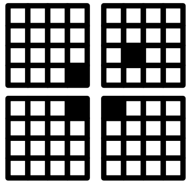
CS 3100
Data Structures and Algorithms 2
Lecture 21: Reductions, Bipartite Matching

Co-instructors: Robbie Hott and Tom Horton

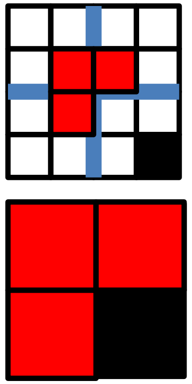
Fall 2023

Readings from CLRS 4th Ed:
Chapter 24

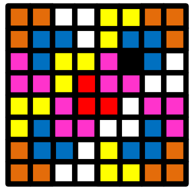
Divide and Conquer*



- **Divide**:
  - Break the problem into multiple subproblems, each smaller instances of the original
- **Conquer**:
  - If the suproblems are "large":
    - Solve each subproblem recursively
  - If the subproblems are "small":
    - Solve them directly (base case)



- **Combine**:
  - Merge together solutions to subproblems



*CLRS Chapter 4

Dynamic Programming

- Requires Optimal Substructure
  - Solution to larger problem contains the solutions to smaller ones
- Idea:
  1. Identify recursive structure of the problem
  2. Select a good order for solving subproblems
     - Usually smallest problem first

Greedy Algorithms

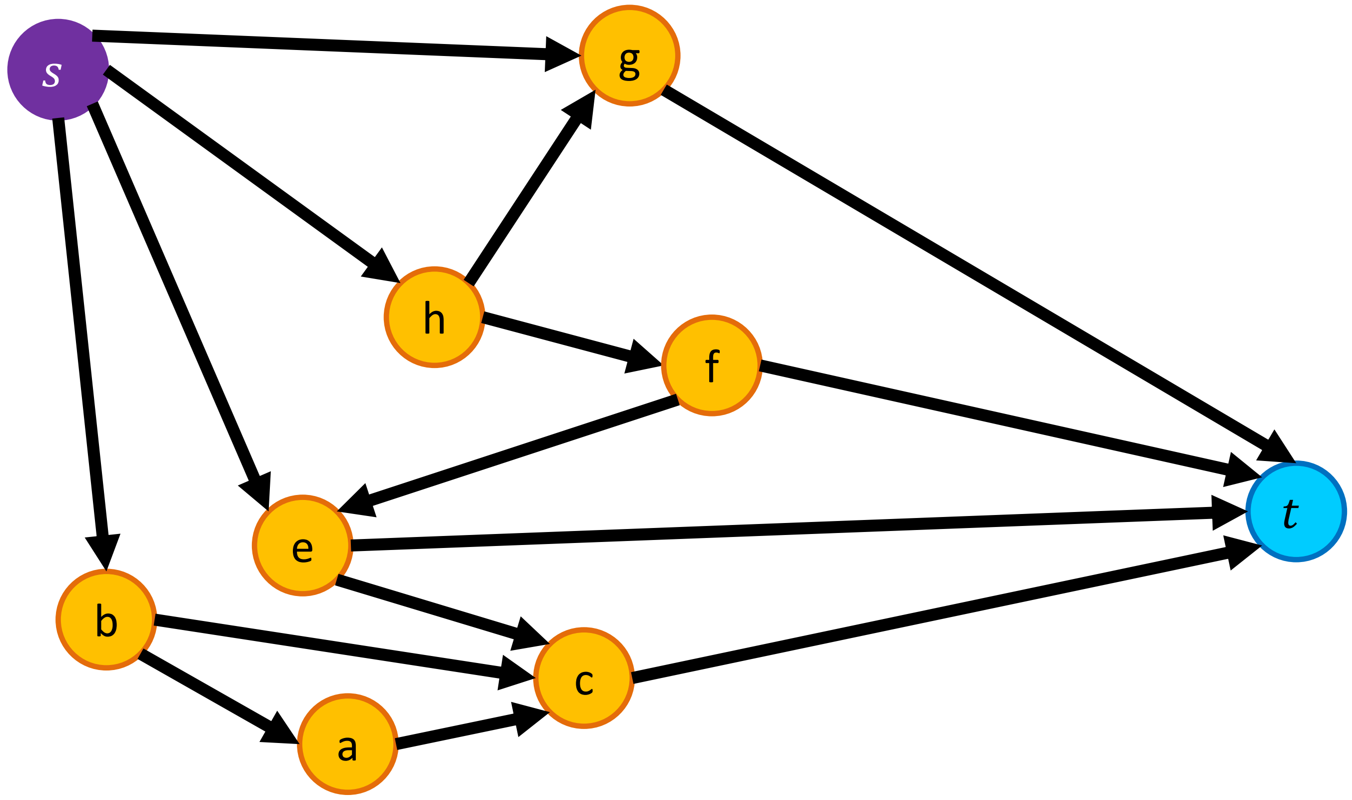
- Require Optimal Substructure
  – Solution to larger problem contains the solution to a smaller one
  – Only one subproblem to consider!
- Idea:
  1. Identify a greedy choice property
     - How to make a choice guaranteed to be included in some optimal solution
  2. Repeatedly apply the choice property until no subproblems remain

So far

- Divide and Conquer, Dynamic Programming, Greedy
  - Take an instance of *Problem A*,
    relate it to smaller instances of *Problem A*
- Next:
  - Take an instance of *Problem A*,
    relate it to an instance of ***Problem B***

