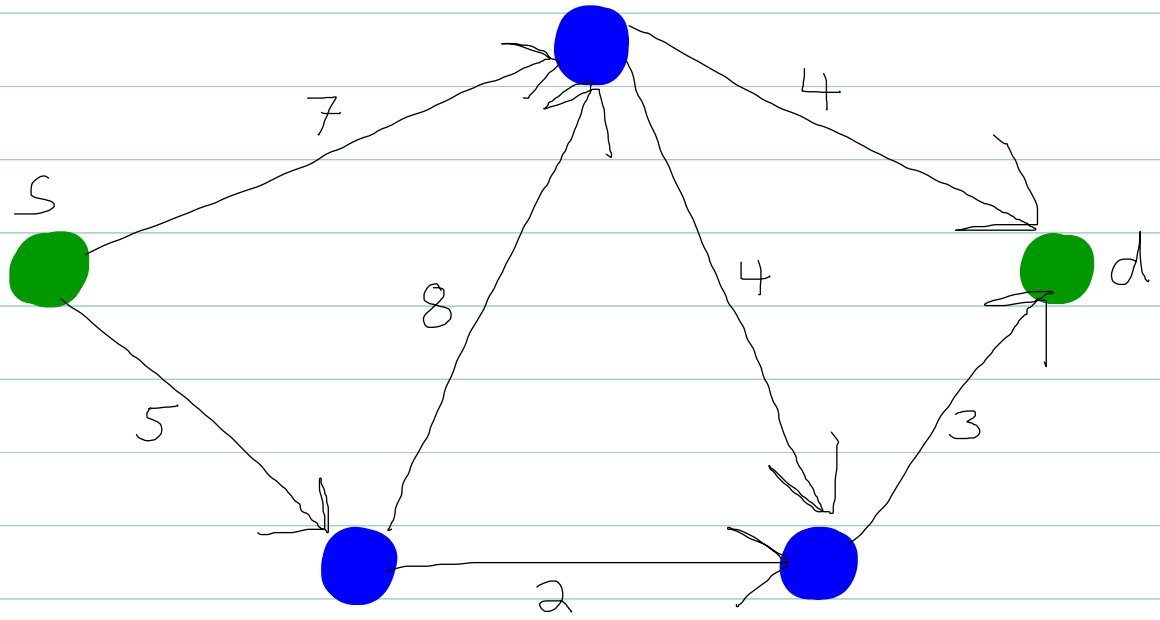
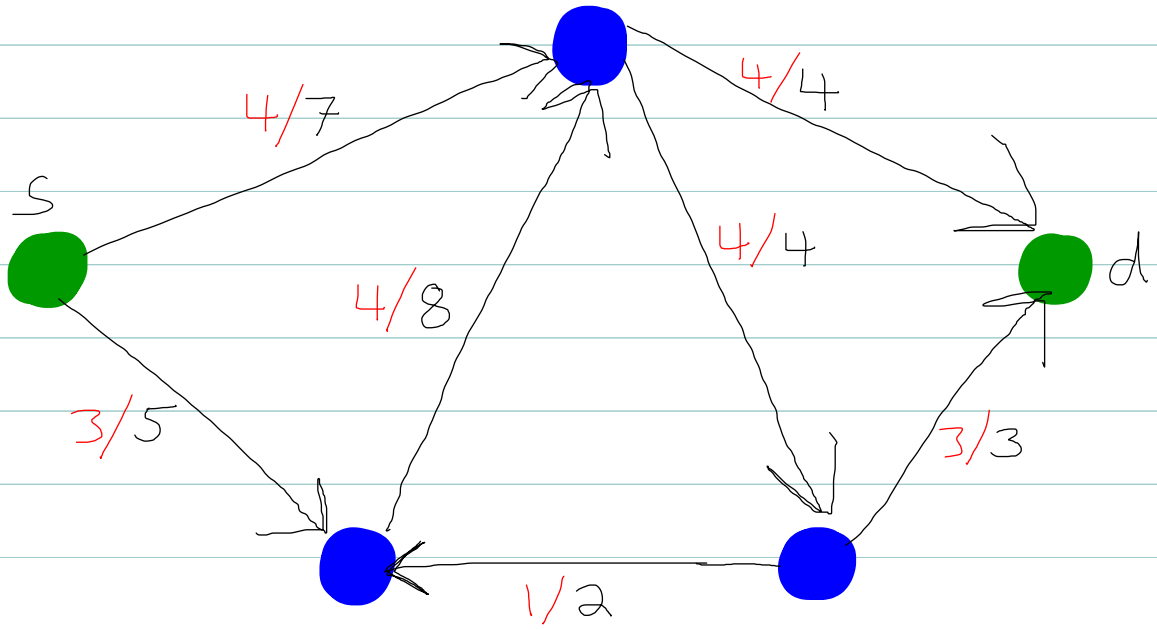
Quick Review:

Flow network = directed graph
- with source s
- with destination d
- with capacities on edges



