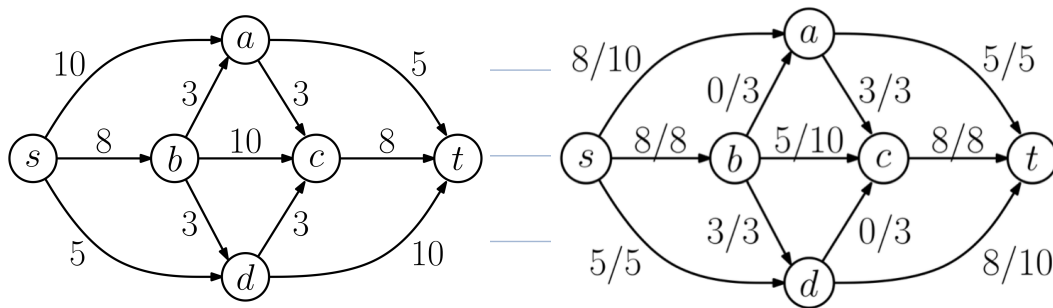


CMSC 451 - Algorithm Design

Lecture 13 - Network Flow - Algorithms

- Previous lecture -
- s-t Networks - source, sink, capacities
 - Flows - Capacity & Conservation
 - Max-Flow Problem



- Residual Network & Augmenting paths
- Ford-Fulkerson Algorithm
- Cuts
- Max-Flow/Min-Cut Theorem

- This lecture:
- Analysis of Ford-Fulkerson
 - Efficient algorithms
 - Scaling
 - Edmonds-Karp
 - Maximum bipartite matching

How Fast is Ford-Fulkerson (F-F)?

Observe:

If capacities are integers, F-F augmentations are integer valued. \Rightarrow

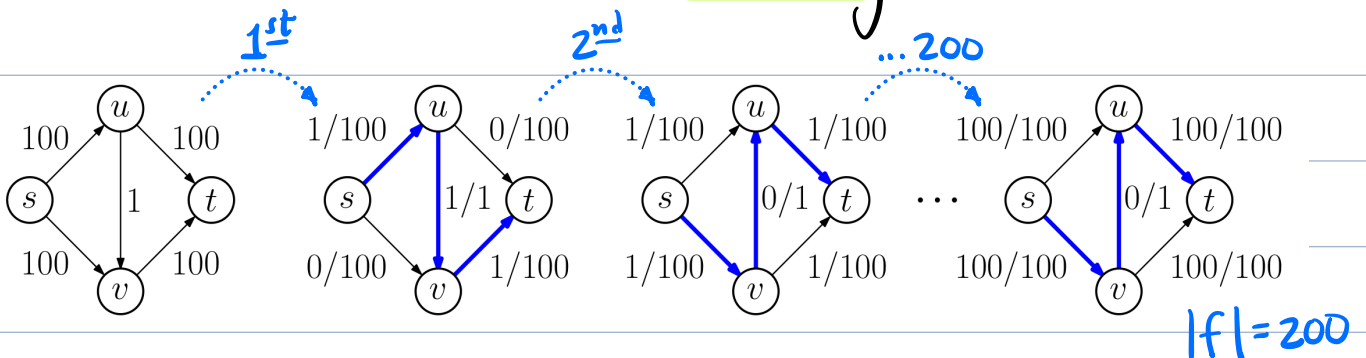
Lemma: Given an s-t network with integer-valued capacities, the max flow value will be an integer

F-F running time:

- Each iteration takes $O(n+m)$ time
- Assuming G is connected $O(n+m) = O(m)$
- Running time is $O(m \cdot (\text{num. of iterations}))$
- How many iterations?

F-F does not specify which augmenting path

- What if we are unlucky?



