CMSC 27200 - Guerilla Section 6.

Topics.

1. Network flow.
   - Ford-Fulkerson
   - Augmenting paths and residual graphs.
   - Remarks on running time.
2. Flow reductions.
   - Bipartite matching:

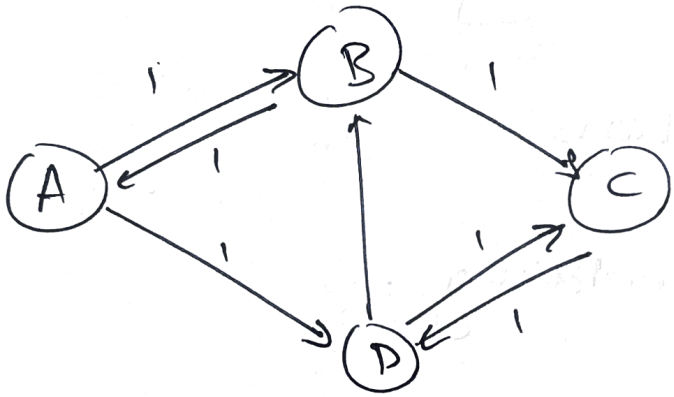
~~Linear Programming~~ & Network Flow

So what's the SparkNotes version of ~~LPs~~ and network flow?

⟹ Network Flow.

directed graph.

Suppose you have a computer network.



Each edge is labeled ~~as~~ w/ a maximum amt of data that can be sent b/w pairs of computers (bandwidth).

Now you pick a source vertex A and sink vertex B.

Your goal is to send the maximum amt of data btw source and sink.

↳ This is called maximum flow.

How do you model this?

Defn: Given $G=(V,E)$ directed ~~and $s,t \in V$~~.

capacities $c_e > 0$ $\forall e \in E$ and source-sink pair $s,t \in V$., an s-t-flow in G is a collection of values $\{f_e\}_{e \in E}$ s.t. associated w/ each edge s.t.

1) It satisfies capacity constraints: $\forall e \in E$.

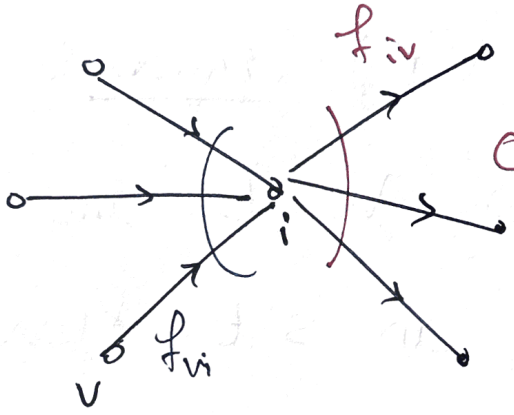
$$0 \leq f_e \leq c_e.$$

2) Flow is conserved at each vertex except the source and sink.

$\forall i \in V \setminus \{s,t\}$ we have

$$\underbrace{\sum_{(v,i)\in E} f_{vi}}_{\text{flow going into } i.} = \underbrace{\sum_{(iv)\in E} f_{iv}}_{\text{flow leaving } i.}$$

Pictorially...



incoming flow

$\sum_{vi \in E} f_{vi}$

Outgoing flow

$\sum_{iv \in E} f_{iv}$

flow conservation $\equiv$ "in flow = out-flow"

And now we can state the problem precisely

the maximum flow problem (aka "single-commodity maximum flow") is defined as...

INPUT: A directed graph $G = (V, E)$, edge capacities $c_e > 0$ $\forall e \in E$, source vertex $s \in V$ and sink vertex $t \in V$

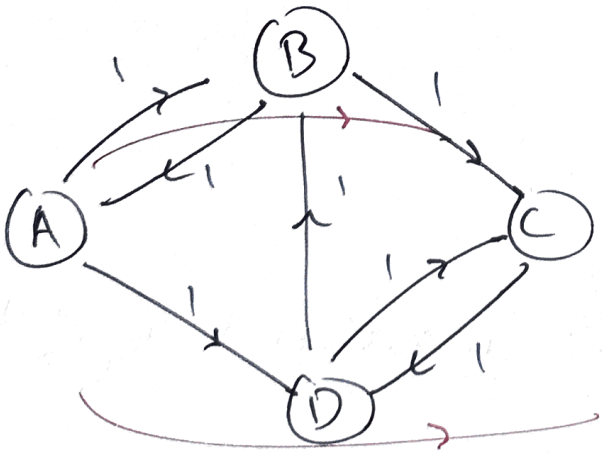
OUTPUT: An s-t flow of G w/ maximum size ~~i.e.~~ where

~~max size $(f) := \max \sum$~~

$$\text{size}(f) := \sum_{(s,v) \in E} f_{sv}.$$

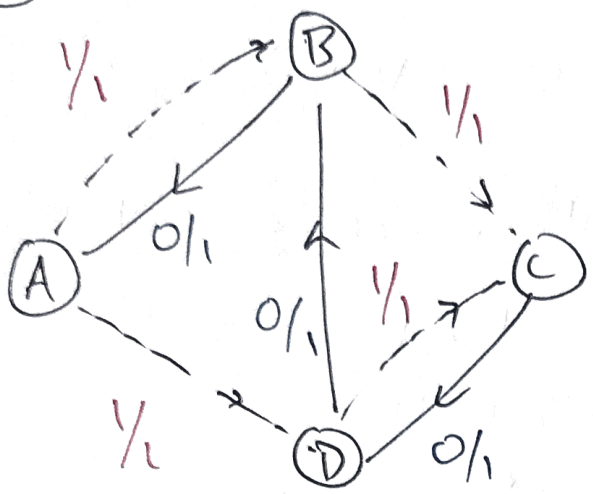
amt of flow leaving s.

