P versus NP, NP hard and NP complete

Admin

- Assignment 6 is due tonight
- Assignment 7 has been released:
    - Typically would be due next Wed (April 21)
    - But....
        - April 21 and 22 are health days ! Woohoo!
        - Please use them to relax and rest
- As a result, is due next-to-next Wed (April 28), and is a bit longer
- Highly encourage you to divide it into two parts
    - Do questions 1-3 before health days, 4-6 after

Reminders

Become a Teaching Assistant or Tutor for computer science

This is a great opportunity to teach other students and to work closely with faculty- it can be a very rewarding experience.

Application can be found here: https://csci.williams.edu/tatutor-application/

Applications due
April 26, 202

Reminders

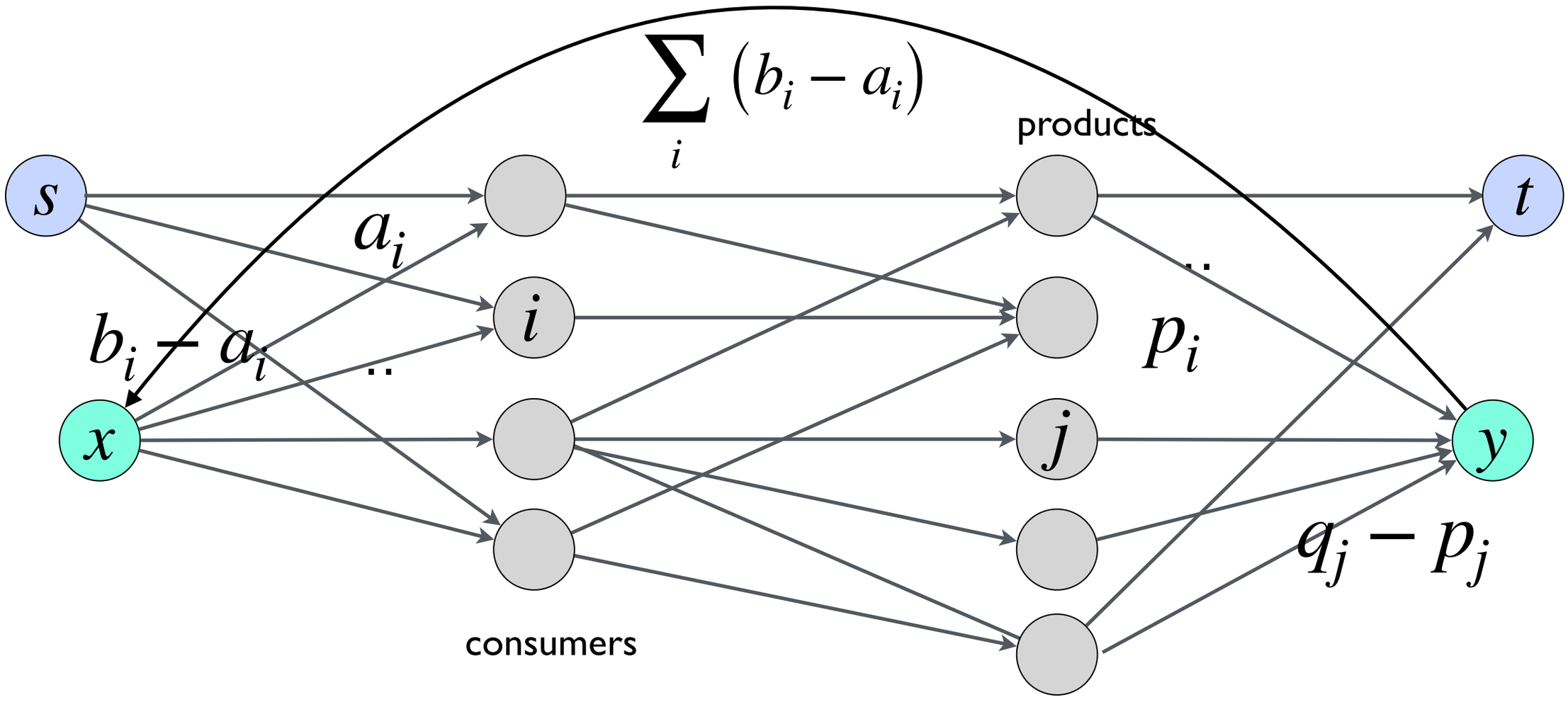
- Fill out the **CSCI 256 TA survey** !
    - https://forms.gle/cmUQdkZQbFaGzWWe6 (you must have received an email from Lauren with link)

Reminders

- **Colloquium tonight** 7pm: Conversation about work at the intersection of art, music, and computer science

