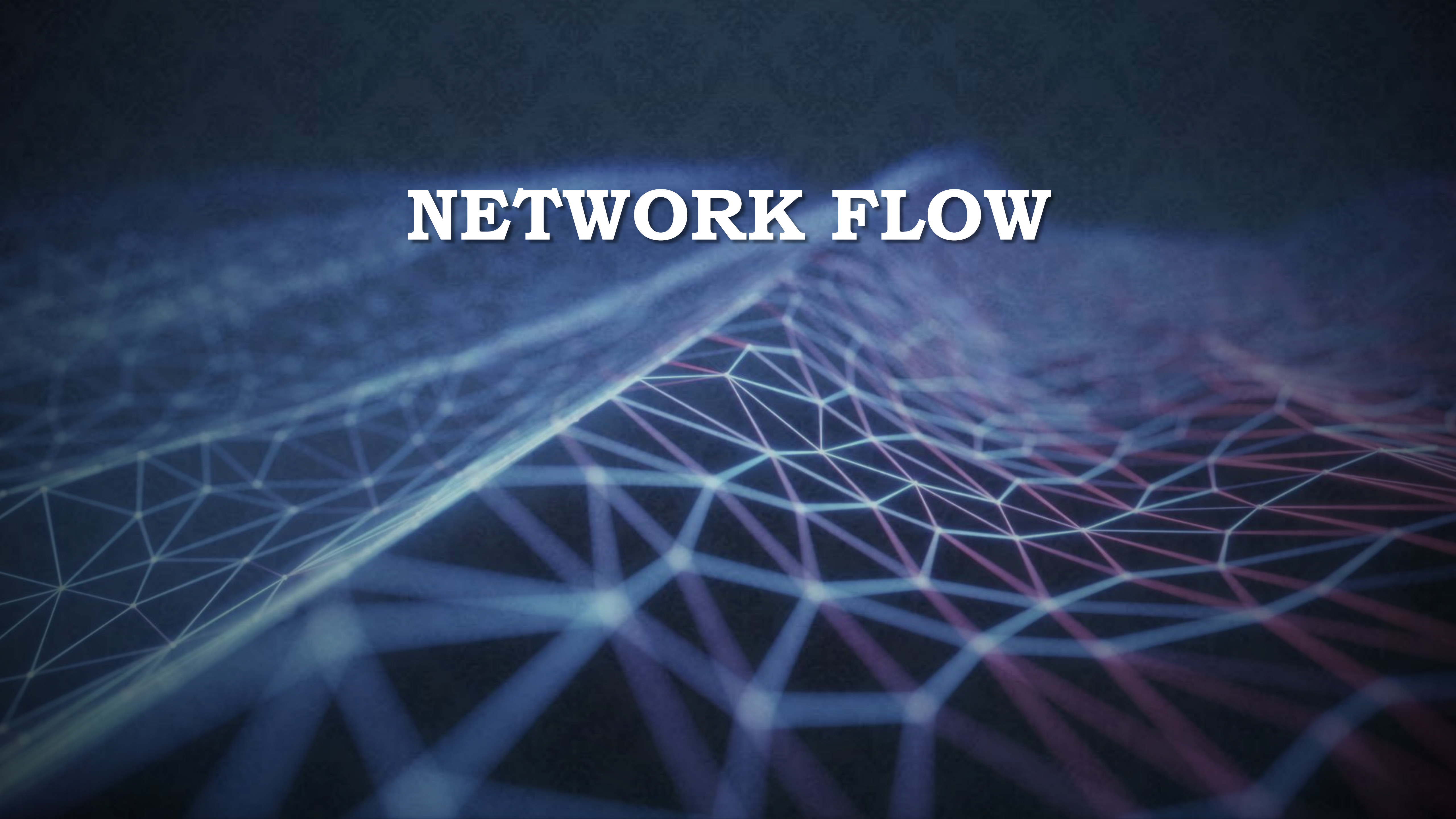
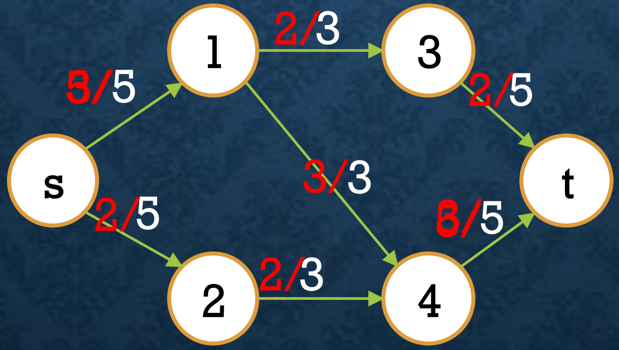
NETWORK FLOW



NETWORK FLOW

In the **Network Flow** problem, you are given a weighted, directed graph G = ⟨V,E⟩, with source node s and sink node t, where the "weights" of edges indicate the capacity of the edge.

- These values measure how much data can flow through the edge per time unit.
- We want to determine the maximum rate that data can be pushed from s to t in G.



Can you find a better flow?

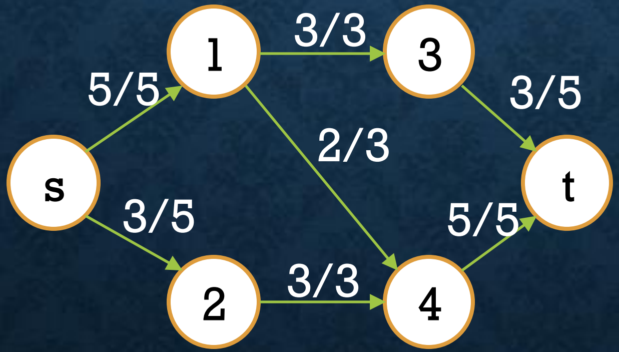
- You can achieve a flow of 8.

FORMAL DEFINITION

Each edge e has a capacity c(e). An **s-t flow** is a function f which satisfies:

- $0 \leq f(e) \leq c(e)$, for all e
- $\sum_{\langle x,v\rangle \in E} f(\langle x, v\rangle) = \sum_{\langle v,y\rangle \in E} f(\langle v, y\rangle)$, for all nodes $v \in V - \{s,t\}$

The value of a flow is $v(f) = \sum_{\langle s,v\rangle \in E} f(\langle s, v\rangle)$



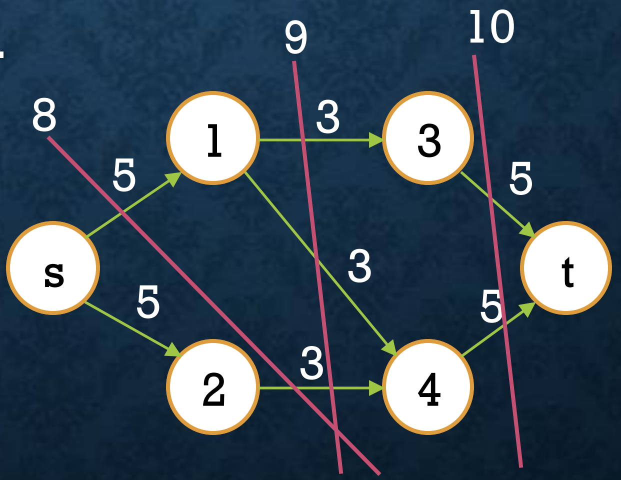
MINIMUM CUT

Given a weighted, directed graph $G = \langle V, E \rangle$, with source node s and sink node t, an **s-t cut** is a partition of the nodes $\langle P, V - P \rangle$ where $s \in P$ and $t \in V - P$.

