

# BLG 336E

## Analysis of Algorithms II

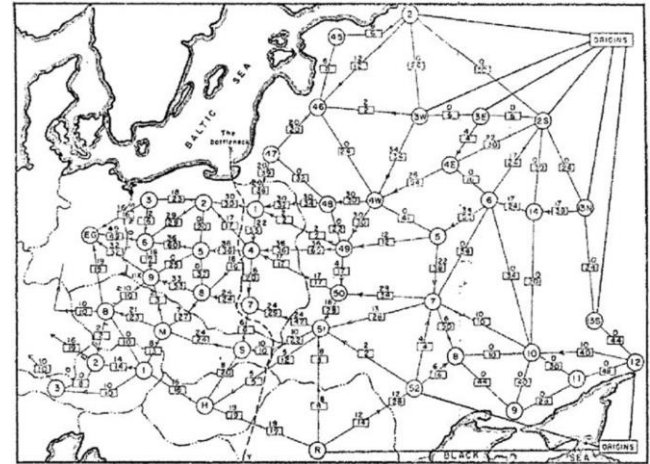
Lecture 12:

### **Polynomial Time, Reductions**

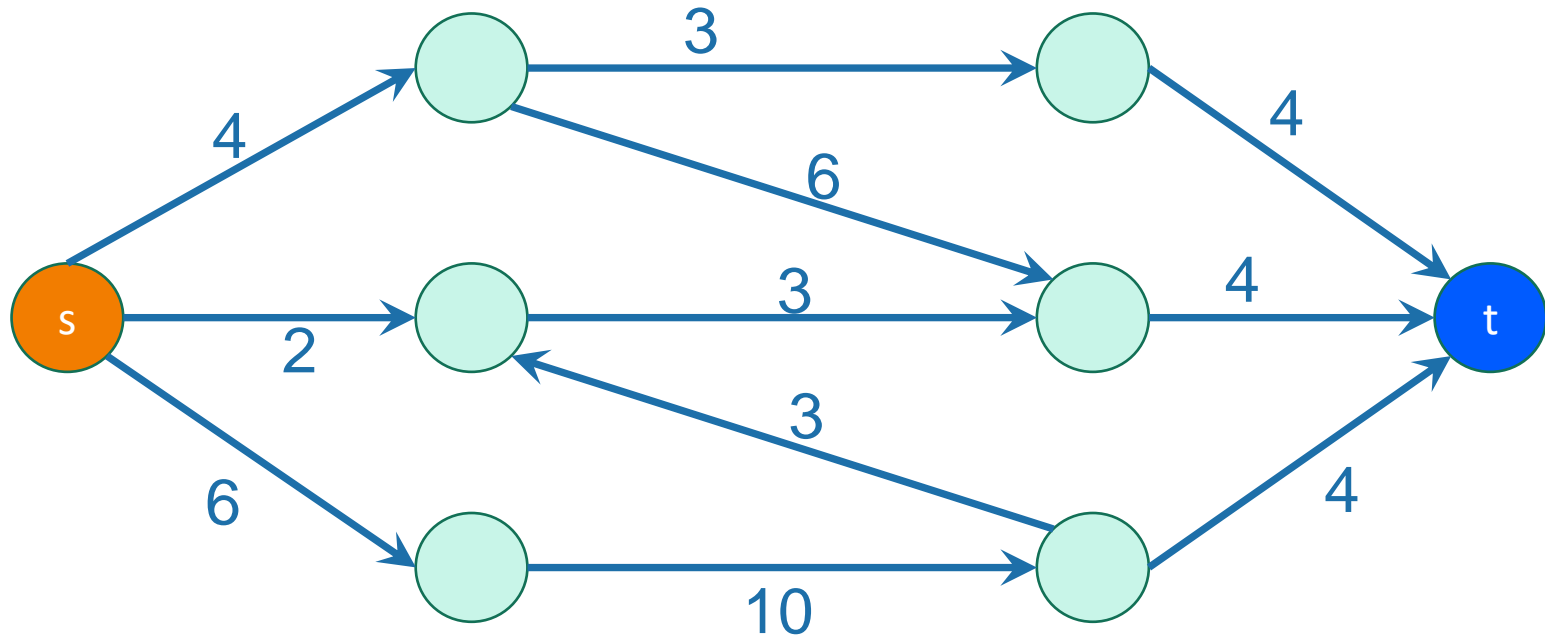
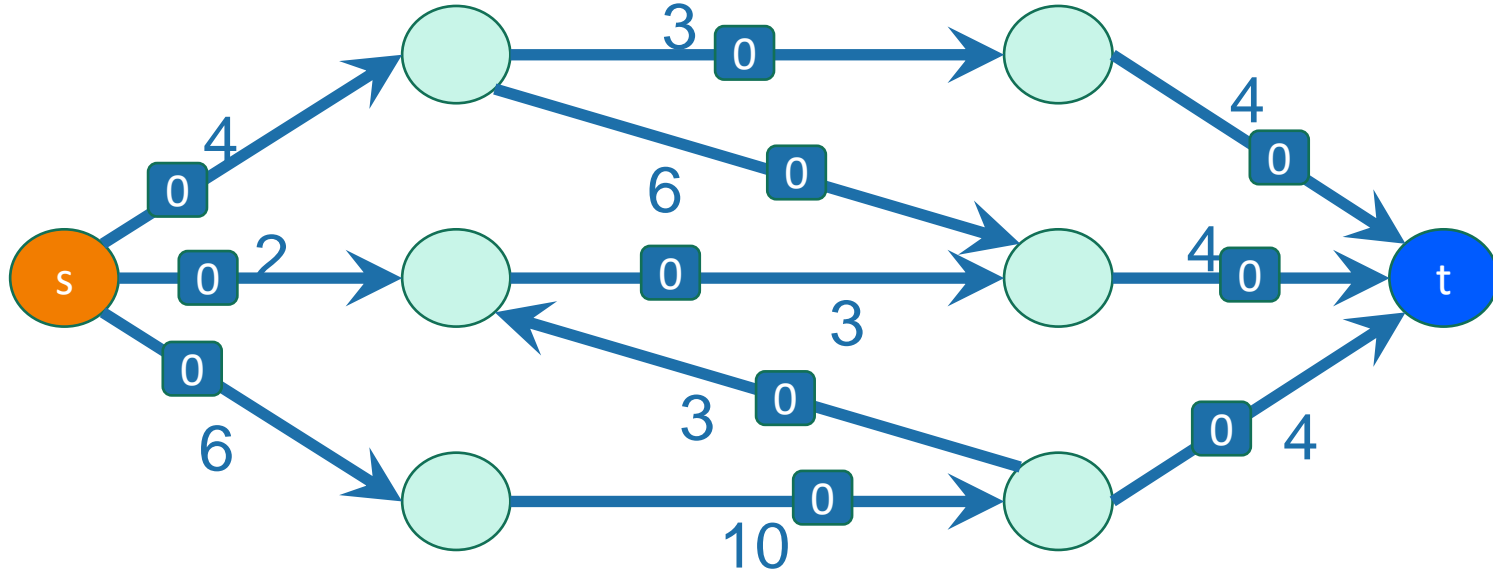
Independent Sets, Vertex Cover, Set Cover  
Satisfiability, Hamiltonian Cycles

# What have we learned?

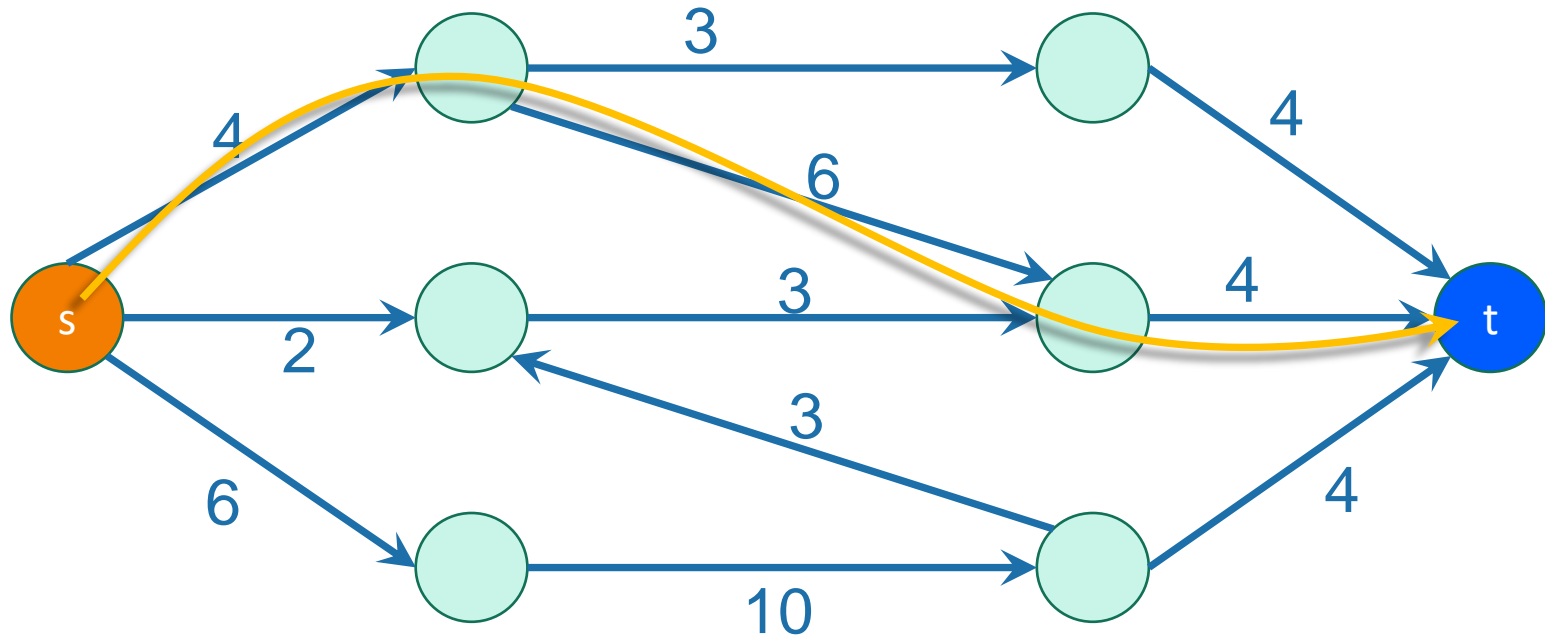
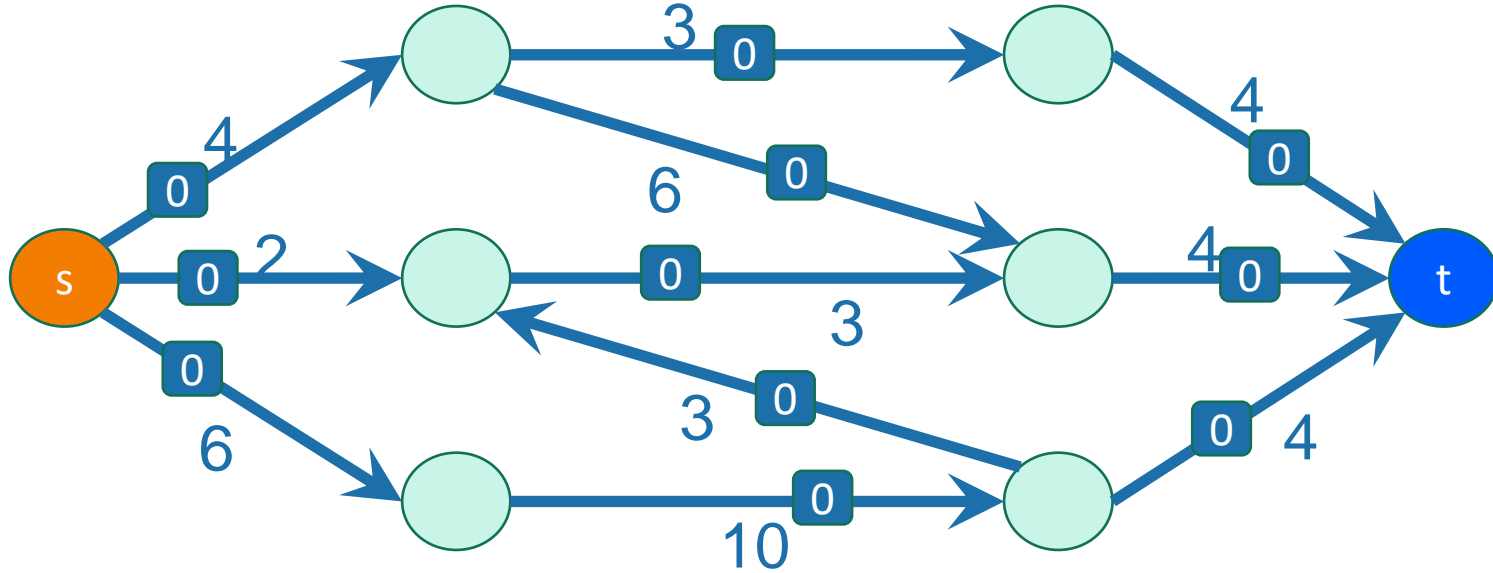
- Max s-t flow is equal to min s-t cut!
  - The USSR and the USA were trying to solve the same problem...
- The Ford-Fulkerson algorithm can find the min-cut/max-flow.
  - Repeatedly improve your flow along an augmenting path.
- How long does this take???



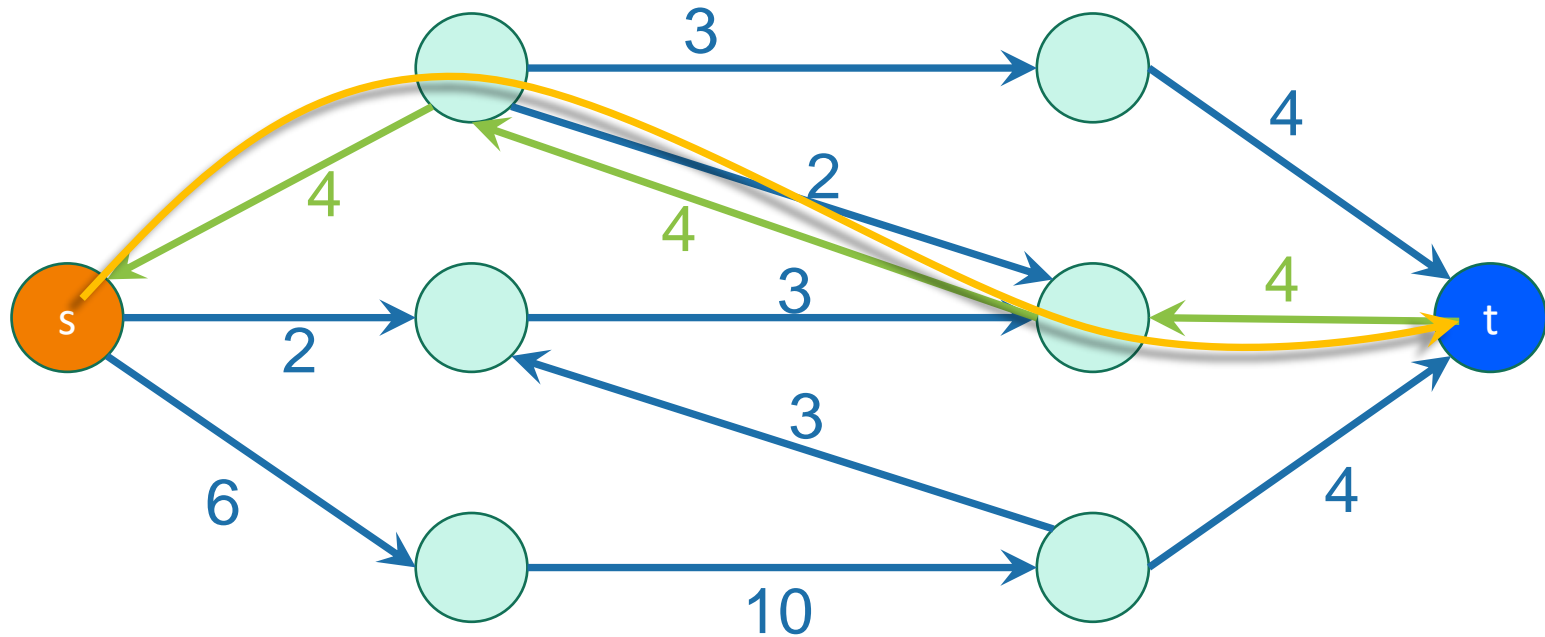
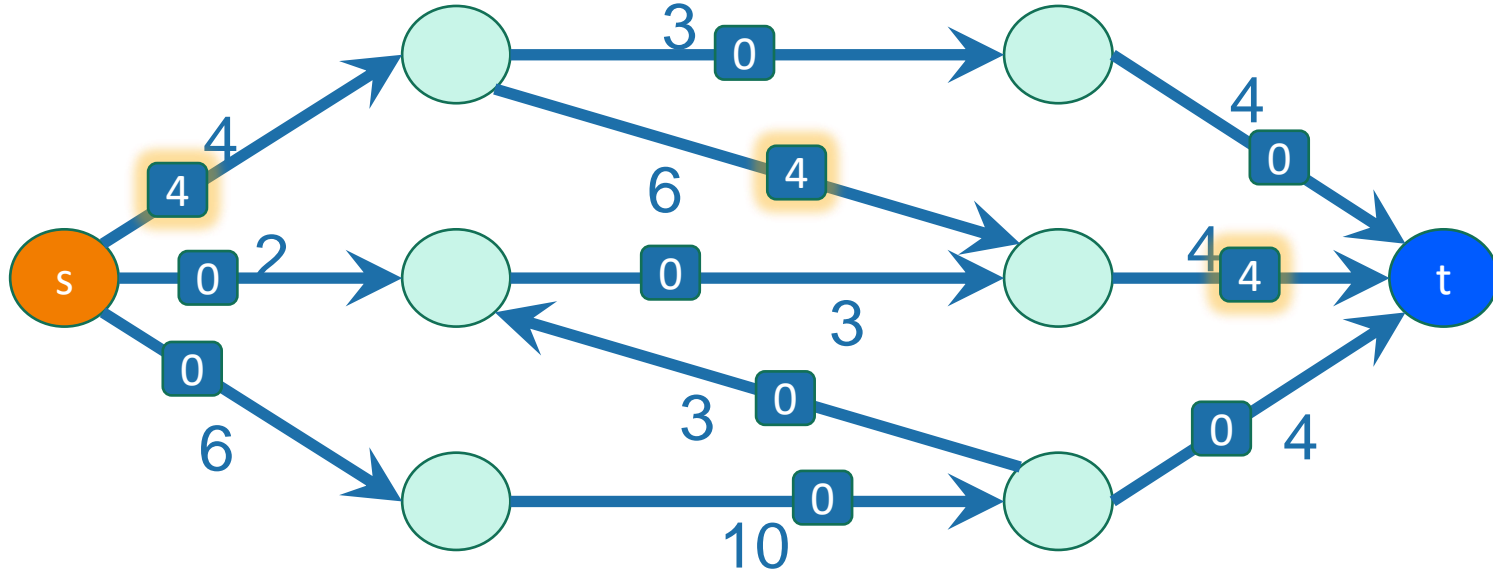
# Example of Ford-Fulkerson