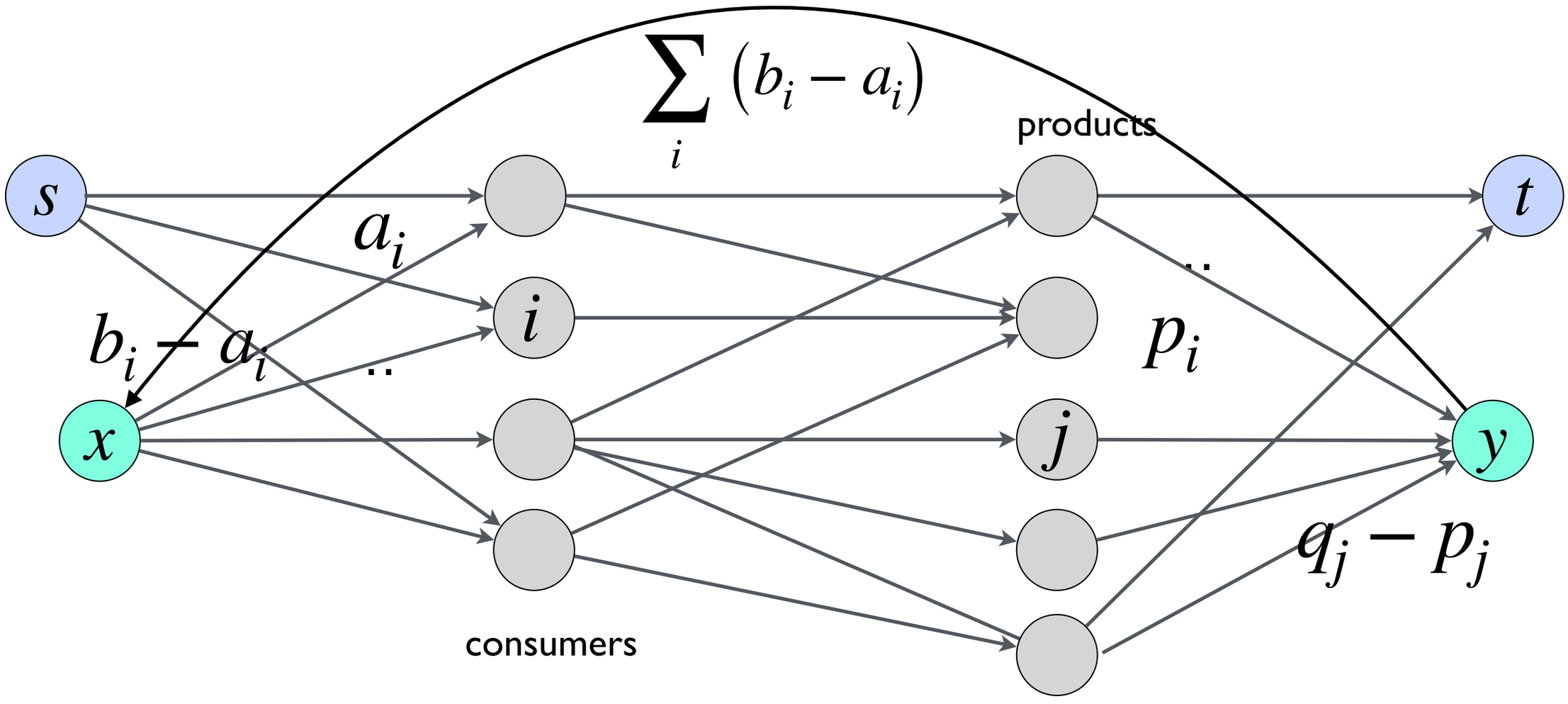
Reduction from Last Lecture

- Instead of going over the correctness today, created a handout
- Serves the additional goal of showing you how to structure your reductions



Circulations & Lower Bounds

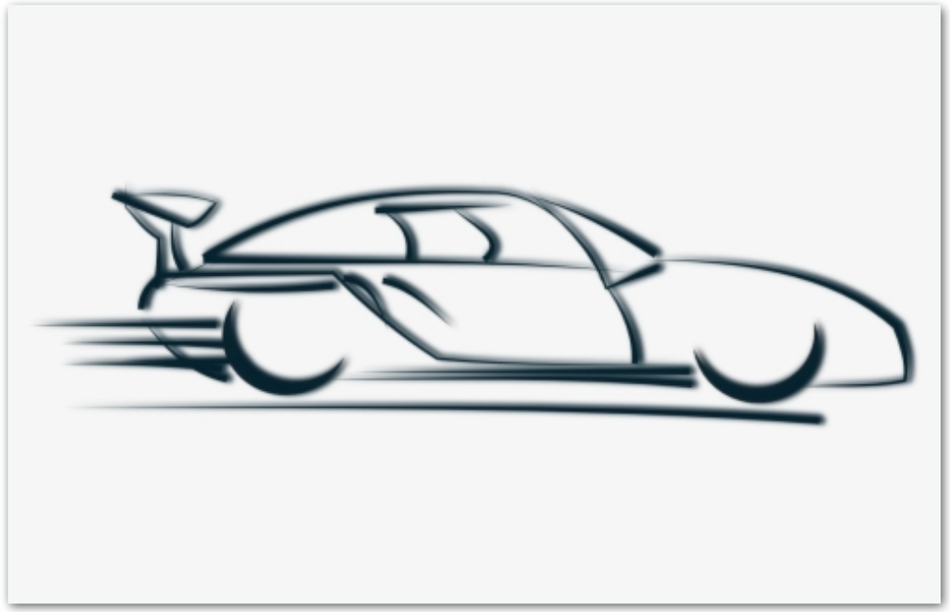
- The textbook handles reductions with lower bounds using the concept of "circulations with demands"
- We avoided this framework to do a more direct reduction; but thats why we may have "circulation" that may not originate at s



