Network Flow

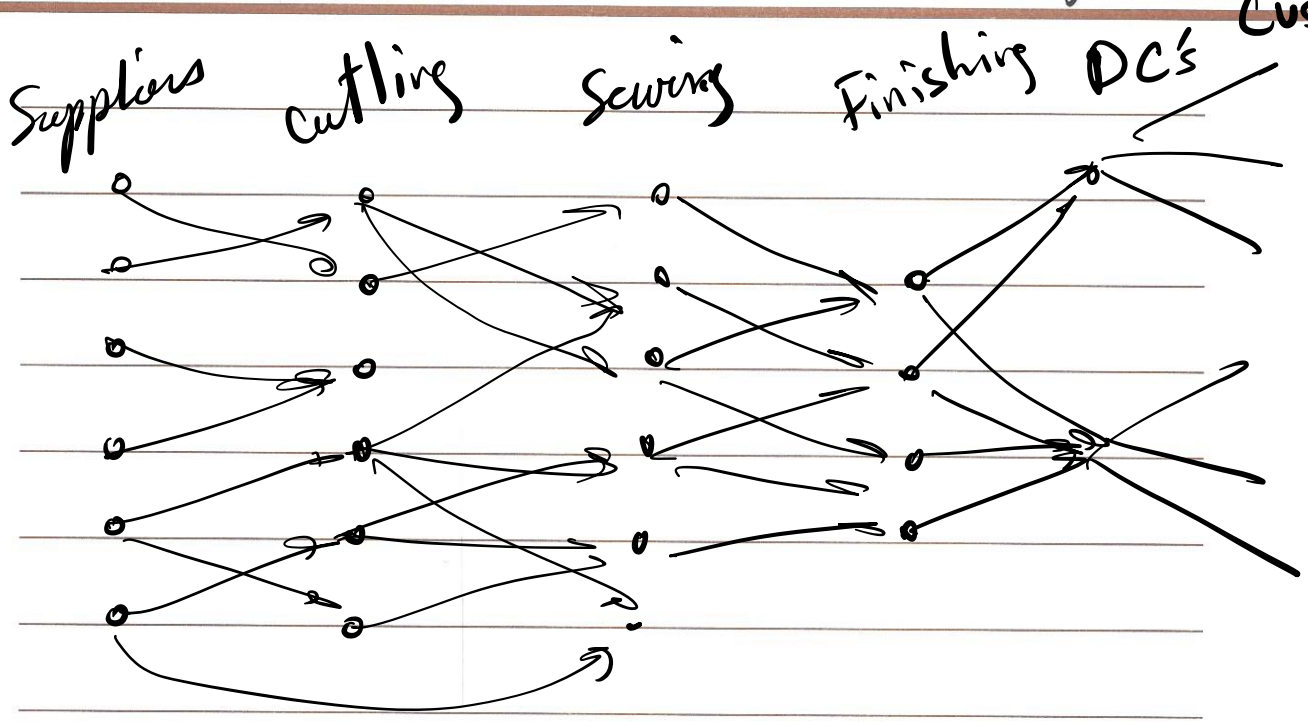
Thursday class this week
pre-recorded 9-12 in the morning
@ OHE 100D

Examples of network flow problems:

- How much data can we send from one point in the network to another point?

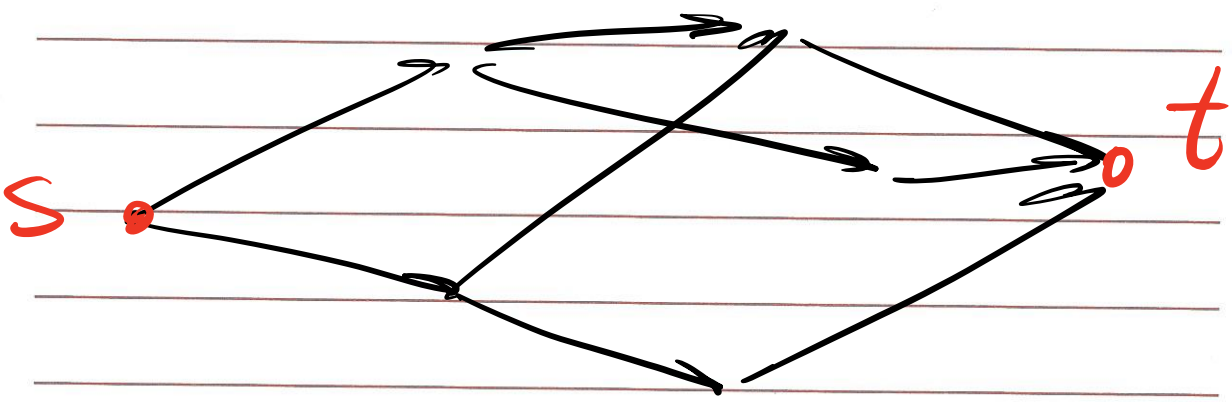
- How much traffic does the freeway system sustain for travel between two cities?

- How profitable could a manufacturing supply chain be?



Def. A flow network is a directed graph $G=(V,E)$ with following features:

- Each edge e has a non-negative capacity c_e
- Has a single source node $s \in V$
- Has a single sink node $t \in V$



Assumptions:

- no edges enter s or leave t
- at least one edge is connected to each node $O(m+n) \rightarrow O(m)$
- all capacities are integers

Notation

We will call $f(e)$ flow through edge e. $f(e)$ has the following properties:

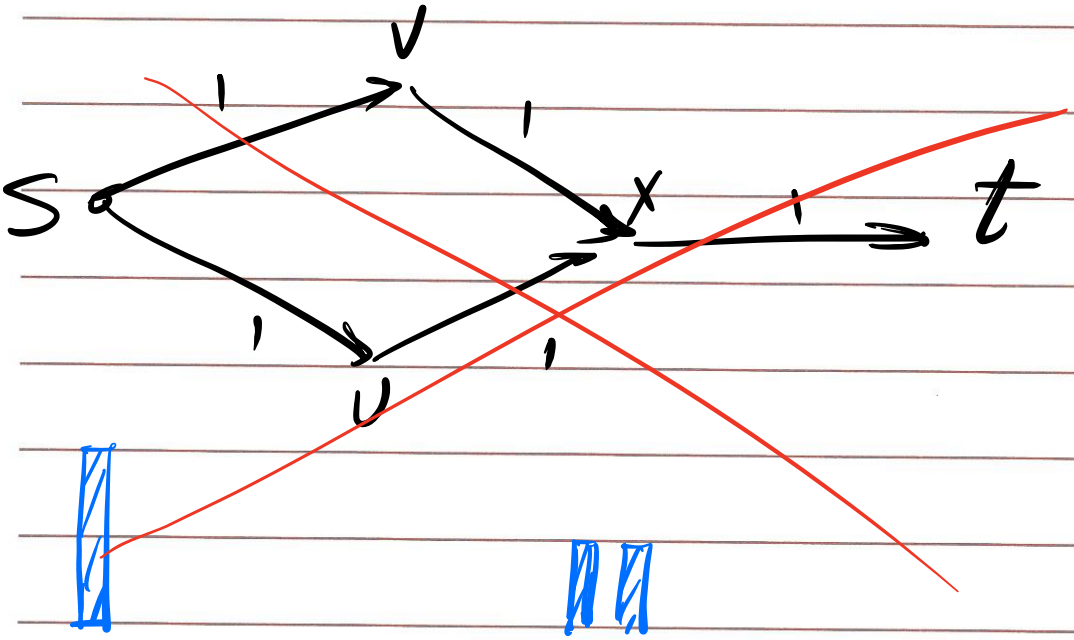
1- Capacity constraint:

for each edge $e \in E$, $0 \leq f(e) \leq c_e$

2- Conservation of flow

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e), \text{ except for } s \& t$$

We are looking for steady state flows.

