CSE 421
Introduction to Algorithms

Lecture 17: Polynomial-Time MaxFlow/MinCut Algorithms

Come get Halloween candy at the front

Announcements

Midterm next **Monday, November 4, 6:00 – 7:30 pm in this room**

- Use Monday's class time to study
- See post on Important Midterm Information
- Links to sample midterm, practice problems, and reference sheet posted earlier this week
- Zoom review session for Q&A this Sunday Nov 3 at 4:45 pm.

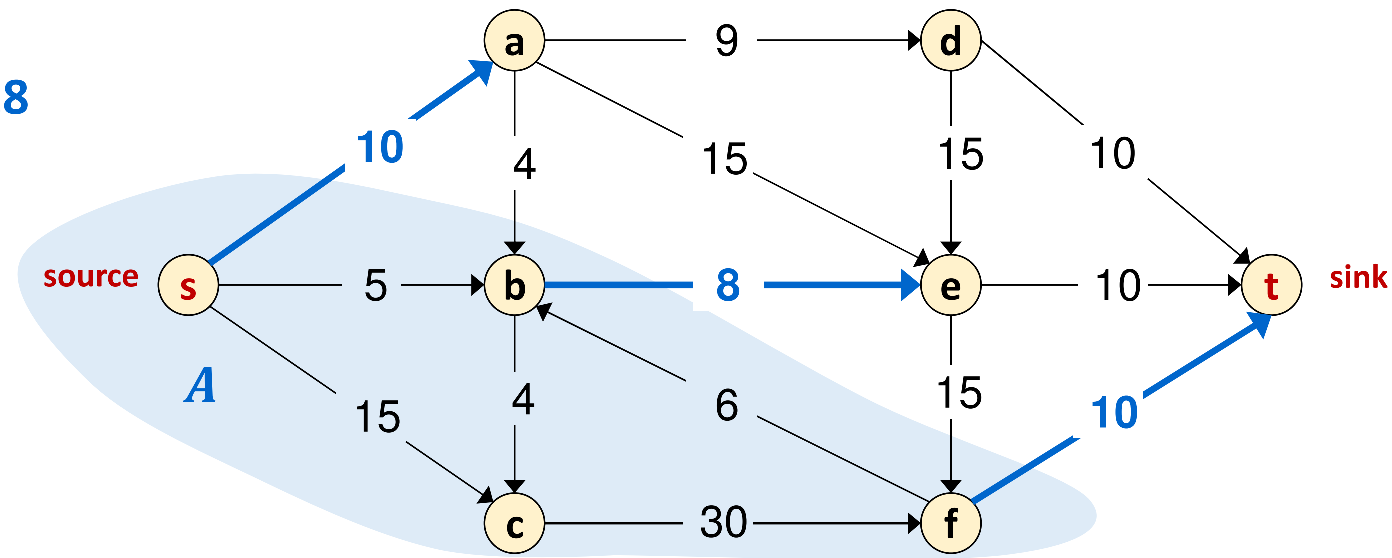
Minimum Cut Problem

Minimum s-t cut problem:

Given: a flow network

Find: an s-t cut (A, B) of minimum capacity $c(A, B) = \sum_{e \text{ out of } A} c(e)$

capacity **28**

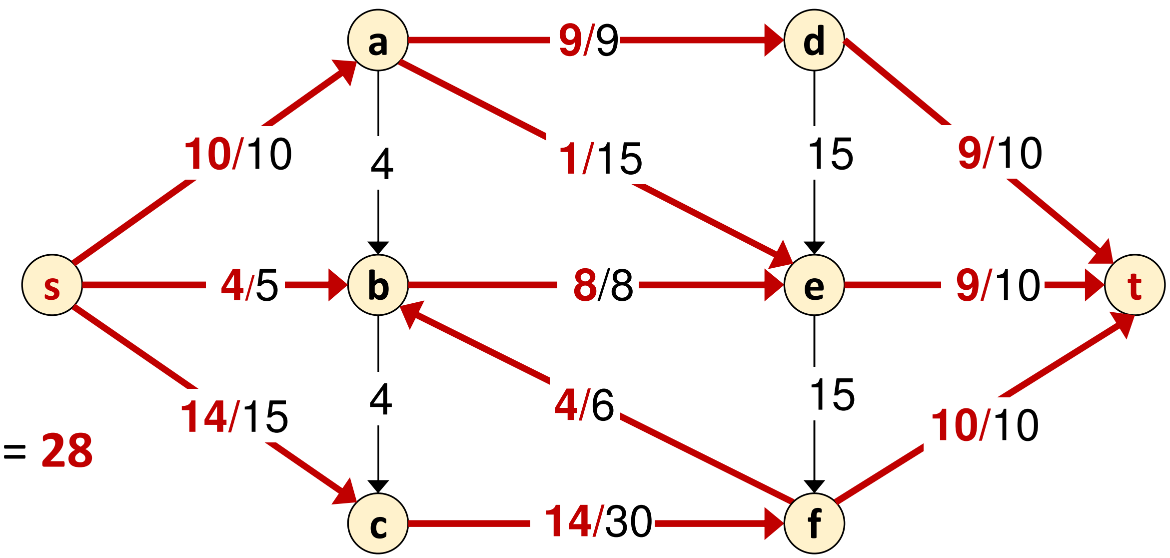


Maximum Flow Problem

Given: a flow network

Find: an s-t flow of maximum value

value = **28**



Ford-Fulkerson Augmenting Path Algorithm

```
Ford-Fulkerson(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   Gf ← G

   while (Gf has an s-t path P) {
      f ← Augment(f, c, P)
      update Gf
   }
   return f
}
```

```
Augment(f, c, P) {
   b ← bottleneck(P)
   foreach e ∈ P {
      if  (e ∈ E)  f(e)  ← f(e)  + b
      else         f(eR)← f(eR)  - b
   }
   return f
}
```