Today: Overview and Classes P and NP

What We Focused on So Far

- Number of inversions—can solve in $O(n^2)$ easily, but $O(n \log n)$ with a clever algorithm
- Weighted interval scheduling—can try all combinations in $O(2^n)$, but can solve in $O(n^2)$ using Dynamic programming
- Network flow seems very difficult to solve, but we saw how to solve it in $O(nmC)$ and even $O(n^2m)$



Shifting Focus

- Most of the class has been about how to efficiently solve problems
- Now we're going to shift to a higher-level question
  - What problems can a computer solve efficiently?
  - What problem can a computer not solve efficiently?

Efficiency: Polynomial time

- What problems can a computer solve in polynomial time?
- What problems can a computer (probably) **not** solve in polynomial time?

Technical Setup

- We will now focus on **decision problems** — problems with a yes or no answer
  - "Does this directed graph have a topological order?"
  - Is this graph bipartite?
  - Do these two strings have Edit Distance at most 10?
  - Does this flow network have a max flow of at least 20?

Technical Setup

- Most problems have a decision analog
    - Find the flow of this network -> “does this network have flow at least k?”
    - Find the optimal schedule of these intervals -> “can we schedule at least k intervals?”
- These are (essentially) the same—-after all, can always binary search for the optimal value

Technical Setup

- Decision problem means that every solution is “yes” or “no”
- Yes instances can represented as a set of inputs A
  - $x \in A$ means that the solution to x is “yes”
  - $x \notin A$ means that the solution to x is “no”
- So can have (for example): A is the set of all flow networks which permit flow at least k
- Or can have: A is the set of all pairs of strings (a, b) where the edit distance between a and b is at most k

Class P

- **P**: the class of decision problems that can be solved in polynomial time
  - Edit distance is in **P**
  - Max flow is in **P**
  - Bipartite matching is in **P**
  - Knapsack?
    - dynamic prorgamming algorithm we saw is pseudo-polynomial! So we don’t know yet

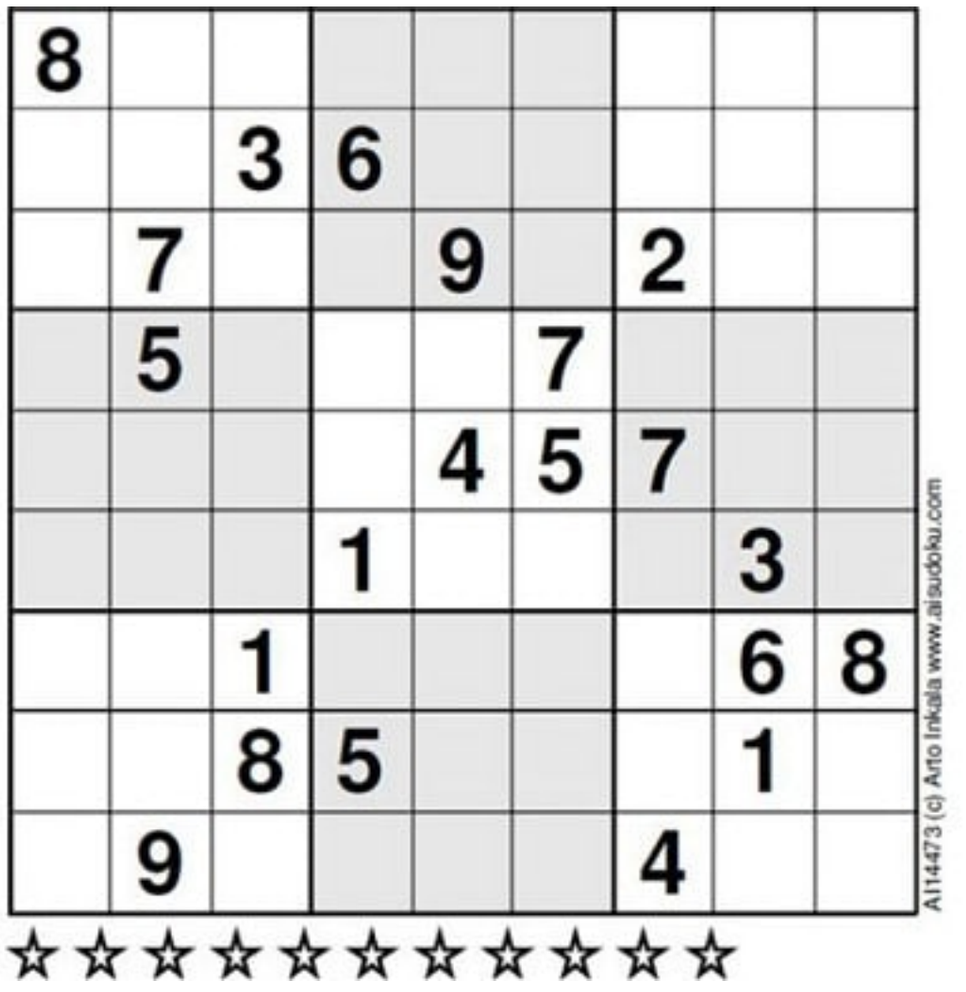
Class NP

Class NP—Intuition

- **NP** is the class of problems that can be *verified* in polynomial time

- If I give you helpful information, say a proposed solution, you can easily check that it is correct