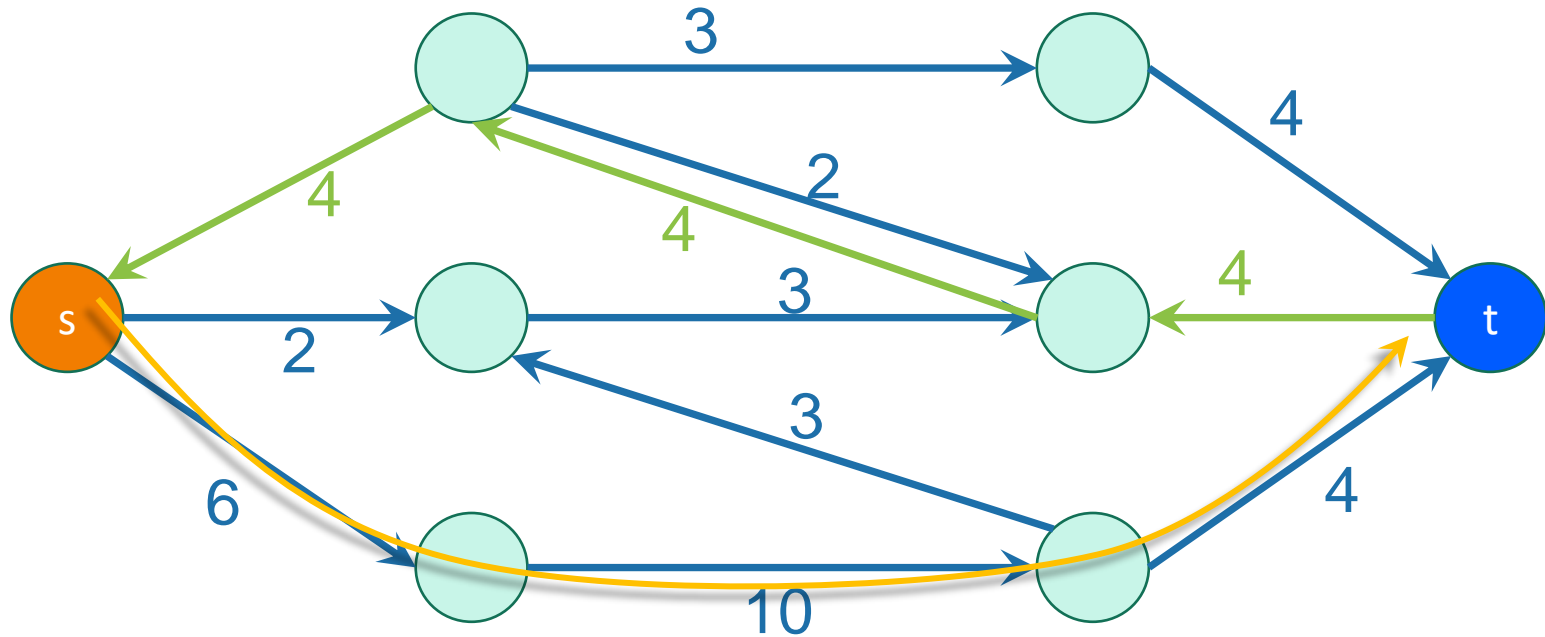
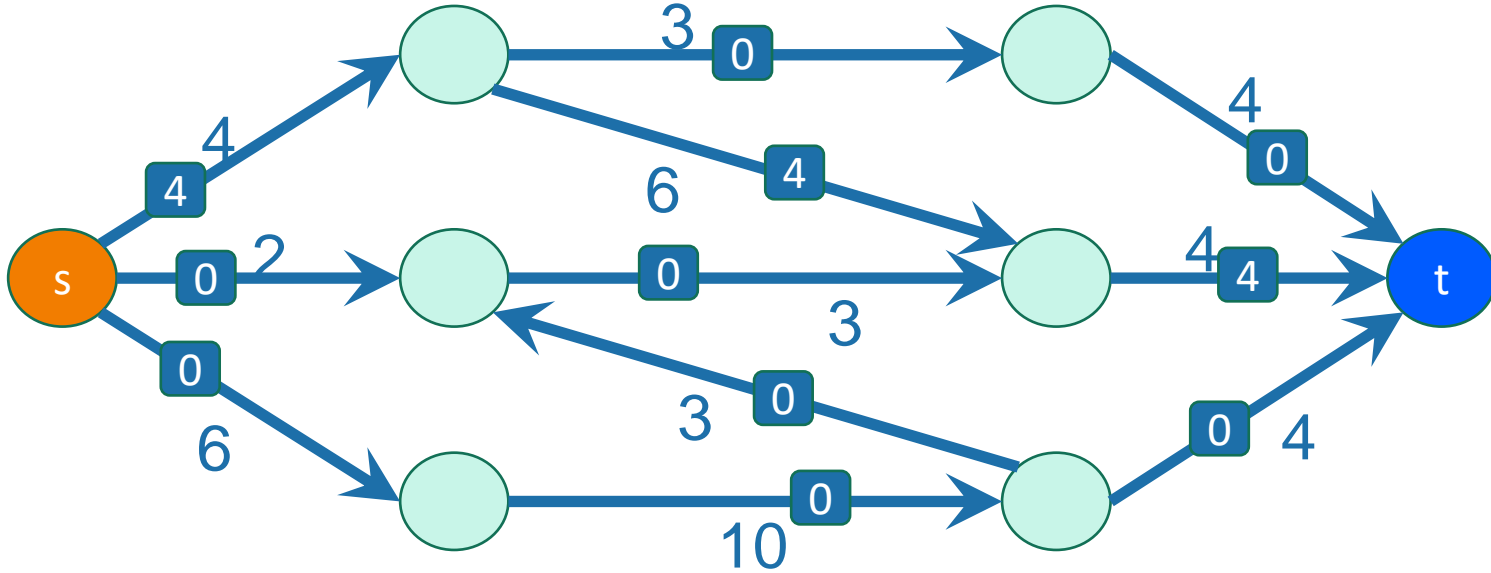
# Example of Ford-Fulkerson



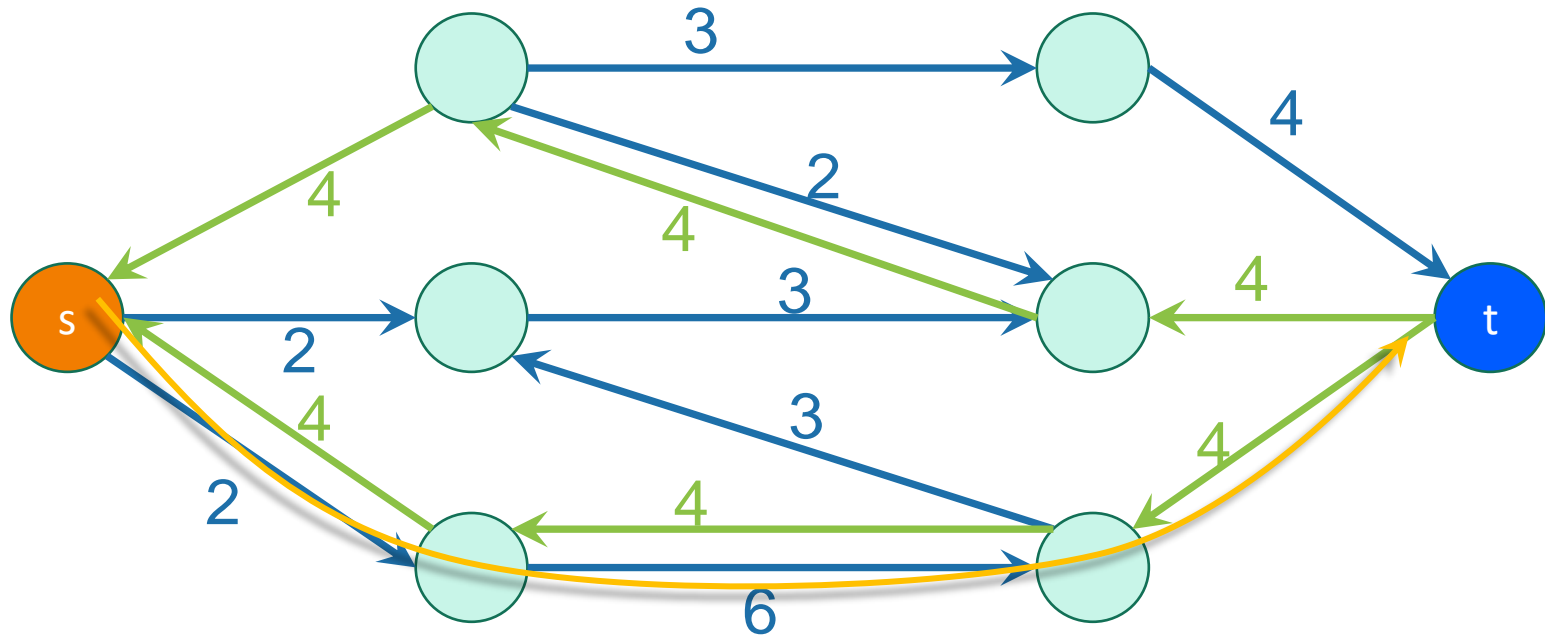
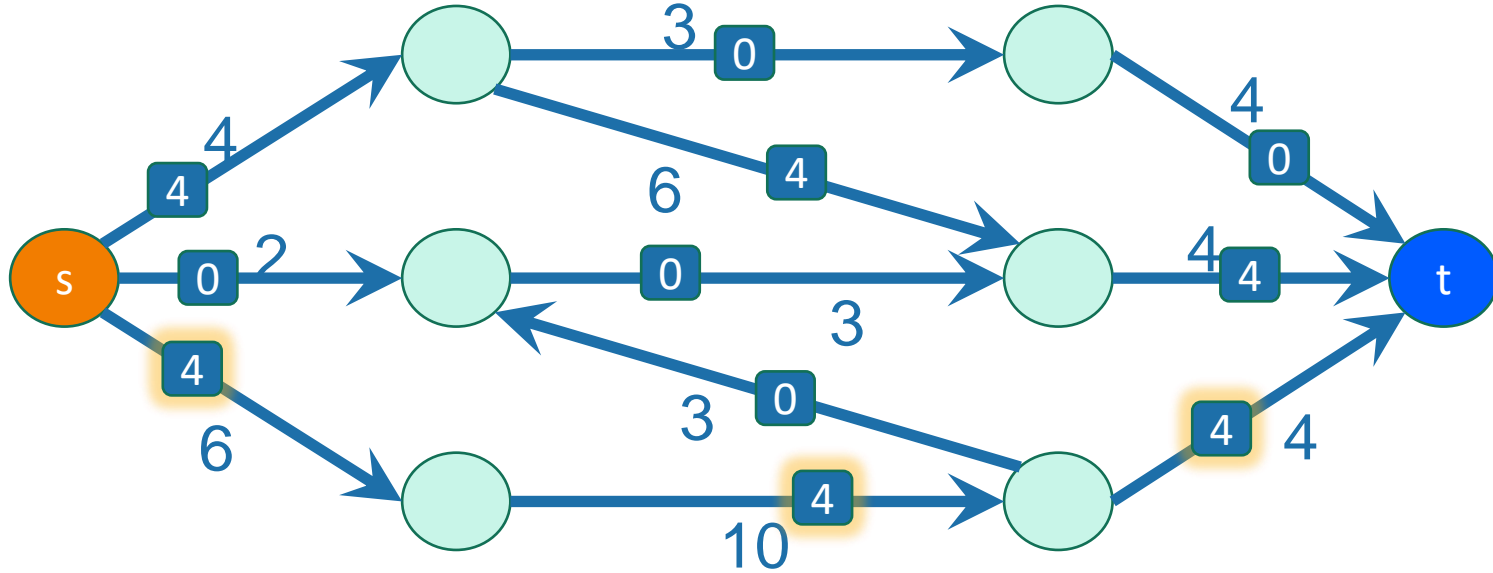
# Example of Ford-Fulkerson



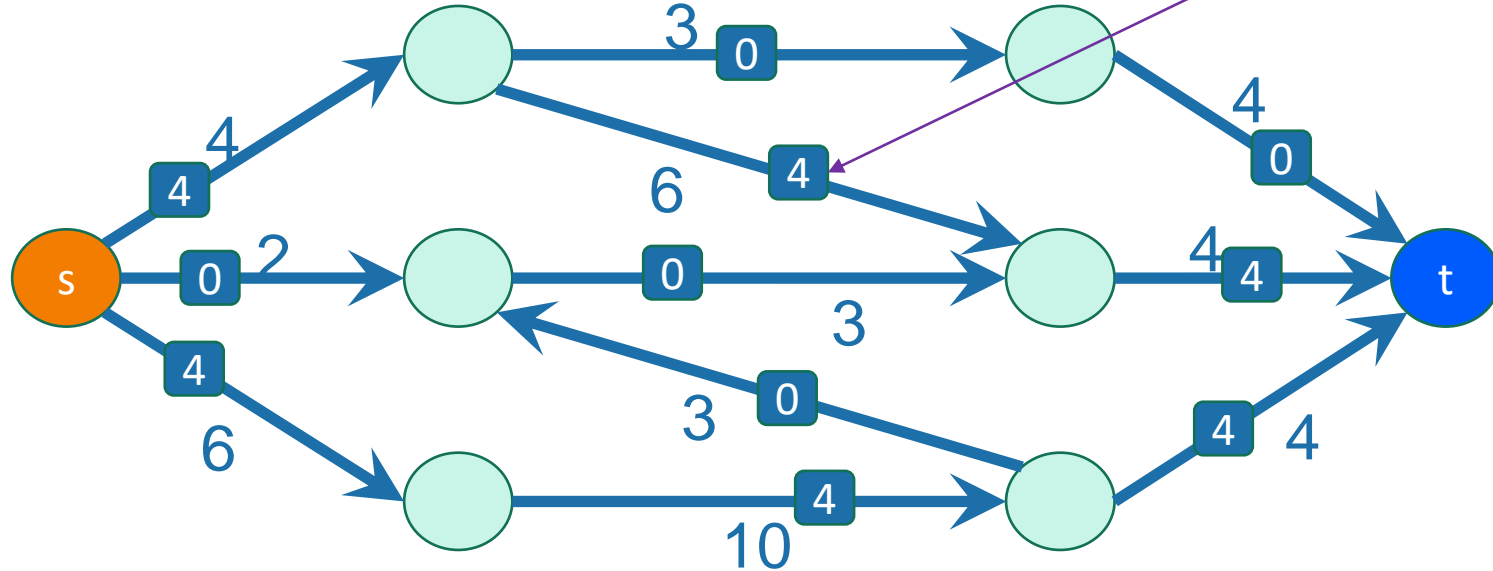
# Example of Ford-Fulkerson



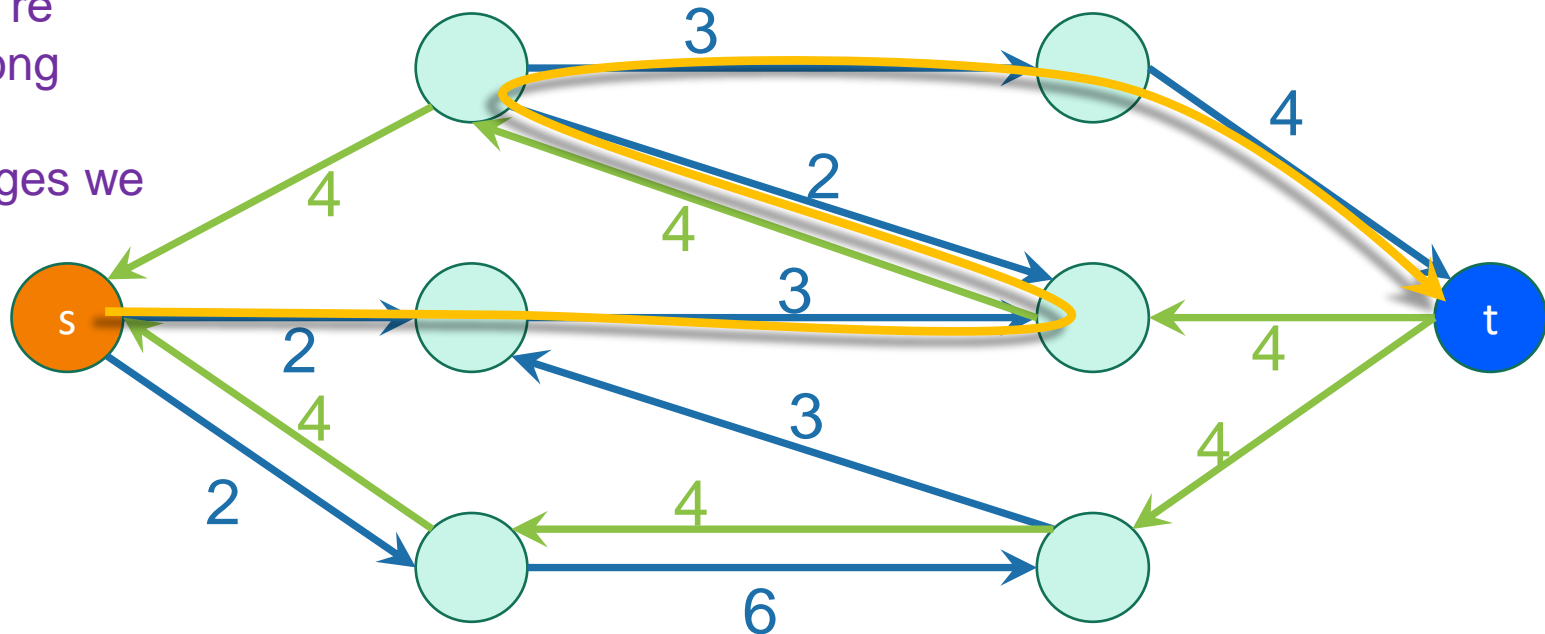
# Example of Ford-Fulkerson



# Example of Ford-Fulkerson

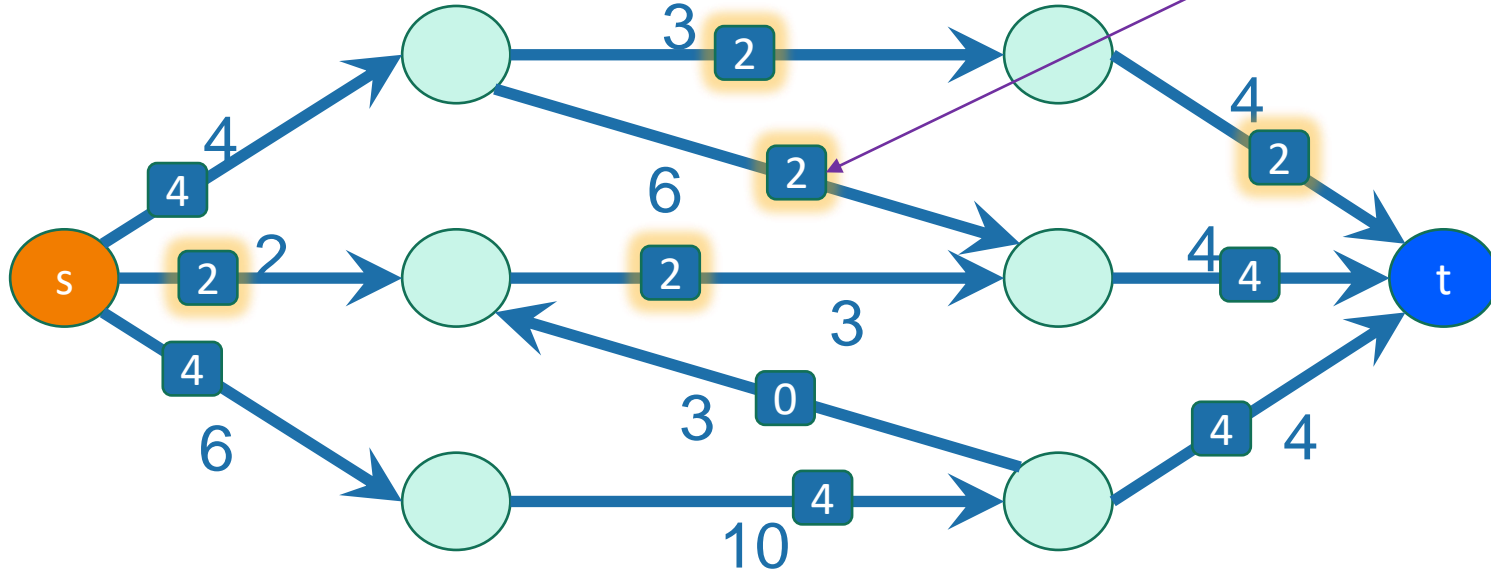


Notice that we're going back along one of the backwards edges we added.