MaxFlow/MinCut & Ford-Fulkerson Algorithm

Augmenting Path Theorem: Flow f is a max flow $\Leftrightarrow$ there are no augmenting paths wrt f

Max-Flow Min-Cut Theorem: The value of the max flow equals the value of the min cut.
[Elias-Feinstein-Shannon 1956, Ford-Fulkerson 1956] **"MaxFlow = MinCut"**

Flow Integrality Theorem: If all capacities are integers then there is a maximum flow with all-integer flow values.

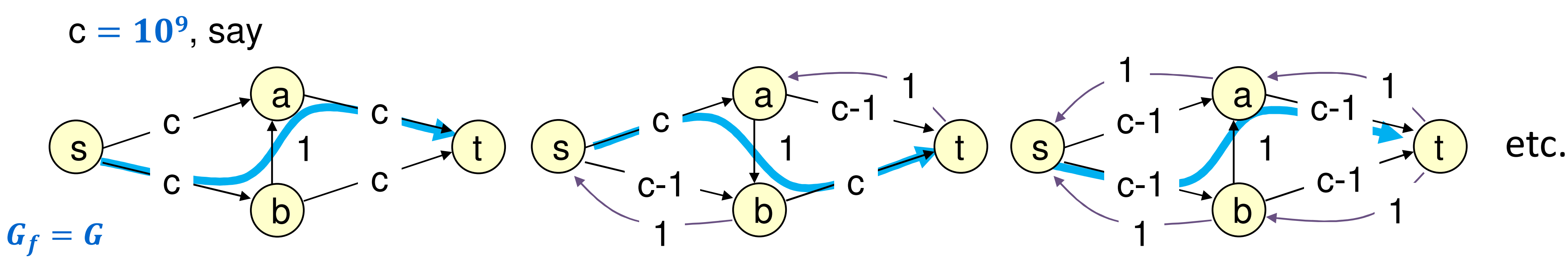
Ford-Fulkerson Algorithm: $O(m)$ per iteration. With integer capacities each at most C need at most **MaxFlow** $< nC$ iterations for a total of $O(mnC)$ time.

Ford-Fulkerson Efficiency

Worst case runtime $O(mnC)$ with integer capacities $\le C$.

- $O(m)$ time per iteration.
- At most nC iterations.
- This is “pseudo-polynomial” running time.

- May take exponential time, even with integer capacities:



Choosing Good Augmenting Paths

Polynomial-Time Variants of Ford-Fulkerson

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.
- If capacities are irrational, algorithm not guaranteed to terminate!

Goal: Choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

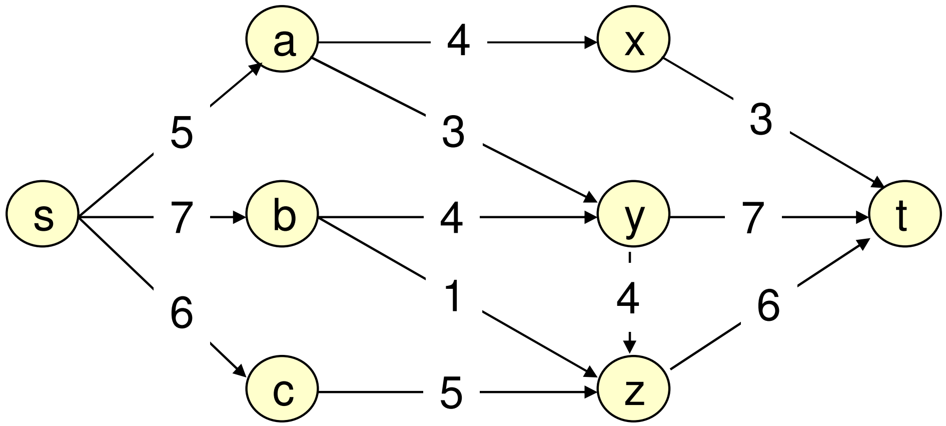
- Choose augmenting paths with: [Edmonds-Karp 1972, Dinitz 1970]
  - Max bottleneck capacity.
  - Sufficiently large bottleneck capacity.
  - Fewest number of edges.

Polynomial-Time MaxFlow: Capacity Scaling

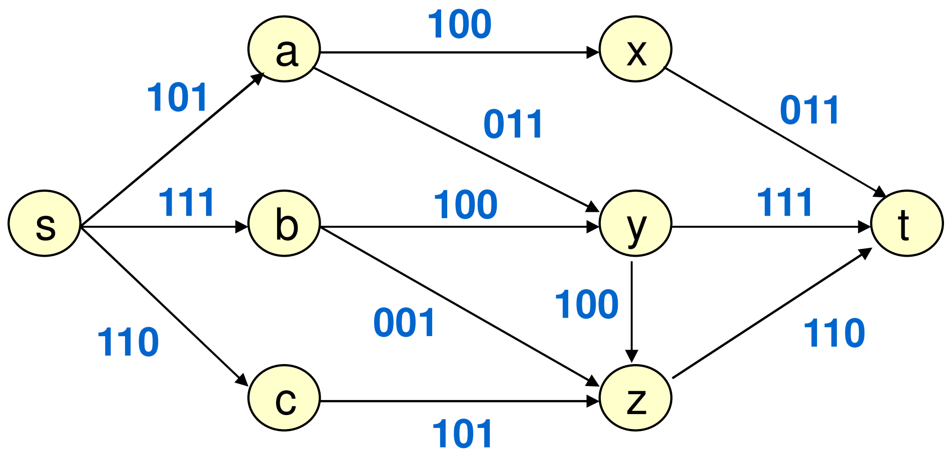
General idea:

- Choose augmenting paths **P** with 'large' capacity.
- Can augment flows along a path **P** by any amount $\leq$ **bottleneck**$(\boldsymbol{P})$
  - Ford-Fulkerson still works
- Choose that amount to be "nice round number" (i.e. a big power of 2.)
- Get a flow that is maximum for the high-order bits first and then add more bits later

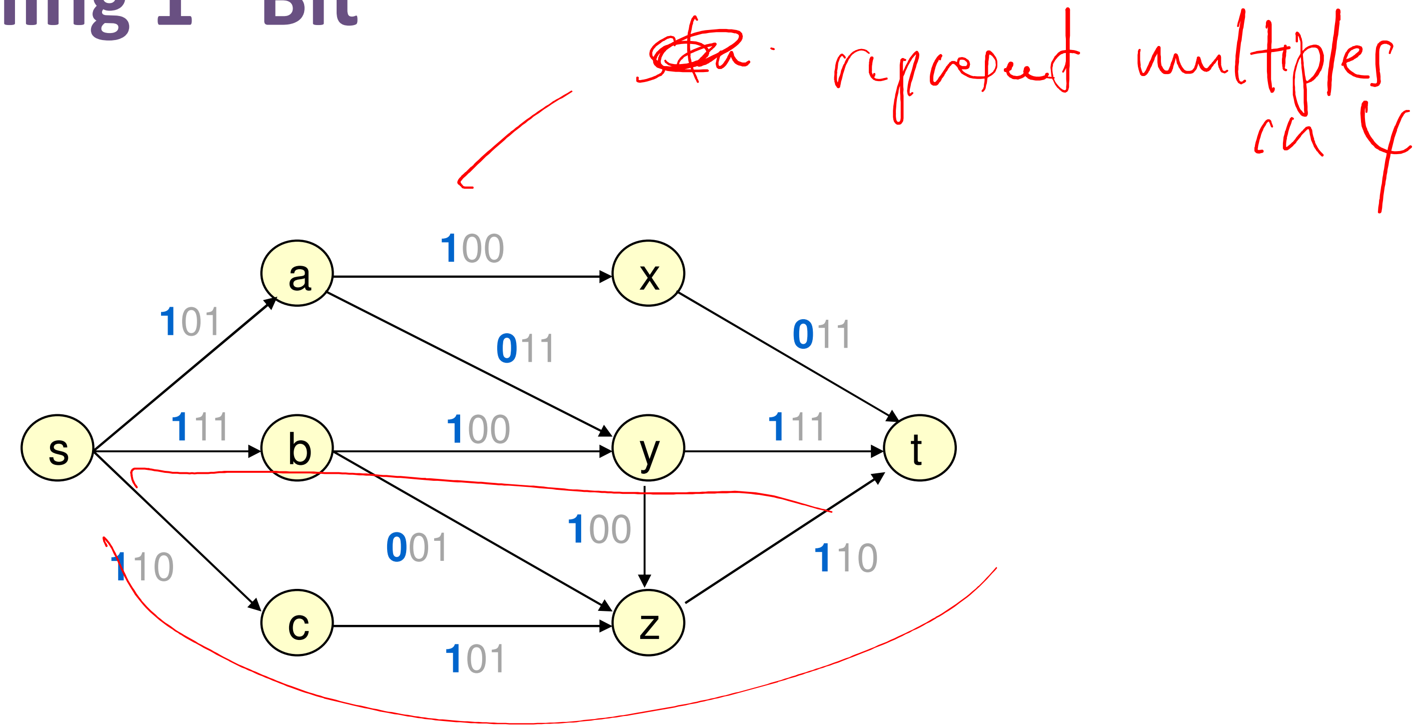
Capacity Scaling



Write Capacities in Binary

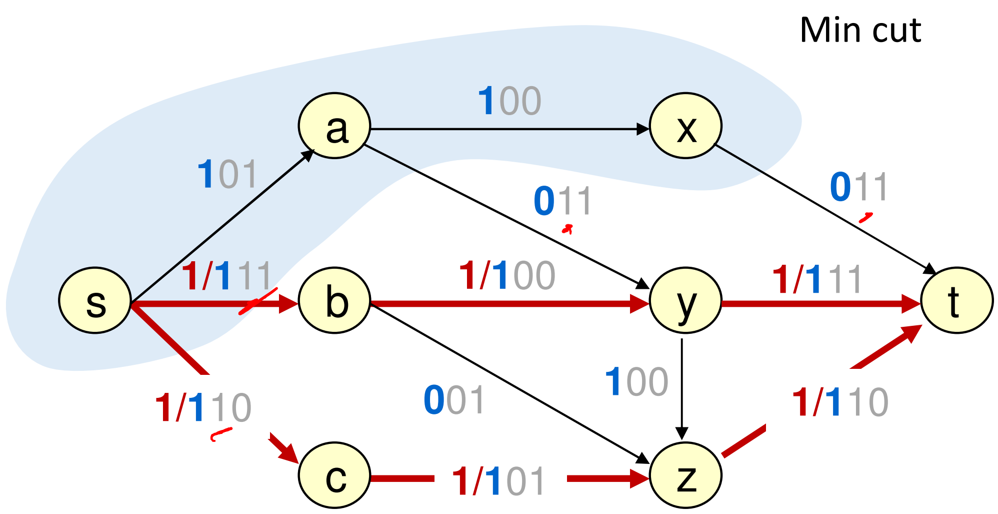


Capacity Scaling 1st Bit



Solve flow problem with capacities with just the high-order bit:

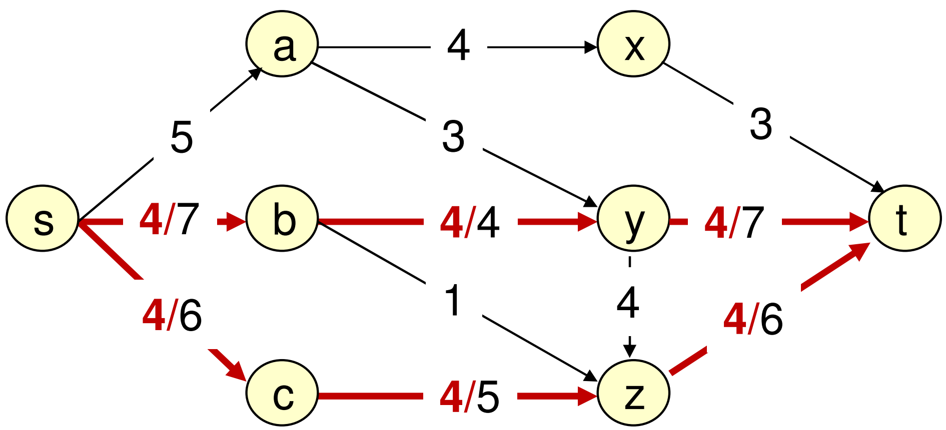
Capacity Scaling 1st Bit



Solve flow problem with capacities with just the high-order bit:

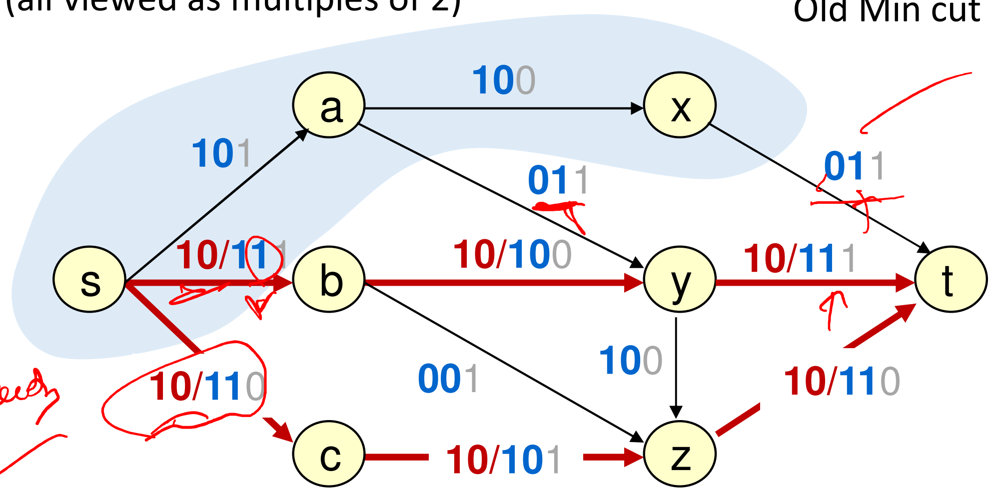
- Each edge has "capacity" ≤ 1 (equivalent to 4 here)
- Time $O(mn)$

Capacity Scaling 1st Bit



Capacity Scaling add 2nd Bit

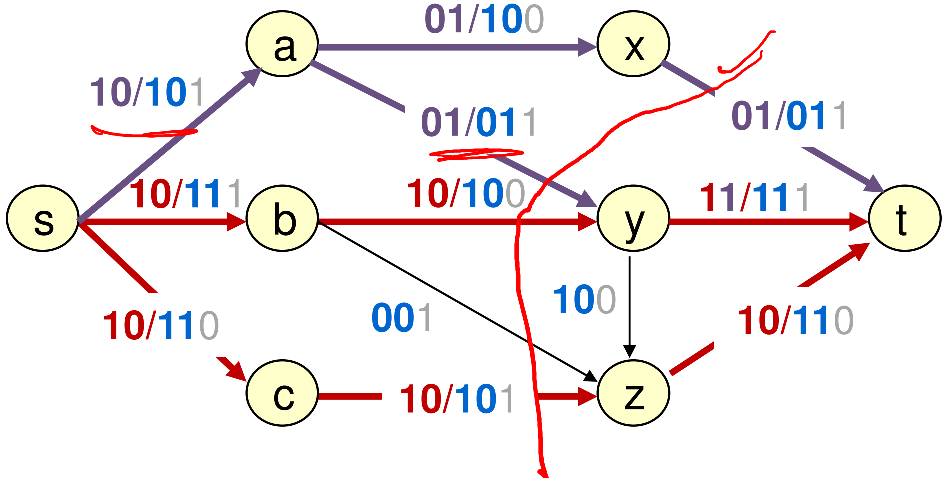
Add 0 bit to the end of the flows
Add 2nd bit to capacities (all viewed as multiples of 2)



Solve flow problem with capacities with the **2** high-order bits:

- Capacity of old min cut goes up by ≤ 1 per edge (equivalent to 2 here) for a total residual capacity $\leq m$.