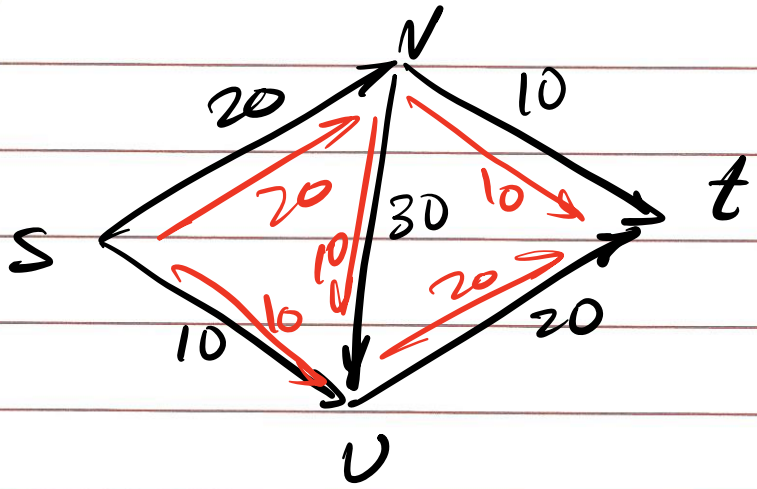


Def. For a steady state flow, the value of flow $v(f)$ is defined as follows:

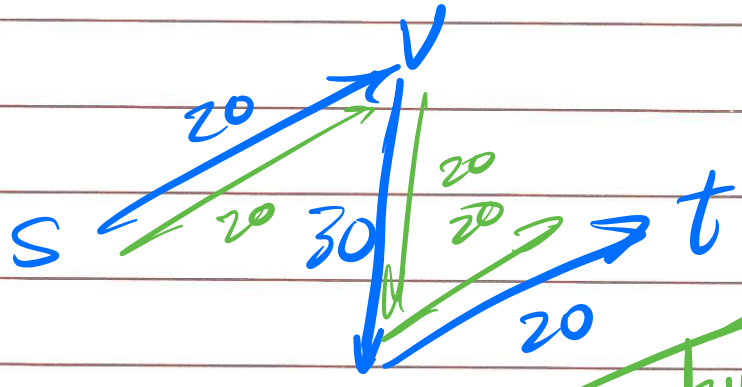
$$v(f) = \sum_{e \text{ out of } s} f(e)$$

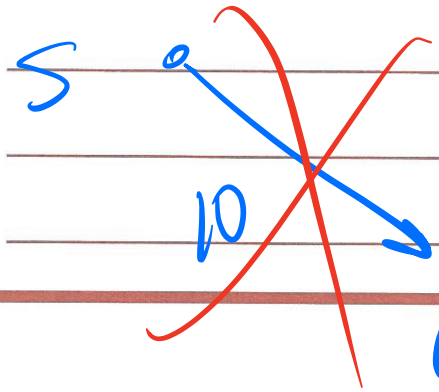
Max Flow problem

Given a flow network G, find an s-t flow with max. value.