Number of iterations can be as large as $|f|$

- This is bad! (replace 100 with 1,000,000,000)

Generally - Let C be any upper bound on $|f|$

- F-F running time is $O(m \cdot C)$



Let's explore more efficient algorithms -

Scaling Algorithm (Gabow, 1980's)

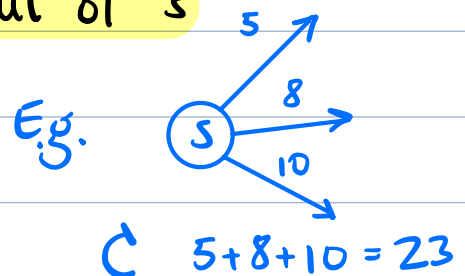
- Augment high capacity paths first
- Can compute max capacity s-t path in $O(m \log n)$ time
- Faster to compute a close-to-max capacity path in $O(m)$ time

Close-to-max capacity?

- Assume capacities are all integers
- $C \leftarrow$ any upper bound on max flow value
- Eg.

$C \leftarrow$ sum of capacities out of s

$$\leftarrow \sum_{(s,v) \in E} c(s,v)$$



- $\Delta \leftarrow$ largest power of 2 $\leq C$

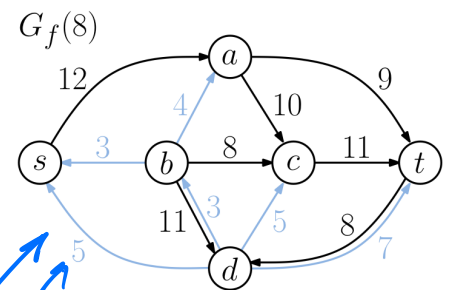
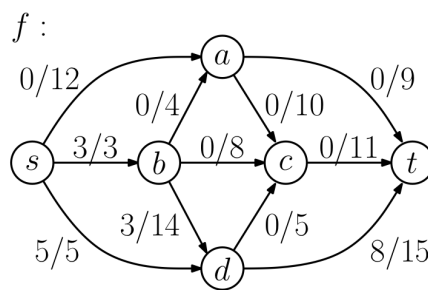
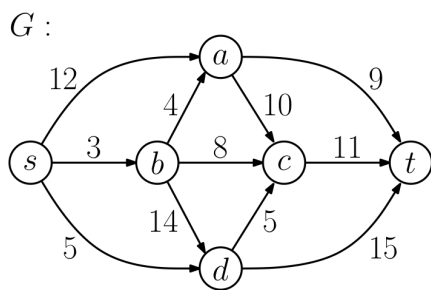
$\leftarrow 2^{\lfloor \log_2 C \rfloor}$

$\Delta \leftarrow 16$

- Initial flow: $f \leftarrow 0$ (0 flow on all edges)

- Given any flow $f + \Delta$, define

$G_f(\Delta) =$ residual network G_f keeping only edges of capacity $\geq \Delta$



By computing augmenting paths in $G_f(\Delta)$, flow increases rapidly

These edges are omitted since $< \Delta$

- Run F-F on $G_f(\Delta)$

- When no aug. path exists $\Delta \leftarrow \Delta/2$

- Stop when $\Delta < 1$.

scaling-flow(G) // scaling alg for network flow

$f \leftarrow 0$ // init flow is zero

$C \leftarrow \sum_{(s,v) \in E} c(s,v)$ // capacity out of s

$\Delta \leftarrow 2^{\lfloor \log_2 C \rfloor}$ // initial Δ

while ($\Delta \geq 1$) // stop if $\Delta < 1$

$G_f(\Delta) \leftarrow$ residual G_f , with // heavy residual
capacities $< \Delta$ removed

if ($G_f(\Delta)$ has an s - t path π) // augment

$c \leftarrow$ min capacity on π

$f \leftarrow$ add c to edges of π

else

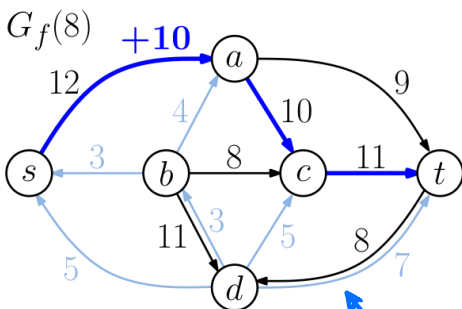
// no more augments

$\Delta \leftarrow \Delta/2$

// reduce Δ

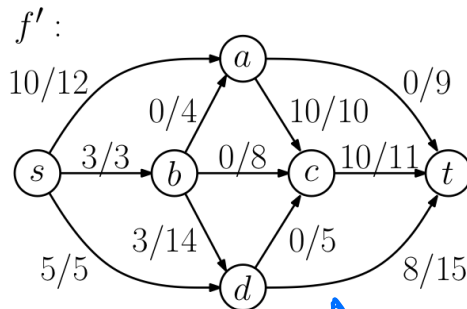
return f

Example: (see previous figure)