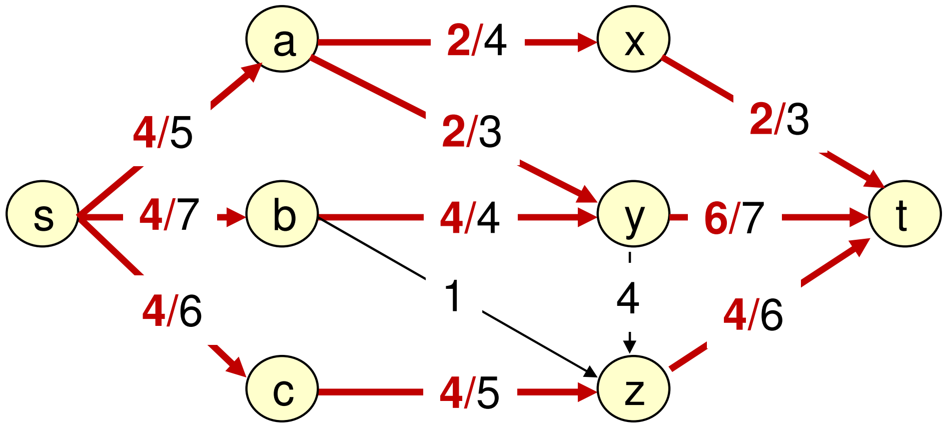
Capacity Scaling add 2nd Bit



Solve flow problem with capacities with the **2** high-order bits:

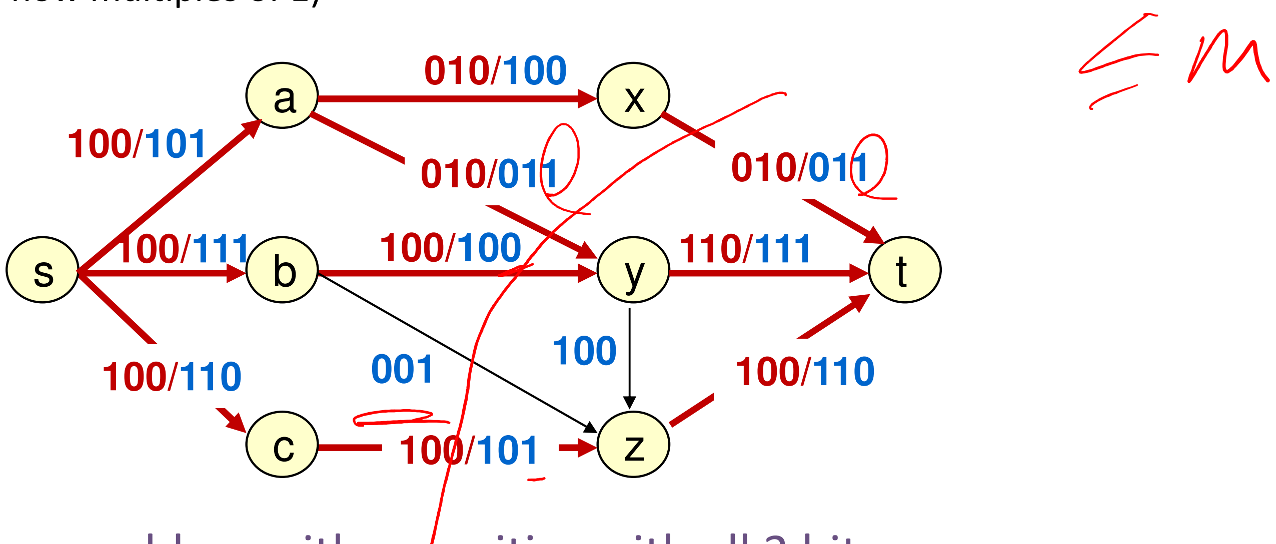
- Capacity of old min cut goes up by $\leq \mathbf{1}$ per edge (equivalent to $\mathbf{2}$ here) for a total residual capacity $\leq \boldsymbol{m}$.
- Time $O(\boldsymbol{m}^2)$ for $\leq \boldsymbol{m}$ iterations.

Capacity Scaling 1st and 2nd Bits



Capacity Scaling add 3rd Bit

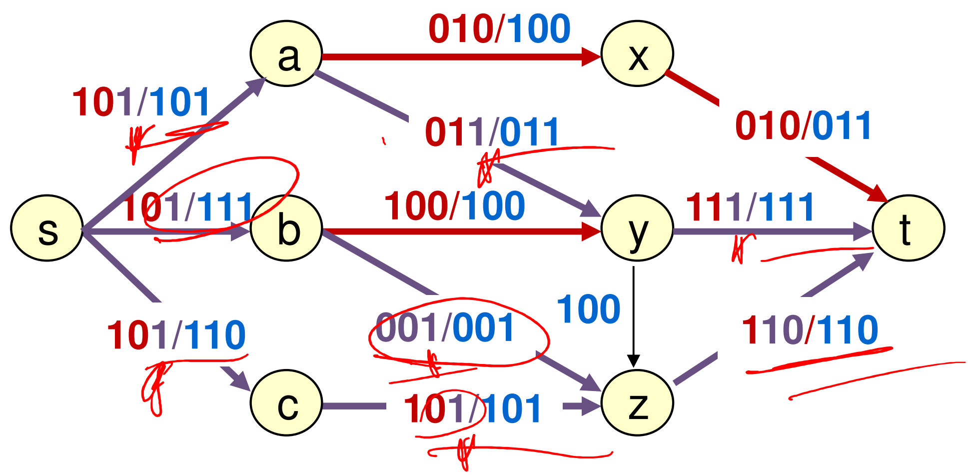
Add 0 bit to the end of the flows
Add 3rd bit to capacities (all now multiples of 1)



