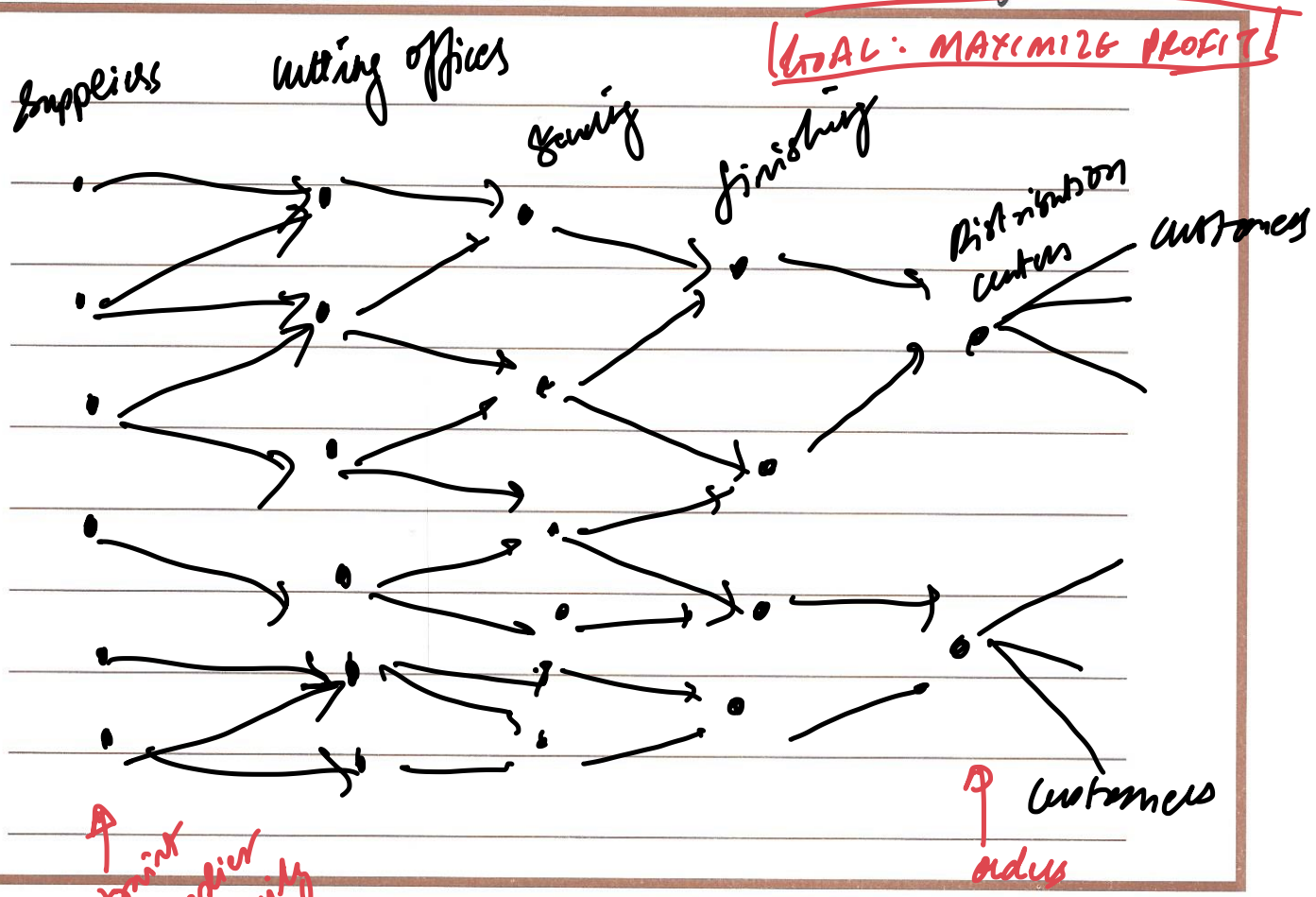
Network Flow

Examples of network flow problems:

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

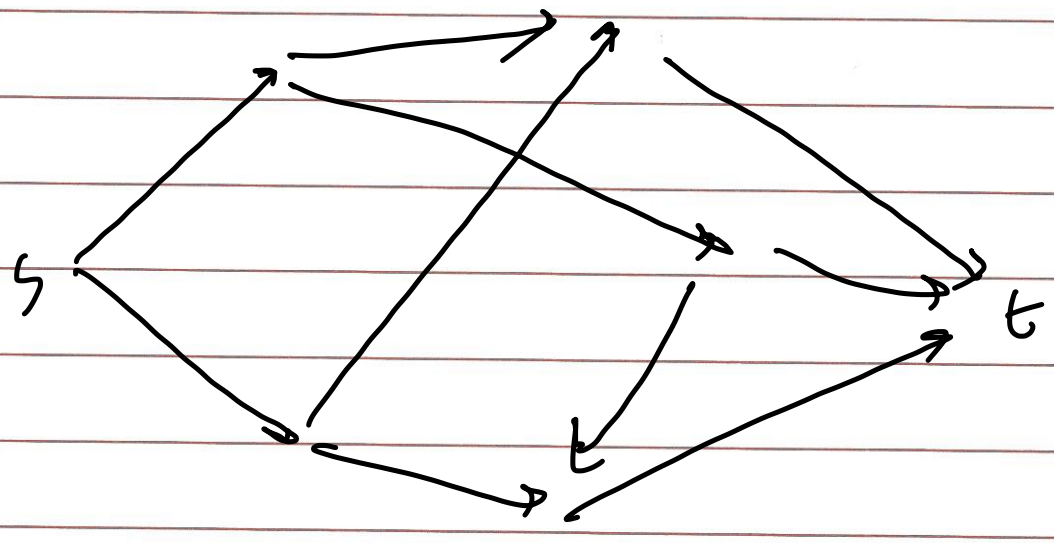
- How much traffic does the freeway system sustain for travel between two cities?

