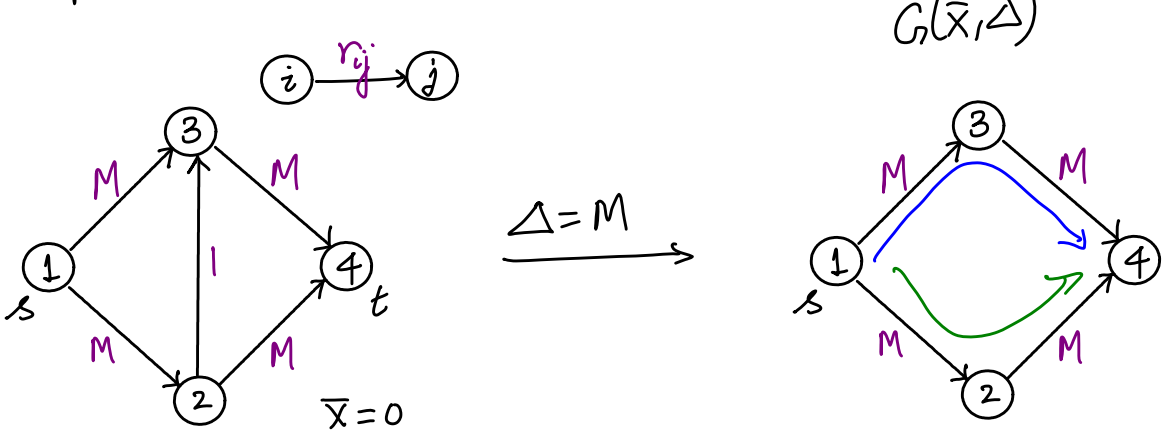
MATH 566: Lecture 21 (10/29/2024)

Today: * Complexity of capacity scaling algorithm
* Preflow-Push algorithm

Capacity Scaling on pathological example



We find max flow in the first scaling phase itself, after augmenting along two paths of residual capacity equal to $\Delta = M$.

Complexity

* Initially, $\Delta \leq U$.
* Δ is halved after each scaling phase
* Stop when $\Delta < 1$.

$\Rightarrow O(\log U)$ scaling phases.

* Each scaling phase has $O(m)$ augmentations
  - each augmentation saturates at least one arc.
  - can find augmenting path using search ($O(m)$).

  Time per augmentation $= O(n)$ → at most $n-1$ arcs in an augmenting path

$\Rightarrow$ Time per scaling phase $= O(m(m+n)) = O(m^2)$

$\Rightarrow$ Overall running time $= \underline{O(m^2 \log U)}$.

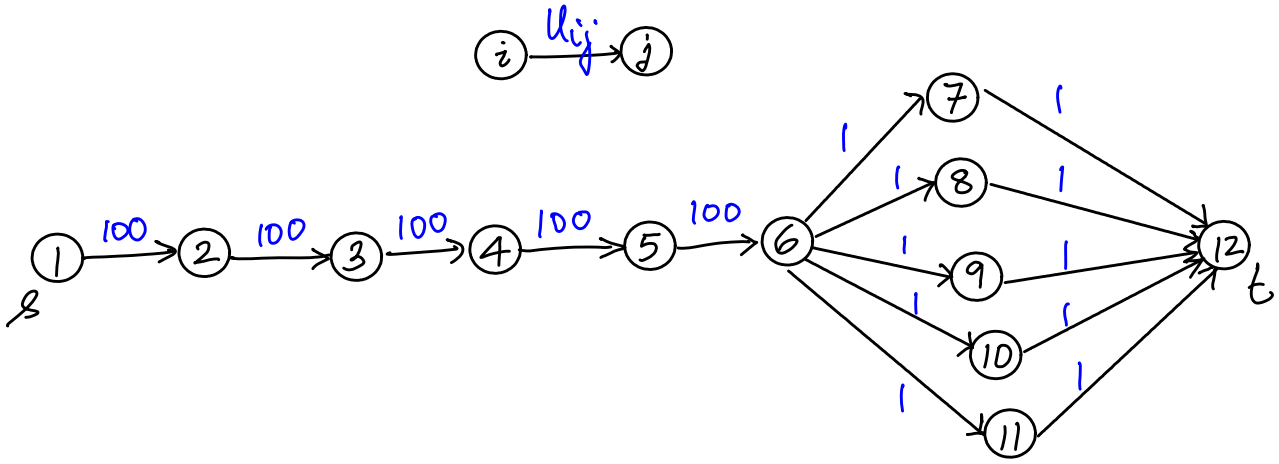
AMO Theorem 7.4

→ Could improve to $O(mn \log U)$ using d(i)'s: combine SAP ideas with capacity scaling.

Compare this worst-case complexity with that of the shortest augmenting path algorithm — $O(n^2m)$. When U is small, the $\log U$ factor could be much smaller than n, the capacity scaling algorithm could be more efficient than the SAP algorithm.

Preflow-Push Algorithms

Here is another pathological instance.