Example: What's a maxflow of this graph? w/ $s = A$, $t = B, C$.



send one unit b/w $A \to B \to C$

another b/w $A \to D \to C$.



Edge	Flow
AB	1
BA	0
BC	1
AD	1
DB	0
DC	1
CD	0

$$\text{Size}(f) = \sum_{(s,i) \in E} f_{si} = f_{AB} + f_{AD} = 1 + 1 = 2.$$

→ Questions?

~~Sparknote pandalines.~~

⟹ Sparknote pandalines.

~~= How do you solve~~

There are two questions to ask:

1. How do you compute a maxflow (algo?)

2. How do you know you've computed a maxflow?

3. What can I use maxflow to solve?

↳ Item #1: how do I compute maxflow?

- Ford-Fulkerson: on first pass this can be described as the following method.

  1) Start w/ zero flow.

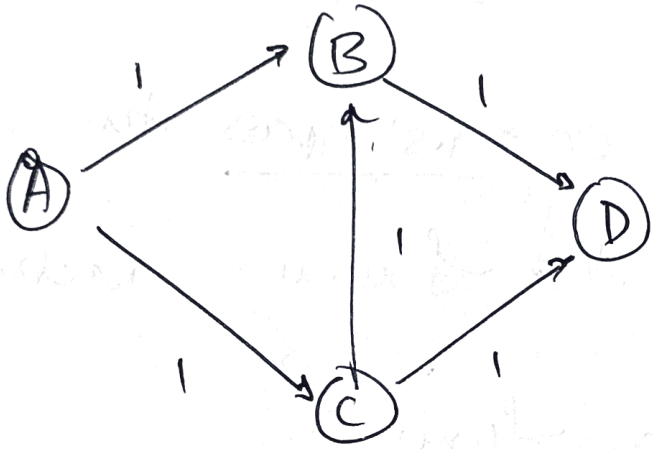
  2) Repeat: find an s-t path w/ non-zero capacity and push as much flow as possible.

But this is not an algorithm!

↳ we didn't specify how to find the paths.

→ if you're not careful, this method won't find a maxflow

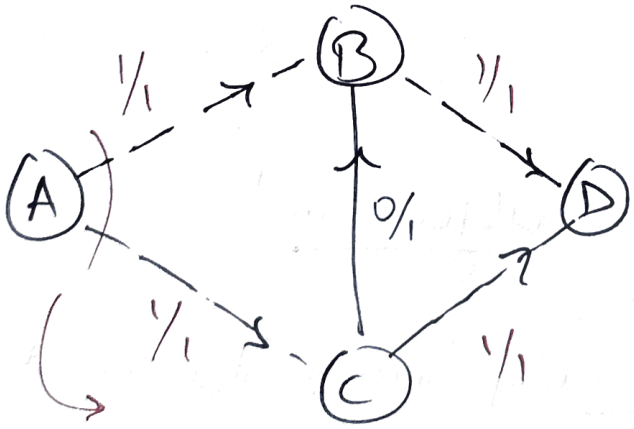
↳ Good example: graphs that look like.



s = A

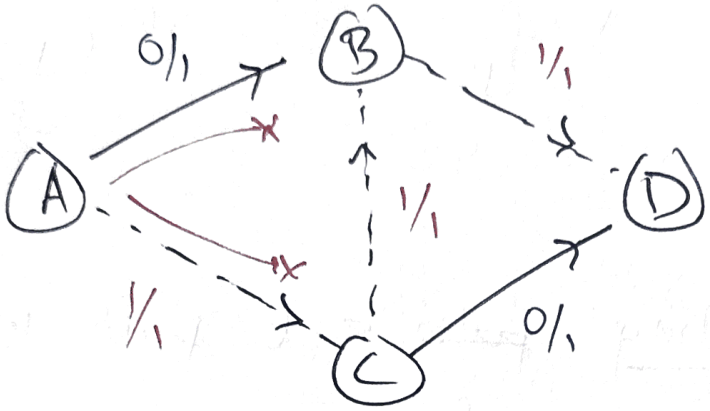
t = D.

If Ford-Fulkerson picks paths A-B-D then A-C-D then you get a maxflow.



Size = 2.

But if Ford-Fulkerson picks A-C-B-D then you're toast.



There's no A→D path w/ available capacity and you've only computed a size = 1 flow

→ The fix is to use the residual graph and find augmenting paths

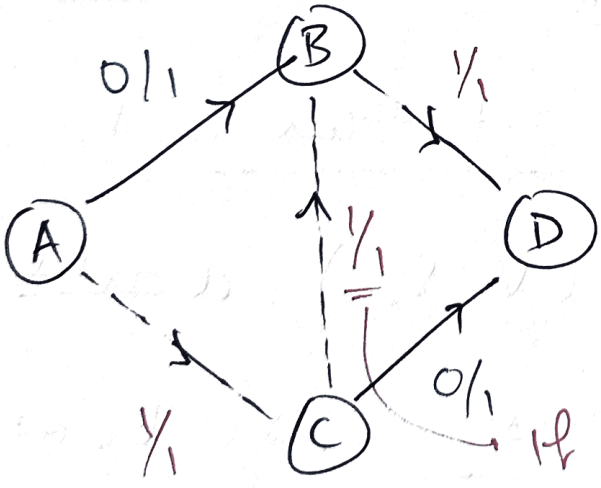
Defn: Given a directed graph $G=(V,E)$ ~~and~~ w/ capacities $c_e > 0$ and s-t flow f, the residual graph is $G_f = (V, E_f)$. ~~a tuple~~ w/ residual capacities. $c_e^f > 0$ $\forall e \in E_f$.