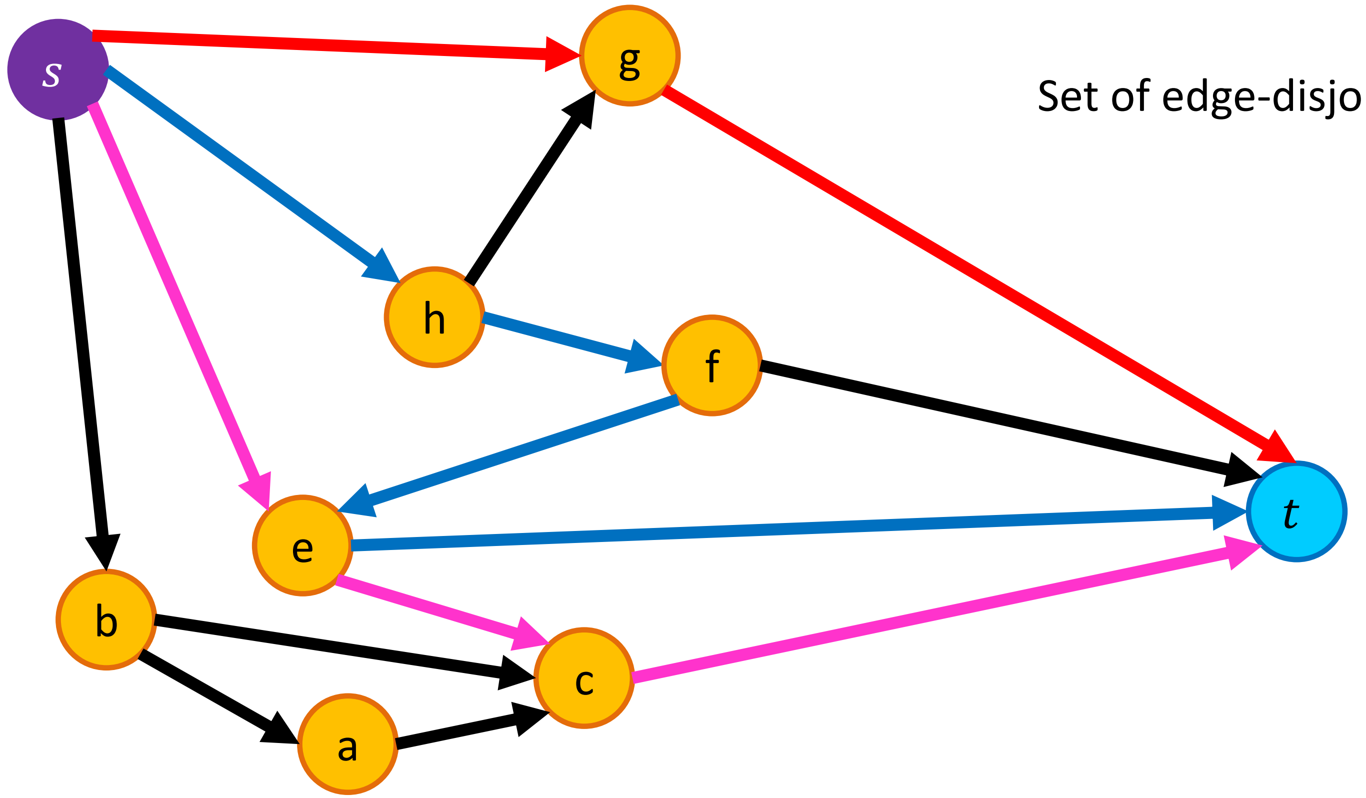
Edge-Disjoint Paths

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



Edge-Disjoint Paths

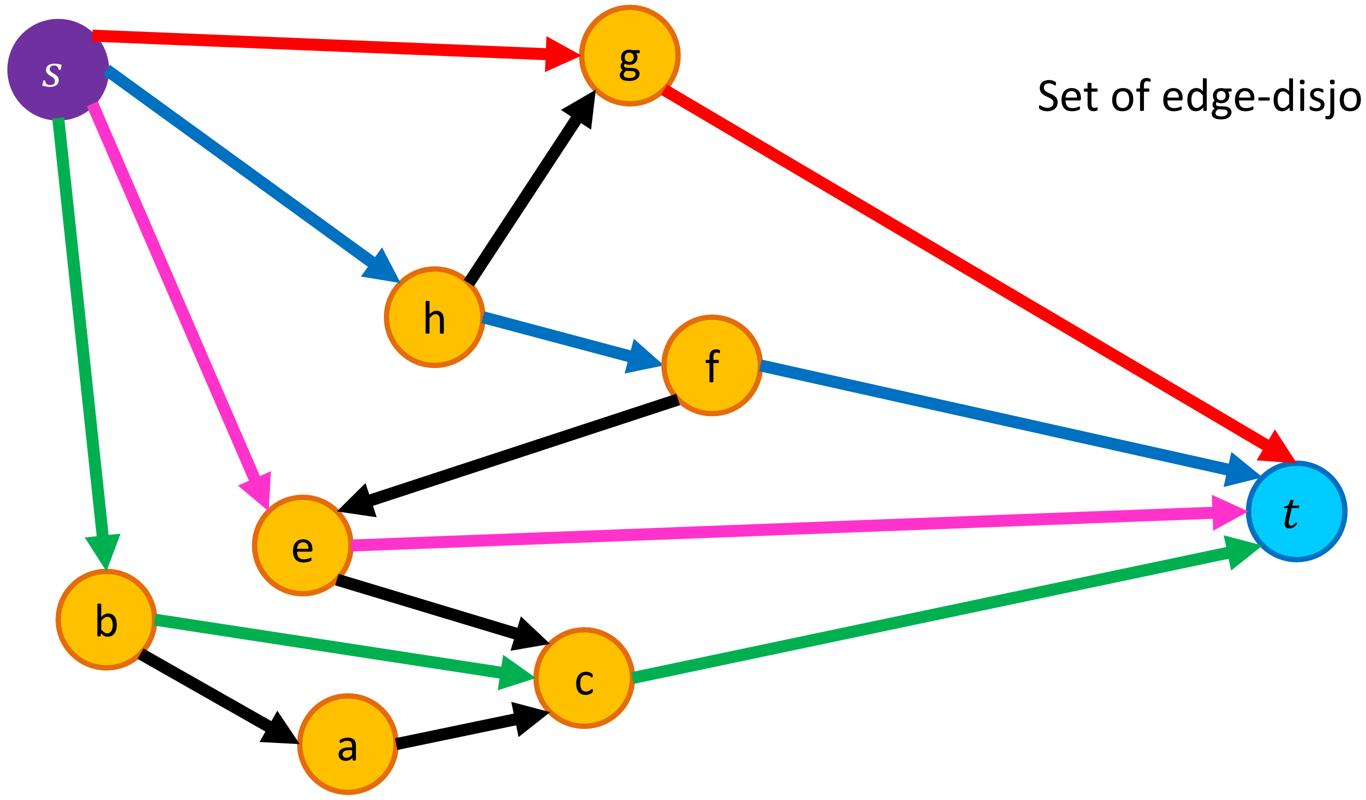
Given a graph $G = (V, E)$, a start node s and a destination node t, give the maximum number of paths from s to t which share no edges



Set of edge-disjoint paths of size 3

Edge-Disjoint Paths

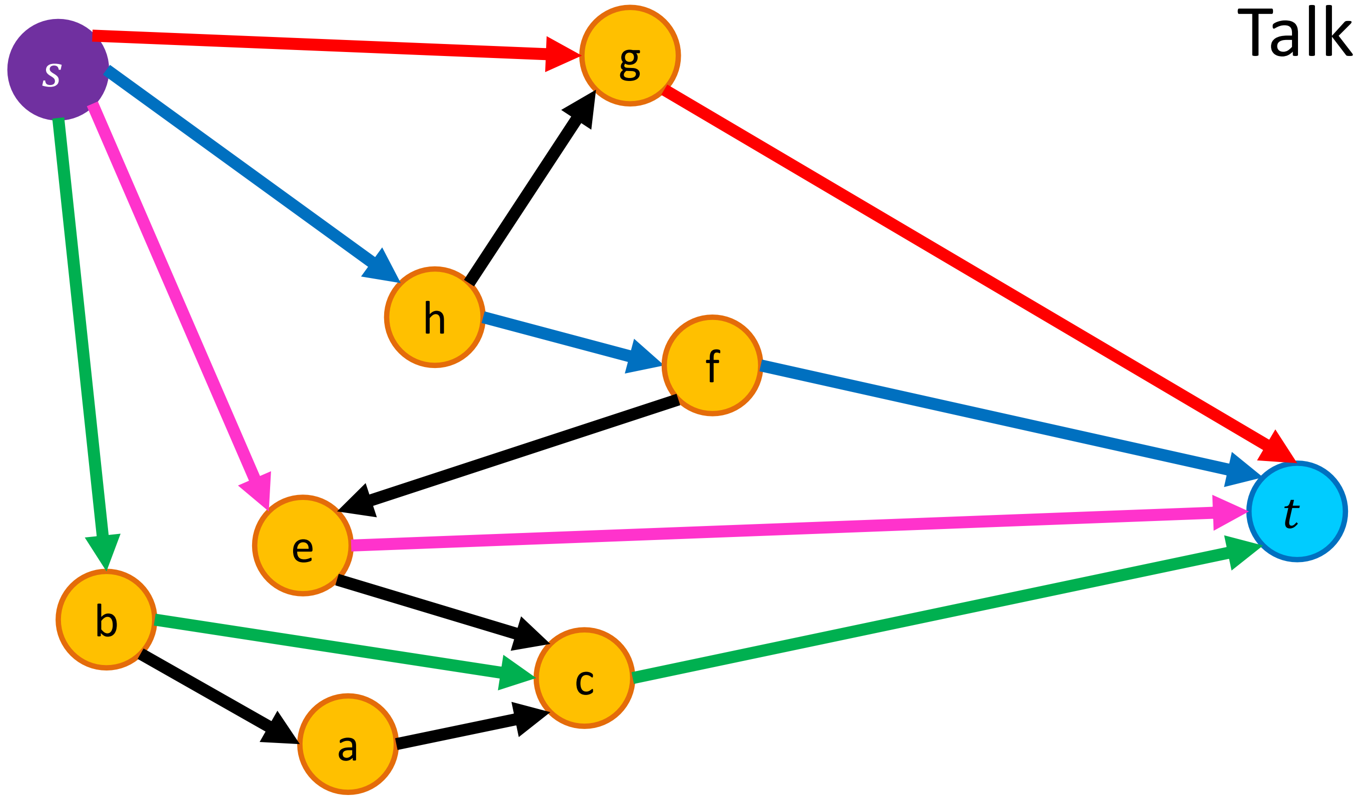
Given a graph $G = (V, E)$, a start node s and a destination node t, give the maximum number of paths from s to t which share no edges



