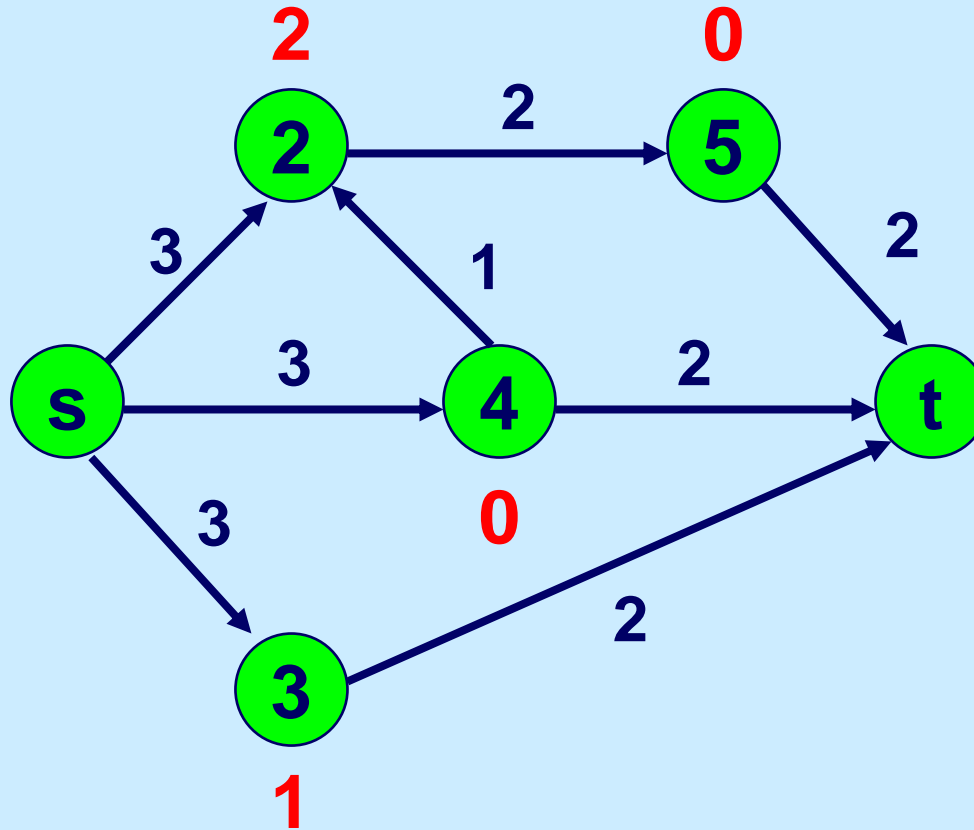

15.082 and 6.855J
March 13, 2003

Max Flows 4

Overview of today's lecture

- ◆ **Very quick review of Preflow Push Algorithm**
- ◆ **The Excess Scaling Algorithm**
 - $O(n^2 \log U)$ non-saturating pushes
 - $O(nm + n^2 \log U)$ running time.
- ◆ **A proof that Highest Preflow Push uses $O(n^2 m^{1/2})$ non-saturating pushes.**

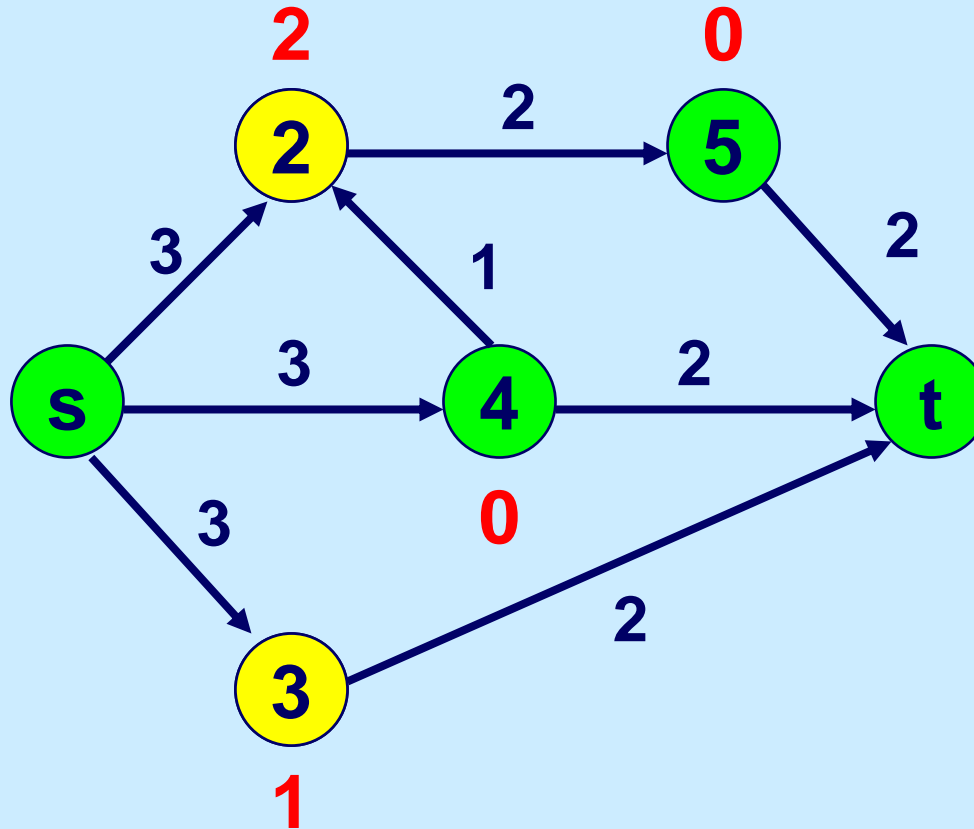
A Feasible Preflow



The excess $e(j)$ at each node $j \neq s, t$ is the flow in minus the flow out.

Note: total excess = flow out of s minus flow into t .

Active nodes



Nodes with positive excess are called **active**.

The preflow push algorithm will try to push flow from active nodes towards the sink, relying on $d(\cdot)$.

Review of Distance Labels

Distance labels $d(\cdot)$ are **valid** for $G(x)$ if

- i. $d(t) = 0$
- ii. $d(i) \leq d(j) + 1$ for each $(i,j) \in G(x)$

Defn. An arc (i,j) is **admissible** if $r_{ij} > 0$
and $d(i) = d(j) + 1$.

Lemma. Let $d(\cdot)$ be a valid distance label. Then $d(i)$ is a lower bound on the distance from i to t in the residual network.

Goldberg-Tarjan Preflow Push Algorithm

Procedure Preprocess

begin

$x := 0;$

compute the exact distance labels $d(i)$ for each node;

$x_{sj} := u_{sj}$ for each arc $(s,j) \in A(s)$; $d(s) := n$;

end

Algorithm PREFLOW-PUSH;

begin

preprocess;

while there is an active node i **do**

begin

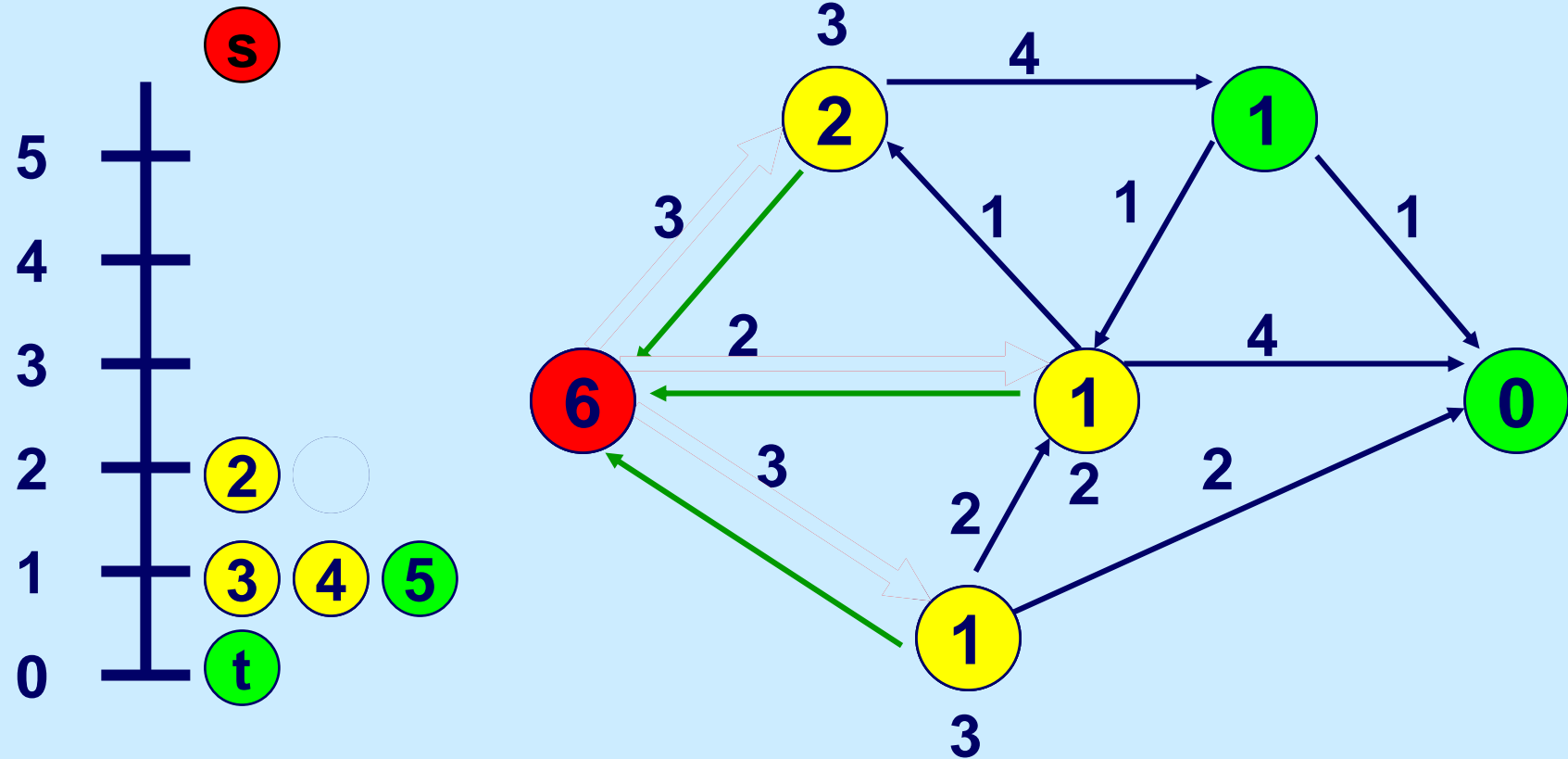
select an active node i ;

push/relabel(i);

end;

end;