$$c_{ij}^f = \begin{cases} c_{ij} - f_{ij} & \text{if } (i,j) \in E \text{ and } f_{ij} \le c_{ij} \\ f_{ji} & \text{if } (j,i) \in E \text{ and } f_{ji} > 0. \end{cases}$$

→ An augmenting ~~path~~ s-t path in G^f is an s-t path in G^f w/ ~~min-flow~~ > 0 capacity on each edge

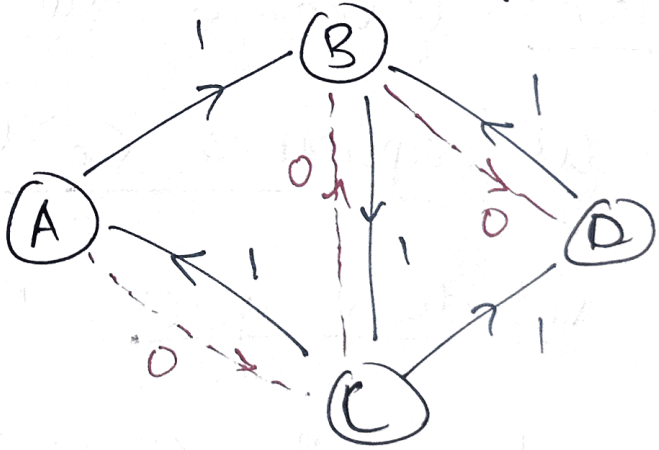
→ What's the intuition?

→ The residual graph tells you how to "undo" a flow that you pushed.



If we could "undo" this ~~edge~~ flow along this edge then we can get unstuck.

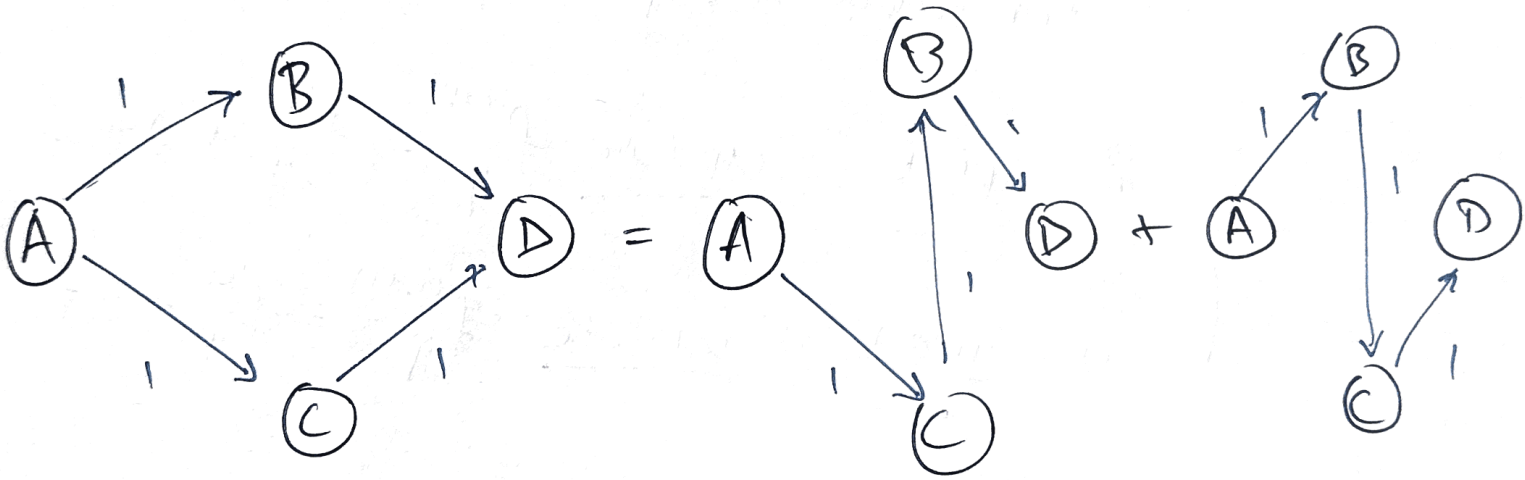
G_f.



A → B → C → D is an aug path!

If you route ~~an augment~~ flow along the augmenting path $A \to B \to C \to D$ in G. the residual graph G^f. then you undo flow sent in the $C \to B$ direction.

↳ Summing the flows ~~across all~~ gives you the max flow.



→ This is Ford-Fulkerson.

PROC: FORD-FULKERSON.

INPUT: $G = (V, E)$ directed, $c_e > 0$ $\forall e \in E$.

source $s \in V$, sink $t \in V$.

1. Initialize $f_e^{(0)} = 0$. $\forall e \in E$.

2. Do for $t = 0, 1, \ldots$: ~~until no augmenting~~

   - Compute $G^{f^{(t)}}$
   - Compute P an augmenting ~~path~~ s-t path in $G^{f^{(t)}}$, let flow pushed = Δf.
   - if no path: return $f^{(t)}$
   - For each $(i,j) \in P$:

$$\begin{cases} \text{if } ij \in E: \text{ update } f_{ij}^{(t+1)} = f_{ij}^{(t)} + \Delta f. \\ \text{if } ji \in E: \text{ update } f_{ij}^{(t+1)} = f_{ij}^{(t)} - \Delta f. \end{cases}$$

Do you need to recompute G^{f_t} from scratch in each iteration? Nope!

