CS4102 Algorithms

Spring 2021 – Floryan and Horton

Module 4, Day 3: Recorded Lecture

Roadmap: Where We're Going and Why

- ***Reductions*** between problems
  - Why? Can be a practical way of solving a new problem
  - Also: A proof about one problem's complexity can be applied to another
  - Formal definition of a reduction
- Examples
  - Bipartite graphs, matching
  - Vertex cover and independent set

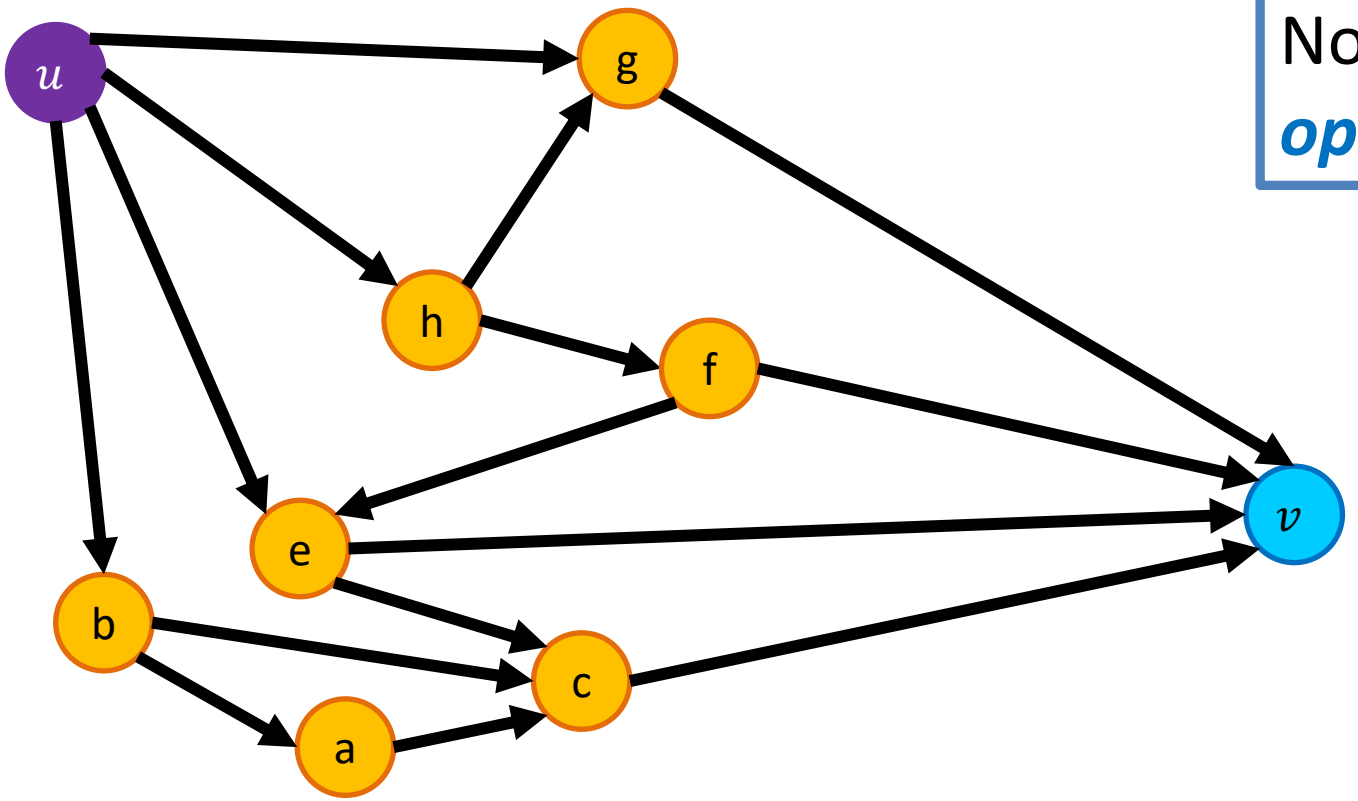
Using One Solution to Solve Something Else

- Sometimes we can solve a “new” problem using a solution to another problem
  - We need to “re-cast” the “new” problem as an ***instance*** of the other problem
  - We may need to relate how the answer found for the other problem gives the answer for the “new” problem
- Some examples coming in this lecture:
  - We’ll see how to solve ***edge-disjoint path*** problem.
    Use that to solve ***vertex-disjoint path*** problem.
  - We know how to find ***max network flow***.
    Use that to solve ***bi-partite matching***.

Edge-Disjoint Paths

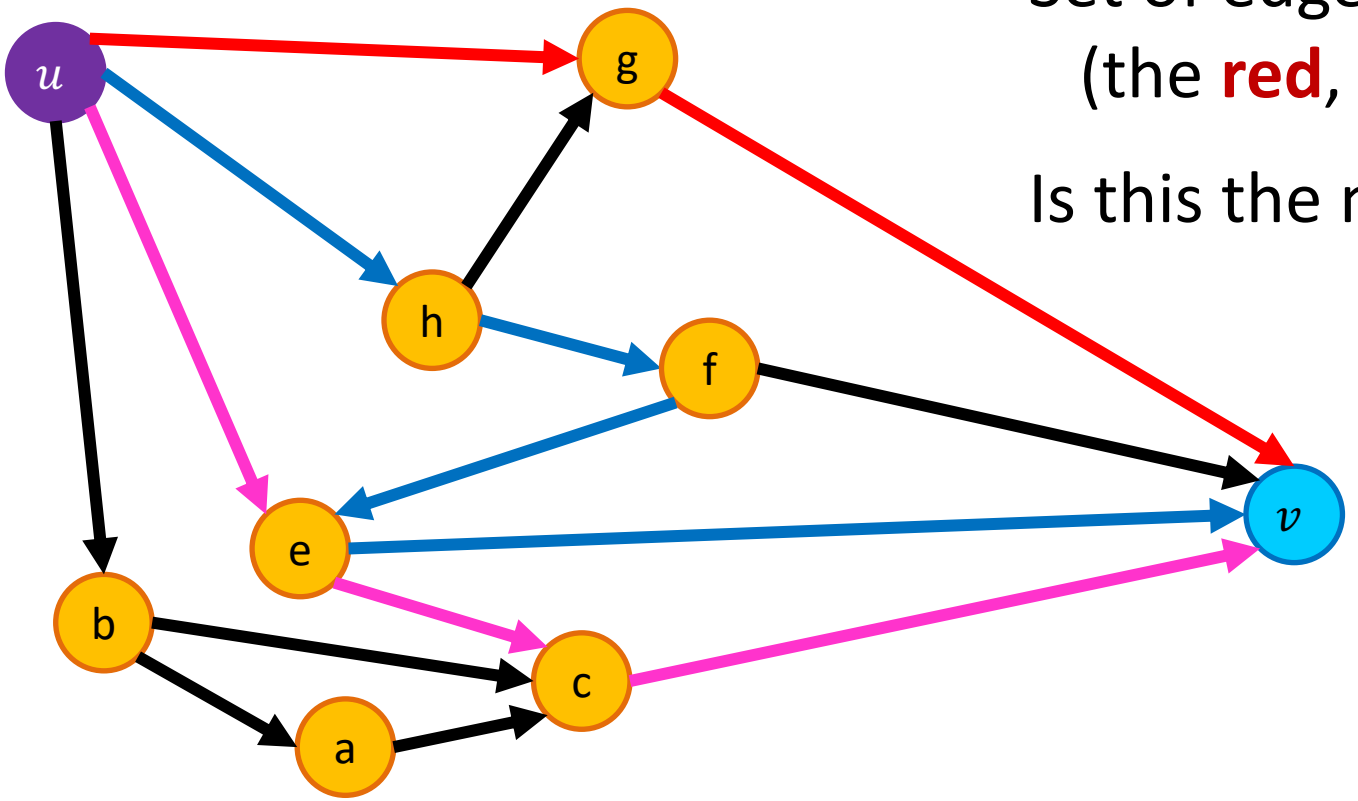
Given a graph $G = (V, E)$, a start node u and a destination node v, give the maximum number of paths from u to v which share no edges

Note this is an ***optimization problem.***



Edge-Disjoint Paths

Given a graph $G = (V, E)$, a start node u and a destination node v, give the maximum number of paths from u to v which share no edges

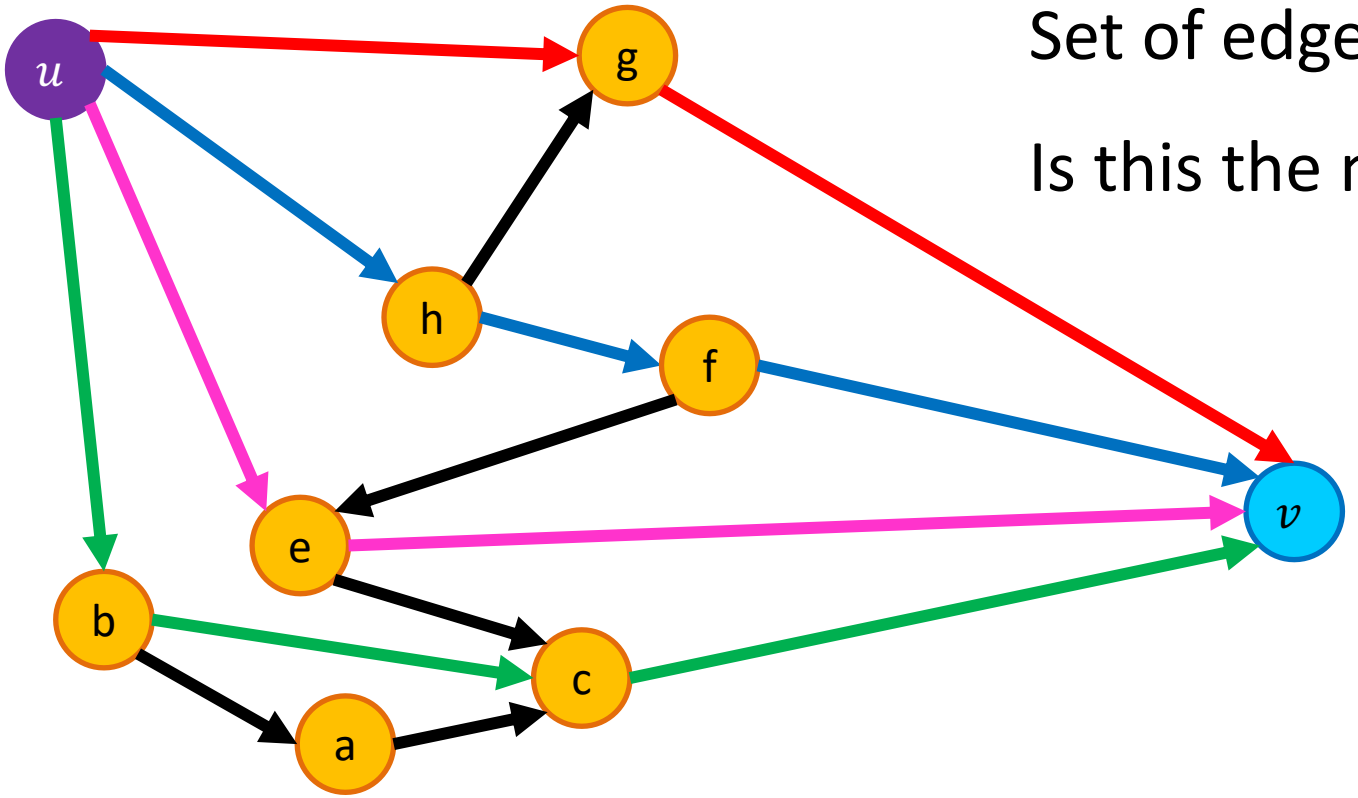


Set of edge-disjoint paths of size 3 (the **red**, **blue**, **magenta** paths)

Is this the max number?