- 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
- all capacities are integers

$O(m+n) \rightarrow O(m)$

$m+n$: at max $m = n^2$

$O(n^2+n) = O(n^2) = O(m)$

Notation

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

1- Capacity constraint:

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

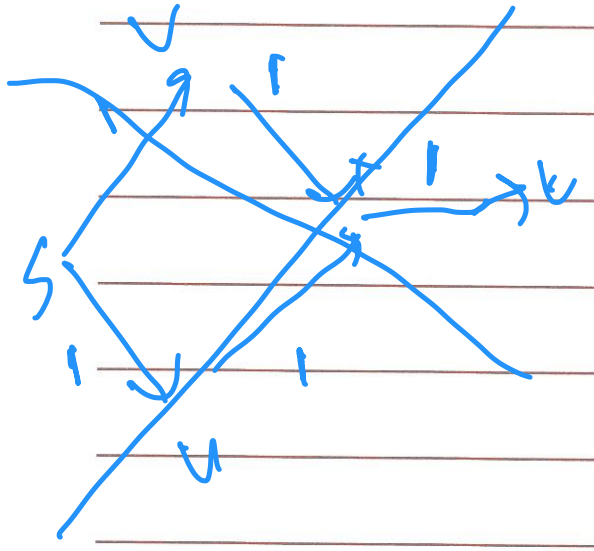
2- Conservation of flow

incoming flow = outgoing 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 flow.

same flow at all times



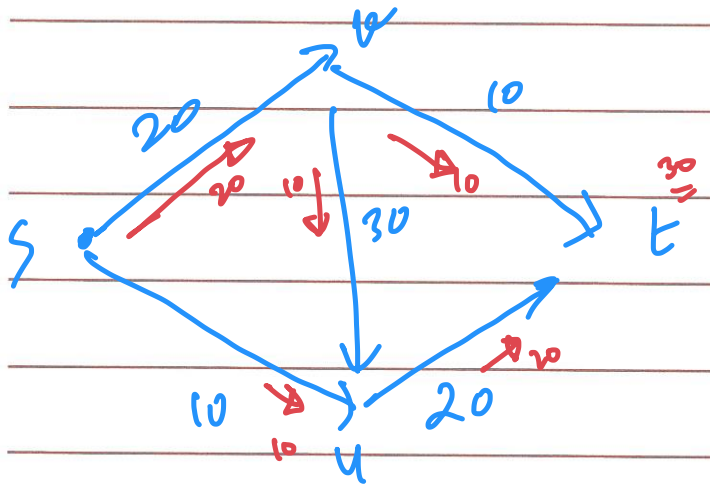
if it is 5 mL/s then it will always be 5 mL/s

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

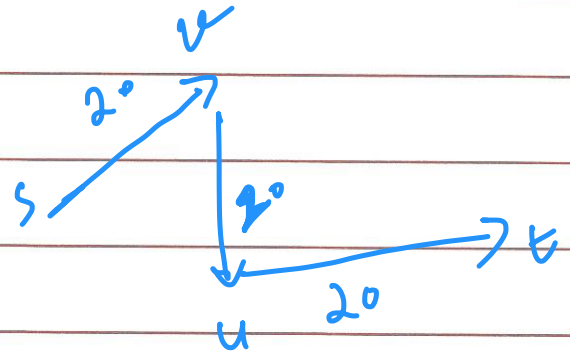
Given a flow network G, find an s-t flow with max. value.



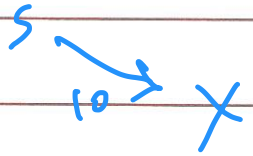
optimal flow 30

Try #1 Greedy

use highest capacity edges first



max flow = 20



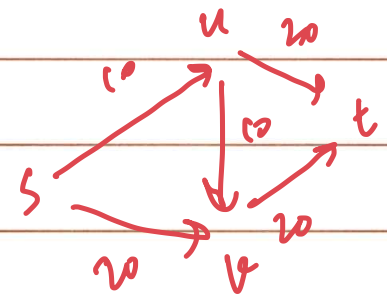
no more flow can be pushed

WRONG ANS

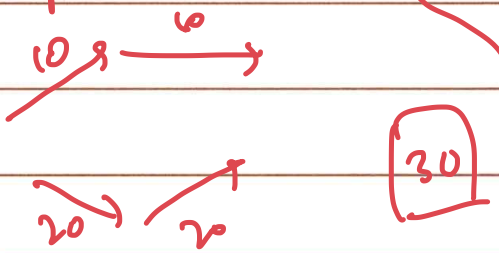
Try #2

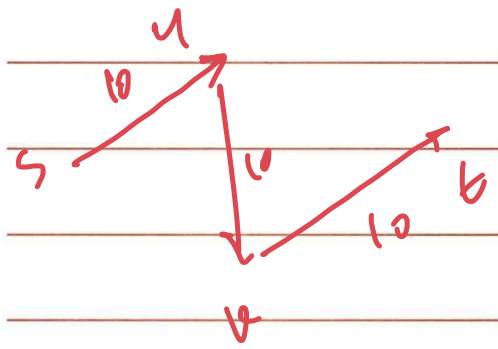
Use lowest capacity edges first

Counter example

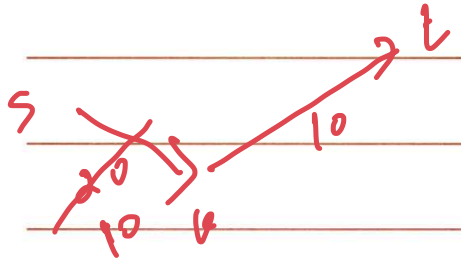


optimal





Flow = 10



Flow = 10 + 10 = 20

WRONG ANSWER

If we can undo decisions taken previously

we can use greedy iteratively

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

- 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 $c_e - f(e)$ (residual capacity) — 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 with capacity c_e

- if $f(e) = c_e$

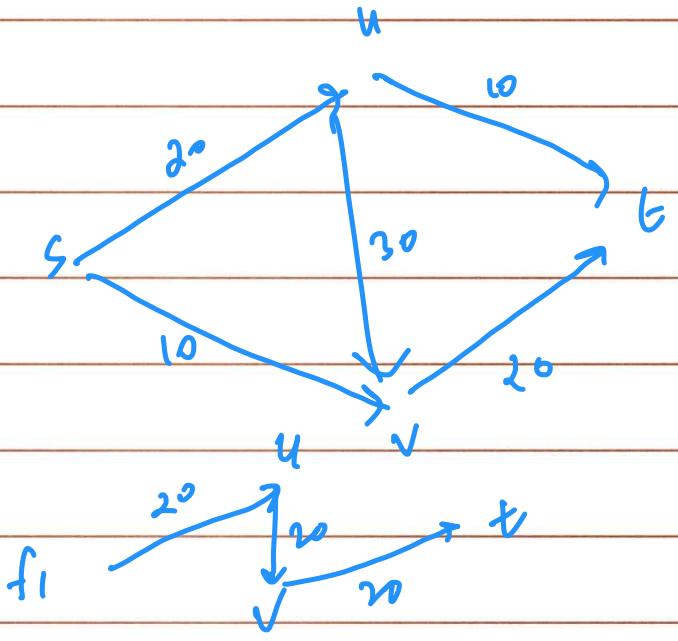
  (one backward edge with capacity c_e)