- The **cutset** is the set of edges whose origin is in P and destination is in V – P.
- The value of an s-t cut is the sum of the weights of the edges in the cutset.

Your goal is to find the min-cost s-t cut.

Anyone think the value of the min-cut is a coincidence?



WEAK DUALITY

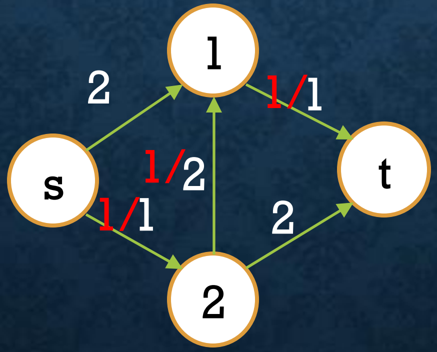
Given a flow f and a cut ⟨P, V – P⟩, v(f) ≤ v(⟨P, V – P⟩)

- **Corollary to Weak Duality**: If v(f) = v(⟨P, V – P⟩), then f is a max-flow, and ⟨P, V – P⟩ is a min-cut.

Is the converse true? Is there always a flow and cut with equal values?

- As usual, we'll arrive at the answer by trying to construct an algorithm for the problem.

We'll start with a basic greedy algorithm: choose an arbitrary s-t path, and route as much flow as possible along this path.



RESIDUAL GRAPHS

Given a graph G, and the flow so far f, the residual graph G_f indicates which edges can still support more flow, and how much.

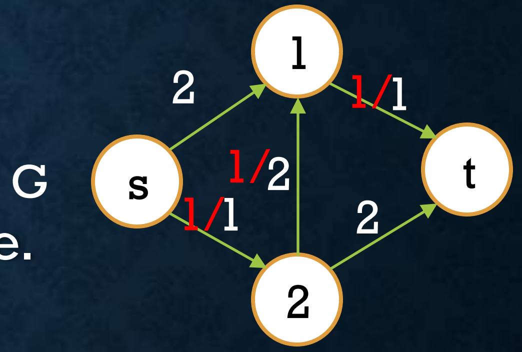
- Forwards edges: $c_f(u,v) = c(u,v) – f(u,v)$

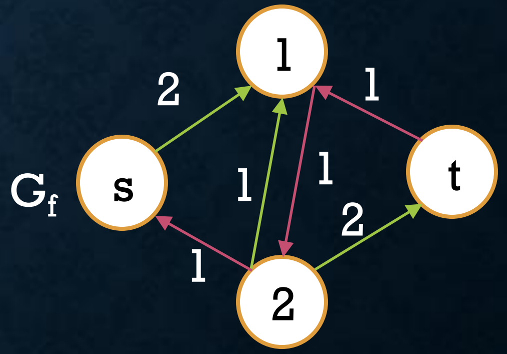
Forwards edges indicate you can still augment the flow along that edge.

- Backwards edges: $c_f(v,u) = f(u,v)$

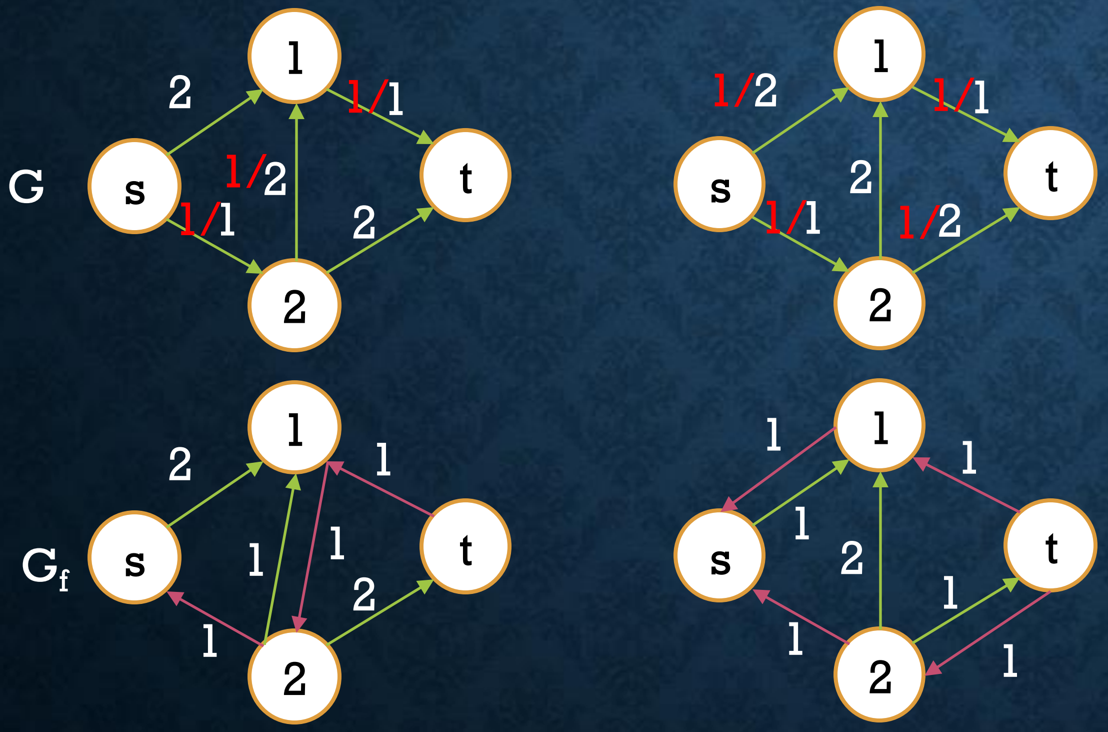
Backwards edges indicate that you can "take back" your choice to push flow along that edge and find a better solution.

- Will these backwards edges actually find valid paths?





USING THE RESIDUAL GRAPH



- When we transition from forwards edge to backwards edge, we are redirecting where the flow came from
- When we transition from backwards edge to forwards edge, we are redirecting where the flow goes.
- When we transition from backwards edge to backwards edge, we are removing both outgoing and incoming flow.
- We always maintain conservation of flow!

FORD-FULKERSON

The **Ford-Fulkerson Algorithm** looks for an **augmenting path** on the residual graph, pushes the maximum possible flow along that path, and repeats until there are no more s-t paths.

- The **Augmenting Path Theorem** states that f is a max flow iff there are no s-t paths in G_f.
- The **Max-flow Min-cut Theorem** states that the value of the max flow always equals the value of the min cut.

We will prove these theorems by showing that, given G and f, the following three statements are equivalent:

1. There is a cut $\langle P, V - P \rangle$ with value equal to v(f)
2. f is a max flow.
3. There is no s-t path in G_f.

THE PROOF

1. There is a cut $\langle P, V - P \rangle$ with value equal to $v(f)$
2. f is a max flow.
3. There is no s-t path in G_f.

$1 \rightarrow 2$: Already proven (the Corollary to Weak Duality).

$2 \rightarrow 3$: Proof by contraposition. If there is an s-t path in G_f, then we can augment the flow, which means f is not a max flow.

$3 \rightarrow 1$: Let f be a flow with no augmenting path.

- Let P be the set of vertices reachable in G_f from s.
- Note that $s \in P$ and $t \in V - P$.
- All edges from P to $V - P$ in G are full to capacity (otherwise we chose P incorrectly)
- All edges from $V - P$ to P in G are empty (otherwise there would be a backwards edge from P to $V - P$ in G_f).
- All flow approaches the divide, crosses, and never crosses back.
- Therefore, the value of the cut equals the value of the flow.