Class NP—Intuition



5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

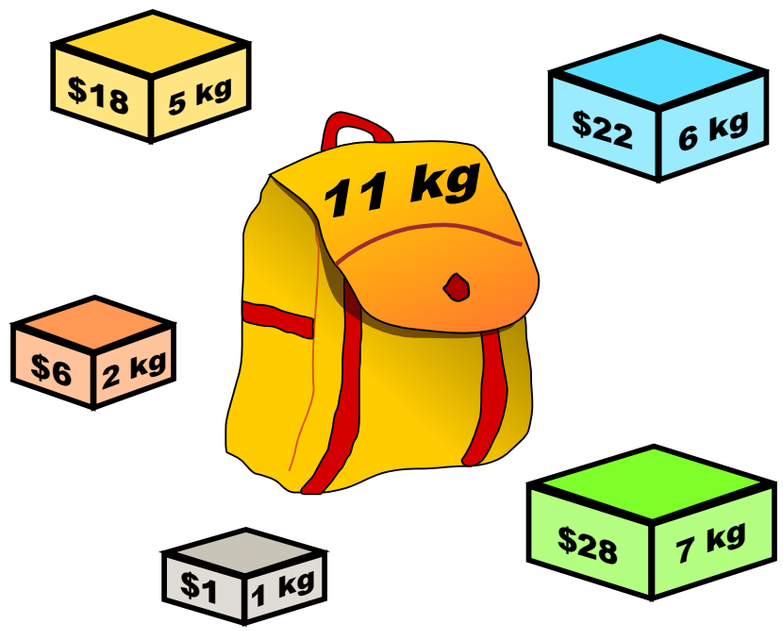
Sudoku is easy if I give you information (by giving you the solution). So sudoku is in **NP**

Class NP—Intuition

- Example (Knapsack capacity C = 11)
  - {3, 4} has value $40 (and weight 11)

i	v_i	w_i
1	$1	1 kg
2	$6	2 kg
3	$18	5 kg
4	$22	6 kg
5	$28	7 kg

knapsack instance
(weight limit W = 11)



Knapsack is easy if I give you information (by giving you the solution). So knapsack is in NP

Class NP: Formally

Definition. Algorithm $V(s, c)$ is a verifier for problem X if for every input $x \in X$ there exists a certificate, a string c, such that $V(s, c) =$ yes iff $s \in X$.

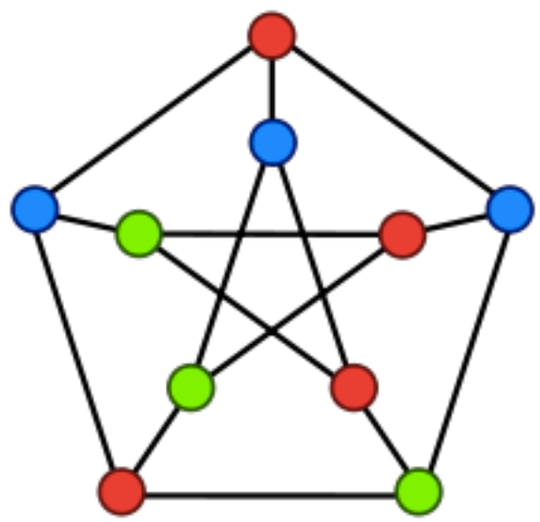
Definition. NP = set of decision problems for which there exists a polynomial-time verifier

- $V(s, c)$ is a polynomial time algorithm
- Certificate c is of polynomial size:
    - $|c| \leq p(|s|)$ for some polynomial $p(\,.\,)$

Graph-Coloring $\in$ NP

Graph-Coloring. Given a graph $G = (V, E)$, is it possible to color the vertices of G using only three colors, such that no edge has both end points colored with the same color.

