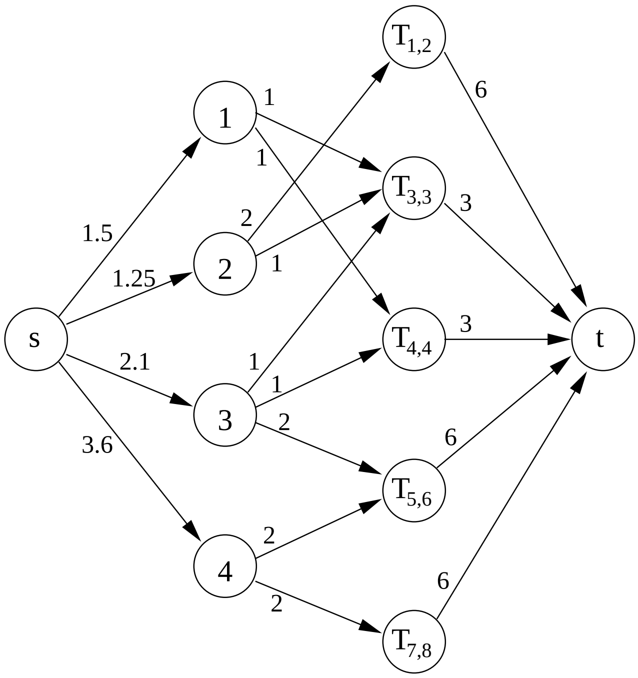
Technische Universität München
Department of Computer Science
Chair for Efficient Algorithms
Prof. Dr. Sami Khuri

WS 1999/2000
Solutions 4
June 13, 2000

Efficient Algorithms and Data Structures II

Problem 1 (10 credits)

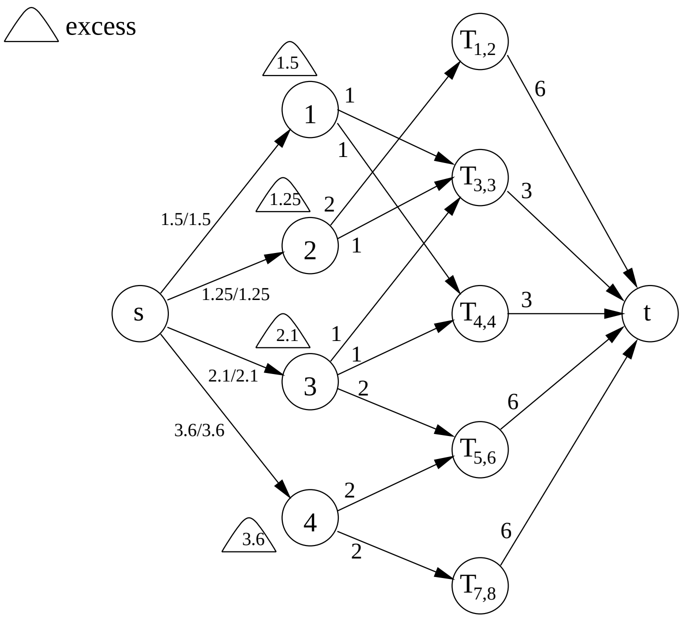
Use the FIFO preflow-push algorithm to produce a maximum flow for the following network.



Recall that this is the network we obtain by casting the problem instance of "Scheduling on uniform parallel machines" that we studied in class into a maximum flow problem. In the diagram, we have 4 jobs with processing times 1.5, 1.25, 2.1, and 3.6, and $T_{k,\ell}$ denotes the interval that starts at the beginning of day k and ends at the beginning of day $\ell + 1$.

Solution.

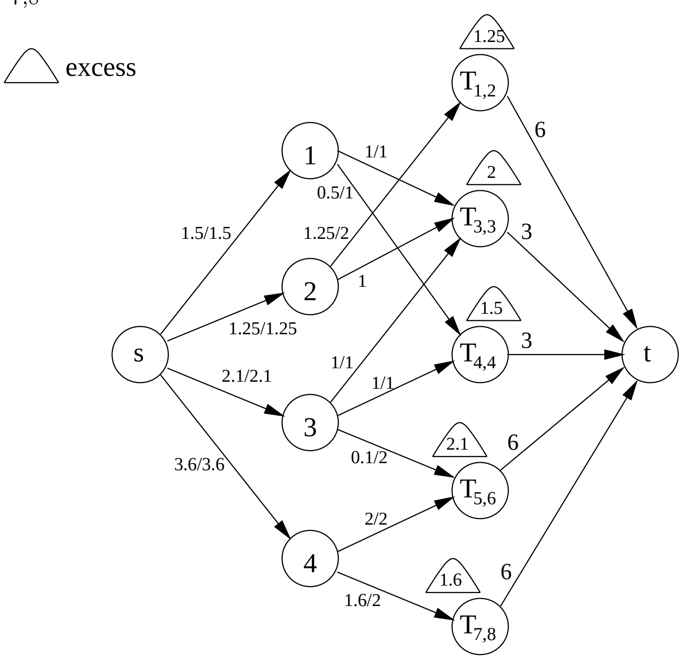
After the initialization, the network looks as follows.



