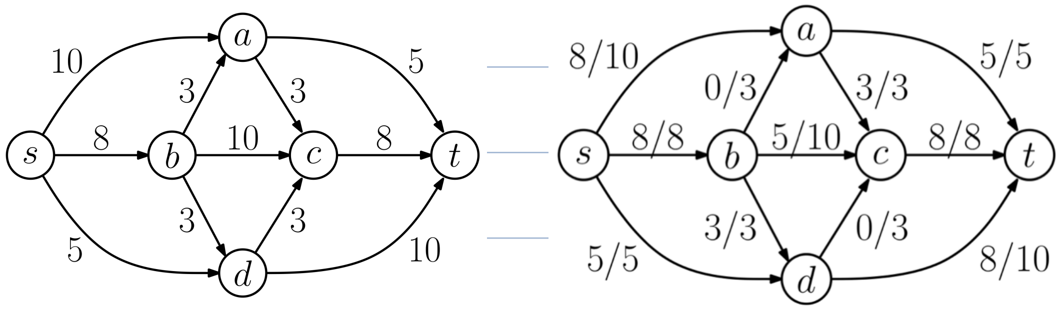
CMSC 451 - Algorithm Design
Lecture 13 - Network Flow - Algorithms

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. $\Rightarrow$

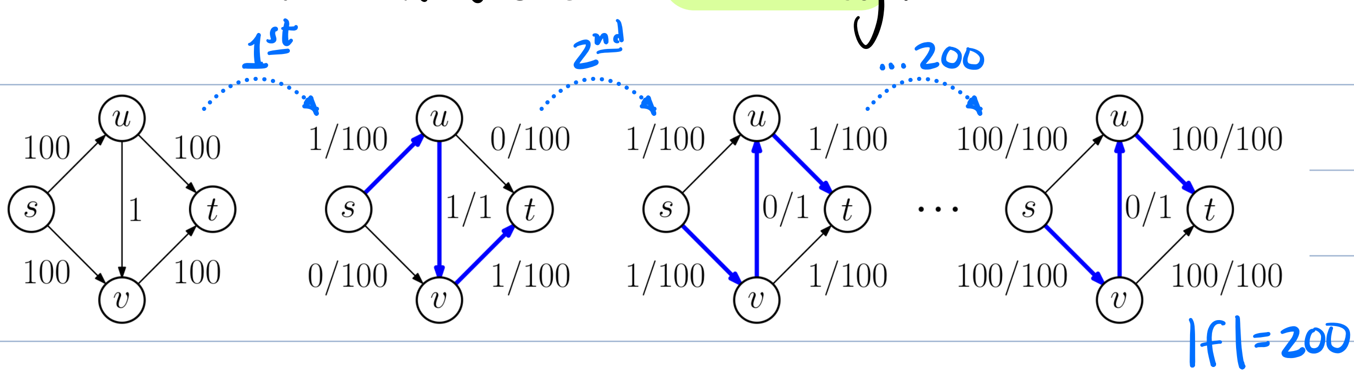
> **Lemma:** Given an s-t network with integer-valued capacities, the max flow value will be an integer

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?

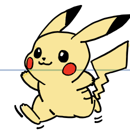


Number of iterations can be as large as $|f|$

- 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

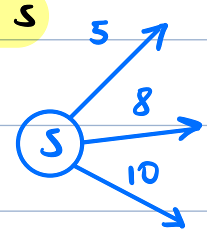
Close-to-max capacity?

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

$C \leftarrow$ sum of capacities out of s

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

E.g.



$C \quad 5+8+10=23$

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

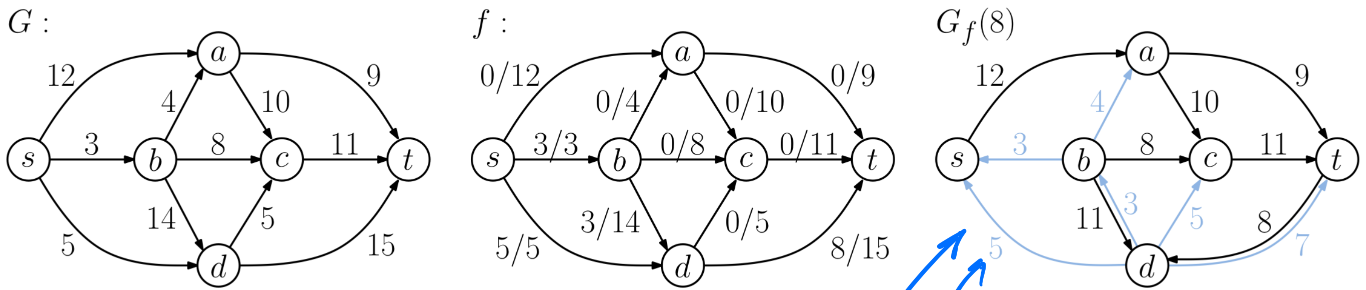
$\leftarrow 2^{\lfloor \log_2 C \rfloor}$