- Graph-Coloring $\in$ NP
- **Certificate:** assignment of colors to vertices
- **Poly-time verifier:** check if at most 3 colors used, check for each edge if ends points same color or not

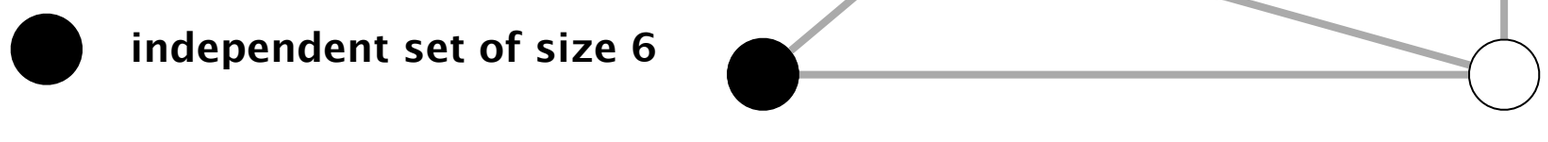


A 3-colorable graph

Independent Set

- Given a graph $G = (V, E)$, an independent set is a subset of vertices $S \subseteq V$ such that no two of them are adjacent, that is, for any $x, y \in S$, $(x, y) \notin E$

- **IND-SET Problem.**
  Given a graph $G = (V, E)$ and an integer k, does G have an independent set of size at least k?



IND-SET $\in$ NP

- Given a graph $G = (V, E)$, an independent set is a subset of vertices $S \subseteq V$ such that no two of them are adjacent, that is, for any $x, y \in S$, $(x, y) \notin E$

- **IND-SET Problem.** Given a graph $G = (V, E)$ and an integer k, does G have an independent set of size at least k?

- **IND-SET** $\in$ NP.
    - **Certificate:** a subset of vertices
    - **Poly-time verifier:** check if any two vertices are adjacent and check if size is at least k

Quick Question

- Is **P** ⊆ **NP**?
    - If a problem is in **P**, does that mean that it is in **NP**?
- Yes! If a problem can be solved in polynomial time, it can be verified in polynomial time.
- Just solve directly (Can just set $c = ""$)

Satisfiability

- The next problem is the classic example of a problem in **NP**
  - (and, as we'll soon see, probably not in **P**)
- Many different small variations on the same problem (we'll see a couple)
- **Idea**: given a logical equation, can we assign "true" and "false" to the variables to satisfy the equation?

SAT, 3SAT $\in$ NP

- **SAT**. Given a CNF formula ϕ, does it have a satisfying truth assignment?
- **3SAT.** A SAT formula where each clause contains exactly 3 literals (corresponding to different variables)

- $\phi = (\overline{x_1} \vee x_2, \vee x_3) \wedge (x_1, \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$
- Satisfying instance: $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0$, where 1 : true, 0 : false
- SAT, 3-SAT $\in$ NP
    - Certificate: truth assignment to variables
    - Poly-time verifier: check if assignment evaluates to true

