Week 10 - Friday

COMP 4500

Last time

- What did we talk about last time?
- Bipartite matching

Questions?

Logical warmup

- Two vegetarians and two cannibals are on one bank of a river
- They have a boat that can hold at most two people
- Come up with a sequence of boat loads that will convey all four people safely to the other side of the river
- The cannibals on any given bank cannot outnumber the vegetarians…or else!

NP-completeness

Efficient algorithms

- Until this point in the course, we have studied efficient algorithms
  - Polynomial time algorithms
- Perhaps more important than the list of algorithms we studied were the principles behind such algorithms
- Are there problems for which there are no efficient algorithms?
- The study of computational complexity tries to rank all problems based on their difficulty

Hard problems

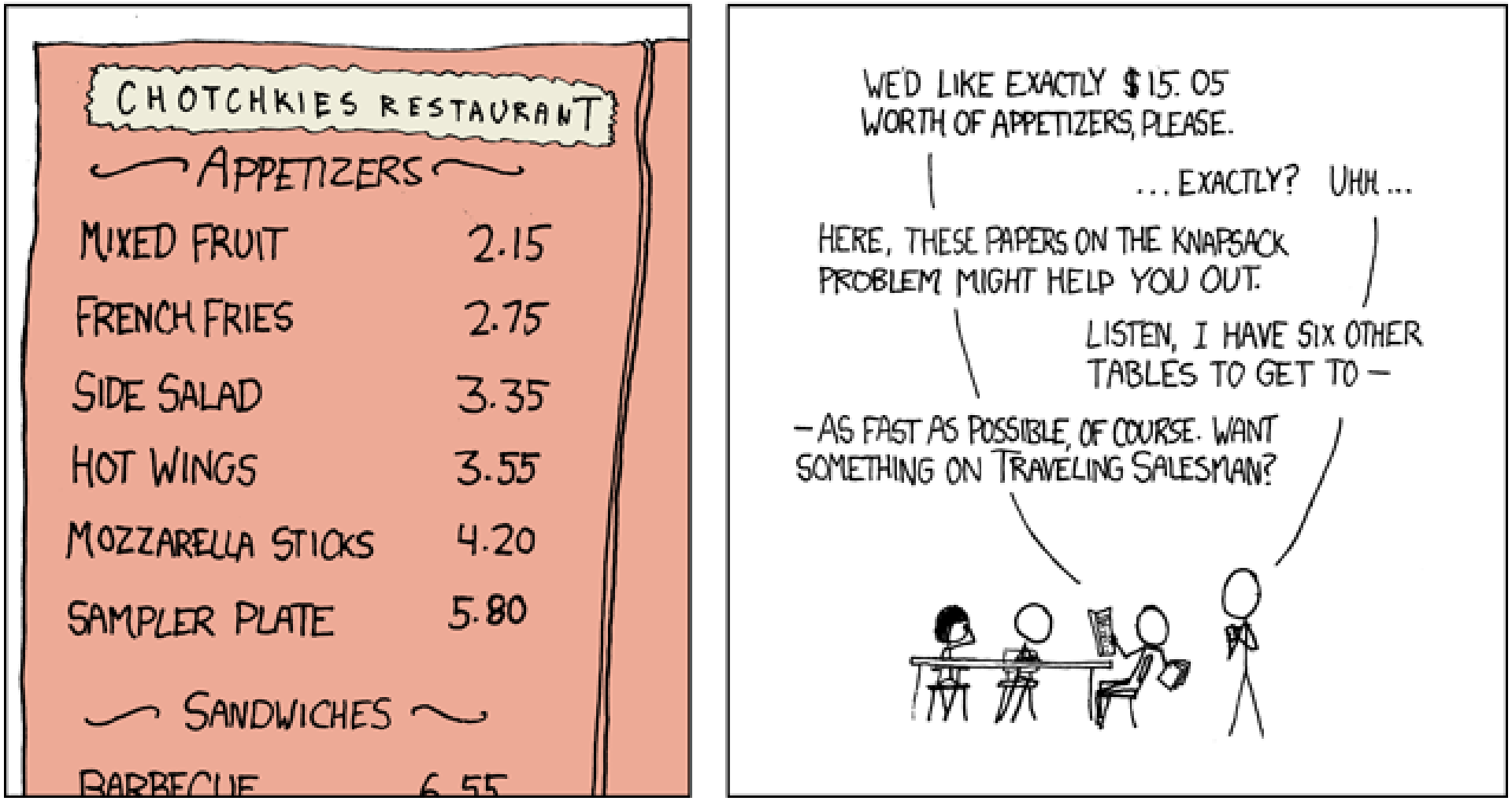
- Some very hard problems have been proven to have no polynomial-time algorithms
- Unfortunately, a large number of important problems in optimization, AI, combinatorics, logic, and other areas have resisted categorization
  - We have not been able to find polynomial-time algorithms for these problems
  - **But** we have also failed to prove that polynomial time algorithms cannot exist