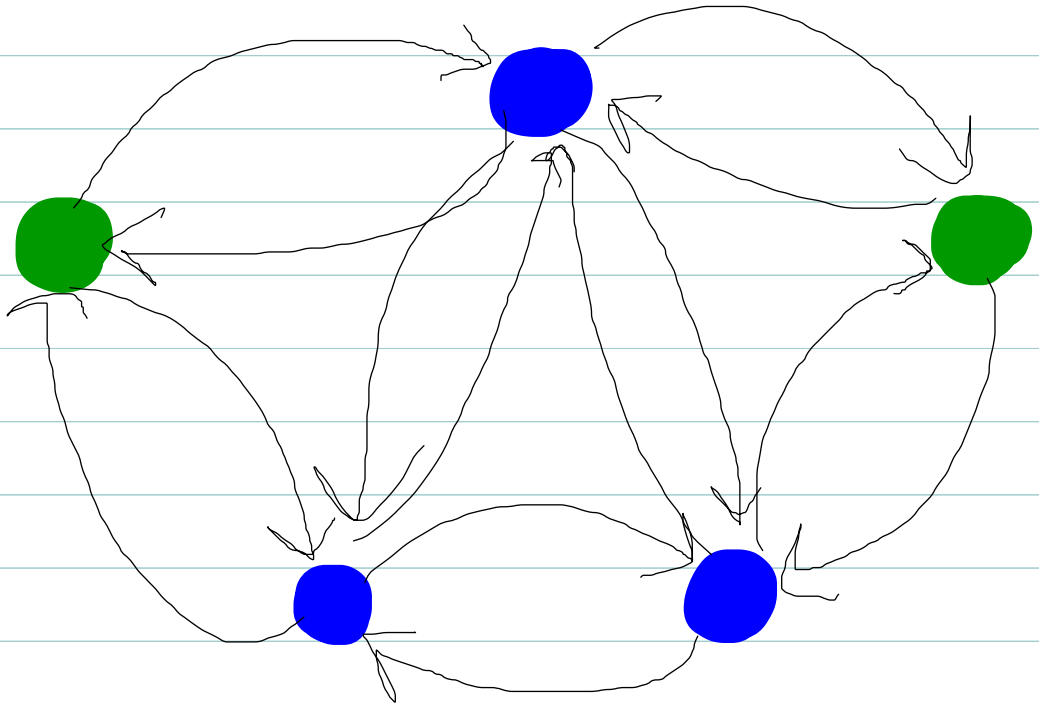
Goal: Find a maximum flow



- satisfy capacity constraints
- satisfy flow conservation

Residual Network:

$$c'(u,v) = c(u,v) - f(u,v) + f(v,u)$$



→ constructed w.r.t. some flow f

→ Residual network <u>is</u> a flow network

Exercise ↓

<u>Lemma</u> Let G be a flow network
f be a feasible flow
R be residual network wrt f
f' be max flow in R
Then $(f + f')$ is max flow in G

Proof

1) $f + f'$ is feasible in G

By definition of $c'(u,v)$

(4 cases: ...)

2) If F is feasible flow in G, then $F - f$ is feasible in R.

By definition of $c'(u,v)$

3) Let M be max flow in G.

Let $M' = M - f$, M' feasible in R.

If M' is not max flow in R, then $M'' > M'$.

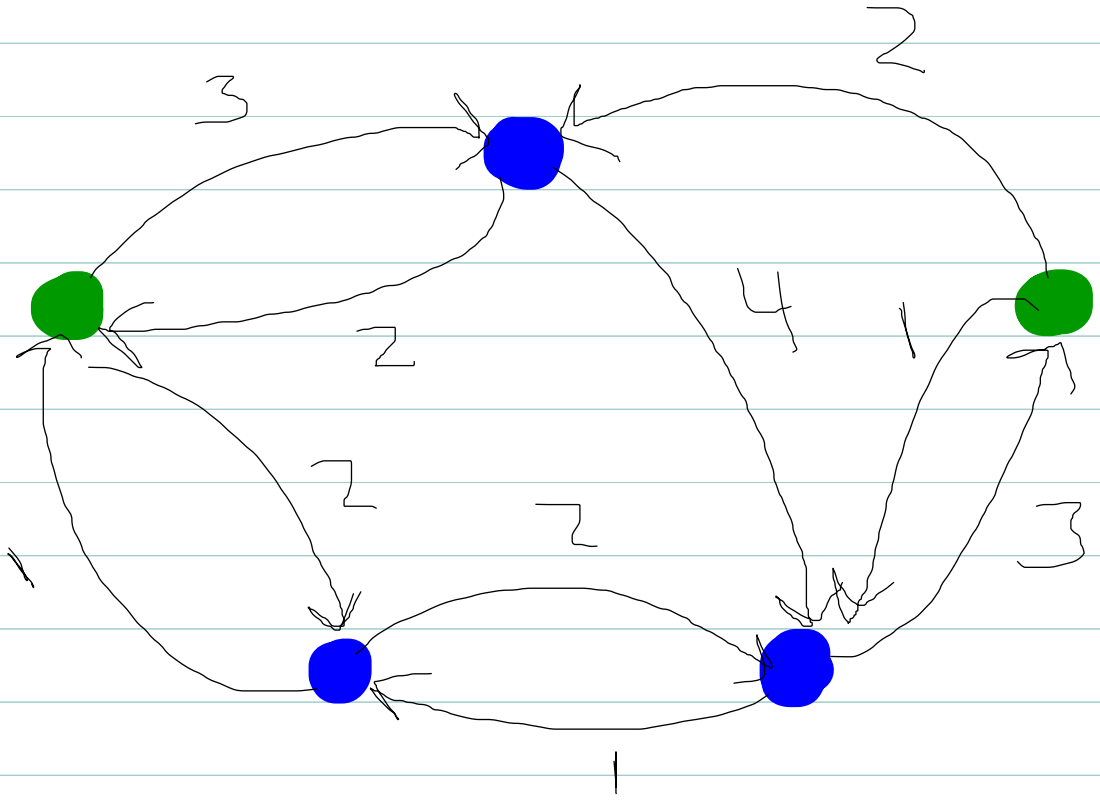
But $f + M''$ is feasible in G and $f + M'' > M$.

Contradiction.

Ford-Fulkerson Algorithm

1) Set $f = 0$
2) Construct residual graph R
3) While $\exists$ path from S to d in R:
    a) find augmenting path P in R
    b) let v be value of bottleneck edge
    c) Add v to flow on P
    d) Subtract v from R on P

Example



Running time:

Each iteration: $O(m)$ ← DFS in R

Update G, R

How many iterations?