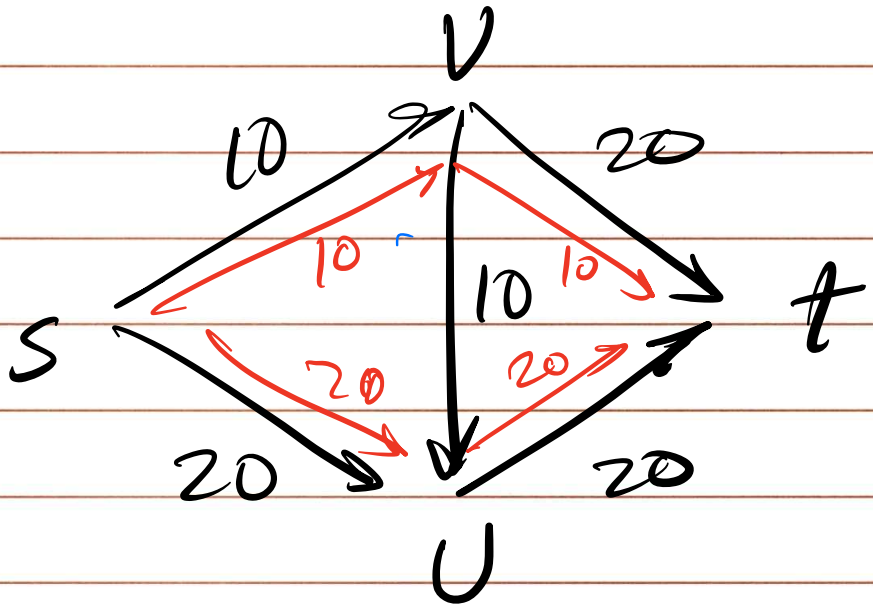


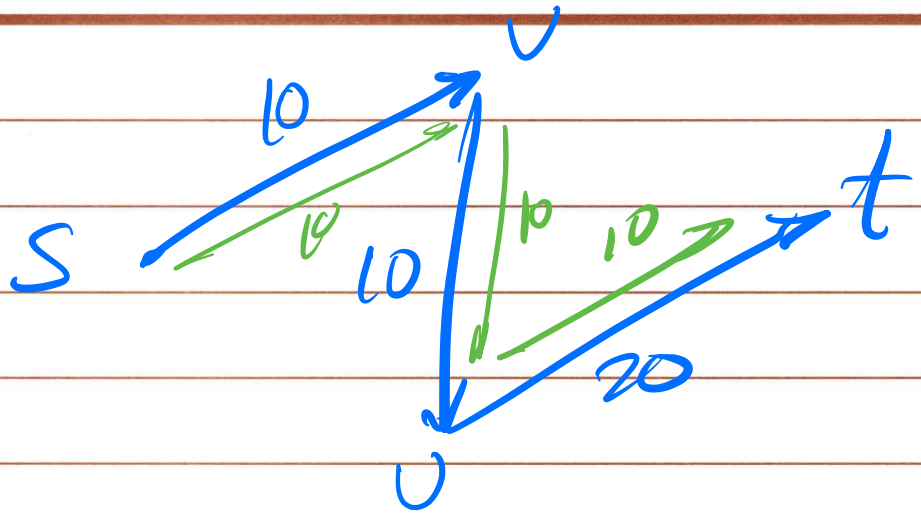
try #1 use highest cap edges first ✗

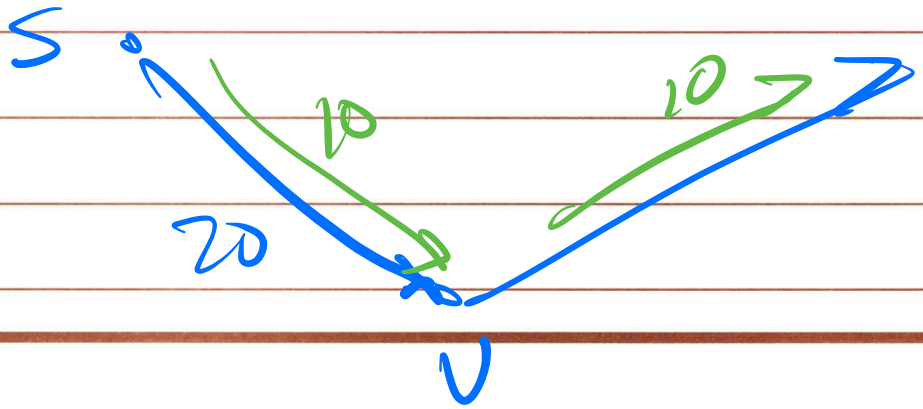




try #2 use lowest cap. edges first ✗



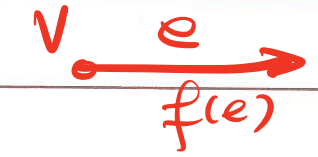




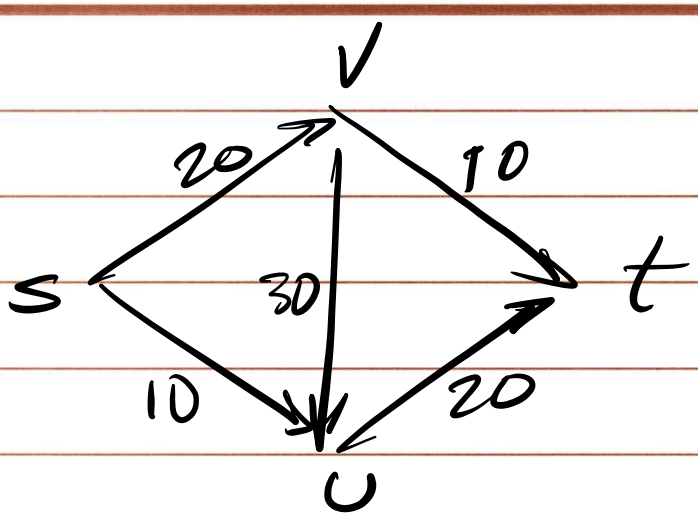
Def. G_f is the residual graph of G with the following definitions:

- G_f has the same set of nodes as G
- for each edge e w/ $f(e) < c_e$, we include e in G_f with capacity $c_e - f(e)$ — forward edge
- for each edge e w/ $f(e) > 0$, we include edge e' (opposite direction to e) in G_f with $f(e)$ units of capacity — backward edge

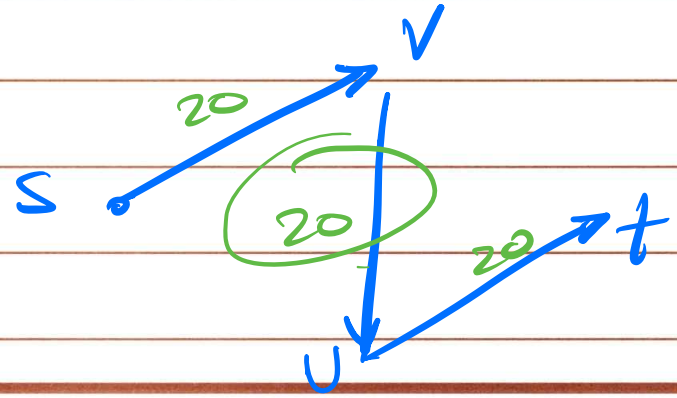
To create G_f:



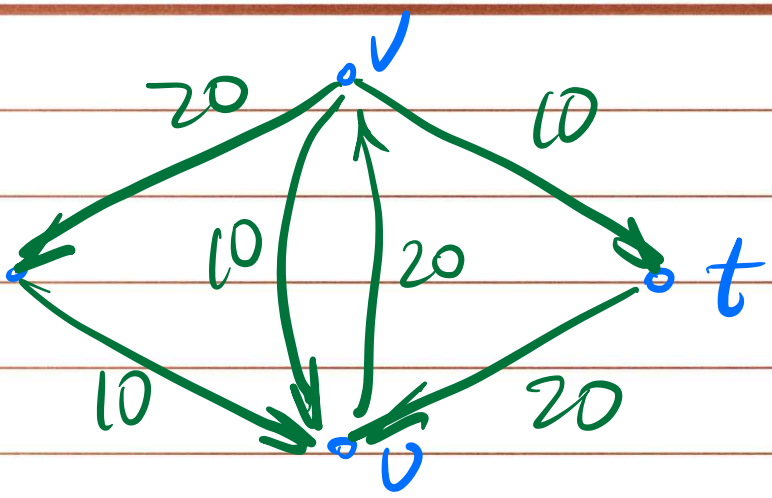
- if $f(e) = 0$
  one forward edge w/ Cap. c_e
- if $f(e) = c_e$
  one backward edge w/ Cap c_e
- if $f(e) < c_e$
  one forward edge w/ Cap $c_e - f(e)$
  one backward " " " $f(e)$