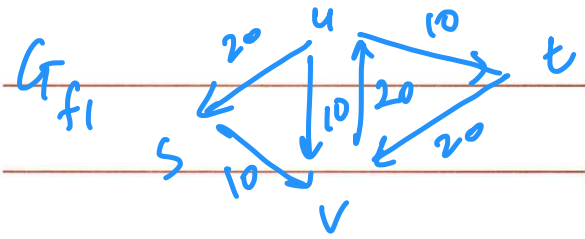
- if $f(e) < c_e$

  forward $\quad c_e - f(e)$

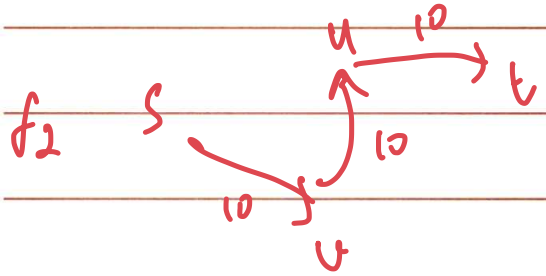
  backward $\quad f(e)$

G_f can have almost twice the number of edges in G

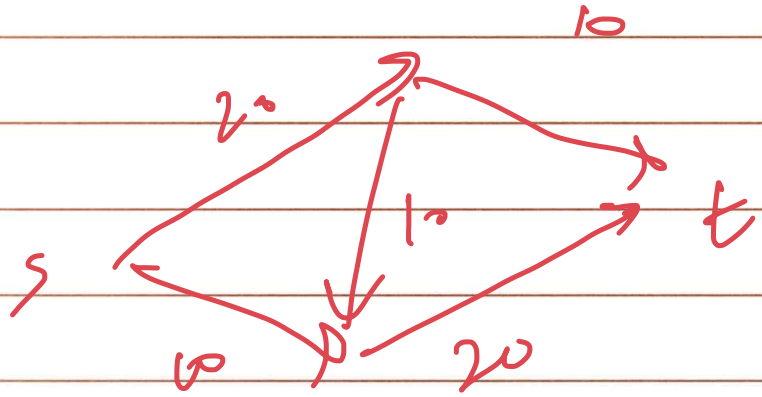




Now rather than pushing more flow in G we try to push flow in G_{f_1}



$f = f_1 + f_2$



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

fnc that gives capacity of all the edges

```
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) in G by b
        else (v,u) is a backward edge
            and let e=(u,v)
            then decrease f(e) in G by b
        end if
    endfor
```

```
Return (f')
```

We want to prove augmentation gives valid flow at every step

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,

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

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

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

$0 \leq f'(e)$; $f(e) + \text{bottleneck}(P)$; new flow $f'(e)$; ∴ new flow meets capacity constraint

$0 \leq f(e)$; +ve ; +ve

$$\boxed{0 \leq f'(e) \leq c(e)}$$

B- If e is a backward edge,

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

$\times -1$ $\quad -\text{bottleneck}(P) \leq -f(e)$

$+f(e)$ $\quad f(e) - \text{bottleneck}(P) \geq f(e) - f(e)$

$f(e)$; backward edge flow is going to be reduced by $f(e)$

$f'(e)$; $\boxed{f'(e) \geq 0}$

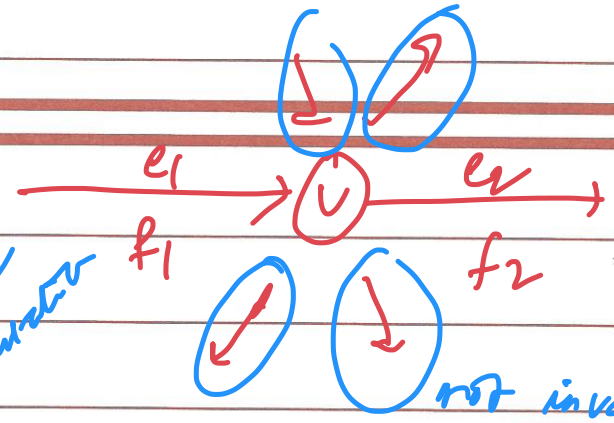
$\boxed{f'(e) \leq c_e}$ known

$$\boxed{0 \leq f'(e) \leq c_e}$$

2- Check conservation of flow

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

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



before augmentation

case 1
both edges in G_f corresponding to e_1 & e_2 or are forward edges

not involved in augmentation step

e_1 and e_2 are part of augmentation step

after augmentation

CASE 2
$e_1 \rightarrow$ $e_2 \leftarrow$

CASE 3
$e_1 \leftarrow$ $e_2 \rightarrow$

CASE 4
$e_1 \leftarrow$ $e_2 \leftarrow$

Similarly conservation of flow preserved in all cases

If we had conservation of flow before then after augmentation also conservation of flow.

∴ Invariant: If we start with valid flow before augmentation then after augmentation we will end up in valid flow