Edge-Disjoint Paths

Given a graph $G = (V, E)$, a start node u and a destination node v, give the maximum number of paths from u to v which share no edges



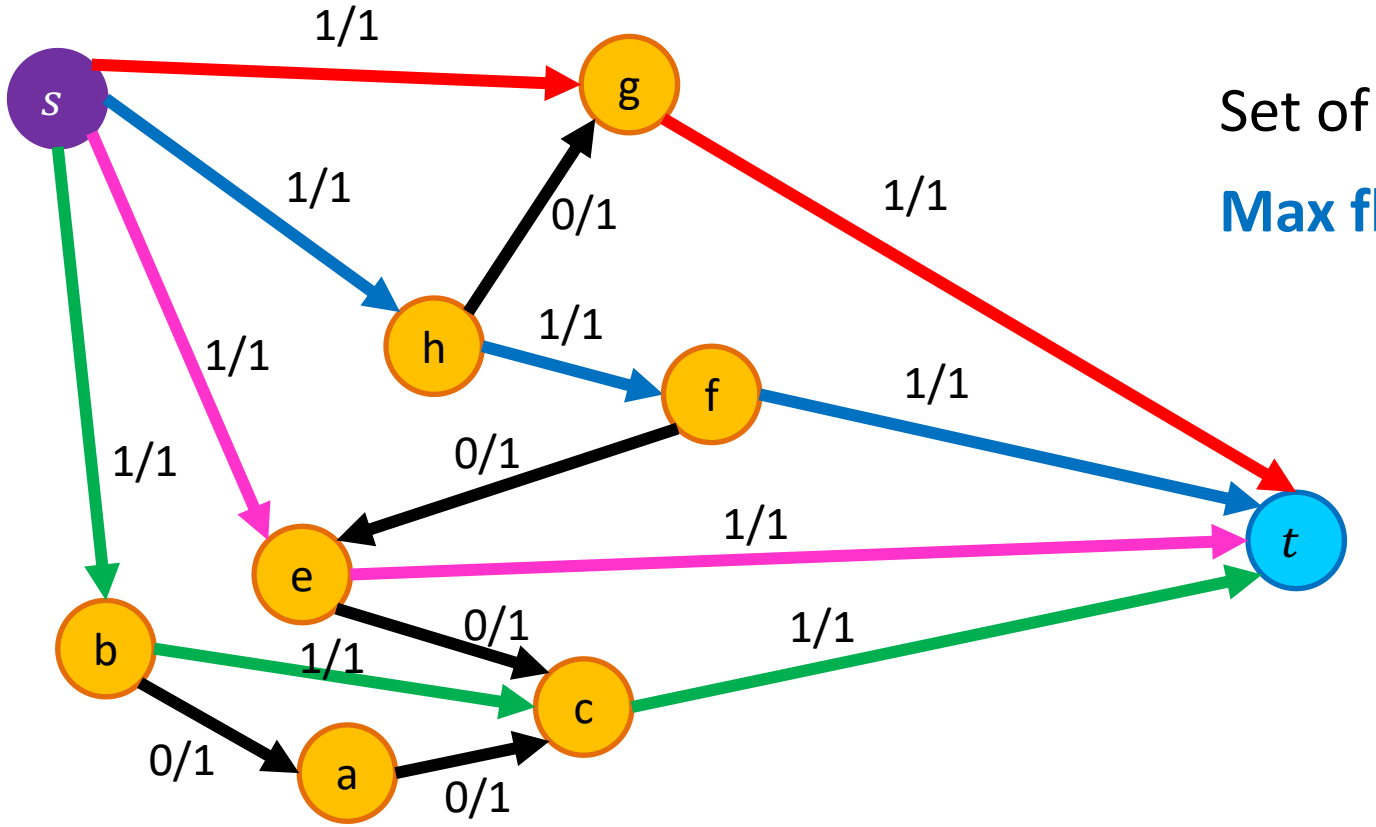
Set of edge-disjoint paths of size 4

Is this the max number?

Edge-Disjoint Paths Algorithm

Use a problem we know how to solve, ***max network flow***, to solve this!

Make u and v the source and sink, give each edge capacity 1, find the max flow.



Set of edge-disjoint paths of size 4

Max flow = 4

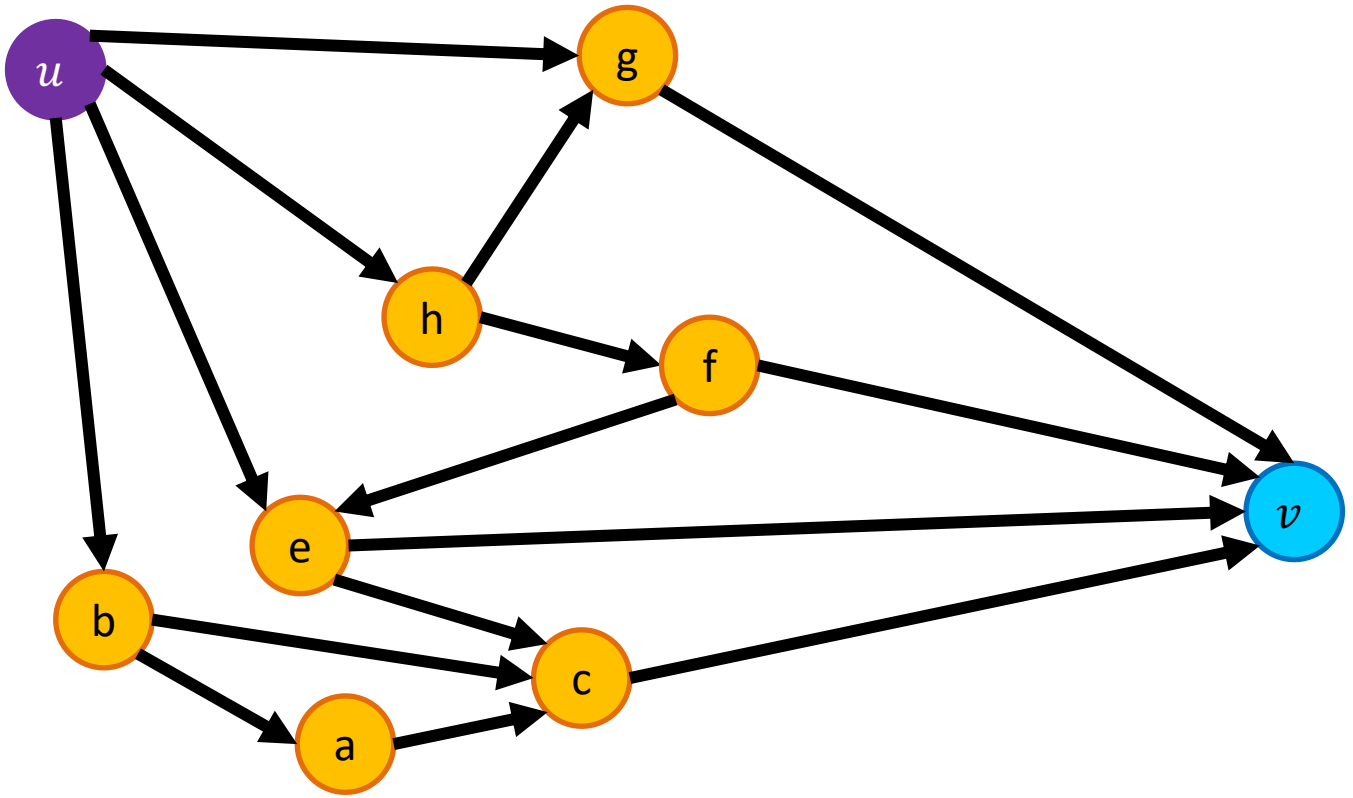
Why does this work?
We need to be able to make a valid argument that it always does.

What's the situation?

