Lec 16

Lots of Algorithms Independently Discovered!

- Ford-Fulkerson
  - Integer capacities: maximum flow f in $O(m \cdot v(f))$
  - Real capacities: ∞. *It may never halt*!
  - Useful for ***thinking about*** cuts & flows; not a great algorithm.
- Dinic/Dinitz (USSR→Israel)
  - Maximum flow in $O(mn^2)$ time. $n = |V|, m = |E|$.
- Edmonds-Karp (Canada-USA)
  - Maximum flow in $O(m^2 n)$ time.

Recall that:

Formal Definition of *Flow*

- $f : V \times V \to \mathbb{Z}^+$ is a legal flow if it satisfies:
  - CAPACITY CONSTRAINT:
    $f(u,v) \leq c(u,v)$ $\quad (c(u,v) = 0 \text{ if } (u,v) \notin E)$
  - SKEW SYMMETRY:
    $f(u,v) = -f(v,u)$
  - FLOW CONSERVATION:
    for all $u \in V - \{s,t\}$
    $\sum_{(u,x)\in E} f(u,x) = 0.$ (flow out of u = 0)

$v(f)$ = total flow leaving s = total flow entering t

Residual Networks

- A flow f in G defines a *residual network* G_f
  - $c_f(u,v)$ represents the amount of additional flow that could be sent through (u,v) without violating the capacity constraint
  - $c_f(u,v) = c(u,v) - f(u,v)$
  - Only edges w/ ***positive*** capacity appear in G_f
- If f is a legal flow in G and f' is a legal flow in G_f then $f + f'$ is a legal flow in G.

Max-Flow — Min-Cut Theorem

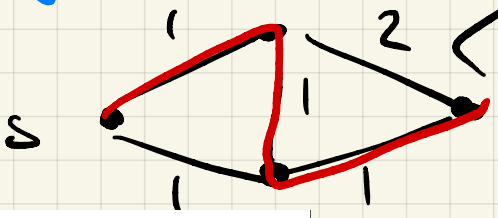
- The following are equivalent:
  (1) f is a maximum flow in G
  (2) There is no path from s to t in G_f
  (3) $v(f) = c(A,B)$ for some s-t cut (A,B)

Blocking Flows

Def) An edge $e \in E(G)$ is saturated by f if $f(e) = c(e)$

Def) A flow f in G is called a "blocking flow" if every path from s to t has a saturated edge

Note: blocking $\not\Rightarrow$ maximal



this is blocking with flow 1
but maximal is 2.

Dinitz's Max-Flow Algorithm

- $f := 0$
- While(f is not a maximum flow) {
  - Let G_f be the residual network for G w.r.t. f.
  - Let $G' = (V, E')$, where $E' \subseteq E(G_f)$ are those edges in ***shortest paths from s*** in G_f.
  - $f' :=$ any blocking flow in G'
  - $f := f + f'$