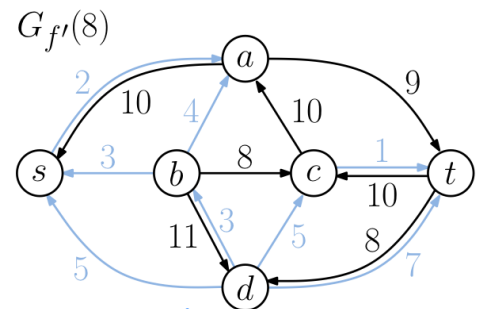


$\Delta = 8$

Find any path in $G_f(\Delta)$



Augment



- Update $G_f(\Delta)$
- No s - t path so
 $\Delta \leftarrow \Delta/2 = 4$

Correctness:

Same as F-F, just picking paths smarter.

Running time:

- Each time we augment in $G_f(\Delta)$

the flow increases by at least Δ

\Rightarrow residual capacities on these edges decrease by at least Δ

\Rightarrow After $O(m)$ augmentations, all residual capacities fall below Δ

\Rightarrow No augmentation $\Rightarrow \Delta \leftarrow \Delta/2$

Summary: $O(m)$ augmentations $\Rightarrow \Delta$ halved

- After $O(\log C)$ halvings, $\Delta < 1$

- From F-F, each augmentation takes $O(m)$ time.

- Total time: $O(m \cdot m \cdot \log C)$

Time per augmentation

Num. augmentations until Δ is halved

Number of halvings

$$= O(m^2 \log C)$$



Is this really efficient?

- C = sum of capacities is an input parameter
 - It could be arbitrarily large, independent of $n + m$
- $\xrightarrow{10}$ 😊
 $\xrightarrow{100}$ 😐
 $\xrightarrow{1,000,000,000}$ 😱!!

- "Efficient" \equiv Polynomial time

- Running time polynomial in

input size - $O(n+m)$, $O(n \cdot m^3)$, $O(n \log n)$

- As opposed to exponential time

- $O(2^n)$, $O(3^{n+m})$, $O(n!)$

- But $\log C$ is related to input size

= Num. of bits needed to rep. capacities

- So $O(m \cdot \log C)$ is a polynomial in input size

- if we are counting bits of input

To avoid confusion, we distinguish between:

Strongly Polynomial Time -

- Polynomial in number of words of input

(ignoring no. of bits)

- E.g. $O(n \cdot m^2)$, $O(n \log n)$, $O(n^{5.12} + m)$

Weakly Polynomial Time -

Polynomial in number of bits of input

- E.g. $O(m^2 \log C)$ [but not $O(m \cdot C)$]

Is there a strongly polynomial alg. for max flow?

Edmonds-Karp Algorithm

- Discovered indep. by Dinitz (Dinic)
+ Edmonds + Karp in early 1970's.

- Just run Ford-Fulkerson, but select the augmenting path with the fewest edges.

See text
for proof

- Converges in $O(n \cdot m)$ augmentations

- Total time = $O(n \cdot m^2)$

(Dinitz further reduced to $O(n^2 m)$.)

Better since
 $n \leq m \leq n^2$

Faster still?

- Goldberg + Tarjan (1986) - $O(n \cdot m \log \frac{n^2}{m})$

- King, Rao, Tarjan (1994) - $O(n \cdot m \cdot \frac{\log n}{\log(m/n \log n)})$

- Orlin (2013) - $O(n \cdot m)$

These are all quite complicated!

(To be continued)

(Figures for later)