v
20
10
s
30
t
10
20
u

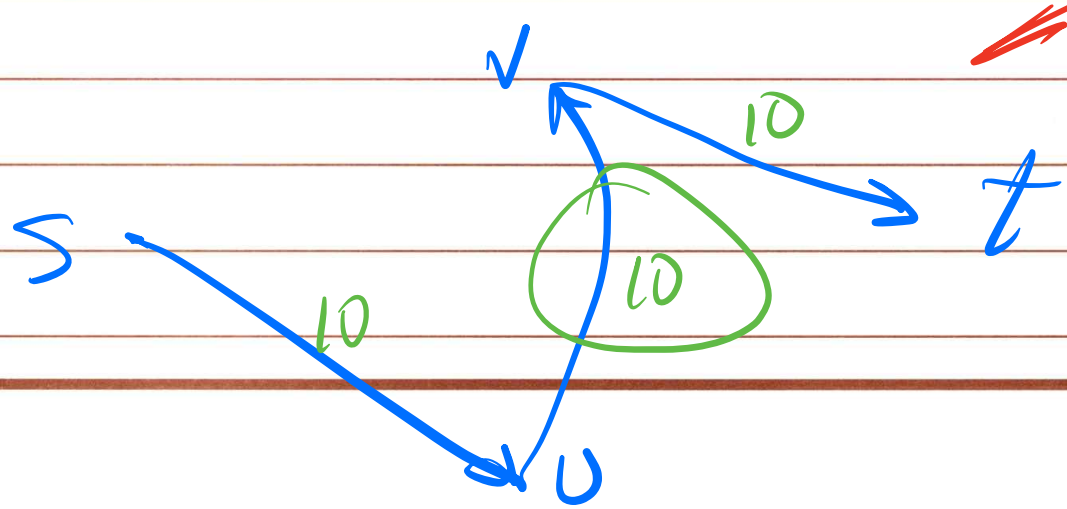
f_1

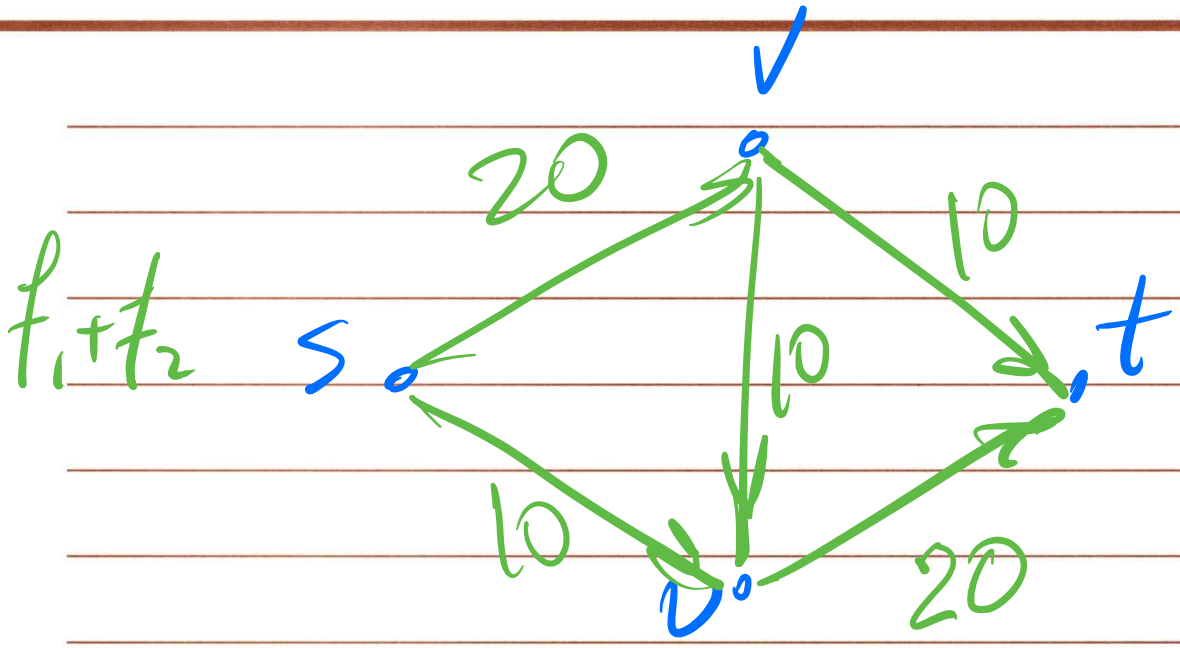

v
20
s
20
20
t
u

G_{f_1}


v
20
10
s
10
20
t
10
20
u

f_2


v
10
t
s
10
10
u


v
20
10
f_1+f_2
s
10
t
10
20
v

Def. If P is a simple path from s to t in G_f, then bottleneck(P) is the minimum residual capacity of any edge on P.

Overall strategy to find Max Flow

- Find a path from s to t
- Find the bottleneck value for this path
- Push flow through this path with value equal to bottleneck value
- Repeat

```
Augment (f, c, P)
    let b = bottleneck (P)
    for each edge (v,u) ∈ P
        if e=(v,u) is a forward edge,
            then increase f(e) by b units
        else (v,u) is a backward edge
            and let e=(u,v)
            then decrease f(e) by b units
        end if
    end for
```

Return (f')

If f is flow before augmentation, and f' is flow after augmentation, we need to show that if f is a valid flow, then f' will also be a valid flow.

Proof: 1- Check capacity condition ✓

Need to show that for each edge $e \in E$, we have $0 \leq f'(e) \leq c_e$

A- If e is a forward edge, ✓

$f(e) + \underbrace{\text{bottleneck}(P)}_{b} \leq c_e - f(e) + f(e)$

$0 \leq f'(e) \leq c_e$

B- If e is a backward edge,

$\text{bottleneck}(P) \leq f(e)$

$\underbrace{f(e) - \text{bottleneck}(P)}_{b} \geq f(e) - f(e)$

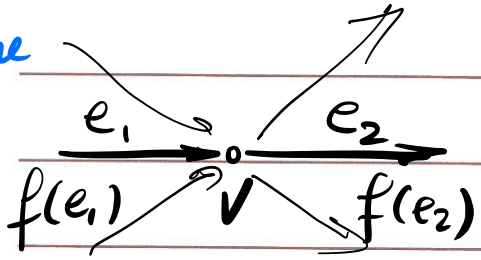
$c_e \geq f'(e) \geq 0$

2- Check conservation of flow ✓

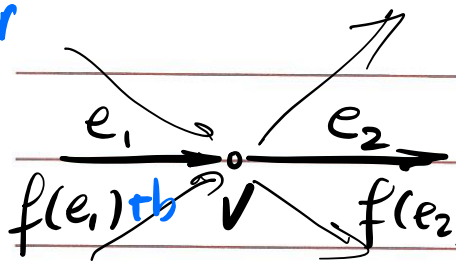
Since f is a valid flow, for each node v, other than s & t we have:

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

before aug.



after aug.



3 more cases to examine!

Case 1 ↓ assuming e_1 & e_2 correspond to forward edges.

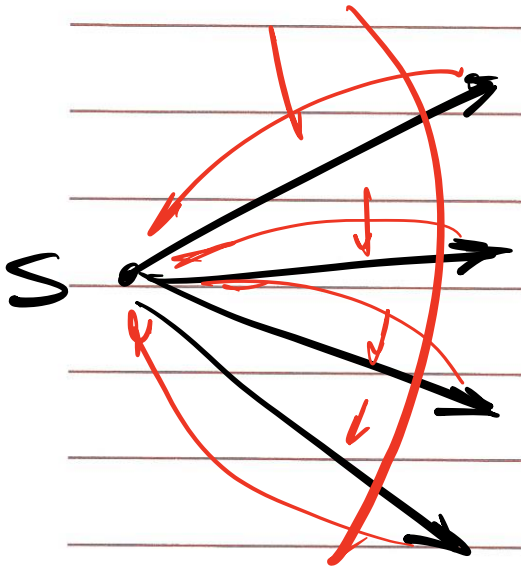
Ford-Fulkerson algorithm for Max Flow.

```
MaxFlow(G, s, t, c)
    Initially f(e) = 0 for all e in G
    While there is an s-t path in Gf
        let P be a simple s-t path in Gf
        f' = augment(f, c, P)
        f = f'
        update Gf
    endwhile
    Return f.
```

Proof of correctness should include:

- Proof of termination
- Proof that f is a Max Flow.