  }

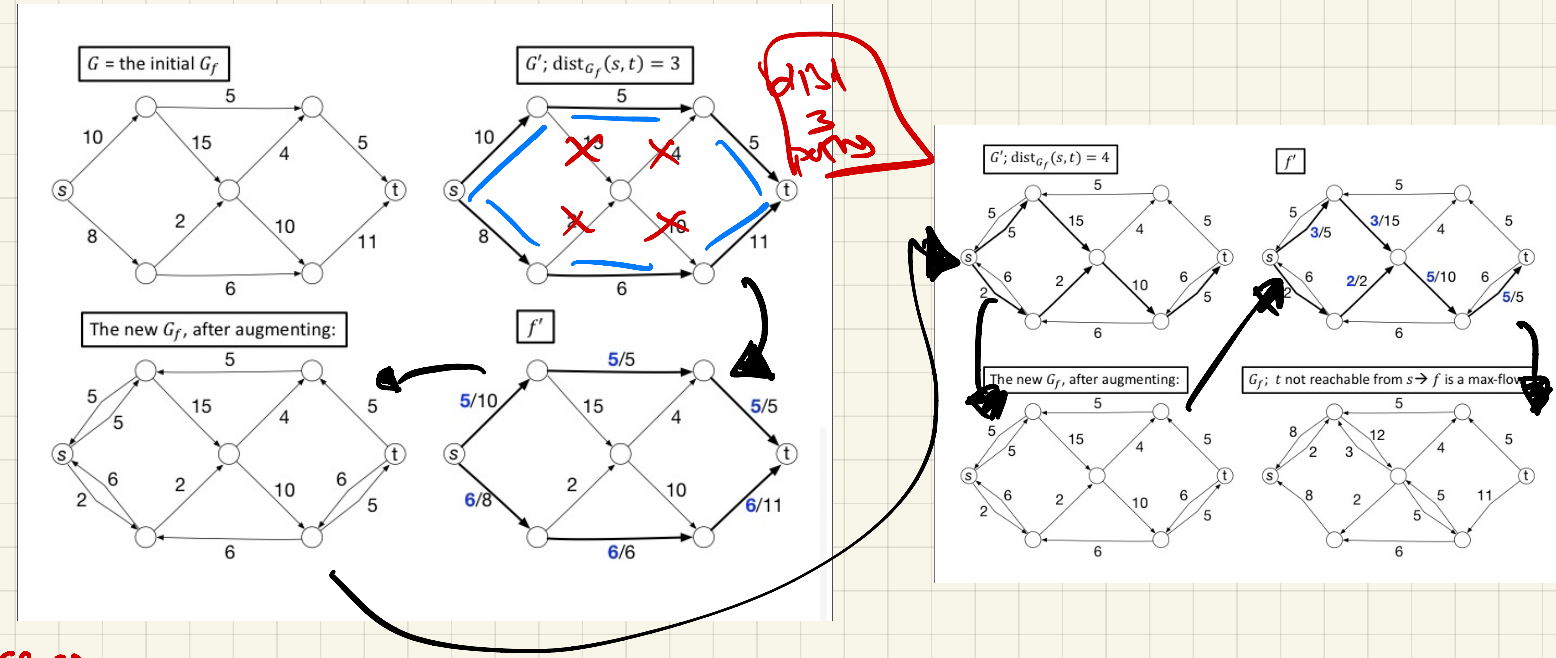
  Return f.

→ by max flow min cut ⇔ path from s→t in G_f

↳ flow in G'

↳ flow in G_f

⇒ $f + f'$ flow in G



Issues

1) How to find a blocking flow
2) How many iterations

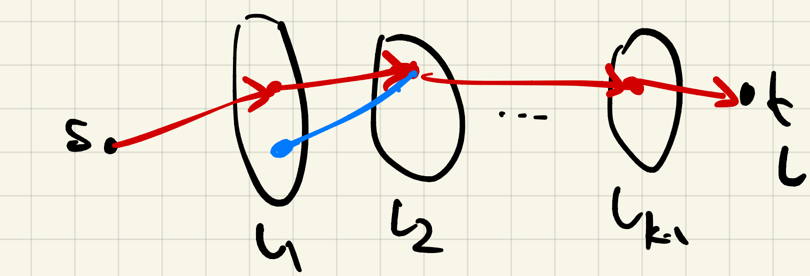
The key lemma

- **Lemma**: In every iteration of the while loop of Dinitz's algorithm, $\text{dist}_{G_f}(s,t)$ increases by at least 1.
- **Corollary**: The algorithm computes at most $n-1$ blocking flows.

unit weight edge graph

longest dist in here is n-1 edges redundant

Think of G' as below



$L_i = \{v \mid \text{dist}(s,v) = i\}$

all good paths (no edge with capacity >2 skips forward 2 layers)

f' is a blocking flow in G'

$f'' = f + f'$ dist(s,t) in $G_{f''}(s,t) > k$

Let $P = (s = u_0, u_1, \dots, t)$ be a path in $G_{f''}$ → no flow in either direct.

cases ① $\forall i$ $u_i \in L_j \Rightarrow u_{i+1} \in L_0 \cup \dots \cup L_{j+1}$ → if $u_{i+1} \in L_{j+2}$ or farther

② $\exists i$ so $u_i \in L_j$ & $u_{i+1} \notin L_{j+1}$

$c_{f''}(u_i, u_{i+1}) = 0$ ⇐ but can't! ← only way it shows up is if we flow ← couldn't have existed in G'

note: $0 = c_f(u_i, u_{i+1})$

② By contra: if we advance exactly one layer

→ ($s_0 = u_0$, u_1, ..., $u_k = t_0$) in $G_{f''}$ → same layer as earlier
$\quad L_0 \quad L_2 \quad\quad L_k$

→ wasn't saturated in G_f!