Preprocess Step

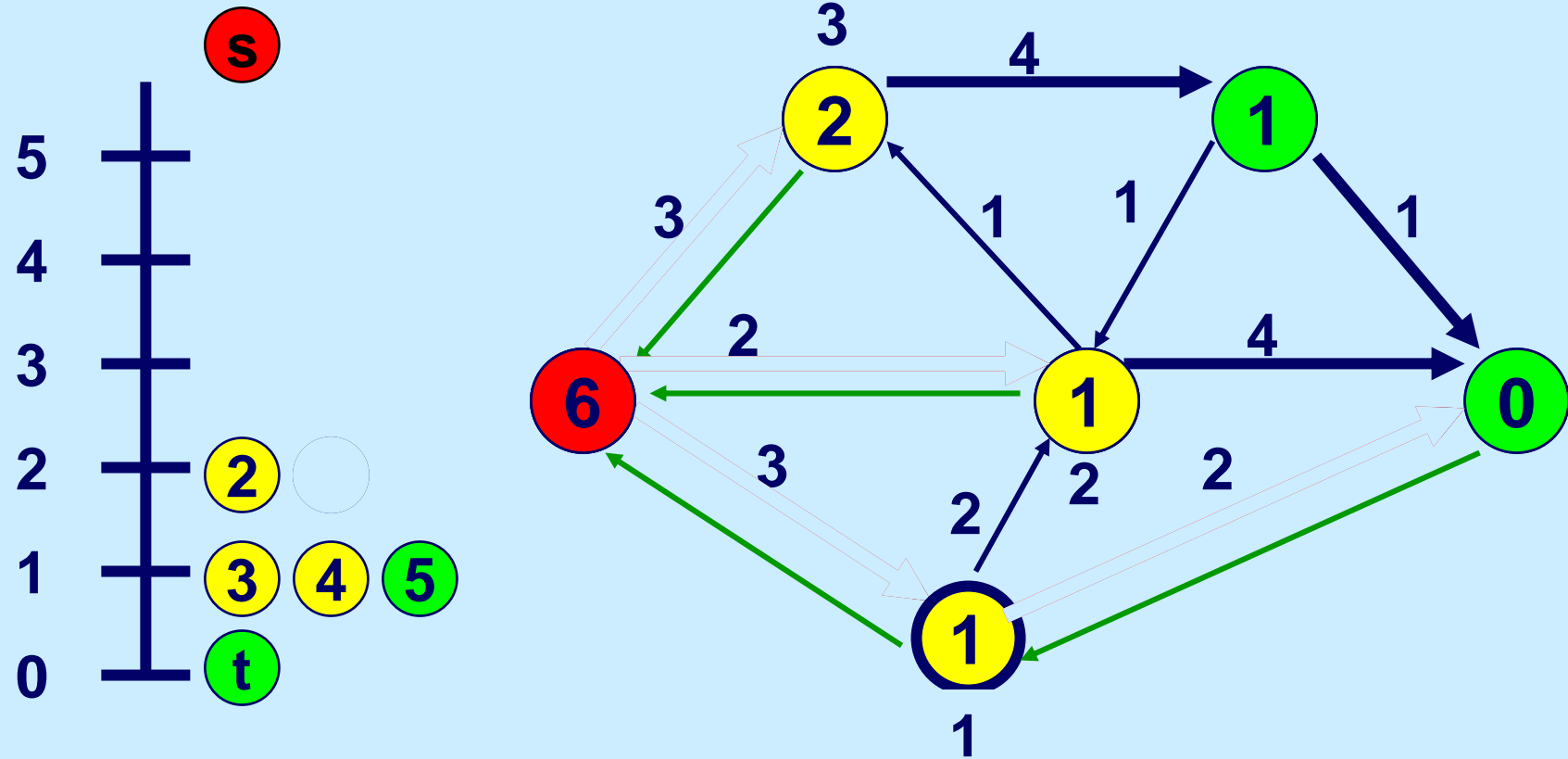


Saturate arcs out of node s .

Move excess to the adjacent arcs

Relabel node s after all incident arcs have been saturated.

Select, then relabel/push



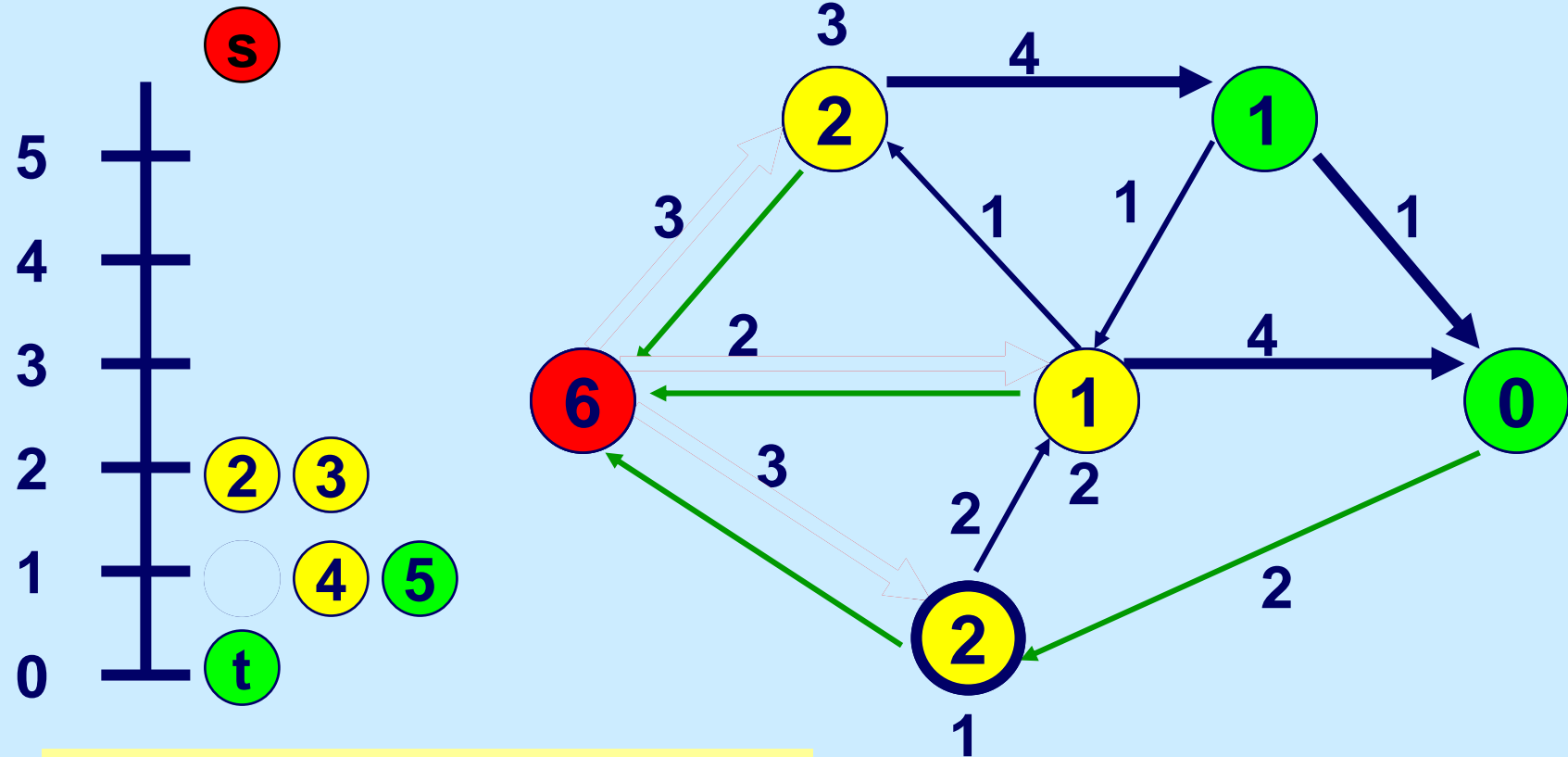
Select an active node.

Push/Relabel

Update excess after a push

Note: Each arc gets saturated $O(n)$ times.

Select, then relabel/push

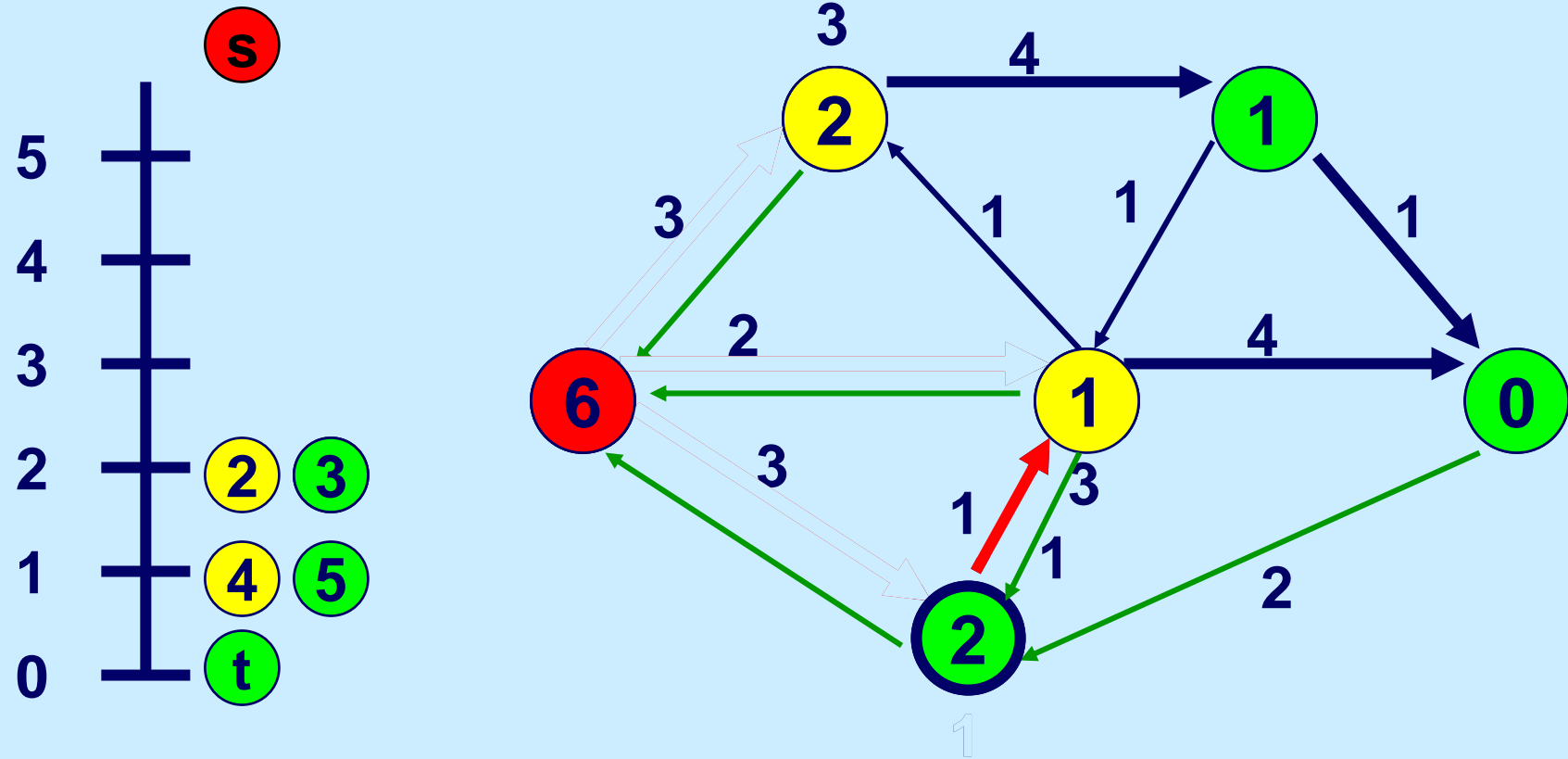


Select an active node.