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
```

Simple path is a path without a cycle

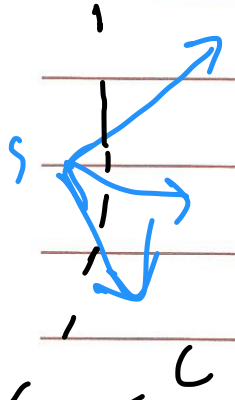
augment just sticks any path from s to t

Return f.

Proof of correctness should include:

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

① While loop terminates



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

$\sum c_e$ ≤ C

s ← No iteration possible with backward edge on s

There is a constraint that flow going out of s cannot be more than C

value of flow is going up in every iteration and there is a CAP to it

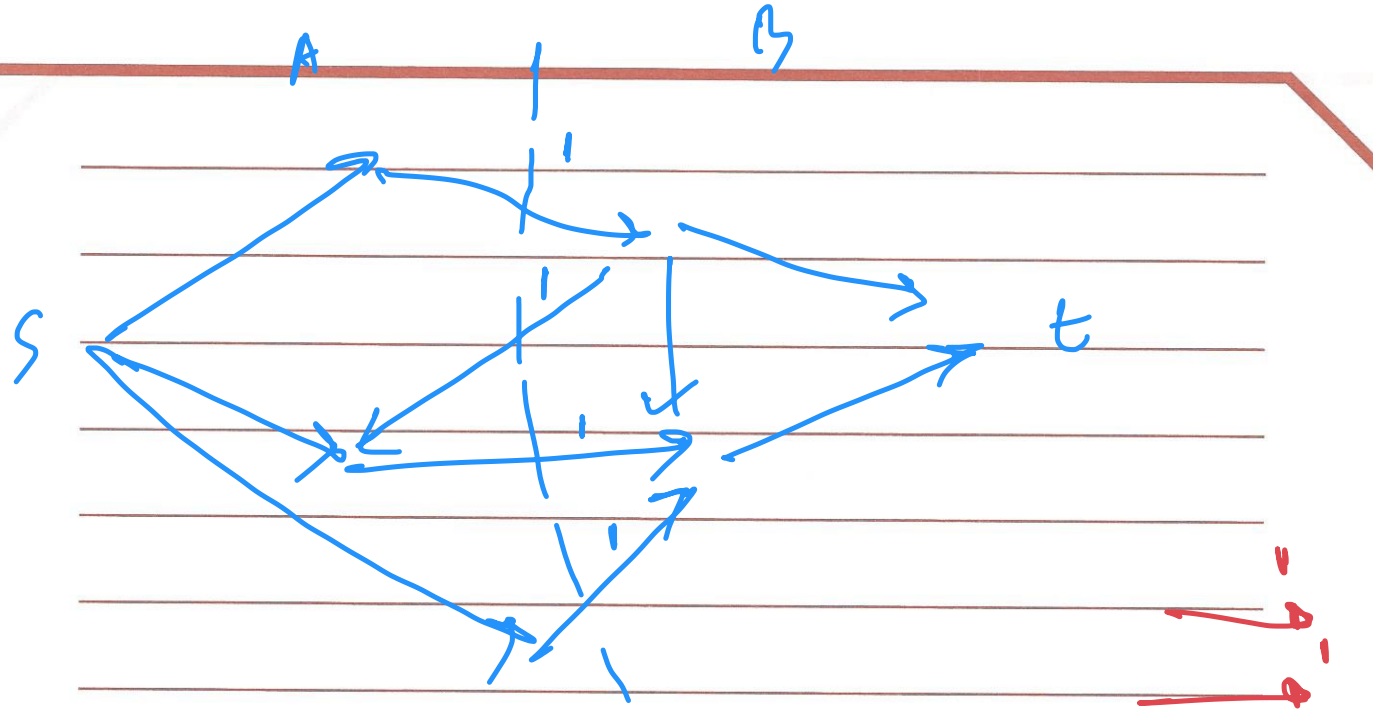
At max O(C) iterations as all values are integers

If it was not integer there can be ∞ iterations

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 $t \in B$



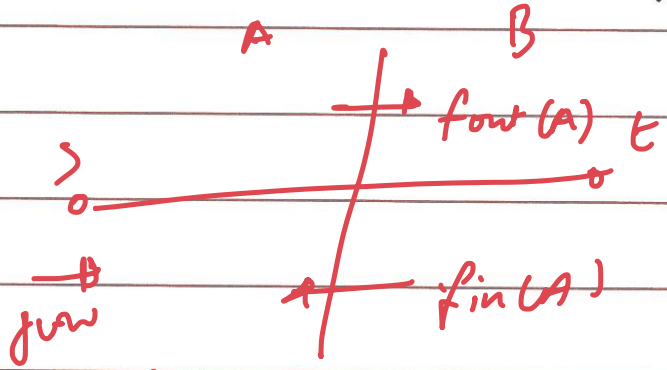
$C(A,B)$ = cap. of cut (A,B)

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

$C(A,B) = 3$

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

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



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

PROOF GIVEN IN BOOK

BUT ABOVE IS INTEGRABLE

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

Proof:

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

$f_{in}(A) \geq 0$

$$\therefore \quad v(f) \leq f_{out}(A)$$

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

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

capacity of (A, B) cut

$v(f) \leq$ Capacity of (A, B) cut

$v(f) \leq$ All cut capacity (A, B)

So, the maxFlow 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

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

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

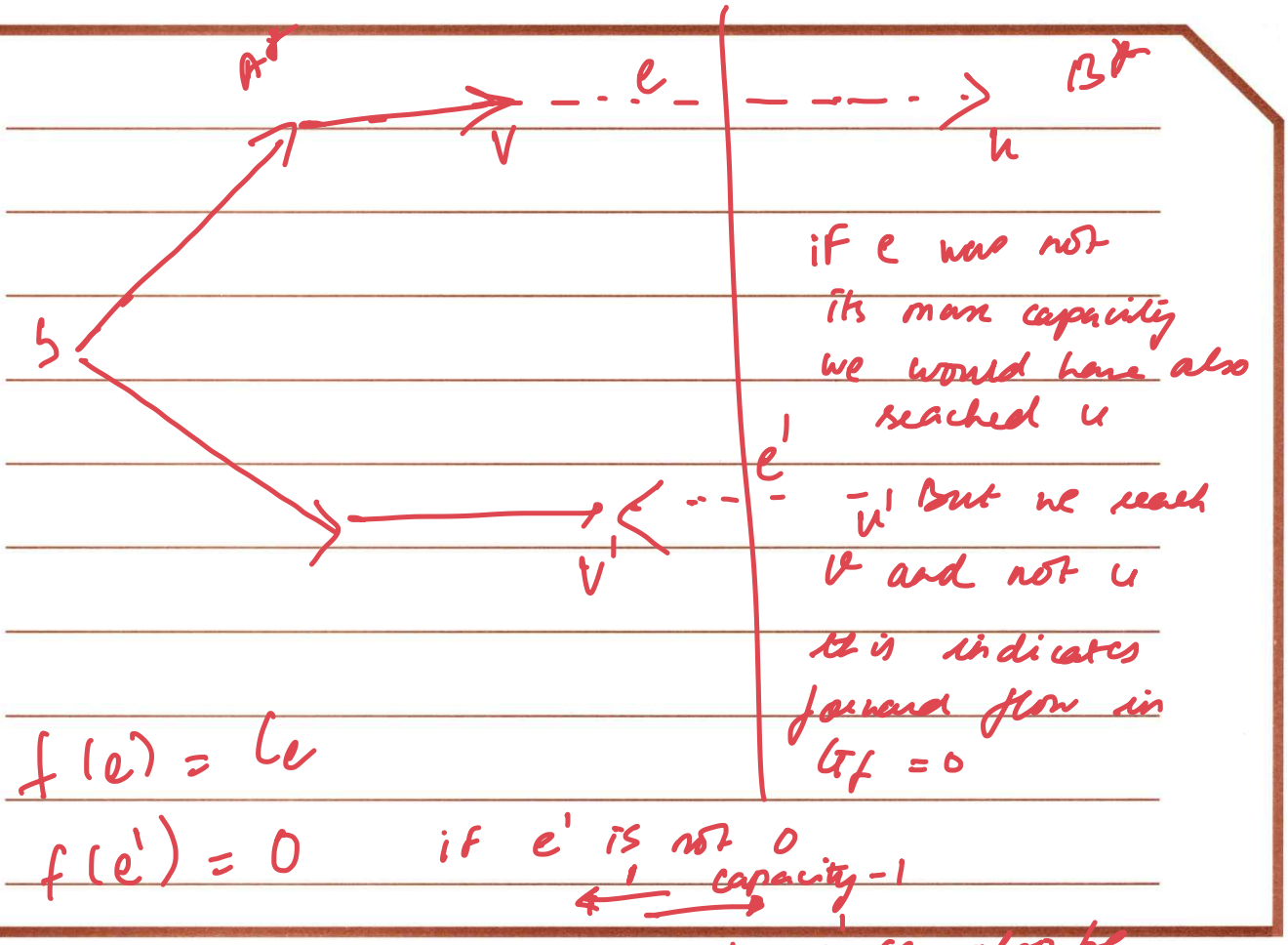
if $v(f) = c(A, B)$

that is maximum flow.

Now we have to prove cut $(A^* B^*)$ exist such that

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

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



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

