Max Flow Lecture II

15-451/651

10/13/20

Recall the max-flow Problem:

Def A **flow network** is

1) $G=(V,E)$ directed (oriented)
2) Edge capacities $c: V \times V \to R$
   st $c(u,v) \geq 0$
   a) $(u,v) \in E$ then $c(u,v), c(v,u) \geq 0$ (formally)
3) $s \neq t \in V$ $s \equiv$ source & $t \equiv$ sink

Def $f: V \times V \to \mathbb{R}$ is a **flow** for network G if

1) Capacity constraints: $f(u,v) \leq c(u,v)$
2) Skewed symmetric: $f(u,v) = -f(v,u)$
3) Flowin = Flowout for $u \in V - \{s,t\}$

$$\sum_{v \in V} f(u,v) = 0$$

Def **Netflow** $\equiv |f| = \sum_{v \in V} f(s,v)$

The Maximum Flow Prob

Input: Flow-Network $G=(V,E), s, t, c$
Output: Flow f with maximum net-flow.

Residual Network

Consider Network $G=(V,E,c)$ & flow f

Def Residual Capacity: $c_f(u,v) = c(u,v) - f(u,v)$

Residual Network:

Edges $E_f = \{(u,v) \in V^2 \mid c_f(u,v) > 0\}$

$G_f = (V, E_f, c_f)$

Def P is an aug path if P is a path in G_f using edges with positive capacities.

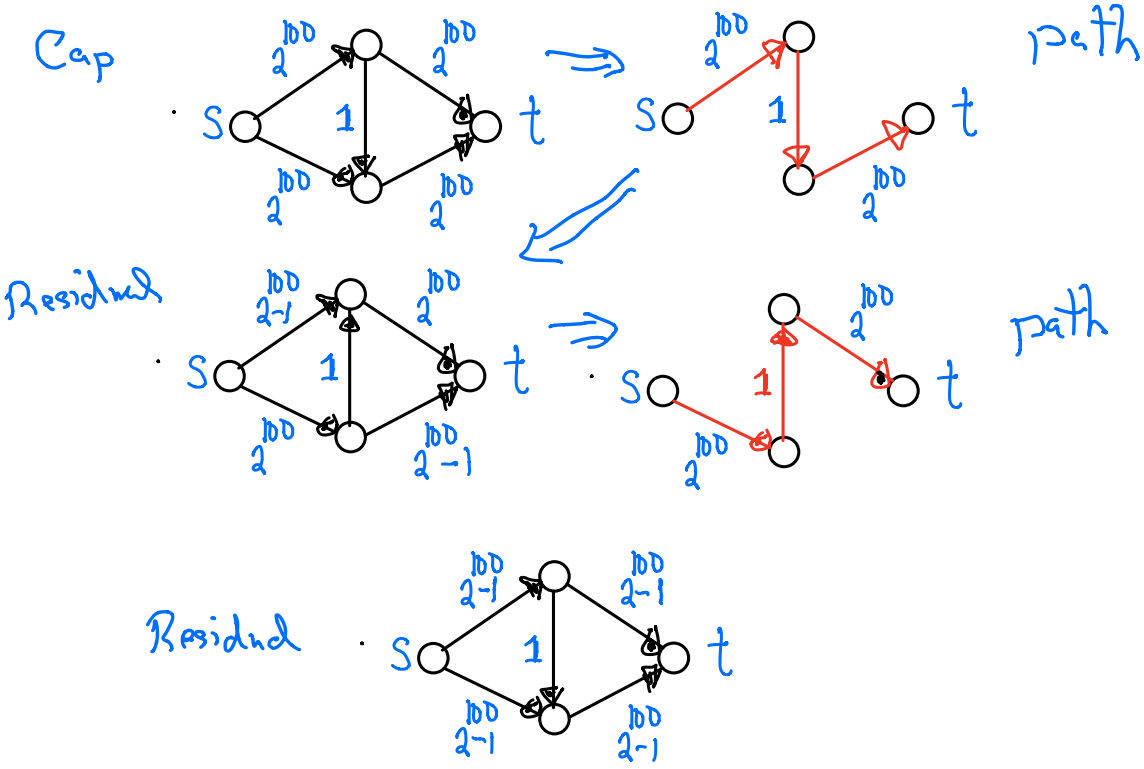
Ford-Fulkerson Method

F-F-Method (G, s, t, c)

1) Initialize flow f to zero.
2) While ∃ augmenting Path p in G_f
   Set f_p be max flow on p.
   Set $f = f_p + f$
3) Return f.

Issue: Of the many augmenting path which one should we pick.

There are bad choices!



Faster Max-Flow

1) the example uses 2^{100} iteration!
2) For nonintegral capacities may never converge!

There are many-many ideas on better algorithms

Types of solutions:

1) Find and update augmenting paths.
   But be more careful on picking paths
2) Drop loop invariant that
   flow in = flow out
3) Find faster approximate flows.
4) Solve as a linear program

Today we consider only

1) augmenting path algorithms.