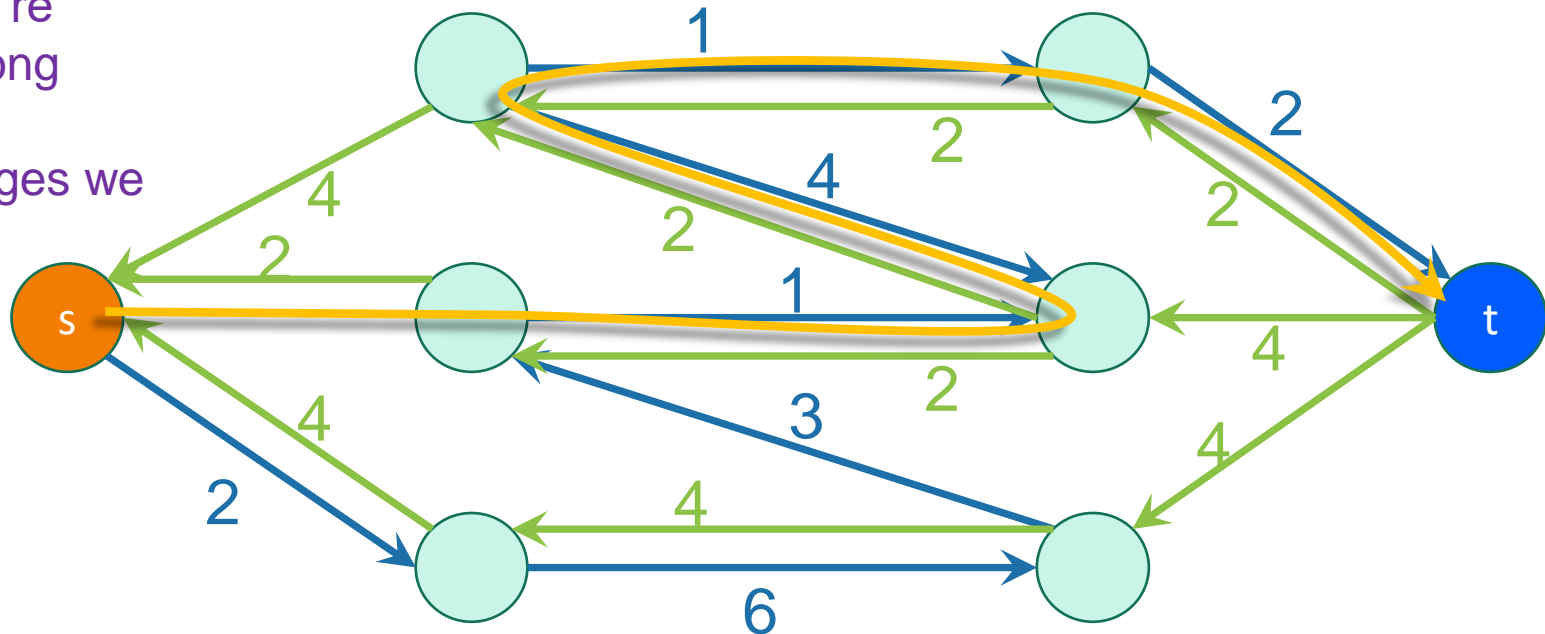




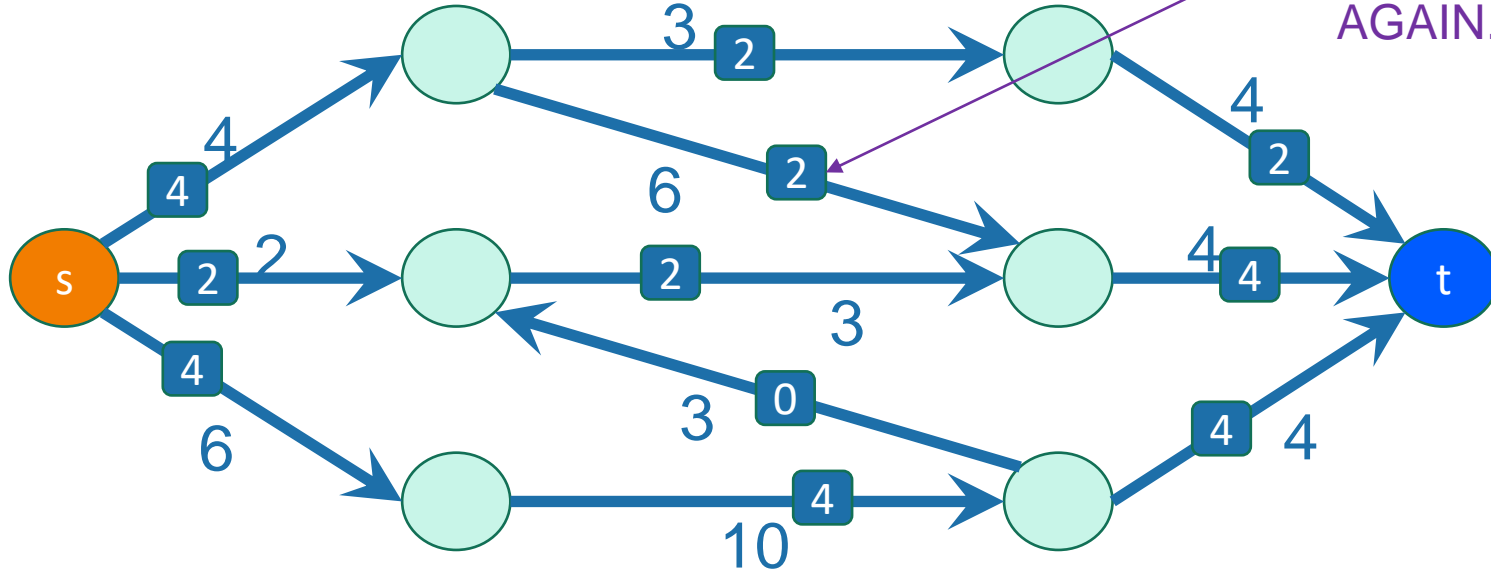
# Example of Ford-Fulkerson



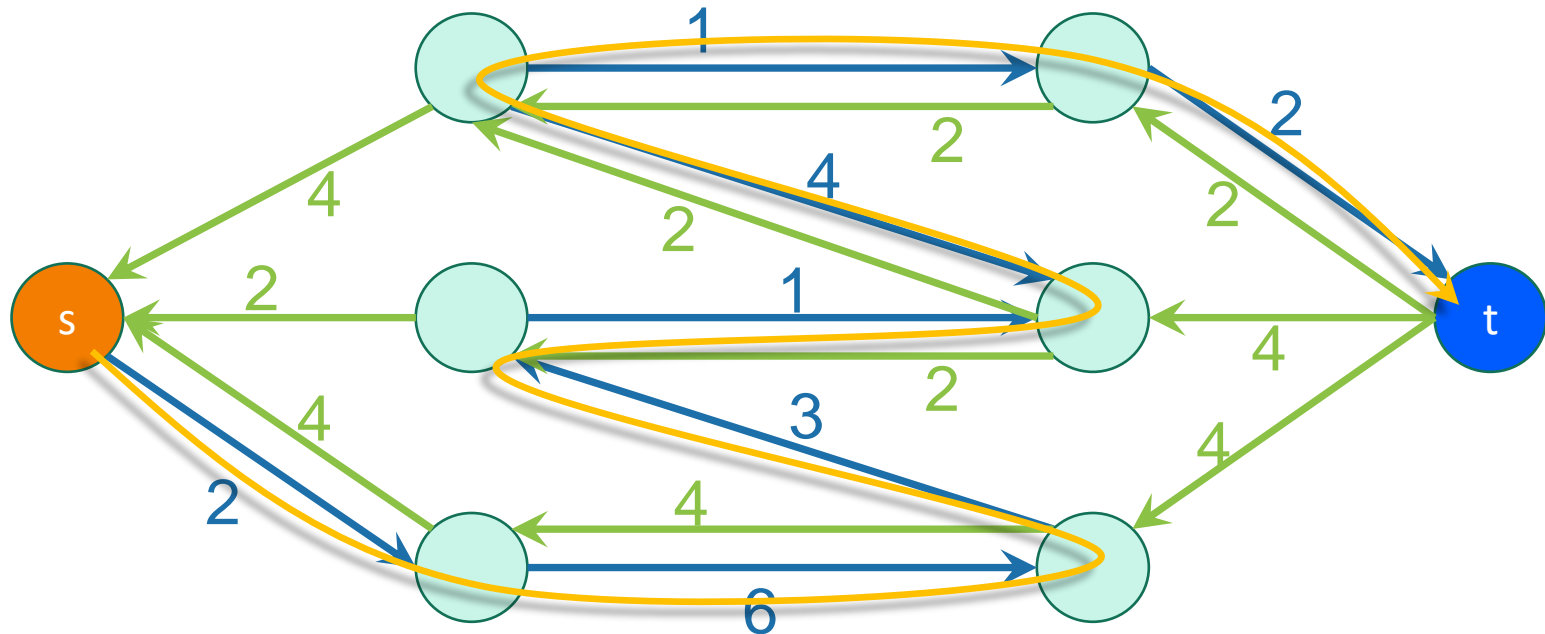
Notice that we're going back along one of the backwards edges we added.



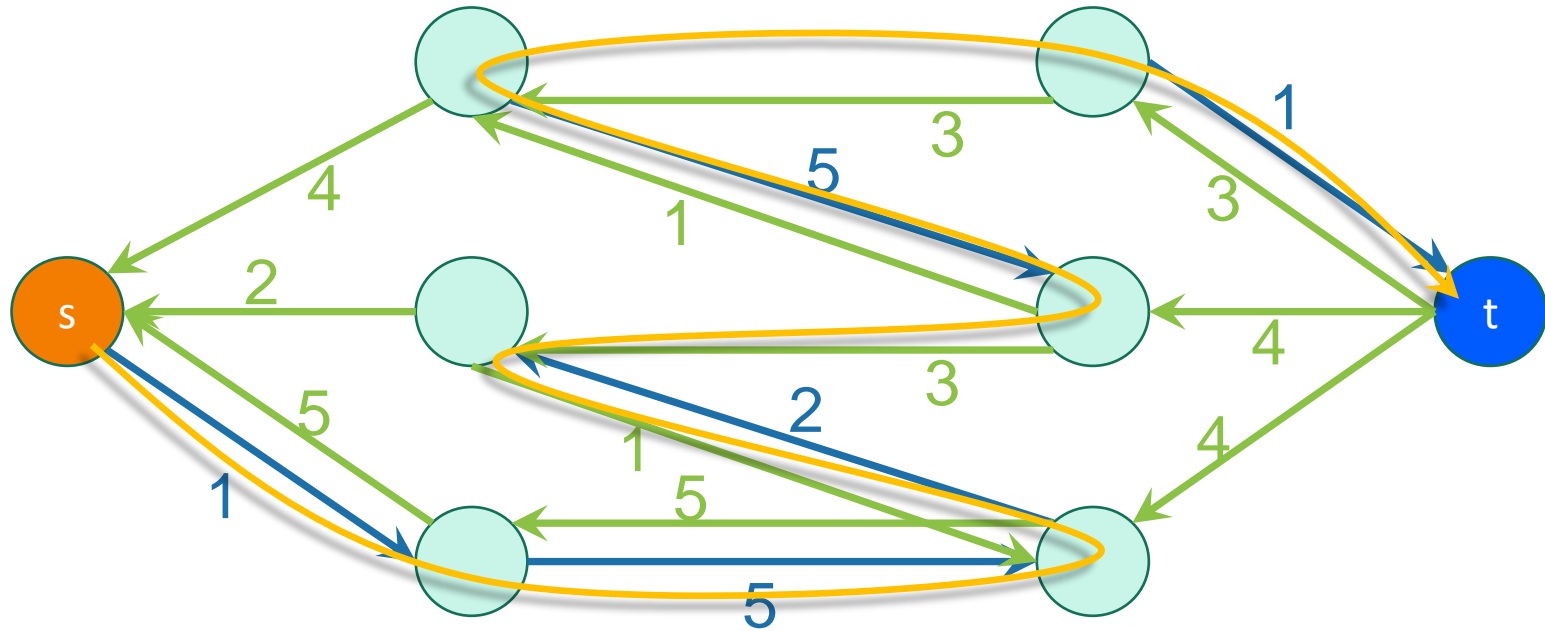
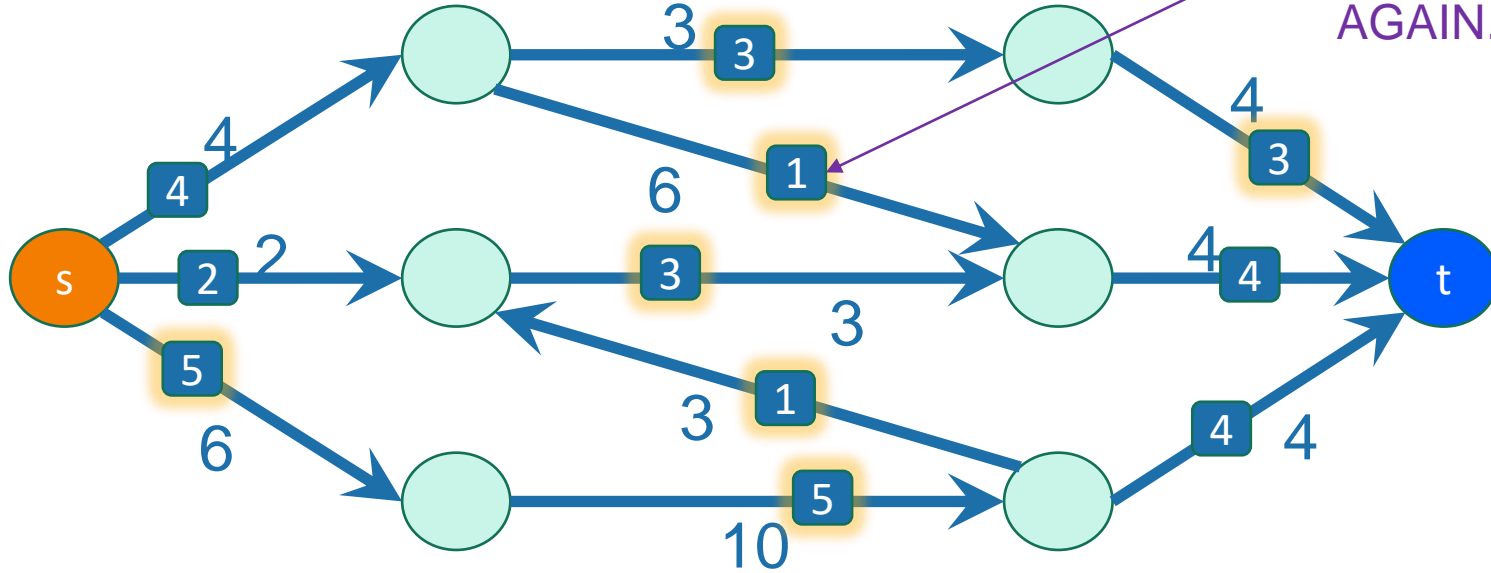
# Example of Ford-Fulkerson



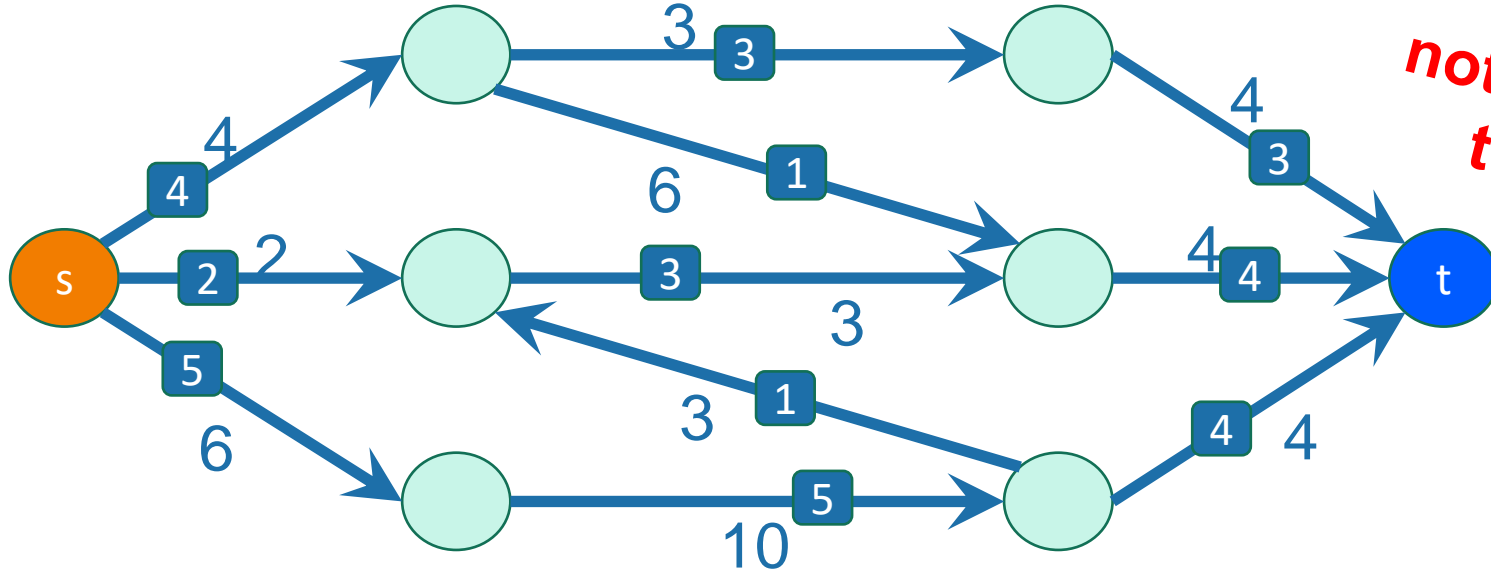
We will remove flow from this edge AGAIN.



