## 14.2 Relabel to Front

**Algorithm 50** relabel-to-front($G, s, t$)

1: initialize preflow
2: initialize node list $L$ containing $V \setminus \{s, t\}$ in any order
3: **foreach** $u \in V \setminus \{s, t\}$ **do**
4:     $u.current\text{-}neighbour \leftarrow u.neighbour\text{-}list\text{-}head$
5: $u \leftarrow L.head$
6: **while** $u \neq$ null **do**
7:     $old\text{-}height \leftarrow \ell(u)$
8:     discharge($u$)
9:     **if** $\ell(u) > old\text{-}height$ **then** // relabel happened
10:         move $u$ to the front of $L$
11:     $u \leftarrow u.next$

# 14.2 Relabel to Front

**Lemma 1 (Invariant)**

*In Line 6 of the relabel-to-front algorithm the following invariant holds.*

1. *The sequence $L$ is topologically sorted w.r.t. the set of admissable edges; this means for an admissable edge $(x, y)$ the node $x$ appears before $y$ in sequence $L$.*

2. *No node before $u$ in the list $L$ is active.*

**Proof:**

- ▶ Initialization:
    1. In the beginning $s$ has label $n \geq 2$, and all other nodes have label $0$. Hence, no edge is admissable, which means that any ordering $L$ is permitted.
    2. We start with $u$ being the head of the list; hence no node before $u$ can be active

- ▶ Maintenance:
    1. 
        - ▶ Pushes do no create any new admissable edges. Therefore, if discharge() does not relabel $u$, $L$ is still topologically sorted.
        - ▶ After relabeling, $u$ cannot have admissable incoming edges as such an edge $(x, u)$ would have had a difference $\ell(x) - \ell(u) \geq 2$ before the re-labeling (such edges do not exist in the residual graph).
        Hence, moving $u$ to the front does not violate the sorting property for any edge; however it fixes this property for all admissable edges leaving $u$ that were generated by the relabeling.

# 14.2 Relabel to Front

**Proof:**

▶ Maintenance:

    **2.** If we do a relabel there is nothing to prove because the only node before $u'$ ($u$ in the next iteration) will be the current $u$; the discharge($u$) operation only terminates when $u$ is not active anymore.

    For the case that we do not relabel, observe that the only way a predecessor could be active is that we push flow to it via an admissable arc. However, all admissable arc point to successors of $u$.

Note that the invariant means that for $u =$ null we have a preflow with a valid labelling that does not have active nodes. This means we have a maximum flow.

# 14.2 Relabel to Front

**Lemma 2**

*There are at most $\mathcal{O}(n^3)$ calls to discharge(u).*

Every discharge operation without a relabel advances $u$ (the current node within list $L$). Hence, if we have $n$ discharge operations without a relabel we have $u = \text{null}$ and the algorithm terminates.

Therefore, the number of calls to discharge is at most $n(\#relabels + 1) = \mathcal{O}(n^3)$.

# 14.2 Relabel to Front

**Lemma 3**

*The cost for all relabel-operations is only $\mathcal{O}(n^2)$.*

A relabel-operation at a node is constant time (increasing the label and resetting $u.current\text{-}neighbour$). In total we have $\mathcal{O}(n^2)$ relabel-operations.

# 14.2 Relabel to Front

Note that by definition a saturing push operation $(\min\{c_f(e), f(u)\} = c_f(e))$ can at the same time be a non-saturating push operation $(\min\{c_f(e), f(u)\} = f(u))$.

**Lemma 4**

*The cost for all saturating push-operations that are **not** also non-saturating push-operations is only $\mathcal{O}(mn)$.*

Note that such a push-operation leaves the node $u$ active but makes the edge $e$ disappear from the residual graph. Therefore the push-operation is immediately followed by an increase of the pointer $u.current\text{-}neighbour$.

This pointer can traverse the neighbour-list at most $\mathcal{O}(n)$ times (upper bound on number of relabels) and the neighbour-list has only $degree(u) + 1$ many entries (+1 for null-entry).

# 14.2 Relabel to Front

**Lemma 5**

*The cost for all non-saturating push-operations is only $\mathcal{O}(n^3)$.*

A non-saturating push-operation takes constant time and ends the current call to discharge(). Hence, there are only $\mathcal{O}(n^3)$ such operations.

**Theorem 6**

*The push-relabel algorithm with the rule relabel-to-front takes time $\mathcal{O}(n^3)$.*