Set of edge-disjoint paths of size 4

Edge-Disjoint Paths

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

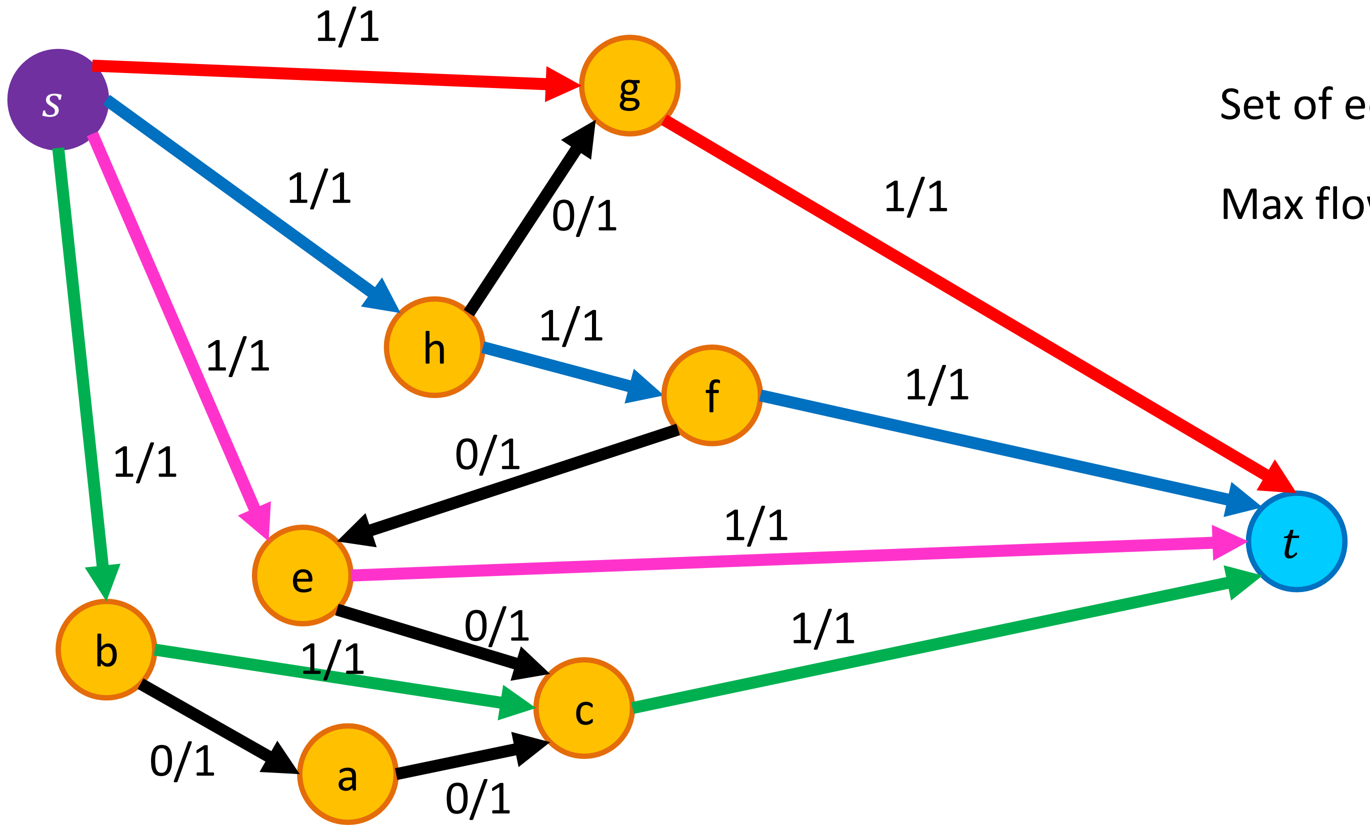


How could we solve this?

Talk with your neighbors!

Edge-Disjoint Paths Algorithm

Make s and t the source and sink, give each edge capacity 1, find the max flow.