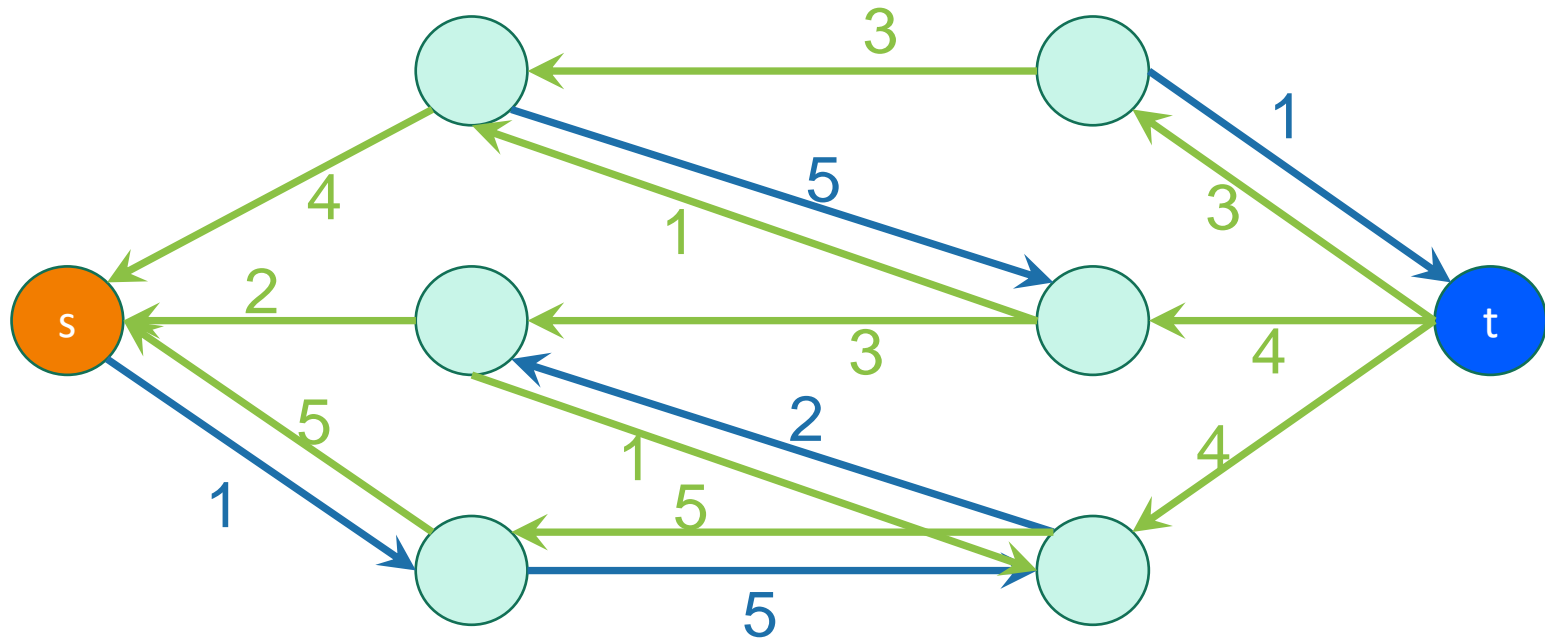
# Example of Ford-Fulkerson



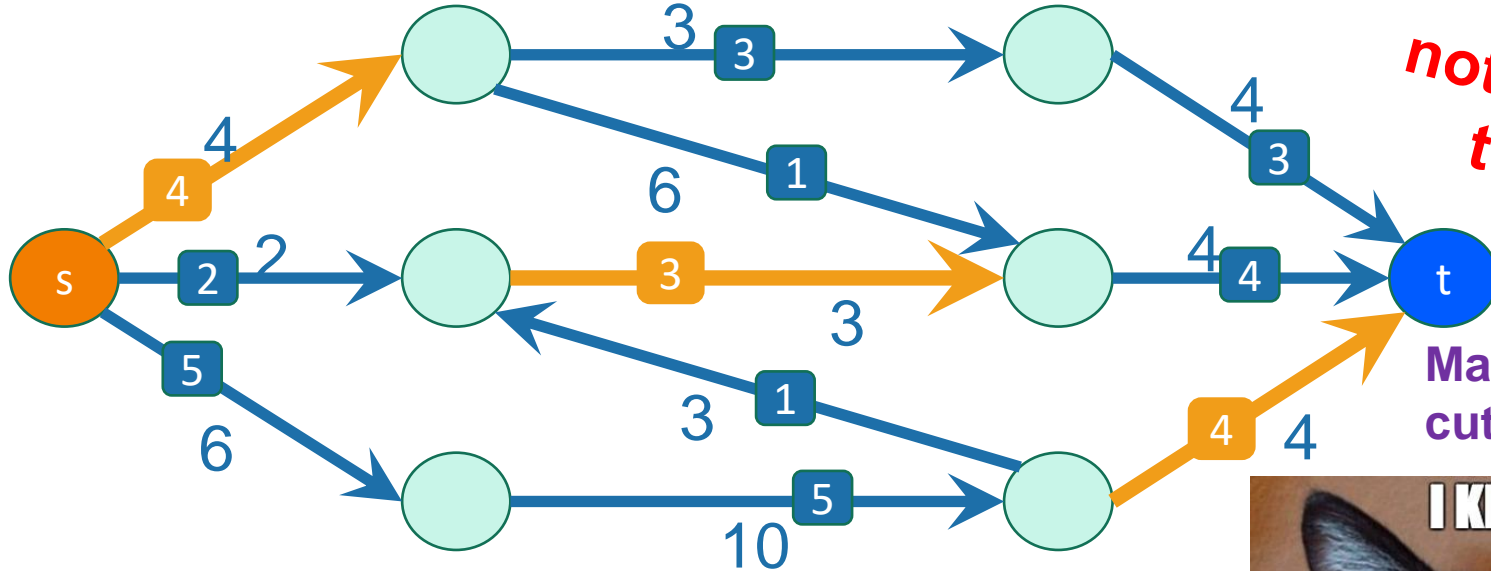
# Example of Ford-Fulkerson



*Now we  
have  
nothing left  
to do!*

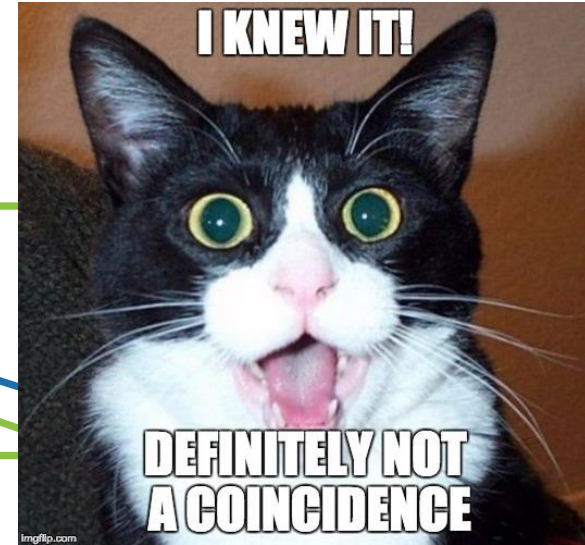


# Example of Ford-Fulkerson

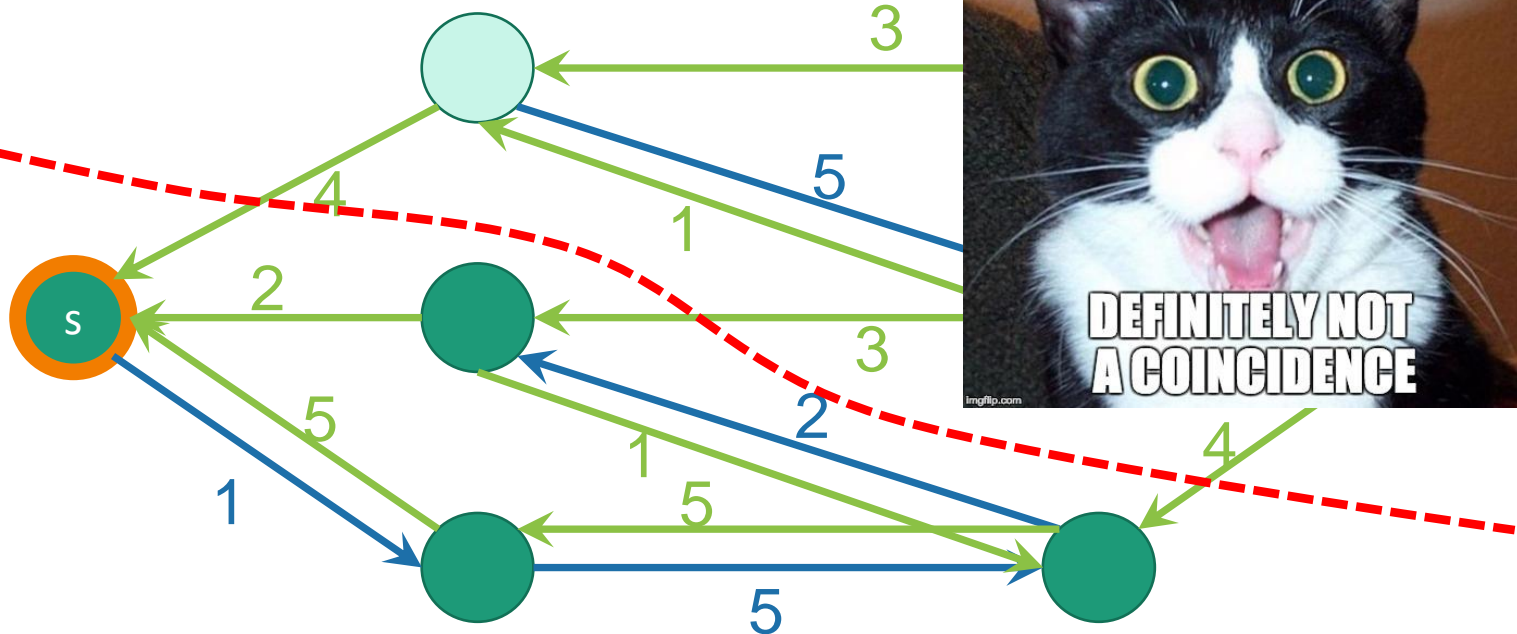


*Now we  
have  
nothing left  
to do!*

Max flow and min  
cut are both 11.



There's no  
path from s to  
t, and here's  
the cut to prove



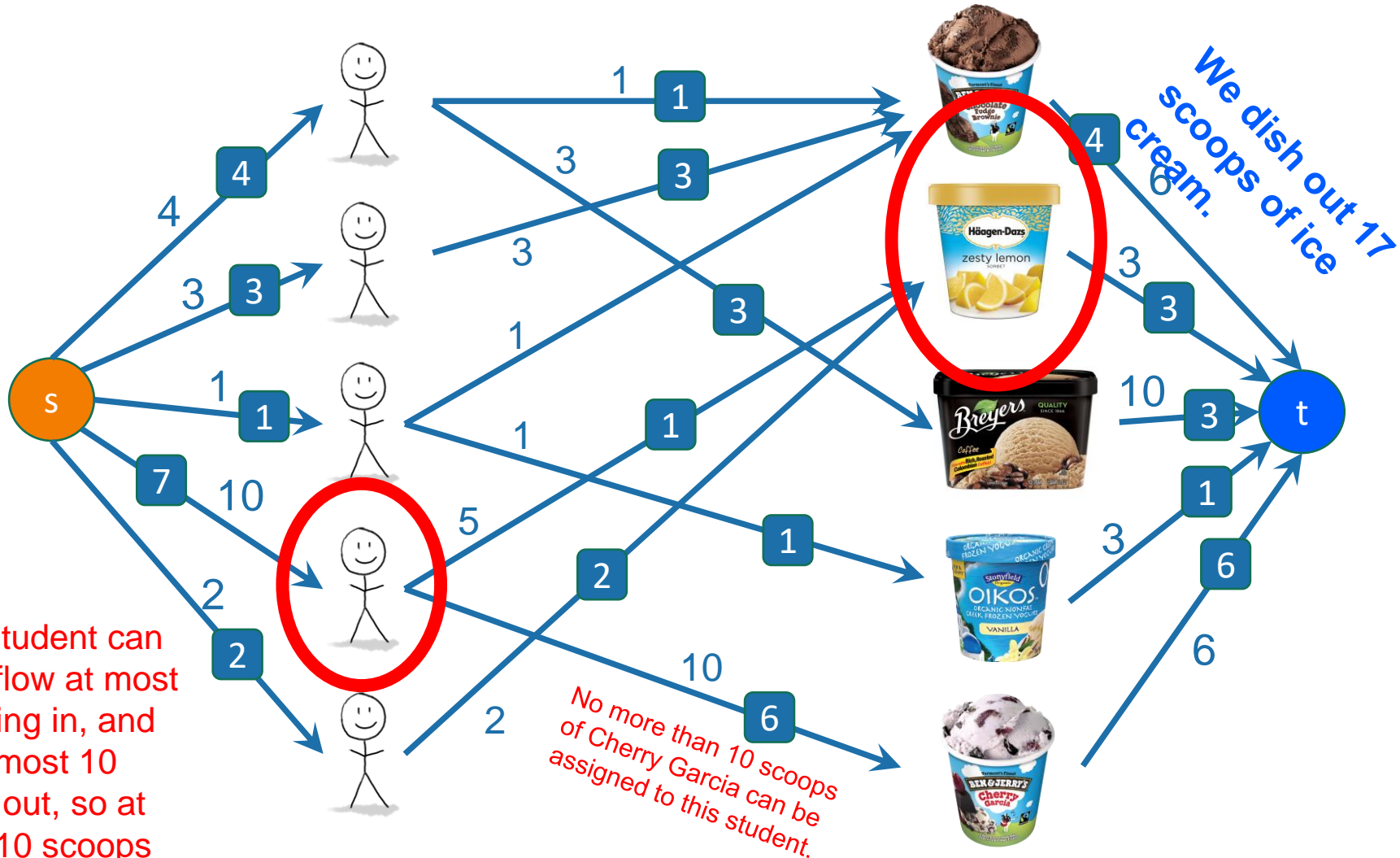
Doing Ford-Fulkerson with BFS is called the **Edmonds-Karp algorithm**.

# Theorem

- If you use BFS, the Ford-Fulkerson algorithm runs in time  **$O(nm^2)$** .  
Doesn't have anything to do with the edge weights!
- We will skip the proof in class.
- Basic idea:
  - The number of times you remove an edge from the residual graph is  $O(n)$ .
    - This is the hard part
  - There are at most  $m$  edges.
  - Each time we remove an edge we run BFS, which takes time  $O(n+m)$ .
    - Actually,  $O(m)$ , since we don't need to explore the whole graph, just the stuff reachable from  $s$ .

# Solution via max flow

No more than 3 scoops of sorbet can be assigned.



This student can have flow at most 10 going in, and so at most 10 going out, so at most 10 scoops assigned.

As before, flows correspond to assignments, and max flows correspond to max assignments.

# Recap

- Today we talked about s-t cuts and s-t flows.
- The **Min-Cut Max-Flow Theorem** says that minimizing the cost of cuts is the same as maximizing the value of flows.
- The Ford-Fulkerson algorithm does this!
  - Find an augmenting path
  - Increase the flow along that path
  - Repeat until you can't find any more paths and then you're done!
- An important algorithmic primitive!
  - eg, assignment problems.



# Polynomial-Time Reductions

---

# Classify Problems According to Computational Requirements

Q. Which problems will we be able to solve in practice?

A working definition. [von Neumann 1953, Godel 1956, Cobham 1964, Edmonds 1965, Rabin 1966]

Those with polynomial-time algorithms.

Yes	Probably no
Shortest path	Longest path
Matching	3D-matching
Min cut	Max cut
2-SAT	3-SAT
Planar 4-color	Planar 3-color
Bipartite vertex cover	Vertex cover
Primality testing	Factoring

# Classify Problems

**Desiderata.** Classify problems according to those that can be solved in polynomial-time and those that cannot.

**Probably requires exponential-time.**

- Given a Turing machine, does it halt in at most  $k$  steps?
- Given a board position in an  $n$ -by- $n$  generalization of chess, can black guarantee a win?

**Frustrating news.** Huge number of fundamental problems have defied classification for decades.

**Today and Next week.** Show that these fundamental problems are "computationally equivalent" and appear to be different manifestations of one **really hard** problem.

# Polynomial-Time Reduction

**Reduction.** Problem  $X$  **polynomial-time reduces to** problem  $Y$  if arbitrary instances of problem  $X$  can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to oracle that solves problem  $Y$ .

**Notation.**  $X \leq_p Y$ .

**Remarks.**

- We pay for time to write down instances sent to black box  $\Rightarrow$  instances of  $Y$  must be of polynomial size.
- Note: Cook reducibility (vs. Karp reducibility)

Means we can solve  $X$  in polynomial time IF we can solve  $Y$  in polynomial time!

# Polynomial-Time Reduction

**Purpose.** Classify problems according to **relative** difficulty.

**Design algorithms.** If  $X \leq_p Y$  and  $Y$  can be solved in polynomial-time, then  $X$  can also be solved in polynomial time.

**Establish intractability.** If  $X \leq_p Y$  and  $X$  cannot be solved in polynomial-time, then  $Y$  cannot be solved in polynomial time.

**Establish equivalence.** If  $X \leq_p Y$  and  $Y \leq_p X$ , we use notation  $X \equiv_p Y$ .

# Reduction By Simple Equivalence

---

Basic reduction strategies.

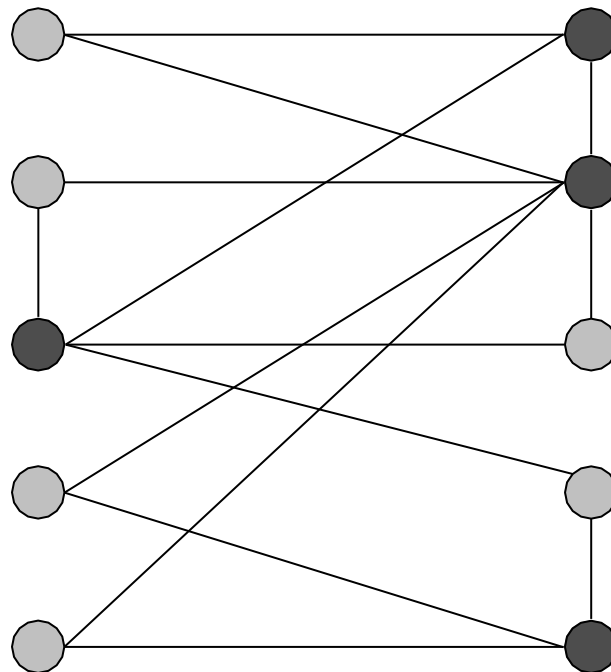
- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

# Independent Set

**INDEPENDENT SET:** Given a graph  $G = (V, E)$  and an integer  $k$ , is there a subset of vertices  $S \subseteq V$  such that  $|S| \geq k$ , and for each edge at most one of its endpoints is in  $S$ ?

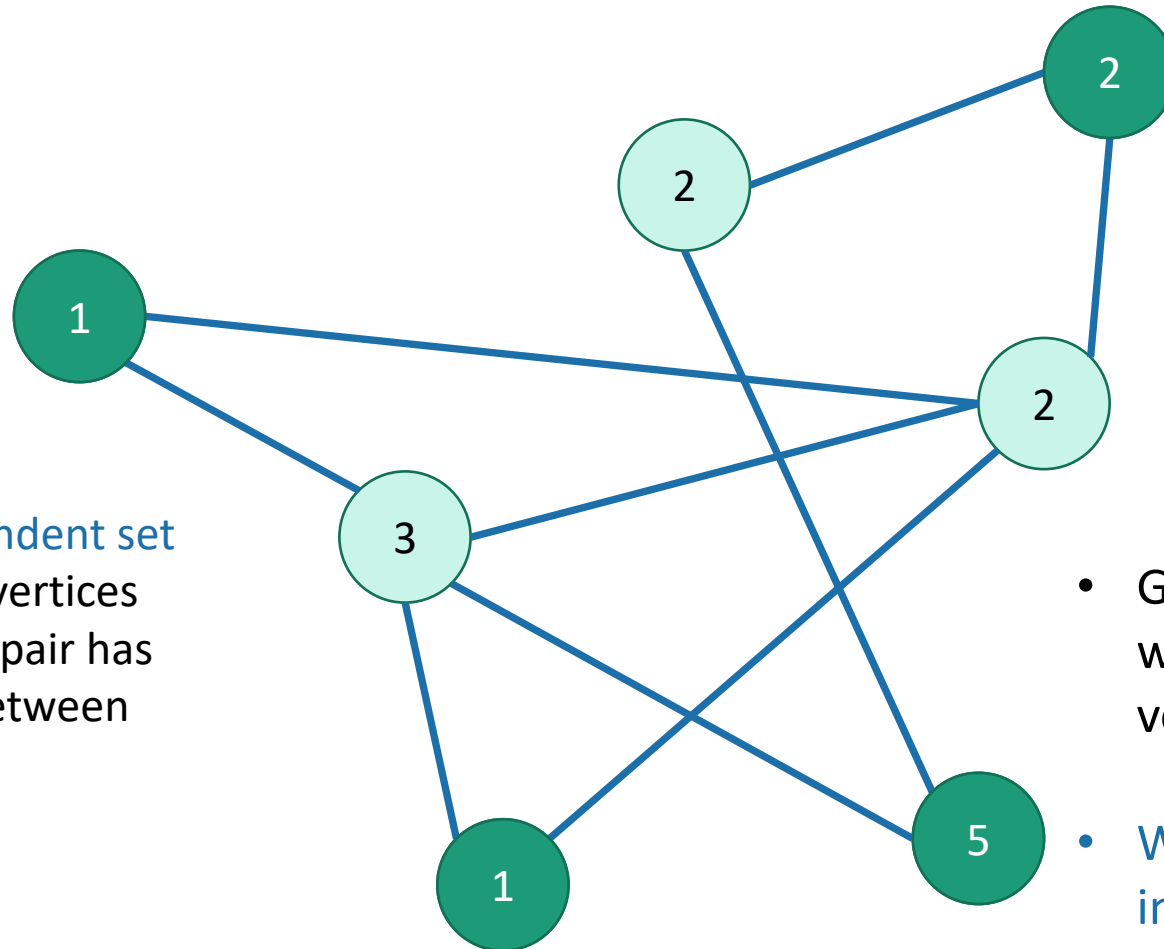
Ex. Is there an independent set of size  $\geq 6$ ? **YES**