NP-complete

- **NP-complete** problems are one of the classes in this gray area
- All NP-complete problems are essentially equivalent because a polynomial-time algorithm for one such problem would imply a polynomial-time algorithm for all of them
- Thus, we can think of this set of thousands of problems as really **one** fundamental problem

My Hobby:

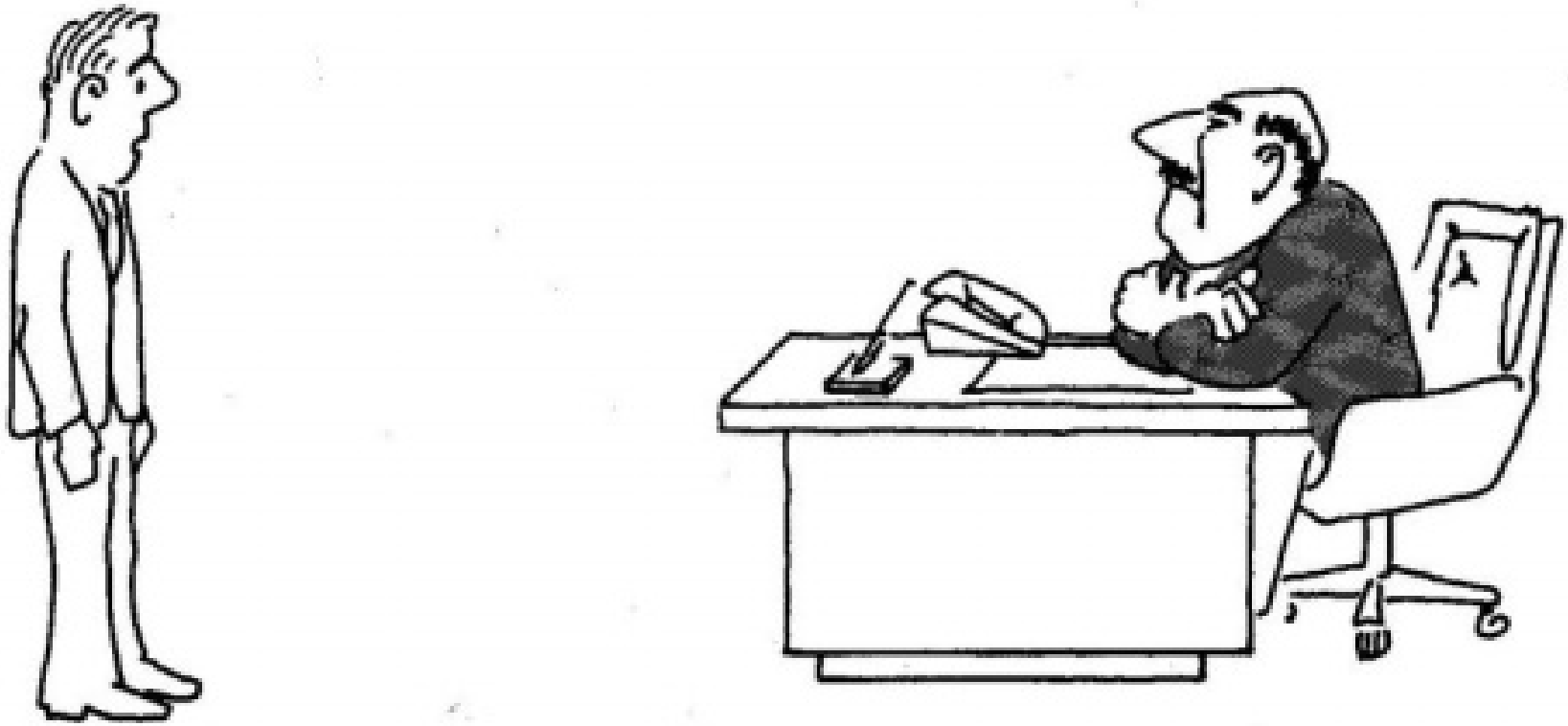
Embedding NP-complete problems in restaurant orders



From xkcd: https://xkcd.com/287/

What happens when you need to solve an NP-complete problem?