MINIMUM CUT: THE ALGORITHM

1. Run Ford-Fulkerson on G to get f.
2. Create the residual graph G_f.
3. Let P = the set of nodes reachable from s in G_f.
4. Return $\langle P, V - P \rangle$

Therefore, the runtime for Ford-Fulkerson is the same as the runtime to solve Min-Cut.

- The **Integrality Theorem** states that, if all capacities are integers, then there is a max flow f where every value f(e) is an integer

This follows trivially from the fact that Ford-Fulkerson will always find integer flows if all capacities are integers.

RUNTIME ANALYSIS

Assume that all edge capacities are integers between 1 and C.

Each time we find an augmenting path, what is the minimum amount we will increase the flow by?

- 1

How large might the max flow conceivably be?

- $(n-1) \cdot C$

How long does it take to find an augmenting path?

- $\Theta(m)$, can just use DFS. We presume the graph is connected, so m dominates n.

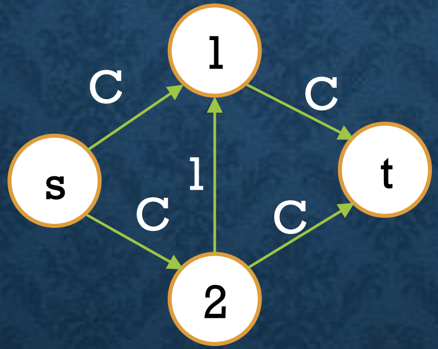
What is the runtime of Ford-Fulkerson?

- $\Theta(mnC)$

Is this a polynomial runtime?

- Nope, the C is the problem.

THE WORST-CASE GRAPH



IMPROVING FORD-FULKERSON

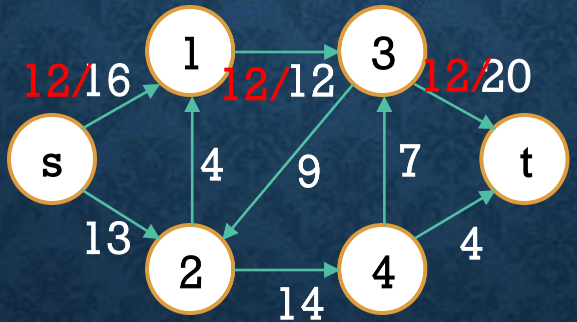
Ford-Fulkerson takes pseudo-polynomial time. The problem is that it takes any old augmenting path. What might be a better choice of path?

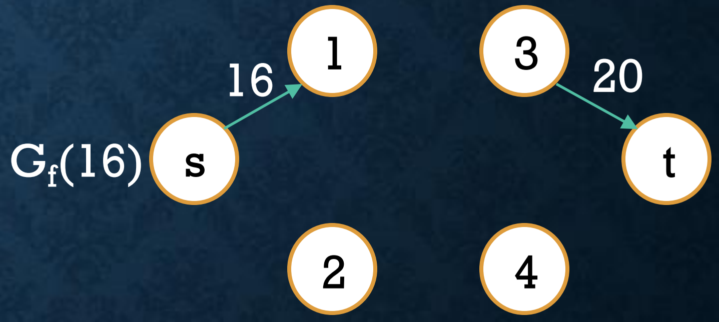
- Choose the path of maximum capacity
- Choose the path with fewest edges
- Choose a path of sufficiently large capacity.

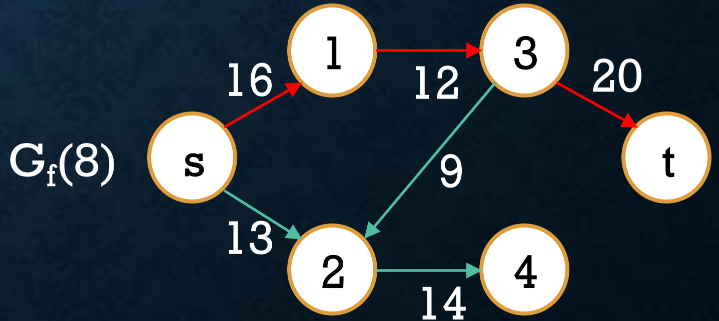
Each of these ideas can be used to improve Ford-Fulkerson. We will specifically be using the third one.

CAPACITY-SCALING

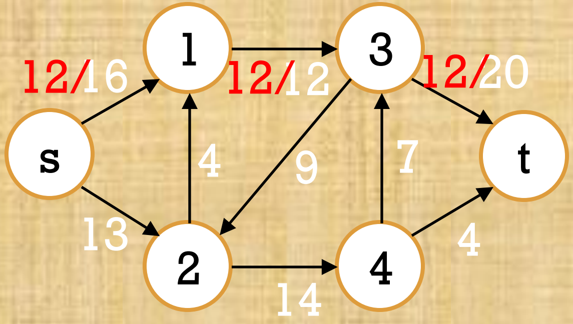
Let $G_f(\Delta)$ be the subgraph of G_f consisting of exactly the edges with capacity $\geq \Delta$.