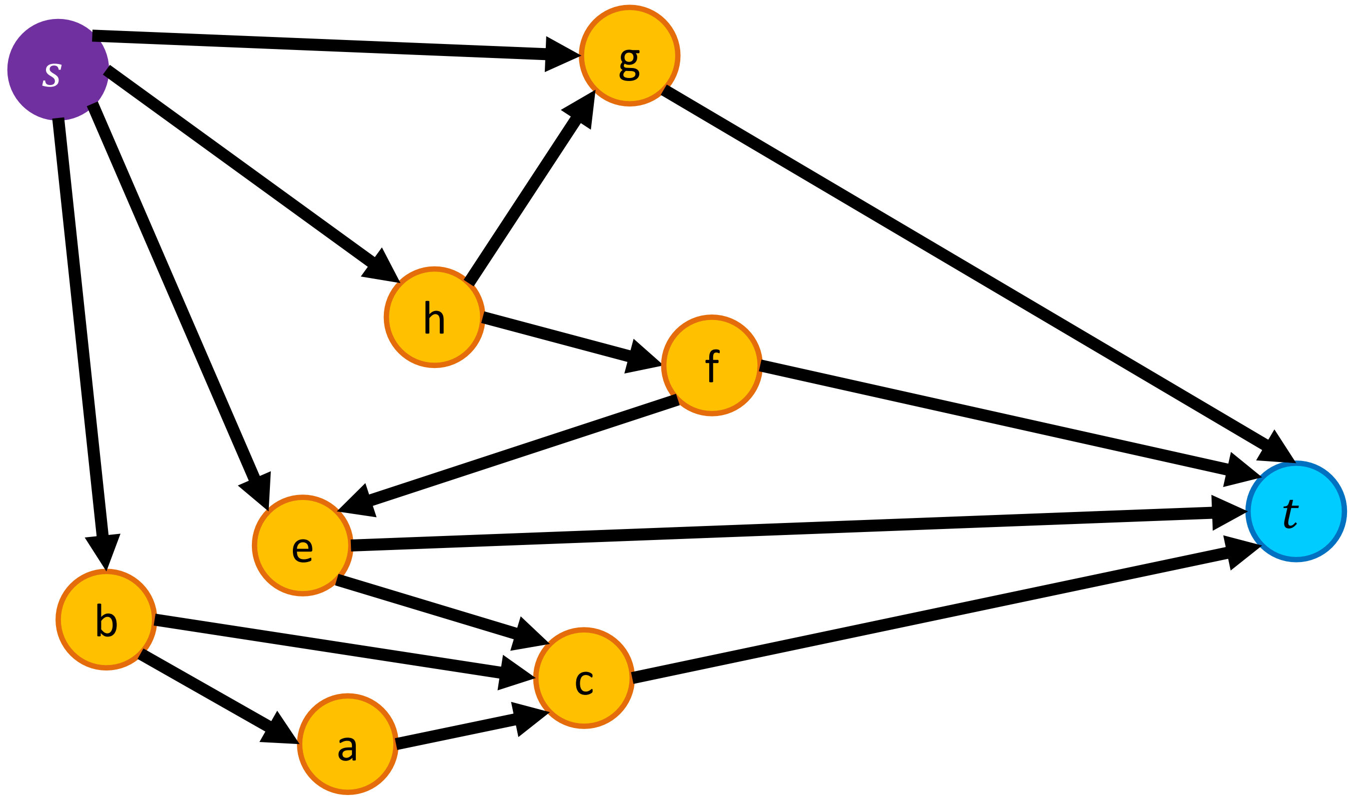


Set of edge-disjoint paths of size 4

Max flow = 4

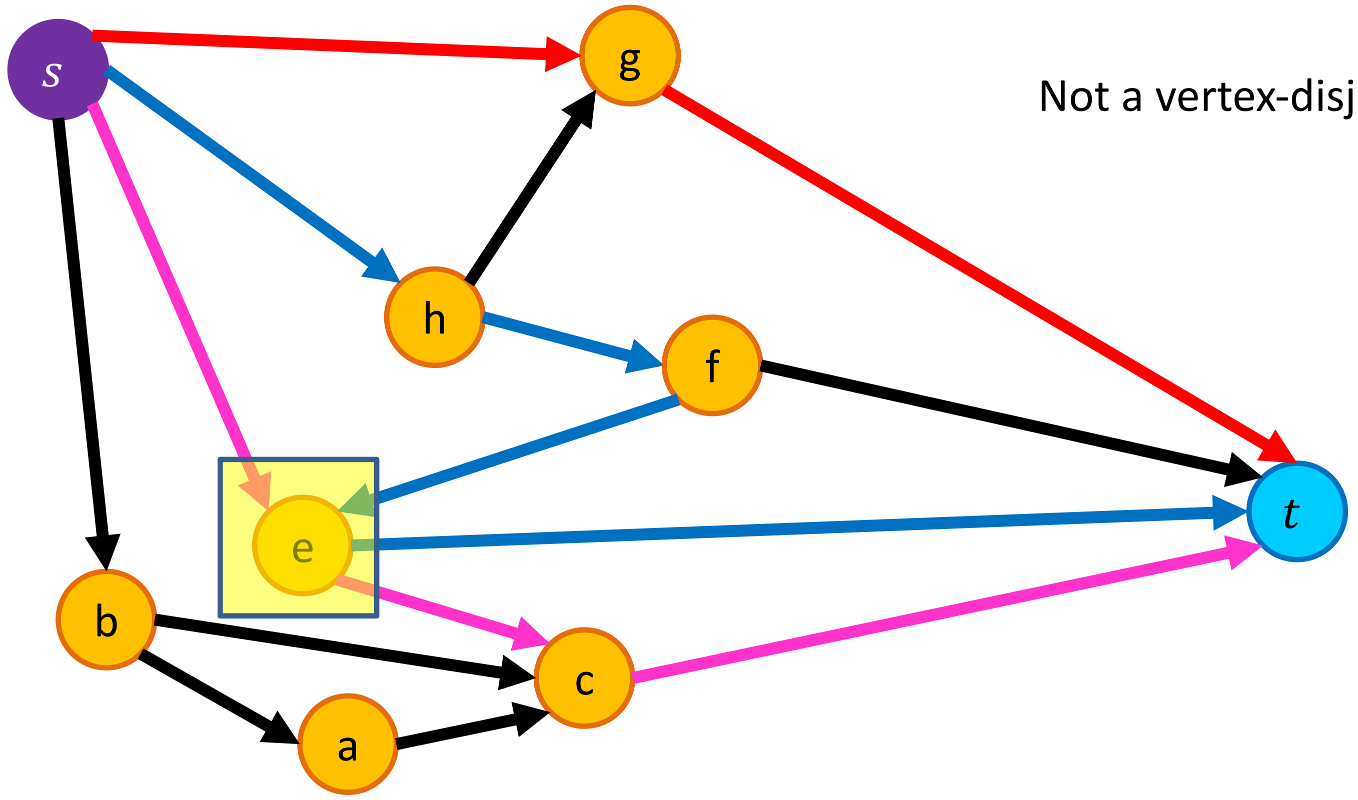
Vertex-Disjoint Paths

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



Vertex-Disjoint Paths

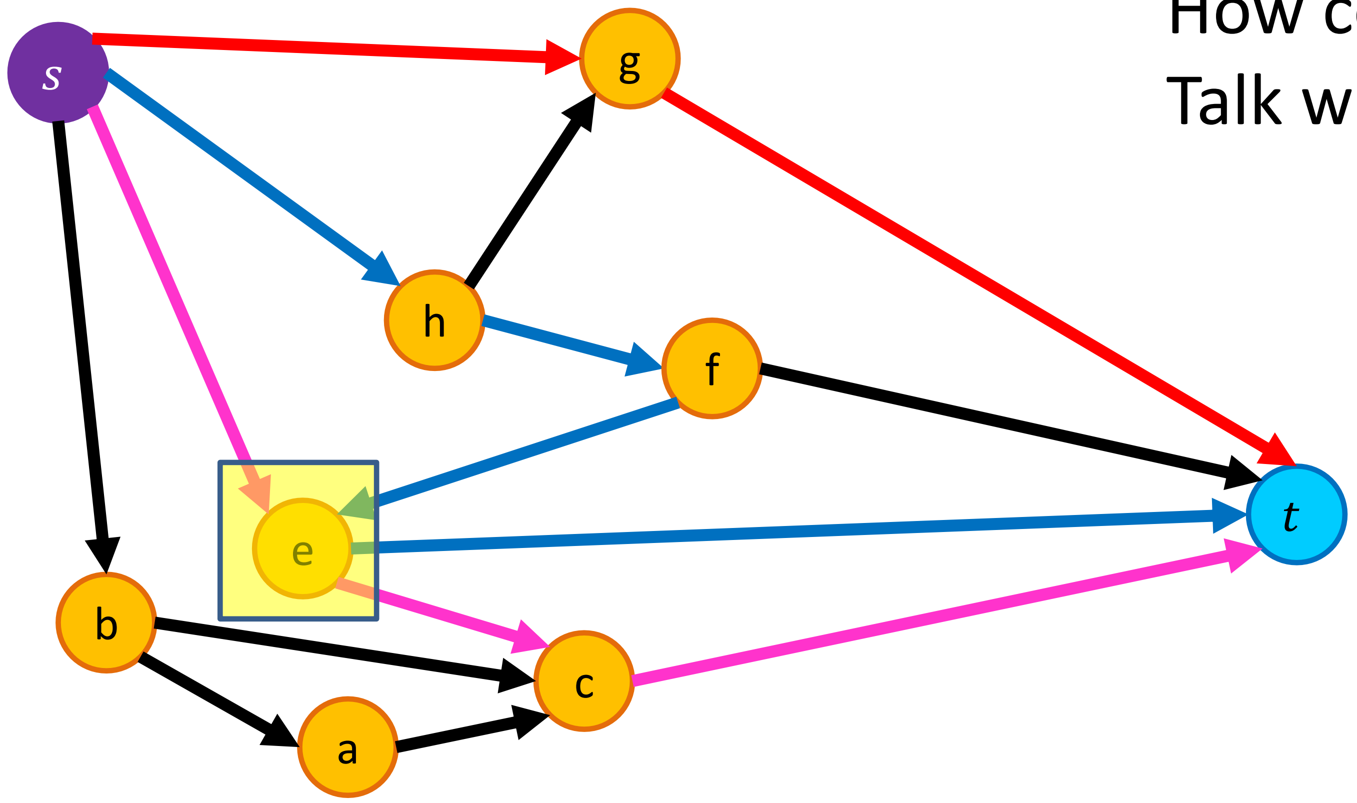
Given a graph $G = (V, E)$, a start node s and a destination node t, give the maximum number of paths from s to t which share no vertices



Not a vertex-disjoint path!

Vertex-Disjoint Paths

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



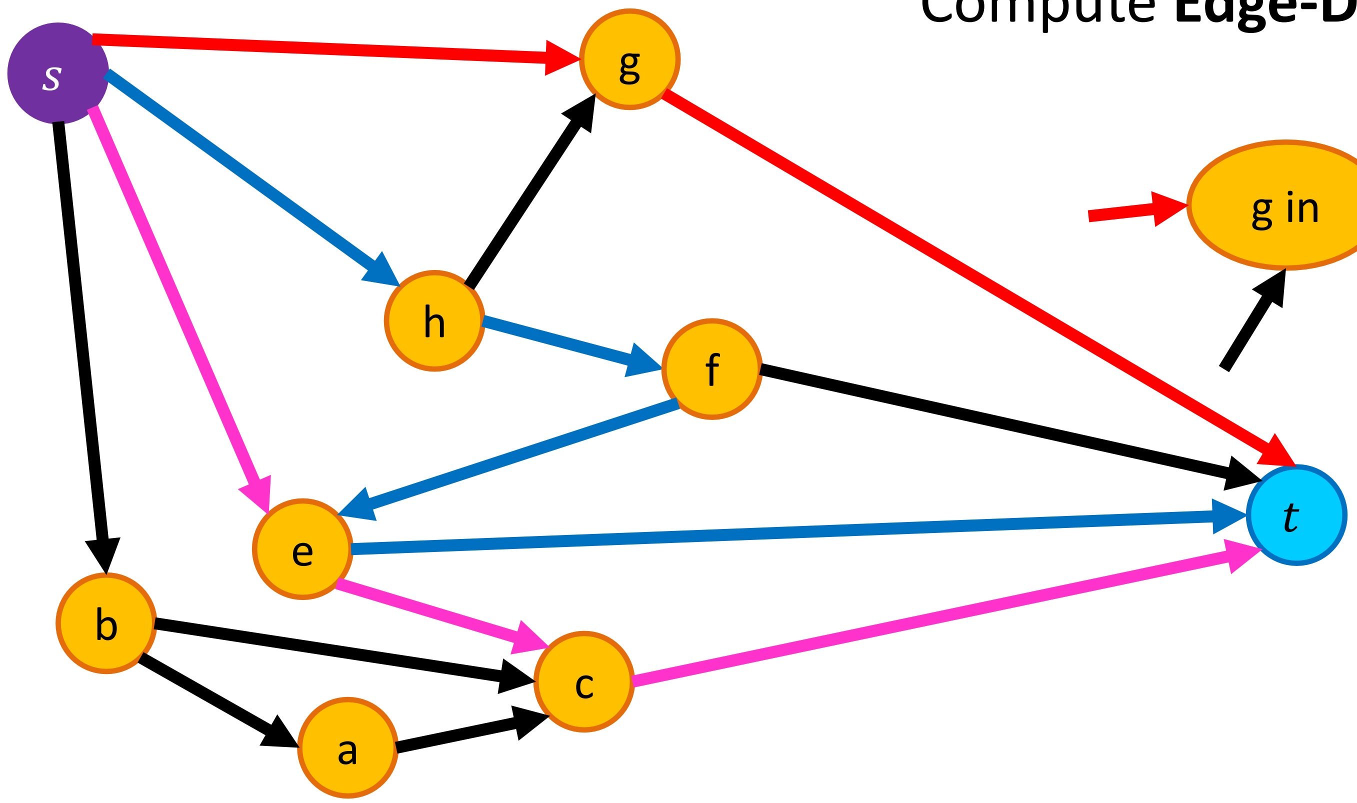
How could we solve this?
Talk with your neighbors!

