1 Flow in graphs

Let $G = (V, E)$ be a directed graph, and let $c(e) \in \mathbb{N}$ denote the *capacity* of edge e. Furthermore, there are two special nodes: a source s and a sink/target t. Our goal is to send as much *flow* as possible from s to t without exceeding the edge capacities. For each node except the source and the sink the amount of flow entering the node must equal the amount exiting the node.

More exactly, a *feasible flow* from s to t is a mapping $f : E \to \mathbb{R}$ specifying the amount of flow over each edge such that the following properties are satisfied:

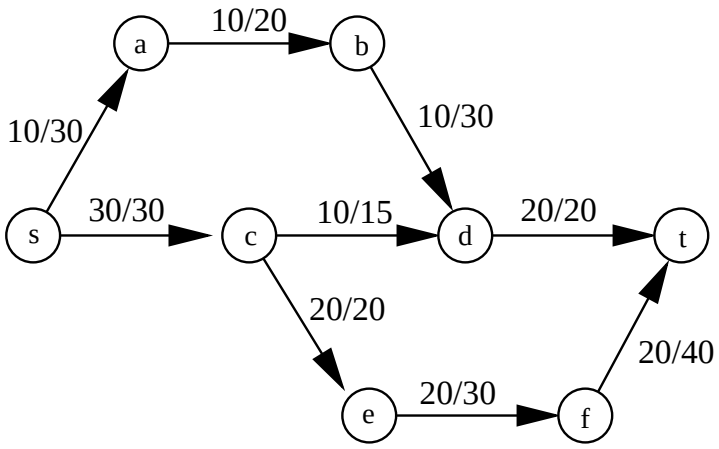
- Capacity constraint: $\forall e \in E : \ 0 \leq f(e) \leq c(e)$
- Flow conservation: $\forall v \in V \setminus \{s, t\} : \ \sum_{e=(u,v)\in E} f(e) = \sum_{e=(v,w)\in E} f(e)$

The flow value F of f is the amount of flow which flows from s to t, that is:

$$F = \sum_{e=(s,w)\in E} f(e) - \sum_{e=(u,s)\in E} f(e) = \sum_{e=(u,t)\in E} f(e) - \sum_{e=(t,w)\in E} f(e)$$

A *maximum flow* is a flow that maximizes the flow value F over all feasible flows.

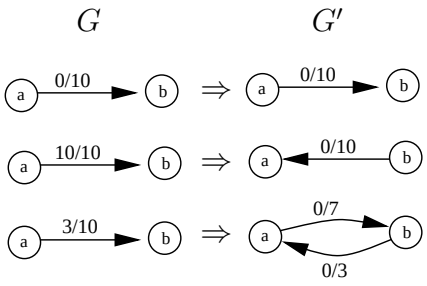
The following picture illustrates a maximum flow with flow value 40 from s to t in a graph. For each edge, the edge label shows the flow over the edge and its capacity.



1.1 The residual network

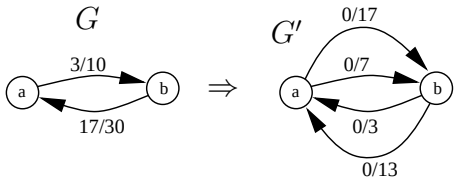
For a (not necessarily feasible) flow $f : E \to \mathbb{R}$ in $G = (V, E)$ we can construct the so called *residual network* $G' = (V, E')$:

Create the graph G' from G by copying all nodes of G and adding edges to G' under the following rules.



If $f(e) < c(e)$ for an edge $e = (a, b) \in E$, then add the (forward) edge $e' = (a, b)$ with capacity $c'(e') = c(e) - f(e)$ to E'. If, on the other hand, $f(e) > 0$ for an edge $e = (a, b) \in E$, then add the (backward) edge $e' = (b, a)$ with capacity $c'(e') = f(e)$ to E'. Hence, for each edge of G, one or two edges will be added to the residual network G'. This is illustrated in the next picture.

If there are two antiparallel edges between two nodes a and b, then up to four edges between a and b are added as illustrated alongside.



Thus, to construct the residual network G', we only have to iterate over all edges $e \in E$ and add up to two edges to G' depending on the flow $f(e)$ and capacity $c(e)$ of the edge. In particular, if $f(e) = 0$ for all edges $e \in E$, then the residual network equals the original graph G and only contains forward edges. If the flow of an edge changes, then we can apply the corresponding modification of the residual network G' (using appropriate data structures) in constant time.

Remark: The algorithm of Dinic computes the maximum flow in a graph by searching for shortest s, t-pathes in the residual network. The running time of this is $O(|V|^2|E|)$. We will consider some other algorithm with running time $O(|V|^3)$.

1.2 The flow algorithm of Goldberg and Tarjan

We assume for the sake of simplicity that each node $v \in V \setminus \{s, t\}$ is reachable from the source and that the sink is reachable from each such node. (All nodes that don't satisfy these requirements don't contribute anything to the value of the flow. These nodes can be found and removed in linear time.) We want find a *maximum flow* in G from s to t. To this end, we consider the Preflow-Push-Algorithm of Goldberg and Tarjan. The running time of this algorithm (or more specifically of its considered implementation) is in $O(|V|^3)$.