$\Delta \leftarrow 16$

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

- Given any flow f + Δ, define

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



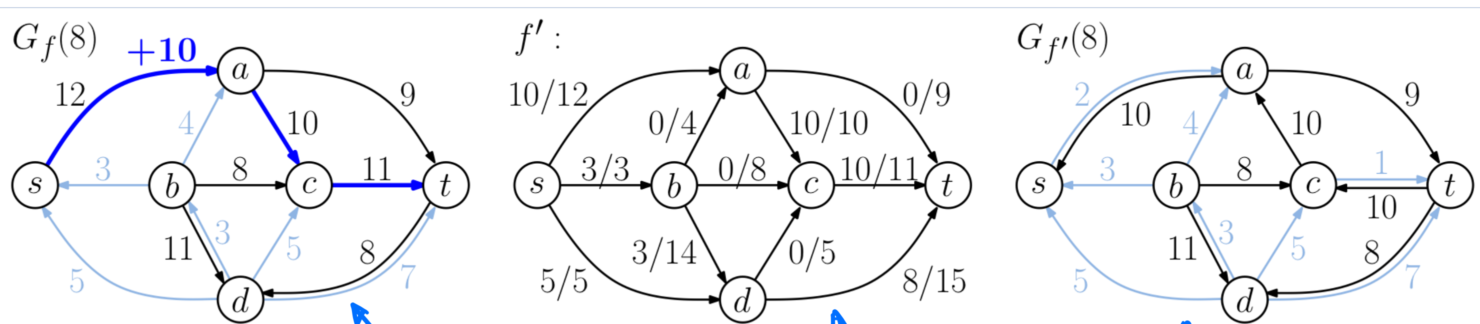
These edges are omitted since $< \Delta$

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

- 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 ← 0                                    // init flow is zero
    C ← Σ_{(s,v)∈E} c(s,v)                   // capacity out of s
    Δ ← 2^⌊log₂ C⌋                           // initial Δ
    while (Δ ≥ 1)                            // stop if Δ<1
        G_f(Δ) ← residual G_f, with          // heavy residual
                 capacities < Δ removed
        if (G_f(Δ) has an s-t path π)        // augment
            c ← min capacity on π
            f ← add c to edges of π
        else                                 // no more augments
            Δ ← Δ/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 $\Rightarrow$ Δ halved

- After $O(\log C)$ halvings, $\Delta < 1$
- From F-F, each augmentation takes $O(m)$ time.

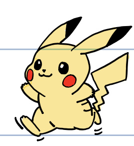
- Total time: $O(m \cdot m \cdot \log C)$

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" ≡ 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) - $O\left(n \cdot m \log \frac{n^2}{m}\right)$
- King, Rao, Tarjan (1994) - $O\left(n \cdot m \cdot \frac{\log n}{\log(m/n \log n)}\right)$
- Orlin (2013) - $O(n \cdot m)$

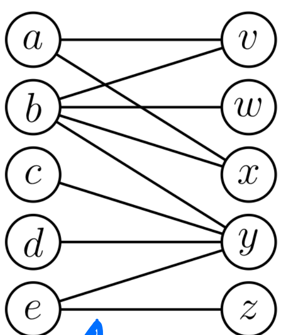
These are all quite complicated!

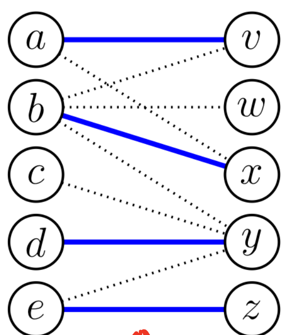
Applications of Network Flow:

Maximum matching in bipartite graphs

- Given two sets of objects
  (e.g. males + females, students + grad schools, customers + agents)
- ... and some compatibility relation:
  (e.g. female swipes right on male, student wants to attend grad school, agent has expertise to serve customer)
- ... pair up as many as possible (one to one)