So f' wasn't blocking → oops

①② $\Rightarrow \text{dist}_{G_{f''}}(s,t) > \text{dist}_{G_f}(s,t)$!

Computing Blocking Flows

- Problem on HW5:
  (1) A blocking flow can be computed in $O(mn)$ time.
  (2) If, $\forall e \in E. c(e) = 1$, a blocking flow can be computed in $O(m)$ time.

- **Theorem**. Dinitz's max flow algorithm takes $O(mn^2)$ time.
- **Pf.** There are $< n$ blocking flows, each takes $O(mn)$ time.

Idea: trivial $O(m^2)$

DFS $O(m)$
→ saturate edge on path
→ since it saturates an edge
→ at most m iter

Dynamic Trees, link cut trees

Edmonds-Karp MaxFlow

- $f := 0$
- While(f is not a maximum flow) {
  - Let G_f be the residual network for G w.r.t. f.
  - Let P be a ***shortest path*** from s to t in G_f.
  - $f' :=$ maximum flow along P.
  - $f := f + f'$

  }

  Return f.

Unit-Capacity Networks

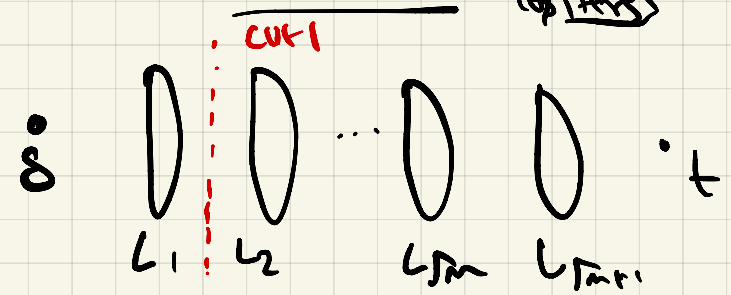
- **Theorem.** [Karzanov, Even-Tarjan] If $G = (V, E, c)$ is a flow network with
  - $c(s,u)$ and $c(u,t)$ arbitrary.
  - $c(u,v) = 1$ for all other edges $(u,v) \in E$.
- then Dinitz's alg. takes $O(\min\{m^{3/2}, mn^{2/3}\})$ time.

Holds even if G is a multigraph | Only holds if G is simple (multiple copies of same edge disallowed)

Maximum Matching

- **Theorem.** [Karzanov, Hopcroft-Karp] A maximum matching in a *bipartite* graph $G = (A \cup B, E)$ can be computed in $O(m\sqrt{n})$ time.
- The algorithm:
  - Convert G to a flow network. (Direct edges from A to B, add s, t, put capacity $c(e) = 1$ on all edges.)
  - Run Dinitz's algorithm. Running time is $O(m \cdot (\#\text{BF}))$ where (#BF) is the number of blocking flows.

→ well in unit cap graph finding blocking flow is linear.
→ $O(m)$
→ wts $\#BF$ $O(\sqrt{m})$ → iters



i^{th} cut is layers $L_0 \ldots L_i$ & $L_{i+1}, \ldots, L_{\sqrt{m}+1}, t$

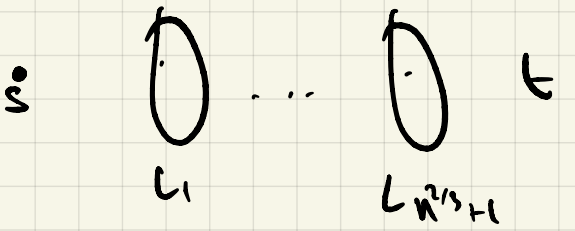
$v(f^*) - v(f) \le c(\text{cut}_1), \ldots, \text{cut}(\sqrt{m})$

$\Rightarrow$ → $\le \min_{1 \le i \le \sqrt{m}} \text{cut}(i) \le \sqrt{m}$
by pigeonhole

Every blocking flow get at least 1.
At most another $\sqrt{m}$ iter
→ $2\sqrt{m}$ iter $\Rightarrow O(m^{1/2})$ iter

Claim # B-f $\in O(n^{2/3})$ suppose simple graph

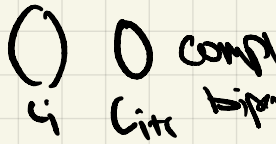
Similar argument $n^{2/3}+1$ BF.