Solve flow problem with capacities with all 3 bits:

- Capacity of old min cut goes up by ≤ 1 per edge for a total residual capacity $\leq m$.

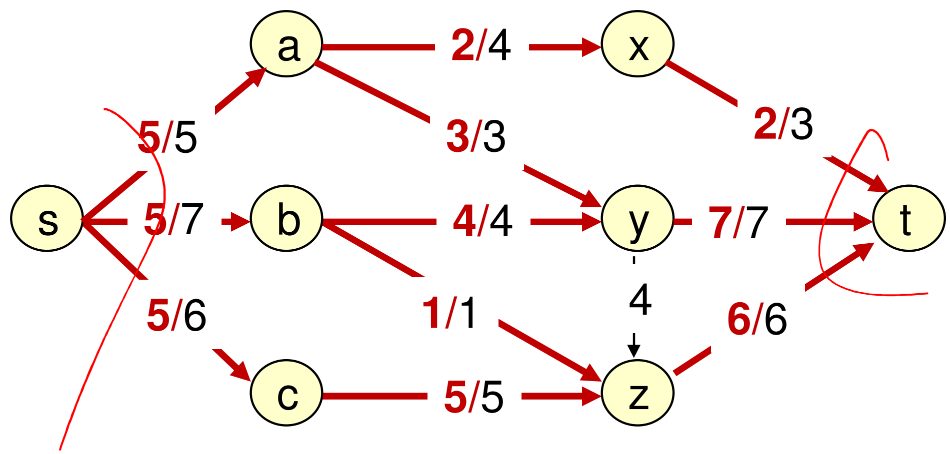
Capacity Scaling add 3rd Bit



Solve flow problem with capacities with all 3 bits:

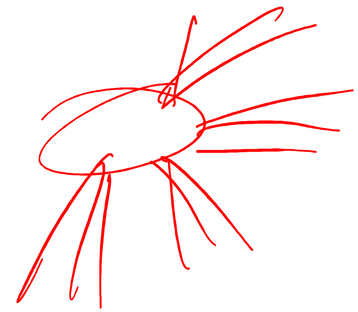
- Capacity of old min cut goes up by ≤ 1 per edge for a total residual capacity $\leq m$.
- Time $O(m^2)$ for $\leq m$ iterations.

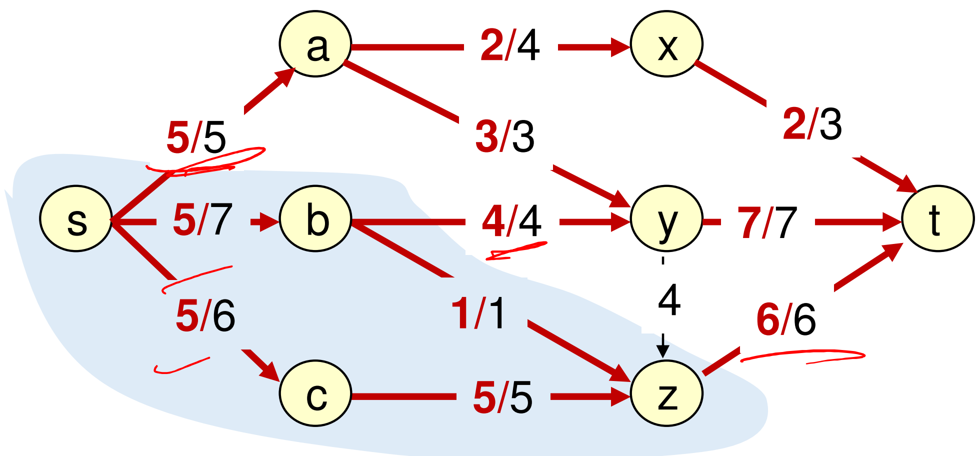
Capacity Scaling All Bits



Flow = **15**

Capacity Scaling All Bits





Flow = **15**

Cut Value = **15**

Flow is a MaxFlow

Total time for capacity scaling

- Number of rounds = $\lceil \log_2 \boldsymbol{C} \rceil$ where $\boldsymbol{C}$ is the largest capacity
- Time per round $O(\boldsymbol{m}^2)$
  - At most $\boldsymbol{m}$ augmentations per round
  - $O(\boldsymbol{m})$ time per augmentation

Total time $O(\boldsymbol{m}^2 \log \boldsymbol{C})$

Great!   This is now polynomial time in the input size.

Can we get more?

- What about an algorithm with a number of arithmetic operations that doesn't depend on the size of the numbers?

Polynomial-Time Variants of Ford-Fulkerson

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.
- If capacities are irrational, algorithm not guaranteed to terminate!