Bipartite Graph

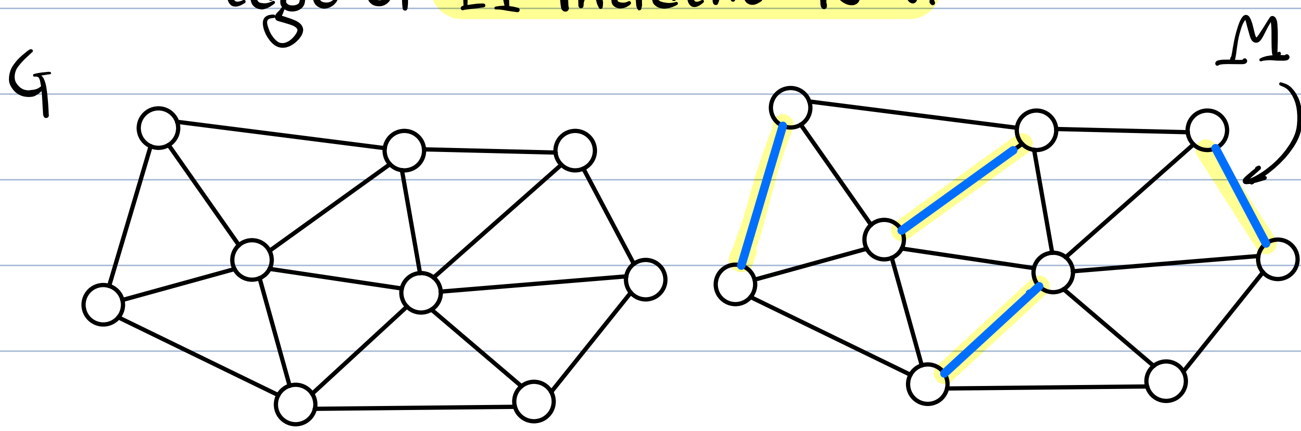




edge ≡ compatible

pairing up

Def: Given a graph $G=(V,E)$, a matching is a subset of edges $M \subseteq E$ such that for each $v \in V$, there is at most one edge of M incident to v.



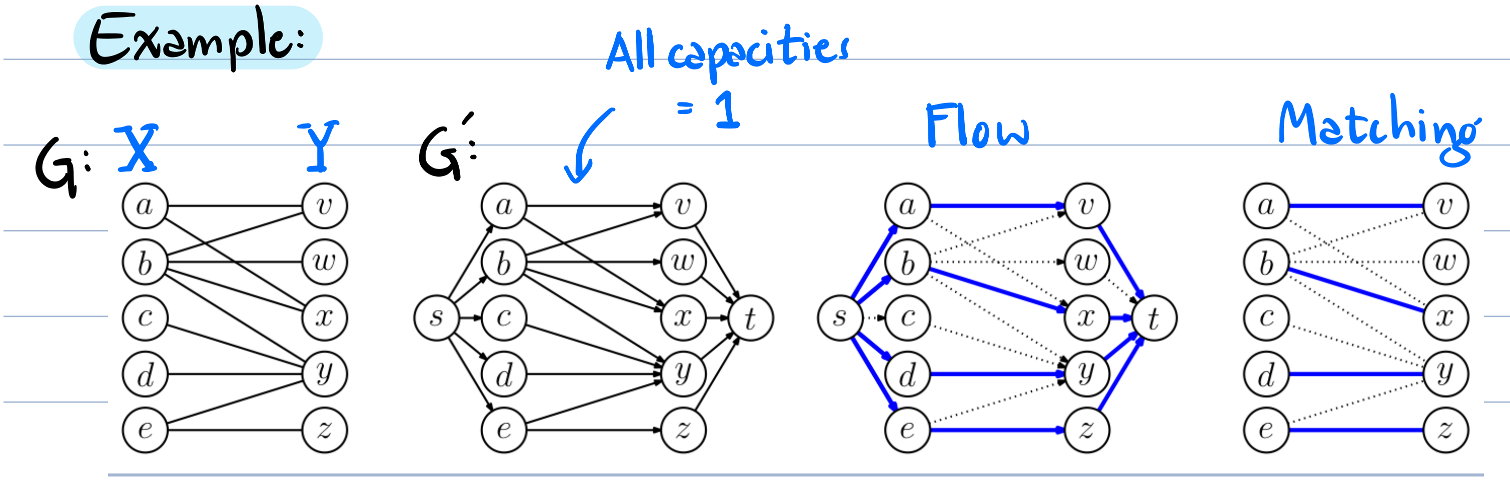
Recall: G is bipartite if $V = X \cup Y$ + all edges $\in X \times Y$

Problem: Given a bipartite graph G, compute the max. sized matching in G.

> **Claim:** We can reduce max. bipartite matching to network flow.

- Create source s + add edges (s,u) $\forall u \in X$
- Create sink t + add edges (v,t) $\forall v \in Y$
- All capacities $= 1$

Example:



> **Claim:** G has a matching of size k iff G′ has a flow of value k

> **Proof:** (Sketch)
> - Because capacity-in + capacity-out = 1, cannot push more than one unit of flow through each vertex
> - ⇒ At most edge will carry flow (assuming integer flow values)

Summary:

- Ford-Fulkerson - Could be very slow
- Scaling Algorithm - $O(m^2 \log C)$ simple
- Strong / Weak Polynomial Time
- Application - Bipartite Max Matching