$cut_i = (\dots L_i)(\dots L_{n^{2/3}})$

$c(cut_i) \le |L_i| \cdot |L_{i+1}|$

worst $\le \left(\frac{|L_i|+|L_{i+1}|}{2}\right)^2$ geom mean



$\therefore v(f) - v(f^*) = \min_i c(cut_i) \le (n^{1/3})^2$

$\sum_i \left(\frac{|L_i|+|L_{i+1}|}{2}\right) \le n$

min across all

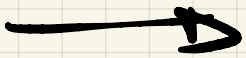
$\frac{n}{n^{2/3}} = n^{1/3}$

$\Rightarrow$ need $n^{2/3} + n^{2/3}$ iter $\Rightarrow O(n^{2/3})$

Maximum Matching

- **Theorem.** [Karzanov, Hopcroft-Karp] A maximum matching in a *bipartite* graph $G = (A \cup B, E)$ can be computed in $O(m\sqrt{n})$ time.
- The algorithm:
  - Convert G to a flow network. (Direct edges from A to B, add s, t, put capacity $c(e) = 1$ on all edges.)
  - Run Dinitz's algorithm. Running time is $O(m \cdot (\#\text{BF}))$ where $(\#\text{BF})$ is the number of blocking flows.

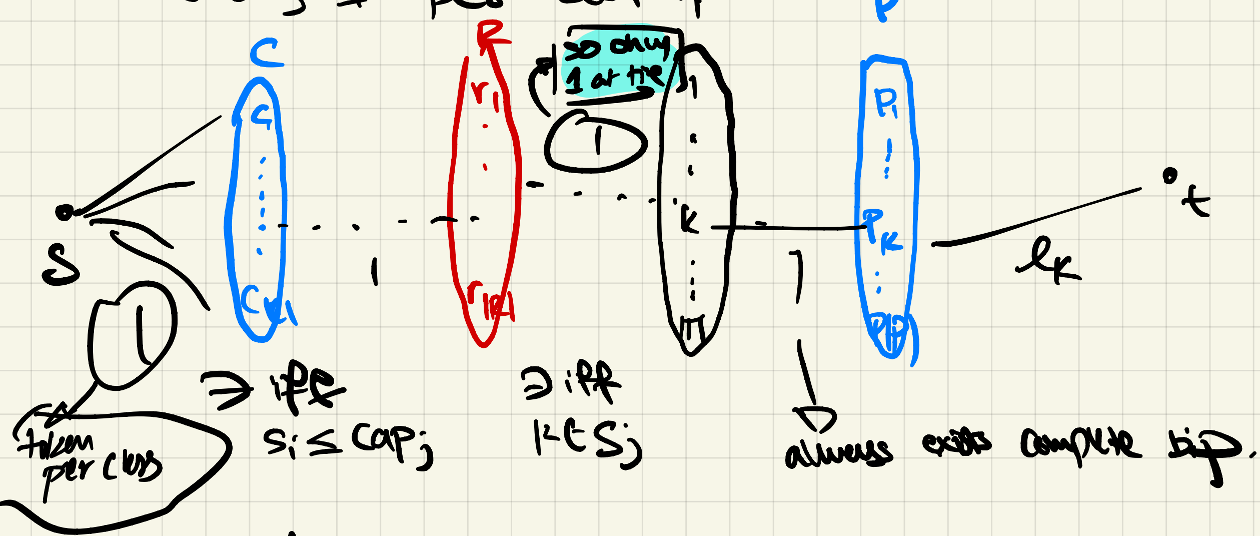
$< \sqrt{n}$ bp



Application

Scheduling Finals

- C classes. Class c_i has s_i students
- R rooms. room r_j has capacity cap_j
- T time slots. Room r_j available in slots $S_j \subseteq \{1, \dots, T\}$
- P proctors that can proctor. p_k does at most ℓ_k exams. Also only 1 per slot



Note $f^* \in \mathbb{Z}$ as int edges.

Another example: precedence constr..