The maximum flow has value $v=5$ here. But any augmenting path algorithm would push 1 unit of flow from s to node 6 repeatedly before pushing it to t along one of the five are-disjoint paths from 6 to t. It would be better if we could push 5 units of flow from s to 6 at start, and then push this 5 units out to t along the disjoint paths in the end.

To push "excess" flow in this fashion, we allow flow balance constraints to be violated in the intermediate steps.

In a bigger network, it may not be feasible/easy to identify that the total capacity from ⑥ to ⑫ is actually 5. We may want to be greedy and push 100 units from ⑤ to ⑥. Then we'll have to push back the excess 95 units from ⑥ back to ①!

Preflow: At **intermediate stages**, we permit more inflow than outflow at nodes other than s and t. Eventually, we want flow balance to hold at all nodes.

Def A **preflow** is $\bar{x} \in \mathbb{R}^m_{\geq 0}$ (x_{ij} for all $(i,j) \in A$) such that $\bar{0} \leq \bar{x} \leq \bar{u}$ ($0 \leq x_{ij} \leq u_{ij}$) and

$$e(i) = \sum_{(j,i) \in A} x_{ji} - \sum_{(i,j) \in A} x_{ij} \geq 0 \quad \forall\, i \in N \setminus \{s,t\}.$$

($\sum_{(j,i) \in A} x_{ji}$: inflow; $\sum_{(i,j) \in A} x_{ij}$: outflow)

$e(i)$ is the **excess flow** at node i, defined as inflow − outflow.
We do not consider $e(s)$ and $e(t)$.

Def A node $i \in N \setminus \{s,t\}$ with $e(i) > 0$ is an **active node**.
s and t are never active.

IDEA: Push flow from active nodes toward t using $d(i)$'s.
Try to achieve flow balance at all nodes except s and t.
If there is excess flow left in intermediate nodes, then push that flow back to s.

Recall: distance labels $d(i)$

* valid: $d(i) \leq d(j)+1 \quad \forall (i,j) \in G(\bar{x})$
* exact: $d(i)$ = SP distance from i to t in $G(\bar{x})$ (w.r.t. # arcs)
* admissible: $(i,j) \in G(\bar{x})$ is admissible if $d(i) = d(j)+1$ (and $r_{ij} > 0$ by definition)

Generic Preflow-Push Algorithm (Goldberg-Tarjan)

```
algorithm preflow-push;
begin
    preprocess;
    while the network contains an active node do
    begin
        select an active node i;
        push/relabel(i);
    end;
end;
```

↳ node examination

Figure 7.12 Generic preflow-push algorithm.

```
procedure preprocess;
begin
    x := 0;
    compute the exact distance labels d(i);
    x_sj := u_sj for each arc (s, j) ∈ A(s);
    d(s) := n;
end;
```

↳ reverse BFS to t in $G(\bar{x})$

↳ create active nodes $\{j \mid (s,j) \in A\}$

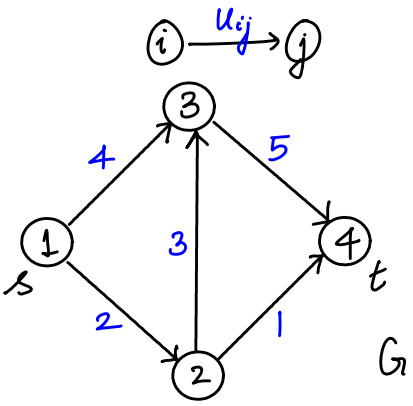
```
procedure push/relabel(i);
begin
    if the network contains an admissible arc (i, j) then
        push δ := min{e(i), r_ij} units of flow from node i to node j
    else replace d(i) by min{d(j) + 1 : (i, j) ∈ A(i) and r_ij > 0};
end;
```

↗ $d(i) = d(j)+1$, $r_{ij} > 0$

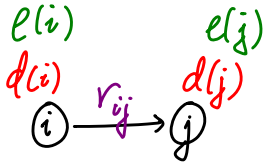
↳ relabel

Recall pipe network analogy: first raise node s to level n (above ground) and push as much flow "downstream" as possible in the pipes going out of s. We keep pushing flow "downhill" from nodes that have excess flow. If there is excess in an intermediate node but no "downhill" pipe, then raise that node. The new "downhill" pipes might take flow back toward s. Keep pushing one arc at a time till all excess in intermediate nodes is cleared.

Example

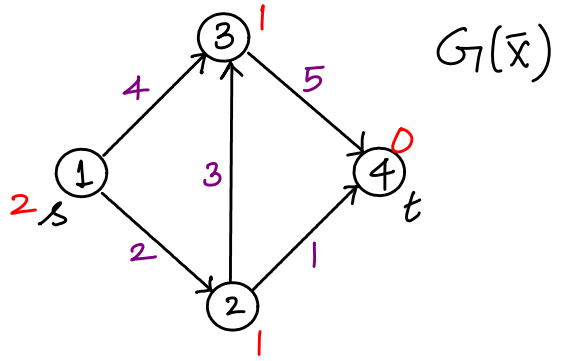


Let's consider our favorite instance.