$$= \sum_{e \text{ out of } A^*} c_e - 0$$

$$v(f) = \text{capacity of cut } (A', B')$$

when ford fulkerson terminates

value of flow = capacity of cut (A', B') indicating we have reached max flow.

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

$(A^* B^*)$ has minimum capacity of any s-t cut in G

- Say we are given max flow f
  How do you find min cut?

Given max flow

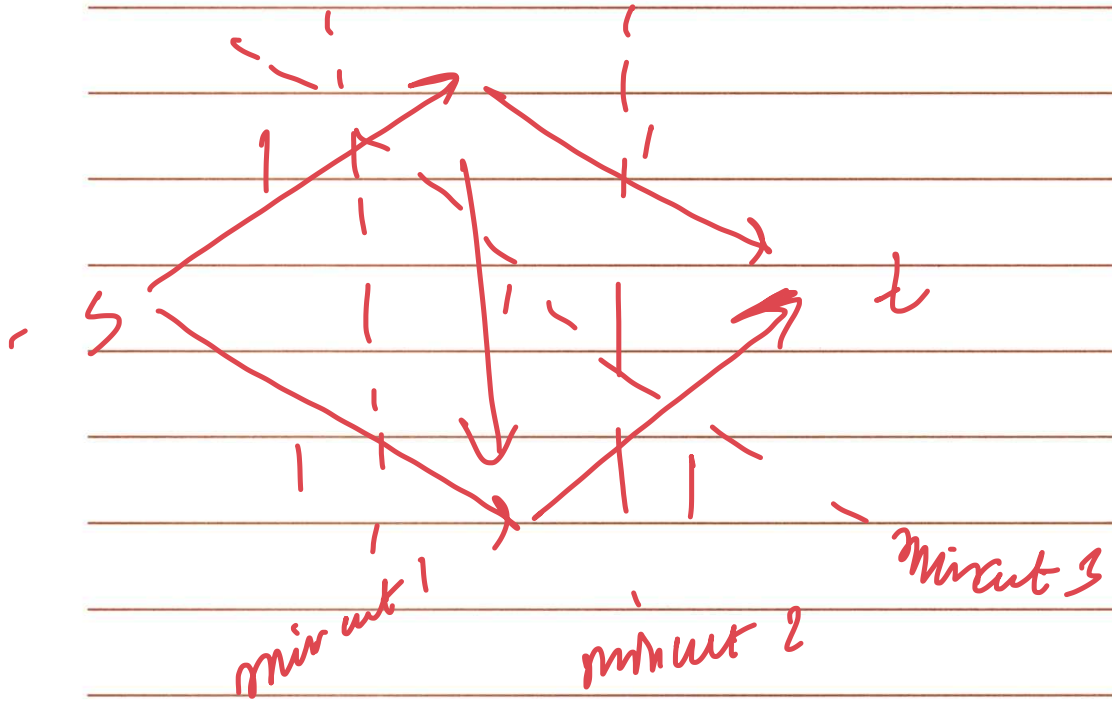
↓

find residual graph → BFS or DFS

↓

find all nodes reachable from s A^*

others B^*



One process finds closest mincut from s

How to find if there is only 1 mincut-

find mincut from s

Rev graph

find mincut from T

If both are same

There is a unique mincut

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 G_f

Let P be a simple s-t path in G_f

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

$f = f'$

update G_f

endwhile

Return f.

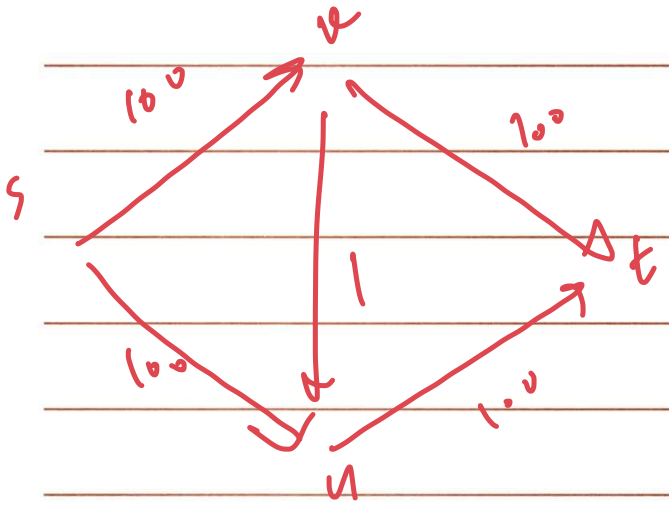
$O(Cm)$ — pseudo-polynomial run time

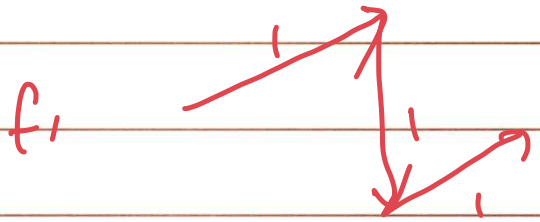
C = numerical value

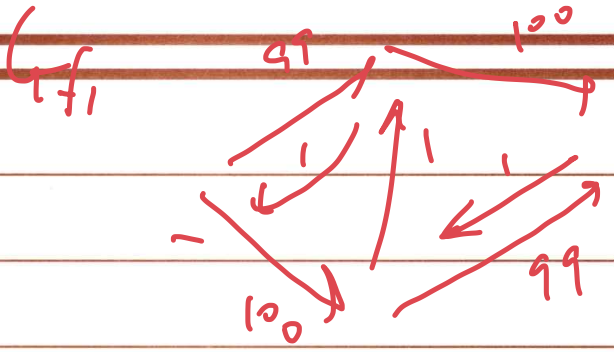
There are efficient sol's for max flow prob

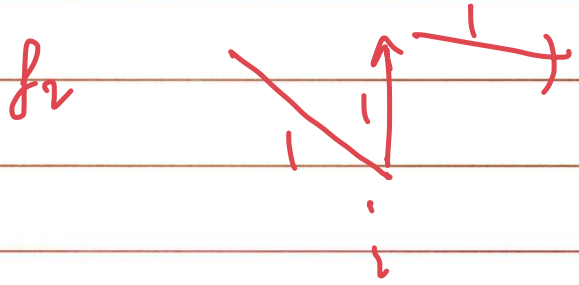
But Ford Fulkerson is not one of them.

You have successfully computed a maximum s-t flow f for a network G = (V; E) with integer edge capacities. Your boss now gives you another network G’ that is identical to G except that the capacity of exactly one edge is decreased by one. You are also explicitly given the edge whose capacity was changed. Describe how you can compute a maximum flow for G’ in O(|V| + |E|) time.









If we keep doing this we would have
100 iterations

But we can do in 2 iterations

How to present?

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.

start consider low capacity edges

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

  $\Delta = \Delta / 2$

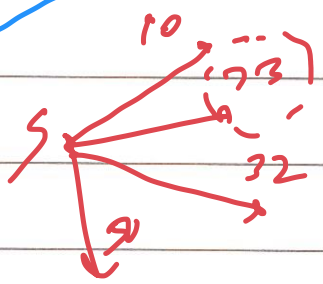
endwhile

Return f

$\log_2 C$

Ford Fulkerson

Scaling phase