↪ Maintain a counter on residual capacities.

→ Questions?

→ How do you know terminating = maxflow?

① Duality: mincut = maxflow.

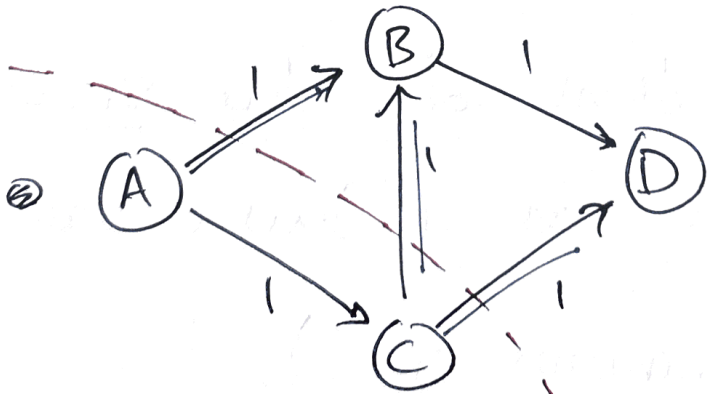
– ~~Hmmk~~ If I'm sending data from $s \to t$. then all the data I send has to cross from one side to another.

↳ Formalize: Let $S \subseteq V$ be a subset of vertices. (this is called a cut).

↳ The capacity of a cut is

$$Cap(S) := \sum_{ij \in E : i \in S, j \notin S} c_{ij}$$

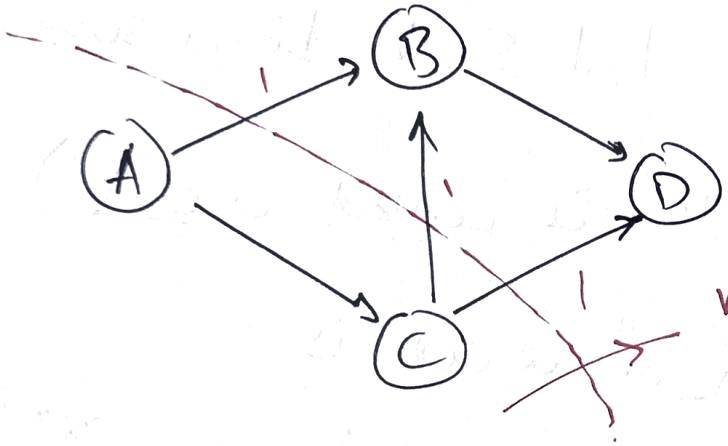
] sum of capacities for edges leaving S.



$S = \{A, C\}$

$Cap(S) = 1 + 1 + 1 = 3.$

- If the flow I send respects capacity constraints., then the size of the flow can only be at most the capacity of any s-t cut. (i.e. cut S s.t. $i \in S$ and $t \notin S$).



↳ maxflow value $\leq$ Cap(S)

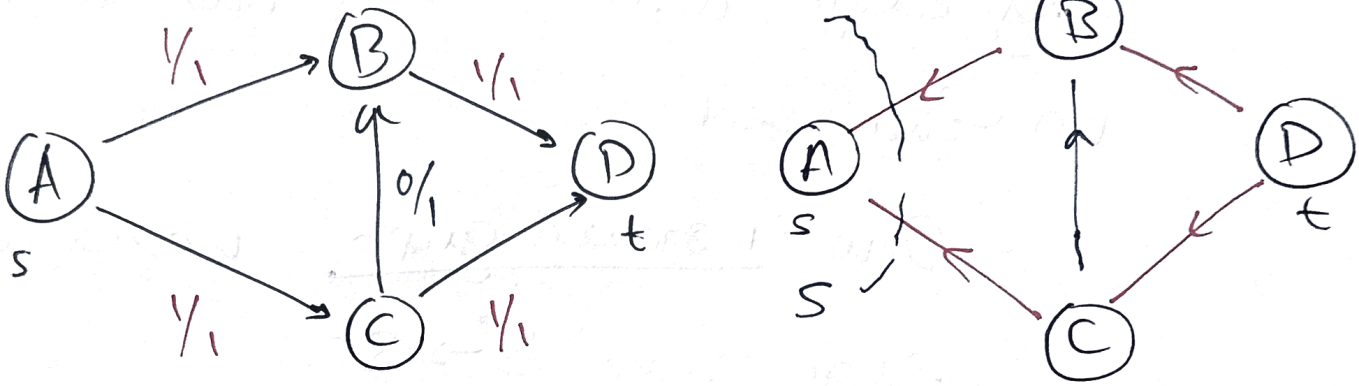
$\forall$ s-t cuts $S \subseteq V$.

- But now... think of the smallest capacity s-t cut (this is called the ~~mincut~~ minimum cut.).

  - Let's compare it to the size. ~~value~~ of the

maxflow computed by ford-fulkerson

- If FORD-FULKERSON terminates then there are no more augmenting s-t paths in the residual graph.



- Take the set of vertices reachable from s in G^f. This is a cut S^*.
- The capacity of S^* (in G.) = value of flow computed by FORD-FULKERSON.