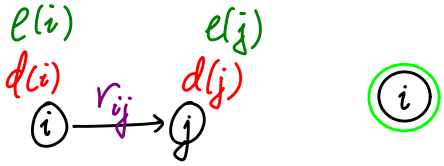


Preprocessing

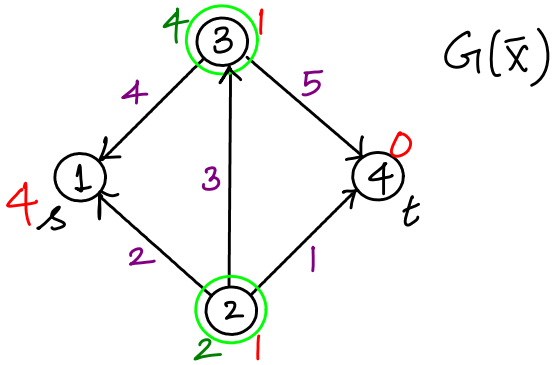
Start with $\bar{x}=\bar{0}$, find exact $d(i)$'s in $G(\bar{x})$.



Saturate all (s,j), and set $d(s)=n$.

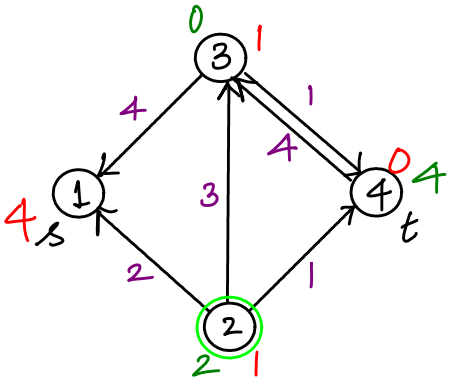


$\textcircled{i}$: i is active



Nodes 2, 3 are active now.

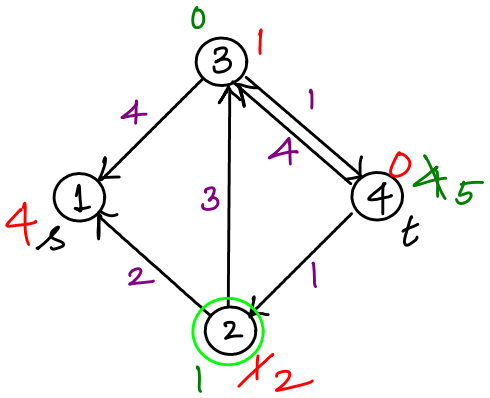
Let's choose 3. Arc (3,4) is admissible, as $d(3)=1=d(4)+1=0+1$. We push $\delta=\min\{e(3), r_{34}\}=\min\{4,5\}=4$.



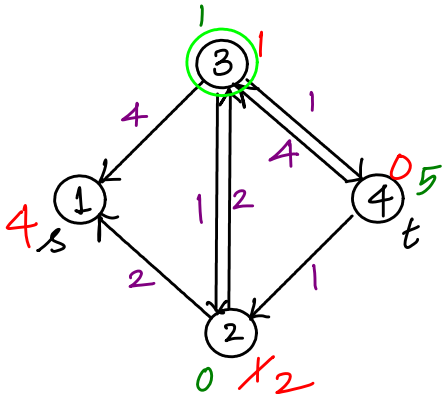
3 is not active, but 2 is still active.

pick node 2, (2,4) is admissible.

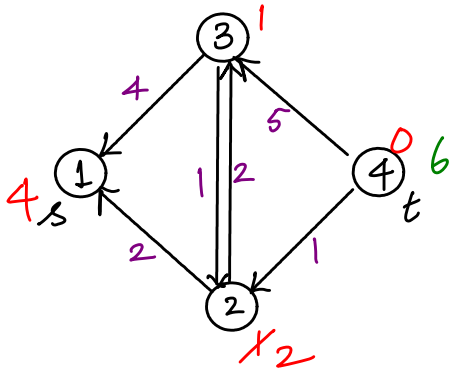
We push $\delta=\min\{e_2, r_{24}\}=\min\{2,1\}=1$.



2 is still active, but has no admissible arcs. So we relabel node 2 to $\min\{d(3), d(1)\} + 1 = 1 + 1 = 2$. After relabel, (2,3) is now admissible, so we push $\delta = \min\{e(2), r_{23}\} = \min\{1, 3\} = 1$.



2 is no longer active, but 3 is. (3,4) is admissible, so we push $\delta = \min\{e(3), r_{34}\} = \min\{1, 1\} = 1$.



There are no more active nodes. So flow is maximum, with value

$v = 5+1 = 4+2 = 6$.

coming into t — going out of s

Complexity of generic preflow push algorithm $= O(n^2 m)$ — same as that of SAP algorithm (see AMO for details).