$\therefore \Delta = 64$

get rid of edges

- with value less than Δ
- value of bottleneck value is at least Δ as all edges less than Δ are removed

Background:

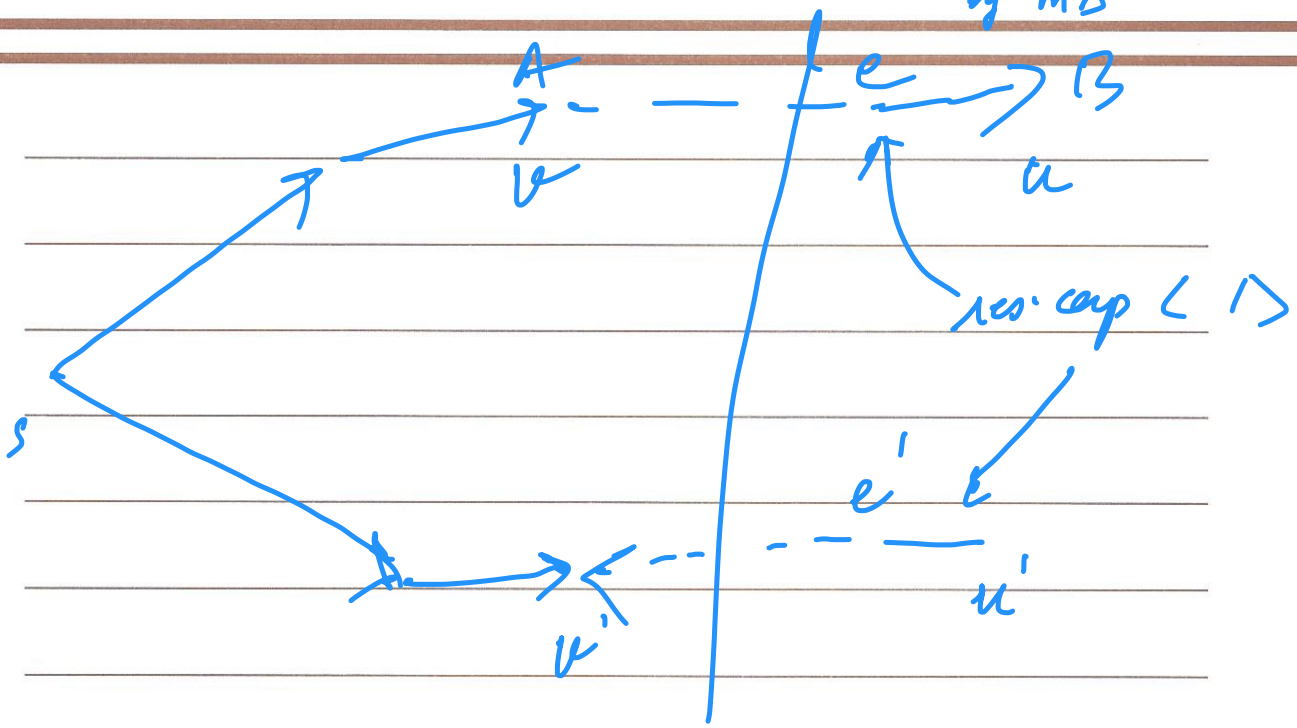
- During the Δ-scaling phase, each augmentations increases the flow value by at least Δ.

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

watch video again

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

at any given time I cannot increase flow of cut by $m\Delta$



a res cap(e) < Δ as if it was greater than Δ we would have reached t

and u would belong to A but that is not the case

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?

we have to get $\frac{m\Delta}{\Delta/2} = 2m$

$\Delta/2$ = next scaling phase

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

$O(\log_2 C)$ — $O(m)$ — $O(m)$

$O(n^2 \log_2 C)$ — efficient

$\log(C)$ = number of bits required to represent C

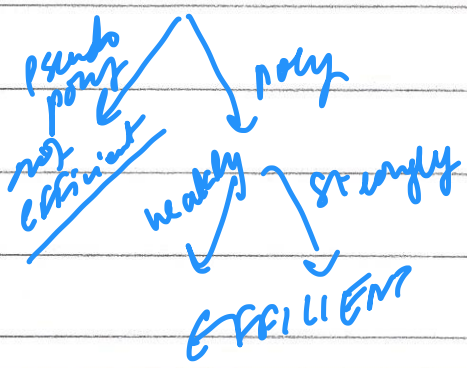
representation size ↑ input

weakly polynomial time

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.

shortest = least number of edges

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

Same as Ford Fulkerson just when selecting augmenting path choose shortest path

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	$O(nm)$	" "

Choose among these depending on dense graph or sparse graph

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

approximation

Discussion 8

1. You have successfully computed a maximum s-t flow f for a network G = (V; E) with integer edge capacities. Your boss now gives you another network G' that is identical to G except that the capacity of exactly one edge is decreased by one. You are also explicitly given the edge whose capacity was changed. Describe how you can compute a maximum flow for G' in O(|V| + |E|) time.

