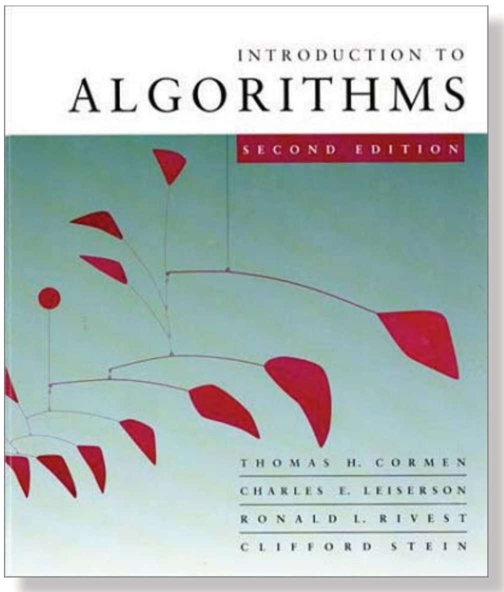
Design and Analysis of Algorithms

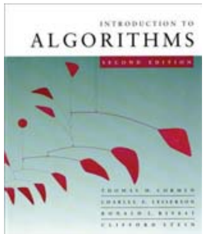
6.046J/18.401J



LECTURE 14

Network Flow & Applications

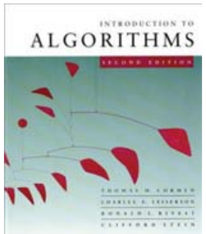
- Review
- Max-flow min-cut theorem
- Edmonds Karp algorithm
- Flow Integrality
- Part II: Applications



Recall from Lecture 13

- ***Flow value:*** $|f| = f(s, V)$.
- ***Cut:*** Any partition (S, T) of V such that $s \in S$ and $t \in T$.
- **Lemma.** $|f| = f(S, T)$ for any cut (S, T).
- **Corollary.** $|f| \leq c(S, T)$ for any cut (S, T).
- ***Residual graph:*** The graph $G_f = (V, E_f)$ with strictly positive ***residual capacities*** $c_f(u, v) = c(u, v) - f(u, v) > 0$.
- ***Augmenting path:*** Any path from s to t in G_f.
- ***Residual capacity*** of an augmenting path:

$$c_f(p) = \min_{(u,v)\in p} \{c_f(u,v)\}.$$



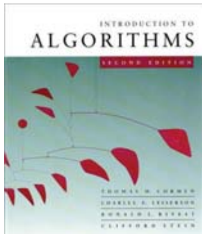
Ford-Fulkerson max-flow algorithm

Algorithm:

$f[u, v] \leftarrow 0$ for all $u, v \in V$

while an augmenting path p in G wrt f exists

do augment f by $c_f(p)$



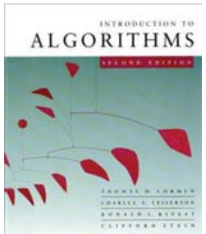
Max-flow, min-cut theorem

Theorem. The following are equivalent:

1. $|f| = c(S, T)$ for some cut (S, T).
2. f is a maximum flow.
3. f admits no augmenting paths.

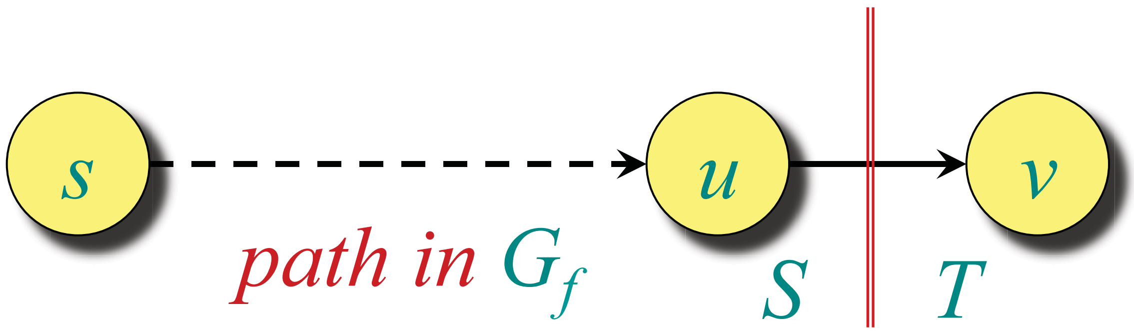
Proof.
(*1*) ⇒ (*2*): Since $|f| \leq c(S, T)$ for any cut (S, T), the assumption that $|f| = c(S, T)$ implies that f is a maximum flow.

