Lecture 9

We saw a high-level view of Dinic's algorithm.

The main step here was to find a blocking flow f_b in the residual network G_f and augment f along f_b.

- Before we get into the details of how to find f_b, we first wanted to show that this approach achieves $D_{i+1}(t) > D_i(t)$,

where for any vertex v,

$D_i(v)$ = number of edges in a shortest $s-v$ path in G_f^i.

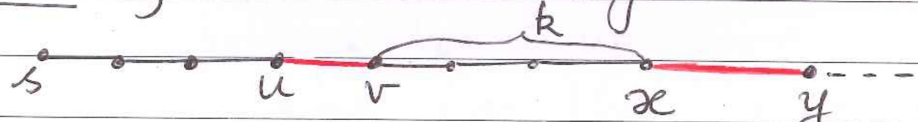
Recall that G_f^i is the residual graph in the i-th iteration.

Let p be a shortest s-t path in G_f^{i+1}.

Case 1: Every edge in p is in G_f^i.

We showed that $D_{i+1}(t) = |p| > D_i(t)$.

Case 2: p has some edges not in G_f^i.



We saw that $D_{i+1}(v) \geq D_i(v) + 2$.

We have $D_{i+1}(x) = D_{i+1}(v) + k$

$\geq D_i(v) + 2 + k$

Observe that $D_i(x) \leq D_i(v) + k$ since black edges are present in G_f^i.

So we get $D_{i+1}(x) \geq \underbrace{D_i(v) + k}_{\geq D_i(x)} + 2$

Thus $D_{i+1}(x) \geq D_i(x) + 2$.

Now use the same argument that we used to show $D_{i+1}(v) \geq D_i(v) + 2$ to show $D_{i+1}(y) \geq D_i(y) + 4$.

So if p has $l > 0$ edges that are not in G_f^i then we get $D_{i+1}(t) \geq D_i(t) + 2l$. Date

Hence in both case 1 and case 2 we have $\boxed{D_{i+1}(t) > D_i(t)}$.
This means the repeat-loop in Dinic's algorithm runs for at most n iterations.

The question that we need to answer now is:
- how do we compute a blocking flow in the layered network L_f?

current vertex

1. Initialize $v = s$; $p = \epsilon$ (empty path).
2. while true do:

{
    if $v \neq t$
        - $\boxed{\text{extend}}$ p if there is an outgoing edge (call it (v, w))
            - $p = p + (v, w)$
            - $v = w$

        else (so there is no outgoing edge)
            if $v = s$ then stop
            (f is a blocking flow)
            else
                - $p = p$ - last edge in p (retreat)
                - remove this last edge from L_f
                - v = current last vertex in p.

    else
        - an s-t path has been found (success)
        - add this path to f_b
        - remove saturated edges from L_f.
        - re-initialize $p = \epsilon$, $v = s$.
}

There will be 3 operations: retreat, extend, and success.

Date

The algorithm to find f_b starts at s - this is the current vertex and current path $p = \epsilon$.

(*) choose any outgoing edge e.
$p = p + e$
and current vertex = head(e)
if current vertex = t then success
else go to (*)
- if there is no outgoing edge
then retreat : $p = p$ - last edge.

Step (*) is the extend step.

How many "success" operations can be there?
How many "retreat" operations can be there?
How many "extend" operations can be there?

We claim the number of successes $\leq m$.
Similarly, the number of retreats $\leq m$.

- This is because L_f has at most m edges.
Every success opn. removes at least 1 edge (a saturated edge)
and every retreat opn. also removes 1 edge.
- So the number of success + retreat opns. $\leq m$.

After n extends, we have either a success or a retreat. Hence the number of extends $\leq m \cdot n$

Running time of this algo. to find f_b
= number of extends + number of retreats
+ (number of successes) $\cdot\, n$ $= O(mn)$

So the running time of Dinic's algorithm is $O(mn \cdot n) = O(mn^2)$

Both the max-flow algorithms (Ford-Fulkerson and Dinic's algorithms) that we saw are based on the same principle:

Invariant		Termination
f is a valid flow in G	$\xrightarrow[\text{flow } f \text{ such that}]{\text{keep augmenting}}$	There is no s-t path in G_f

We will see a new algorithm now that turns things the other way around, i.e., it swaps the invariant and termination conditions.

Invariant		Termination
There is no s-t path in G_f	$\xrightarrow[f \text{ such that}]{\text{keep changing}}$	f is a valid flow in G

That is, throughout the algorithm, f is <u>not</u> a valid flow in G.

f will be a function on the edge set E such that

$$\begin{cases} (1) \; 0 \leq f(e) \leq c(e) \quad \forall e \in E \\ (2) \displaystyle\sum_{e:\, e \text{ entering } u} f(e) \geq \sum_{e:\, e \text{ leaving } u} f(e) \quad \forall u \in V - \{s\}. \\ \text{That is, } excess(u) \geq 0 \quad \forall u \in V - \{s\}. \end{cases}$$

Such a function f is called a <u>preflow</u>.

So as before, s generates flow. In the previous max-flow algorithms, every intermediate vertex had to maintain the flow conservation constraint. Now imagine a large tank next to each vertex. Each vertex can use its tank to temporarily store the excess that it has. Finally the tanks have to be empty and only t is allowed to have positive excess.

- Now G_f is the residual network wrt preflow f

Our main operation here is push. Suppose vertex v has positive excess — then v has to get rid of this.

- let $e = (v, w)$ be an edge in G_f.
- we can do push(e, δ): this operation sends δ units of flow along e. where $\delta \leq$ residual-capacity(e) and $\delta \leq$ excess(v).

However we want to coordinate flow "towards t."

- so a push operation should be such that it helps increase excess(t).

push seems to be a local operation. So how do we achieve a global goal such as routing flow towards t?

- let each vertex have a level number
  $l(v)$ = level number of v

Always push flow from a higher level vertex to a lower level vertex.

- we will introduce the notion of eligible edge. Edge (u, v) is eligible if $(u, v) \in G_f$ and $l(u) > l(v)$.

We are now ready to write down the basic preflow push algorithm. Throughout the algorithm, f is a preflow.

1. Initialize $l(s) = n$ and
$l(u) = 0 \ \forall \ u \in V - \{s\}$.

[so the level of s is n and the level of all other vertices is 0. Other vertices can increase their level but t will always be at level 0 and s will always be at level n.]

2. Initialize $f(e) = c(e) \ \forall \ e \in E$ that are outgoing from s.
$f(e) = 0$ for all other edges.

[so the preflow f sends as much flow as possible along edges leaving s. Now it is the responsibility of all out-neighbours of s to get rid of all the flow they received.]

3. Construct the residual graph G_f wrt f.

4. while there is a vertex $u \neq t$ with positive excess do:
{
- if there is an eligible edge (u,v) out of u then
  - push$((u,v), \delta)$ where $\delta = \min(res(u,v), excess(u))$
  - update f, G_f, excess(u) and excess(v)

  else
  - relabel(u)

}

5. Return f.

What do we do when excess(v) > 0 but there is no eligible edge out of u? * relabel(u): increase the level number of u by 1.