Ex. Is there an independent set of size  $\geq 7$ ? **NO**



● independent set

# Independent Set



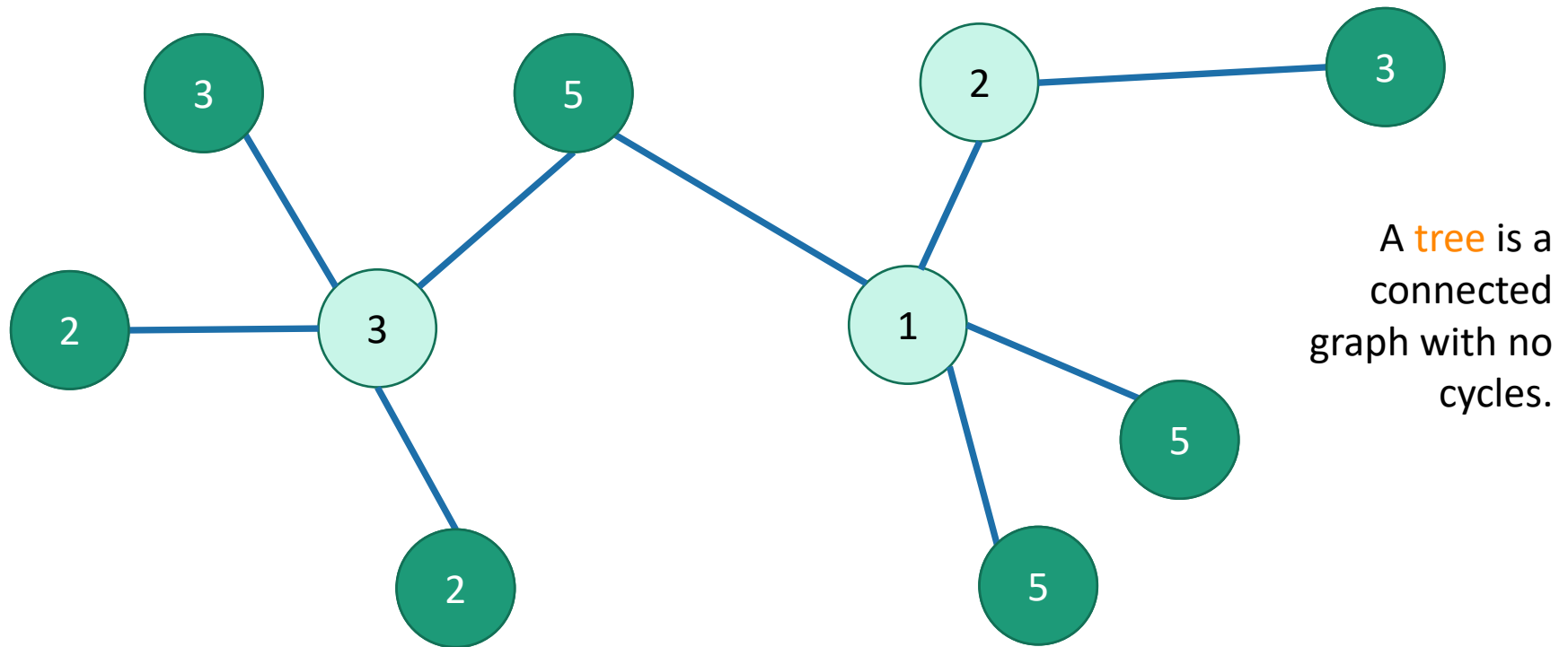
An **independent set** is a set of vertices so that no pair has an edge between them.

- Given a graph with weights on the vertices...
- What is the independent set with the largest weight?



Actually this problem is **NP-complete**.  
So we are unlikely to find an efficient algorithm

- But if we also assume that the graph is a **tree**...



**Problem:**

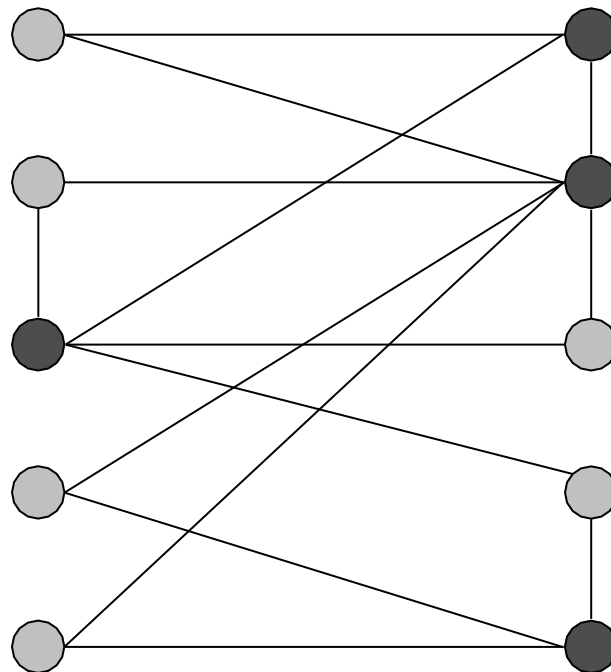
find a maximal independent set in a tree (with vertex weights).

# Vertex Cover

**VERTEX COVER:** Given a graph  $G = (V, E)$  and an integer  $k$ , is there a subset of vertices  $S \subseteq V$  such that  $|S| \leq k$ , and for each edge, at least one of its endpoints is in  $S$ ?

**Ex.** Is there a vertex cover of size  $\leq 4$ ? **YES**

**Ex.** Is there a vertex cover of size  $\leq 3$ ? **NO**

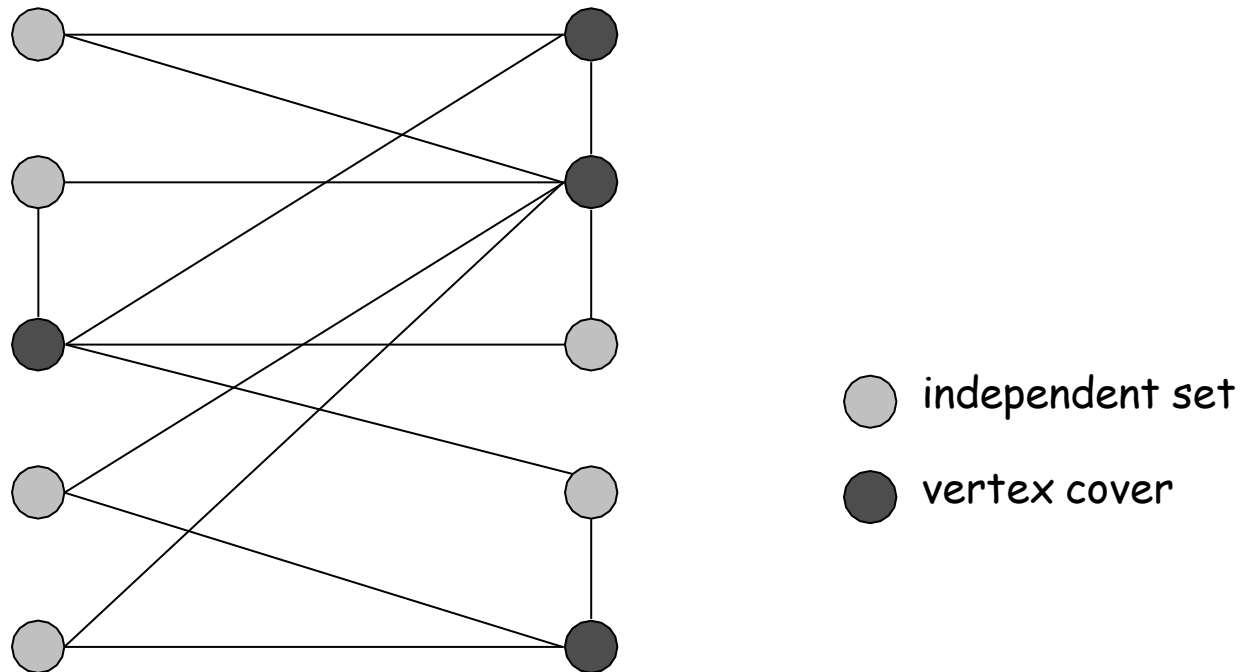


● vertex cover

# Vertex Cover and Independent Set

**Claim.** VERTEX-COVER  $\equiv_p$  INDEPENDENT-SET.

**Pf.** We show  $S$  is an independent set iff  $V - S$  is a vertex cover.



# Reduction from Special Case to General Case

---

Basic reduction strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

# Set Cover

**SET COVER:** Given a set  $U$  of elements, a collection  $S_1, S_2, \dots, S_m$  of subsets of  $U$ , and an integer  $k$ , does there exist a collection of  $\leq k$  of these sets whose union is equal to  $U$ ?

## Sample application.

- $m$  available pieces of software.
- Set  $U$  of  $n$  capabilities that we would like our system to have.
- The  $i$ th piece of software provides the set  $S_i \subseteq U$  of capabilities.
- Goal: achieve all  $n$  capabilities using fewest pieces of software.

Ex:

$$U = \{1, 2, 3, 4, 5, 6, 7\}$$

$$k = 2$$

$$S_1 = \{3, 7\}$$

$$S_4 = \{2, 4\}$$

$$S_2 = \{3, 4, 5, 6\}$$

$$S_5 = \{5\}$$

$$S_3 = \{1\}$$

$$S_6 = \{1, 2, 6, 7\}$$

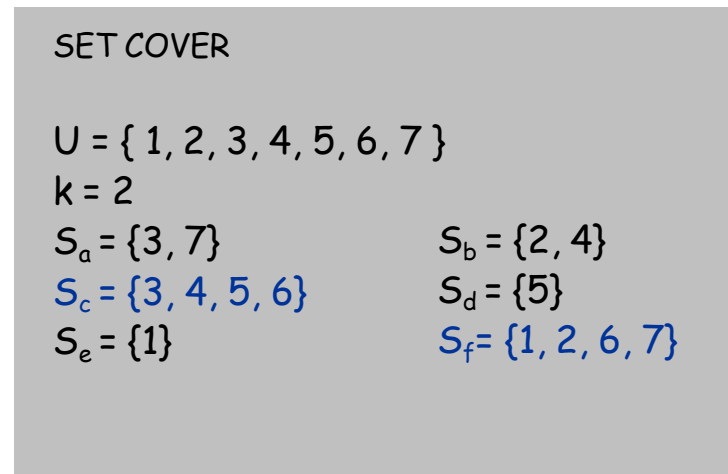
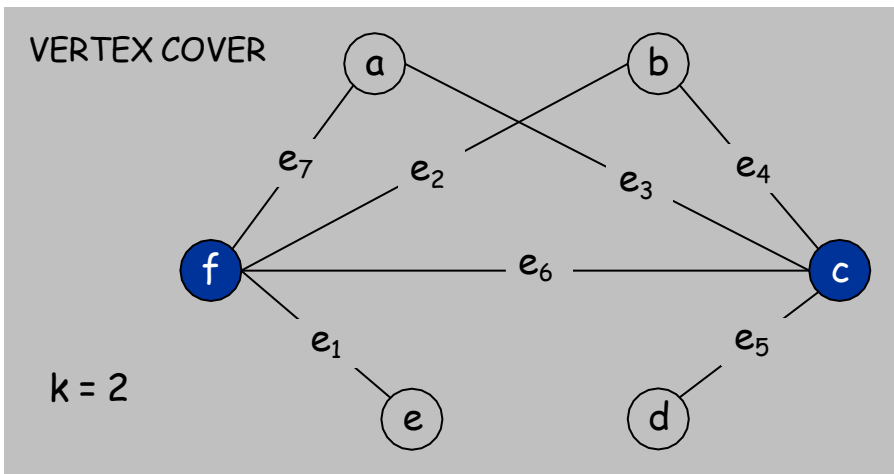
# Vertex Cover Reduces to Set Cover

**Claim.** VERTEX-COVER  $\leq_p$  SET-COVER.

**Pf.** Given a VERTEX-COVER instance  $G = (V, E)$ ,  $k$ , we construct a set cover instance whose size equals the size of the vertex cover instance.

**Construction.**

- Create SET-COVER instance:
  - $k = k$ ,  $U = E$ ,  $S_v = \{e \in E : e \text{ incident to } v\}$
- Set-cover of size  $\leq k$  iff vertex cover of size  $\leq k$ . ▪



# Polynomial-Time Reduction

## Basic strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

## 8.2 Reductions via "Gadgets"

---

Basic reduction strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction via "gadgets."



# Satisfiability

**Literal:** A Boolean variable or its negation.

$$x_i \text{ or } \overline{x_i}$$

**Clause:** A disjunction of literals.

$$C_j = x_1 \vee \overline{x_2} \vee x_3$$

**Conjunctive normal form:** A propositional formula  $\Phi$  that is the conjunction of clauses.

$$\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$$

**SAT:** Given CNF formula  $\Phi$ , does it have a satisfying truth assignment?

**3-SAT:** SAT where each clause contains exactly 3 literals.

 each corresponds to a different variable

**Ex:**  $(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$

**Yes:**  $x_1 = \text{true}, x_2 = \text{true}, x_3 = \text{false}.$

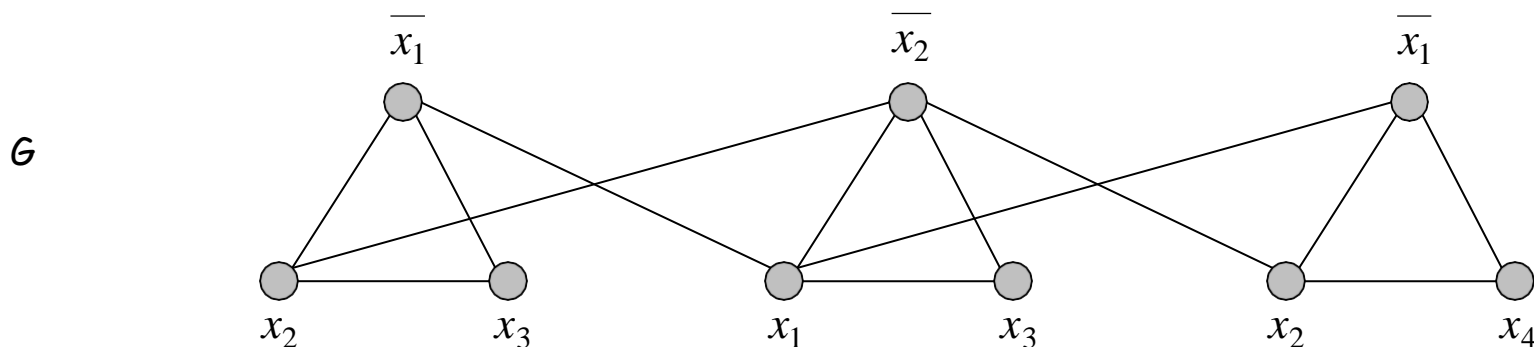
### 3 Satisfiability Reduces to Independent Set

**Claim.**  $3\text{-SAT} \leq_p \text{INDEPENDENT-SET}$ .

**Pf.** Given an instance  $\Phi$  of 3-SAT, we construct an instance  $(G, k)$  of INDEPENDENT-SET that has an independent set of size  $k$  iff  $\Phi$  is satisfiable.

**Construction.**

- $G$  contains 3 vertices for each clause, one for each literal.
- Connect 3 literals in a clause in a triangle.
- Connect literal to each of its negations.



$k = 3$

$$\Phi = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4)$$

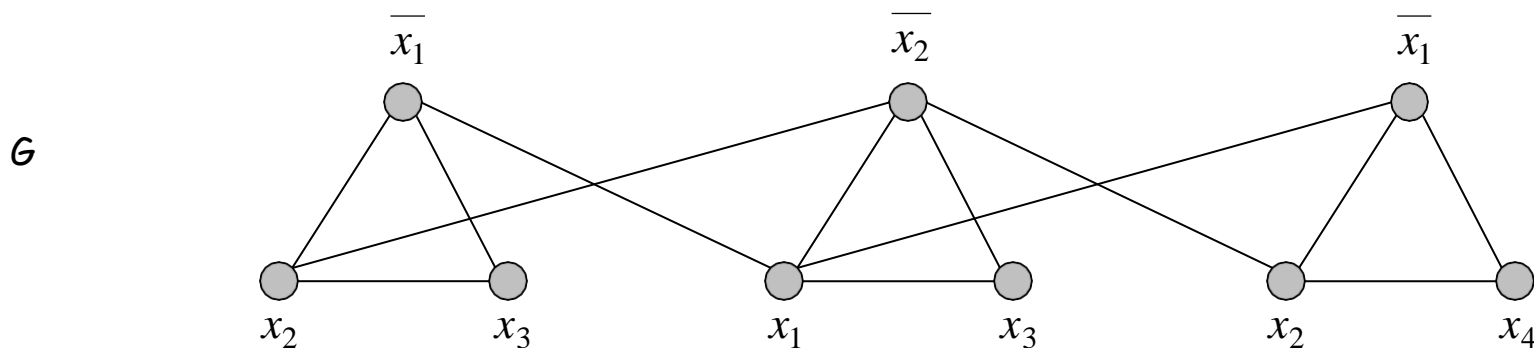
### 3 Satisfiability Reduces to Independent Set

**Claim.**  $G$  contains independent set of size  $k = |\Phi|$  iff  $\Phi$  is satisfiable.

**Pf.**  $\Rightarrow$  Let  $S$  be independent set of size  $k$ .

- $S$  must contain exactly one vertex in each triangle.
- Set these literals to true.  $\leftarrow$  and any other variables in a consistent way
- Truth assignment is consistent and all clauses are satisfied.

**Pf**  $\Leftarrow$  Given satisfying assignment, select one true literal from each triangle. This is an independent set of size  $k$ . ■



$k = 3$

$$\Phi = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4)$$

# Review

## Basic reduction strategies.

- Simple equivalence:  $\text{INDEPENDENT-SET} \equiv_p \text{VERTEX-COVER}$ .
- Special case to general case:  $\text{VERTEX-COVER} \leq_p \text{SET-COVER}$ .
- Encoding with gadgets:  $3\text{-SAT} \leq_p \text{INDEPENDENT-SET}$ .

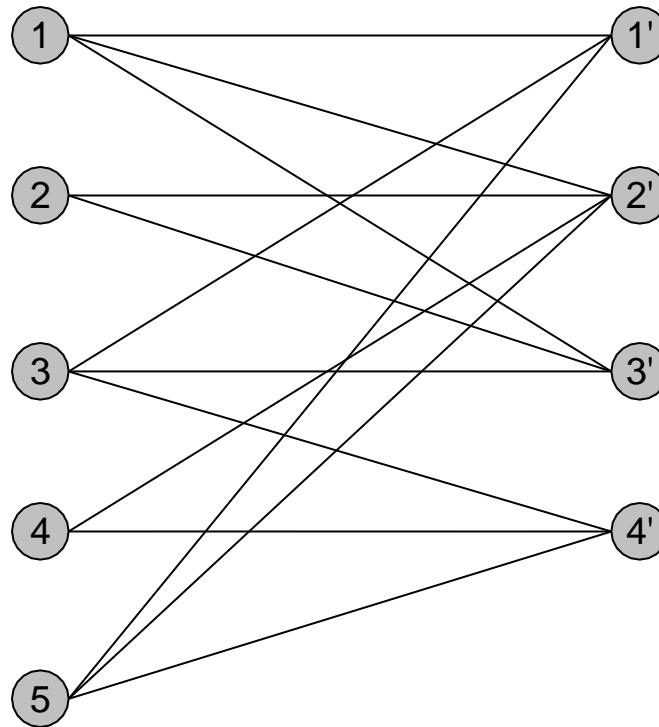
**Transitivity.** If  $X \leq_p Y$  and  $Y \leq_p Z$ , then  $X \leq_p Z$ .