① while loop terminates



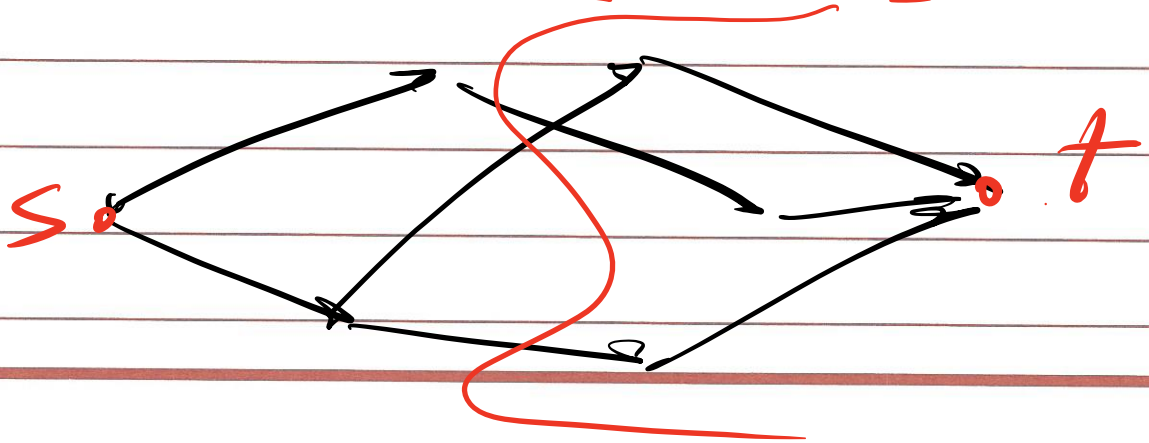
while loop terminates in at most C iterations

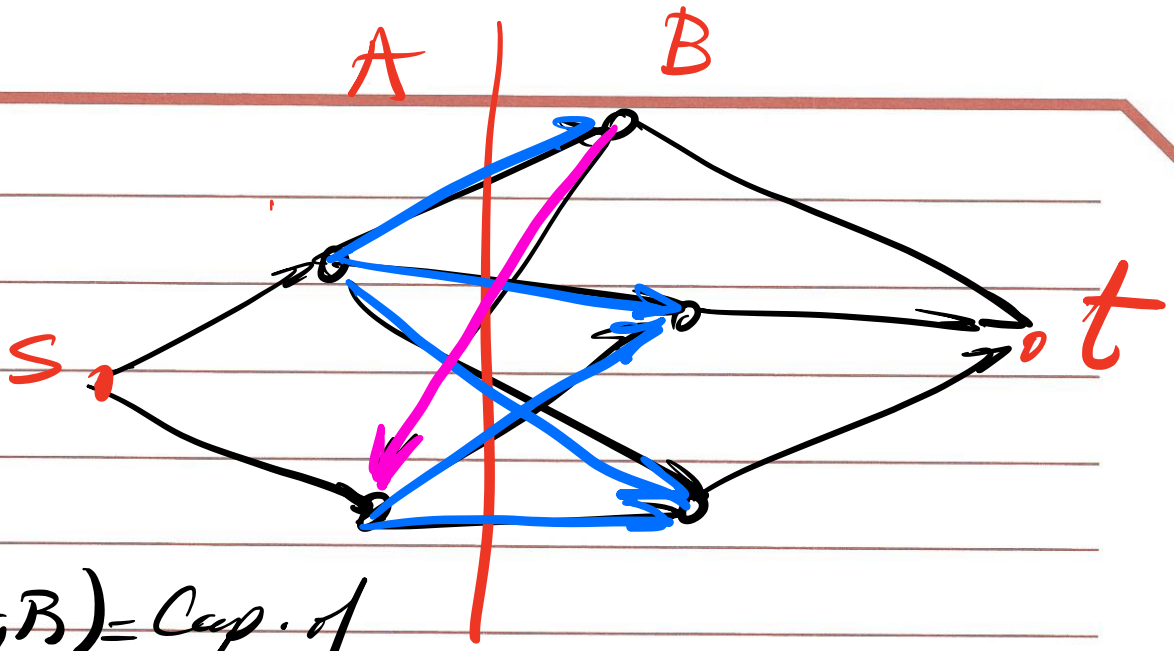
$v(f) \leq \sum_{\text{out of } s} c_e = C$

② f is a Max Flow

We first need some definitions

Define a cut: A cut divides nodes in the graph into 2 sets A & B such that $s \in A$ and $t \in B$



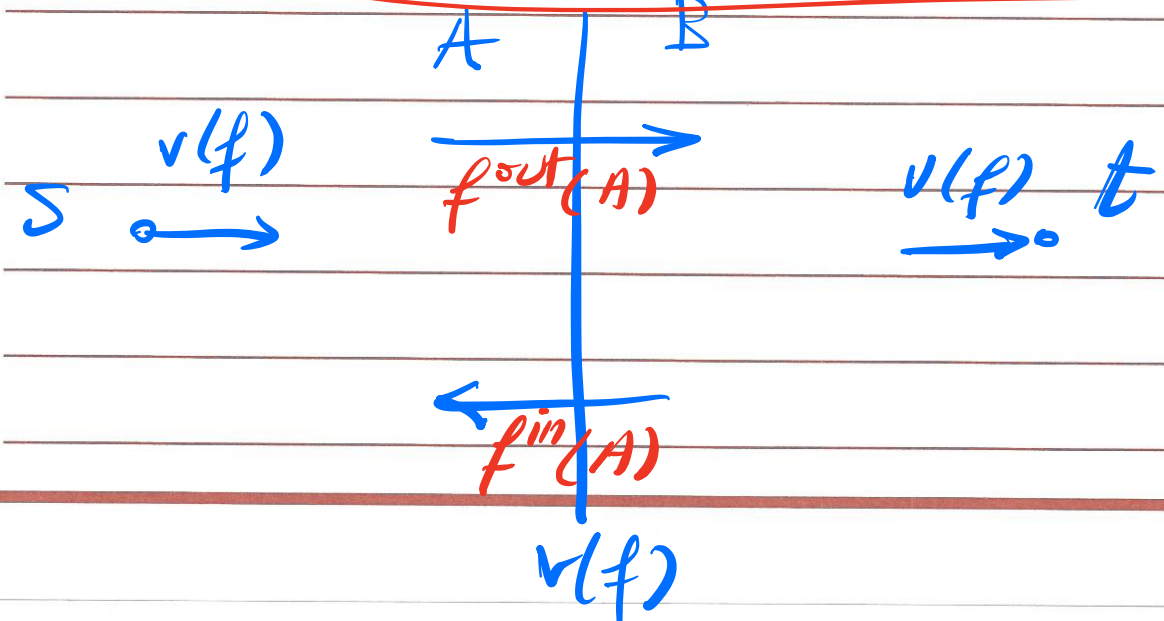


$C(A,B) = \text{Cap. of } cut(A,B)$

$= \sum_{e \text{ out of } A} c_e$

FACT: Let f be any s-t flow and (A,B) any s-t cut, then

$$v(f) = f^{out}(A) - f^{in}(A)$$



Max. value of flow $\leq$ Cap. of the cut (A, B)

Proof:

$$v(f) = f^{out}(A) - f^{in}(A)$$

$$v(f) \leq f^{out}(A) \leq \sum_{e \text{ out of } A} c_e$$

$$v(f) \leq c(A, B)$$

So, the max. flow is bounded by the cap. of every s-t cut.