↳ ~~if not! Then $\exists$ $e \in E$ s.t. e leaves S and~~

↳ Any edge from $S^* \to \bar{S}^*$ has to be at full capacity

- As the residual graph only has edges going into S^*

↳ Any edge from $\bar{S}^* \to S^*$ has to have no flow sent

- O.w. residual graph would have edge from $S^* \to \bar{S}^*$

- And now what we get is.

$$Cap(S^*) = Size(f)$$

- But remember ~~$\forall$~~ $\forall$ ~~cut~~ s-t cut S.

~~max flow~~ $Size(\text{max flow}) \leq Cap(S)$ so

let ~~S_{min}~~ $S = \text{min cut}$.

$$\text{Cap}(\text{min s-t cut}) \leq \text{Cap}(S^*)$$

$$= \text{Size}(f)$$

$$\leq \text{Size}(\text{max flow})$$

$$\leq \text{Cap}(\text{min s-t cut}).$$

→ min cut = max flow and Ford-Fulkerson outputs a max flow.

→ Questions?

Question: ~~Suppose $e \in E$ is an edge such that decreasing its~~

~~Suppose I pick an edge e~~

Suppose S is a minimum capacity s-t cut and e is an edge leaving S.

↳ How does the max flow value change if I decrease c_e?

↳ How does the maxflow value change if I increase c_e?

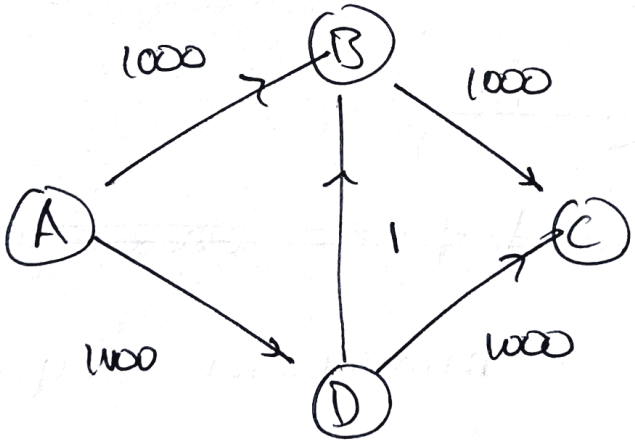
→ Think about it.

(in slides)

⟹ A remark about running time.

Does Ford-Fulkerson always run in polynomial time?

↳ No! Because we're still not careful in specifying what augmenting path to take



give me a sequence of augmenting paths s.t. it takes 1000 augmenting paths to compute the maxflow.

→ If $U = \max_{e \in E} c_e$. FORD-FULKERSON

might take $O(|E| \cdot U)$ time.

→ Doomed? nah.

↳ If you pick the augmenting path of shortest length from s-t then it terminates in time $O(|V| \cdot |E|^2)$

→ Very active line of research...

↳ Current fastest approximate maxflow alg runs in time $\tilde{O}(n \cdot \log^{1/2} n \cdot \log\log^b n \cdot \log(1/\varepsilon))$ if $n = |V|$.

↳ Faster than sorting.

↳ Current fastest exact $O(m^{1+o(1)} \cdot \log^c n \cdot \log U)$

w/ $c = O(1)$, $m = |E|$.

~~Linear Programming~~

~~So what was the bigger~~

~~⇒ Using flow to solve other problems.~~

~~A key part about learning maxflow is knowing~~

~~how to reduce other problems to maxflow.~~

~~Problem: Bipartite matching.~~

~~INPUT: A bipartite graph $G = (A \cup B, E)$.~~

~~OUTPUT:~~

⇒ Maxflow reductions.

A key part about learning network flow is also seeing how to use it to solve other problems

Example : scheduling.

Suppose We're trying to figure out how to assign n TAs. to n office hour slots. ∴ ~~each~~

1. Every TA has a certain # of OH slots that they ~~can~~ are available for
2. Every OH slot needs to be filled by exactly 1 TA.
3. Every TA must be assigned to 1 OH slot.

Given a set of OH slots and TA preferences can you tell if its possible to schedule the TAs s.t. the above constraints hold ?.

⟶ Let's formalize this.

~~INPUT: n TAs $\{1 \ldots n\}$ and n OH slots.~~

~~$\{s_1 \ldots s_n$~~

INPUT: n TAs $A = \{1 \ldots n\}$, n OH slots

$B = \{s_1 \ldots s_n\}$ and a preference map.