1) Edmonds-Karp #1 (Max Cap Paths)
2) Edmonds-Karp #2 (Shortest Paths)
3) Dinic's Alg (Blocking Flow)

Edmonds-Karp #1

Alg: Pick Aug-Path with maximum capacity.
(Max bottle-neck path)

Thm For EK#1 the number of iterations is at most $m \ln(F/c)$ where F is max-flow & c is min capacity

We use the following claim

claim: $\exists$ a path of capacity $\geq F/m$.

Pf: By contradiction

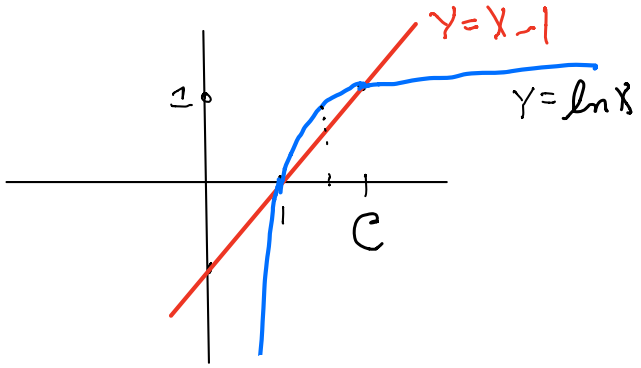
Remove edges with cap $< F/m$
(there will be at most m.)
These edges form a cut of size
$F/m \cdot m < F$ Contra!. □

After first path new max-flow $\leq F - F/m$
or $F(1-1/m)$

After i iterations
$F(1-1/m)^i \geq c$.

Fact: $(1-1/m)^m < 1/e$

The pf follow from following diagram!



Proof of thm

Goal find an i s.t.

$$F(1-1/m)^i \le C$$

iff $$((1-1/m)^m)^{i/m} \le C/F$$

$\Leftarrow$ $$(1/e)^{i/m} \le C/F \quad \text{(suffice)}$$

iff $$F/C \le C^{i/m}$$

iff $$\ln(F/C) \le i/m$$

iff $$m\ln(F/C) \le i \text{ works.}$$ □

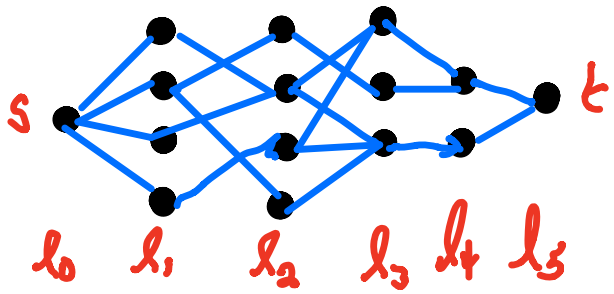
Edmonds-Karp #2

Alg: Pick Aug-Path with smallest length
(#edges, #hops)

Def: The level graph L_G of G on G_f

$level_i = \{v \in V \mid i \text{ hops from } s\}$

eg



Observations about L_G.

1) No edges from l_i to l_j $j > i+1$ (Why?)
2) Edge from l_i to l_j $j \leq i$ OK.
3) More levels after $t \in l_k$.
4) Max # of levels $\leq n$
5) If f is aug-path then level # can only increase G_f.
6) At most m aug-path that saturate an edge.

Thm EK2 is $O(nm^2)$

pf: 1) cost per path $O(m)$
2) paths per level m
3) # levels n

} cost per level

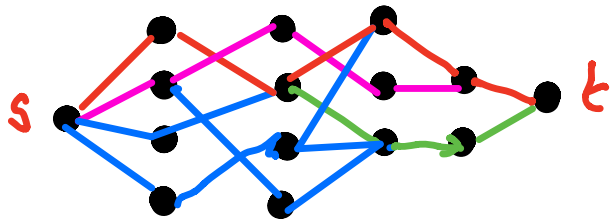
Total: $O(n \cdot m^2)$

Goal: Find $O(n \cdot m)$ cost per level Alg!
Find $O(n^2)$ cost per level Alg!

Def: For each vertex v fix an ordering of edges out of v to next level.

Thus for each path from s to t we get a sequence of integers.
This gives a lexiographic ordering of the paths.

Dinic: Add aug-path in lexiographic order.
Same as DFS order.