No arc incident to the selected node is admissible. So relabel.

Note. Each node gets relabeled $O(n)$ times.

Select, then relabel/push



Select an active node

Push/Relabel

Bounding non-saturating pushes is more complex. Note that the active node became inactive.

Excess Scaling Approach

Let Δ be a “scaling” parameter.

In the Δ -scaling phase $e(j) \leq \Delta$ for all j .

At the end of the scaling phase $e(j) < \Delta/2$, for all j , at which point the $\Delta/2$ -scaling phase begins.

We start with $\Delta > U$. The last scaling phase is the 1-scaling phase. $e(j) \leq 1$ for all j .

At the end of the 1-scaling phase, we have a flow, and the flow is optimal.

Note: the number of phases is $O(\log U+1)$

Algorithm excess scaling

begin

preprocess

$\Delta := 2^{\lceil \log U \rceil};$

while $\Delta \geq 1$ **do**

begin

while the network contains a node j with $e(j) \geq \Delta/2$ **do**

begin

among nodes j with $e(j) \geq \Delta/2$ (called *large excess nodes*), choose i with minimum distance label

perform push/relabel(i) while ensuring that no node excess exceeds Δ ;

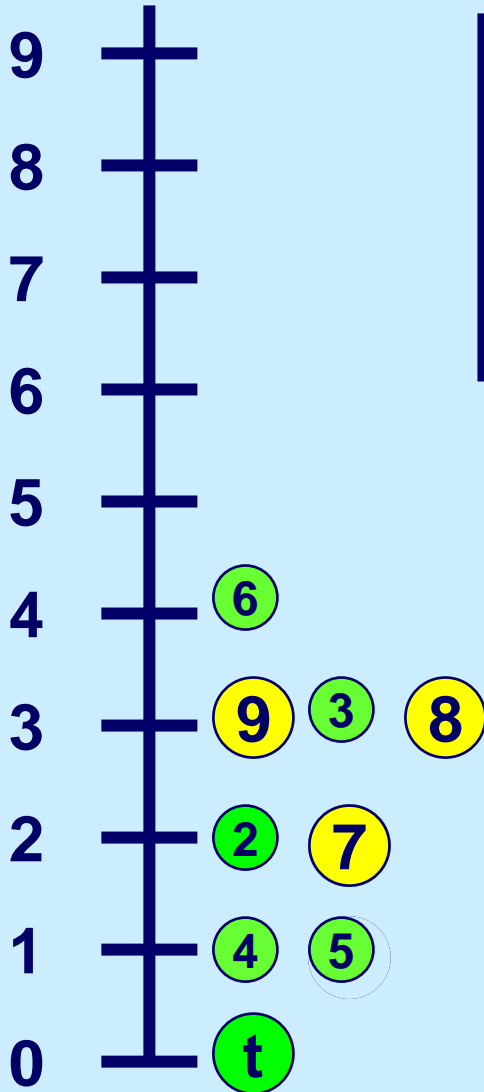
end

$\Delta := \Delta/2$

end

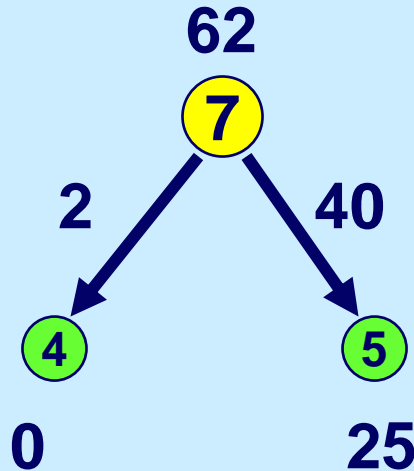
Pushing in the 64-scaling phase

S



j $32 \leq e(j) \leq 64$ **“large excess”**

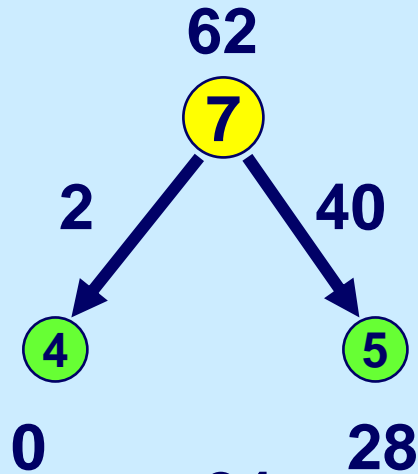
i $e(i) < 32$



Choose a large excess node *i* at minimum distance level.

For each admissible arc (*i*,*j*), *j* is not large excess.

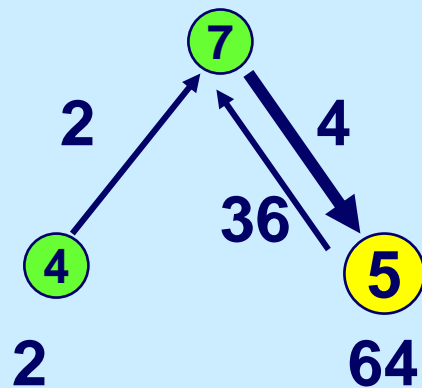
Pushing in the 64-scaling phase.



j $32 \leq e(j) \leq 64$ “large excess”

i $e(i) < 32$

Rule: push as before, except that no node excess can exceed Δ .



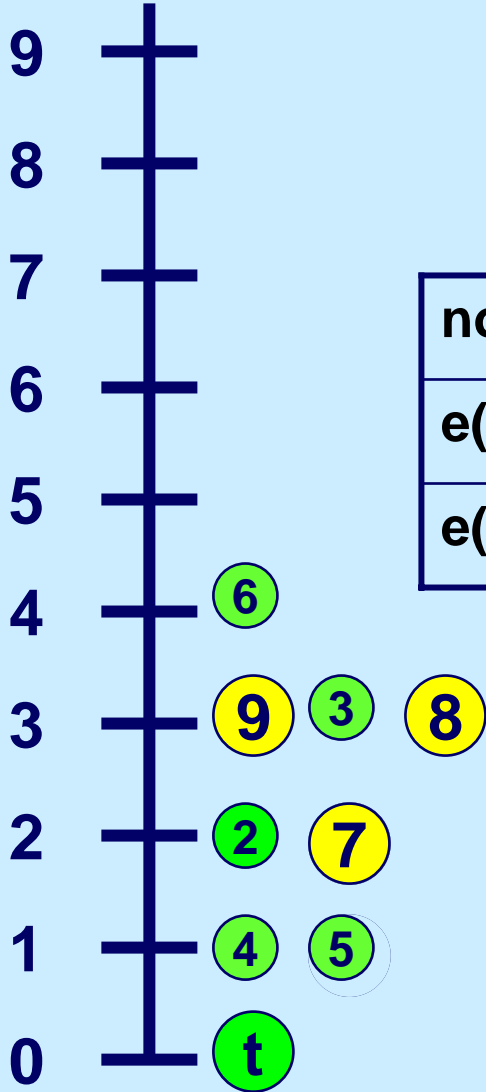
Push 2 units in (7,4).
Then push 36 units in (7,5)

Send $\min(e(i), r_{ij}, \Delta - e(j))$ units of flow in (i,j)

Any non-saturating push has at least $\Delta/2$ units of flow

A new potential function

S



$$\Phi = \sum_{j \in N} e(j) d(j) / \Delta$$

node	2	3	4	5	6	7	8	9
e(j)	4	0	1	0	5	35	55	40
e(j)d(j)	8	0	1	0	20	70	165	120

$$\Phi = 384 / 64 = 6$$

The potential is the “gravitational potential” measured in units of Δ

Deposits and Relabels

S

$$\Phi = \sum_{j \in N} e(j) d(j) / \Delta$$

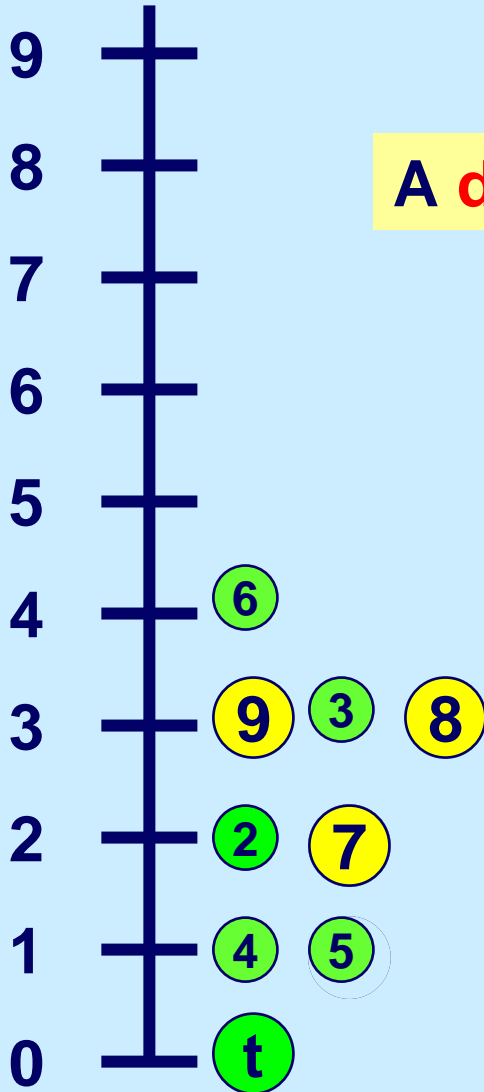
A **deposit** is an operation that increases Φ .

If 9 is relabeled then

$$\Delta/2 \leq e(9) \leq \Delta.$$

Increasing $d(9)$ from 3 to 4 increases Φ by at most 1.