- Given an input I_1 for the ***max network flow problem*** (graph G with edge capacities), we can find the max flow for that input
- Given an input I_2 for ***edge-disjoint path problem***, we can:
  - Convert that input I_2 to make a valid input I_1 for ***network flow problem***, by using same graph G but adding capacity=1 for each edge
  - Solve ***max network flow problem*** for I_1 and get result R_1
  - Use R_1 to give the solution R_2 for ***edge-disjoint path*** for input I_2
    - In this case, **|f|** = **the number of paths**
- Next, let's solve another problem using our new *edge-disjoint path* solution

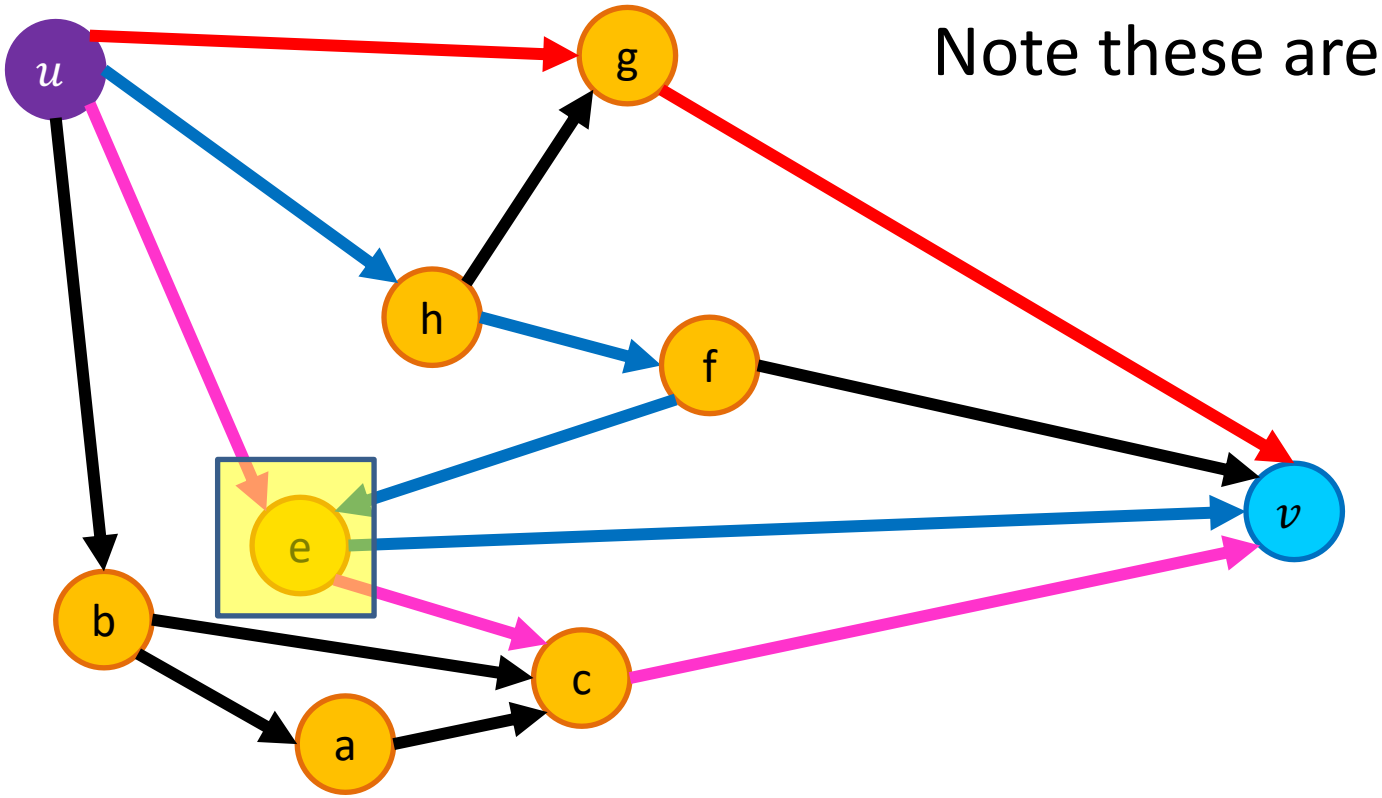
Vertex-Disjoint Paths

Given a graph $G = (V, E)$, a start node u and a destination node v, give the maximum number of paths from u to v which share no vertices



Vertex-Disjoint Paths

Given a graph $G = (V, E)$, a start node u and a destination node v, give the maximum number of paths from u to v which share no vertices



This shows 3 edge-disjoint paths.

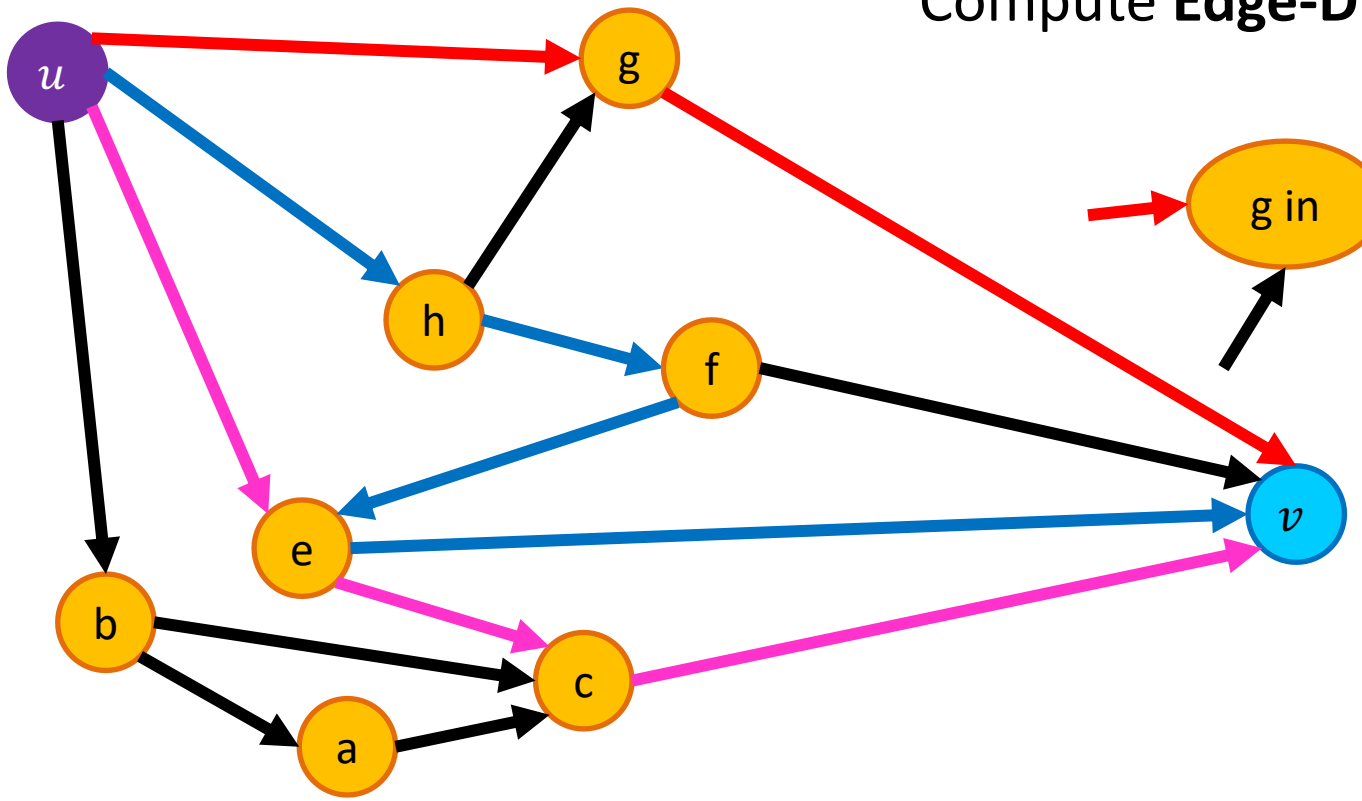
Note these aren't vertex-disjoint paths!

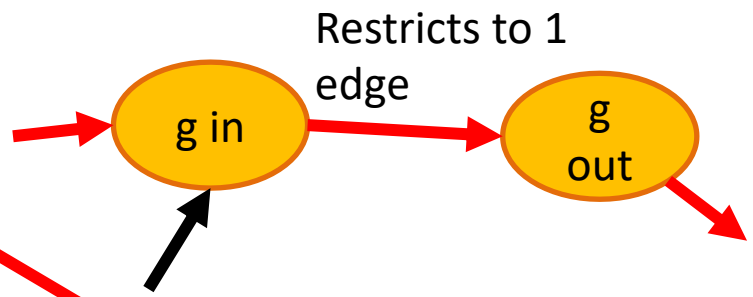
Vertex-Disjoint Paths Algorithm

Idea: Convert an instance of the vertex-disjoint paths problem into an instance of edge-disjoint paths

Make two copies of each node, one connected to incoming edges, the other to outgoing edges

Compute **Edge-Disjoint Paths** on new graph





Why does this work?
We need to be able to make a valid argument that it always does.

What's the situation now?

- Given an input I_1 for the ***max network flow problem*** (graph G with edge capacities), we can find max flow for that input
- Given an input I_2 for ***edge-disjoint path problem***, we can:
  - Convert that input I_2 to make a valid input I_1 for ***network flow problem***, and solve that to find **number of edge-disjoint paths**
- Given an input I_3 for ***vertex-disjoint path problem***, we can:
  - Convert that input I_3 to make a valid input I_2 for ***edge-disjoint path problem***
  - See above! Convert I_2 to I_1 and solve ***max network flow problem***
- This chain of "problem conversions" finds lets us solve ***vertex-disjoint path problem***
  - **Time complexity?** Cost of solving max network flow plus two conversions

Reductions

(We’re about to get interested in problems that seem to require exponential time…)

Max-flow vs. min-cut

- These two problems are “equivalent”
  - Remember? ***max-flow min-cut theorem***
  - Here we’re saying: if you can solve one, you can solve the other
- Alternatively, we can say that one problem ***reduces*** to the other
  - The problem of finding min-cut ***reduces to*** the problem of finding max-flow
  - Maybe this ***reduction*** requires some work to “convert”
    - Could be nothing or minimal
  - For these problems, the cost of the conversion is ***polynomial***

Reduction

- A ***reduction*** is a transformation of one problem into another problem
  - Min-cut is reducible to max-flow because we can use max-flow to solve min-cut
  - Formally, problem A is ***reducible*** to problem B if we can use a solution to B to solve A
- We're particularly interested in reductions that happen in *polynomial time*
- If A is ***polynomial-time reducible*** to B, we denote this as:

  $\mathbf{A \leq_p B}$

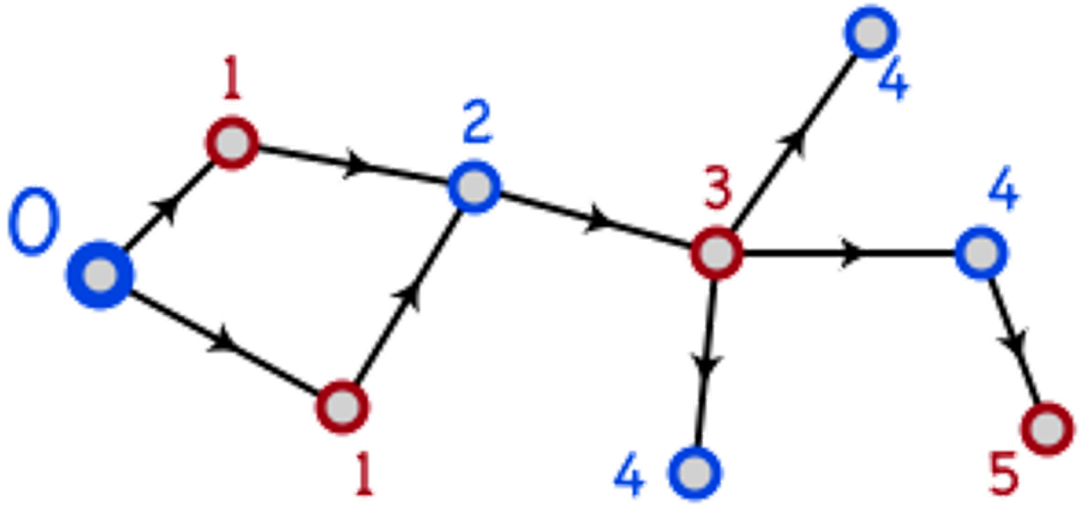
Reducing both ways

- It's easy to see that:
  - Min-cut $\leq_p$ max-flow
  - Max-flow $\leq_p$ min-cut
- Because they reduce both ways, they are ***polynomial-time equivalent***
  - If we find a polynomial solution for one, the other is also polynomial
  - What if we prove an exponential lower-bound for one?
    Is it possible that the other one could have a polynomial solution?