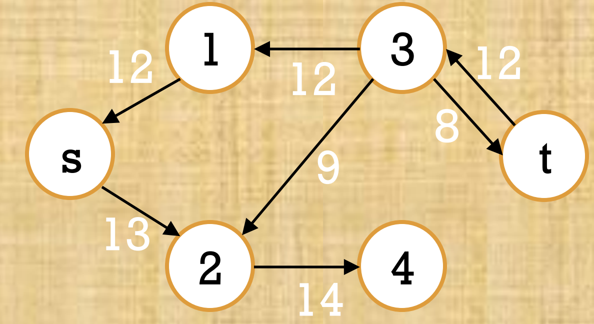




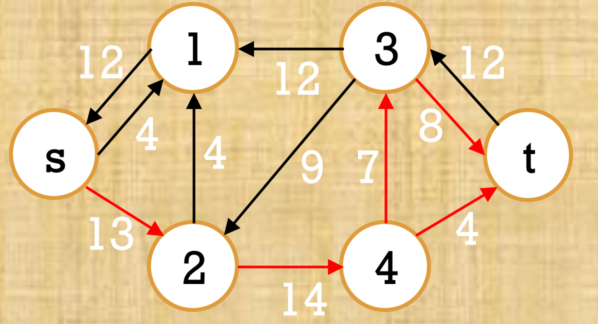
CAPACITY-SCALING: AN EXAMPLE

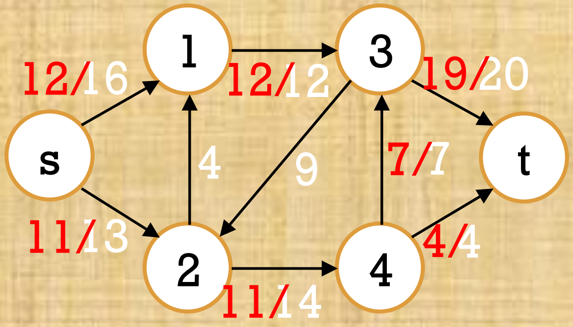


$G_f(8)$

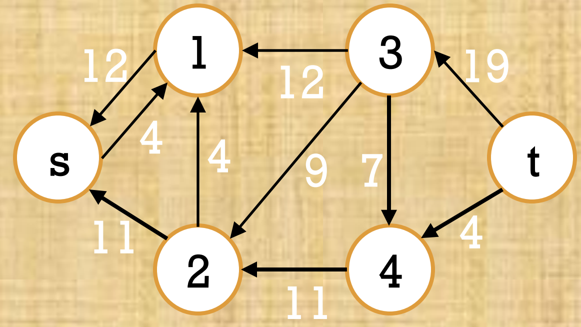


$G_f(4)$

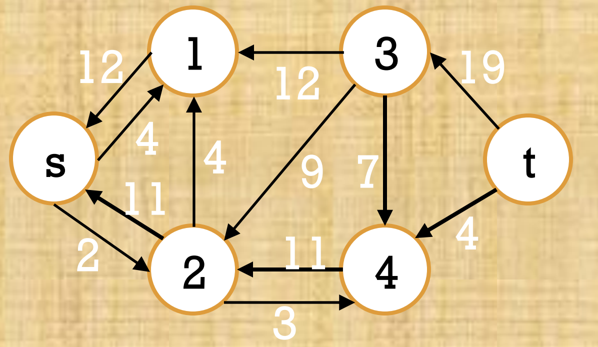




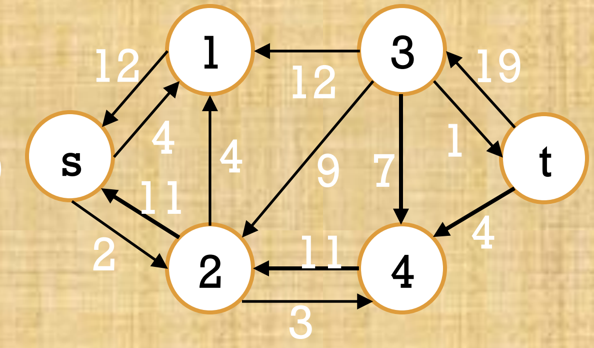
$G_f(4)$



$G_f(2)$



$G_f(1)$



FORD-FULKERSON WITH CAPACITY-SCALING

1. Let $\Delta = 2^i$, for max i such that $\Delta \leq C$
2. While $\Delta \geq 1$
    3. Augment the current max flow using Ford-Fulkerson on $G_f(\Delta)$
    4. $\Delta = \frac{\Delta}{2}$

We'll refer to each iteration of the loop as a phase. We need to figure out:

- The number of phases = log C
- How long it takes to find a path = m + n (simplified to m)
- How many paths we might find per phase = ???

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

Let P be the set of nodes reachable from s in $G_f(\Delta)$

G_f still has some edges with capacity $< \Delta$

- Consider an arbitrary edge from P to V – P. It has capacity $< \Delta$
- In the worst case, all m edges (or a constant fraction of them) span this divide.
- Therefore, we might be able to push up to $m\Delta$ more flow.

In phase-$\frac{\Delta}{2}$, we will push at least $\frac{\Delta}{2}$ for each path.

- Therefore, we will have no more than 2m paths in this phase.

PATHS PER PHASE

NETWORK FLOW ALGORITHMS

- Ford-Fulkerson with Capacity-Scaling = $\Theta(m^2 \log C)$
- Preflow-Push = $\Theta(n^3)$
- Goldberg-Rao = $\Theta(\min(n^{\frac{2}{3}}, m^{\frac{1}{2}})\, m \log n \log C)$

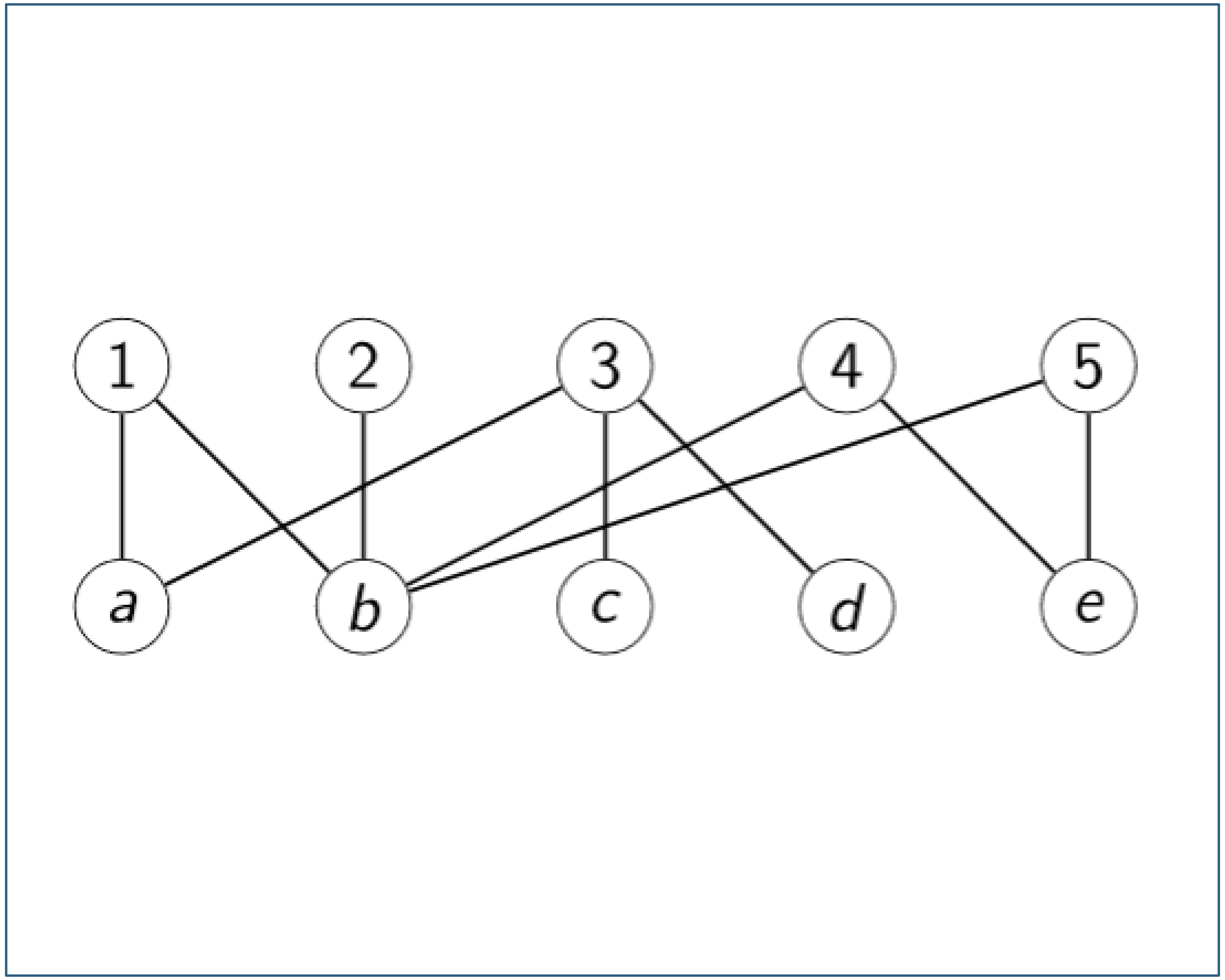
If we improve Network-Flow, we simultaneously improve Minimum-Cut.

- We wrote Minimum-Cut in such a way that it called Max-Flow as a subroutine. This is known as a **reduction**. We **reduced** Min-Cut to Max-Flow.
- We write this as Min-Cut $\leq$ Max-Flow

POLYNOMIAL-TIME REDUCTIONS

A poly-time reduction is one where all of the work **except** the subroutine call takes no more than polynomial-time.