The deposits due to increasing $d(9)$ over all iterations is at most $2n$. The total deposits due to relabels is $O(n^2)$



Withdrawals and Pushes

S

$$\Phi = \sum_{j \in N} e(j) d(j) / \Delta$$

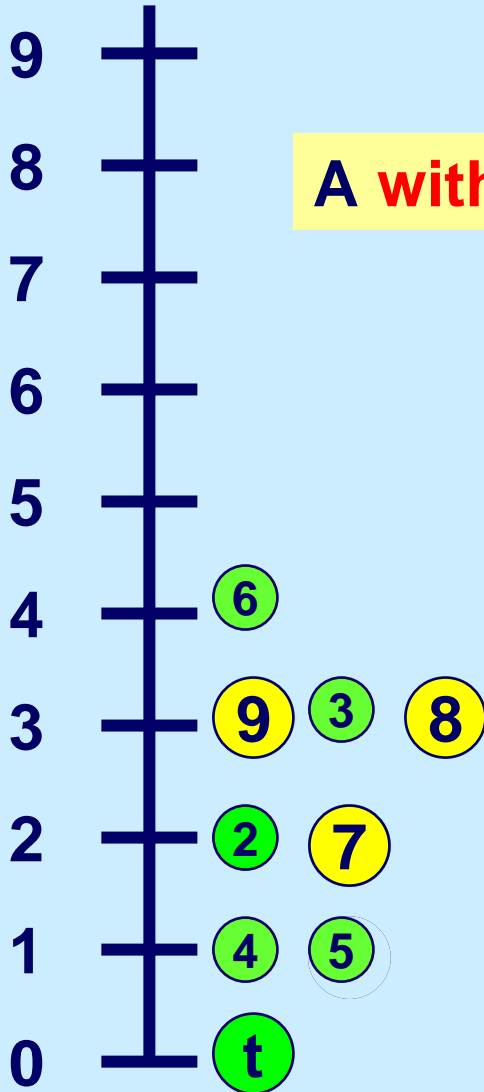
A **withdrawal** is an operation that decreases Φ .

Pushing 2 units of flow in (7,4)
decreases Φ by $2 / \Delta$.

$e(7)d(7)$ decreases by $2d(7) = 4$
 $e(4)d(4)$ increases by $2d(4) = 2$.

Every push is a withdrawal.

Each non-saturating push sends at
least $\Delta/2$ units of flow, and so
reduces Φ by at least $1/2$.



Putting it all together

$$\Phi = \sum_{j \in N} e(j) d(j) / \Delta$$

Deposits = increases in Φ .

The total deposits over all relabels is $O(n^2)$.

Withdrawals = decreases in Φ .

Each push is a withdrawal. Each non-saturating push is a withdrawal of at least $1/2$.

At the beginning of a scaling phase $\Phi < 2n^2$.

Conclusion: The number of non-saturating pushes is $O(n^2)$ during a scaling phase, and $O(n^2 \log U)$ over all scaling phases.

Running time: $O(nm + n^2 \log U)$.

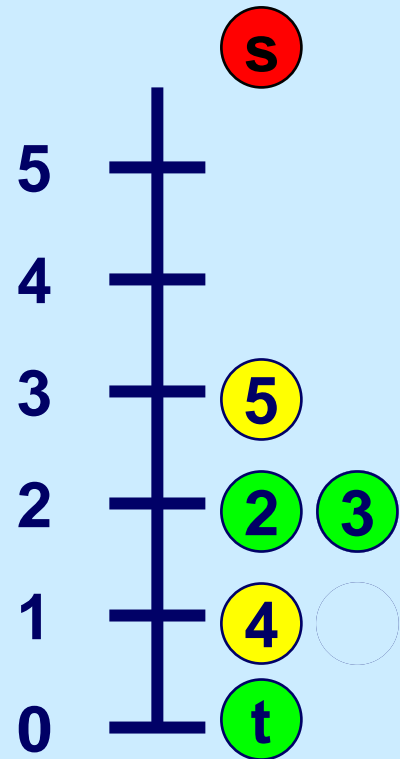
A comment on scaling

Optimality condition for preflow push: $e(j) = 0$ for all j .

In scaling techniques, we usually have the optimality conditions getting closer and closer to being satisfied.

In this case $e(j) \leq \Delta$ for all j during a Δ -scaling phase, and then Δ gets reduced by a factor of 2 at each subsequent phase.

Highest Level Pushing



In highest level pushing, we always select an active node with the highest level.

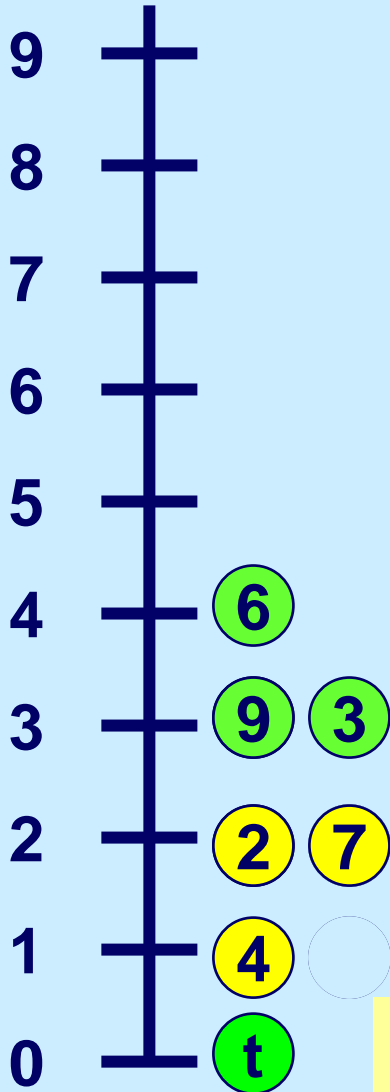
In the example to the left, we would select node 5.

Claim: if we select the active node with the highest level, the number of pushes is $O(n^2m^{1/2})$.

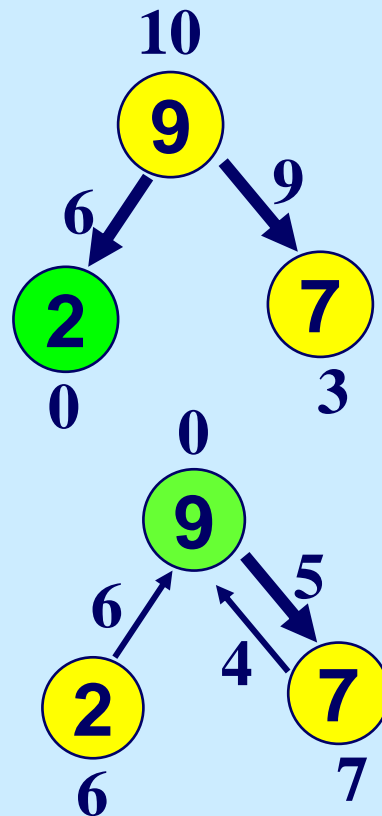
Note: The proof is due to Cheriyan and Mehlhorn, and is much easier than the proof in AMO.

More on highest level pushing

S



The algorithm selects a node from level 3. Say node 9.

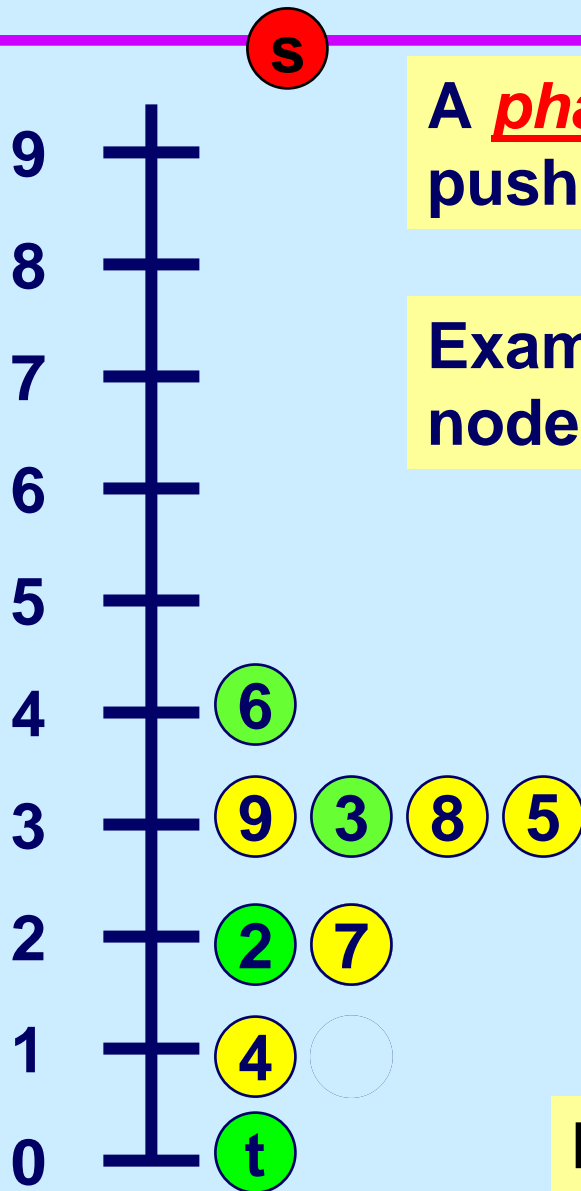


Case 1: the selected node can get rid of all of its flow through admissible arcs.

Then the selected node becomes inactive, and another node at level 3 is selected.

There is at most one non-saturating push in getting rid of flow from node 9.

Phases



A **phase** is a consecutive sequence of pushes all at the same level.

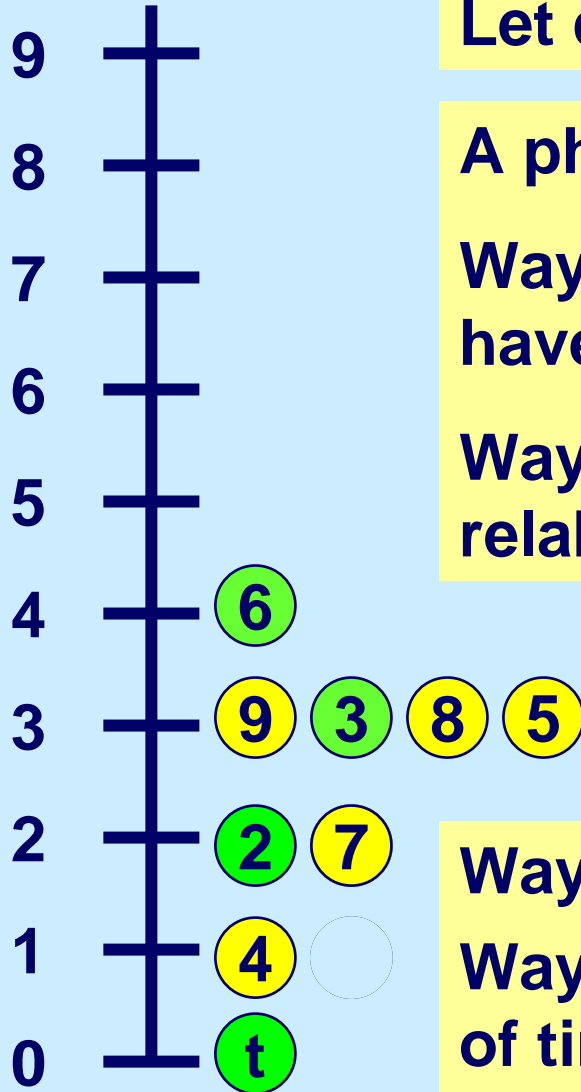
Example: we may get rid of flow from nodes 9, 8, and 5.