P versus NP

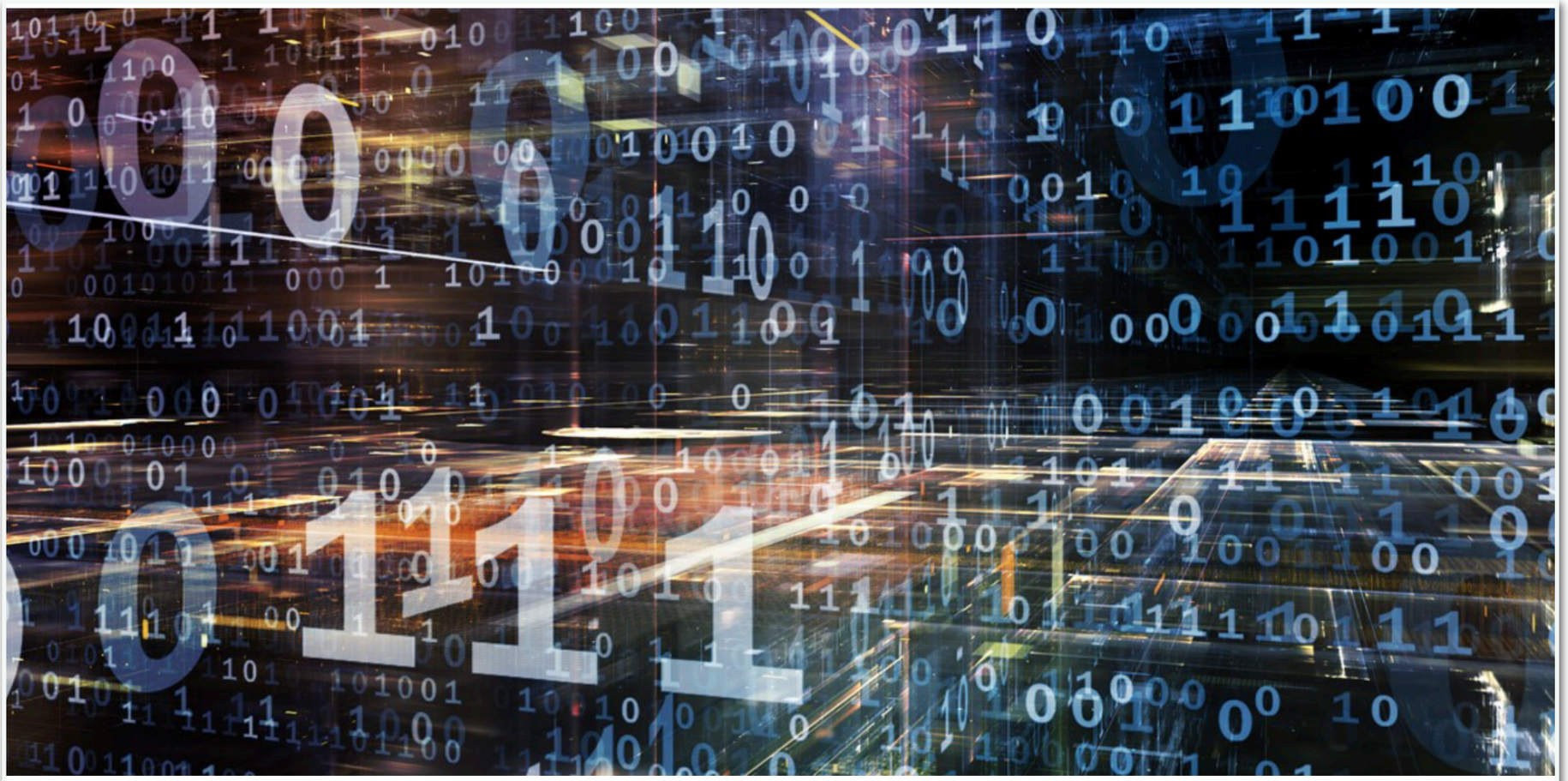
P vs NP

- We know that every problem in **P** is also in **NP**
- What about the reverse? That is to say:
  - If a problem can be efficiently *verified*, does that mean it can be efficiently solved in the first place?
  - Or, do there exist problems that can be verified quickly that are *impossible* to solve quickly?

Why Do We Care?

- If **P = NP**, the consequences:
  - Lots of important problems can be solved quickly!
  - Can build things better, faster, more efficiently
  - (Public key) cryptography does not exist
- If **P ≠ NP**:
  - Many problems can’t be solved quickly
  - Can stop trying to solve them
  - We believe this to be true

Million Dollar Question: P vs NP



P vs NP and the $1M Millennium Prize Problems

What's the most difficult way to earn $1M US Dollars?

https://medium.com/@mpreziuso/

Million Dollar Question: P vs NP

- The biggest open problem in computer science
- One of the biggest in math as well
- We are not even close to solving it!