Now we perform the following operations.

1. Push 1 unit from 1 to $T_{3,3}$ (saturating). Activate $T_{3,3}$.
2. Push 0.5 units from 1 to $T_{4,4}$ (nonsaturating). Activate $T_{4,4}$. Deactivate 1.
3. Push 1.25 units from 2 to $T_{1,2}$ (nonsaturating). Activate $T_{1,2}$. Deactivate 2.
4. Push 1 unit from 3 to $T_{3,3}$ (saturating).
5. Push 1 unit from 3 to $T_{4,4}$ (saturating).
6. Push 0.1 units from 3 to $T_{5,6}$ (nonsaturating). Activate $T_{5,6}$. Deactivate 3.
7. Push 2 units from 4 to $T_{5,6}$ (saturating).
8. Push 1.6 units from 4 to $T_{7,8}$ (nonsaturating). Activate $T_{7,8}$. Deactivate 4.

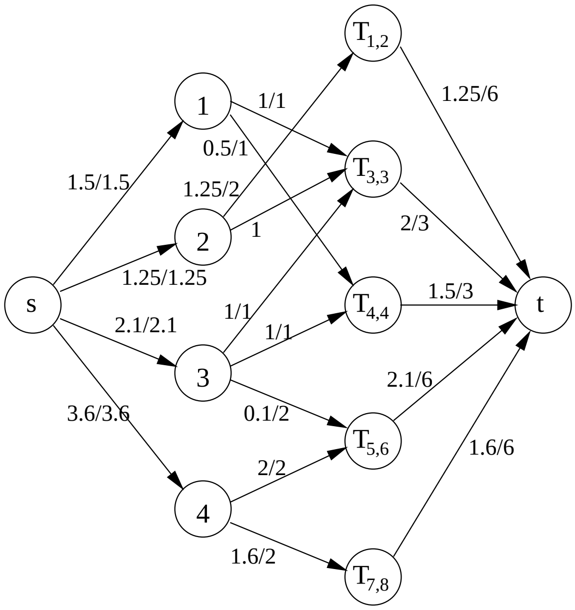
Now, the network looks as follows. Note that nodes have been activated in the order $T_{3,3}$, $T_{4,4}$, $T_{1,2}$, $T_{5,6}$, and $T_{7,8}$.



Finally, we can perform the following operations.

1. Push 2 units from $T_{3,3}$ to t (nonsaturating). Deactivate $T_{3,3}$.
2. Push 1.5 units from $T_{4,4}$ to t (nonsaturating). Deactivate $T_{4,4}$.
3. Push 1.25 units from $T_{1,2}$ to t (nonsaturating). Deactivate $T_{1,2}$.
4. Push 2.1 units from $T_{5,6}$ to t (nonsaturating). Deactivate $T_{5,6}$.
5. Push 1.6 units from $T_{7,8}$ to t (nonsaturating). Deactivate $T_{7,8}$.

Since there no longer is any active node, the algorithm terminates. The outcome can be found in the following figure.



Problem 2 (10 credits)

If $G = (V, E)$ is a flow network with source s and sink t, f is a preflow in G, and h is a height function, then we say that (u, v) is an *admissible* edge if $c_f(u, v) > 0$, and $h(u) = h(v) + 1$. Otherwise, (u, v) is *inadmissible*. The *admissible network* is $G_{f,h} = (V, E_{f,h})$, where $E_{f,h}$ is the set of admissible edges. Show that $G_{f,h}$ is acyclic.

Solution.

Proof by contradiction. Assume that $G_{f,h}$ contains a cycle $C = (v_0, v_1, \ldots, v_k)$ with $v_0 = v_k$ for some $k > 0$. Since every edge in $G_{f,h}$ is admissible, we have $h(v_{i-1}) = h(v_i) + 1$ for $i = 1, \ldots, k$. Hence,

$$\sum_{i=1}^{k} h(v_{i-1}) = \sum_{i=1}^{k} (h(v_i) + 1) = k + \sum_{i=1}^{k} h(v_i).$$

We sum over the same nodes on RHS and LHS (recall that $v_0 = v_k$), hence we have a contradiction $0 = k$.