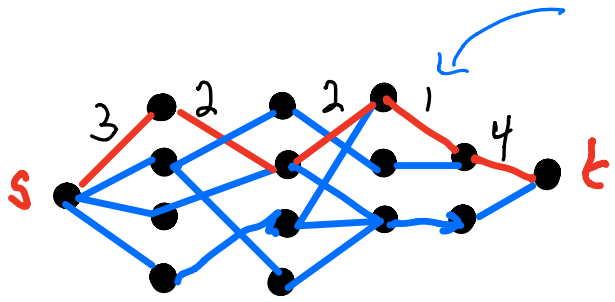


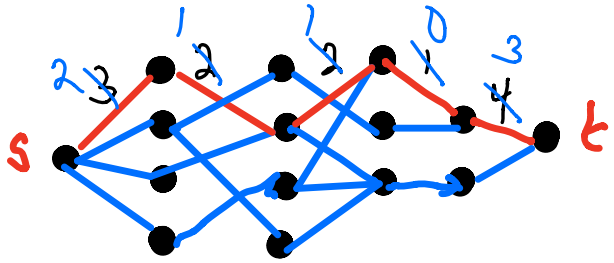
first path red
second green
third violet

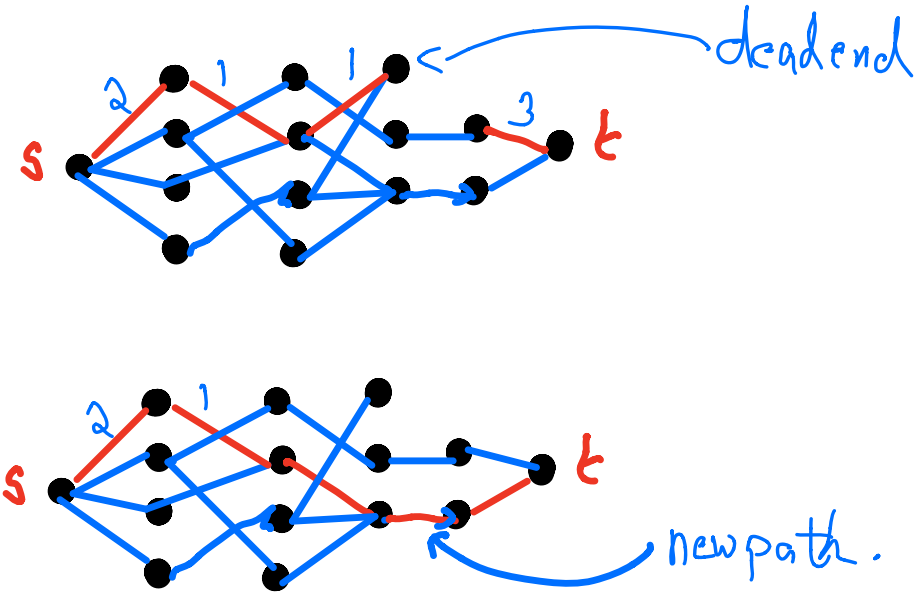
Dinic's Blocking Flow Alg:

1) Find Lex first path from s to t.
(Removing deadend edges)

2) Update residual graph
(Removing saturated edges.)







Efficient Implementation

Edges are not deleted. We use pointers to keep track of "deleted" edges.

For each vertex maintains the following Attributes:

bottle(v) ≡ Bottleneck on active path from s to v.
edge(v) ≡ Bottleneck edge. (first)
next(v) ≡ Next live edge out of v.

Timing for Dinic's Blocking Flow

Observe:

1) An edge is traversed twice for each path found
   (Once in each direction)

2) An edge is traversed twice for dead ends
   (Once in each direction)
   (Edge is deleted)

Charging Rule:

1) Charge type 1) to paths.
2) Charge type 2) to edge

Total cost:

1) m paths of at most n edges
2) m edges

$O(nm+m) = O(nm)$ □

MPM Alg for Blocking-Flow

Def: $\text{Capacity}(v \in V) = \min(\text{InCap}(v), \text{OutCap}(v))$

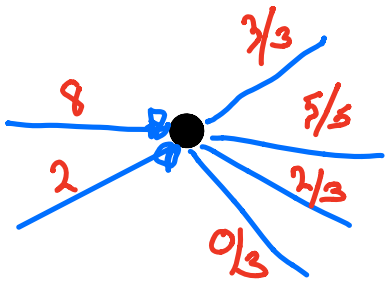
Let $\bar{v}$ have min Cap

Claim: $\exists$ flow of size $\text{Cap}(\bar{v})$ and greedy alg works.

MPM Alg

1) Find min cap vertex $\bar{v}$ and cap c.
2) Lexiographically push c flow from $\bar{v}$ to t
3) Lexiographically pull c flow from $\bar{v}$ to s.
4) Repeat while min cap > 0.

Note Always saturate an edge before using new edge.



Thm MPM Alg is $O(n^3)$ time to find Blocking-Flow.

Charging Rule:

1) If edge become saturated charge to edge.
2) If only partial flow added edge then charge node.

Total cost of one MPM push-pull flow is at most

#saturated edges + n

Total cost of MPM blocking-flow is at most:

1) # all saturated edges $\leq m$
2) # of push-pulls $\times\ n \leq n^2$

Total cost $O(n^2+m) = O(n^2)$

MPM Alg is $O(n^3)$