2. You need to transport iron-ore from the mine to the factory. We would like to determine how long it takes to transport. For this problem, you are given a graph representing the road network of cities, with a list of *k* of its vertices ($t_1, t_2, \ldots, t_k$) which are designated as factories, and one vertex *S* (the iron-ore mine) where all the ore is present.
We are also given the following:

- Road Capacities (amount of iron that can be transported per minute) for each road (edges) between the cities (vertices).
- Factory Capacities (amount of iron that can be received per minute) for each factory (at $t_1, t_2, \ldots, t_k$)
- The amount of ore to be transported from the mine, *C*

Give a polynomial-time algorithm to determine the minimum amount of time necessary to transport and receive all the iron-ore at factories.

3. In a daring burglary, someone attempted to steal all the candy bars from the CS department. Luckily, he was quickly detected, and now, the course staff and students will have to keep him from escaping from campus. In order to do so, they can be deployed to monitor strategic routes.

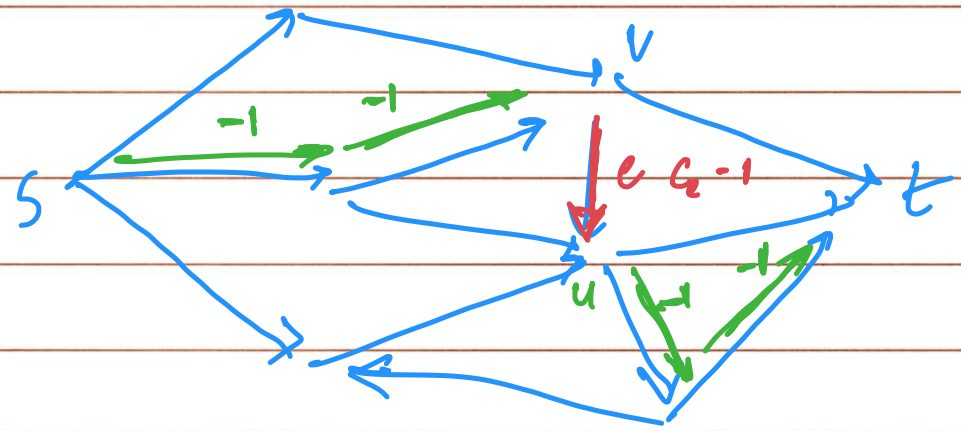
More formally, we can think of the USC campus as a graph, in which the nodes are locations, and edges are pathways or corridors. One of the nodes (the instructor's office) is the burglar's starting point, and several nodes (the USC gates) are the escape points — if the burglar reaches any one of those, the candy bars will be gone forever. Students and staff can be placed to monitor the edges. As it is hard to hide that many candy bars, the burglar cannot pass by a monitored edge undetected.

Give an algorithm to compute the minimum number of students/staff needed to ensure that the burglar cannot reach any escape points undetected (you don't need to output the corresponding assignment for students — the number is enough). As input, the algorithm takes the graph G = (V,E) representing the USC campus, the starting point s, and a set of escape points $P \subseteq V$. Prove that your algorithm is correct and runs in polynomial time.