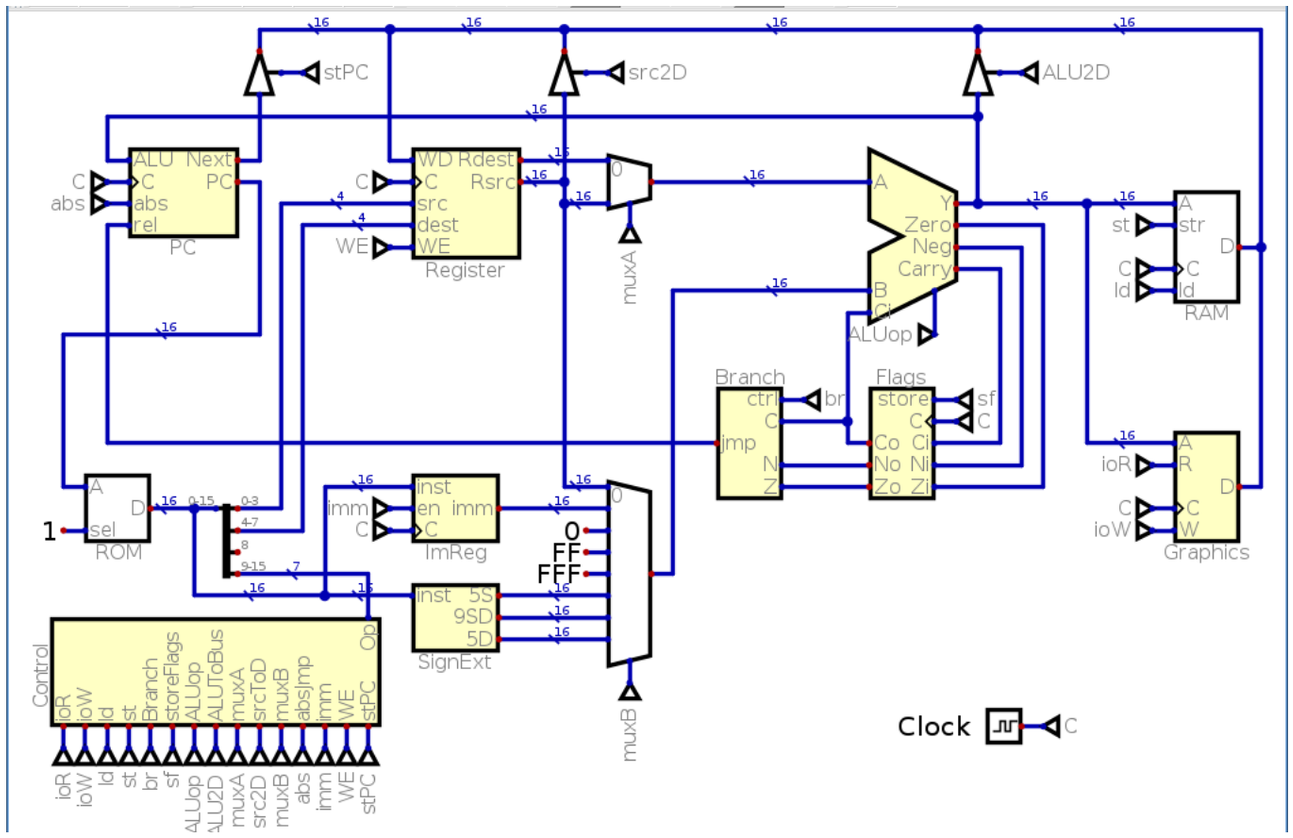
NP-hard and NP-Complete Problems

Cook-Levin Theorem

- If **SAT** can be solved in polynomial time, then *any* problem in **NP** can be solved in polynomial time
- So if **SAT** can be solved in polynomial time, then **P = NP**
- How is this possible?

Cook-Levin Theorem

- Idea: any computer program can be represented by a circuit.
- Solve **SAT** in poly time -> can figure out the answer given by the circuit for NP problem in poly time



You'll see the proof in CS 361

NP-Hard Problems

- A problem X is **NP-hard** if:
  - If X can be solved in polynomial time, then any problem in **NP** can be solved in polynomial time
  - That is, if X can be solved in polynomial time, then

    P = NP

What Does This Mean?

- We think that, probably, **P** ≠ **NP**
- So if a problem is **NP**-hard, then you probably cannot obtain a polynomial-time algorithm for it

Classifying Problems as Hard

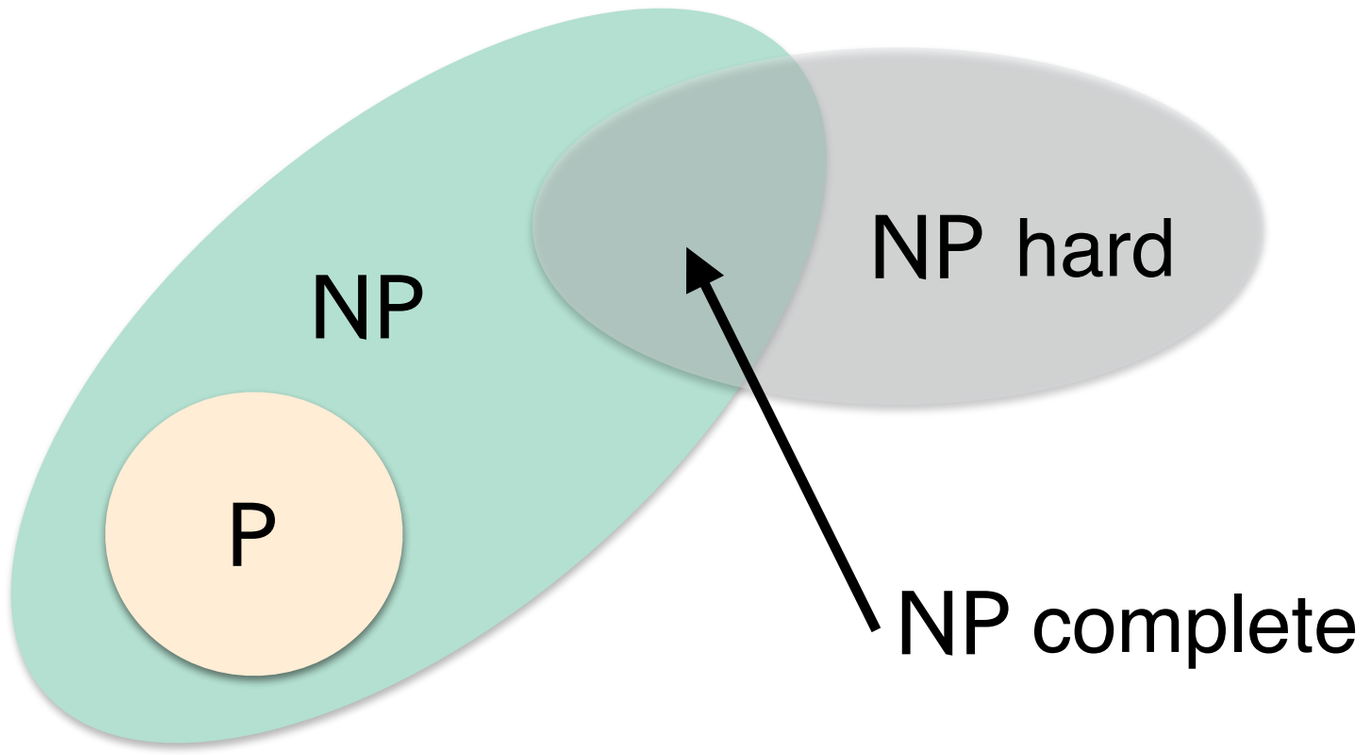
- We are frustratingly unable to prove a lot of problems are **impossible** to solve efficiently
- Instead, we say problem X is likely very hard to solve by saying, *if a polynomial-time algorithm was found for* X*, then something we all believe is impossible will happen*
- Instead we say X is **NP**-hard: if $X \in \mathbf{P}$, then $\mathbf{P} = \mathbf{NP}$