$\sigma : A \to 2^B$ where if $i \in A$ is a TA. then

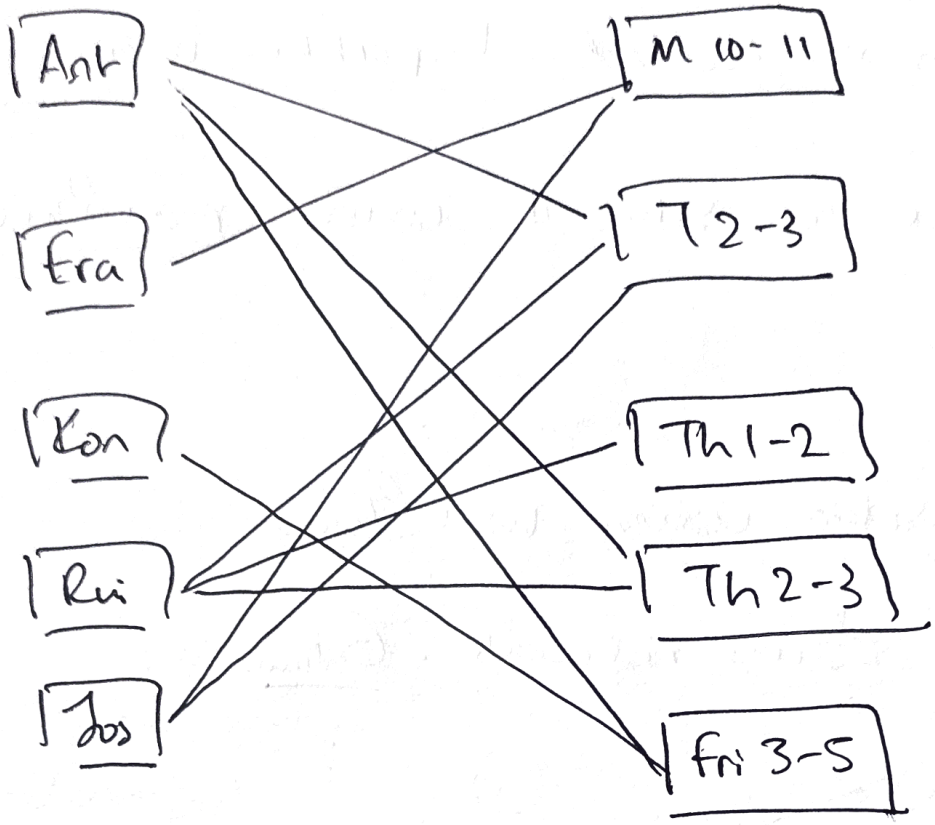
$\sigma(i) = \{$ set of OH slots i is available for $\}$.

OUTPUT: Yes if $\exists$ an assignment of TAs to OH s.t. each TA is paired w/ a unique OH and vice versa.

for example -- $A = \{$ Ant, Era, Yan, Rui, Jay $\}$

~~Ant~~ $B = \{$ M 10-11, Tue 2-3, Th 1-2, Th 2-3, Fri 3-5 $\}$

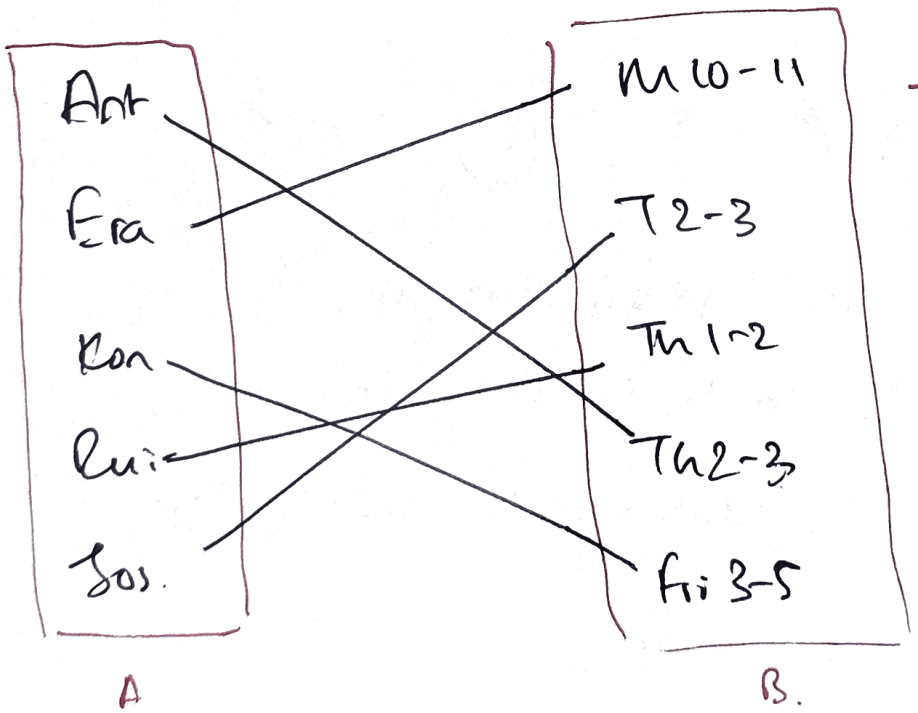
And now the preference map σ.



(*)

This is called a bipartite graph.