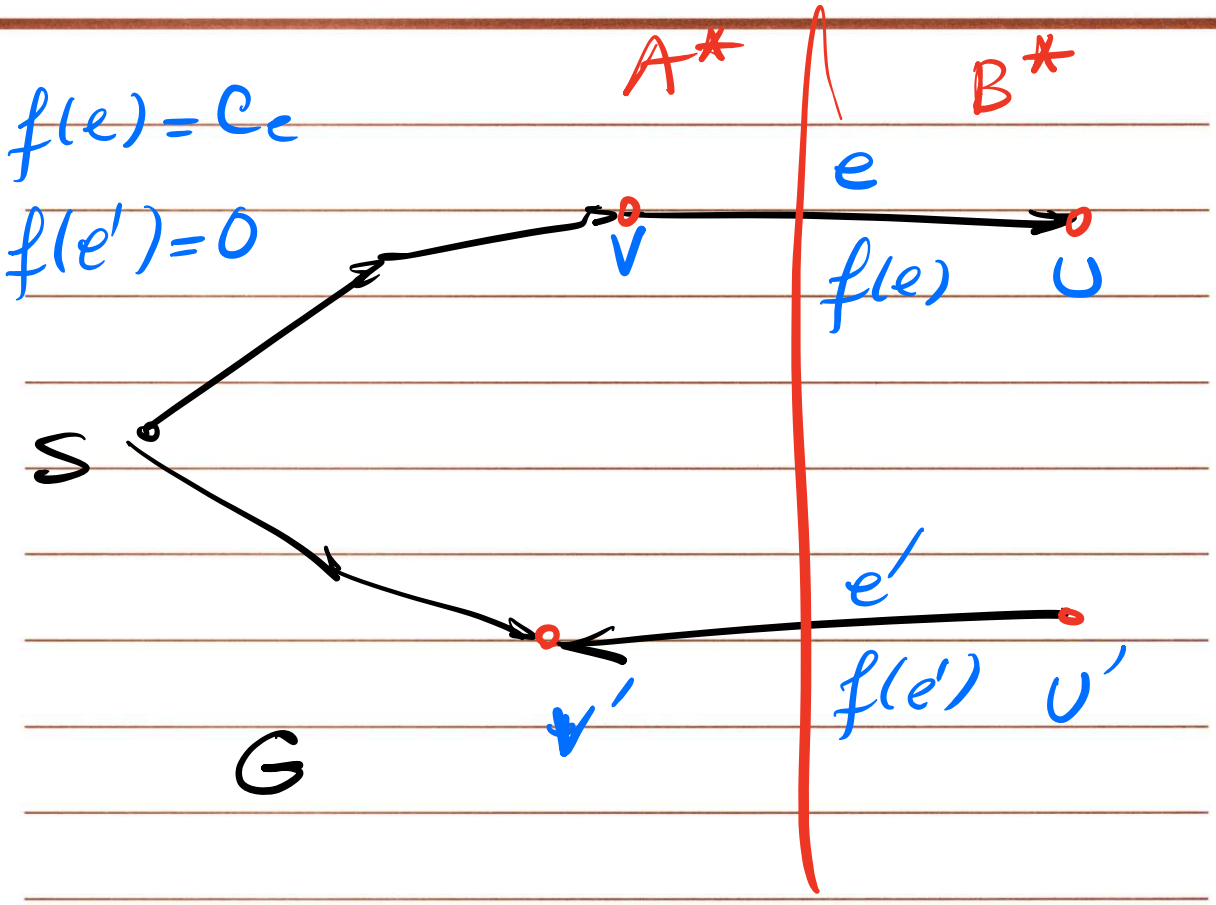
Ford-Fulkerson terminates when the flow f has no s-t paths in G_f.

Claim: If there is no s-t path in G_f, then there is an s-t cut (A^*, B^*) where

$$v(f) = c(A^*, B^*)$$

Proof: Create sets A^* & B^* such that A^* includes all nodes v where there is an s-v path in G_f.

$$B^* = V - A^*$$



$$v(f) = f^{out}(A^*) - f^{in}(A^*)$$

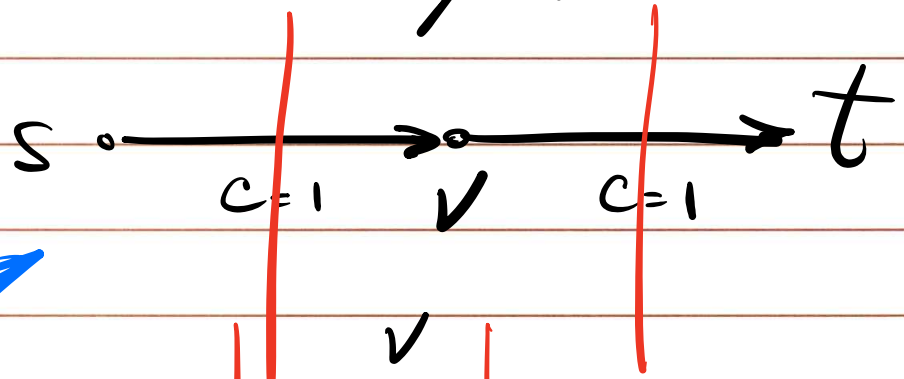
$$= \sum_{e \text{ out of } A^*} c_e - \phi$$

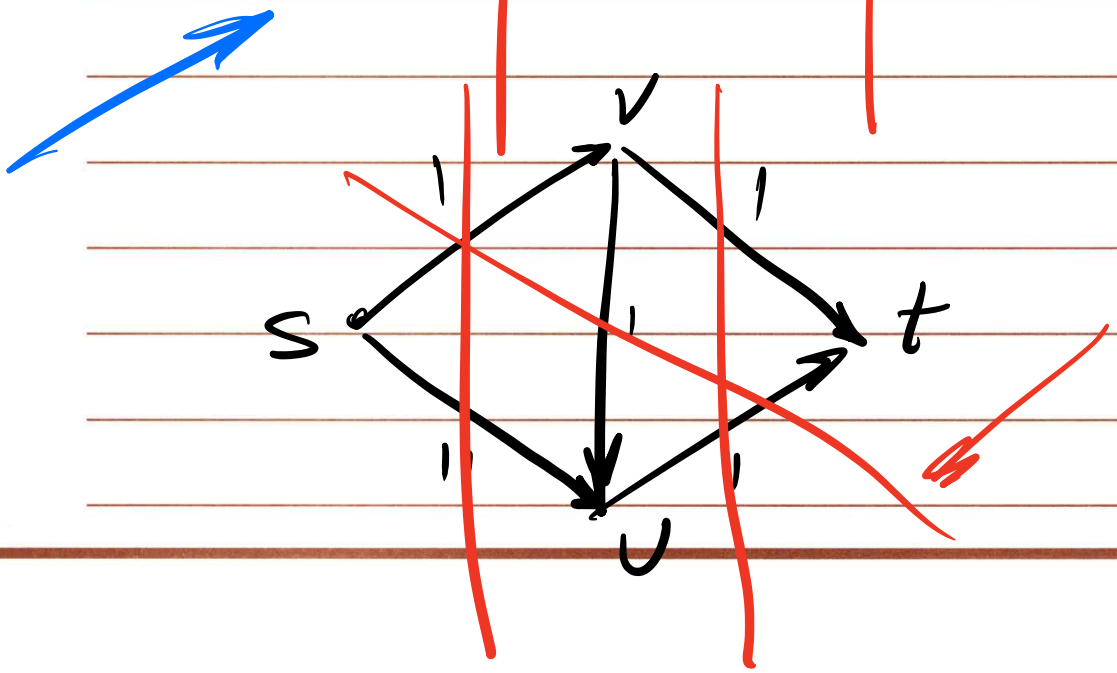
$$v(f) = C(A^*, B^*)$$

$v(f)$ = Max. value of any flow in G

(A^*, B^*) has the min. cap. of any s-t cut in G.