Goal: Choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

- Choose augmenting paths with: [Edmonds-Karp 1972, Dinitz 1970]
    - Max bottleneck capacity.
    - Sufficiently large bottleneck capacity.
    - Fewest number of edges. (i.e. just run BFS to find an augmenting path.)

Edmonds-Karp Algorithm (Ford-Fulkerson with BFS)

Use Breadth First Search as the search algorithm to find an s-t path in G_f.

- Using any shortest augmenting path

Theorem: Ford-Fulkerson using BFS terminates in $O(m^2 n)$ time. [Edmonds-Karp, Dinitz]

"One of the most obvious ways to implement Ford-Fulkerson is always polynomial time"

Why might this be good intuitively?

- Longer augmenting paths involve more edges so may be more likely to hit a low residual capacity one which would limit the amount of flow improvement.

The proof uses a completely different idea…

Edmonds-Karp Algorithm (Ford-Fulkerson with BFS)

Analysis Focus:

For any edge e that could be in the residual graph G_f, (either an edge in G or its reverse)

count # of iterations that e is the first bottleneck edge on the augmenting path chosen by the algorithm.

Claim: This can't happen in more than $n/2$ iterations.

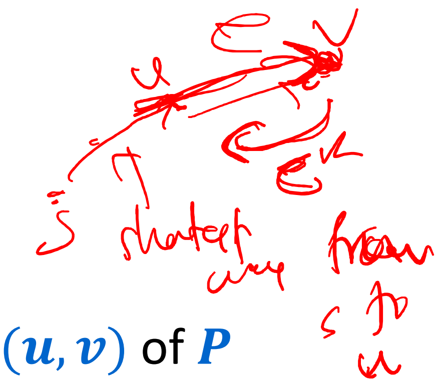
Proof: Write $e = (u, v)$.

Show that each time it happens, the distance from s to u in the residual graph G_f is at least 2 more than it was the last time.

This would be enough since the distance is $< n$
(or infinite and hence u isn't reachable) so this can happen at most $n/2$ times.

Distances in the Residual Graph

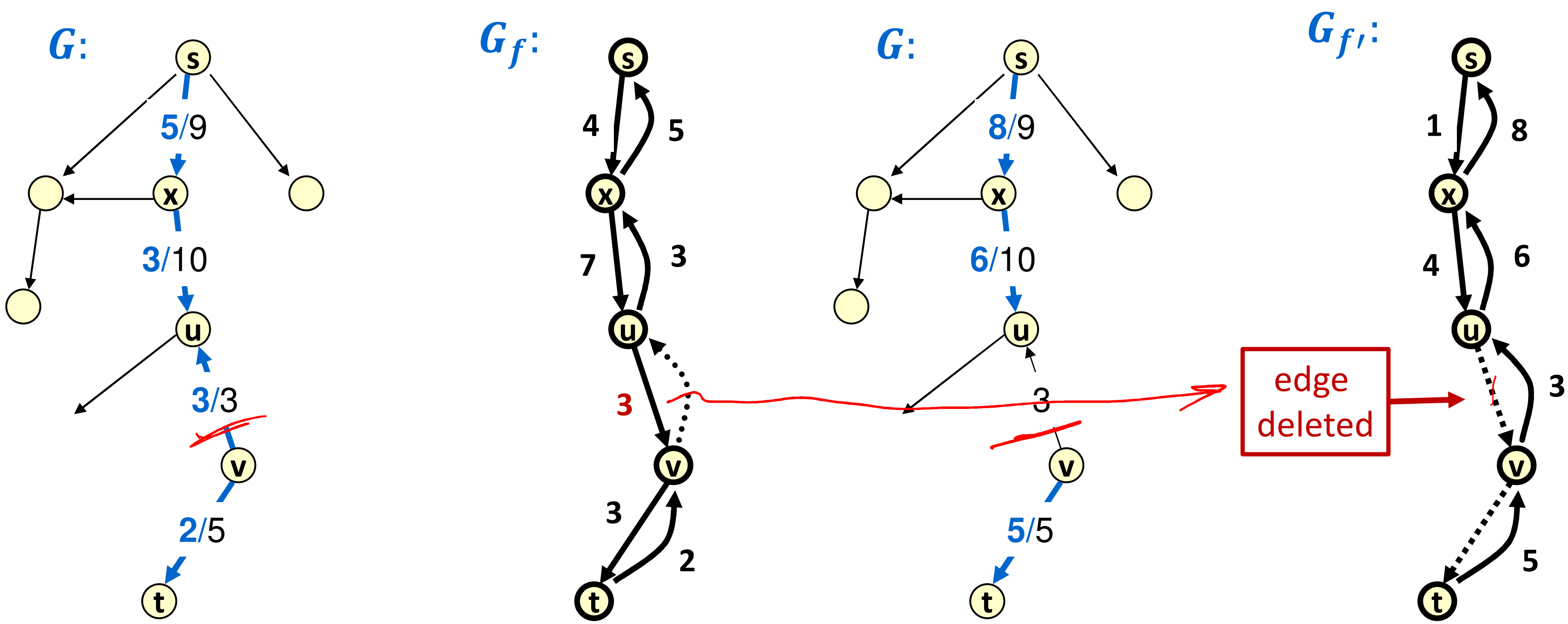
Key Lemma: Let f be a flow, G_f the residual graph, and P be a shortest augmenting path. No vertex is closer to s in the residual graph after augmenting along P.