And as it would turn out this is what we come up w/



This is a perfect matching

~~This graph we drew (*) is called a bipart~~

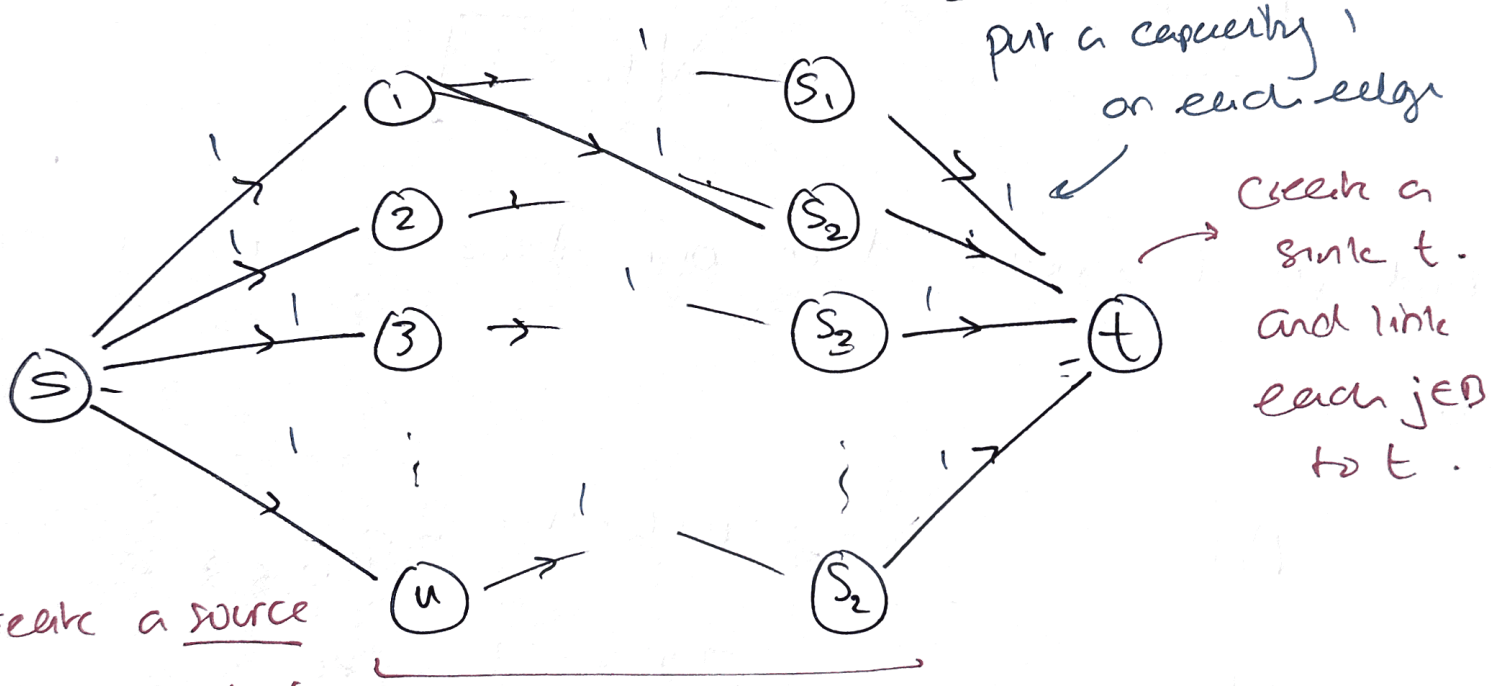
This problem is called bipartite matching.

→ And we can solve it using maxflow

→ Questions?

⇒ How to solve using maxflow.

① Construct a flow network. G_{flow}



(2) State which nodes to route flow b/w

↳ Rate a maxflow b/w s and t.

(3) State what to output for your problem depending on what maxflow returns

"If the maxflow ~~value~~ size = n then return Yes. o.w. return NO."

And that's our algorithm.

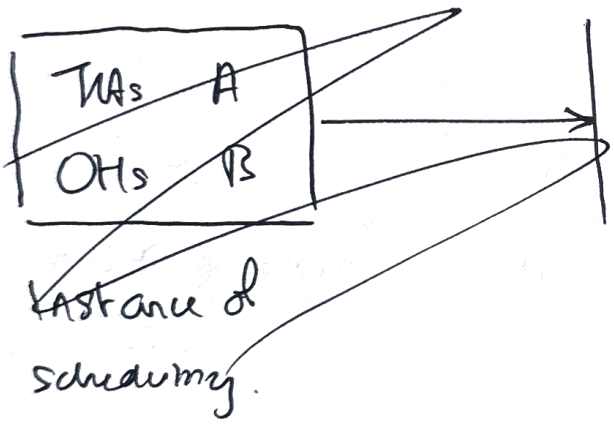
→ Questions?

So... what did it mean to "solve this problem using maxflow"?

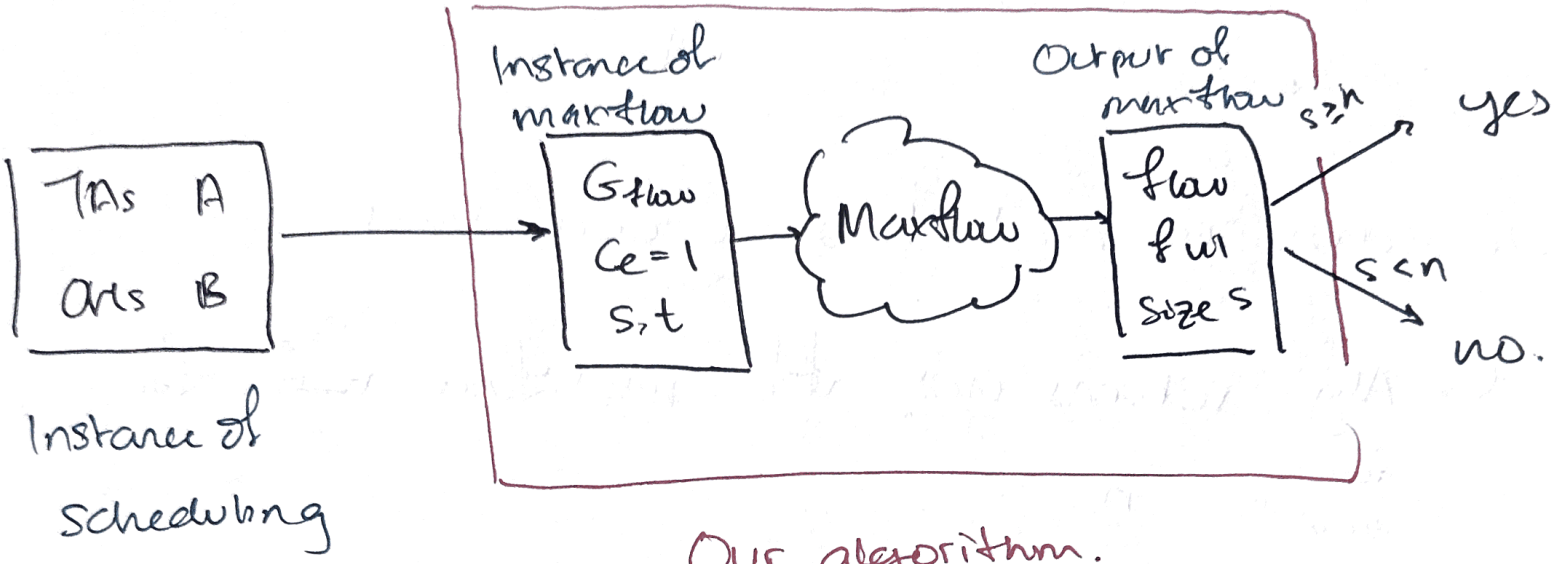
→ We took an instance of our original problem (scheduling) i.e. all the inputs to scheduling.