Bipartite Matching

Bipartite Graphs

- A graph is *bipartite* if node-set V can be split into sets X and Y such that every edge has one end in X and one end in Y
  - X and Y could be colored red and blue
  - Or Boolean true/false



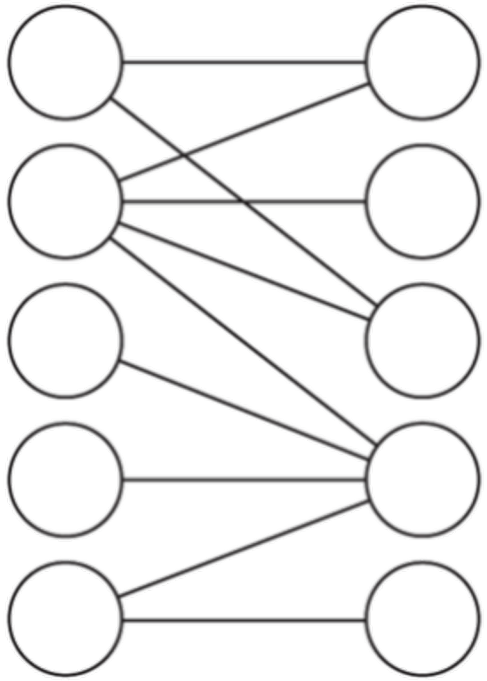
How to determine if G is bipartite?

The numbers and arrows on edges may give you a clue….

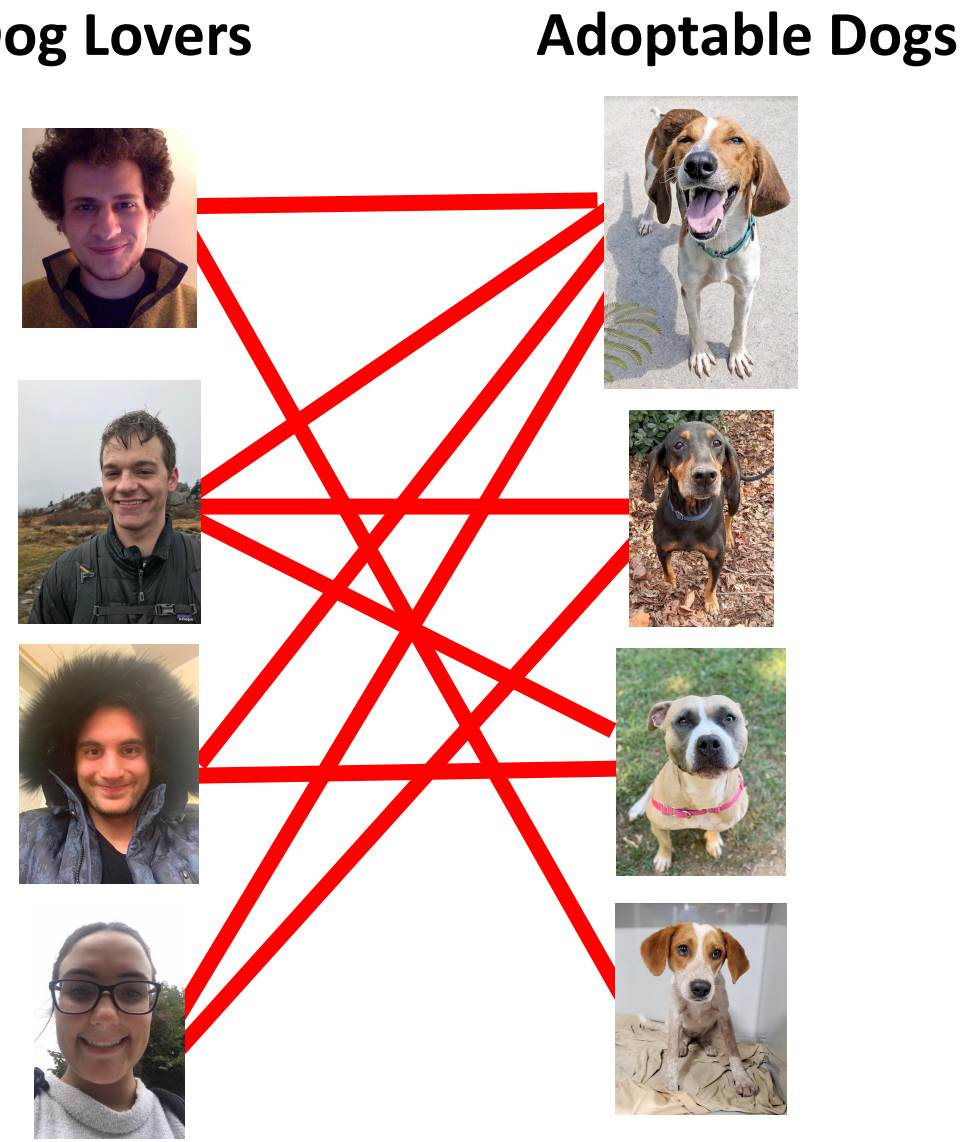
BFS or DFS, and label nodes by levels in tree.
Non-tree edge to node with same label means NOT bipartite.

Notes and assumptions

- We assume the graph is connected
  - Otherwise we will only look at each connected component individually
- A triangle cannot be bipartite
  - In fact, any graph with an odd length cycle cannot be bipartite



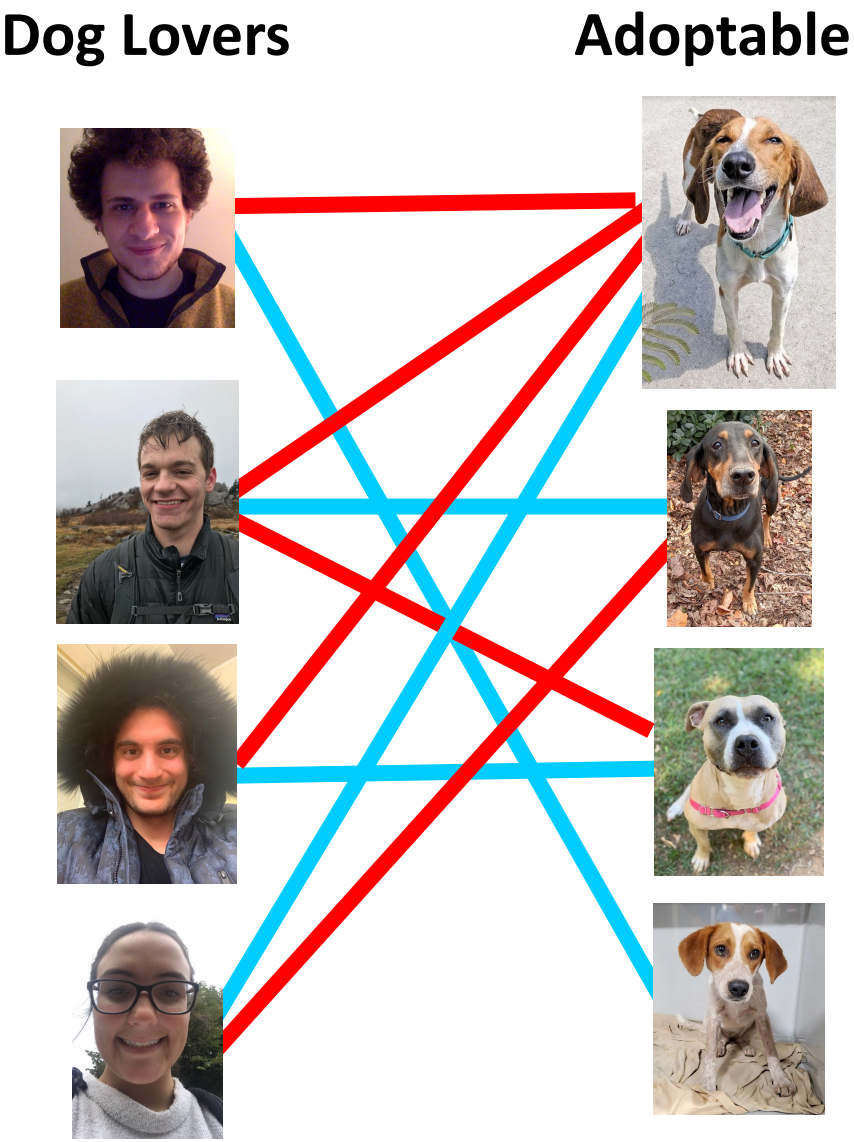
Maximum Bipartite Matching



Maximum Bipartite Matching

Is this the best possible?
The largest possible set of edges?

Maximum Bipartite Matching



Better! In fact, the maximum possible! How can we tell?

