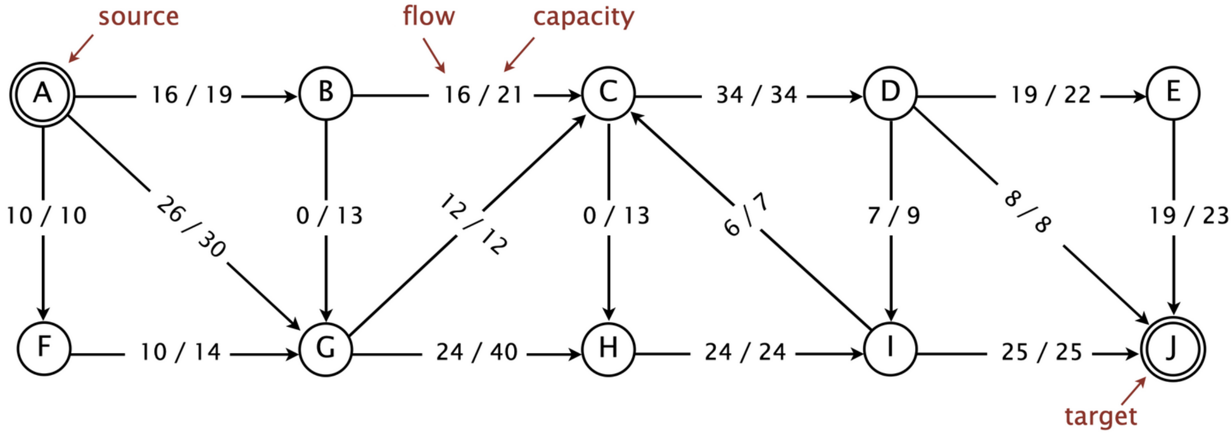
Part 1: The Residual Graph

Defining the Ford-Fulkerson algorithm usually involves the definition of another graph related to a flow network: the *residual graph* (which we work with in the course, but don't define explicitly).

Since push-relabel algorithms also rely on residual graphs, let's define them and work through an example. The residual graph of a flow network $f\colon E \to \mathbb{R}_+$ on the graph $G = (V, E)$ with capacity $c(e)$ and flow $f(e)$ on edge $e \in E$ is the weighted graph $R_f = (V, E')$ defined as follows:

- edge $e = (u, v) \in E$ belongs to E' if $c(e) - f(e) > 0$ (i.e., if the *residual capacity* is positive). Its weight is defined as the residual capacity $w(e) \coloneqq c(e) - f(e)$.
- edge $e' = (v, u)$ belongs to E' if and only if $(u, v) \in E$ has positive flow: $f(u, v) > 0$. Its weight is defined as $w(v, u) \coloneqq f(u, v)$.

Draw the residual graph corresponding to the following flow network, and succinctly explain how it relates to the Ford-Fulkerson algorithm.



Part 2: Preflows, Height Functions and Pushes

Push-relabel algorithms work by violating flow conservation in their execution, but still satisfy it when they finish. They instead preserve a relaxation of flow conservation: we say f is a *preflow* if the net flow at every non-source/sink vertex is *non-negative* (as opposed to 0). If v has positive inflow, we say the vertex is *overflowing* and call the excess flow $e(v)$.

One of the (two) key operations of push-relabel algorithms is `push(u, v)`, applied to an overflowing vertex u and a vertex v adjacent to it in the residual graph. This operation increases the preflow f in the edge by $\min\{e(u), w(u,v)\}$ (where $w(u,v)$ is the residual capacity, i.e., the weight of the edge in the residual graph).

Prove the following properties of preflows and pushes:

- the function $f\colon E \to \mathbb{R}_+$ with $f(s, v) := c(s, v)$ for all vertices v adjacent to the source and $f(e) := 0$ for all other edges is a preflow.
- if f is a preflow and f' the function obtained from f after a `push()` operation, then f' is a preflow.

Part 3: Height Functions

A *height function* h is a function $h\colon V \to \mathbb{N}$ (with respect to a preflow f) that satisfies the following properties:

- $h(s) = 0$ and $h(t) = |V|$ (where s is the source and t is the target);
- if (u, v) is an edge of the residual graph R_f, then $h(u) \leq h(v) + 1$.

Prove that, given any preflow f and height function h, the residual graph R_f does not contain an st-path.

Part 4: Relabels and the Full Algorithm

The operation `relabel(u)` is applied to an overflowing vertex u such that $h(u) \leq h(v)$ for all vertices v adjacent to u in the residual graph. The operation increases $h(u)$ to the smallest value that makes pushing possible on u, namely, $h(u) := 1 + \min_{(u,v)\in E'}\{h(v)\}$.

The full "meta-algorithm" is the following:

1. Initialize the preflow f with $f(s, v) := c(s, v)$ for all vertices adjacent from the source and $f(e) := 0$ otherwise.
2. Repeat until there are no more overflowing vertices:
   - Select an overflowing vertex u. If there is a vertex v adjacent to it with $h(v) = h(u) - 1$, apply `push(u, v)`. Otherwise, apply `relabel(u)`.

Prove that the algorithm is well-defined, i.e., that

- every overflowing vertex either satisfies the conditions for a `push()` or a `relabel()` operation under any height function; and
- if h is a height function and h' is the function obtained from h after a `relabel()` operation, then h' is a height function (with respect to the same preflow).
- if f is a preflow, h is a height function with respect to f and f' is the preflow obtained from f by a `push()` operation, then h is also a height function with respect to f'.

Conclude that if the push-relabel meta-algorithm terminates, then f is a maxflow.