- We created an instance for workflow.
  i.e. a set of inputs for workflow.
- We ran workflow. on our new instance.
- Base on the output, we returned a solution for scheduling

Pictorially.



Pictorially.



→ This is called a "reduction to maxflow"

↳ why? If we have a program to solve maxflow → we immediately have a program to solve Scheduling.

↳ "maxflow is at least as hard as Scheduling"

↳ This is how we reason about hardness of problems when we don't have an algo to solve them.

→ In this case we can definitely solve maxflow

maxflow in polytime so we're good.

→ Questions?

Of course we need to show correctness.

↳ Alg returns yes iff maxflow ~~value~~ size for G_{flow} is $= n$.

→ we need to show. maxflow ~~value~~ size for $G_{flow} = n$ iff. ∃ a perfect matching / scheduling

↳ Key point: if all capacities are integer then maxflow value is integer.

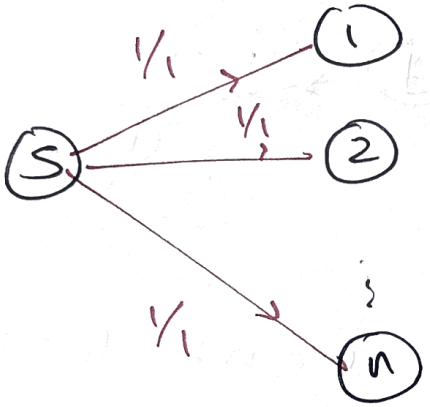
~~WTS~~: f^* is maxflow and size$(f^*) = n$.

Pf: "⟹" Suppose ~~Size of maxflow = n~~.

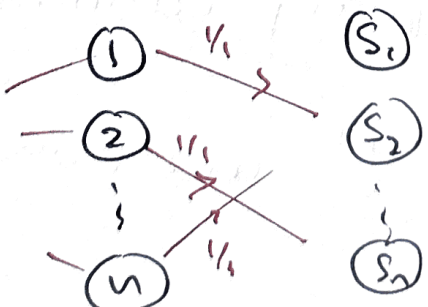
WTS: every TA is assigned and assignment is unique.

→ Every TA is assigned 1 OH.

- Note that $size(f^*) = n$
  - ↳ flow leaving $s = n$.
  - ↳ By flow construction, every $i \in A$ has one unit of flow incoming

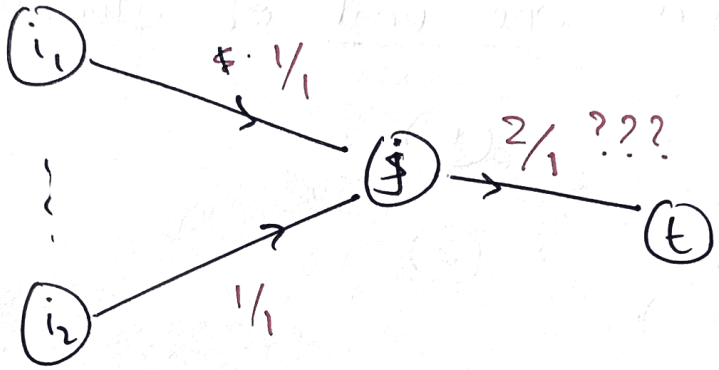


- Maxflow is integral: → for each $i \in A$, it must send all of its 1 unit of incoming flow along a single edge.
- Because all edges go from $A \to B$, all $i \in A$ are matched



→ The assignment is unique i.e. no 2 TAs can match to the same OH.

- Suppose not. then $\exists$ $i_1, i_2 \in A$ matched to $j \in B$.



- But by defn that means $i_1 \rightarrow$ sends 1 unit of flow to S, i_2 sends 1 unit of flow to S.
  - ↳ Incoming flow to $S = 2$.
- Outgoing flow $= 2$ by flow conservation
- Violating capacity constraint on (j, t)
- Contradicting f^* as max flow. □.

→ Questions?

Other problems that you can solve w/ maxflow?

→ Global mincut.

→ Vertex capacitated maxflow.

Bigger picture

→ Maxflow is an <u>linear program</u>.

→ LPs are super ubiquitous. (P-Complete).

Some things you should know how to do w/ an LP.

→ Write an LP which models a combinatorial problem.

→ Take an LP dual.

→ Interpret the dual.

→ Compute value of a zero-sum game.