Is min-cut unique? No!





Ford-Fulkerson algorithm for Max Flow.

```
MaxFlow(G, s, t, C)
    Initially f(e) = 0 for all e in G
    While there is an s-t path in Gf
        Let P be a simple s-t path in Gf
        f' = augment(f, c, P)
        f = f'
        update Gf
    endwhile
    Return f.
```

C

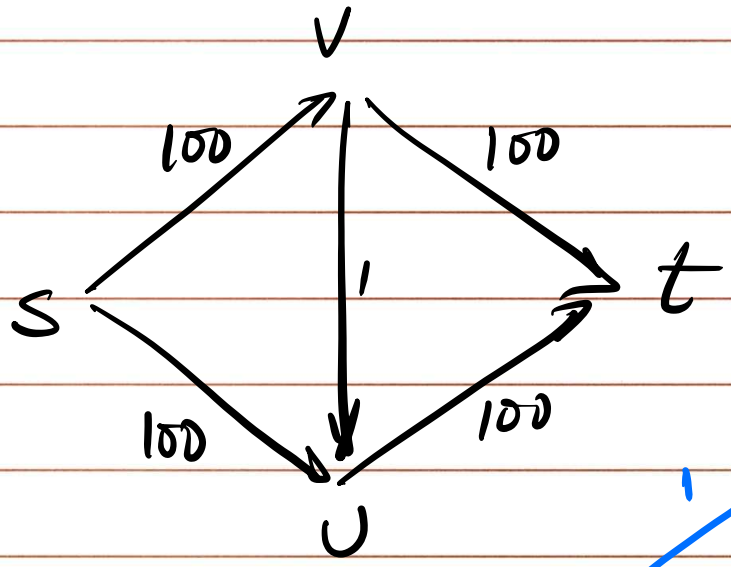
O(m)

O(m)

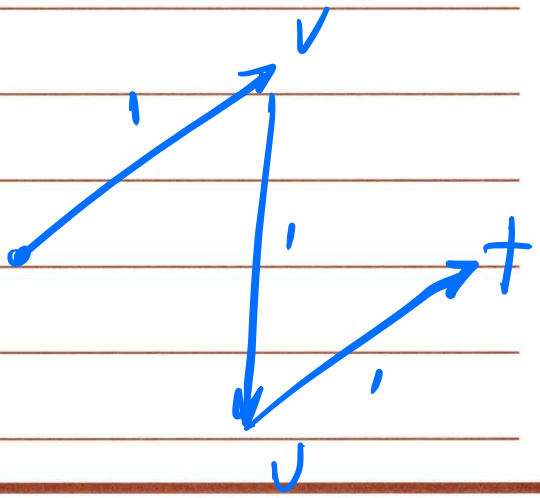
O(m)

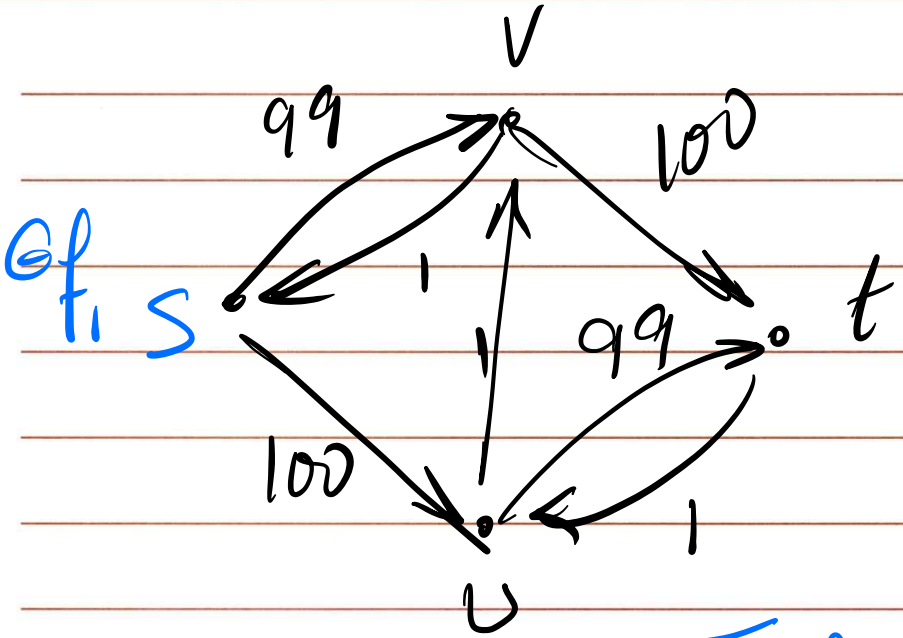
overall complexity = O(Cm)

pseudo-polyn. complexity

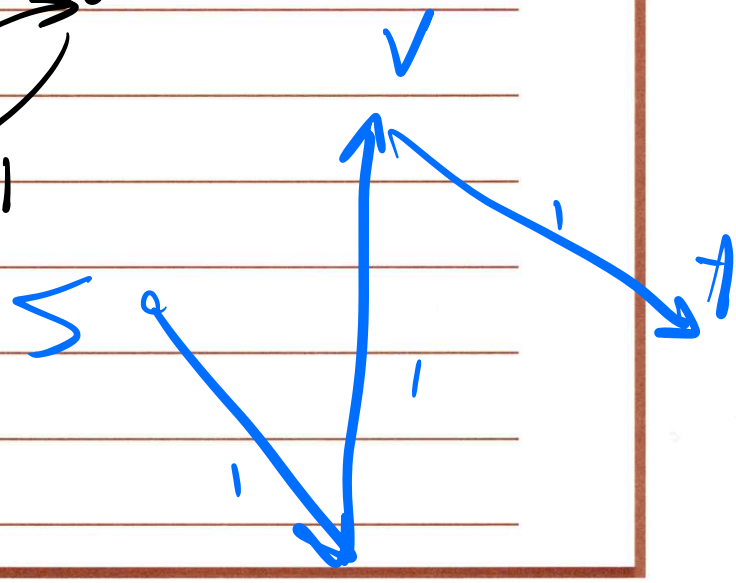


f_1





f_1



Scaled version of Ford-Fulkerson

Initially $f(e) = 0$ for all e in G

Set Δ to be the largest power of 2 that is no larger than the Max. cap. out of s.