(*2*) ⇒ (*3*): If there were an augmenting path, the flow value could be increased, contradicting the maximality of f.

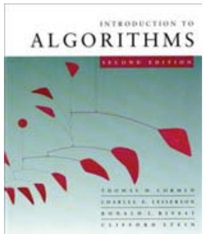


Proof (continued)

(3) ⇒ (1): Suppose that f admits no augmenting paths. Define $S = \{v \in V : \text{there exists a path in } G_f \text{ from } s \text{ to } v\}$, and let $T = V - S$. Observe that $s \in S$ and $t \in T$, and thus (S, T) is a cut. Consider any vertices $u \in S$ and $v \in T$.



We must have $c_f(u, v) = 0$, since if $c_f(u, v) > 0$, then $v \in S$, not $v \in T$ as assumed. Thus, $f(u, v) = c(u, v)$, since $c_f(u, v) = c(u, v) - f(u, v)$. Summing over all $u \in S$ and $v \in T$ yields $f(S, T) = c(S, T)$, and since $|f| = f(S, T)$, the theorem follows. □



Ford-Fulkerson max-flow algorithm

Algorithm:

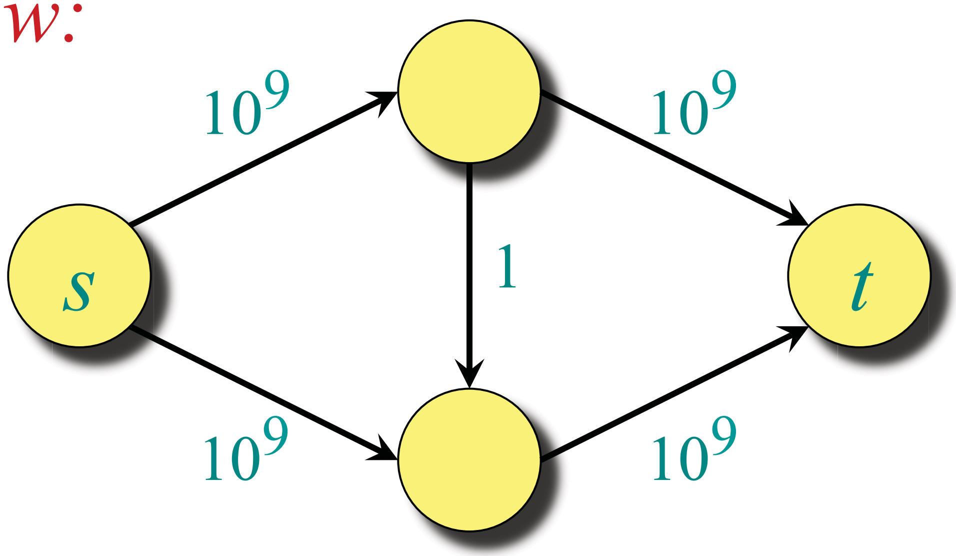
$f[u, v] \leftarrow 0$ for all $u, v \in V$

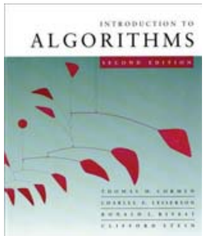
while an augmenting path p in G wrt f exists

do augment f by $c_f(p)$

Can be slow:

G:





Ford-Fulkerson max-flow algorithm

Algorithm:

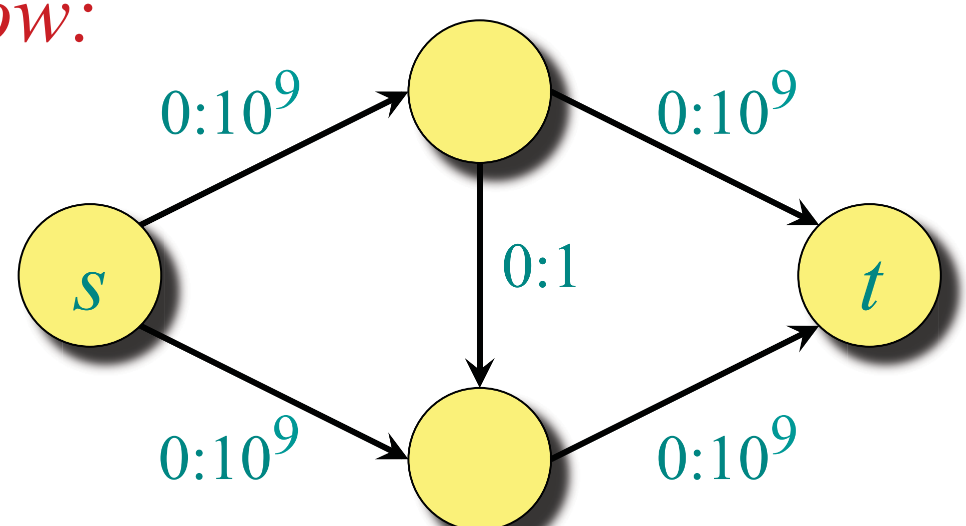
$f[u, v] \leftarrow 0$ for all $u, v \in V$