4. We define a most vital edge of a network as an edge whose deletion causes the largest decrease in the maximum s-t-flow value. Let f be an arbitrary maximum s-t-flow. Either prove the following claims or show through counterexamples that they are false:

(a) A most vital edge is an edge e with the maximum value of c(e).
(b) A most vital edge is an edge e with the maximum value of f (e).
(c) A most vital edge is an edge e with the maximum value of f (e) among edges belonging to some minimum cut.
(d) An edge that does not belong to any minimum cut cannot be a most vital edge.
(e) A network can contain only one most vital edge.

(1)



CASE 1: $f(e) < c_e$ basically no change

if we do $u-1$

still it will be valid flow

---

case 2: $f(e) = c_e$

Find any path from $s-v$ that have valid flow

at least 1 unit flow.

similarly find path from $u-t$

Now this will give us new flow.

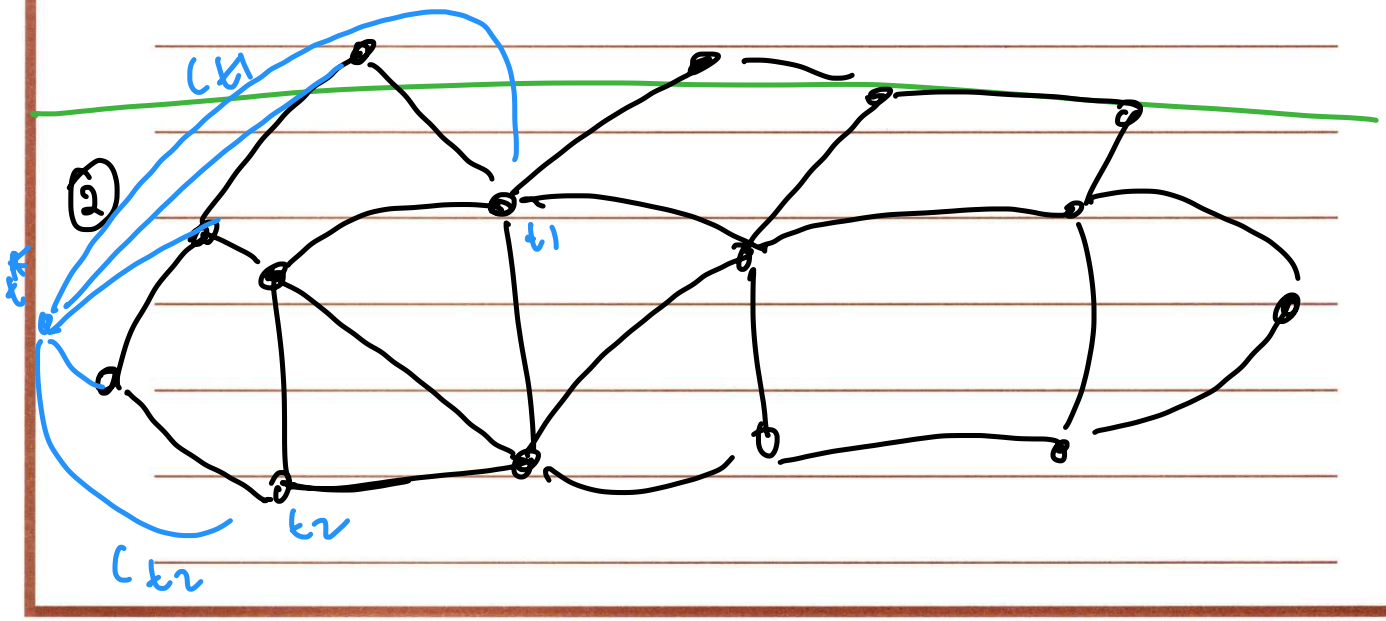
It may be max flow or not

∴ we construct residual graph for new flow

Perform 1 augmentation

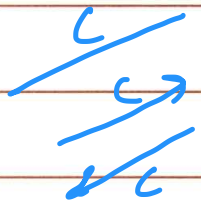
If value decr by k

Atmost k augmentation steps



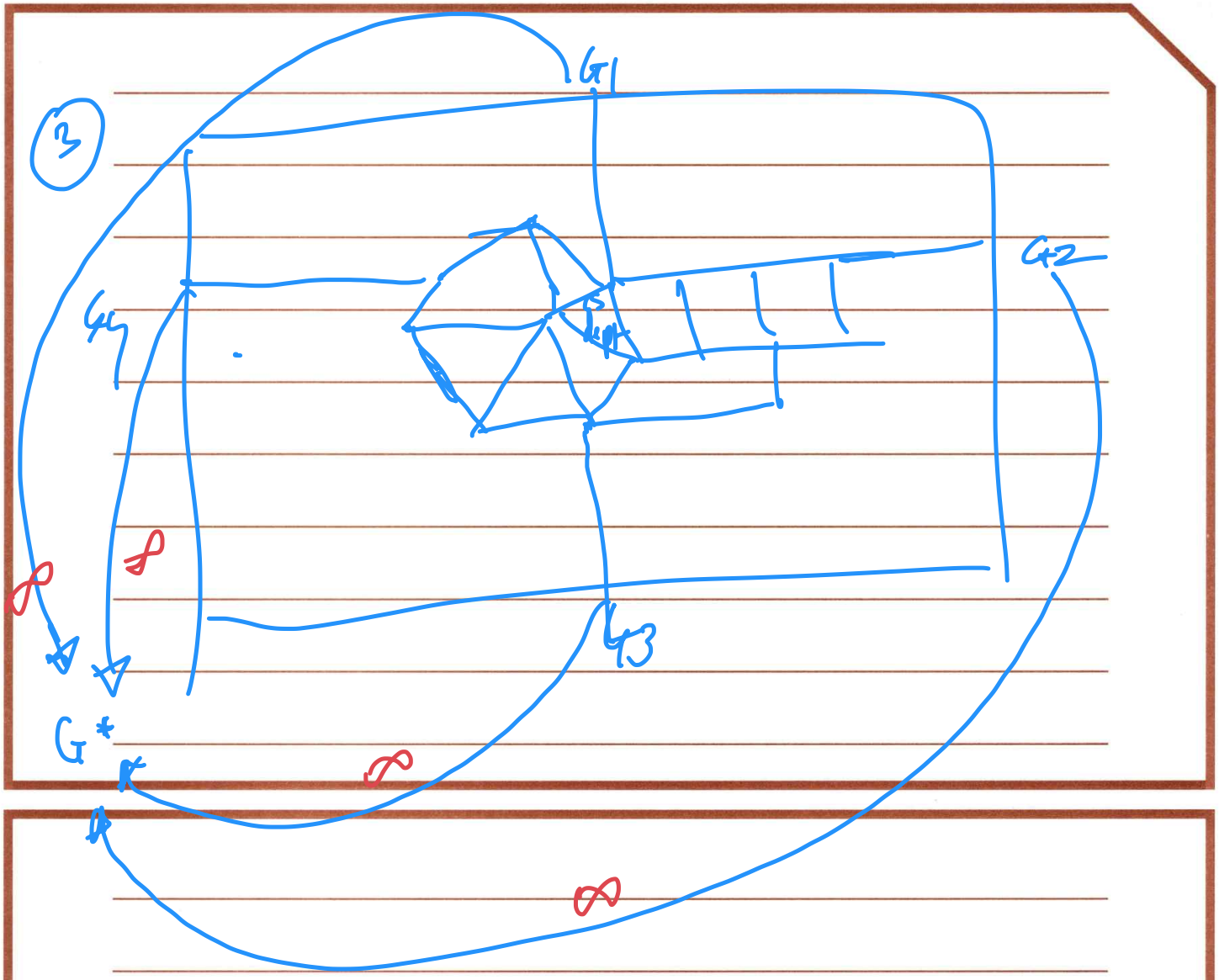
This is also undirectional

make it directional



Run max flow → $v(f) = x$ tons/hr

hrs $t = \frac{C}{x}$ tons



give each edge weight = 1
as it takes 1 person to monitor 1 edge

again convert graph to directional graph.

connect all gates to single sink
weight of that edge ⇒ ∞ because we
don't want min cut on these edges as they
are fictional

rather than ∞ we can use sum(all edges of graph) + 1

as it will ensure it won't consider these edges.

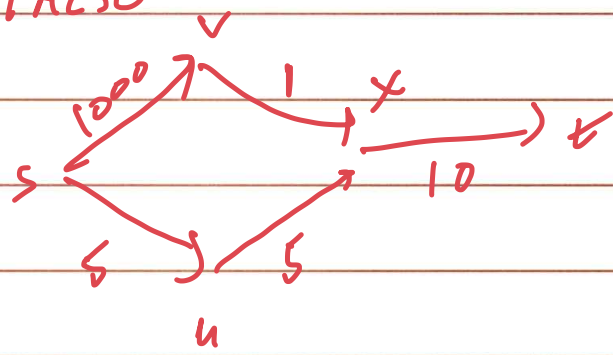
Ford Fulkerson $O(Cm)$ → at most n

$\therefore O(m \cdot n)$ polynomial time

we can use Ford Fulkerson

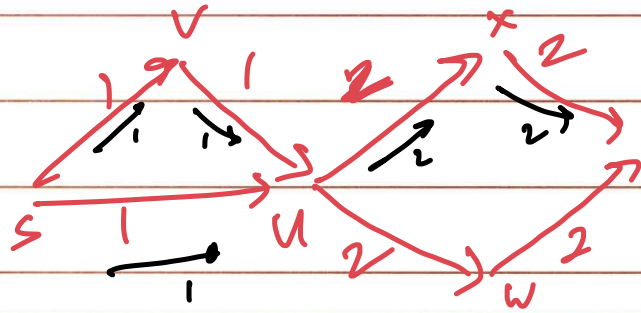
(4)

(a) FALSE



X-t is most vital edge and not s-v

(b) False

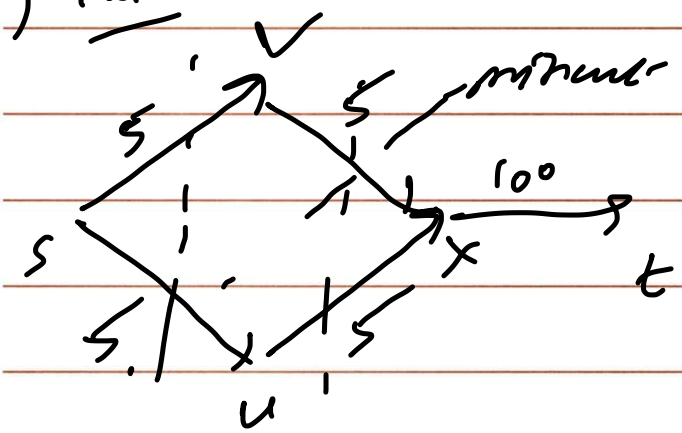


most vital edge vu or su

not ux and xt

(c) False

most vital edge x-t



(d) False same as above

(e) False