→ assume capacities are integers

→ bottleneck = min capacity edge on P

≥ 1

→ each iteration increases flow by ≥ 1

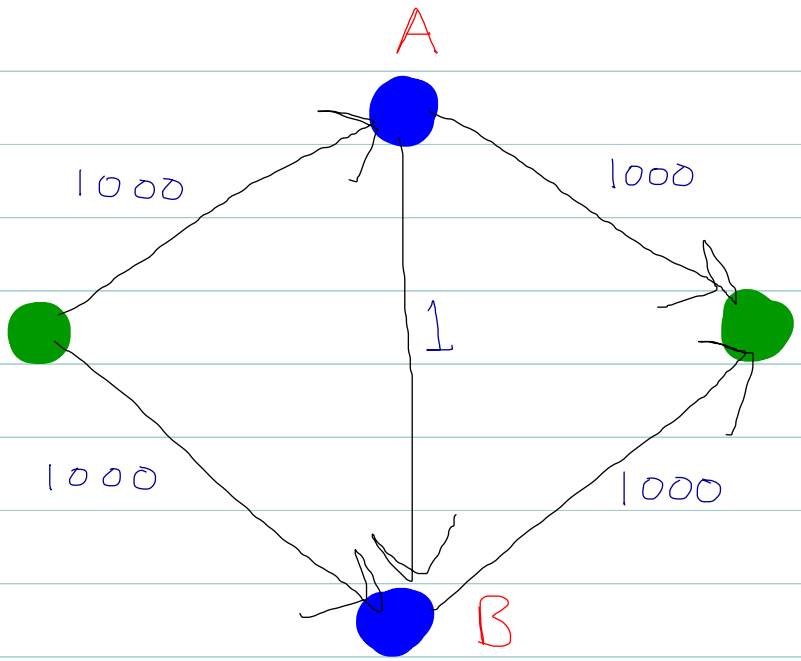
→ Let U = max capacity

→ Max flow $\leq mU$

→ # iterations $\leq mU$

Total cost: $O(m^2 U)$

Is it really so bad? YES



- execute FF using AP through edge (A, B) or (B, A)
- bottleneck = 1
- Repeat 2000 times

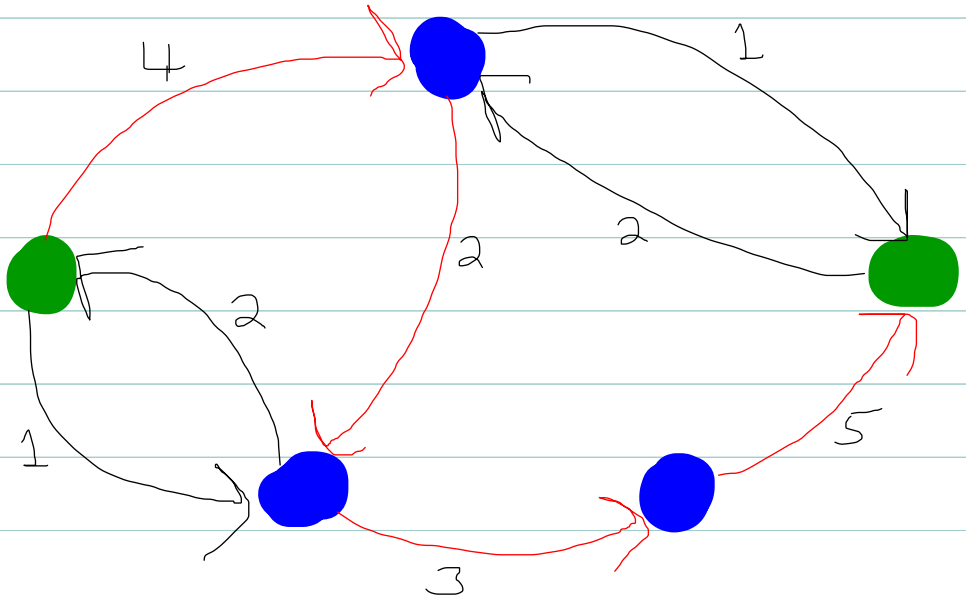
What went wrong?

→ Why did you choose such a bad augmenting path??

Idea: Choose the fattest path

↳ max bottleneck capacity

Residual Network



How to find the fattest path?

Dijkstra's Algorithm !!!

→ replace "+" with min

→ replace min with max

Dijkstra($G=(V,E)$, w, s, d)

1) $\forall u \in V$: $d(u) = \infty$
2) $d(s) = 0$
3) Add $(s, d(s))$ to priority queue PQ
4) While (not PQ.empty)
   a) $u \leftarrow$ PQ.extractMin
   b) $\forall z$: $(u,z) \in E$:
      if $d(z) > d(u) + w(u,z)$ then
         $d(z) = d(u) + w(u,z)$
         PQ.decreaseKey(z, $d(z)$)
      if ($z = d$) then return $d(z)$