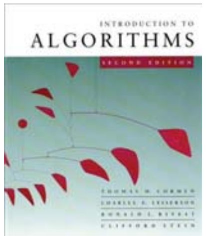
while an augmenting path p in G wrt f exists

do augment f by $c_f(p)$

Can be slow:

G:





Ford-Fulkerson max-flow algorithm

Algorithm:

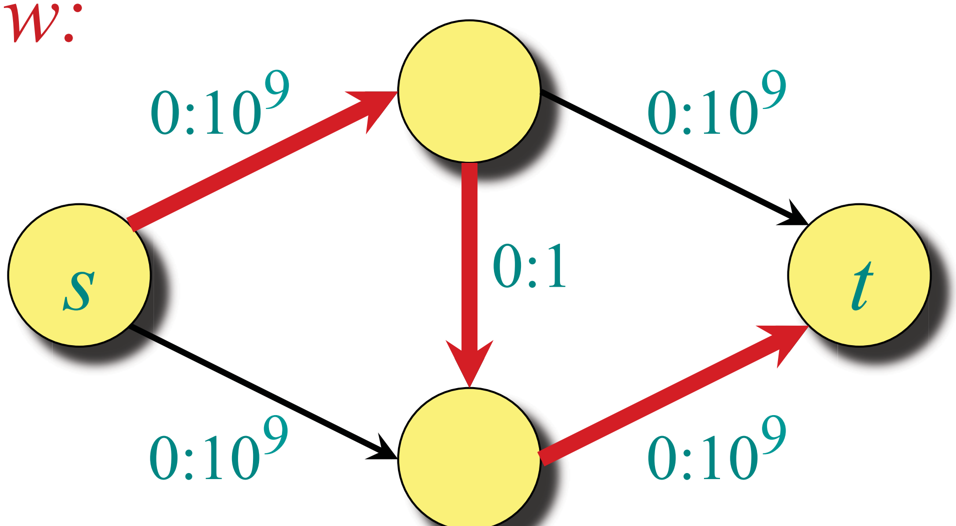
$f[u, v] \leftarrow 0$ for all $u, v \in V$

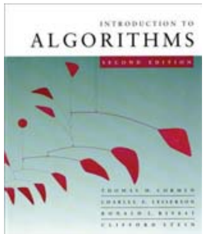
while an augmenting path p in G wrt f exists

do augment f by $c_f(p)$

Can be slow:

G:





Ford-Fulkerson max-flow algorithm

Algorithm:

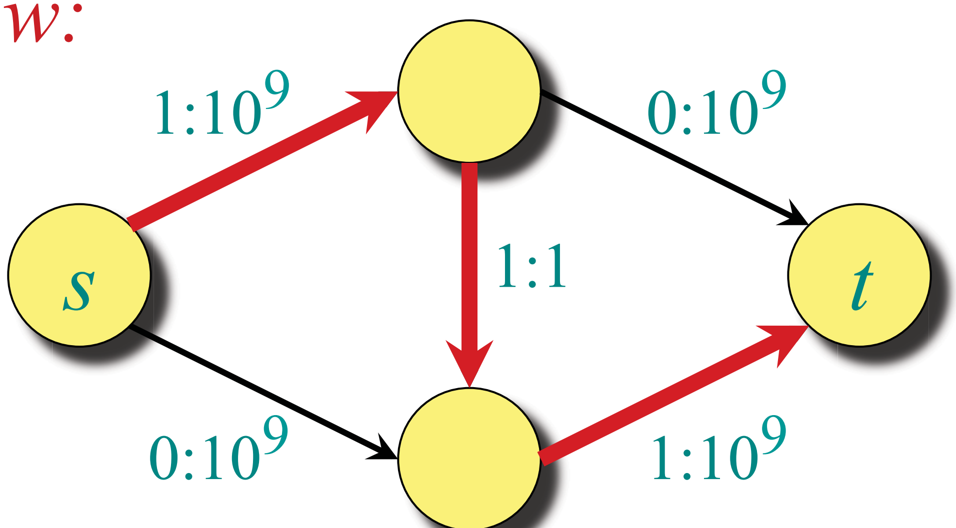
$f[u, v] \leftarrow 0$ for all $u, v \in V$