**Pf idea.** Compose the two algorithms.

**Ex:**  $3\text{-SAT} \leq_p \text{INDEPENDENT-SET} \leq_p \text{VERTEX-COVER} \leq_p \text{SET-COVER}$ .

# Hamiltonian Cycle

**HAM-CYCLE:** given an undirected graph  $G = (V, E)$ , does there exist a simple cycle  $\Gamma$  that contains every node in  $V$ .



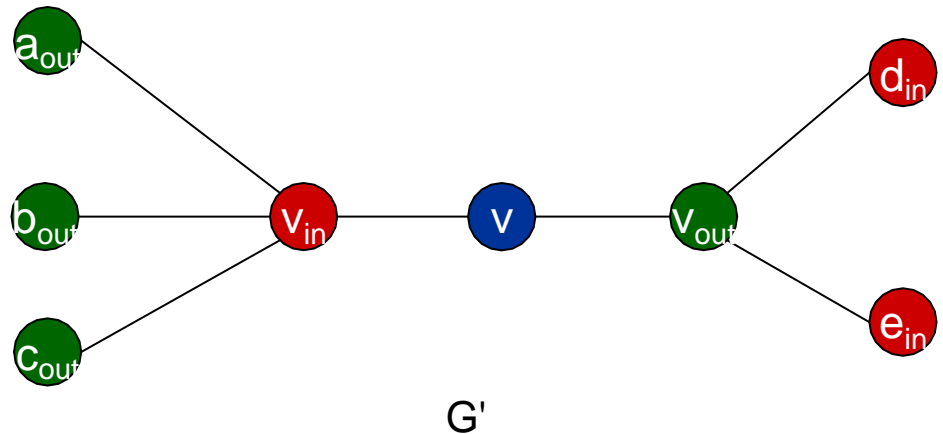
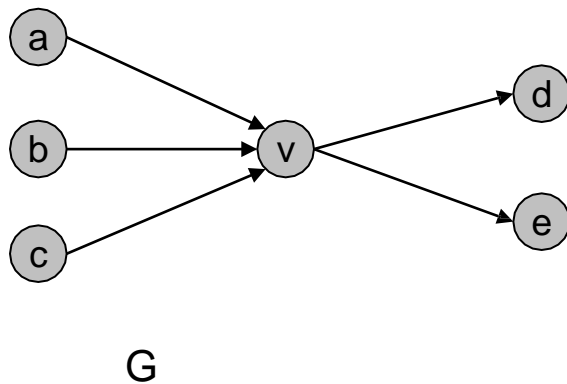
NO: bipartite graph with odd number of nodes.

# Directed Hamiltonian Cycle

**DIR-HAM-CYCLE:** given a **digraph**  $G = (V, E)$ , does there exist a simple directed cycle  $\Gamma$  that contains every node in  $V$ ?

**Claim.**  $\text{DIR-HAM-CYCLE} \leq_p \text{HAM-CYCLE}$ .

**Pf.** Given a directed graph  $G = (V, E)$ , construct an undirected graph  $G'$  with  $3n$  nodes.



# Directed Hamiltonian Cycle

**Claim.**  $G$  has a Hamiltonian cycle iff  $G'$  does.

**Pf.**  $\Rightarrow$

- Suppose  $G$  has a directed Hamiltonian cycle  $\Gamma$ .
- Then  $G'$  has an undirected Hamiltonian cycle (same order).

**Pf.**  $\Leftarrow$

- Suppose  $G'$  has an undirected Hamiltonian cycle  $\Gamma'$ .
- $\Gamma'$  must visit nodes in  $G'$  using one of following two orders:  
    ..., B, G, R, B, G, R, B, G, R, B, ...  
    ..., B, R, G, B, R, G, B, R, G, B, ...
- Blue nodes in  $\Gamma'$  make up directed Hamiltonian cycle  $\Gamma$  in  $G$ , or reverse of one.   ▪

## 3-SAT Reduces to Directed Hamiltonian Cycle

**Claim.**  $3\text{-SAT} \leq_p \text{DIR-HAM-CYCLE}$ .

**Pf.** Given an instance  $\Phi$  of 3-SAT, we construct an instance of DIR-HAM-CYCLE that has a Hamiltonian cycle iff  $\Phi$  is satisfiable.

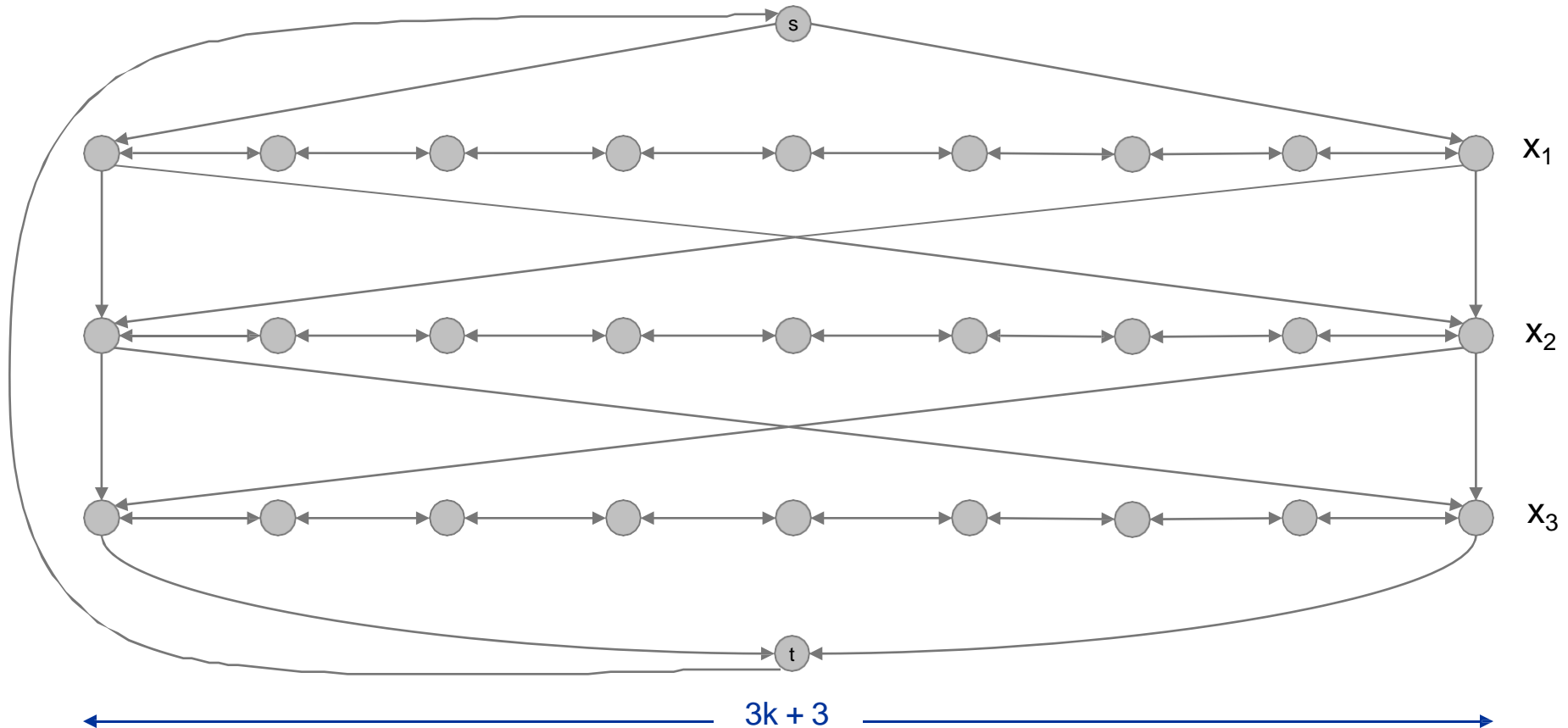
**Construction.** First, create graph that has  $2^n$  Hamiltonian cycles which correspond in a natural way to  $2^n$  possible truth assignments.



# 3-SAT Reduces to Directed Hamiltonian Cycle

**Construction.** Given 3-SAT instance  $\Phi$  with  $n$  variables  $x_i$  and  $k$  clauses.

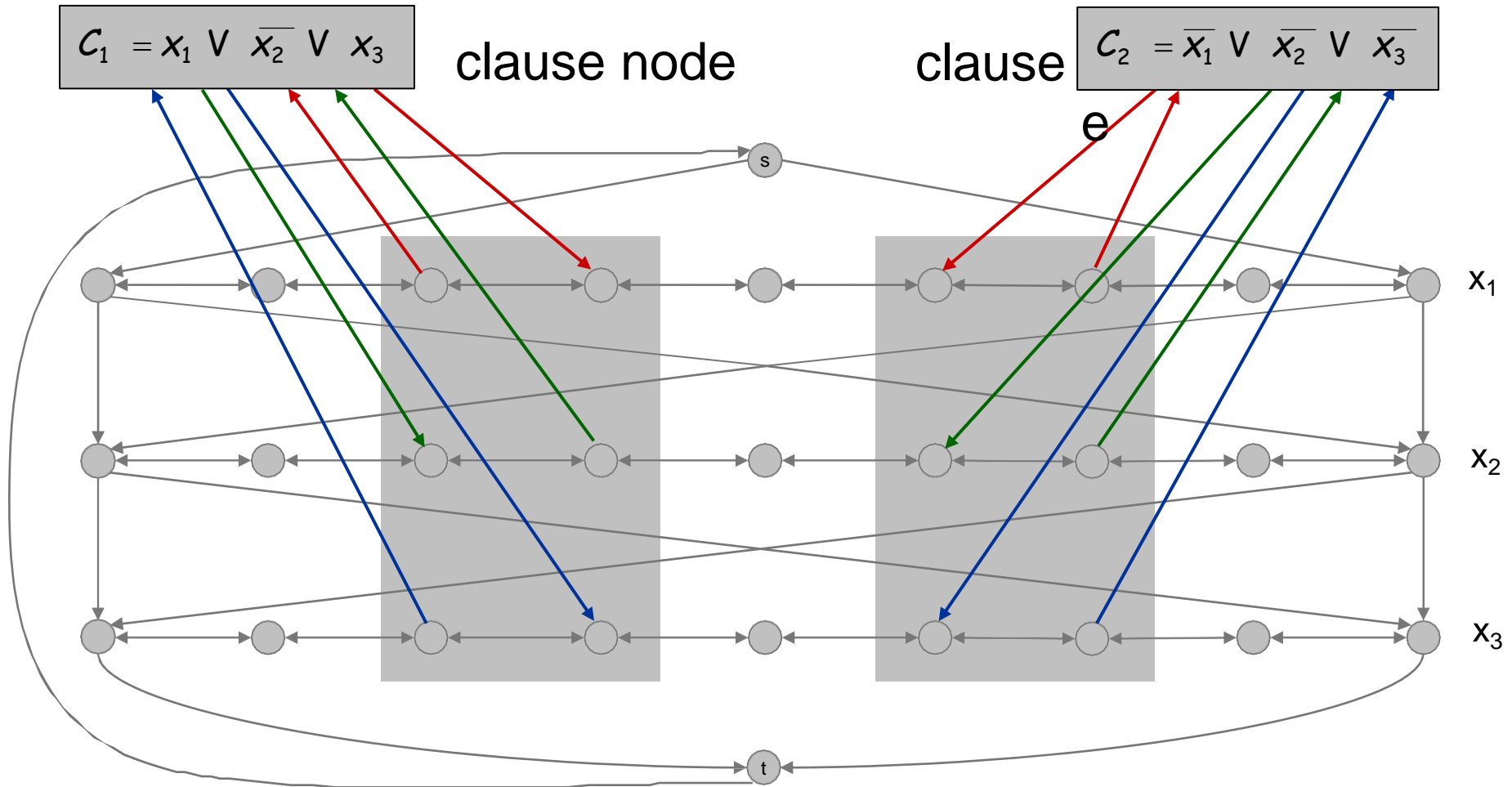
- Construct  $G$  to have  $2^n$  Hamiltonian cycles.
- Intuition: traverse path  $i$  from left to right  $\Leftrightarrow$  set variable  $x_i = 1$ .



# 3-SAT Reduces to Directed Hamiltonian Cycle

**Construction.** Given 3-SAT instance  $\Phi$  with  $n$  variables  $x_i$  and  $k$  clauses.

- For each clause: add a node and 6 edges.



## 3-SAT Reduces to Directed Hamiltonian Cycle

**Claim.**  $\Phi$  is satisfiable iff  $G$  has a Hamiltonian cycle.

**Pf.**  $\Rightarrow$

- Suppose 3-SAT instance has satisfying assignment  $x^*$ .
- Then, define Hamiltonian cycle in  $G$  as follows:
  - if  $x_i^* = 1$ , traverse row  $i$  from left to right
  - if  $x_i^* = 0$ , traverse row  $i$  from right to left
  - for each clause  $C_j$ , there will be at least one row  $i$  in which we are going in "correct" direction to splice node  $C_j$  into tour

## 3-SAT Reduces to Directed Hamiltonian Cycle

**Claim.**  $\Phi$  is satisfiable iff  $G$  has a Hamiltonian cycle.

**Pf.**  $\Leftarrow$

- Suppose  $G$  has a Hamiltonian cycle  $\Gamma$ .
- If  $\Gamma$  enters clause node  $C_j$ , it must depart on mate edge.
  - thus, nodes immediately before and after  $C_j$  are connected by an edge  $e$  in  $G$
  - removing  $C_j$  from cycle, and replacing it with edge  $e$  yields Hamiltonian cycle on  $G - \{C_j\}$
- Continuing in this way, we are left with Hamiltonian cycle  $\Gamma'$  in  $G - \{C_1, C_2, \dots, C_k\}$ .
- Set  $x_i^* = 1$  iff  $\Gamma'$  traverses row  $i$  left to right.
- Since  $\Gamma$  visits each clause node  $C_j$ , at least one of the paths is traversed in "correct" direction, and each clause is satisfied. ■

# Longest Path

**SHORTEST-PATH.** Given a digraph  $G = (V, E)$ , does there exist a simple path of length **at most**  $k$  edges?

**LONGEST-PATH.** Given a digraph  $G = (V, E)$ , does there exist a simple path of length **at least**  $k$  edges?

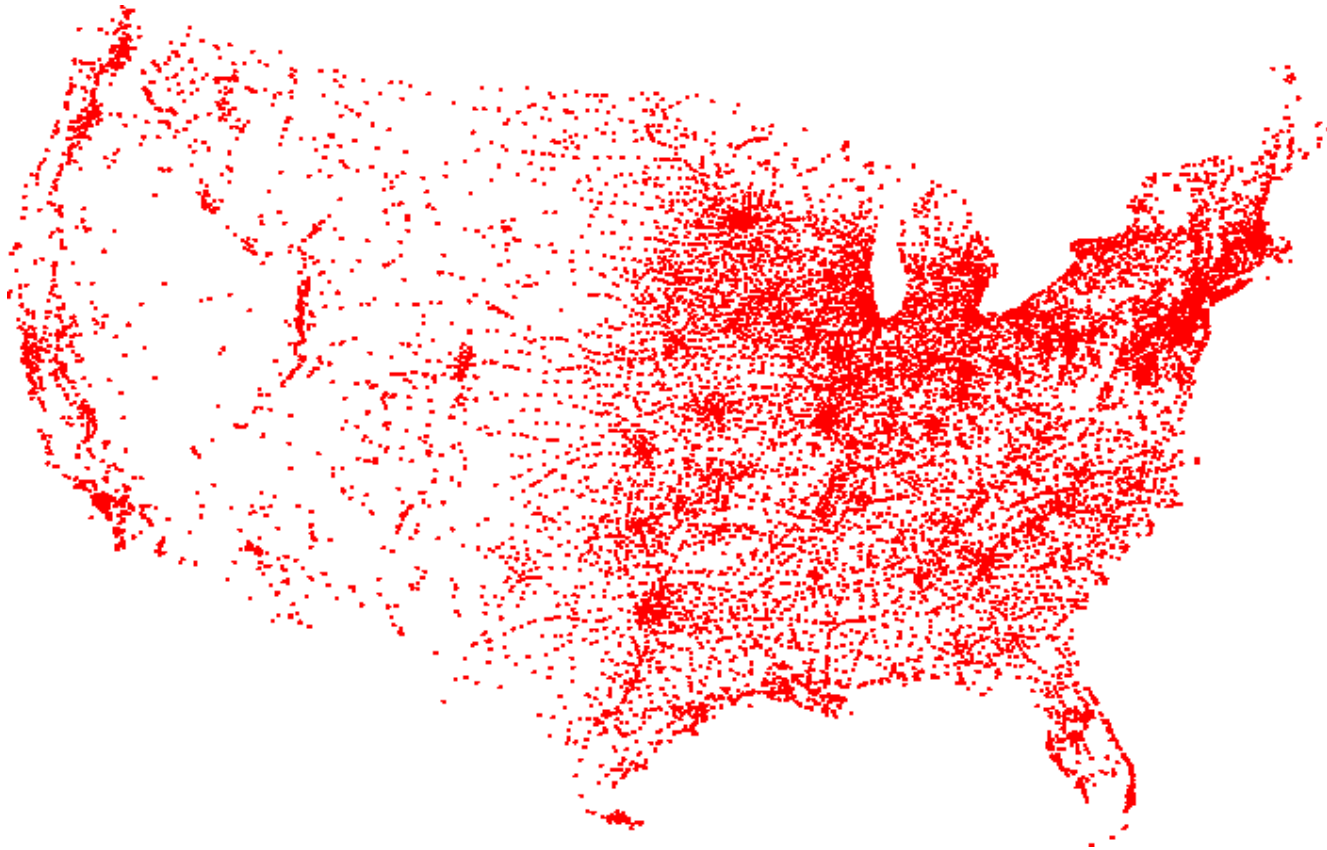
**Claim.**  $3\text{-SAT} \leq_p \text{LONGEST-PATH}$ .