Fat Path

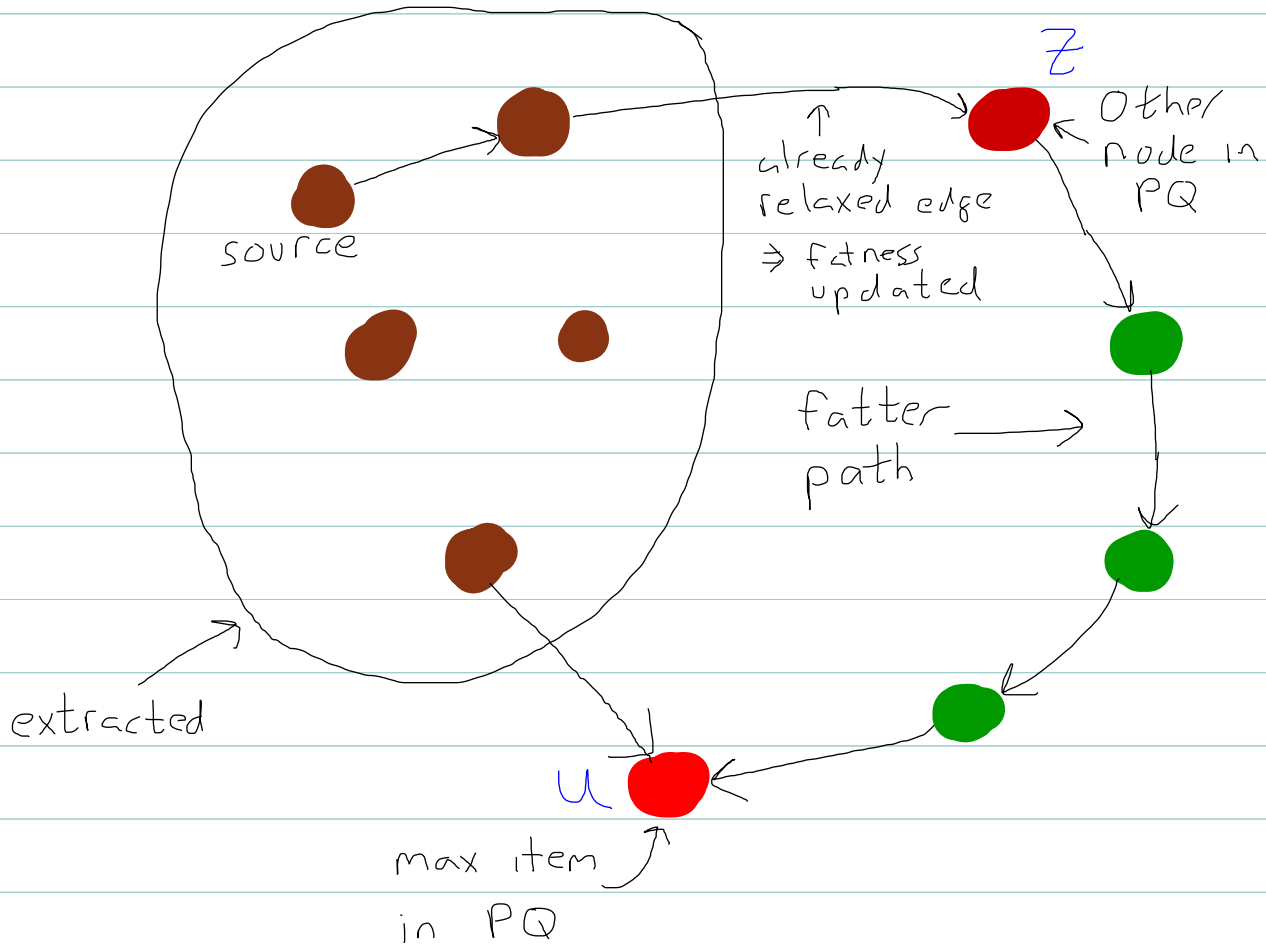
~~Dijkstra~~($G=(V,E)$, w (capacity), s, d)

1) $\forall u \in V$: $d(u) = ~~\infty~~ 0$
2) $d(s) = ~~\infty~~ \infty$
3) Add $(s, d(s))$ to (max) priority queue PQ
4) While (not PQ.empty)
   a) $u \leftarrow$ PQ.extract~~Min~~ Max
   b) $\forall z$: $(u,z) \in E$: $\leq \min[d(u), w(u,z)]$
      if $d(z) > ~~d(u) + w(u,z)$ then
         $d(z) = ~~d(u) + w(u,z)~~ \min[d(u), w(u,z)]$
         PQ.~~decrease~~increaseKey(z, $d(z)$)
      if ($z = d$) then return $d(z)$

Proof

When a node is extracted, we discover its maximum fatness.

Assume not: Let u be first where not true



→ every edge on path $s \to z$ has capacity $> d(u)$ ← fatter path

→ $d(z) > d(u) \Rightarrow$ extract z not u

Running Time:

→ Every edge relaxed once
  ↳ $O(\log n)$ per increaseKey
→ Every node extracted once
  ↳ $O(\log n)$ per extract
→ Other overhead → $O(m)$

⇒ $O([m+n]\log n)$
$= O(m \log n)$

Analysis of "Fattest Path" heuristic

How many iterations?

Flow Decomposition Lemma

Given flow network G, flow f:
$\exists$ flows $f_1, f_2, \ldots, f_K$ where:

1) $f = \sum f_j$
2) $K \leq m$ ← # edges in network
   ↑ # of flows
3) Each f_j is st-path or cycle

Define addition of flows:

$$g(u,v) = \max\left[\begin{array}{l} f_1(u,v) - f_1(v,u) \\ + f_2(u,v) - f_2(v,u) \end{array}, 0 \right]$$

$\Rightarrow g = f_1 + f_2$

[recall: either $f(u,v)=0$ or $f(v,u)=0$]

Corollary There exists a path from $s \to t$ with bottleneck $\geq OPT/m$

OPT = max flow in G

Why? Let F be max flow

Let $f_1, \ldots, f_k$ be decomposition

At least one f_j has average flow:

$$|f_j| \geq \frac{F}{k} \geq \frac{F}{m}$$

where f_j is a path (ignore cycles)

$\Rightarrow$ capacity on path f_j is $\geq F/m$

Proof of Flow Decomposition Lemma (constructive)

Repeat until $f = 0$:

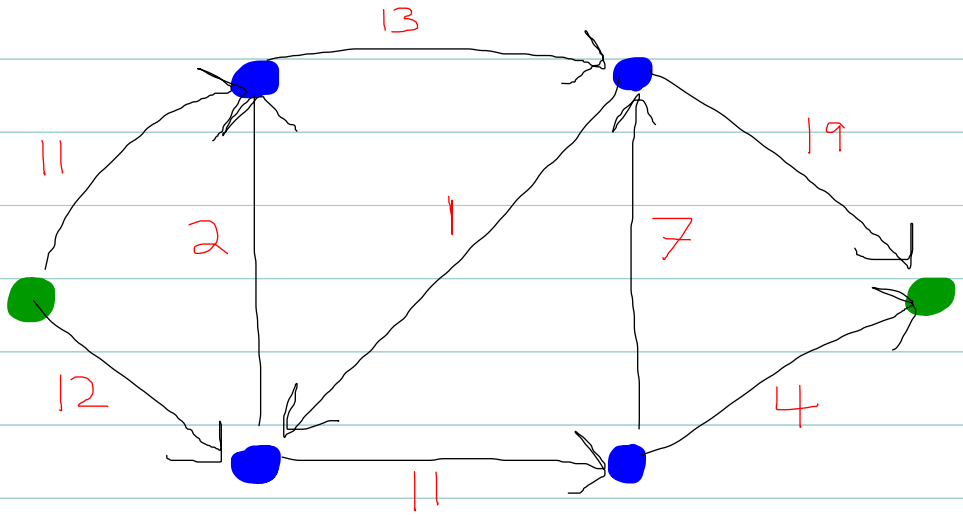
anti-augment

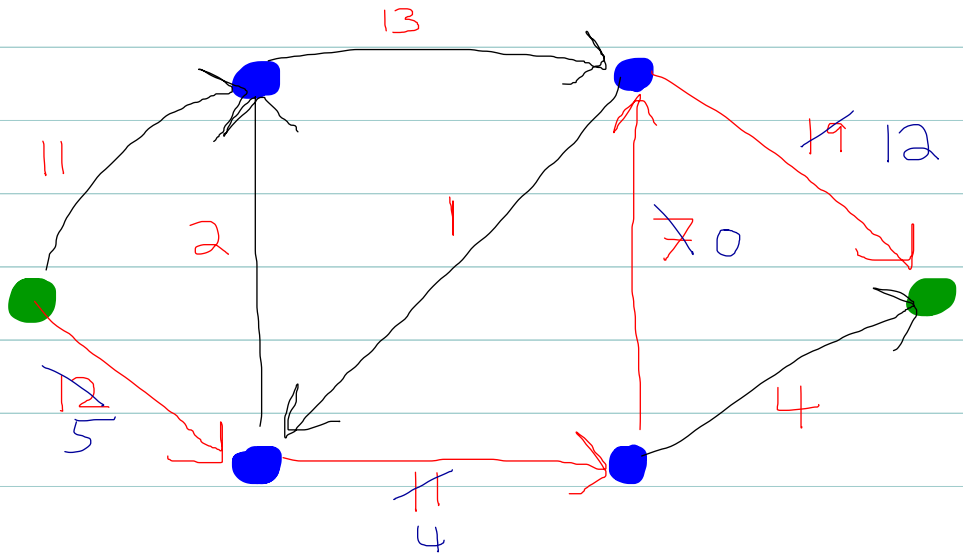
1) Find a path P from $s \to d$ with non-zero flow
2) Let v be min flow on P
3) Create new flow f_j on P with value v
4) Subtract f_j from f.
5) $j = j + 1$

Repeat until all edges have no flow:

1) Find a cycle C with non-zero flow
2) Let v be min flow on C
3) Create new flow f_j on C with value v
4) Subtract f_j from f.
5) $j = j + 1$

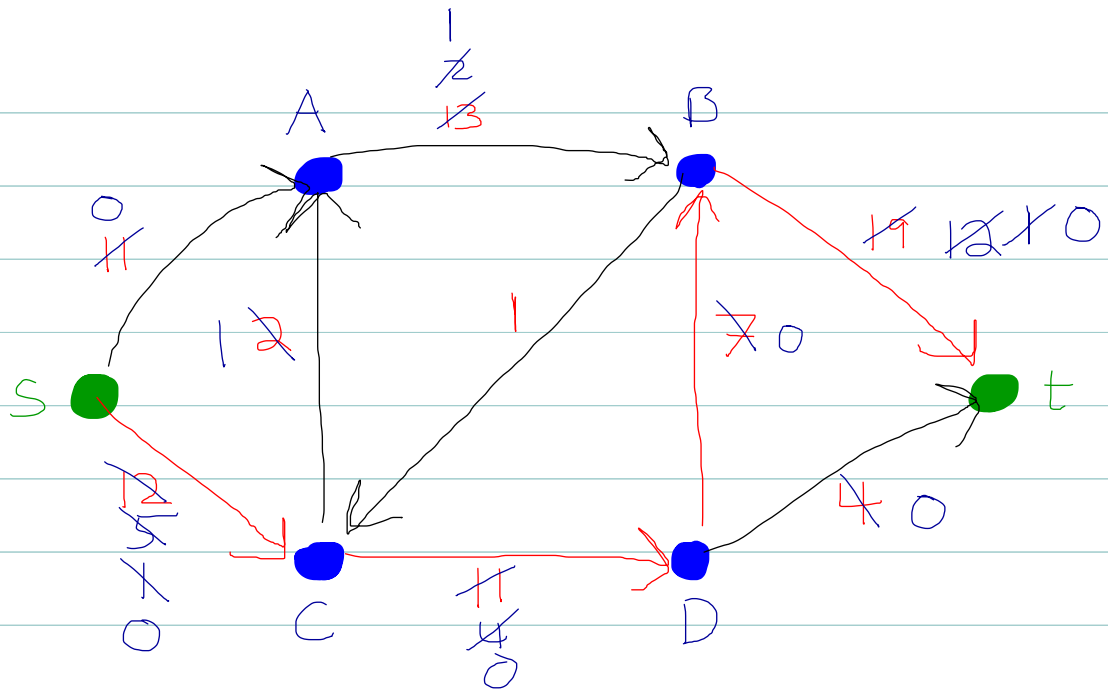
$\Rightarrow$ Flows sum to f by construction