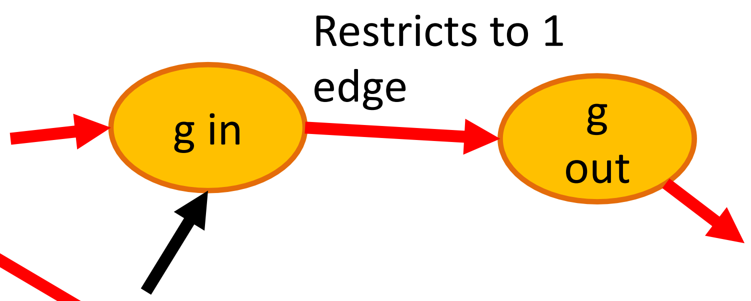
Vertex-Disjoint Paths Algorithm

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

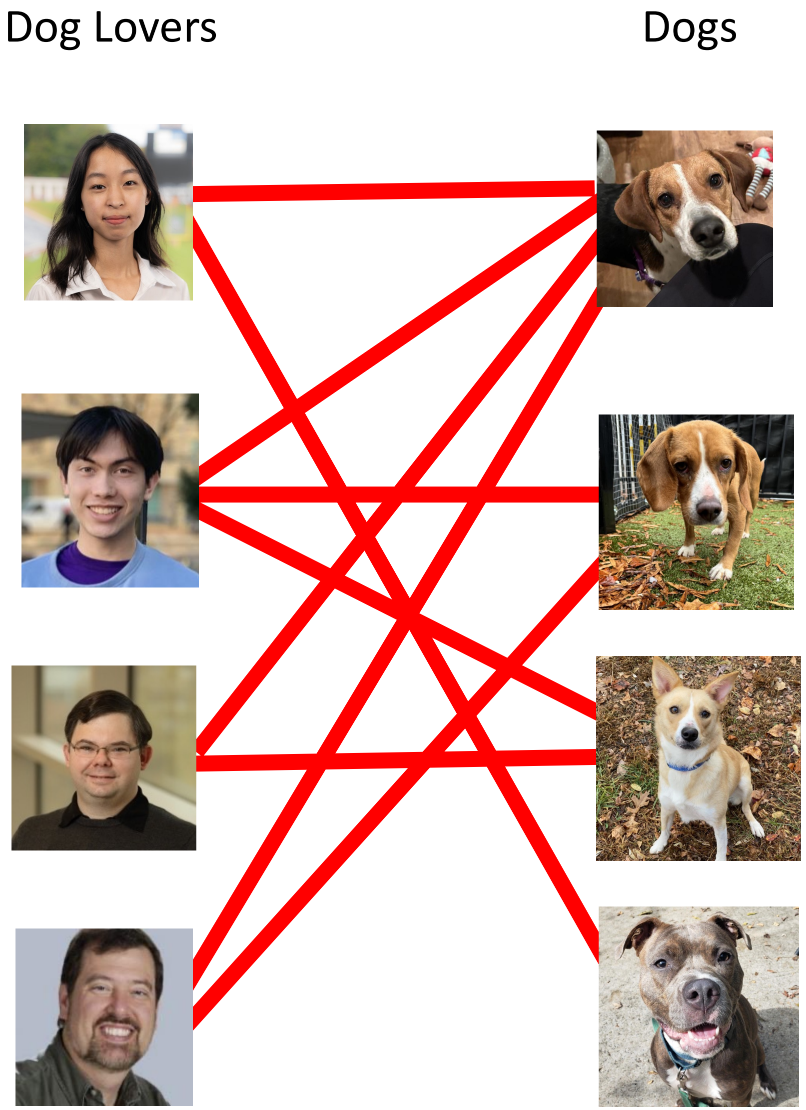
Make two copies of each node, one connected to incoming edges, the other to outgoing edges