A ***perfect bipartite match***: Equal-sized left and right subsets, and all nodes have a matching edge

Maximum Bipartite Matching

Given a graph $G = (L, R, E)$

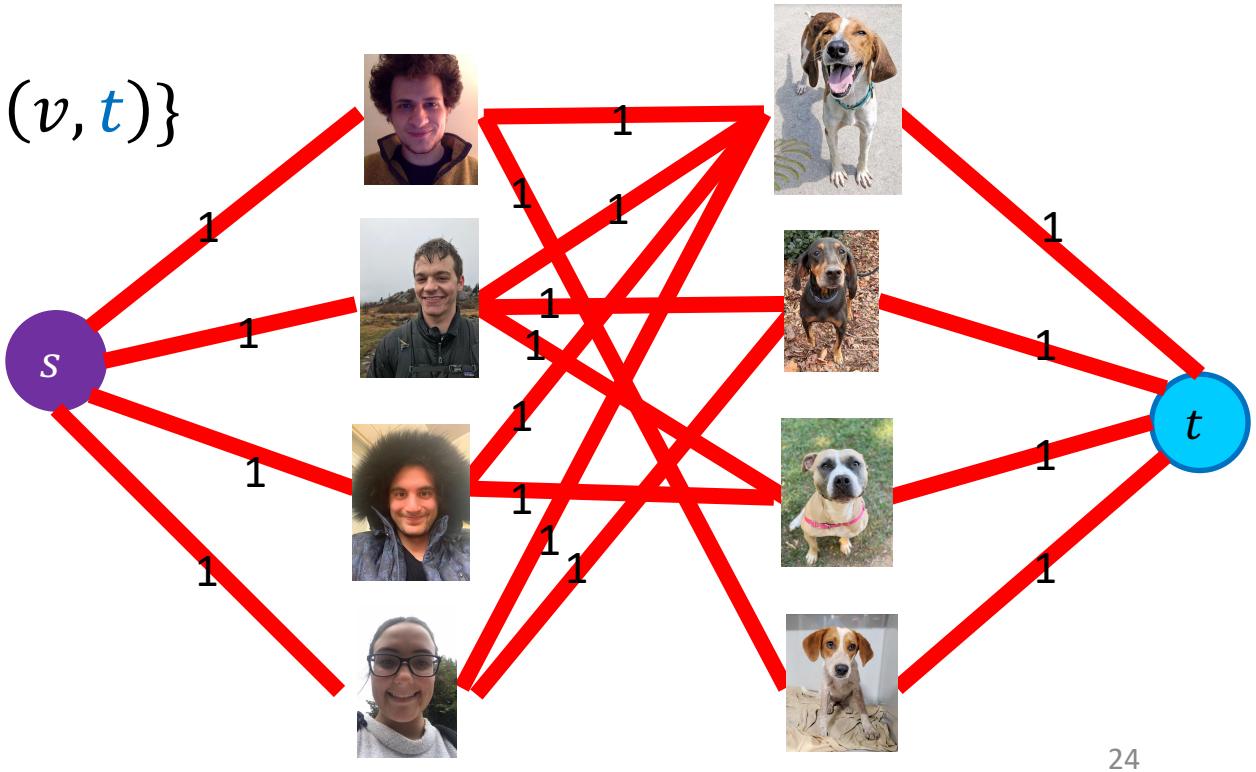
a set of left nodes, right nodes, and edges between left and right

Find the largest set of edges $M \subseteq E$ such that each node $u \in L$ or $v \in R$ is incident to at most one edge.

Maximum Bipartite Matching Using Max Flow

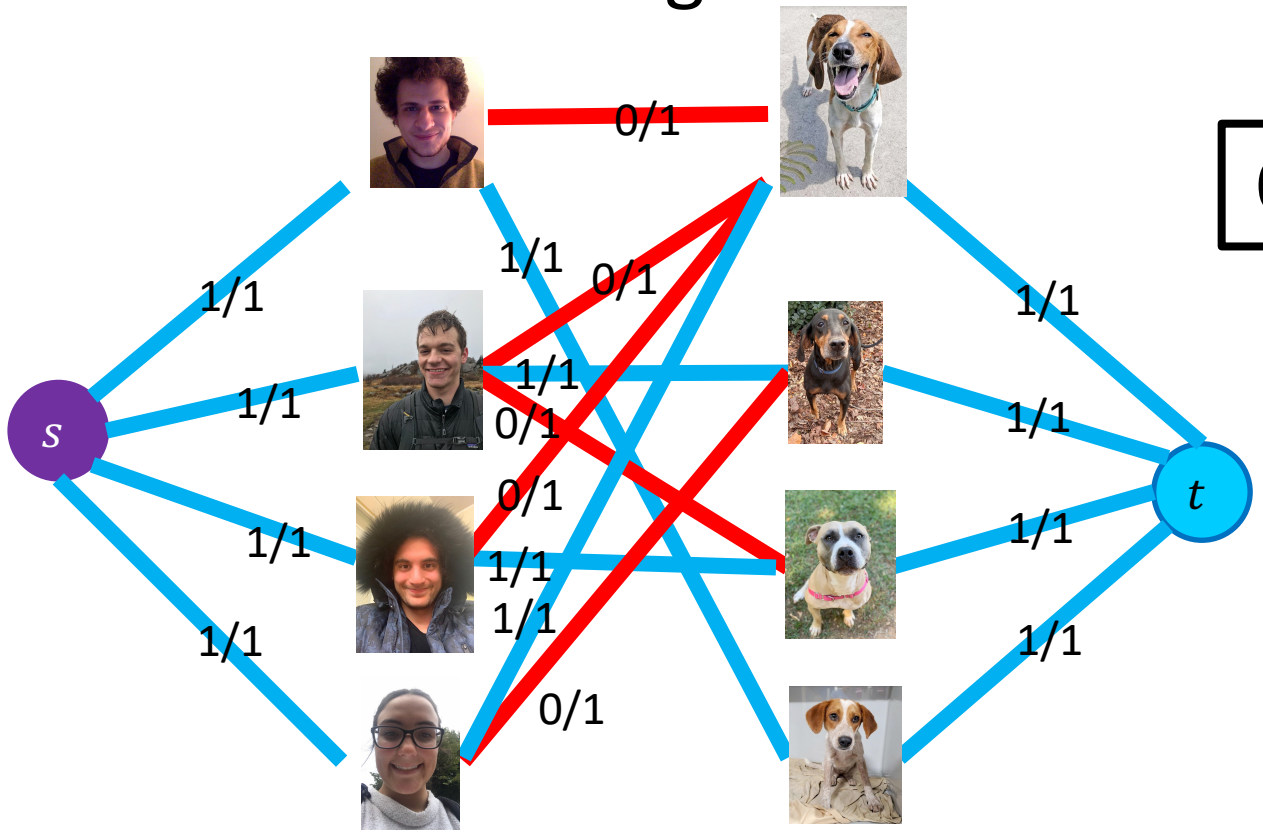
Make $G = (L, R, E)$ a flow network $G' = (V', E')$ by:

- Adding in a source and sink to the set of nodes:
  - $V' = L \cup R \cup \{s, t\}$
- Adding an edge from source to L and from R to sink:
  - $E' = E \cup \{u \in L \mid (s, u)\} \cup \{v \in r \mid (v, t)\}$
- Make each edge capacity 1:
  - $\forall e \in E', c(e) = 1$



Maximum Bipartite Matching Using Max Flow

1. Make G into G' $\Theta(L + R)$
2. Compute Max Flow on G' $\Theta(E \cdot V)$ $|f| \leq L$
3. Return M as all "middle" edges with flow 1 $\Theta(L + R)$



Overall: $\Theta(E \cdot V)$

Roadmap: Where We've Been and Why

- ***Reductions*** between problems
  - Why? Can be a practical way of solving a new problem
  - **Coming soon:** A proof about one problem's complexity can be applied to another
  - Formal definition of a reduction
- Examples
  - Bipartite graphs, matching
- **Next:** example problems: vertex cover and independent set
  - Then, classes of problems: P, NP, NP-Hard, NP-complete

CS4102 Algorithms

Spring 2021 – Floryan and Horton

Module 4, Day 3: Live Session, April 26

Roadmap: Where We've Been and Why

- ***Reductions*** between problems
  - Why? Can be a practical way of solving a new problem
  - **Coming soon:** A proof about one problem's complexity can be applied to another
  - Formal definition of a reduction
- Examples
  - Bipartite matching and max network flow
- **Today:** example problems: vertex cover and independent set
- **Next Lecture:** classes of problems: P, NP, NP-Hard, NP-complete

- Today: Two graph problems that are similar:
  minimum vertex cover and ***maximum independent set***

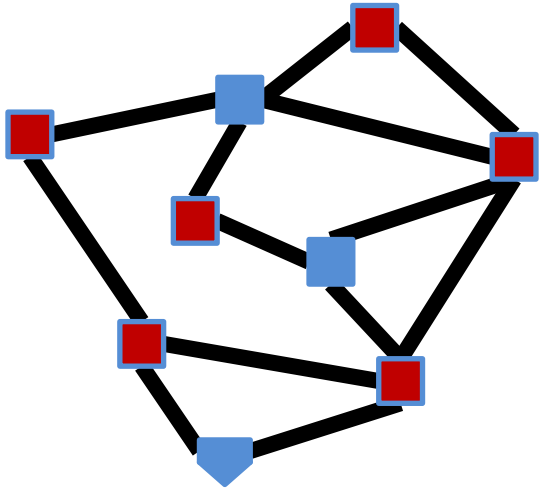
  – We'll do a formal proof that these are ***polynomial-time equivalent***
    - Each reduces to the other

  – BTW, vertex cover is needed for Basic Written HW, Question 2

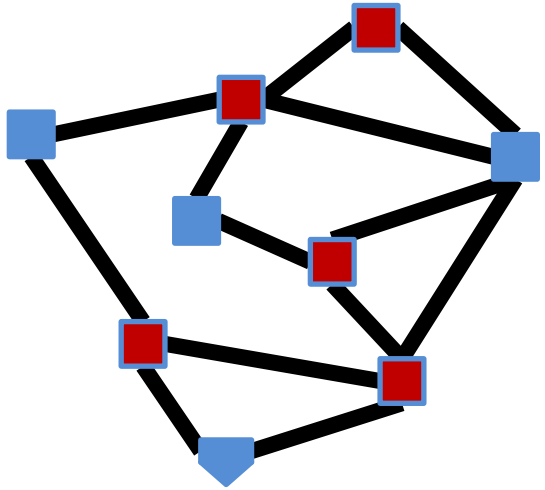
Minimum Vertex Cover

- Vertex Cover: $C \subseteq V$ is a vertex cover if every edge in E has one of its endpoints in C
- Minimum Vertex Cover Problem: Given a graph $G = (V, E)$ find the minimum vertex cover C

Vertex cover of size 6



Vertex cover of size 5

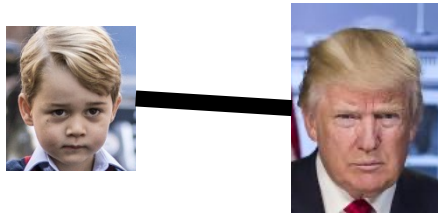


Is 5 the smallest?
Is there one with 4?