Every iteration: ≥1 edge is reduced to 0

⇒ ≤m flows constructed



Flows:

$S \to C \to D \to B \to t$	:	7
$S \to A \to B \to t$	:	11
$S \to C \to D \to t$	:	4
$S \to C \to A \to B \to t$	:	1
$A \to B \to C \to A$	:	1

23 = flow

Analysis of "Fattest Path" heuristic

How many iterations?

Graph G

flow f Let F = max flow

Residual R

1) First iteration → guaranteed to find a path with bottleneck $\geq \frac{F}{m}$

Before augmenting:

max flow in $R = F$

After augmenting:

max flow in $R = F - F/m = F(1-1/m)$

2) Second iteration → guaranteed to find a path in R with bottleneck $\geq \frac{F(1-1/m)}{2m}$

↖ Why 2? $2m$ edges in R

After augmenting:

max flow in $R = F(1-1/m) - \frac{F(1-\frac{1}{m})}{2m}$

$\leq F(1-\frac{1}{m})(1-\frac{1}{2m}) \leq F(1-\frac{1}{2m})^2$

K) After K iterations:

max flow in $R \leq F\left(1-\frac{1}{2m}\right)^K$

Choose $K = 2m \ln(F)$

Recall: $\left(1-\frac{1}{x}\right)^x \leq 1/e$

$e^x = 1 + x + x^2/2 + \cdots$

max flow in $R \leq F\left(1-\frac{1}{2m}\right)^{2m \ln F}$

$\leq F e^{-\ln F}$

$\leq F F^{-1}$

≤ 1

$\Rightarrow$ After $2m \ln F + 1$ iterations, no augmenting path

$\Rightarrow$ done

Conclusion: $O(m \ln F)$ iterations

Each iteration: $O(m \log n)$ ← Dijkstra's Fat Path

Total time: $O(m^2 \log n \log F)$

Another Approach: Scaling

Problem: FF takes $O(m^2U)$ time

↑ max capacity

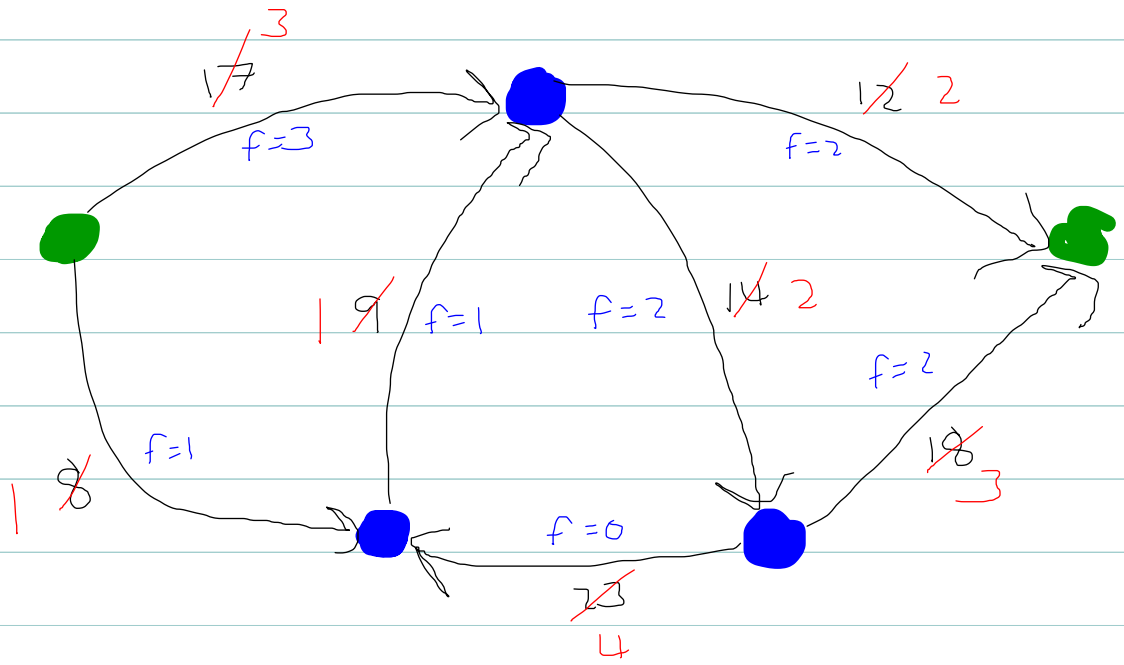
Idea: Scale down capacities

$$c'(e) = \left\lfloor \frac{c(e)}{K} \right\rfloor$$

← still an integer

for some value K

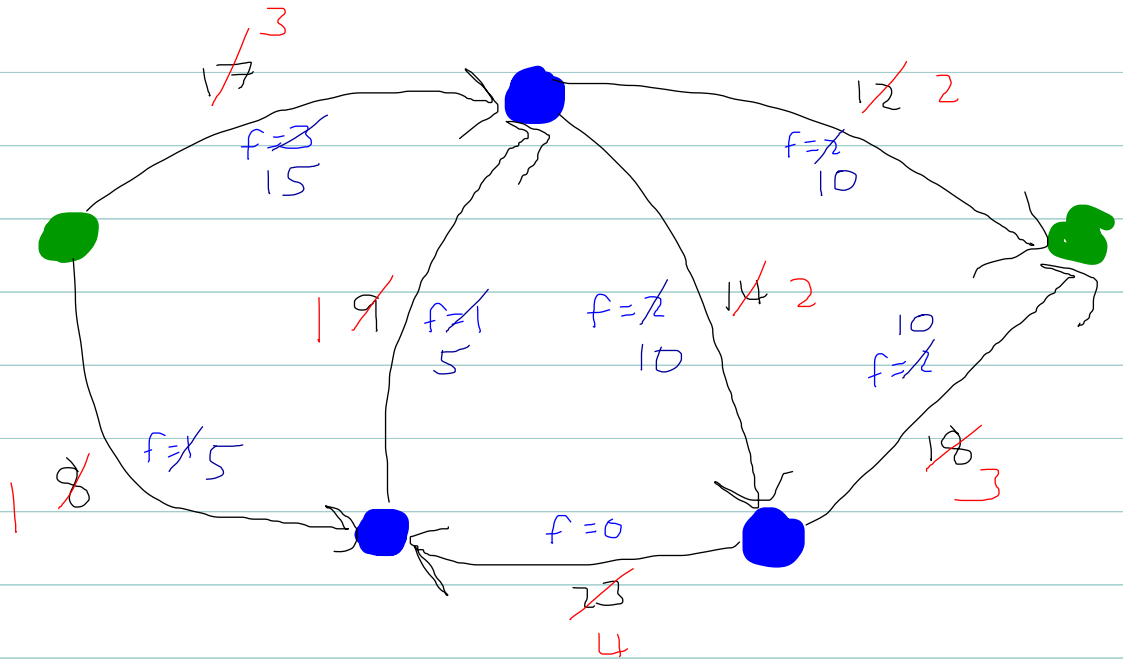
$$c(e) = K\,c'(e) + X \quad \text{for } X \le K$$



Ex → $K=5$

$F=4$