Besides the concept of the residual network, the algorithm makes use of two new concepts: *preflow* and *distance function* (also called *height function*). A preflow $\tilde{f}$ is a mapping $\tilde{f} : E \to \mathbb{R}$ having the following properties:

(a) $\forall e \in E : \ 0 \leq \tilde{f}(e) \leq c(e)$

(b) $\forall v \in V \setminus \{s, t\} : \ \sum_{e=(u,v)\in E} \tilde{f}(e) - \sum_{e=(v,w)\in E} \tilde{f}(e) \geq 0$

In contrast to a feasible flow, a preflow doesn't need to satisfy the flow preservation property: the amount of flow entering a node can be larger than the amount exiting the node. The *excess* (also called *overflow*) of a node v is defined by

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

and is saved at node v by the algorithm, and appropriately updated when changes take place. The nodes $v \in V \setminus \{s,t\}$ with $e(v) > 0$ are called *active* nodes, and are stored in a queue. The algorithm starts with a preflow and iteratively tries to *push* the excess of an active node towards the sink, until we obtain a feasible flow at the end.

As described in section 1.1, in context of a preflow $\tilde{f}$ in G a corresponding residual network G' exists where $c'(e')$ denotes the capacity of an edge e' of the residual network, i.e., $c'(e') = c(e) - \tilde{f}(e)$ for forward edges, and $c'(e') = \tilde{f}(e)$ for backward edges. The algorithm of Goldberg and Tarjan updates the residual network G' at each change of the preflow in G such that at each point in time the residual network of the preflow is available. (Remark: It is also possible to skip the *explicit* construction of the residual network G' such that the algorithm only uses the graph G and additional administrative information.)

A function $d : V \to \mathbb{N}_0$ is a distance function of G with respect to a preflow $\tilde{f}$, if $d(t) = 0$ and if for each edge $e = (v,w)$ of the residual network (!) $d(v) \leq d(w)+1$ holds. A distance function d is *legal* if for each node d the value $d(v)$ is *at most* the distance from v to t (i.e., the least number of edges needed to go from v to t) *in the residual network*. In context of a distance function, an edge $e = (v,w)$ of the residual network is *legal* if $d(v) = d(w) + 1$.

The algorithm of Goldberg and Tarjan iteratively chooses an active node v and applies a so called push/relabel-operation: as long as v has positive excess and there is a legal edge emanating from v, an amount of this excess as large as possible is pushed over this edge (push); if there is no legal edge emanating from v but the excess of v is still positive, the distance $d(v)$ is modified (relabel). From there, the complete algorithm can be described as follows:

1. **Initializing the preflow:**
   Set $\tilde{f}(e) = c(e)$ for all edges emanating s and set $\tilde{f}(e) = 0$ for all other edges. Add all nodes $v \neq t$ having $(s,v) \in E$ to the queue of active nodes.

2. **Initializing the distance function:**
   For each node $v \neq s$ set $d(v)$ to the shortest directed path from v to t in the residual network where the length of a path is the number of edges of the path. (In particular, set $d(t) = 0$). This can be done in time $O(|V| + |E|)$ by a breadth first search starting at t and using all edges in the opposite direction. The source s isn't reached by this BFS. We initialize $d(s)$ by setting $d(s) = |V|$.

3. **Main loop:**
   As long as the queue containing the active nodes isn't empty pop the first node v from the front of the queue and apply a push/relabel-operation to v.

The push/relabel-operation for a node v is defined as follows:

1. While $e(v) > 0$ and the residual network G' contains an edge that is legal with respect to the current distance function (i.e., there is an edge $e' = (v,w)$ having $d(v) = d(w) + 1$) apply a push:

Push: Push $\delta = \min\{e(v), c'(e')\}$ amount of flow from v to w. More precisely, if e' is a forward edge with respect to an edge e of G, increase the preflow $\tilde{f}(e)$ of e by δ, and if e' is a backward edge, then decrease the preflow $\tilde{f}(e)$ of e by δ. By doing so, $e(v)$ is decreased by δ, and $e(w)$ is increased by δ. If $w \neq s, t$, and if w wasn't active before, w becomes active, and is added at the end of the queue containing the active nodes.

2. If v has still excess, i.e., $e(v) > 0$, and if there is no legal edge emanating from v in the residual network, then apply a relabel as described in the following, and add v at the end of the queue after that.