Classifying Problems as Hard

- Instead, we say problem X is likely very hard to solve by saying, *if a polynomial-time algorithm was found for X, then something we all believe is impossible will happen*
- Instead we say X is **NP**-hard: if $X \in$ **P**, then **P** = **NP**
- (Erickson) Calling a problem **NP** hard is like saying, *"If I own a dog, then it can speak fluent English"*
  - You probably don't know whether or not I own a dog, but you are definitely sure I don't own *a talking dog*
  - Corollary: No one should believe that I own a dog
- If a problem is **NP** hard, no one should believe it can be solved in polynomial time

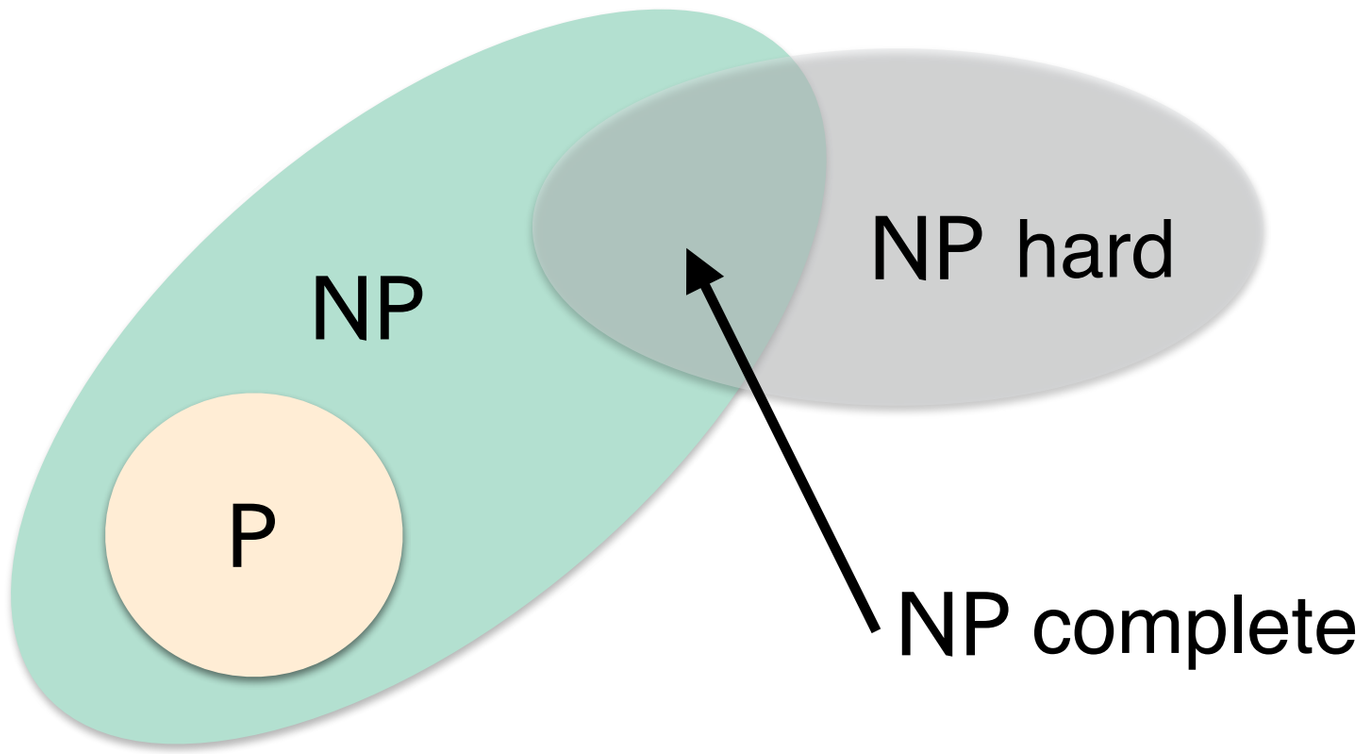
NP Completeness

- **Definition.** A problem X is NP complete if X is NP hard and $X \in$ NP
- SAT is **NP** complete
    - SAT $\in$ **NP**: given an assignment to input gates (certificate), can verify whether output is one or zero in poly-time
    - SAT is **NP** hard (Cook-Levin Theorem)



Summary

- X is **NP**-hard **NP**-hard $\Leftrightarrow$ if $X \in$ **P**, then **P** = **NP**
- A problem X is NP complete if X is NP hard and $X \in$ **NP**
- Alternate definition of NP hard:
  - X is NP hard if all languages in NP reduce it to in polynomial time
- Thus, NP-complete problems are the hardest problems in NP



Acknowledgments

- Some of the material in these slides are taken from
  - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
  - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)