Then: Multiply flow by $K=5$



→ KF is feasible in original graph

$Kc'(e) \leq c'(e)$

→ Is KF max flow? No

→ Run FF until done

CS-Alg

1) $c' = c/K$
2) $f' = FF(G, c')$
3) $f = Kf$
4) Keep running FF on G, c

How long does Step 4 take?

For edge e: $c(e) = K c'(e) + X$

1) Multiply both c', f' by K

→ still max flow

edges with 0 capacity in R

still have 0 capacity $r(u,v) = c(u,v) - f(u,v) + f(v,u)$

only 1 is $\neq 0$

2) Add X to $c'(e)$

→ $X \leq K$ added to R

→ After adding all edges, max flow in R is $\leq mK$

(think about min cut in R)

Conclusion: need mK iterations, each $O(m)$

$\Rightarrow O(m^2 K)$

Total time: $O(m^2 U/K + m^2 K)$

↑ Scaled FF, U = max capacity

↳ scale up FF

$= O(m^2 \sqrt{U})$

How to do better → Iterate!!!

Edge e: $c(e) = 11 = 1011$ in binary

$c_1 = 1$ ×2
$c_2 = 2 = 10$ ×2+1
$c_3 = 5 = 101$ ×2+1
$c_4 = 11 = 1011$

CS-FF($G=(V,E)$, s, t, C)

1) $f = 0$
2) for $K = \log(U)$ down to 1 → always feasible
   a) $f = 2f$ ← double flow on each edge after every iter.
   b) $c' = \lfloor c/2^K \rfloor$
   c) Run FF on G, c'
      Augment f until no path in R
3) Return f

At every iteration:

1) $f = 2f$
2) $c' = 2c' + X$ where $X \leq 1$

Claims:

1) New flow is always feasible
2) max flow in R after $\leq m$

Total time for 1 iteration:

$O(m^2)$

Total time:

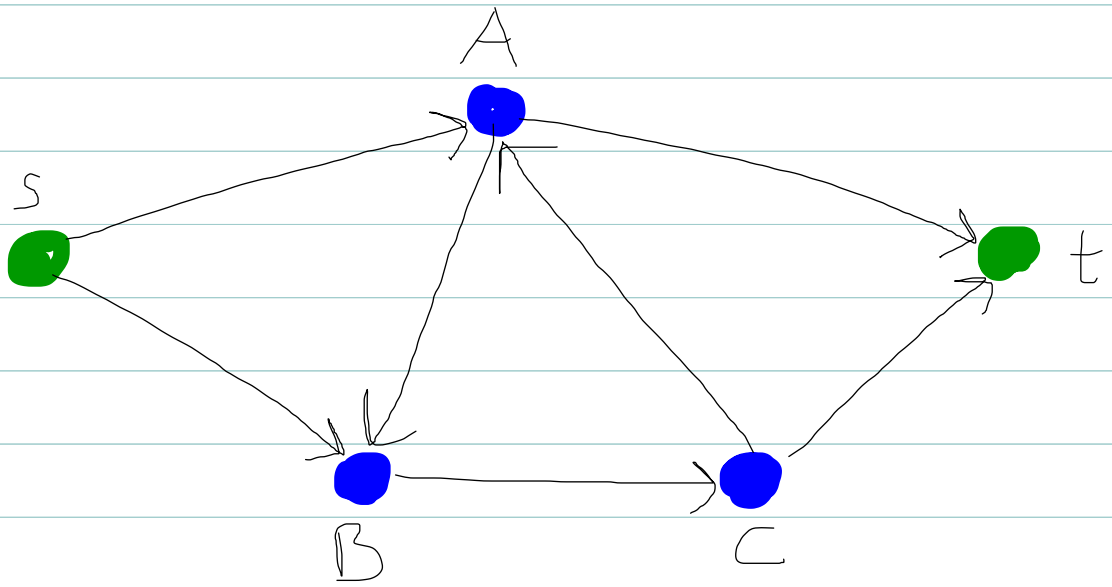
$O(m^2 \log U)$

Another idea: Edmonds-Karp

→ Pretend edges in R have weight 1

→ Find shortest augmenting path

↳ fewest edges



shortest path = s → A → t

↳ Use BFS

↳ O(E) per iteration

Claims

1) Distance from source never decreases
   - ↳ augmenting paths push s and t apart in R