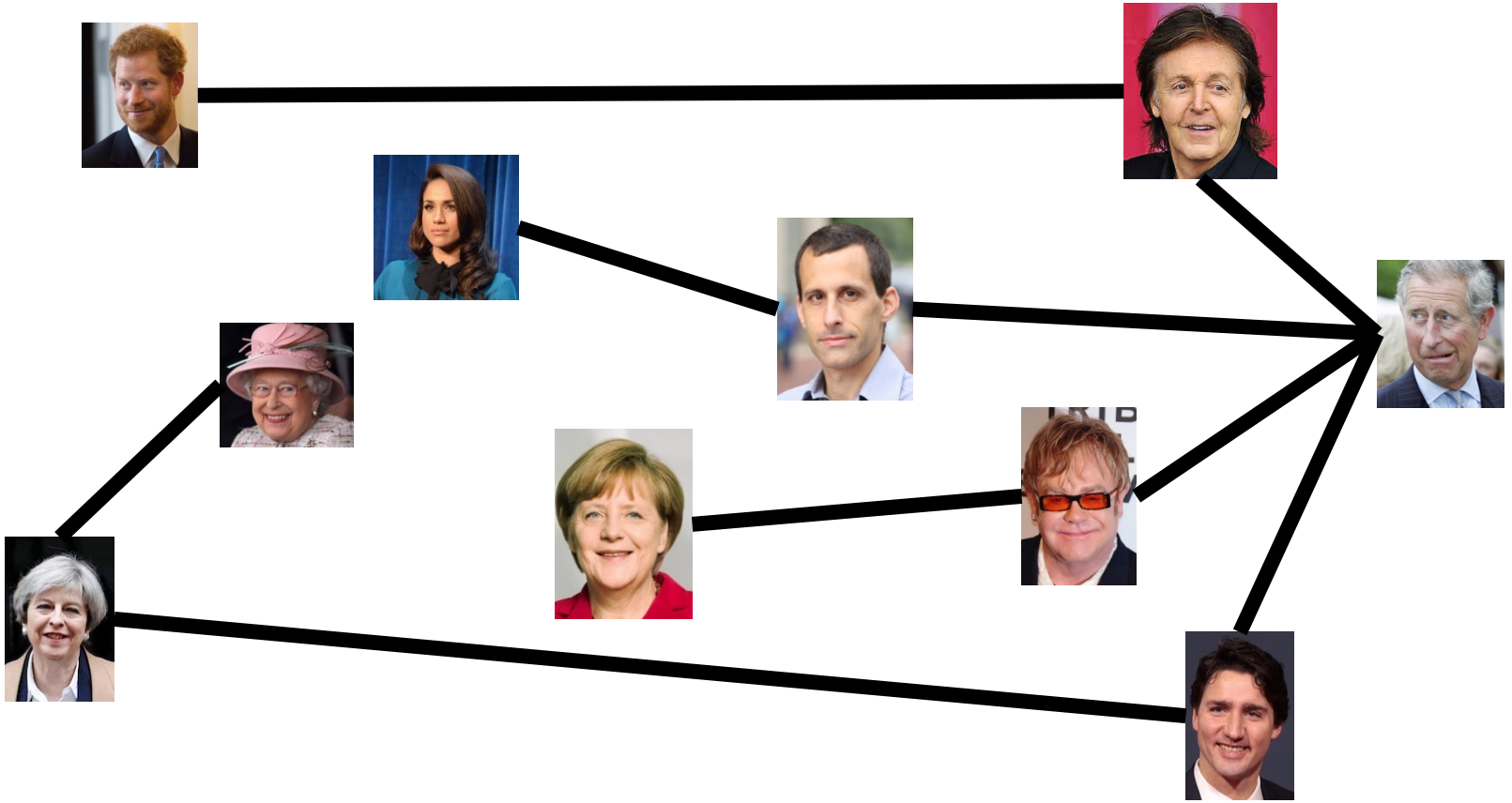
Maximum Independent Set

- Independent set: $S \subseteq V$ is an independent set if no two nodes in S share an edge
- Maximum Independent Set Problem: Given a graph $G = (V, E)$ find the maximum independent set S

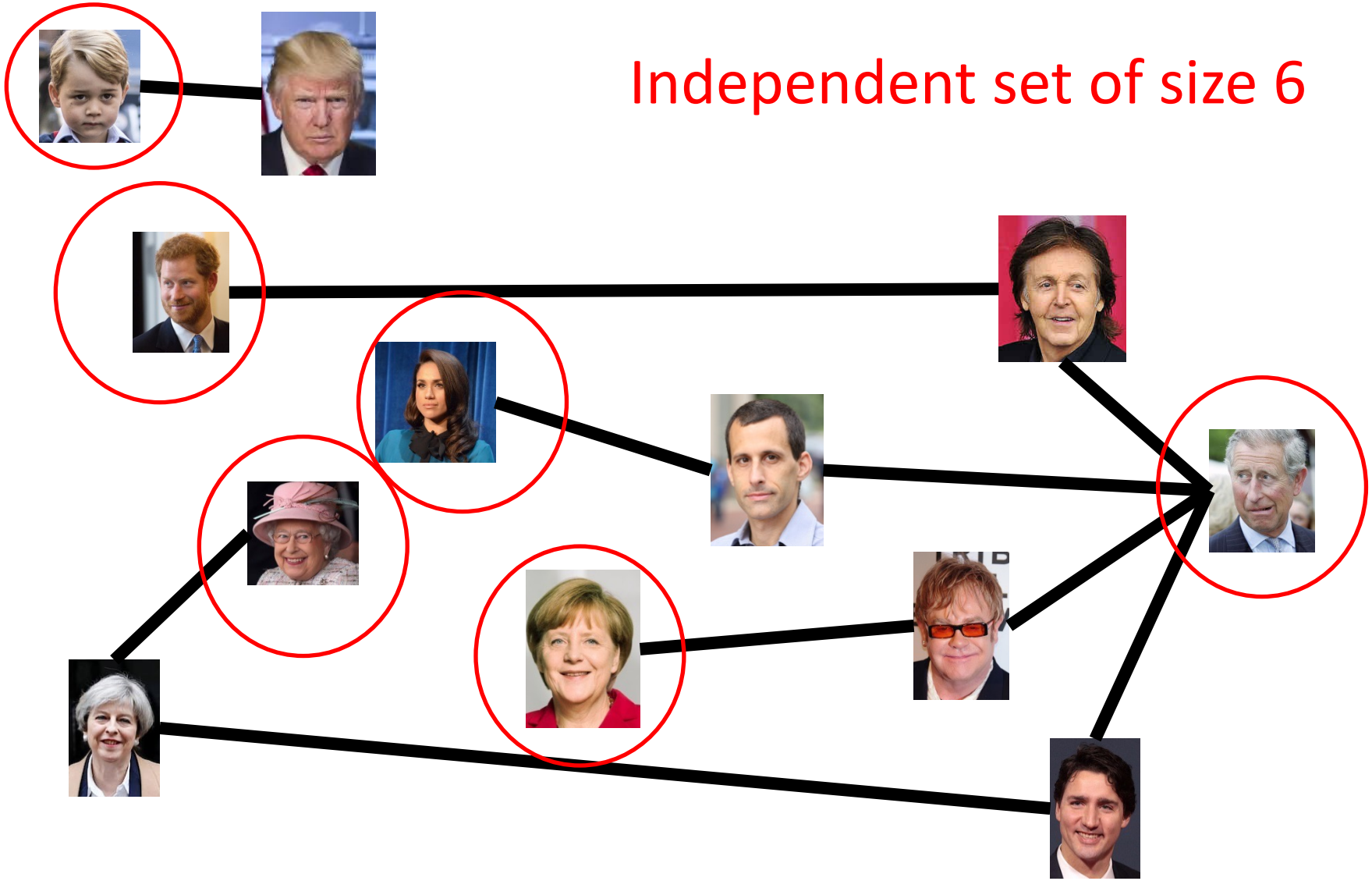
Party Problem



Draw edges between people who don't get along.
Find the maximum number of people who DO get along.

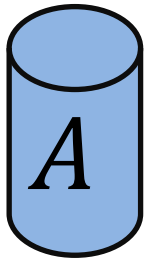


Example



In General: A Reduction

A problem we <u>don't</u> know how to solve

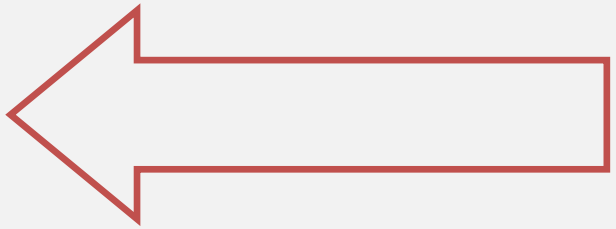


Solution for A

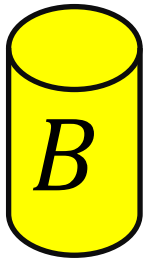
Reduction

Map Instances of problem A to Instances of B

Map Solutions of problem B to Solutions of A

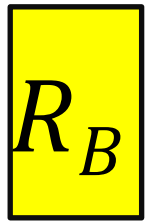


A problem we <u>do</u> know how to solve



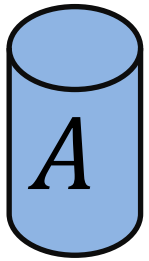
Using any Algorithm for B

Solution for B



In General: Reduction

Problem we <u>don't</u> know how to solve

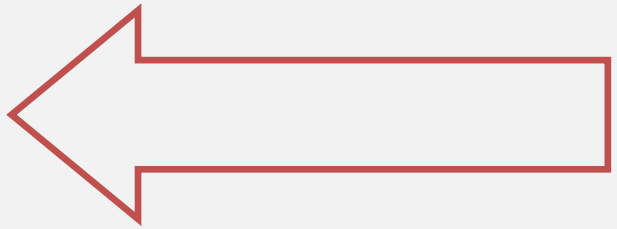


Solution for A

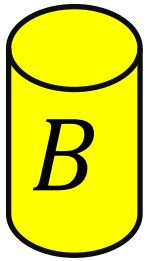
Reduction

Map Instances of problem A to Instances of B

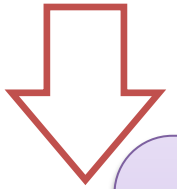
Map Solutions of problem B to Solutions of A



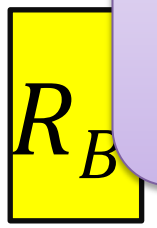
Problem we <u>do</u> know how to solve



Using any Algorithm for B



Solutio



For now: we are NOT focusing on an algorithm to solve one of these, <u>just on the reduction</u>!