Note: the maximum number of non-saturating pushes during a phase is the number of active nodes at highest level at the beginning of the phase. (e.g., 3)

But: a phase can end sooner.

Lemma. The number of phases is $O(n^2)$

S



Let $d^* = \max (d(j) : j \text{ is active.})$

A phase can end in two ways:

Way 1: when all active nodes at level d^* have sent flow to level d^*-1 .

Way 2: an active node at level d^* must be relabeled, and d^* will increase.

Way 2 can happen at most $2n^2$ times.

Way 1 can happen at most $n +$ the number of times Way 2 happens. (“ d^* cannot decrease more times than it increases”).

Overview

We want to bound the number of non-saturating pushes.

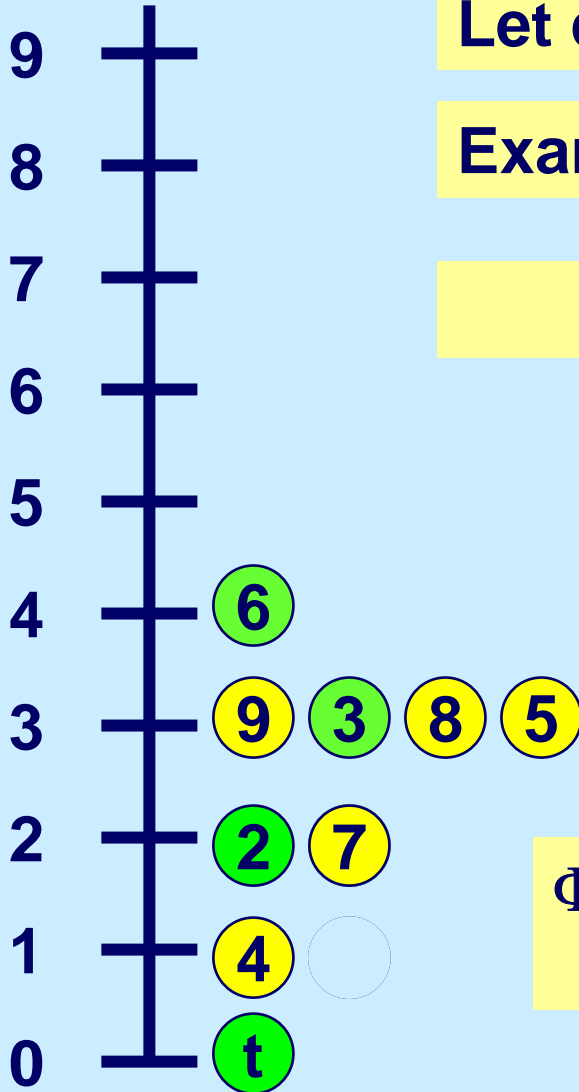
Instead, we will bound the total number of non-saturating pushes per phase.

Cheap phases: ones with fewer than $O(m^5)$ non-saturating pushes. This leads to $O(n^2m^5)$ non-saturating pushes over all phases.

Expensive phase: ones with more than $O(m^5)$ non-saturating pushes. We will use a potential function to bound non-saturating pushes in expensive phases.

A new potential function

S



Let $d'(j) = |\{i : d(i) \leq d(j)\}|$

Example: $d'(2) = |\{t, 4, 2, 7\}| = 4$

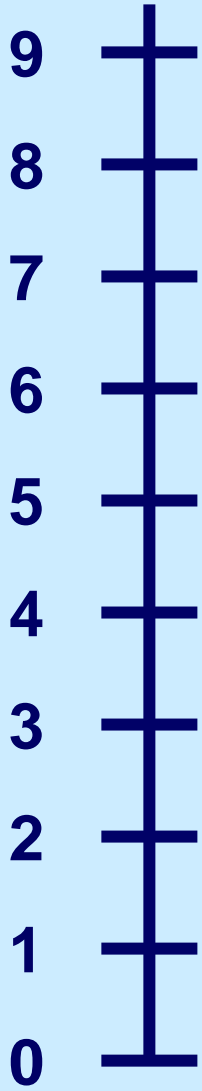
$d'(8) = |\{t, 4, 2, 7, 9, 3, 8, 5\}| = 8$

$$\Phi = \sum_{j \text{ active}} d'(j).$$

$$\begin{aligned} \Phi &= d'(4) + d'(7) + d'(9) + d'(8) + d'(5) \\ &= 2 + 4 + 8 + 8 + 8 = 30 \end{aligned}$$

Deposits and relabels

S



$$\Phi = \sum_{j \text{ active}} d'(j).$$

A **deposit** is an operation that increases Φ .

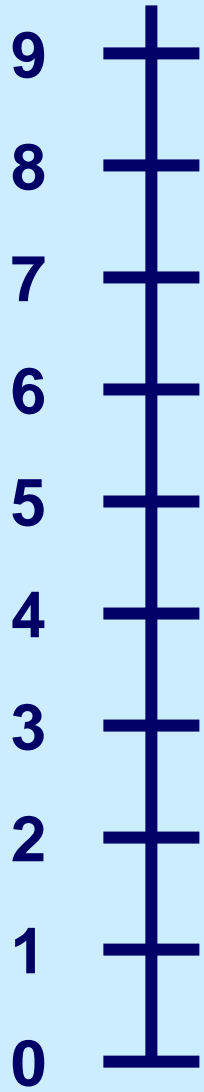
A relabel of 9 increases $d'(9)$ by 1, but it decreases $d'(3)$, $d'(8)$ and $d'(5)$.

A relabel increases Φ by less than n .

The total value of all deposits from relabels is $O(n^3)$

Deposits and saturating pushes

S



$$\Phi = \sum_{j \text{ active}} d'(j).$$

A saturating push in (9,2) makes node 2 active, and increases Φ by 4.

A saturating push increases Φ by less than n .

The total value of all deposits from saturating pushes is $O(n^2m)$

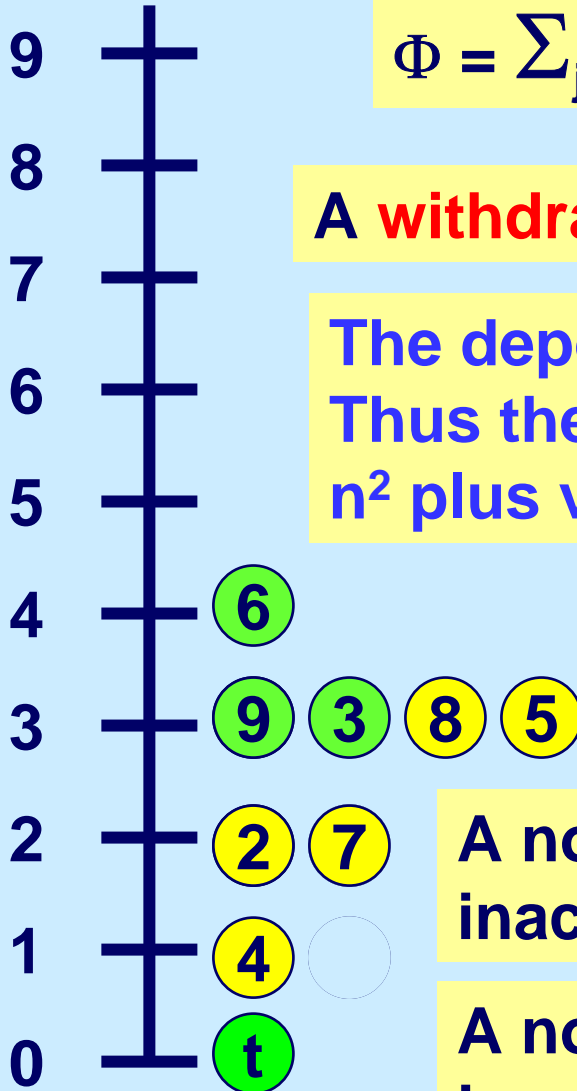
Withdrawals and non-saturating pushes

S

$$\Phi = \sum_{j \text{ active}} d'(j).$$

A **withdrawal** is an operation that decreases Φ .

The deposit from initialization is at most n^2 .
Thus the total value of withdrawals is at most n^2 plus value of deposits = $O(n^2m)$.



A non-saturating push in (9,7) makes 9 inactive, and reduces Φ by 8.

A non-saturating push in (9,2) makes 9 inactive, 2 active and reduces Φ by 4.

Withdrawals and non-saturating pushes

S

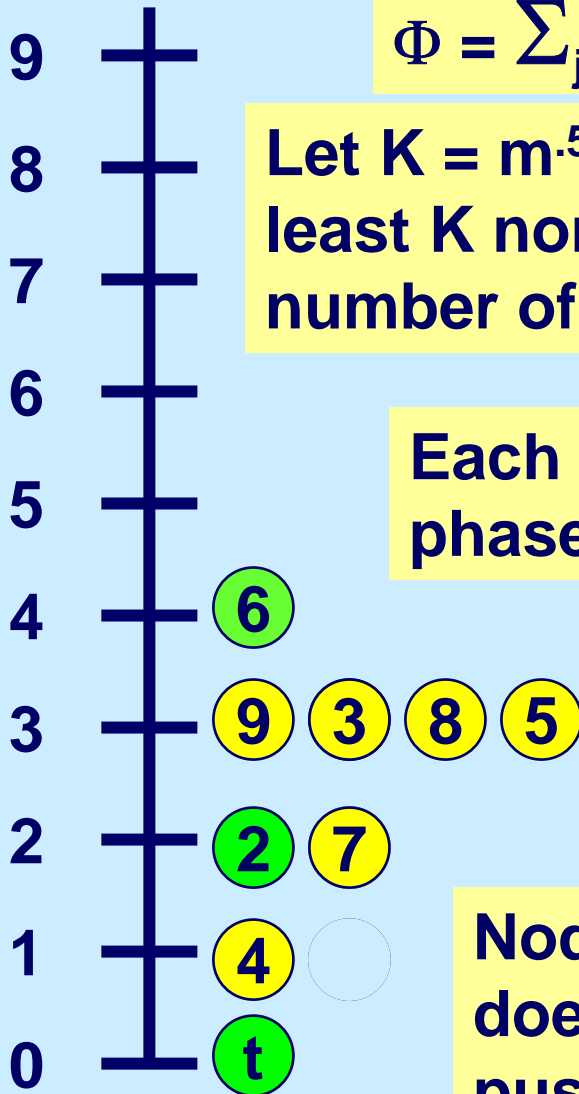
$$\Phi = \sum_{j \text{ active}} d'(j).$$

Let $K = m \cdot 5$. In an expensive phase, there are at least K non-saturating pushes, and thus the number of nodes at level d^* is at least K .