2) Every m+1 iterations, $d(s,t)$ increases
   - ↑ # of edges on shortest path

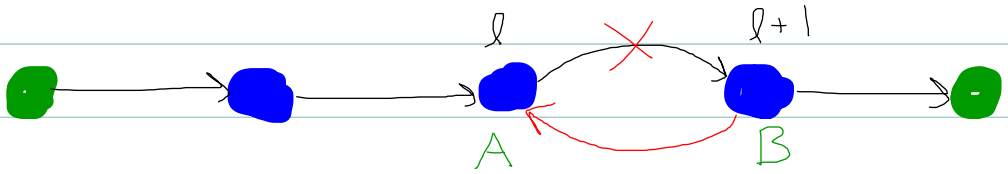
↳ $d(s,t) \leq n \Rightarrow \leq (m+1)n$ iterations

↳ Total time: $O(m^2 n)$

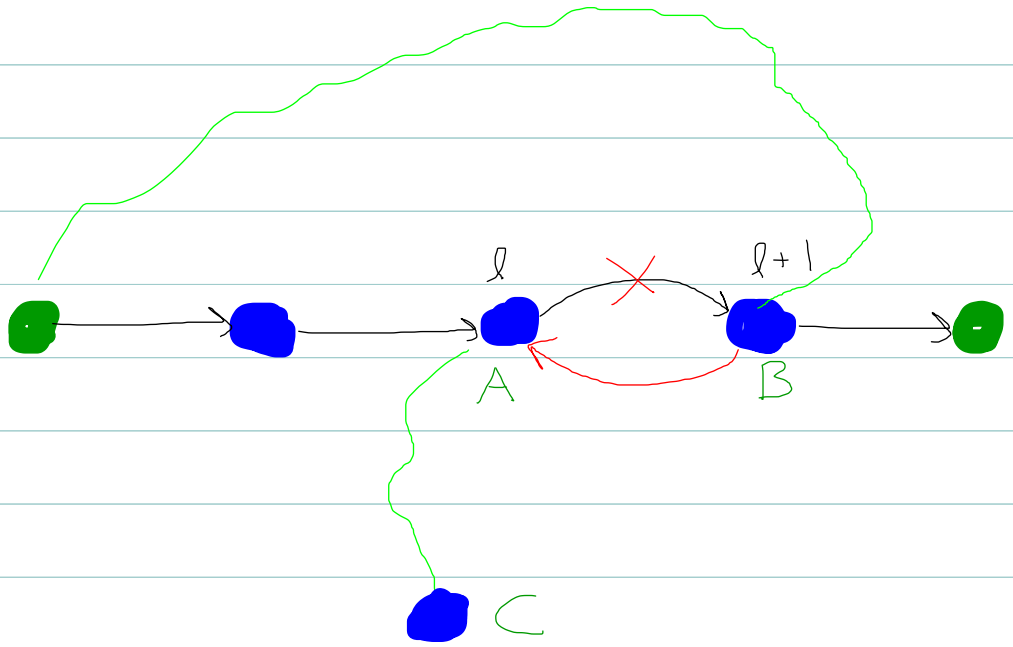
1) Distance from source never decreases

→ On augment: a) delete edges in R
↳ no problem
cannot shorten path
b) add edges to R

→ Only add reverse edges on shortest s→t path in R



→ every step on shortest path increases distance
→ shorter path must cross new edge
→ d(s, B) cannot get shorter
→ so d(s, A) over new edge ≥ l+2 > l
→ so shortest path to A does not cross new edge → d(s, A) does not decrease
→ No shorter path across new edge exists



Shortest path from $s \to C$ cannot cross edge (B, A)

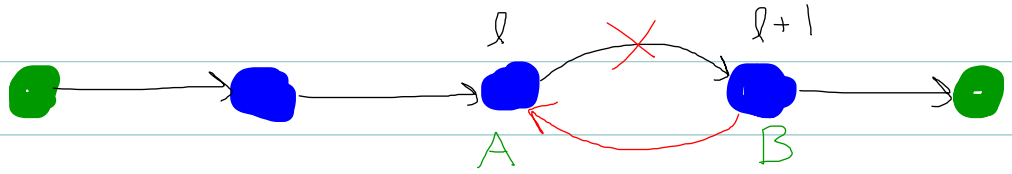
↳ path from $s \to C$ did not get shorter when (B, A) was added

Note: inductive argument

add one reverse edge at a time

After m+1 iterations, $d(s,t)$ increases

pigeonhole principle → m+1 iterations → some edge
m edges reversed twice



In every iteration, ≥1 edge reversed ← bottle neck edge

Assume $e=(A,B)$ reversed twice

1) $s \rightsquigarrow A \to B \rightsquigarrow t$
2) $s \rightsquigarrow B \to A \rightsquigarrow t$

⇐ shortest paths

At 1) $d(s,A) < d(s,B)$
At 2) $d(s,B) < d(s,A)$

⇒ Either, between (1) and (2):
i) $d(s,A)$ increased
~~ii) $d(s,B)$ decreased~~

⇒ Since $d(A,t)$ does not decrease,
$d(s,t)$ has increased

be bottleneck

Conclusion: Edge $e = (A, B)$ can flip at most $2n$ times

At most $2m$ edges in R

At most $O(mn)$ bottleneck edges

↳ At most $O(mn)$ iterations