- Sometimes, very small instances or very constrained instances of NP-complete problems are solvable
- Otherwise, we believe that NP-complete problems are computationally infeasible to solve, even though we can't prove it
- Why look for an efficient algorithm for a problem when no one can find an efficient one for all these famous problems?



"I can't find an efficient algorithm, I guess I'm just too dumb."

"I can't find an efficient algorithm, but neither can all these famous people."

Three-Sentence Summary of Polynomial-Time Reductions

Polynomial-Time Reductions

Characterizing hardness

- How can we compare the hardness of problems?
- How are we able to say that NP-complete problems are all the same level of hardness?
- We want a formal way to describe that problem ***X*** is at least as hard as problem ***Y***
- The tool we use to argue that ***X*** is at least as hard as ***Y*** is called a **reduction**

Reductions

- We imagine that we have a black box that can solve problem ***X*** instantly
- Can any instance of problem ***Y*** be solved by doing polynomial work to format the input for ***Y*** into input for ***X*** followed by a polynomial number of calls to the black box that solves ***X***?
- If the answer is **yes**, we write $Y \leq_P X$ and say that ***Y*** is polynomial-time reducible to ***X***

Why are reductions useful?

- Imagine that there is a polynomial-time algorithm to solve ***X***
- We can solve ***Y*** with a polynomial prep time plus polynomial calls to ***X***
- That means that ***Y*** can be solved with polynomial work plus polynomial work times polynomial work
  - Thus, ***Y*** can also be solved in polynomial time
- Formally:
  - Suppose $Y \leq_P X$. If ***X*** can be solved in polynomial time, then ***Y*** can be solved in polynomial time.

What about the other direction?

- We didn't really study logic in this class
  - Since we assumed you got everything you needed in MATH 1230 and COMP 2230
- If you have an implication $p \rightarrow q$ that is true, its **contrapositive** $\sim q \rightarrow \sim p$ is also true
- Implication:
  - Suppose $Y \leq_P X$. If X can be solved in polynomial time, then Y can be solved in polynomial time.
- Contrapositive:
  - Suppose $Y \leq_P X$. If Y cannot be solved in polynomial time, then X cannot be solved in polynomial time.