   Relabel: Increase the distance $d(v)$ of v to the value

   $$\min\{d(w) + 1 \mid (v, w) \text{ is edge of the residual network } G'\}.$$

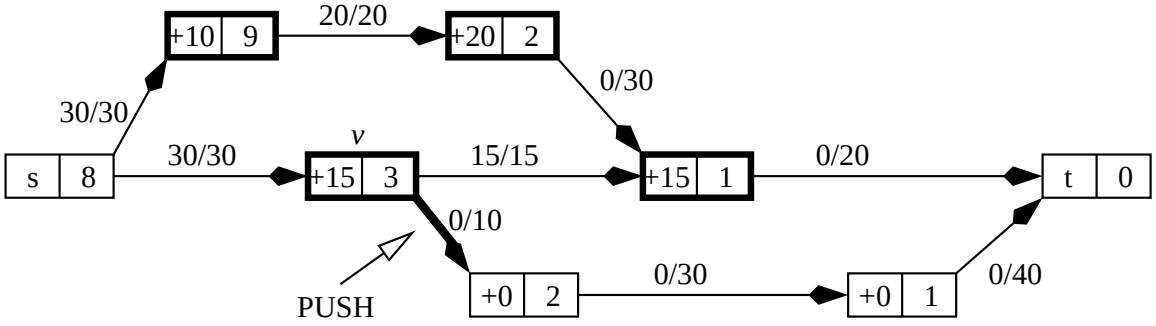
Keep in mind that also the residual network must be updated when a push-operation is applied.

The algorithm terminates when the queue containing the active nodes is empty. When that happens, the preflow is also a feasible maximum flow since the residual network doesn't contain augmenting pathes from s to t (since $d(s) = |V|$). The proof of the running time $O(|V|^3)$ is more complicated and shall not be explained here. It should be mentioned, though, that currently from practical as well as from theoretical aspects certain modifications of the algorithm of Goldberg and Tarjan (e.g., by choosing the order of active nodes which we apply push- and relabel-operations to) are the best known flow algorithms.

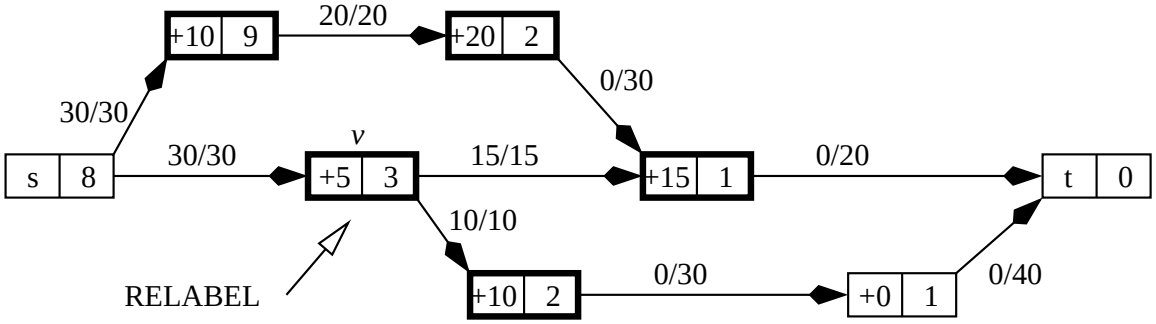
Literature: Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin: *Network Flows.* Prentice Hall, 1993, S. 207 ff.

Example of a push/relabel-operation

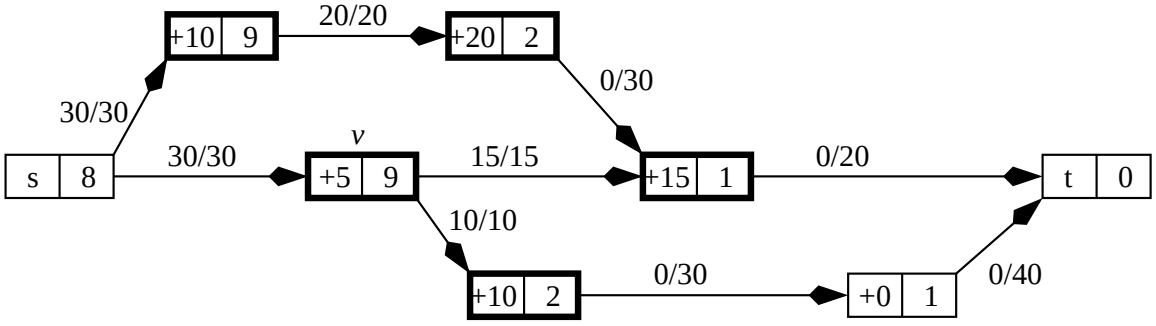
We want to illustrate a push/relabel-operation. Consider the following graph together with the preflow described by the edge labels. Active nodes are bold-framed. Within the nodes (with the exception of the source s and the sink t) the excess is given on the left, and the distance value on the right. The edge labels are $\tilde{f}(e)/c(e)$ formatwise.



A push/relabel-operation is applied to the node v using the fat edge e. The forward edge e' of the residual network which represents the edge e is legal since the difference of the distance values of the end nodes is exactly 1. Therefore, $\min\{c'(e'), e(v)\} = \min\{10, 15\} = 10$ units of flow are pushed over e. After the push we obtain the following situation.



Since the residual network doesn't contain a legal edge emanating from v anymore, the distance value of v is appropriately increased (relabel). The only edge emanating from v in the residual network leads to s which has distance $d(s) = 8$. Therefore, the distance $d(v)$ of v is set to 9.



After that, v is added at the end of the queue. When later on v is again subject to a push/relabel-operation, then its excess can be pushed back to the source.