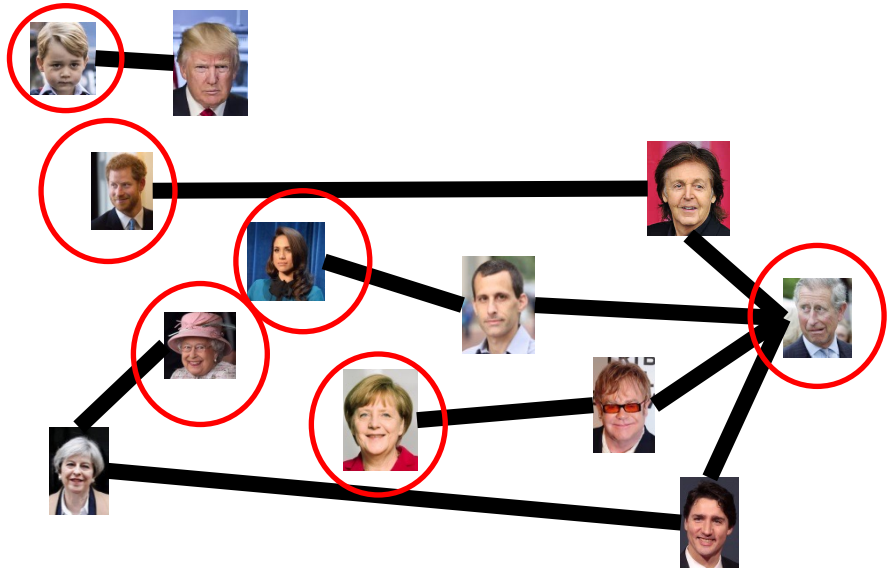
Polynomial-Time Equivalence

- We'll show a reduction in both directions:
  - MinVertCov $\leq_p$ MaxIndSet
  - MaxIndSet $\leq_p$ MinVertCov
- Because they reduce both ways, they are ***polynomial-time equivalent***
  - *(We'll justify the following claims later.)*
  - If we find a polynomial solution for one, the other is also polynomial
  - What if we prove an exponential lower-bound for one?
    Is it possible that the other one could have a polynomial solution?

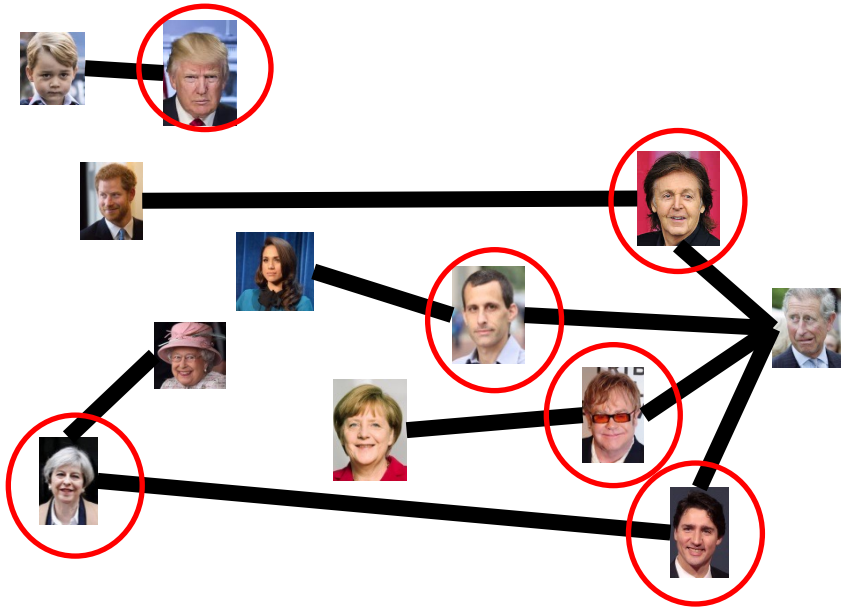
Reduction Idea

S is an independent set of G iff $V - S$ is a vertex cover of G

Independent Set



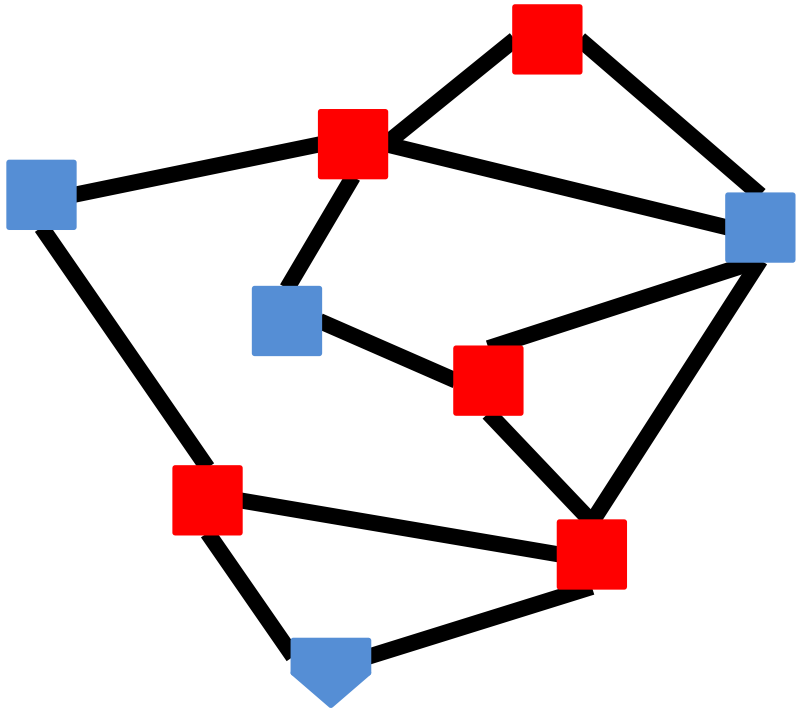
Vertex Cover



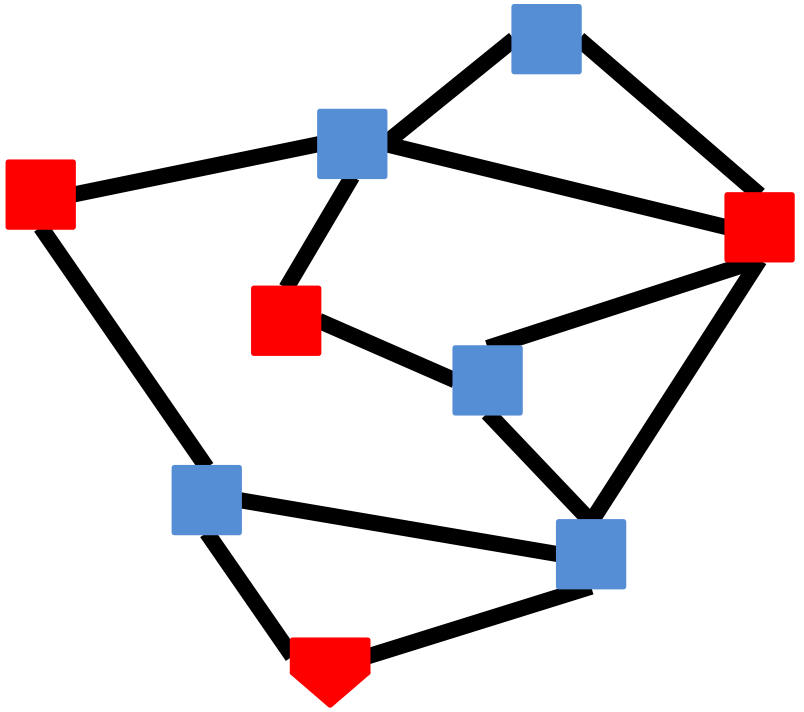
Reduction Idea

S is an independent set of G iff $V - S$ is a vertex cover of G

Vertex Cover



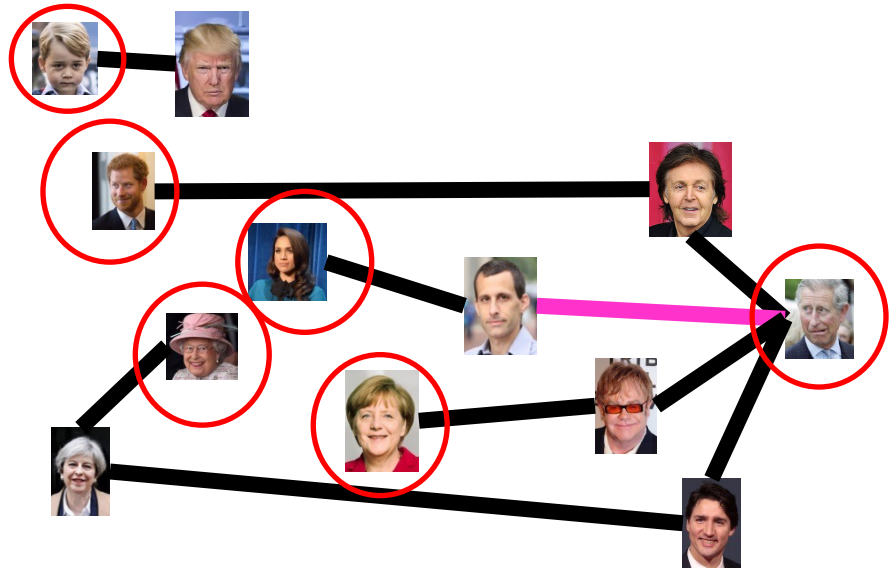
Independent Set



Proof: ⇒

S is an independent set of G iff $V - S$ is a vertex cover of G

Let S be an independent set



Consider any edge $(x, y) \in E$

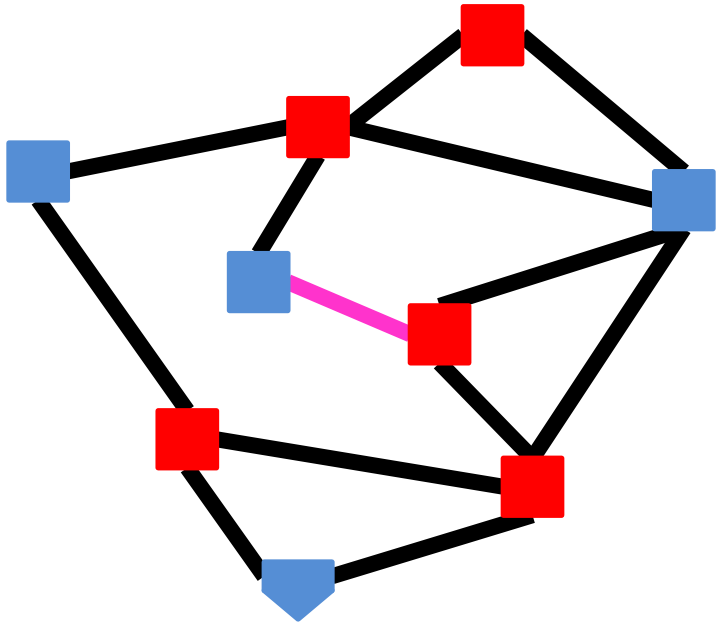
If $x \in S$ then $y \notin S$, because otherwise S would not be an independent set