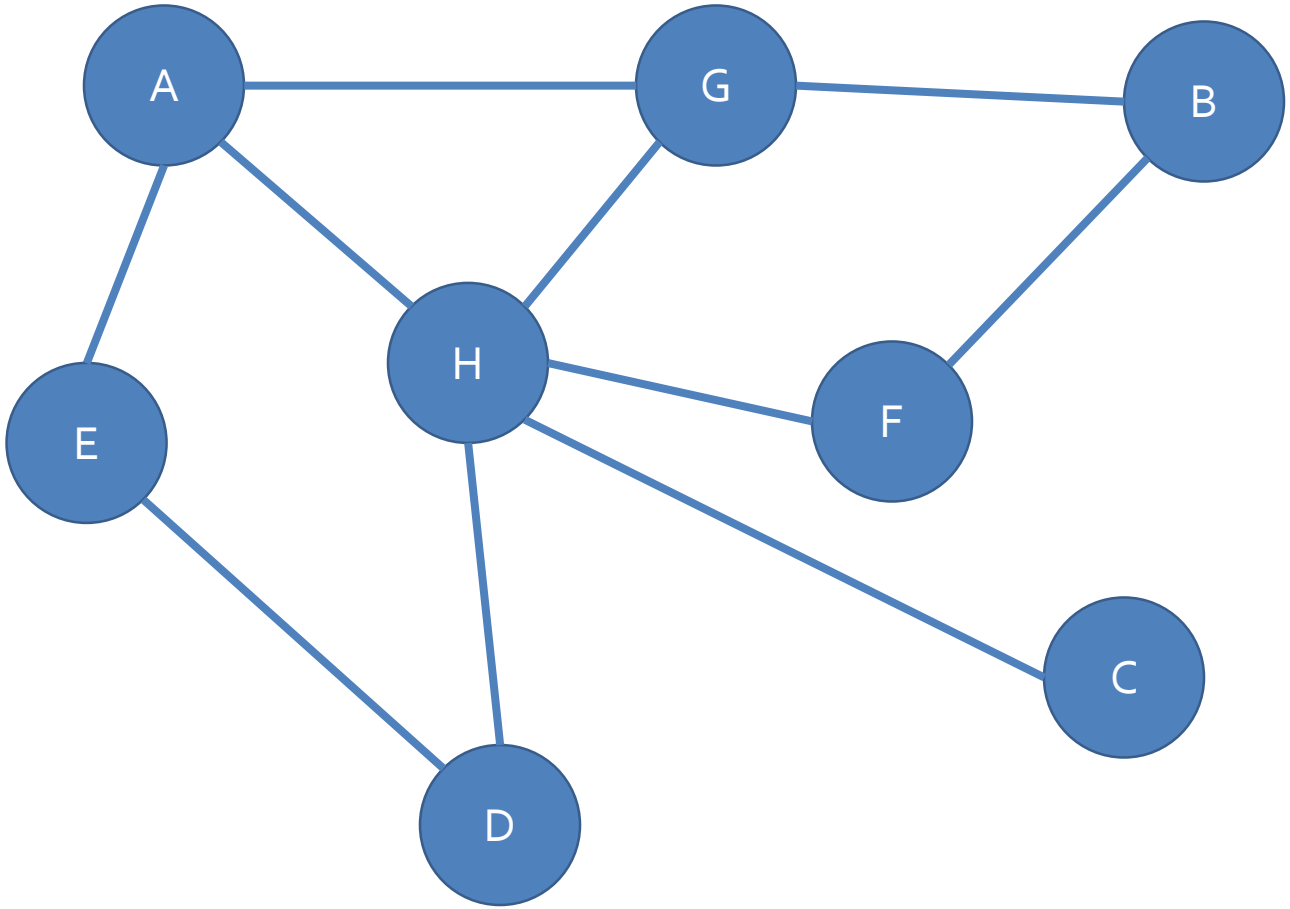
A house of cards

- Reductions are not one tool in our arsenal…
  - They're pretty much our **only** tool for comparing the difficulty of problems
- If you can solve a problem with a polynomial-time algorithm, you know it's polynomial
- For NP-complete problems (and even some higher classes), we have no way of being sure
- All we can do is say that one problem is at least as hard as another

Independent set

- Recall the independent set graph problem
- Given an undirected graph, find the largest collection of nodes that are not connected to each other
- Practical application:
  - Nodes represent friends of yours
  - An edge between those two nodes means they hate each other
  - What's the largest group of friends you could invite to a party if you don't want any to hate each other?

Independent set example



Hardness of independent set

- Independent set is an NP-complete problem
- We don't know a polynomial-time algorithm for it, but we don't know how to prove that there isn't one
- We just stated the **optimization** version of independent set:
  - Find the largest independent set
- But there is also a **decision** version:
  - Given a graph ***G*** and a number ***k***, does ***G*** contain an independent set of size at least ***k***?