Each non-saturating push at an expensive phase is a withdrawal of at least K

Note: the descendants of 9 are 9, 3, 8, 5, 2, 7, 4, t. If 9 becomes inactive, Φ decreases by 8.

Node 2 has 4 fewer descendants than does node 9. If we do a non-saturating push in (9,2), Φ decreases by 4.



Withdrawals and non-saturating pushes

s

$$\Phi = \sum_{j \text{ active}} d'(j).$$

9

8

7

6

5

4

3

2

1

0

Suppose there are $> K$ nodes at level d^* .

Each non-saturating push decreases Φ by $> K$.

Each non-saturating push at an expensive phase is a withdrawal of at least K

6

9

3

8

5

2

7

4

t

Putting it all together

$$\Phi = \sum_{j \text{ active}} d'(j).$$

Deposits = increases in Φ .

The total deposits over all iterations is $O(n^2m)$

Withdrawals = decreases in Φ .

Each non-saturating push during an expensive phase is for at least K . The number is $O(n^2m/K)$

Each non-saturating push during a cheap phase is for at least 1. The number non-saturating pushes during a cheap phase is at most K .

The number over all phases is $O(n^2K)$.

**Number of non-saturating pushes is $O(n^2K + n^2m/K)$
 $= O(n^2m^{.5})$ if $K = m^{.5}$.**

Dynamic Trees

Dynamic trees is a clever data structure that permits one to send flow along a path of arcs, and it takes $O(\log n)$ steps to send the flow, regardless of the length of the path.

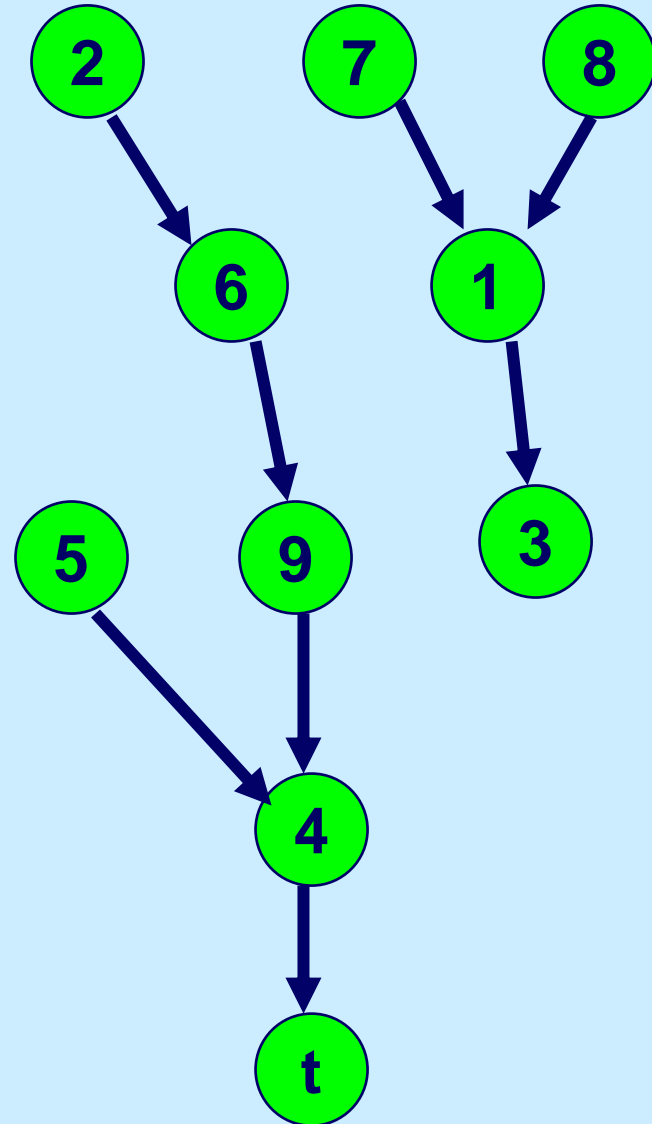
For arcs in the dynamic tree, we keep track of residual capacities implicitly, and do not store them explicitly.

We will describe dynamic trees next.

Dynamic Trees

Let us store all admissible current arcs in a dynamic tree called T .

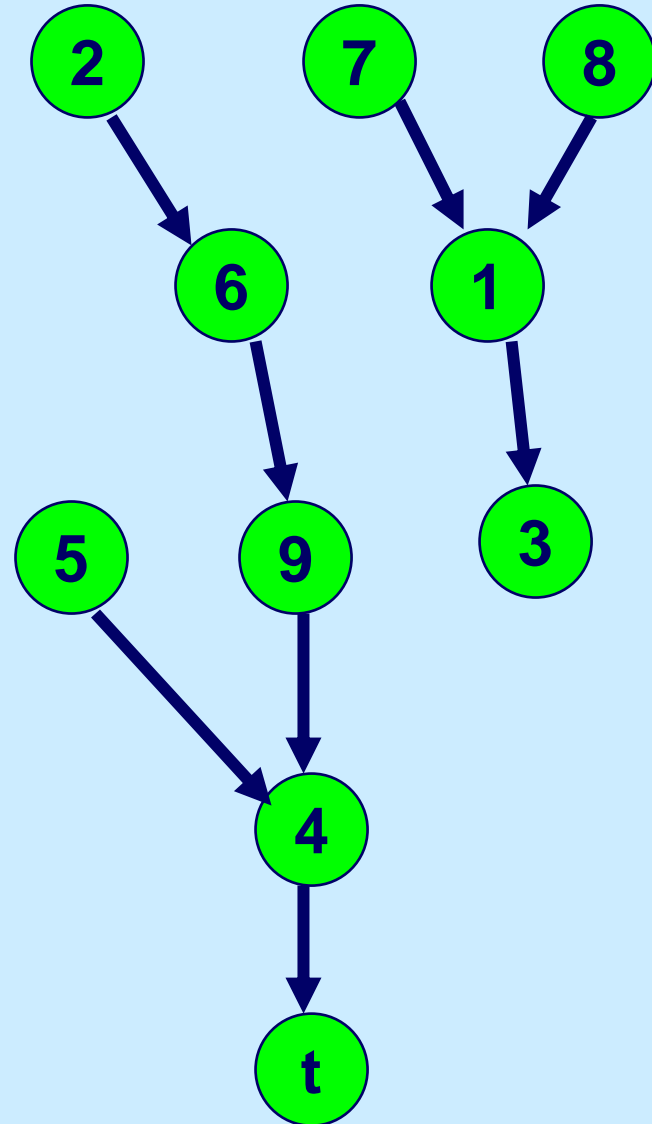
There is at most one arc directed out of each node j , and no arc out of t .



Dynamic Trees

We will permit several “operations” on T .

Each operation will take $O(\log n)$ steps if T has at most n nodes.



Root nodes

Any node with no outgoing arcs is a root node.

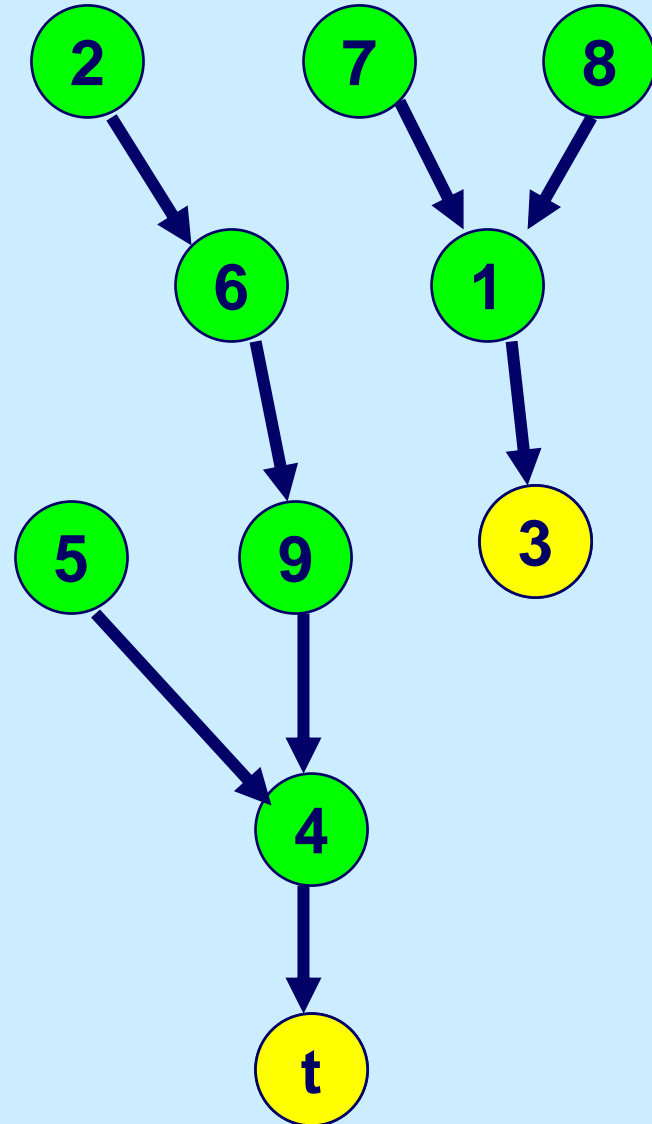
Nodes t and 3 are root nodes.

For a tree T, $\text{root}(j)$ is the root node in the subtree containing j.