Proof: Augmenting along P can only change the edges in G_f by either:

1. Deleting a forward edge
   - Deleting any edge can never reduce distances
2. Add a backward edge (v, u) that is the reverse of an edge (u, v) of P
   - Since P was a shortest path in G_f, the distance from s to v in G_f is already more than the distance from s to u. Using the new backward edge (v, u) to get to u would be an even longer path to u so it is never on a shortest path to any node in the new residual graph. ■

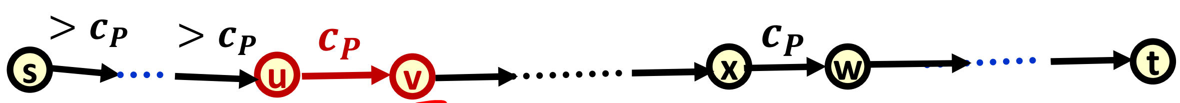
Augmentation vs BFS



First Bottleneck Edges in G_f

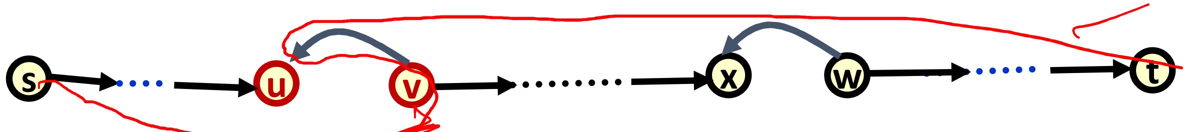
Shortest s-t path P in G_f

Write c_P = bottleneck(P)



$d_f(s, v) = d_f(s, u) + 1$ since P is a shortest path.

After augmenting along P, edge (u, v) disappears; but will have edge (v, u)



distance is ≥ 2 larger than before

For (u, v) to be a first bottleneck edge later, it must get added back to the residual graph by augmenting along a shortest path P' containing (v, u) in $G_{f'}$ for some flow f'

Since P' is shortest $d_{f'}(s, u) = d_{f'}(s, v) + 1 \geq d_f(s, v) + 1 = d_f(s, u) + 2$

The next time that (u, v) is first bottleneck edge is even later so distance is at least as large! ■

Edmonds-Karp Algorithm (Ford-Fulkerson with BFS)

Analysis Focus:

For any edge $\boldsymbol{e}$ that could be in the residual graph $\boldsymbol{G_f}$, (either an edge in $\boldsymbol{G}$ or its reverse)

count # of iterations that $\boldsymbol{e}$ is the first bottleneck edge on the augmenting path chosen by the algorithm.

Claim: This can’t happen in more than $\boldsymbol{n/2}$ iterations

Claim ⇒ Theorem:

Only $\boldsymbol{2m}$ edges and $O(\boldsymbol{m})$ time per iteration so $O(\boldsymbol{m^2 n})$ time overall. ■

Which is better in practice $O(\boldsymbol{m^2 n})$ vs. $O(\boldsymbol{m^2} \log \boldsymbol{C})$?

History & State of the Art for MaxFlow Algorithms

#	year	discoverer(s)	bound
1	1951	Dantzig	$O(n^2 mU)$
2	1955	Ford & Fulkerson	$O(nmU)$
3	1970	Dinitz Edmonds & Karp	$O(nm^2)$
4	1970	Dinitz	$O(n^2 m)$
5	1972	Edmonds & Karp Dinitz	$O(m^2 \log U)$
6	1973	Dinitz Gabow	$O(nm \log U)$
7	1974	Karzanov	$O(n^3)$
8	1977	Cherkassky	$O(n^2 \sqrt{m})$
9	1980	Galil & Naamad	$O(nm \log^2 n)$
10	1983	Sleator & Tarjan	$O(nm \log n)$
11	1986	Goldberg & Tarjan	$O(nm \log(n^2/m))$
12	1987	Ahuja & Orlin	$O(nm + n^2 \log U)$
13	1987	Ahuja et al.	$O(nm \log(n\sqrt{\log U}/(m+2))$
14	1989	Cheriyan & Hagerup	$E(nm + n^2 \log^2 n)$
15	1990	Cheriyan et al.	$O(n^3 / \log n)$
16	1990	Alon	$O(nm + n^{8/3} \log n)$
17	1992	King et al.	$O(nm + n^{2+\epsilon})$
18	1993	Phillips & Westbrook	$O(nm(\log_{m/n} n + \log^{2+\epsilon} n))$
19	1994	King et al.	$O(nm \log_{m/(n \log n)} n)$
20	1997	Goldberg & Rao	$O(m^{3/2} \log(n^2/m) \log U)$ $O(n^{2/3} m \log(n^2/m) \log U)$

Source: Goldberg & Rao, FOCS '97

21	**2013**	**Orlin**	$O(mn)$
22	**2014**	**Lee & Sidford**	$m\sqrt{n} \log^{O(1)} n \log U$
23	**2016**	**Madry**	$m^{10/7} U^{1/7} \log^{O(1)} n$
24	**2021**	**Gao, Liu, & Peng**	$m^{3/2-1/328} \log^{O(1)} n \log U$
25	**2022**	**van den Brand et al.**	$m^{3/2-1/58} \log^{O(1)} n \log U$
26	**2022**	**Chen et al.**	$m^{1+o(1)} \log U$

Tables use U instead of C for the upper bound on capacities

Methods:

Augmenting Paths – increase flow to capacity

Preflow-Push – decrease flow to get flow conservation

Linear Programming – randomized, high probability of optimality