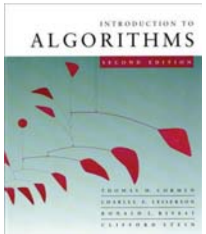
while an augmenting path p in G wrt f exists

do augment f by $c_f(p)$

Can be slow:

G:





Ford-Fulkerson max-flow algorithm

Algorithm:

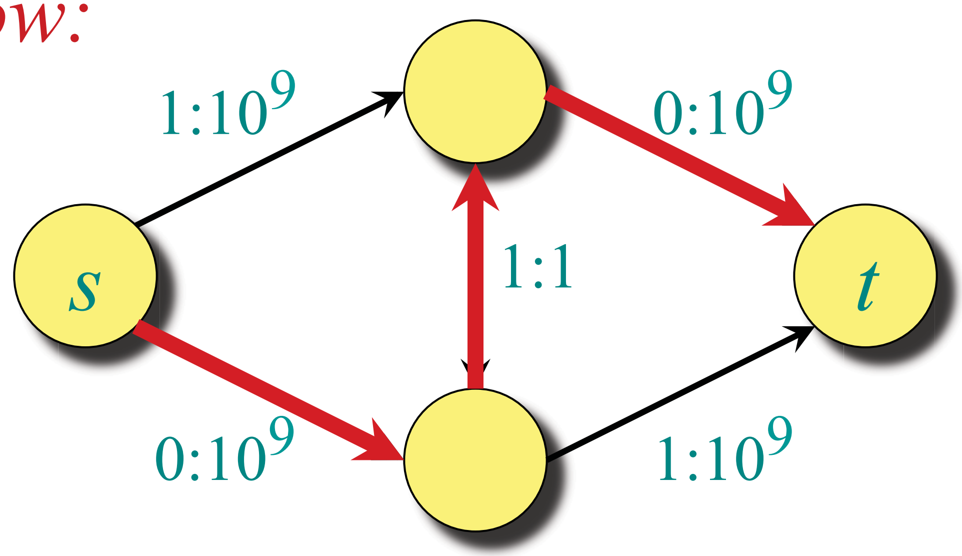
$f[u, v] \leftarrow 0$ for all $u, v \in V$

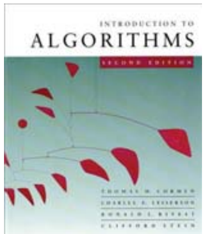
while an augmenting path p in G wrt f exists

do augment f by $c_f(p)$

Can be slow:

G:





Ford-Fulkerson max-flow algorithm

Algorithm:

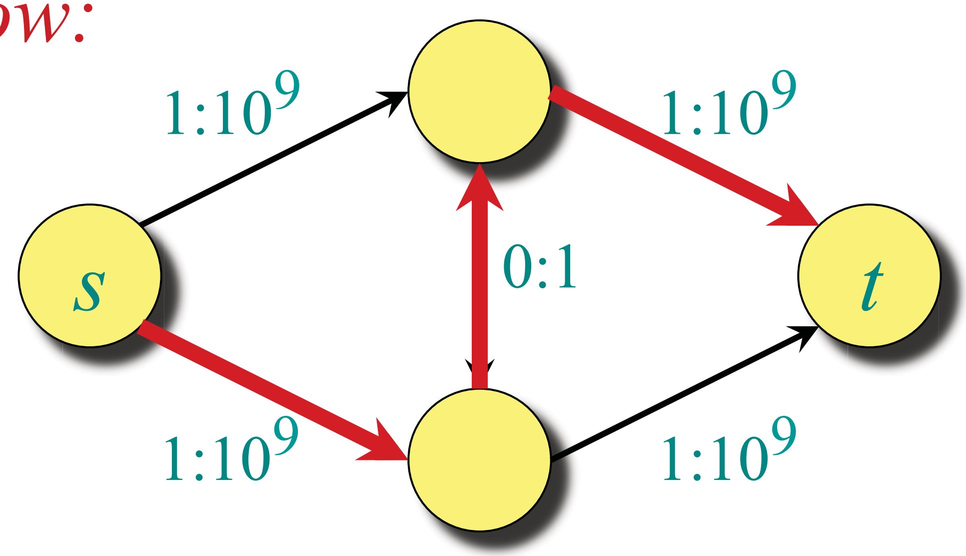
$f[u, v] \leftarrow 0$ for all $u, v \in V$