- We write this as Min-Cut $\leq_P$ Max-Flow
- P is the set of poly-time solvable problems.
- If A $\leq_P$ B, and B $\in$ P, then A $\in$ P
- If A $\leq_P$ B, and A $\in$ P, then…?
    - Nothing. We can't infer B is also poly-time solvable.
- We will be doing many reductions to Max-Flow and Min-Cut in this unit.



BIPARTITE MATCHING

You are given an undirected bipartite graph G = $\langle V_1 \cup V_2, E \rangle$.

- You want to find a max-sized matching M $\subseteq$ E s.t. no endpoints are repeated.

What is the largest matching in this graph?

- 4. { {1,a}, {2,b}, {3,c}, {5,e} }

Why isn't there a larger matching?

- The nodes {c,d} can't both be matched.

LAZY PROGRAMMING

We could try to come up with an original algorithm to solve Bipartite Matching, but that sounds like too much work.

- Let's be **lazy** and use an existing algorithm.

If we can find a poly-time reduction from Bipartite Matching to Max Flow, we'll have an algorithm for our new problem!

- We need to show how to transform an arbitrary instance of Bipartite Matching into a Max Flow instance (which will be passed into our Max Flow solver).
- Then we need to show how to extract a Bipartite Matching answer from our Max Flow answer. Both of these steps must take polynomial time.

REDUCTION

What components are missing in this graph, which are needed in a Max Flow instance?

What else do we need to add, so that there is a useful Max Flow solution?

How do we extract the Bipartite Matching answer from the Max Flow solution?

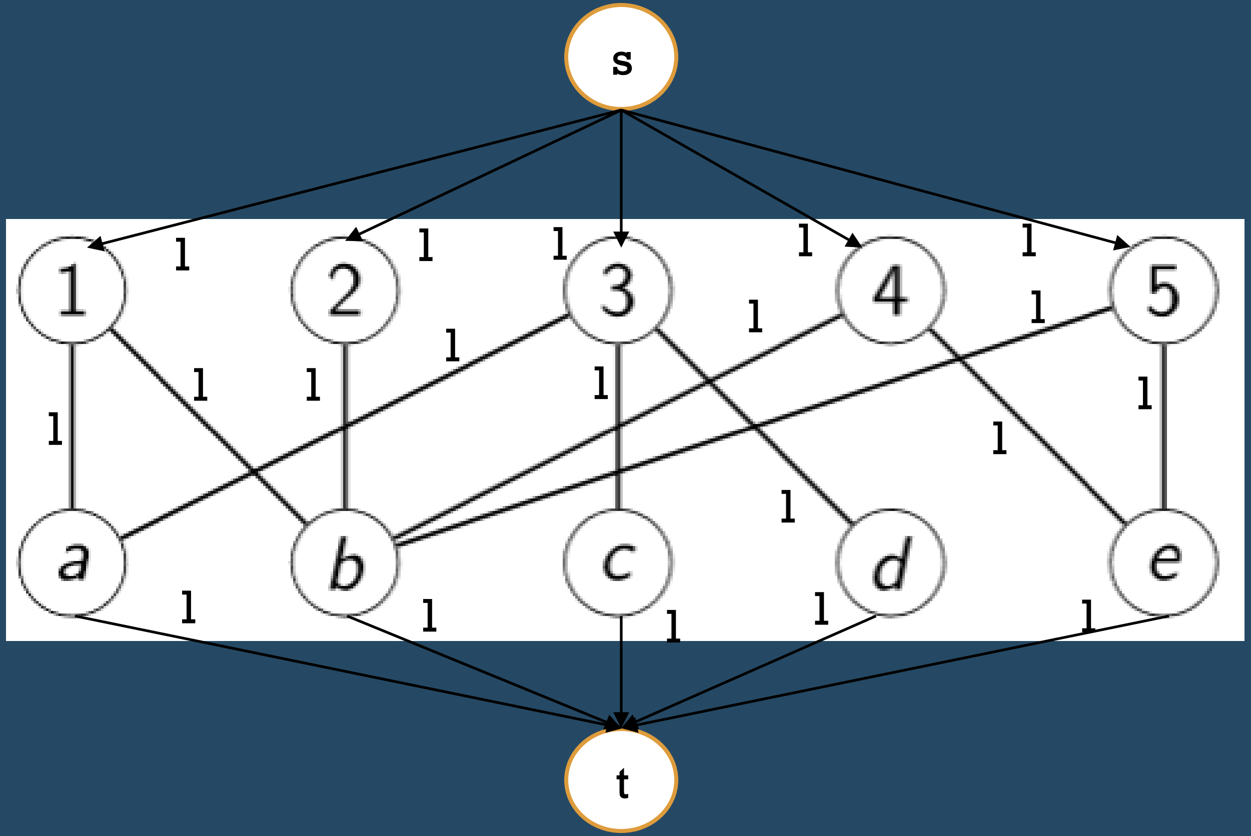
- Choose the original edges that have a unit of flow along them

If there is a Max Flow of x, then those edges would form a valid Bipartite Matching, since we ensure only one unit of flow goes to or from any node.

If there is a Bipartite Matching of x, then those edges would form a Max Flow, since we can route flow to the V_1 nodes and from the V_2 nodes.

We added a linear number of edges and nodes, so this is a poly-time reduction.

Bipartite Matching $\leq_P$ Network Flow



DESIGN TEMPLATE

1. Identify what components need to be added to the graph to make it a Max Flow instance.
2. Identify what needs to be added to the graph to make the Max Flow solution useful.
3. Explain how to extract your solution from the Max Flow solution.
4. Explain why the reduction takes polynomial-time.
5. Explain why a size-X solution to Max Flow implies a size-X solution to your problem.
6. Explain why a size-X solution to your problem implies a size-X solution to Max Flow.

CIRCULATIONS

You are given a directed graph G, with edge capacities c(e) for all edges e.

- Each node has an integer **demand** d(v). If d(v) < 0, we say it has a **supply**.

A circulation is a function f which satisfies:

- $0 \le f(e) \le c(e)$, for all edges e
- $\sum_{\langle x,v\rangle} f(\langle x, v\rangle) - \sum_{\langle v,y\rangle} f(\langle v, y\rangle) = d(v)$, for all nodes v

You want to determine if there is a valid circulation.

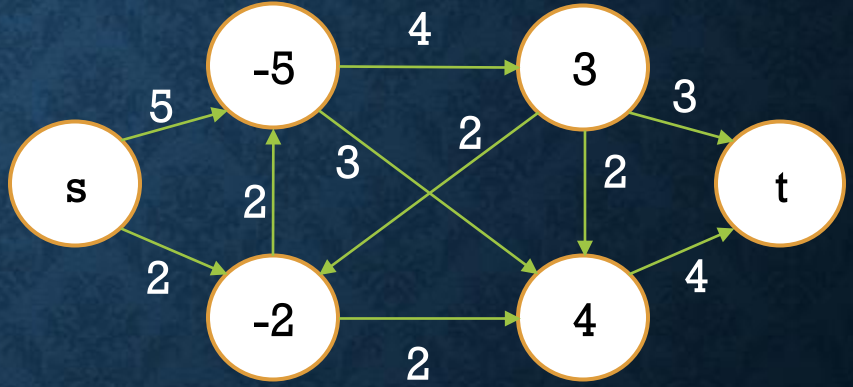
- Note that all supply must be used, and all demand must be met, so if the sum of all d(v) values is not 0, there is no solution.

REDUCTION

What needs to be added to the graph?

What needs to be included to produce a useful solution for circulations?

To extract the solution, simply look at the flow on the original edges.



If there is a max flow equal to the sum of the supplies and the sum of the demands, then this is a valid circulation, since all demands are handled and all supplies are used.