While $\Delta \geq 1$

  While there is an s-t path in $G_f(\Delta)$

    Let P be a simple s-t path in $G_f(\Delta)$

    $f' = \text{augment}(f, P)$

    $f = f'$

    update $G_f(\Delta)$

  endwhile

  $\Delta = \Delta / 2$

endwhile

Return f

$\log C$

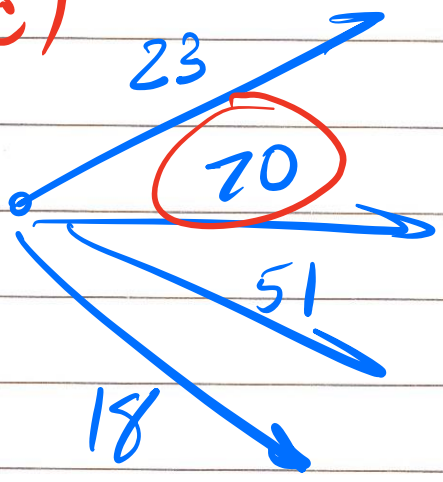
$O(m)$

$O(m)$

Δ-scaling phase

end of a Δ-scaling phase

$\Delta = 64$



Complexity of Scaled version $= O(m^2 \log C)$

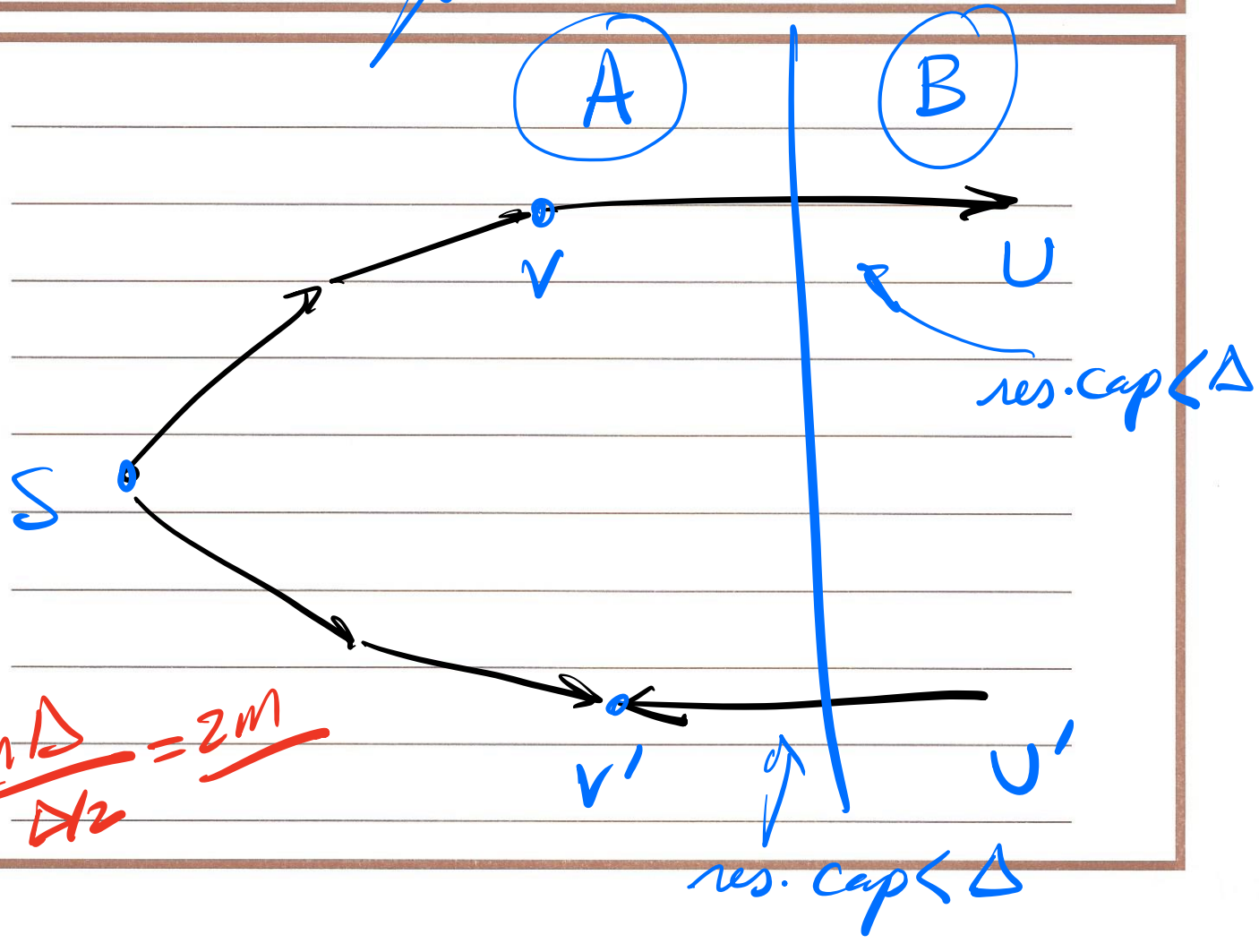
weakly polynomial

Background:

- During the Δ-scaling phase, each augmentations increases the flow value by at least Δ.

- Let f be the flow at the end of the Δ-scaling phase.

Claim: There is an s-t cut (A,B) in G for which $c(A,B) \leq v(f) + m\Delta$



Claim: The number of augmentations in a scaling phase is at most $2m$.

First scaling phase

How many times can we use each edge going out of s?

Other scaling phases

At the end of the Δ-scaling phase, we have already shown that $c(A,B) \leq v(f) + m\Delta$

If f^* is Max flow, Then $v(f^*) \leq c(A,B)$

$\Rightarrow v(f^*) \leq v(f) + m\Delta$

So, in the next scaling phase, where $\Delta' = \Delta/2$ how many iterations can we have?

Scaled version of Ford-Fulkerson

Initially $f(e) = 0$ for all e in G

Set Δ to be the largest power of 2 that is no larger than the Max. cap. out of s.

While $\Delta \geq 1$

While there is an s-t path in $G_f(\Delta)$

Let P be a simple s-t path in $G_f(\Delta)$

$f' = \text{augment}(f, P)$

$f = f'$

update $G_f(\Delta)$

endwhile

$\Delta = \Delta / 2$

endwhile

Return f

Strongly versus weakly polynomial
(relevant if input consists of integers)

An algorithm runs in strongly polynomial time if the no. of operations is bounded by a polynomial in the number of integers in the input.

An algorithm runs in weakly polynomial time if the no. of operations is bounded by a polynomial in the number of bits in the input, but not in the number of integers in the input.

Edmonds-Karp

Same as Ford-Fulkerson, except that each augmenting path must be a shortest path with available capacity.

Can be shown to have running time $O(nm^2)$

Ford-Fulkerson	$O(Cm)$	pseudo-polynomial
Scaled version of FF	$O(m^2 \lg C)$	weakly polynomial
Edmonds-Karp	$O(nm^2)$	strongly polynomial
Orlin + KTR	$O(nm)$	" "

Recently developed methods solve max flow in close to linear time WRT m.

approximation