Examples: $\text{root}(6) = t$
 $\text{root}(7) = 3$

Dynamic Tree Operation:

Find-root(j)



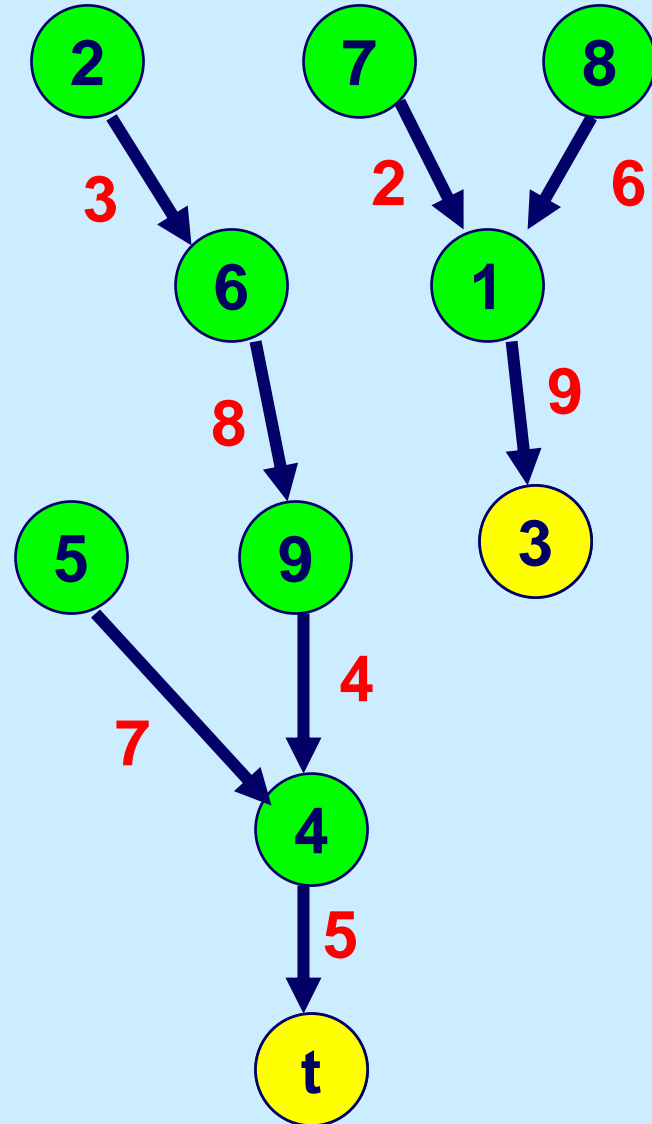
Node values

Each node of the tree has a value, which is the residual capacity of its outgoing arc. (The value of root nodes is ∞ .)

Examples. Value(4) = 5
Value(7) = 2
Value(3) = ∞

Dynamic Tree Operation:

Find-value(j)



Find min

We want to send flow along a path. For this we need to compute the residual capacity of a path.

Dynamic Tree Operation:

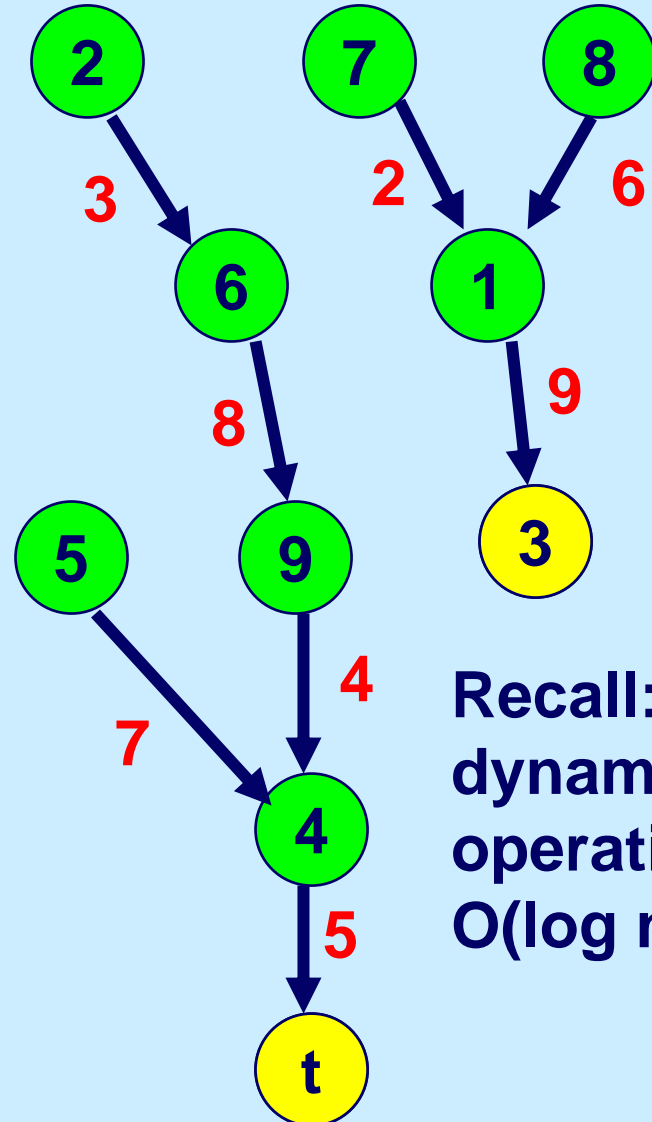
Find-min(j): returns the node on the path from j to $\text{root}(j)$ with minimum value.

Examples:

$\text{Find-min}(6) = \text{node } 9$

$\text{Find-min}(2) = \text{node } 2$

$\text{Find-min}(3) = \text{node } 3$

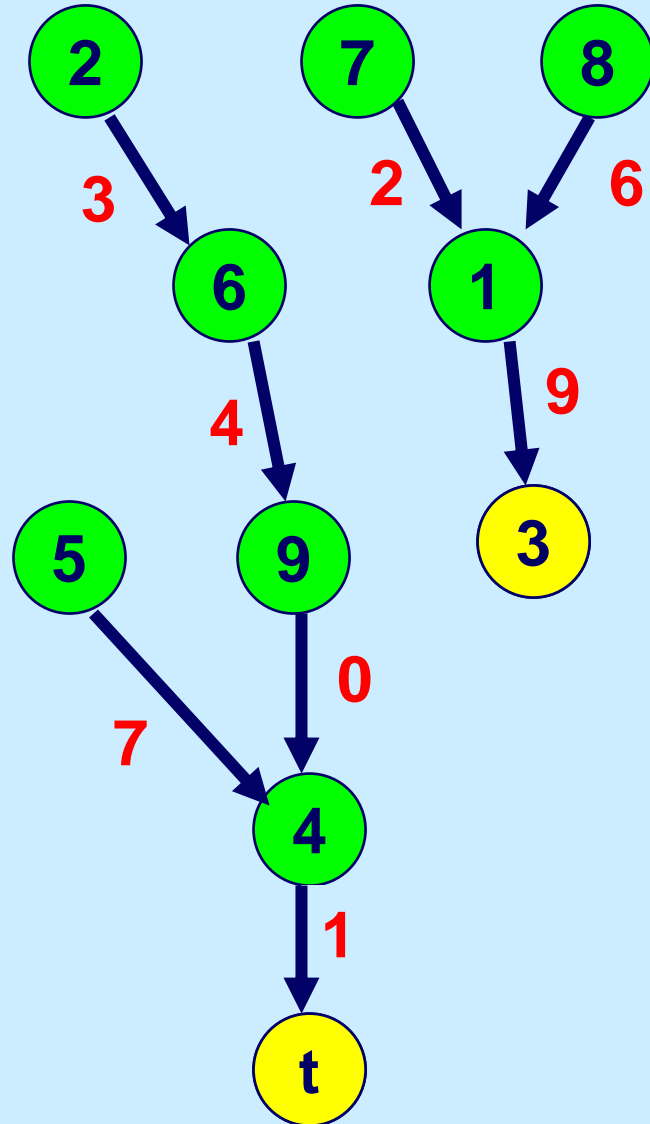


Recall: each dynamic tree operation takes $O(\log n)$ steps.

Change Value

Change-value(j, val) will subtract *val* from every node on the path from *j* to root of *j*.

Example:
Change-value(6, 4)



Cut

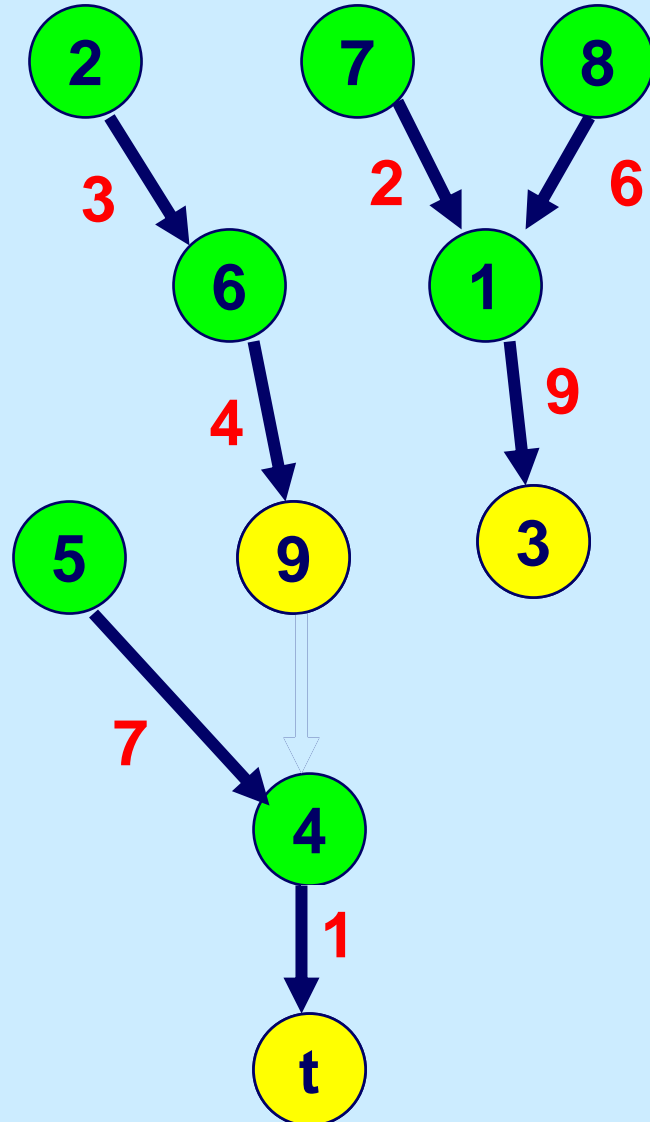
Cut(j) deletes the arc coming out of node j , and updates the root.