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





Maximum Bipartite Matching



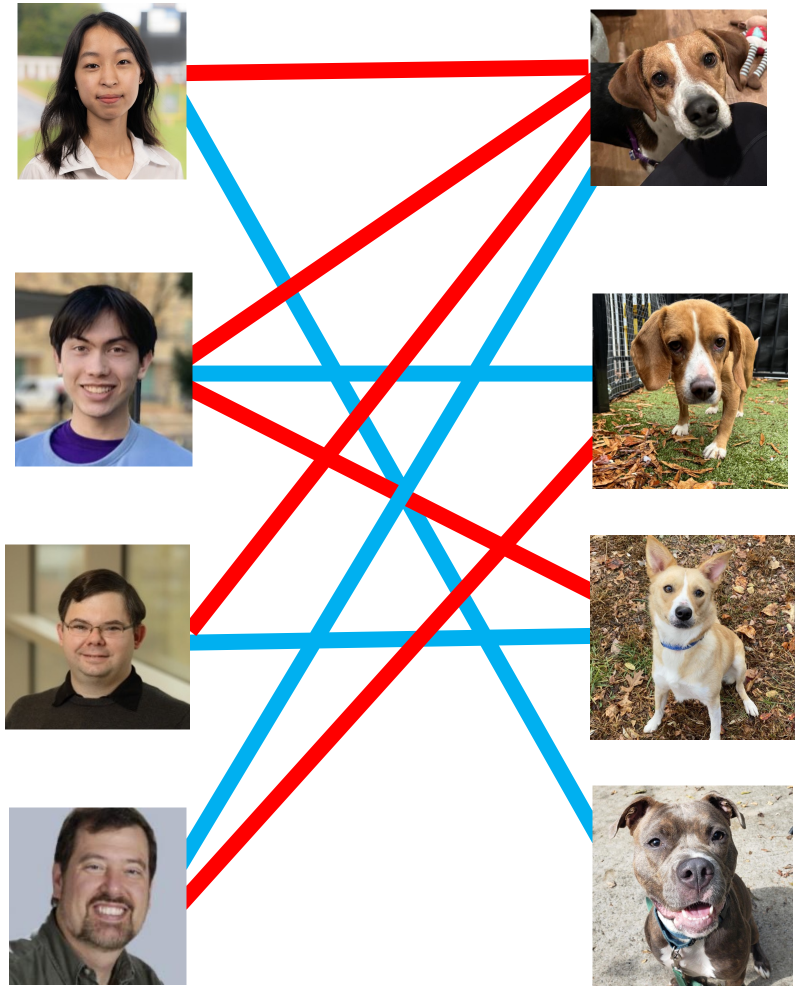
Maximum Bipartite Matching

Dog Lovers

Dogs

Maximum Bipartite Matching

Dog Lovers Dogs



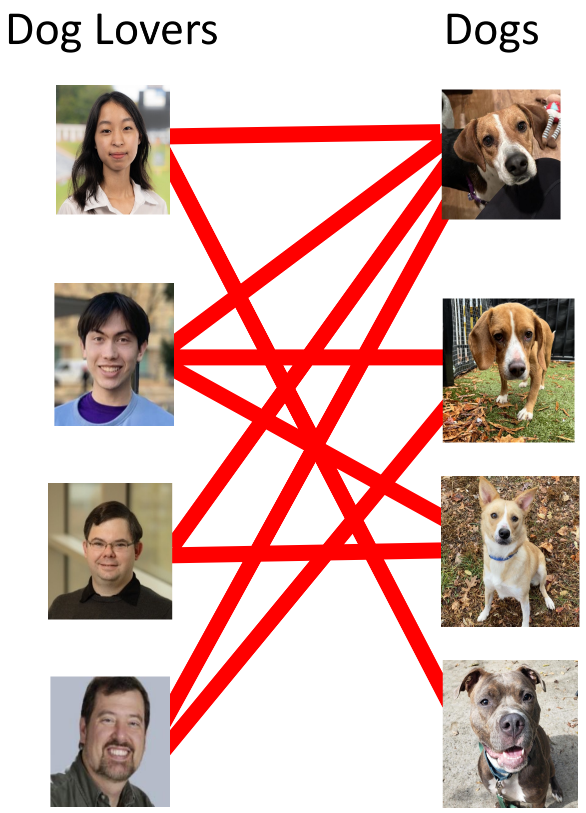
Maximum Bipartite Matching

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



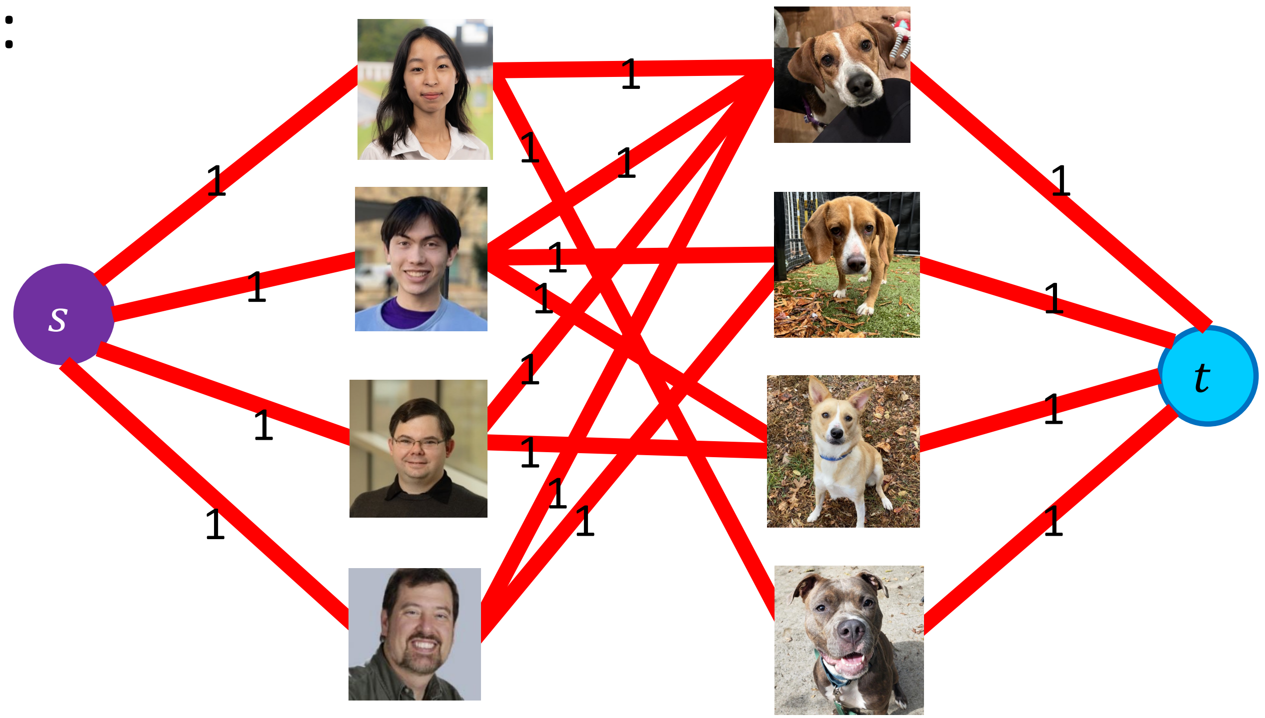
How could we solve this?

Talk with your neighbors!

Maximum Bipartite Matching Using Max Flow

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

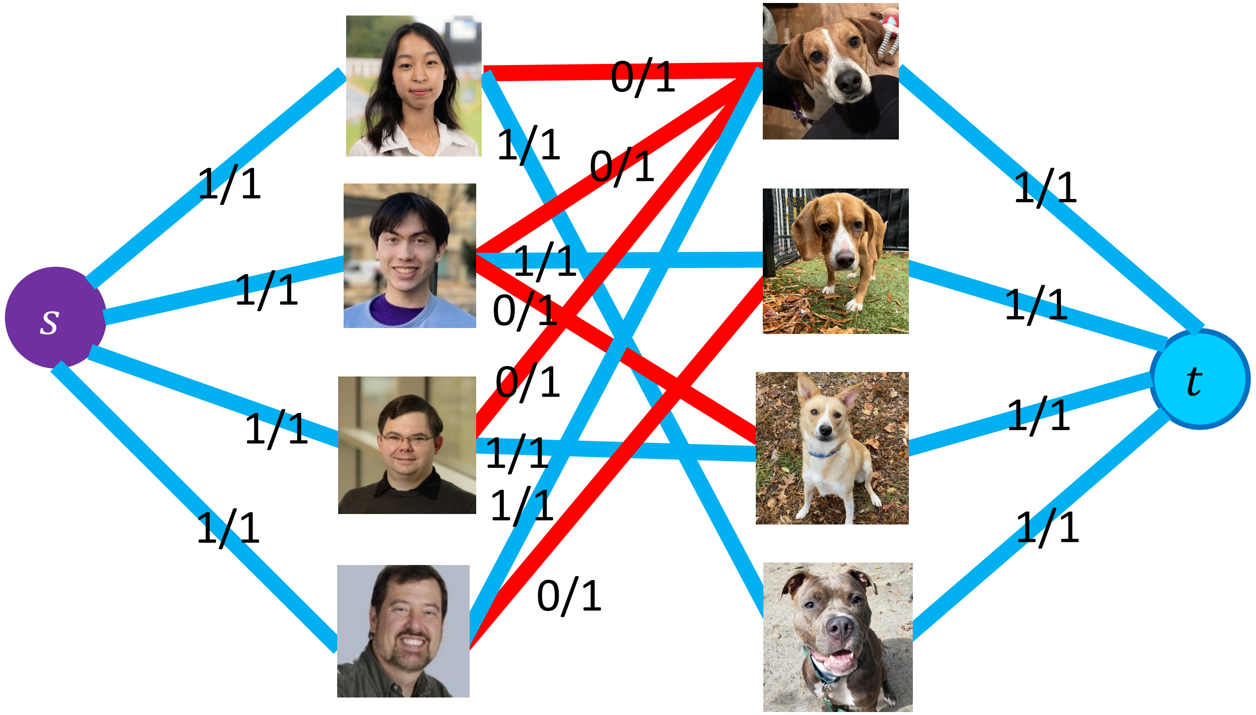
- Adding in a source and sink to the set of nodes:
  - $V' = L \cup R \cup \{s, t\}$
- Adding an edge from source to L and from R to sink:
  - $E' = E \cup \{u \in L \mid (s, u)\} \cup \{v \in r \mid (v, t)\}$
- Make each edge capacity 1:
  - $\forall e \in E', c(e) = 1$



Maximum Bipartite Matching Using Max Flow

$\Theta(E \cdot V)$

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



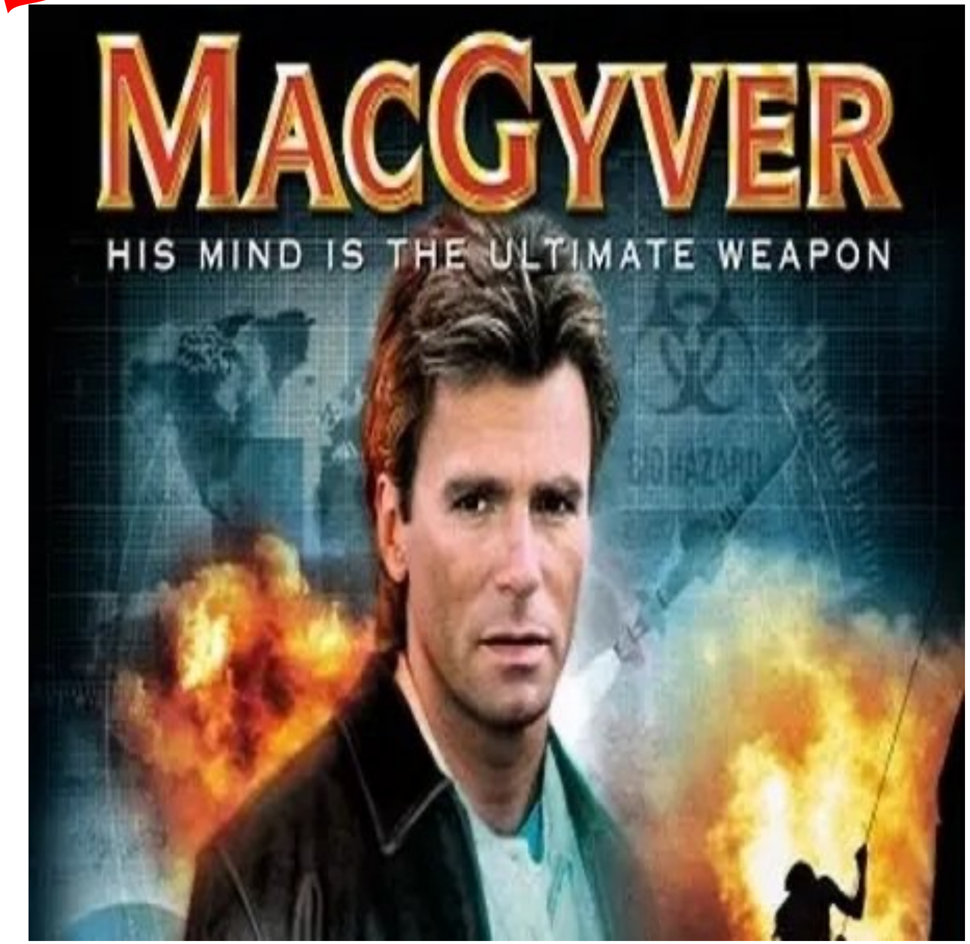
Reductions

- Algorithm technique of supreme ultimate power
- Convert instance of problem A to an instance of Problem B
- Convert solution of problem B back to a solution of problem A

Reductions

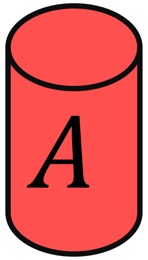
Shows how two different problems relate to each other

MOVIE TIME!