If there is a valid circulation, then there would be a max flow equal to this sum.

We added a linear number of edges and nodes, so this is a poly-time reduction.

Circulations $\leq_P$ Network Flow

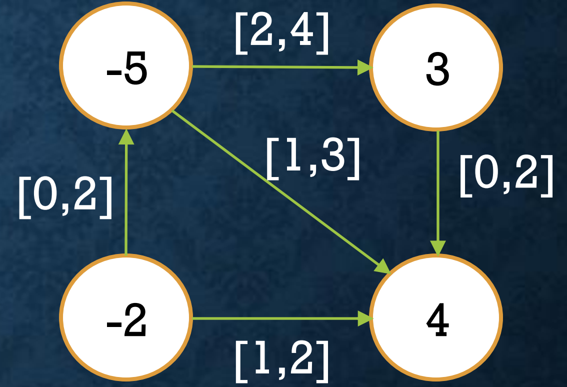
CIRCULATIONS WITH LOWER BOUNDS

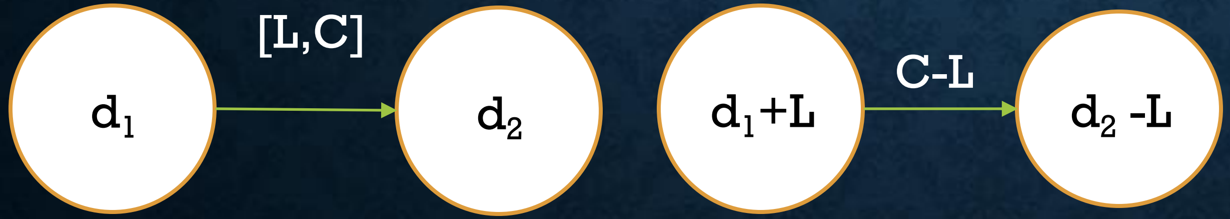
Now edges can have lower bounds $l(e)$, and the capacity rule is $l(e) \leq f(e) \leq c(e)$

What problem should we reduce to?

- Circulations

How should we get rid of the lower bounds?





PRODUCING THE ANSWER

To extract the answer, add the lower-bound of each edge to the flow on that edge.

If there is a valid circulation, then for each edge adding the lower-bound will produce a valid circulation with lower bounds.

If there is a valid circulation with lower-bounds, then subtracting the lower bound will produce a valid circulation on the derived graph.

We do a constant amount of work for each edge, so this is a poly-time reduction.

Circulations with Lower Bounds $\leq_P$ Circulations

REDUCTIONS

In theory, you could give a poly-time reduction to any poly-time solvable problem.

However, in this unit we are expecting you to reduce to one of these 5 problems:

1. Max Flow
2. Min Cut
3. Bipartite Matching
4. Circulations
5. Circulations with Lower Bounds

SURVEY DESIGN

You have n customers $\{c_1, \ldots, c_n\}$ and m products
$P = \{p_1, \ldots, p_n\}$

- c_i owns a subset of our products $S_i \subseteq P$

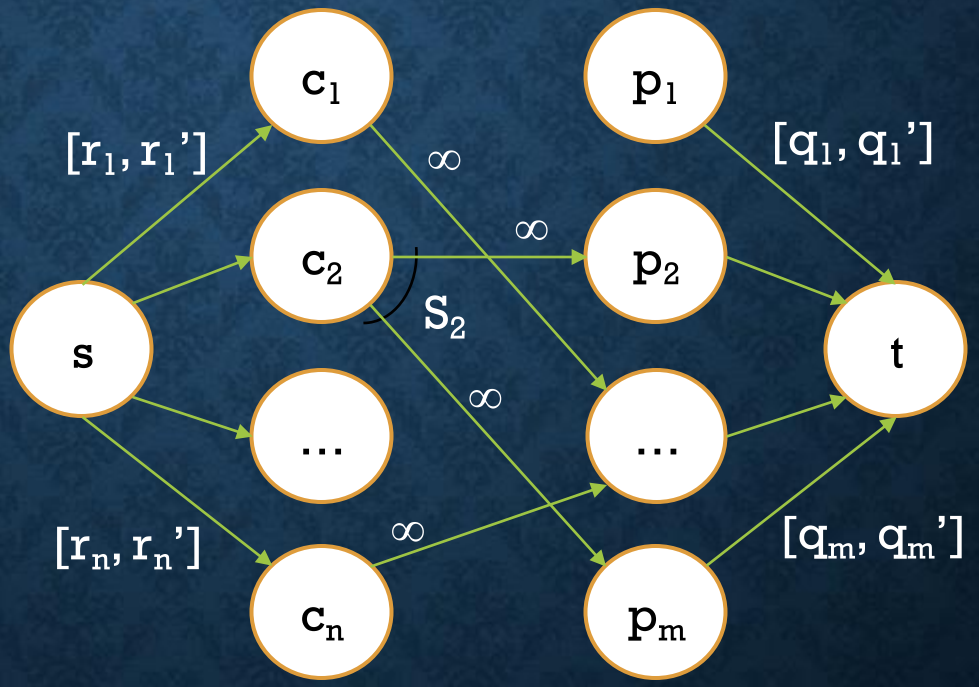
We want to design a survey that gets a sufficient amount of feedback about our products.

- We must ask between q_i and q_i' questions about p_i (too many questions would be redundant).
- We must ask between r_i and r_i' questions to c_i (too few questions is a waste of their time).
- We can only ask questions to a customer about products they own.
- We are not limited to one question per product per customer: there are lots of questions we can ask about the same product.

Design a survey according to the above constraints.

THE REDUCTION

Each of the 5 problems we can reduce to are graph problems. Often you will start constructing a graph before you have decided which problem to reduce to.



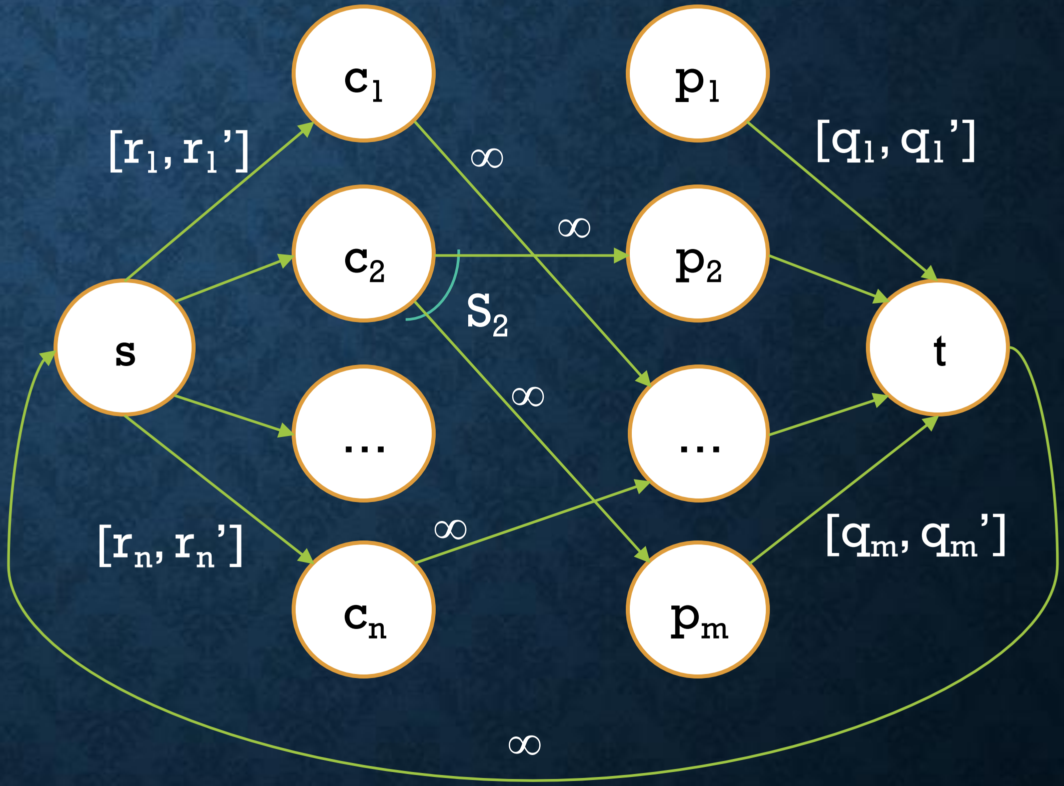
What problem are we reducing to?