Example: **Cut(9)**

Examples after performing **cut(9)**:

$\text{root}(2) = 9$

$\text{find-min}(5) = \text{node } 4$



Link

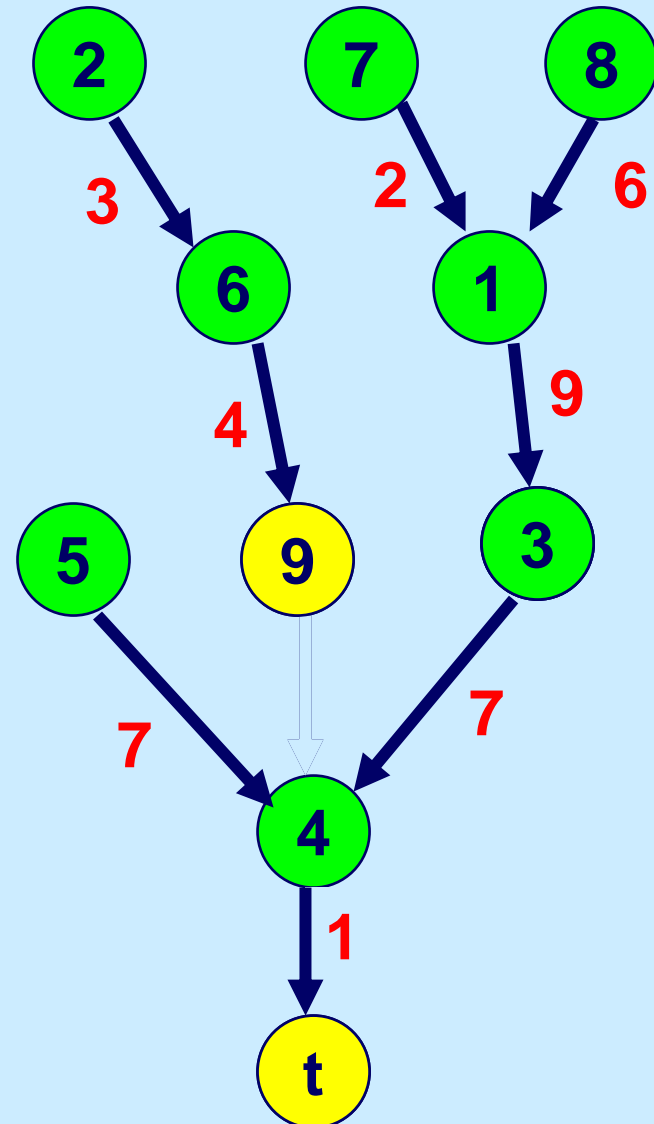
***Link*(i, j, val)** adds arc (i,j) to the dynamic tree with residual capacity val

Example: **Link**(3, 4, 7)

Examples after performing **Link**(3, 4, 7):

$\text{root}(8) = t$

$\text{find-min}(1) = \text{node } 4$



Overview of how dynamic trees are used

- ◆ **The tree will contain current arcs, so long as the current arcs are admissible.**
 - **Delete saturated arcs from T**
 - **Delete current arcs into node j after j is relabeled**
 - **Note: each arc is deleted from T at most n times, and thus added to T at most n times. Total number of cuts and links is $O(nm)$, and the time is $O(nm \log n)$.**

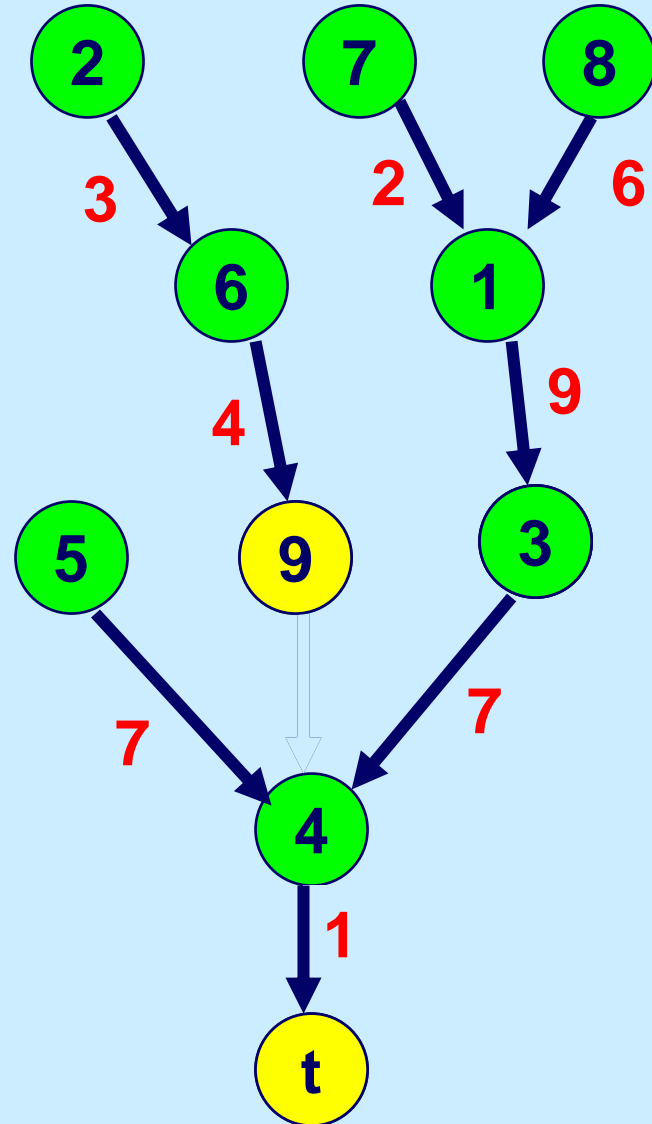
On Sending Flow

After selecting active node j , send flow from j to its root node.

Case 1: not all of the excess from node j is removed.

Example: if $e(7) = 3$ and if node 7 is selected, we can push only 1 unit to t .

This case causes a saturation, and can happen $O(nm)$ times.



On Sending Flow

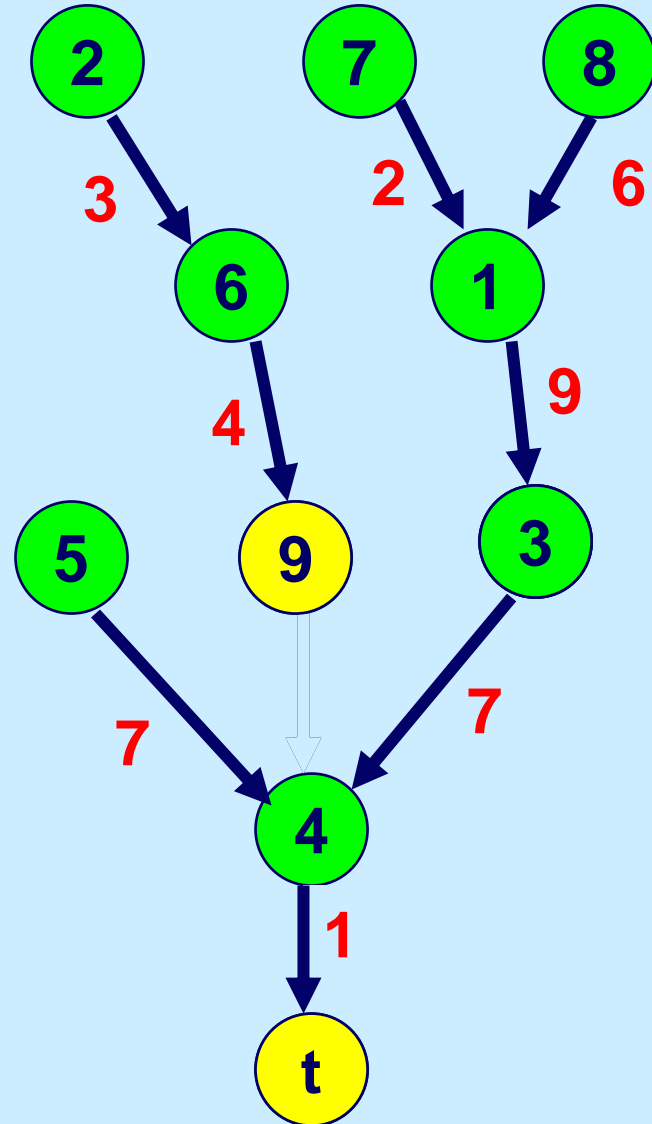
Case 2: all flow is sent to a root node j other than t .

Example: node 2 or node 6 is selected.

Next Step: select the root node j for a push/relabel.

Either j is relabeled, or a new admissible arc from j is found.

Case 2 happens $O(nm)$ times.



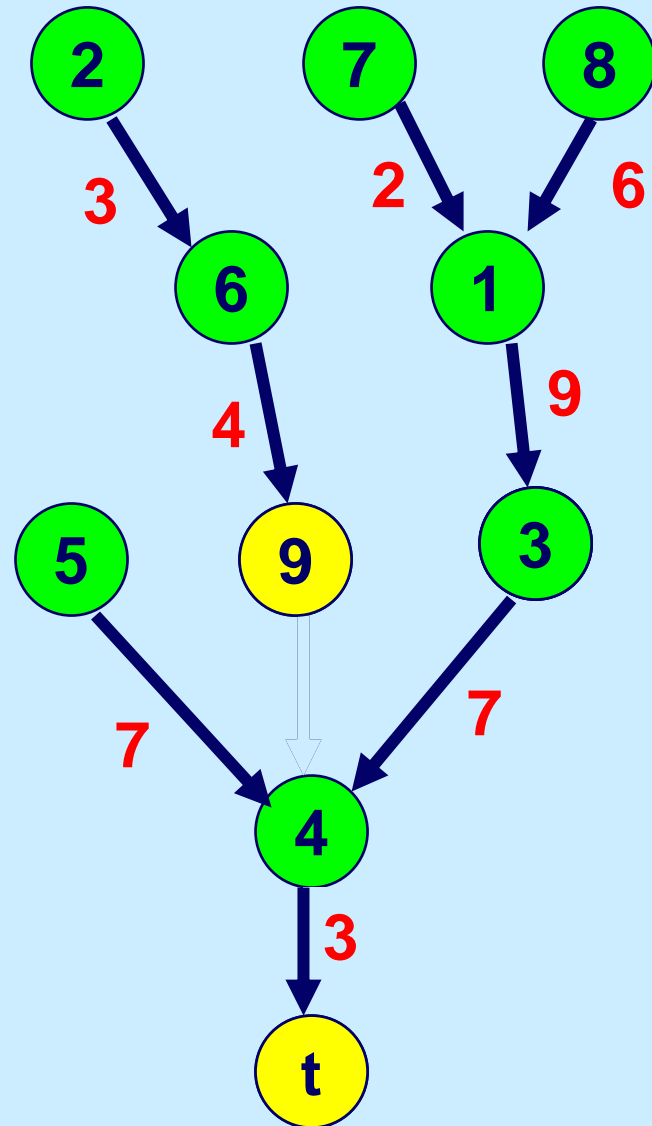
On Sending Flow

Case 3: flow is sent to t and no arc is saturated.

Example: suppose $e(3) = 2$ and node 3 is selected.

Subsequently, the number of active nodes decreases.

Case 3 can happen $O(nm)$ times, since the number of active nodes can increase by one in cases 1 and 2, which each happen $O(nm)$ times.



Conclusions for dynamic trees

We can perform all pushes using dynamic tree operations. We store residual capacities explicitly for arcs not in the tree, but do not store them for arcs in the tree.

We can carry out preflow push with $O(nm)$ dynamic tree operations.

Running time is $O(nm \log n)$.

Summary of Lecture

Excess scaling.

By ensuring that all non-saturating pushes are at least $\Delta/2$ and that no node has excess greater than Δ , we bound the number of non-saturating pushes at $O(n^2)$ per scaling phase.

$O(nm + n^2 \log U)$ time.

Highest level pushing

By using a new potential function, and by carefully analyzing pushes during phases, we bound the number of non-saturating pushes at $O(n^2m^5)$.

Dynamic Trees