**Pf 1.** Redo proof for  $\text{DIR-HAM-CYCLE}$ , ignoring back-edge from  $t$  to  $s$ .

**Pf 2.** Show  $\text{HAM-CYCLE} \leq_p \text{LONGEST-PATH}$ .

# Traveling Salesperson Problem

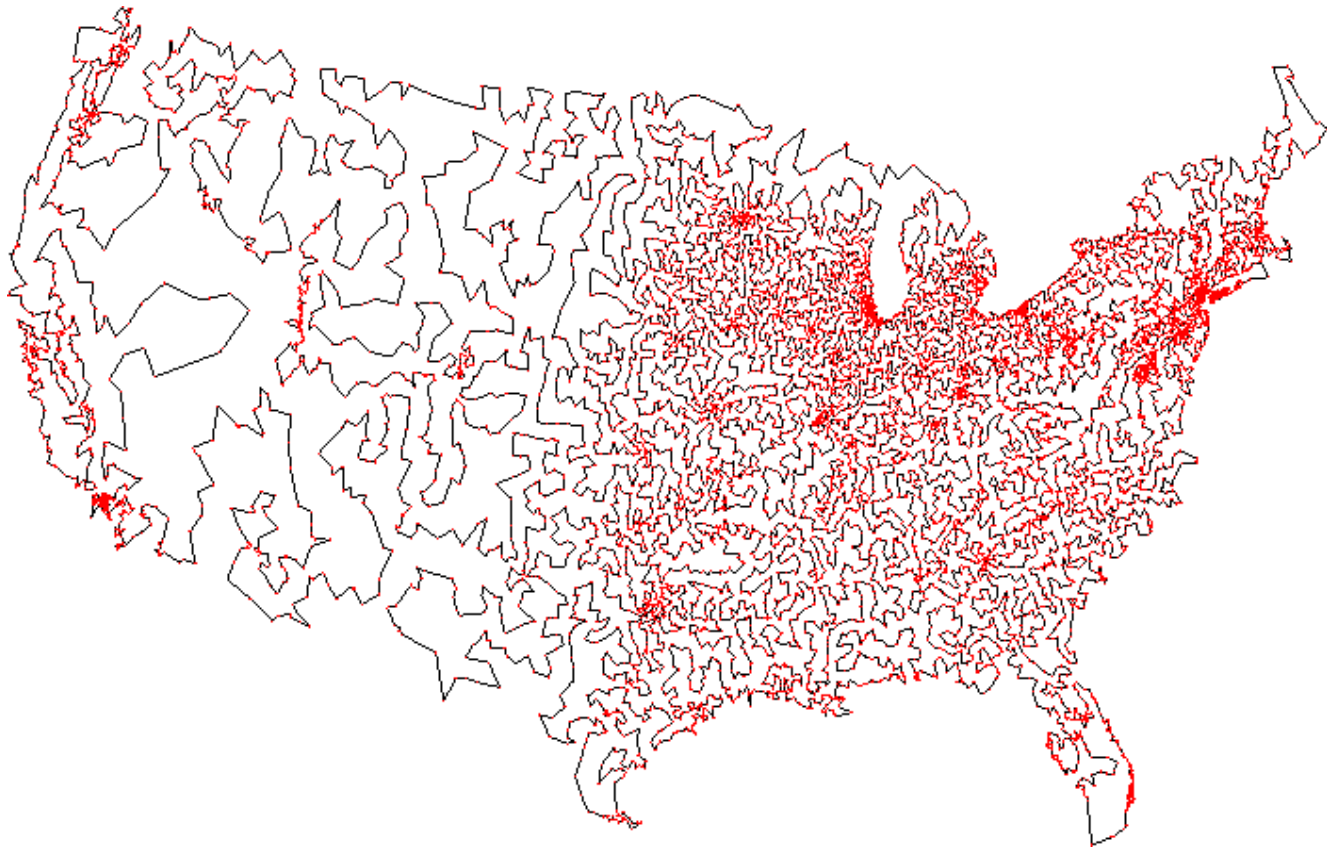
**TSP.** Given a set of  $n$  cities and a pairwise distance function  $d(u, v)$ , is there a tour of length  $\leq D$ ?



All 13,509 cities in US with a population of at least 500  
Reference: <http://www.tsp.gatech.edu>

# Traveling Salesperson Problem

**TSP.** Given a set of  $n$  cities and a pairwise distance function  $d(u, v)$ , is there a tour of length  $\leq D$ ?



Optimal TSP tour  
Reference: <http://www.tsp.gatech.edu>

# Traveling Salesperson Problem

**TSP.** Given a set of  $n$  cities and a pairwise distance function  $d(u, v)$ , is there a tour of length  $\leq D$ ?



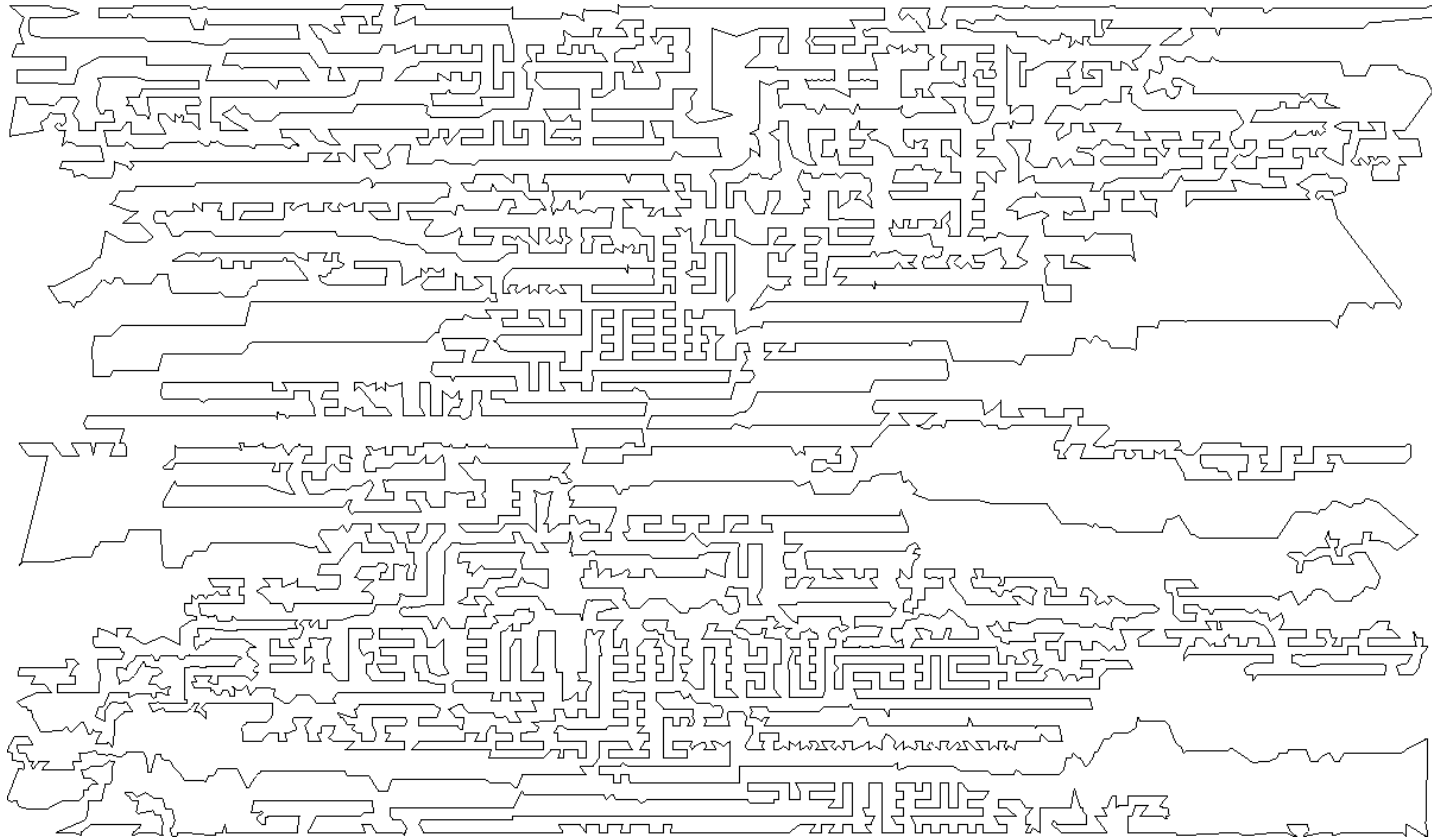
11,849 holes to drill in a programmed logic array

Reference: <http://www.tsp.gatech.edu>



# Traveling Salesperson Problem

**TSP.** Given a set of  $n$  cities and a pairwise distance function  $d(u, v)$ , is there a tour of length  $\leq D$ ?



Optimal TSP tour  
Reference: <http://www.tsp.gatech.edu>

# Traveling Salesperson Problem

**TSP.** Given a set of  $n$  cities and a pairwise distance function  $d(u, v)$ , is there a tour of length  $\leq D$ ?

**HAM-CYCLE:** given a graph  $G = (V, E)$ , does there exist a simple cycle that contains every node in  $V$ ?

**Claim.**  $\text{HAM-CYCLE} \leq_p \text{TSP}$ .

**Pf.**

- Given instance  $G = (V, E)$  of HAM-CYCLE, create  $n$  cities with distance function

$$d(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ 2 & \text{if } (u, v) \notin E \end{cases}$$

- TSP instance has tour of length  $\leq n$  iff  $G$  is Hamiltonian. ■

**Remark.** TSP instance in reduction satisfies  $\Delta$ -inequality.

# NP and Computational Intractability



$P = NP ?$



# P is a subset of NP

- Since it takes polynomial time to run the program, just run the program and get a solution
- But is NP a subset of P?
- No one knows if  $P = NP$  or not
- Solve for a million dollars!
  - <http://www.claymath.org/millennium-problems>
  - The Poincare conjecture is solved today

# Review

## Basic reduction strategies.

- Simple equivalence:  $\text{INDEPENDENT-SET} \equiv_p \text{VERTEX-COVER}$ .
- Special case to general case:  $\text{VERTEX-COVER} \leq_p \text{SET-COVER}$ .
- Encoding with gadgets:  $3\text{-SAT} \leq_p \text{INDEPENDENT-SET}$ .

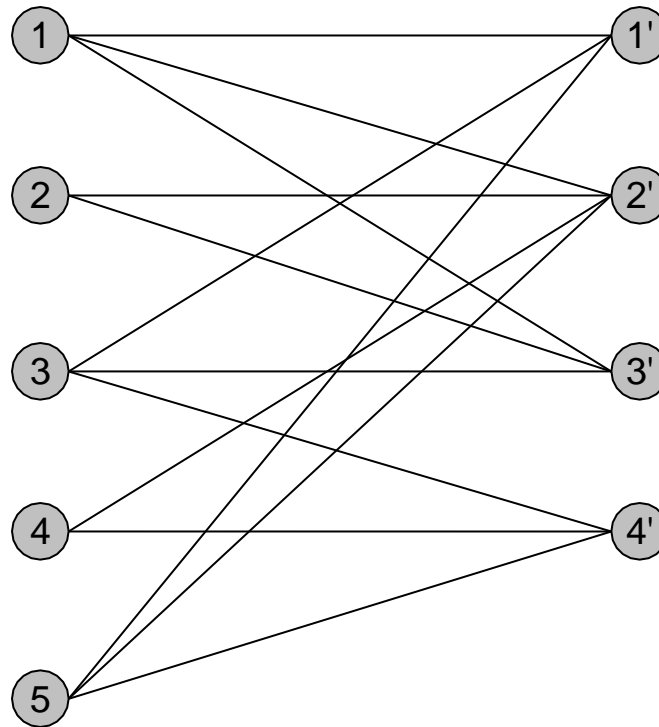
**Transitivity.** If  $X \leq_p Y$  and  $Y \leq_p Z$ , then  $X \leq_p Z$ .

**Pf idea.** Compose the two algorithms.

**Ex:**  $3\text{-SAT} \leq_p \text{INDEPENDENT-SET} \leq_p \text{VERTEX-COVER} \leq_p \text{SET-COVER}$ .

# Hamiltonian Cycle

**HAM-CYCLE:** given an undirected graph  $G = (V, E)$ , does there exist a simple cycle  $\Gamma$  that contains every node in  $V$ .



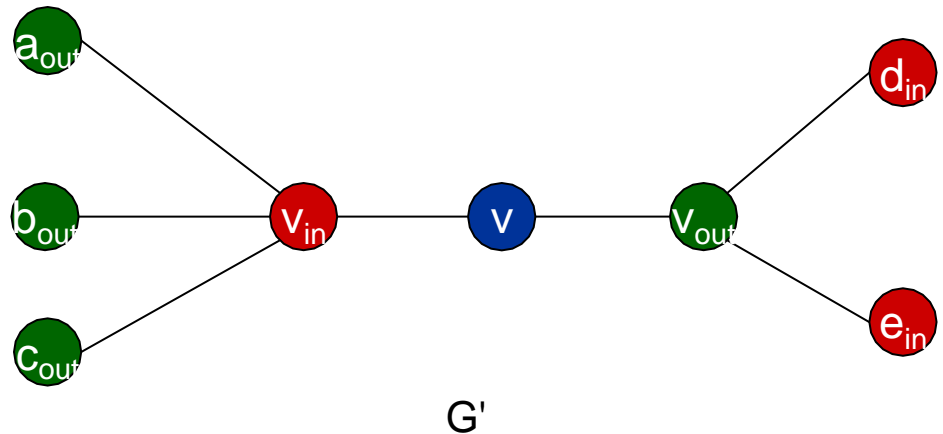
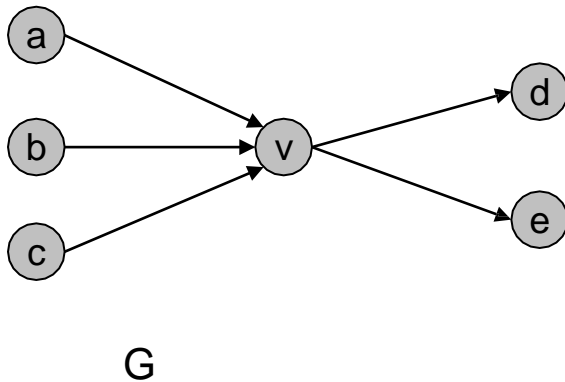
NO: bipartite graph with odd number of nodes.

# Directed Hamiltonian Cycle

**DIR-HAM-CYCLE:** given a **digraph**  $G = (V, E)$ , does there exist a simple directed cycle  $\Gamma$  that contains every node in  $V$ ?

**Claim.**  $\text{DIR-HAM-CYCLE} \leq_p \text{HAM-CYCLE}$ .

**Pf.** Given a directed graph  $G = (V, E)$ , construct an undirected graph  $G'$  with  $3n$  nodes.



# Directed Hamiltonian Cycle

**Claim.**  $G$  has a Hamiltonian cycle iff  $G'$  does.

**Pf.**  $\Rightarrow$

- Suppose  $G$  has a directed Hamiltonian cycle  $\Gamma$ .
- Then  $G'$  has an undirected Hamiltonian cycle (same order).

**Pf.**  $\Leftarrow$

- Suppose  $G'$  has an undirected Hamiltonian cycle  $\Gamma'$ .
- $\Gamma'$  must visit nodes in  $G'$  using one of following two orders:  
    ..., B, G, R, B, G, R, B, G, R, B, ...  
    ..., B, R, G, B, R, G, B, R, G, B, ...
- Blue nodes in  $\Gamma'$  make up directed Hamiltonian cycle  $\Gamma$  in  $G$ , or reverse of one.   ▪



## 3-SAT Reduces to Directed Hamiltonian Cycle

**Claim.**  $3\text{-SAT} \leq_p \text{DIR-HAM-CYCLE}$ .

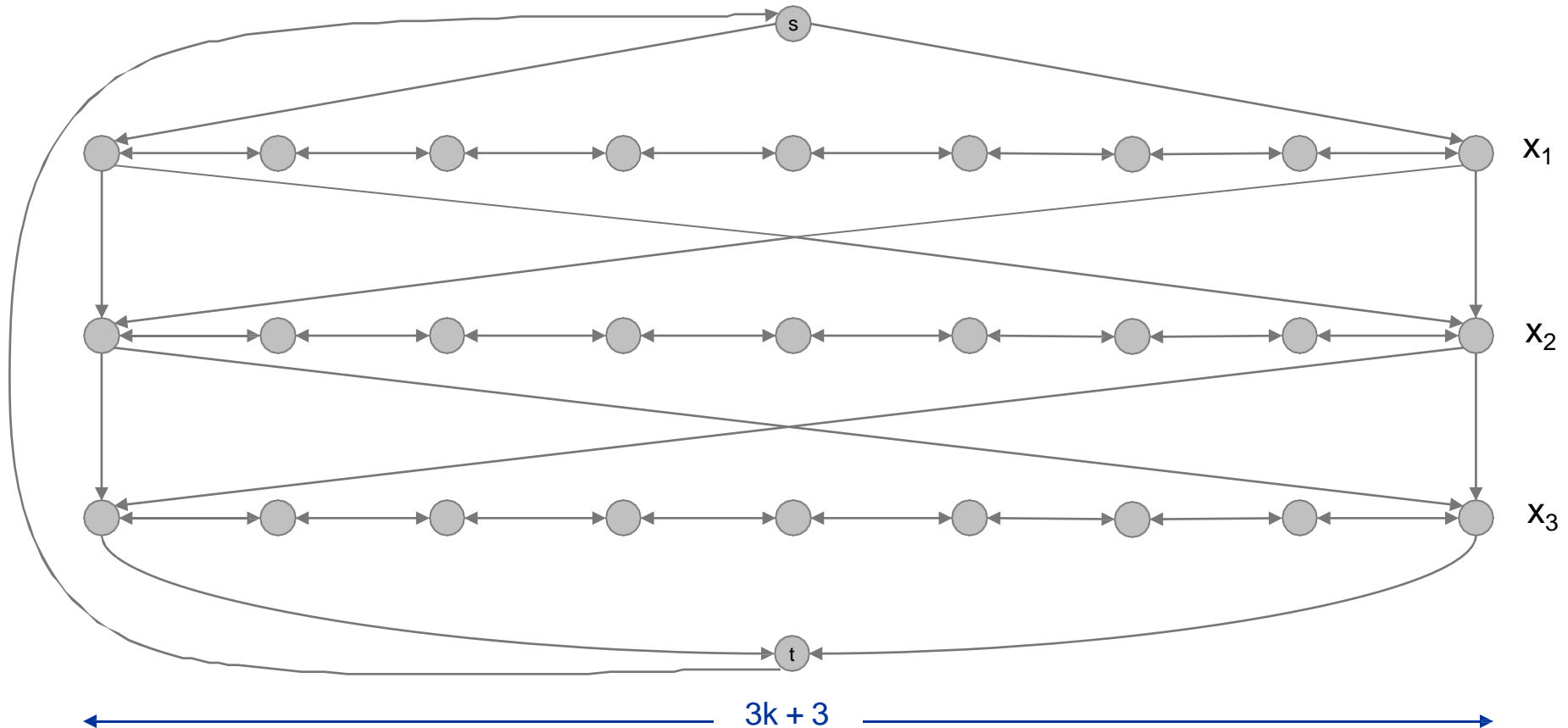
**Pf.** Given an instance  $\Phi$  of 3-SAT, we construct an instance of DIR-HAM-CYCLE that has a Hamiltonian cycle iff  $\Phi$  is satisfiable.

**Construction.** First, create graph that has  $2^n$  Hamiltonian cycles which correspond in a natural way to  $2^n$  possible truth assignments.

# 3-SAT Reduces to Directed Hamiltonian Cycle

**Construction.** Given 3-SAT instance  $\Phi$  with  $n$  variables  $x_i$  and  $k$  clauses.

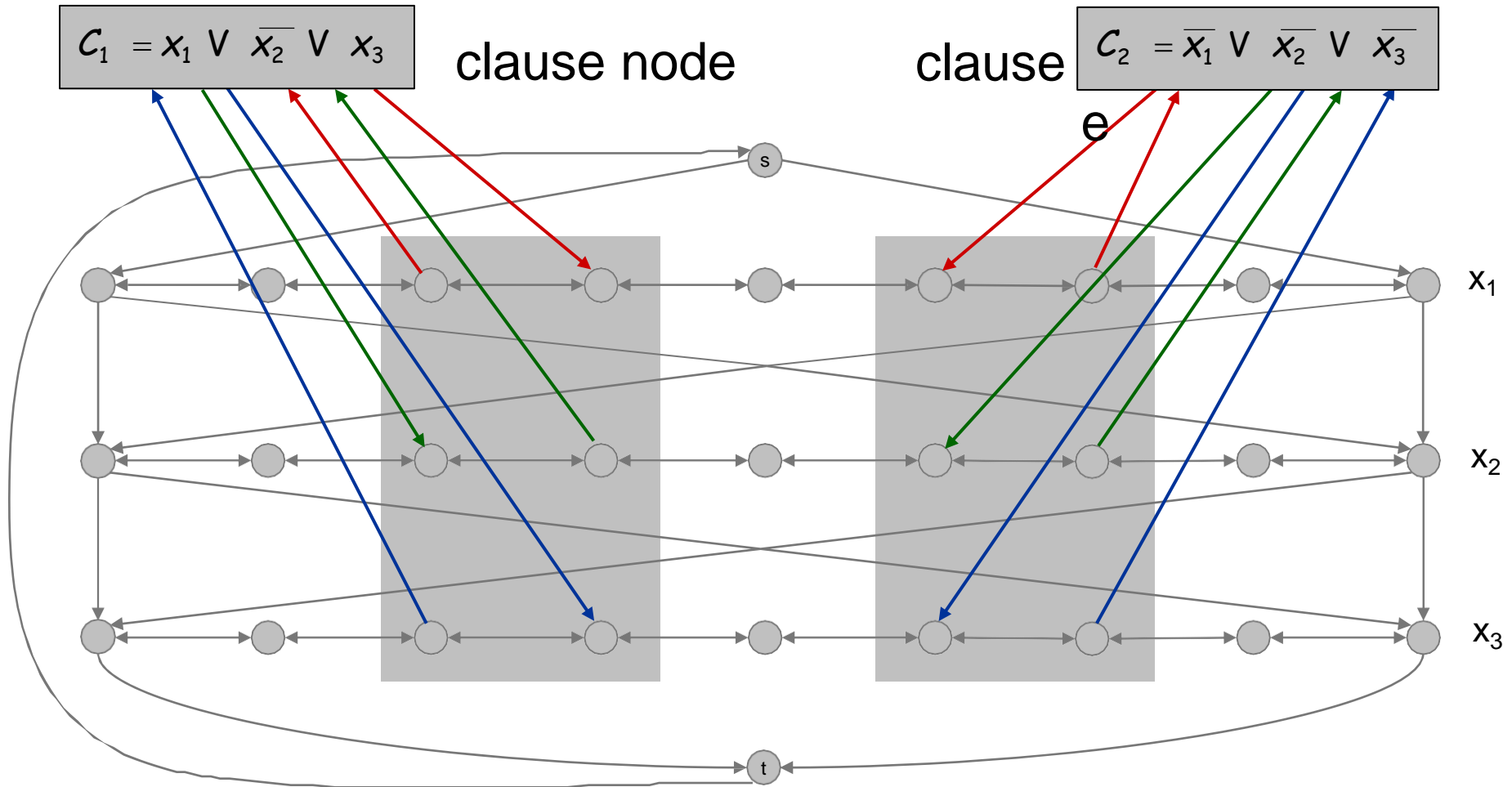
- Construct  $G$  to have  $2^n$  Hamiltonian cycles.
- Intuition: traverse path  $i$  from left to right  $\Leftrightarrow$  set variable  $x_i = 1$ .