WHICH PROBLEM TO REDUCE TO?

Since we have lower bounds, we have to be reducing to circulations with lower-bounds

- However, we're looking for the max flow from s to t, which is an entirely different problem.
- We need to alter the graph so that the demand and supply of each node is specified.
- Problem: we don't know what the supply of s should be!
- 2nd problem: we don't know what the demand of t should be!
- 1 problem is a problem.
- 2 problems… is an opportunity. Maybe there is a way to use one problem to solve the other problem?

THE FIX



c_1
c_2
…
c_n
p_1
p_2
…
p_m
s
t
$[r_1, r_1']$
$[r_n, r_n']$
$[q_1, q_1']$
$[q_m, q_m']$
∞
∞
∞
∞
∞
S_2

Check the edges between the customer and product nodes: the amount of flow indicates the number of questions to pose to that customer about that product.

If there is a valid survey, then that will constitute a valid circulation on our graph.

If we have a valid circulation on our graph, that would be a valid survey.

We added a linear number of edges and nodes, so this is a poly-time reduction.

Survey Design $\leq_P$ Circulations with Lower Bounds

- This trick of adding an infinite-capacity edge from t to s works whenever you accidentally turn the problem in to "Max Flow with Lower Bounds".

EXTRACTION

BASEBALL ELIMINATION

We want to determine if a team can end up with the most wins at the end of the season (or tied for most wins).

Team	Wins	LA	Col	SF	Ari	SD
LA	90	0	0	0	7	4
Col	88	0	0	0	0	1
SF	87	0	0	0	0	4
Ari	86	7	0	0	0	4
SD	75	4	1	4	4	0

Which teams are clearly eliminated?

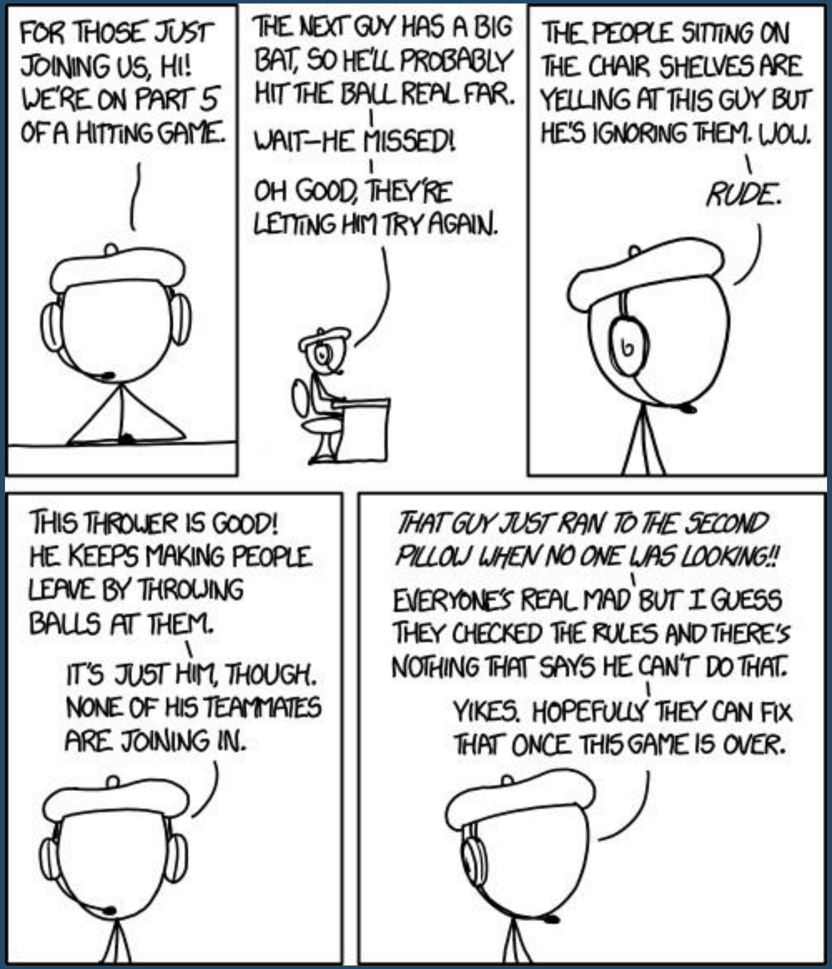
- Colorado, San Diego

There is another team that is mathematically eliminated.

- San Francisco. One of Los Angeles and Arizona will end up with 92 wins or more.