Problem 3 (20 credits)

The Lift-To-Front algorithm administers a chained list of nodes from $V - \{s, t\}$ in topological order w.r.t. $G_{f,h}$. Show that the following algorithm computes a maximum flow in time $O(V^3)$.

```
LIFT-TO-FRONT(G, c, s, t)
    INITIALIZE-PREFLOW(G, s)
    L ← V − {s, t} in topological order
    for all v ∈ V − {s, t}
        current[v] ← first node of v's adjacency list
    u ←HEAD(L)
    while u ≠NIL
        old_height ← h(u)
        DISCHARGE(u)
        if h(u) > old_height then
            move u to the front of list L
        u ←NEXT(L, u)
```

Note that INITIALIZE-PREFLOW(G, s) here is similar to the initialization phase of the preflow-push algorithm.

Solution.

We have to prove two things, namely that (a) the algorithm actually computes a maximum flow, and (b) that the running time can be bounded by $O(V^3)$.

Ad (a). First we note that DISCHARGE (as given in class) performs PUSH or LIFT operations only if they are allowed. Hence, LIFT-TO-FRONT is an implementation of the generic algorithm. It remains to show that after the termination of LIFT-TO-FRONT, there are no further operations applicable.
Note that if u reaches the end of list L, then within DISCHARGE no RELABEL operation has been performed on the nodes in L. If additionally the nodes in L constitute a topological order of the nodes of the admissible network $G_{f,h}$, then flow is pushed to nodes closer to the tail of the list. It remains to show that L indeed contains the nodes of $G_{f,h}$ in topological order.

After the initialization, we have $h(s) = |V| \geq 2$, and for all other nodes $v \in V - \{s\}$, we have $h(v) = 0$. Hence, there are no edges in $G_{f,h}$, and every order on $V - \{s\}$ is a topological order. Now consider the operations PUSH and RELABEL, changing the order of nodes within the operation DISCHARGE.
Let v be a node with excess flow, and let (v, w) be an admissible edge. Then, operation PUSH(v, w) can be applied. This does not produce new admissible edges, and edge (v, w) can no longer be admissible after this operation.
Now, let v again be some node with excess flow, but now there are no outgoing admissible edges. Then, operation RELABEL can be applied on v. After calling RELABEL, there is at least one admissible edge going out of v, but no admissible edge is directed to v. This can be seen as follows. Since no PUSH operation is applicable on v, there must be a RELABEL operation applicable on v (recall the proof given in class). Hence,

$$h(v) = 1 + \min\{h(w) \mid w \text{ is neighbor of } v \text{ in the residual network}\},$$

and for nodes w' actually taking this minimum, edges (v, w') are in $G_{f,h}$. It remains to show that no admissible edges are directed to v in $G_{f,h}$. We prove the claim by contradiction. Assume that the *is* a node u, where (u, v) is an admissible edge in $G_{f,h}$. Then before the RELABEL operation, $h(u) > h(v) + 1$. This is impossible, since in the residual network, there are edges only between nodes with height difference exactly one.

It now follows that after operation DISCHARGE the list L again constitutes a topological order of the nodes of $G_{f,h}$.

Ad (b). We divide the performance of the algorithm into phases, where a phase consists of the operations performed between two consecutive RELABEL operations. We know from class that there are at most $O(|V|^2)$ RELABEL operations. Furthermore, during a phase, at most $O(|V|)$ DISCHARGE operations are performed, since within a phase, by definition no RELABEL operations are performed, and hence each call to DISCHARGE is directed to a node closer to the tail of list L (which has a length of $O(|V|)$. Aside from the work being done within DISCHARGE, we now have an $O(|V|^3)$ bound on the running time of the **while** loop. We now analyze the work being done for operation DISCHARGE.

1. The work being done for the $O(|V|^2)$ DISCHARGE operations is bounded by $O(|V||E|)$.

2. For every node v, the updating of the pointer to the next current neighbor between two RELABEL operations happens at most $O(\text{degree}(v))$ times, and consequently during the complete algorithm happens at most $O(|V| \cdot \text{degree}(v))$ times. Summing up over all nodes, we obtain $O(|V||E|)$ operations.

3. The number of saturating PUSH operations is $O(|V||E|)$, and the number of nonsaturating PUSH operatins is bounded by the number of DISCHARGE calls, since after a nonsaturating PUSH operation DISCHARGE is left.

All in all, we obtain a running time bound of $O(|V|^3 + |V||E|) = O(|V|^3)$.