MacGyver's Reduction

Problem we don't know how to solve



Opening a door

Solution for $\boldsymbol{A}$

Keg cannon battering ram

Aim duct at door, insert keg

Put fire under the Keg

Reduction

Problem we do know how to solve

B

Lighting a fire

How?

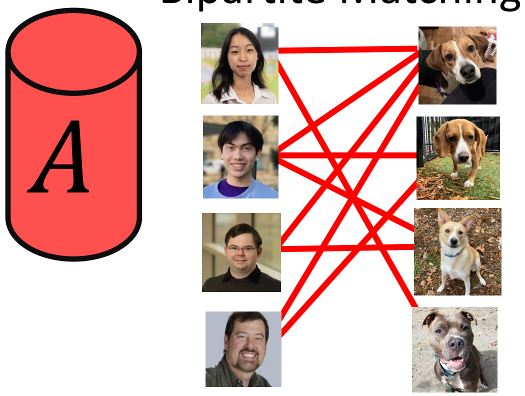
Solution for $\boldsymbol{B}$

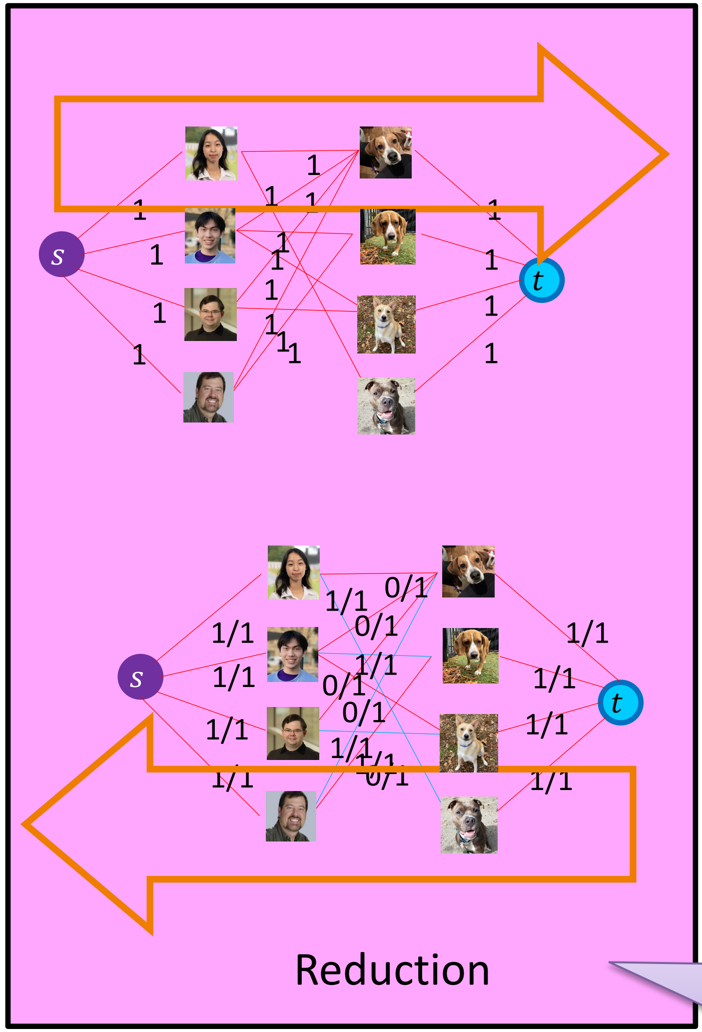
Alcohol, wood, matches

Bipartite Matching Reduction

Problem we don't know how to solve

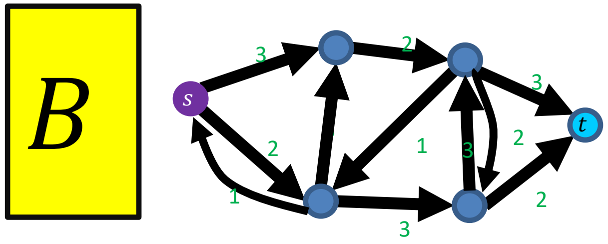
Bipartite Matching



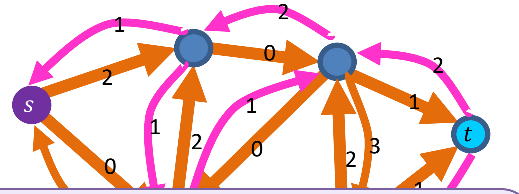


Problem we do know how to solve

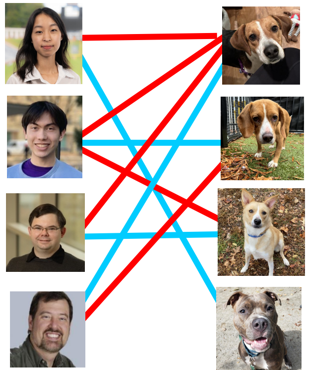
Max Flow



Solution for ***B***

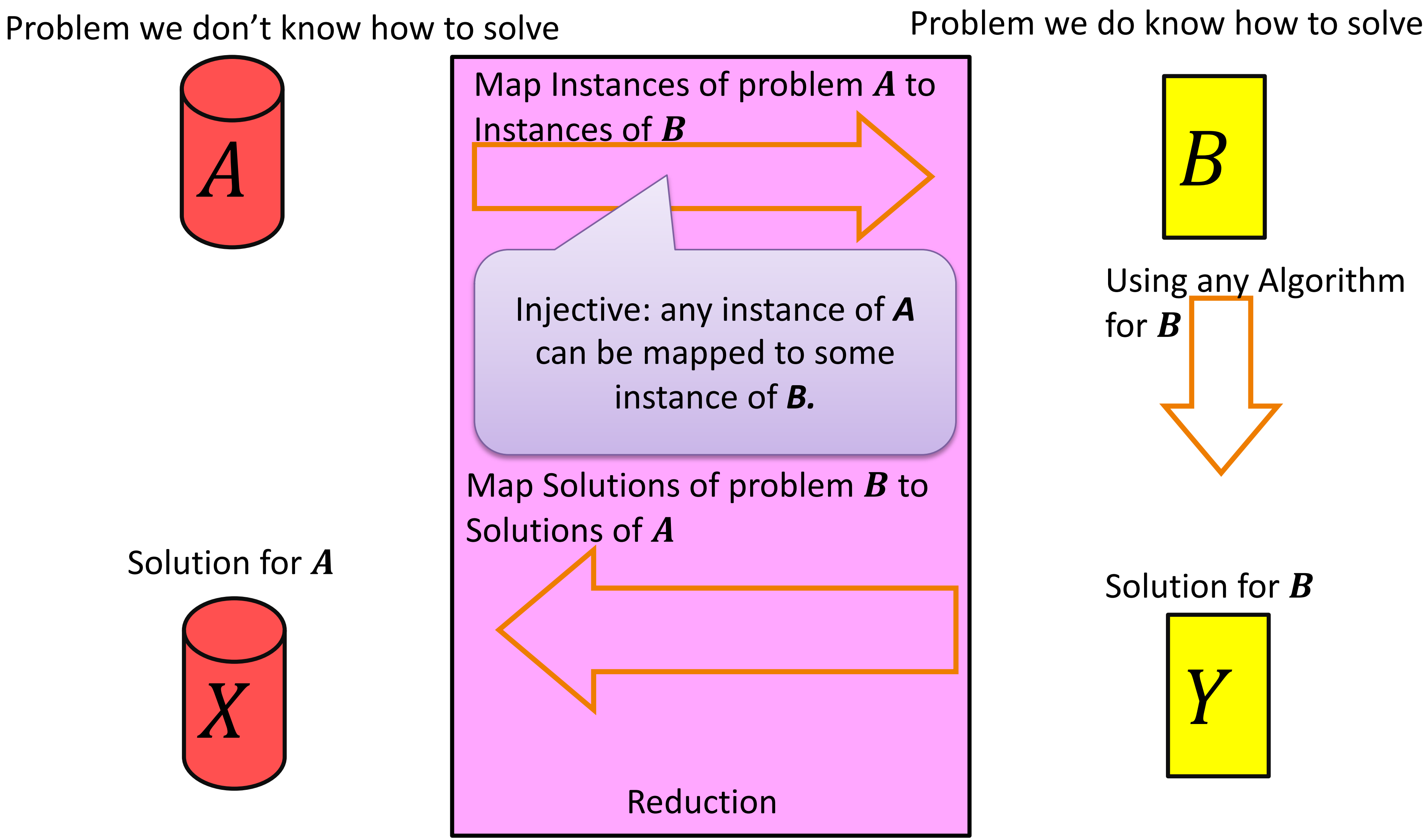


Solution for ***A***

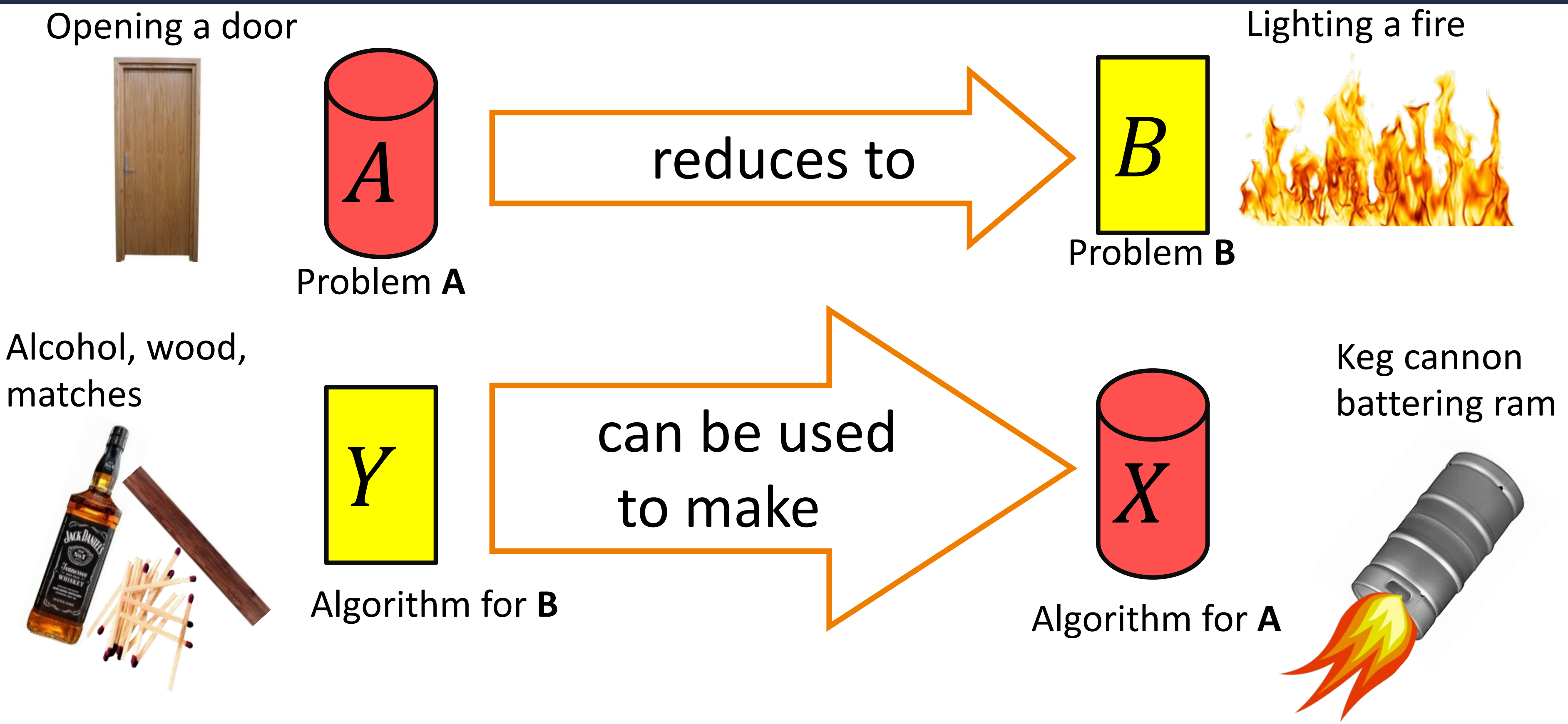


Must show (prove):
1) how to make construction
2) Why it works

In General: Reduction