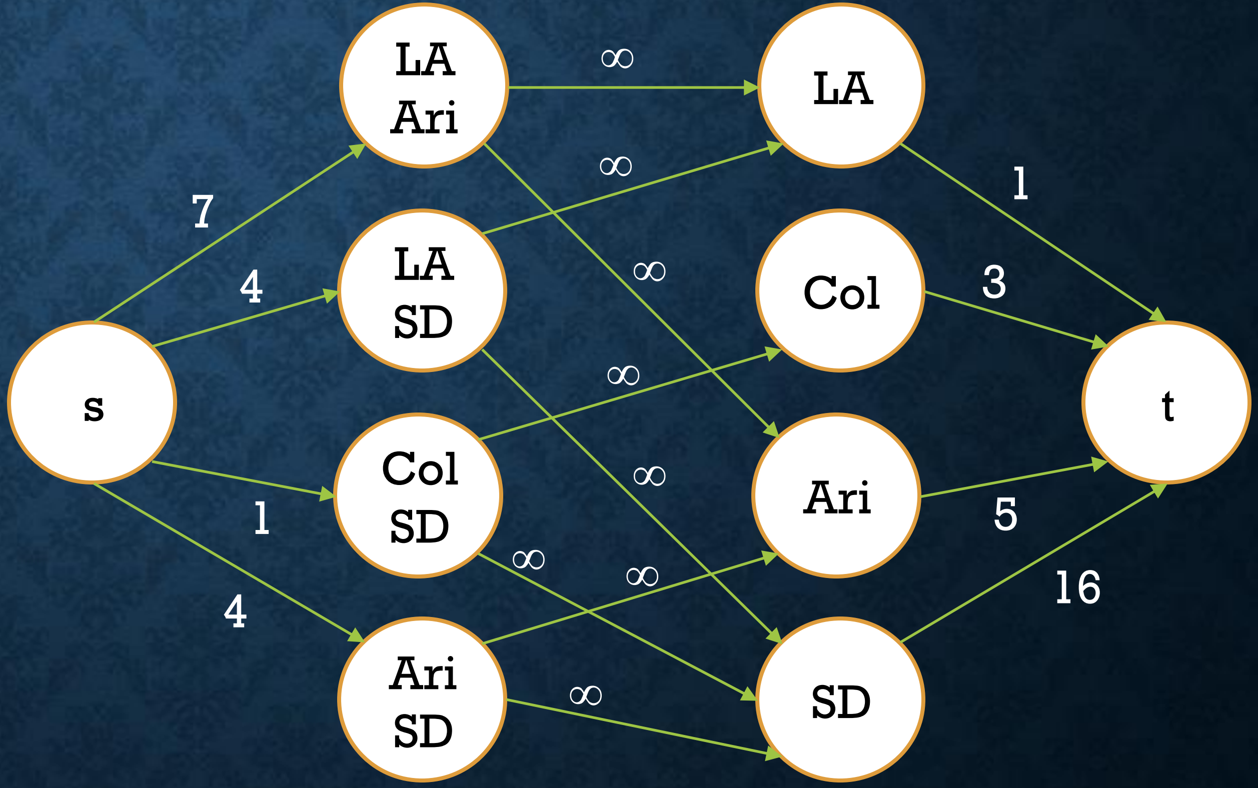
XKCD #1593

The thrower started hitting the bats too much, so the king of the game told him to leave and brought out another thrower from thrower jail.


FOR THOSE JUST JOINING US, HI! WE'RE ON PART 5 OF A HITTING GAME.
THE NEXT GUY HAS A BIG BAT, SO HE'LL PROBABLY HIT THE BALL REAL FAR.
WAIT—HE MISSED!
OH GOOD, THEY'RE LETTING HIM TRY AGAIN.
THE PEOPLE SITTING ON THE CHAIR SHELVES ARE YELLING AT THIS GUY BUT HE'S IGNORING THEM. WOW.
RUDE.
THIS THROWER IS GOOD! HE KEEPS MAKING PEOPLE LEAVE BY THROWING BALLS AT THEM.
IT'S JUST HIM, THOUGH. NONE OF HIS TEAMMATES ARE JOINING IN.
THAT GUY JUST RAN TO THE SECOND PILLOW WHEN NO ONE WAS LOOKING!!
EVERYONE'S REAL MAD BUT I GUESS THEY CHECKED THE RULES AND THERE'S NOTHING THAT SAYS HE CAN'T DO THAT.
YIKES. HOPEFULLY THEY CAN FIX THAT ONCE THIS GAME IS OVER.

THE REDUCTION

We want to design a poly-time algorithm that determines whether a specific team has been mathematically eliminated (in this case, San Francisco).



EXTRACTION

If the Max Flow equals the number of remaining games, then San Francisco is still in the running. Otherwise, they are mathematically eliminated.

- If there is a scenario leading to SF making the playoffs, that distribution of games will result in a Max Flow of the number of games.
- If there is a Max Flow equal to the number of remaining games, the flow on the matchup edges indicate who needs to win which games for SF to still be in it.

We added a quadratic number of nodes and edges, so this is a polynomial-time reduction.

Baseball Elimination $\leq_P$ Network Flow

PROJECT SELECTION

You are given a set of projects $P = \{p_1, \dots, p_n\}$, where p_i has value v_i (possibly negative), and has prerequisites $S_i \subseteq P$.

- If you decide to do p_i, you must also do all the projects in S_i (there is no order in this problem, you do them or you don't do them).

You want to find the max-valued subset of projects to do.

What value do we get for doing all the projects?

- 0

What value do we get for doing none of the projects?

- 0

Can you do better?

- p_1, p_2, p_3, p_4 has value 1. This is the best possible.

Project	v_i	S_i
1	2	p_2
2	-3	p_4
3	5	p_2, p_4
4	-3	-
5	2	p_4, p_6
6	-3	-

STARTING THE REDUCTION

$$\begin{tikzcd}
p_1 \arrow[r] & p_2 \arrow[d] \\
p_3 \arrow[r] \arrow[ur] & p_4 \\
p_5 \arrow[r] \arrow[ur] & p_6
\end{tikzcd}$$

It's not clear where to go from here, so let's consider which problem we're reducing to.

Would Bipartite Matching work?

- No, it's not a bipartite graph.

How about Max Flow or Circulations?

- We can't force flow through the prerequisite edges.

How about Circulations with Lower Bounds?

- That forces us to do all the projects, rather than giving us a choice.

TRY MIN CUT!

Min Cut is the oddest problem in our set of problems to reduce to.

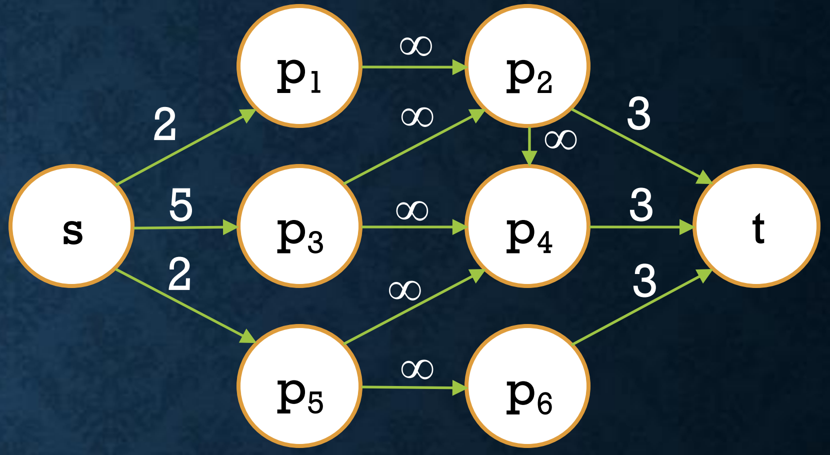
- It is less intuitive and harder to use
- When all else fails, try Min Cut!
- If you are trying to minimize something, try Min Cut!
- If you are trying to partition a set, try Min Cut!

All our other problems failed, and we are trying to partition the projects into the subset of projects we do, and the subset we don't.

- Let's try Min Cut!

We need to make sure we don't do a project without doing it's pre-requisite. How can we ensure this?

THE REDUCTION



Doing a project without doing its prerequisite should be impossible.

- Cutting an infinity-edge results in the pre-requisite being in the t-set, and the dependent project being in the s-set.
- Therefore, the s-set is the projects we do.

If we cut an edge from s to a project, we are not doing that project, and cutting an edge is bad.

- Therefore, add edges from s to the positive valued projects.

PROOF OF CORRECTNESS

For the project with negative value in S (the set we do), we increase the value of the cut by:

$$\sum_{i \in S, pi < 0} -pi$$

For every project with positive value in V – S, we increase the value of the cut by:

$$\sum_{i \in V-S, pi > 0} p_i$$

The value of the cut we find is therefore:

$$\sum_{i \in V-S, pi > 0} p_i - \sum_{i \in S, pi < 0} p_i$$

PROOF, PART 2

Let C be the sum of the capacities of the edges outgoing from the source:

$$C = \sum_{p_i > 0} p_i$$

$$\sum_{i \in V-S, pi>0} p_i = C - \sum_{i \in S, pi>0} p_i$$

So the value of the cut is:

$$C - \sum_{i \in S, pi>0} p_i - \sum_{i \in S, pi<0} p_i$$

$$= C - \sum_{i \in S} p_i$$

That is, if we maximize the value of the projects we do, we minimize the value of the cut. Proven!
Project Selection $\leq_P$ Min Cut

EXTRA PRACTICE

Chapter 7

Exercises 8, 11, 12, 14, 22, 24, 27, 29, 41