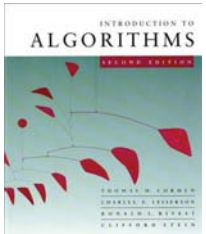
while an augmenting path p in G wrt f exists

do augment f by $c_f(p)$

Can be slow:

G:





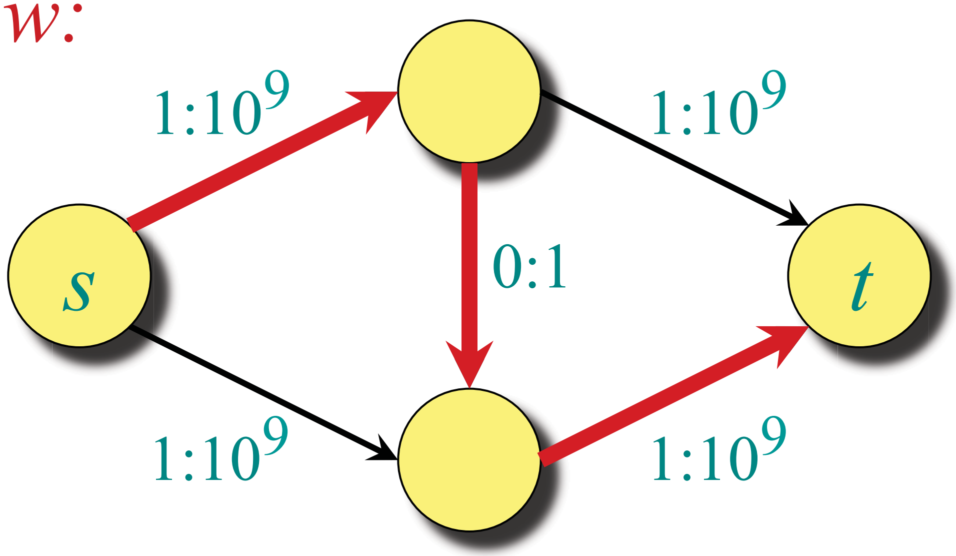
Ford-Fulkerson max-flow algorithm

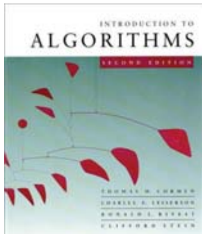
Algorithm:

$f[u, v] \leftarrow 0$ for all $u, v \in V$
while an augmenting path p in G wrt f exists
do augment f by $c_f(p)$

Can be slow:

G:





Ford-Fulkerson max-flow algorithm

Algorithm:

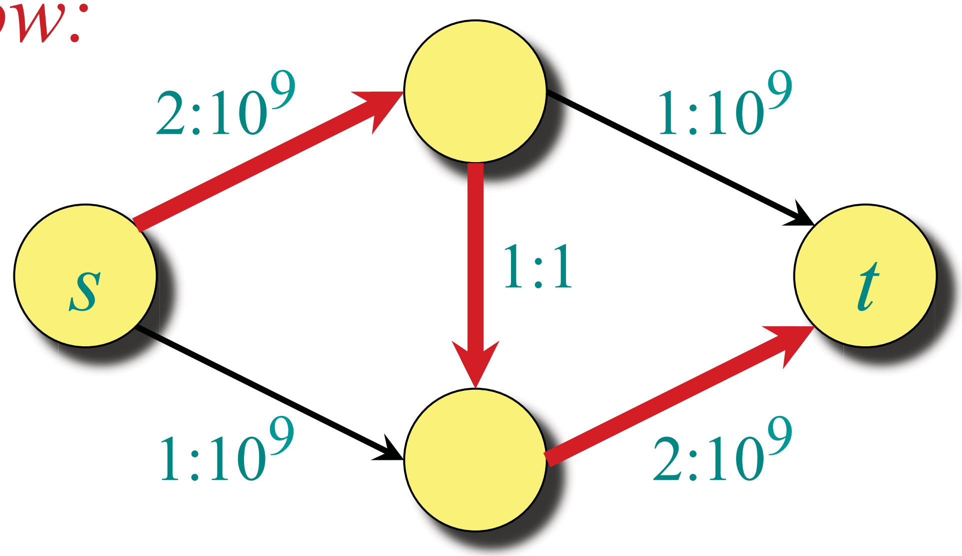
$f[u, v] \leftarrow 0$ for all $u, v \in V$