# 3-SAT Reduces to Directed Hamiltonian Cycle

**Construction.** Given 3-SAT instance  $\Phi$  with  $n$  variables  $x_i$  and  $k$  clauses.

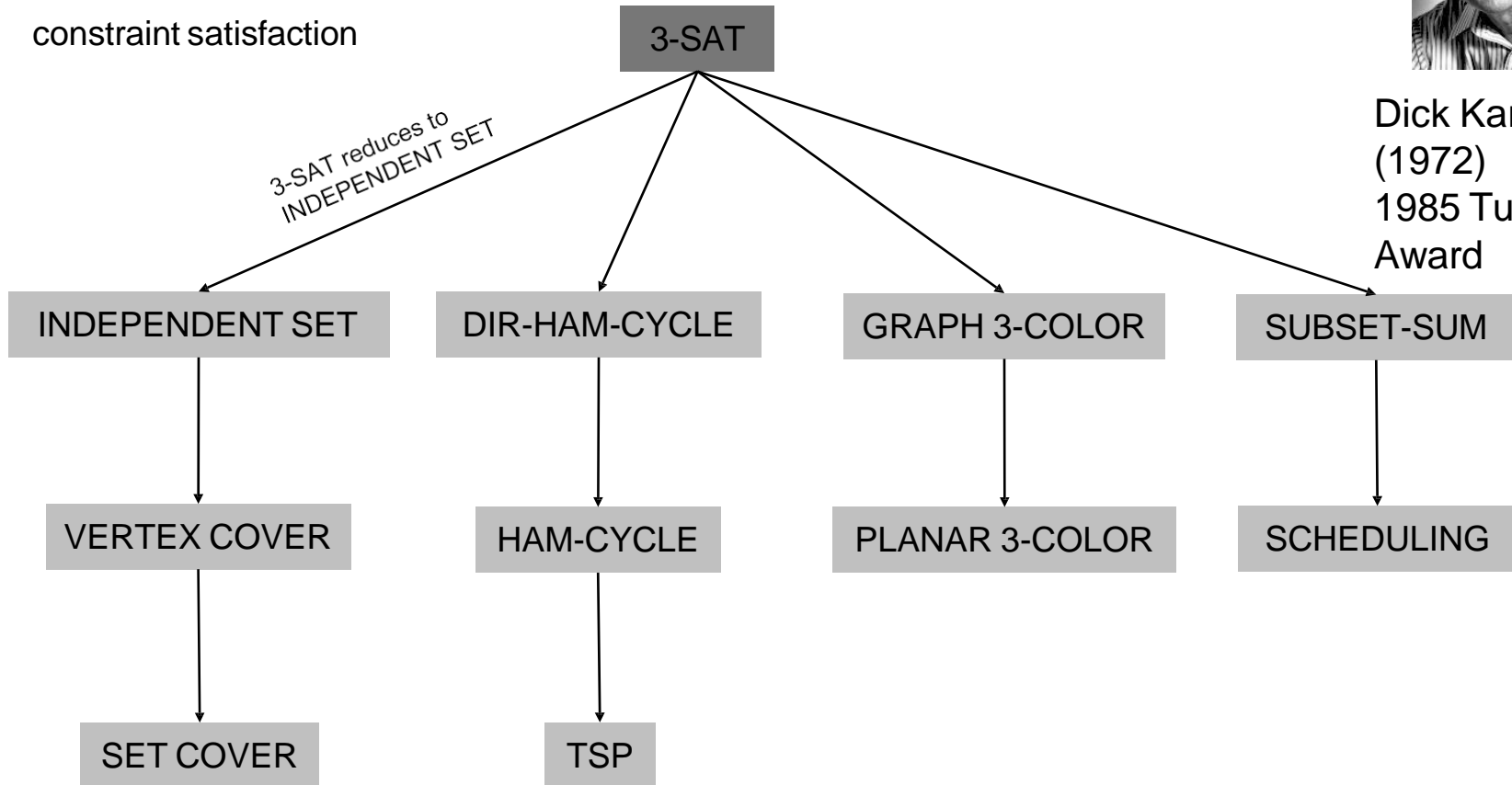
- For each clause: add a node and 6 edges.



# Polynomial-Time Reductions



Dick Karp  
(1972)  
1985 Turing  
Award



packing and covering

sequencing

partitioning

numerical

# Definition of NP

# The reducibility 'tree'

- Richard Karp proved 21 problems to be NP complete in a seminal 1971 paper
- Not that hard to read actually!
- Definitely not hard to read it to the point of knowing what these problems are.
- [karp's paper](#)

# Amusing/tragic NP story

- Breaking up over NP

# Decision Problems

## Decision problem.

- $X$  is a set of strings.
- Instance: string  $s$ .
- Algorithm  $A$  solves problem  $X$ :  $A(s) = \text{yes}$  iff  $s \in X$ .

**Polynomial time.** Algorithm  $A$  runs in poly-time if for every string  $s$ ,  $A(s)$  terminates in at most  $p(|s|)$  "steps", where  $p(\cdot)$  is some polynomial.

↑  
length of  $s$

**PRIMES:**  $X = \{ 2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, \dots \}$

**Algorithm.** [Agrawal-Kayal-Saxena, 2002]  $p(|s|) = |s|^8$ .



# Definition of P

P. Decision problems for which there is a poly-time algorithm.

Problem	Description	Algorithm	Yes	No
MULTIPLE	Is $x$ a multiple of $y$ ?	Grade school division	51, 17	51, 16
RELPRIME	Are $x$ and $y$ relatively prime?	Euclid (300 BCE)	34, 39	34, 51
PRIMES	Is $x$ prime?	AKS (2002)	53	51
EDIT-DISTANCE	Is the edit distance between $x$ and $y$ less than 5?	Dynamic programming	niether neither	acgggt ttttta
LSOLVE	Is there a vector $x$ that satisfies $Ax = b$ ?	Gauss-Edmonds elimination	$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{bmatrix}, \begin{bmatrix} 4 \\ 2 \\ 36 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

## Certification algorithm intuition.

- Certifier views things from "managerial" viewpoint.
- Certifier doesn't determine whether  $s \in X$  on its own; rather, it checks a proposed proof  $t$  that  $s \in X$ .

**Def.** Algorithm  $C(s, t)$  is a **certifier** for problem  $X$  if for every string  $s$ ,  $s \in X$  iff there exists a string  $t$  such that  $C(s, t) = \text{yes}$ .

"certificate" or "witness"

**NP.** Decision problems for which there exists a **poly-time** certifier.

$C(s, t)$  is a poly-time algorithm and  
 $|t| \leq p(|s|)$  for some polynomial  $p(\cdot)$ .

**Remark.** NP stands for **nondeterministic** polynomial-time.

# Certifiers and Certificates: Composite

**COMPOSITES.** Given an integer  $s$ , is  $s$  composite?

**Certificate.** A nontrivial factor  $t$  of  $s$ . Note that such a certificate exists iff  $s$  is composite. Moreover  $|t| \leq |s|$ .

**Certifier.**

```
boolean C(s, t) {  
    if (t ≤ 1 or t ≥ s)  
        return false  
    else if (s is a multiple of t)  
        return true  
    else  
        return false  
}
```

**Instance.**  $s = 437,669$ .

**Certificate.**  $t = 541$  or  $809$ .  $\longleftarrow 437,669 = 541 \times 809$

**Conclusion.** COMPOSITES is in NP.

## Certifiers and Certificates: 3-Satisfiability

**SAT.** Given a CNF formula  $\Phi$ , is there a satisfying assignment?

**Certificate.** An assignment of truth values to the  $n$  boolean variables.

**Certifier.** Check that each clause in  $\Phi$  has at least one true literal.

**Ex.**

$$(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\overline{x_1} \vee \overline{x_3} \vee \overline{x_4})$$

instance  $s$

$$x_1=1, x_2=1, x_3=0, x_4=1$$

certificate  $t$

**Conclusion.** SAT is in NP.

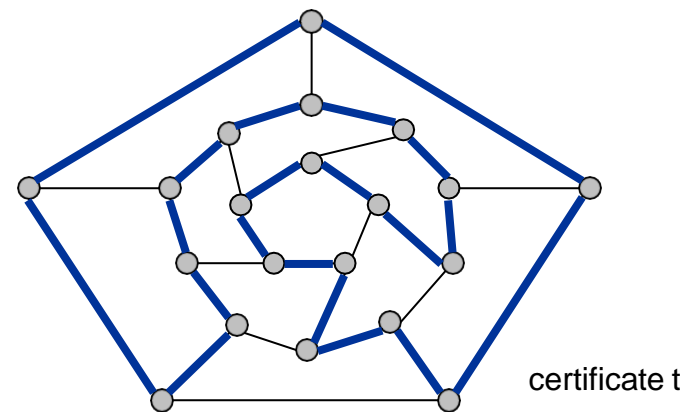
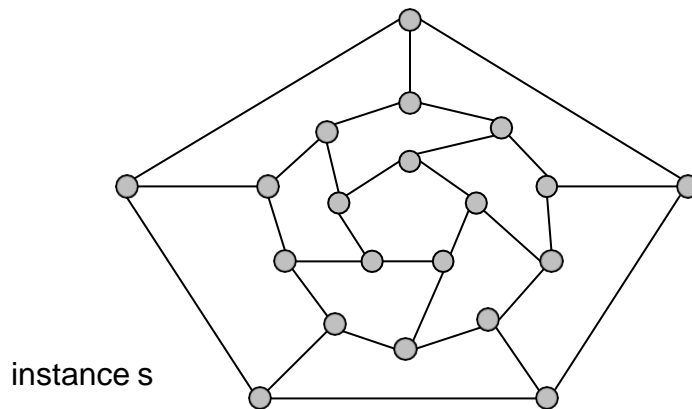
# Certifiers and Certificates: Hamiltonian Cycle

**HAM-CYCLE.** Given an undirected graph  $G = (V, E)$ , does there exist a simple cycle  $C$  that visits every node?

**Certificate.** A permutation of the  $n$  nodes.

**Certifier.** Check that the permutation contains each node in  $V$  exactly once, and that there is an edge between each pair of adjacent nodes in the permutation.

**Conclusion.** HAM-CYCLE is in NP.



# P, NP, EXP

**P.** Decision problems for which there is a **poly-time algorithm**.

**EXP.** Decision problems for which there is an **exponential-time algorithm**.

**NP.** Decision problems for which there is a **poly-time certifier**.

**Claim.**  $P \subseteq NP$ .

**Pf.** Consider any problem  $X$  in  $P$ .

- By definition, there exists a poly-time algorithm  $A(s)$  that solves  $X$ .
- Certificate:  $t = \varepsilon$ , certifier  $C(s, t) = A(s)$ . ▪

**Claim.**  $NP \subseteq EXP$ .

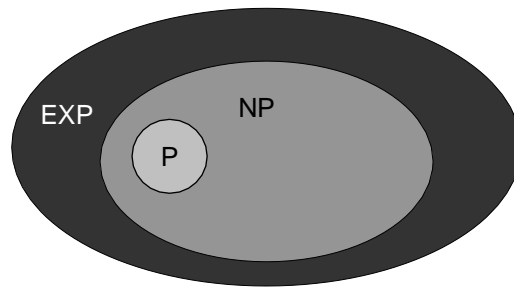
**Pf.** Consider any problem  $X$  in  $NP$ .

- By definition, there exists a poly-time certifier  $C(s, t)$  for  $X$ .
- To solve input  $s$ , run  $C(s, t)$  on all strings  $t$  with  $|t| \leq p(|s|)$ .
- Return **yes**, if  $C(s, t)$  returns **yes** for any of these. ▪

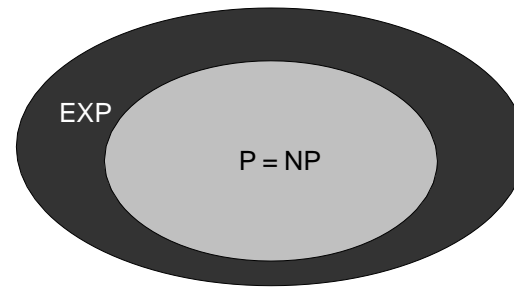
# The Main Question: P Versus NP

Does  $P = NP$ ? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]

- Is the decision problem as easy as the certification problem?
- Clay \$1 million prize.



If  $P \neq NP$



If  $P = NP$

would break RSA cryptography  
(and potentially collapse economy)

If yes: Efficient algorithms for 3-COLOR, TSP, FACTOR, SAT, ...

If no: No efficient algorithms possible for 3-COLOR, TSP, SAT, ...

Consensus opinion on  $P = NP$ ? Probably no.

# NP-Completeness



# NP-Complete

**NP-complete.** A problem  $Y$  in NP with the property that for every problem  $X$  in NP,  $X \leq_p Y$ .

**Theorem.** Suppose  $Y$  is an NP-complete problem. Then  $Y$  is solvable in poly-time iff  $P = NP$ .

**Pf.**  $\Leftarrow$  If  $P = NP$  then  $Y$  can be solved in poly-time since  $Y$  is in NP.

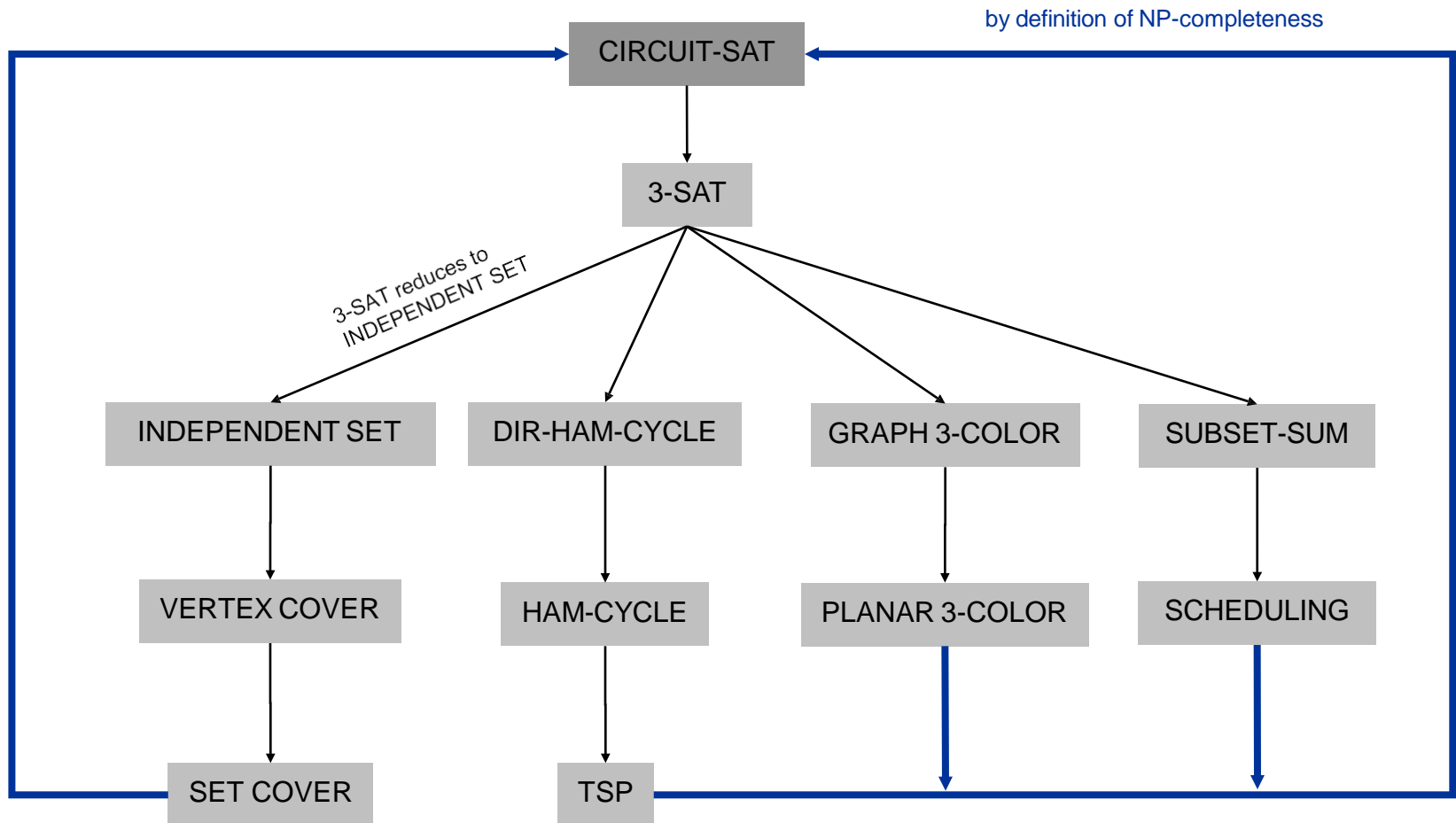
**Pf.**  $\Rightarrow$  Suppose  $Y$  can be solved in poly-time.

- Let  $X$  be any problem in NP. Since  $X \leq_p Y$ , we can solve  $X$  in poly-time. This implies  $NP \subseteq P$ .
- We already know  $P \subseteq NP$ . Thus  $P = NP$ . ▪

**Fundamental question.** Do there exist "natural" NP-complete problems?

# NP-Completeness

**Observation.** All problems below are NP-complete and polynomial reduce to one another!



# NEXT LECTURE

- NP-Completeness
- Review

Week	Date	Topic
1	21-Feb	Introduction. Some representative problems
2	28-Feb	Stable Matching
3	7-Mar	Basics of algorithm analysis.
4	14-Mar	Graphs (Project 1 announced)
5	21-Mar	Greedy algorithms-I
6	28-Mar	Greedy algorithms-II
7	4-Apr	Divide and conquer (Project 2 announced)
8	11-Apr	Dynamic Programming I
9	18-Apr	Dynamic Programming II
10	25-Apr	Network Flow-I (Project 3 announced)
11	2-May	<b>Midterm</b>
12	9-May	Network Flow II
13	16-May	NP and computational intractability-I
14	23-May	NP and computational intractability-II