Therefore $y \in V - S$, so edge (x, y) is covered by $V - S$

Proof: ⇐

S is an independent set of G iff $V - S$ is a vertex cover of G

Let $V - S$ be a vertex cover



Consider any edge $(x, y) \in E$

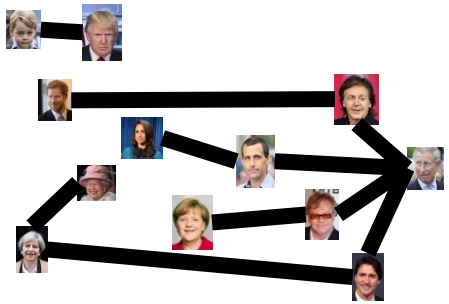
At least one of x and y belong to $V - S$, because $V - S$ is a vertex cover

Therefore x and y are not both in S,

No edge has both end-nodes in S, thus S is an independent set

The Reduction: MaxIndSet $\leq_P$ MinVertCov

MaxIndSet



A

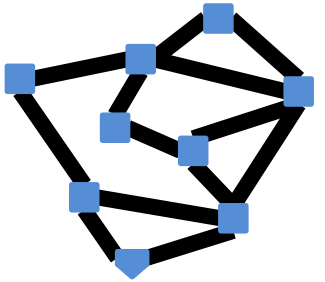
Reduction

Do nothing. $\Theta(1)$

Take complement of solution. $\Theta(V)$

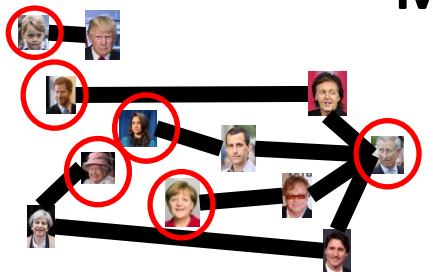
MinVertCov

B



Algorithm for **MinVertCov**

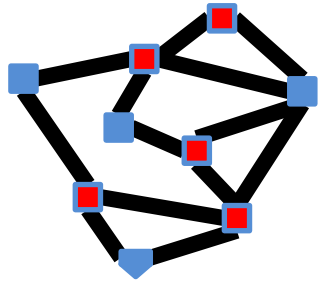
Solution for **MaxIndSet**



R_A

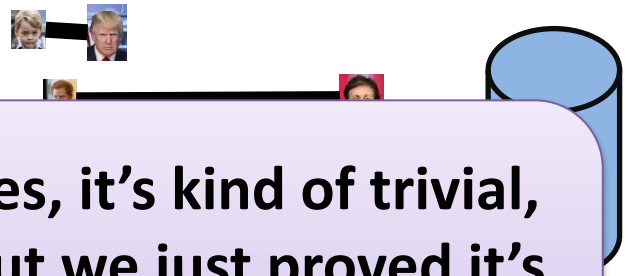
Solution for **MinVertCov**

R_B



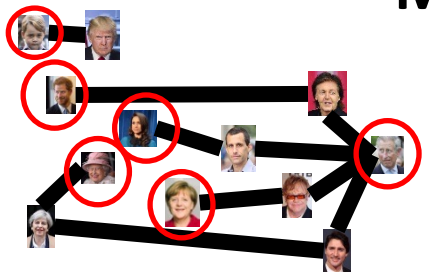
MaxIndSet $\leq_P$ MinVertCov

MaxIndSet



Yes, it's kind of trivial, but we just <u>proved</u> it's a valid reduction. And it's polynomial.

Solution for **MaxIndSet**



R_A

Reduction

Do nothing $\Theta(1)$

Take complement of solution $\Theta(V)$

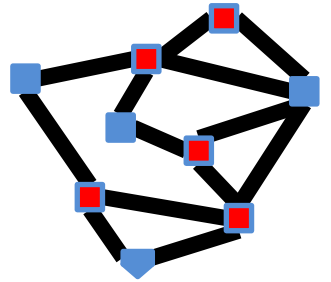
MinVertCov

We won't show it, but showing the other direction MinVertCov $\leq_P$ MaxIndSet is like this slide

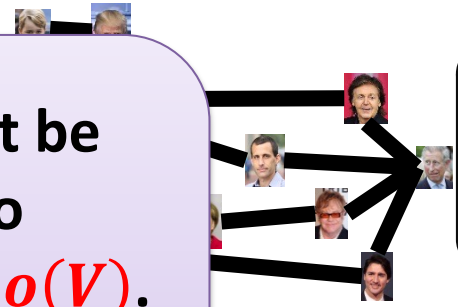
Algorithm for **MinVertCov**

Solution for **MinVertCov**

R_B



MaxIndSet $\leq_V$ MinVertCov



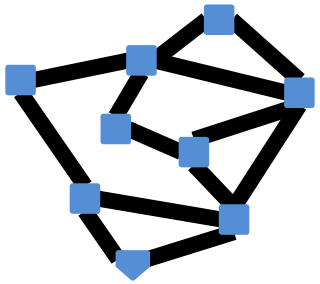
A

Problem **A**

Reduces to, in $\Theta(V)$

B

Problem **B**



…must not be possible to solve B in $o(V)$. Otherwise, reduction gives us a $o(V)$ solution for A.

Algorithm for **B**

can be used to make

With $\Theta(V)$ overhead

Algorithm for **A**

If impossible to solve A in $o(V)$…

Given that A $\leq_V$ B:

If we prove A requires time $\Omega(V)$ time, then B must also require $\Omega(V)$ time.

MaxIndSet $\leq_P$ MinVertCov

For two problems A and B, if we show A $\leq_p$ B then:

1. If **we prove A is exponential**, then B must be exponential.
   - Otherwise, the polynomial reduction from A to B gives us a polynomial solution to A.
2. If **we find a polynomial algorithm to B**, then A is polynomial.
   - Use the reduction: Convert input for A to input for B, solve B in polynomial time, and you have solution for A (in polynomial time)

Therefore:

MaxIndSet and MinVertCov are either both polynomial, or both exponential

MaxIndSet $\leq_P$ MinVertCov

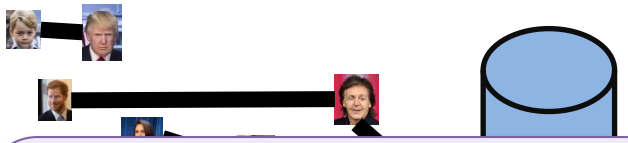
MaxIndSet and MinVertCov are either both polynomial, or both exponential

Which is true?

- Spoiler alert: We don't know which is true!
  (But we think they're both exponential.)
- Both problems are NP-Complete
  (We'll explain what that means soon!)

MaxIndSet $\leq_p$ MinVertCov

MaxIndSet



How about we just find a polynomial solution for MinVertCov? ☺

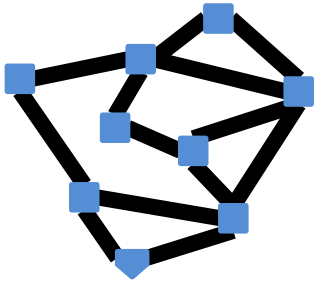
Reduction

Do nothing $\Theta(1)$

Take complement of solution $\Theta(V)$

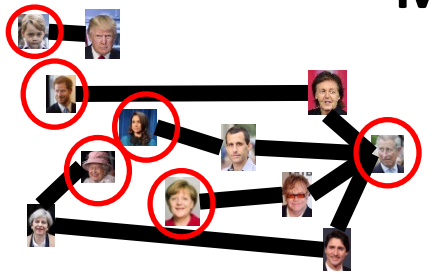
MinVertCov

B



Algorithm for **MinVertCov**

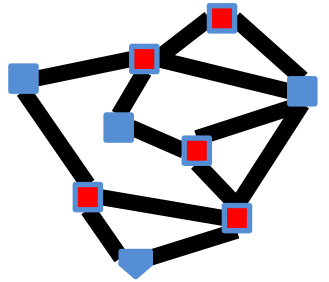
Solution for **MaxIndSet**



R_A

Solution for **MinVertCov**

R_B

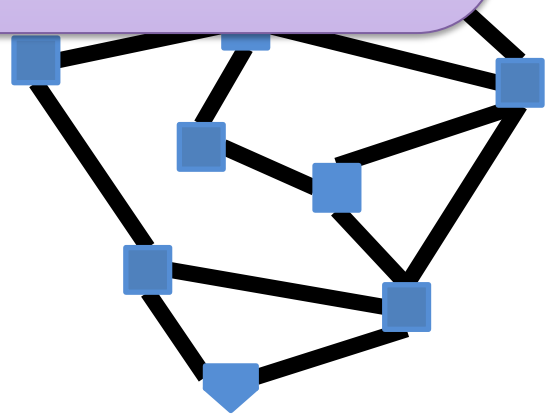


Minimum Vertex Cover

- Vertex Cover: $C \subseteq V$ is a vertex cover if every edge in E has one of its endpoints in C

Cover Problem: Given a graph $G = (V, E)$ find the over C

Guess what?
We can find counterexamples for each of these to show they don't always work.



Strategy? How about greedy?

Greedy choice?

(1) Pick remaining vertex of max degree.
Remove its incident edges.
(2) CLRS 35.1. Pick any edge (u,v).
Add both to result.
Remove all incident edges for u and v.

Wrapping Up

- No one has found a polynomial algorithm for MinVertCov

- Summary of what we've done
  - Two new problems: MinVertCov and MaxIndSet
  - Proved a reduction between them
  - Examined the consequences of polynomial-time equivalence for two problems with regard to whether they're polynomial or exponential