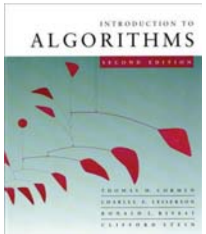
while an augmenting path p in G wrt f exists

do augment f by $c_f(p)$

Can be slow:

G:





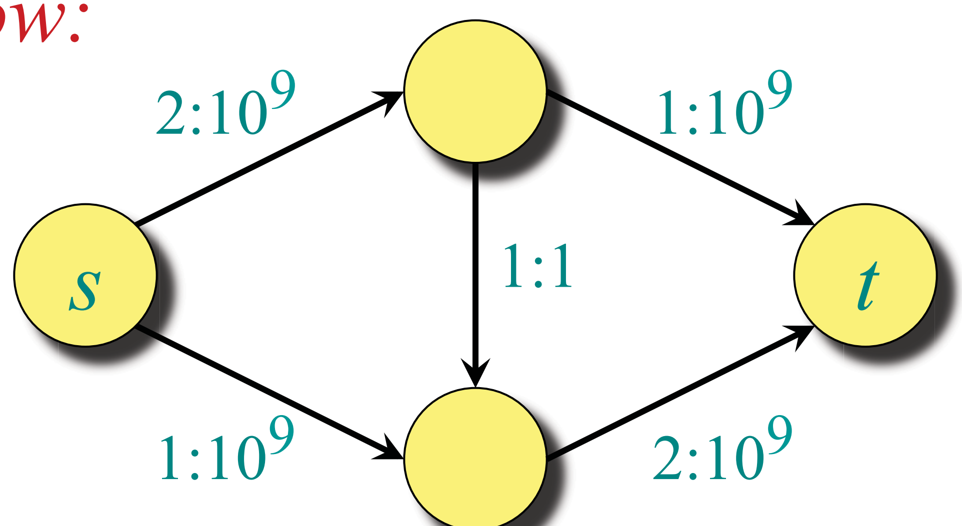
Ford-Fulkerson max-flow algorithm

Algorithm:

$f[u, v] \leftarrow 0$ for all $u, v \in V$
while an augmenting path p in G wrt f exists
do augment f by $c_f(p)$

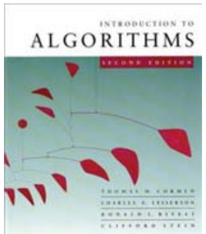
Can be slow:

G:



2 billion iterations on a graph with 4 vertices!

© 2001–15 by Leiserson et al

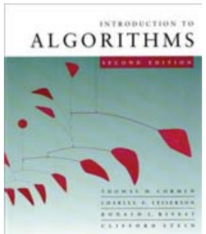


Edmonds-Karp algorithm

Edmonds and Karp noticed that many people' s implementations of Ford-Fulkerson augment along a ***breadth-first augmenting path***: a shortest path in G_f from s to t where each edge has weight 1. These implementations would always run relatively fast.

Since a breadth-first augmenting path can be found in $O(E)$ time, their analysis, which provided the first polynomial-time bound on maximum flow, focuses on bounding the number of flow augmentations.

(In independent work, Dinic also gave polynomial-time bounds.)



Best to date

- The Edmonds-Karp maximum-flow algorithm runs in $O(VE^2)$ time.
  - Breadth-first search takes $O(E)$ time
  - $O(VE)$ augmentations in worst case
- The asymptotically fastest algorithm through 2011 for maximum flow, due to King, Rao, and Tarjan, runs in $O(VE \log_{E/(V \lg V)} V)$ time.
- Recently Orlin came up with an $O(VE)$ time algorithm!
  - One variant uses fast matrix multiplication

Flow Integrality

- Claim: Suppose the flow network has integer capacities. Then, the maximum flow will be integer-valued.

  Proof: Start with a flow of 0 on all edges. Use Ford-Fulkerson. Initially, and at each step, Ford-Fulkerson will find an augmenting path with residual capacity that is an integer. Therefore, all flow values on edges always remain integral throughout the algorithm.

Applications

- Baseball Elimination
- Bipartite Matching
- Flow integrality important to reducing these problems to max flow!
- See additional notes for L14 for Baseball Elimination

MIT OpenCourseWare
http://ocw.mit.edu

6.046J / 18.410J Design and Analysis of Algorithms
Spring 2015

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.