Worst-case lower-bound Proofs



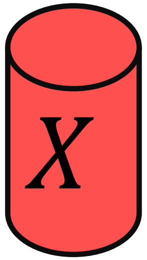
A is not a harder problem than B

$$A \le B$$

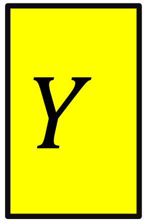
The name "reduces" is confusing: it is in the *opposite* direction of the making

Proof of Lower Bound by Reduction

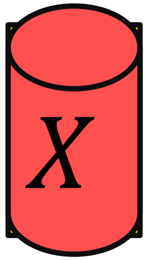
<u>To Show: Y is slow</u>

1. We know X is slow (by a proof)
(e.g., X = some way to open the door)



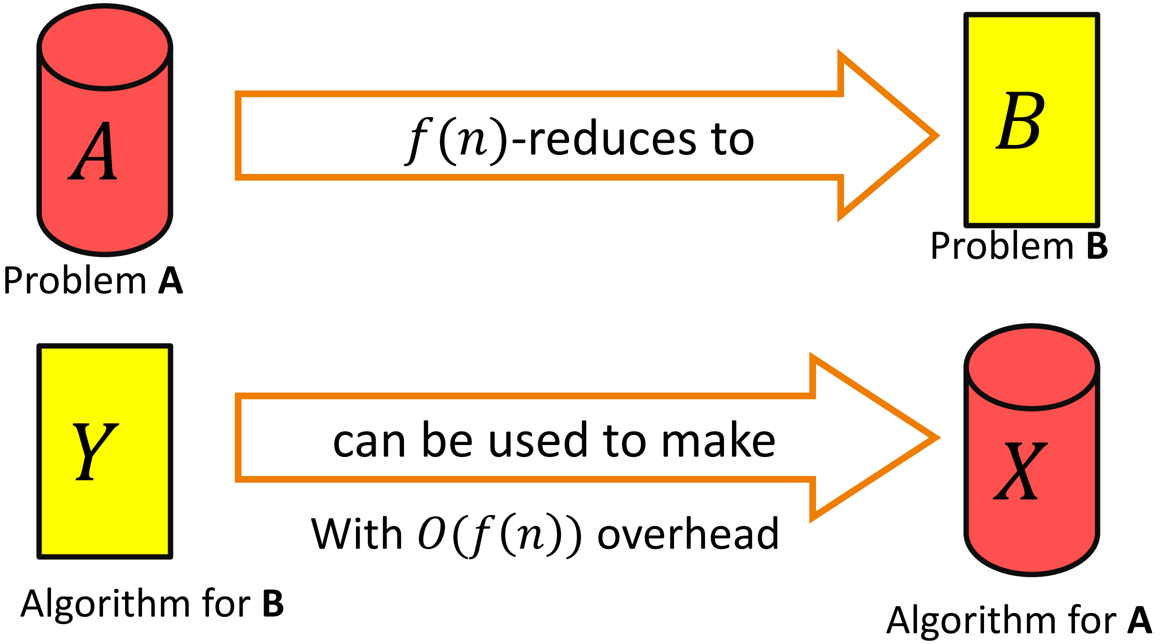
2. Assume Y is quick [toward contradiction]
(Y = some way to light a fire)

3. Show how to use Y to perform X quickly

4. X is slow, but Y could be used to perform X quickly
conclusion: Y must not actually be quick

Reduction Proof Notation



A is not a harder problem than B

$$A \le B$$

If A requires time $\Omega(f(n))$ time then B also requires $\Omega(f(n))$ time

$$A \le_{f(n)} B$$