13.2 Relabel to Front

Algorithm 21 relabel-to-front\( (G, s, t) \)

1: initialize preflow  
2: initialize node list \( L \) containing \( V \setminus \{s, t\} \) in any order  
3: \textbf{foreach} \( u \in V \setminus \{s, t\} \) \textbf{do}  
4: \hspace{1em} \( u.\text{current-neighbour} \leftarrow u.\text{neighbour-list-head} \)  
5: \hspace{1em} \( u \leftarrow L.\text{head} \)  
6: \hspace{1em} \textbf{while} \( u \neq \text{null} \) \textbf{do}  
7: \hspace{2em} \( \text{old-height} \leftarrow \ell(u) \)  
8: \hspace{2em} \text{discharge}(u) \)  
9: \hspace{2em} \textbf{if} \( \ell(u) > \text{old-height} \) \textbf{then} // relabel happened  
10: \hspace{3em} \text{move} \( u \) to the front of \( L \)  
11: \hspace{1em} \( u \leftarrow u.\text{next} \)
13.2 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 admissible edges; this means for an admissible 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 admissible, 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 admissible edges. Therefore, if \( \text{discharge()} \) does not relabel \( u, L \) is still topologically sorted.
  2. After relabeling, \( u \) cannot have admissible 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 admissible edges leaving \( u \) that were generated by the relabeling.
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 admissible arc. However, all admissible arc point to successors of $u$.

Note that the invariant means that for $u = \text{null}$ we have a preflow with a valid labelling that does not have active nodes. This means we have a maximum flow.
Lemma 2

There are at most $O(n^3)$ 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 $n(#\text{relabels} + 1) = O(n^3)$. 
Lemma 3

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

A relabel-operation at a node is constant time (increasing the label and resetting $u$.*current-neighbour*). In total we have $O(n^2)$ relabel-operations.
13.2 Relabel to Front

Note that by definition a saturating 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 \( O(mn) \).*

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 \( u.current-neighbour \).

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

Lemma 5
The cost for all non-saturating push-operations is only $O(n^3)$.

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

Theorem 6
The push-relabel algorithm with the rule relabel-to-front takes time $O(n^3)$.