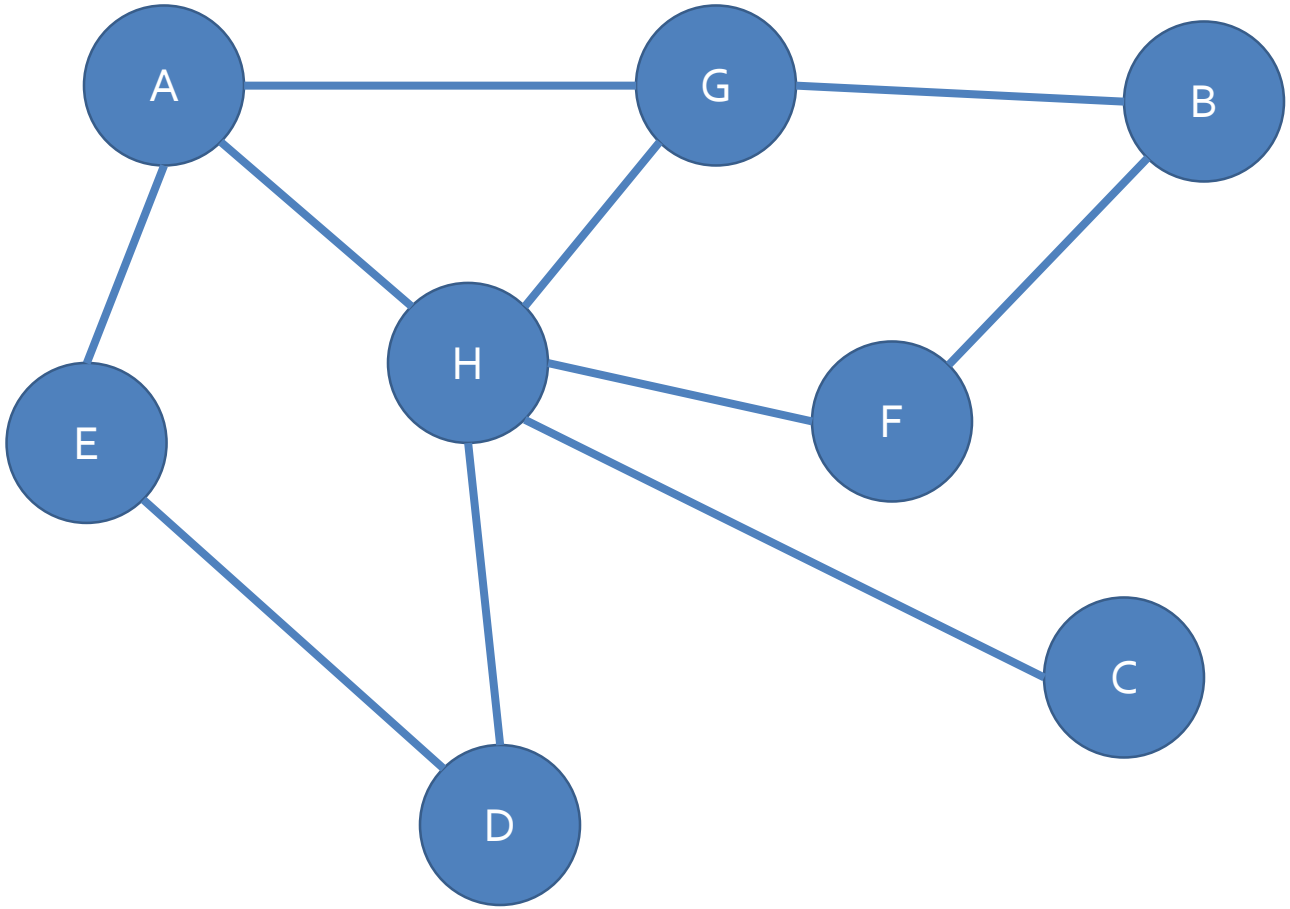
Optimization vs. decision problems

- Optimization problems feel more natural to us
  - We want to know what the largest independent set is
- However, decision problems are usually used for problem reductions
  - Why? The output is simpler.
- If you can efficiently solve the decision version, you can get a lot of information about the optimization version
  - Use a binary search on ***k*** to find the size of the largest independent set
- Many people believe the fundamental computational hardness of a decision version is roughly the same as the optimization version, for most problems

Vertex cover

- The vertex cover problem is another graph problem:
  - Given a graph $G = (V, E)$, we say that a set of nodes $S \subseteq V$ is a **vertex cover** if every edge $e \in E$ has at least one end in S
  - In other words, find a set of vertices such that all edges touch at least one
- It's easy to find a big vertex cover: all vertices
- It's hard to find a small one
- Decision version:
  - Given a graph G and a number k, does G contain a vertex cover at size at most k?

Find a small vertex cover on this graph



Relationship between independent set and vertex cover

- **Claim:** Let $G = (V,E)$ be a graph. S is an independent set if and only if its complement $V - S$ is a vertex cover.
- **Proof:**
  - Suppose that S is an independent set. Consider an edge $e = (u,v)$. Since S is independent, it cannot be the case that both u and v are in S. Thus, one of them must be in $V - S$. It must be the case that every edge has at least one end in $V - S$, so $V - S$ is a vertex cover.

Proof continued

- Suppose that $V - S$ is a vertex cover. Consider any two nodes u and v in S. If they were joined by edge e, then neither end of e would lie in $V - S$, contradicting the assumption that $V - S$ is a vertex cover. Thus, it must be the case that no two nodes in S are joined by an edge, so S must be an independent set.

■

Independent set $\leq_P$ vertex cover

- **Proof:**
  - If we have a black box to solve vertex cover, we can decide whether ***G*** has an independent set of size at least ***k*** by asking the black box whether ***G*** has a vertex cover of size at most ***n*** – ***k***.
  - ■

Vertex cover $\leq_P$ independent set

- **Proof:**
  - If we have a black box to solve independent set, we can decide whether ***G*** has a vertex cover of size at most ***k*** by asking the black box whether ***G*** has an independent set of size at least ***n*** – ***k***.
  - ■

Reductions

- We don't have an efficient algorithm for either independent set or vertex cover
- Even so, we know that they are both approximately as hard as each other
- These two problems are so closely related that it seems like this kind of reduction might not be generally applicable
- However, the entire class of NP-complete problems are **all** reducible to each other, so it's a pretty general technique

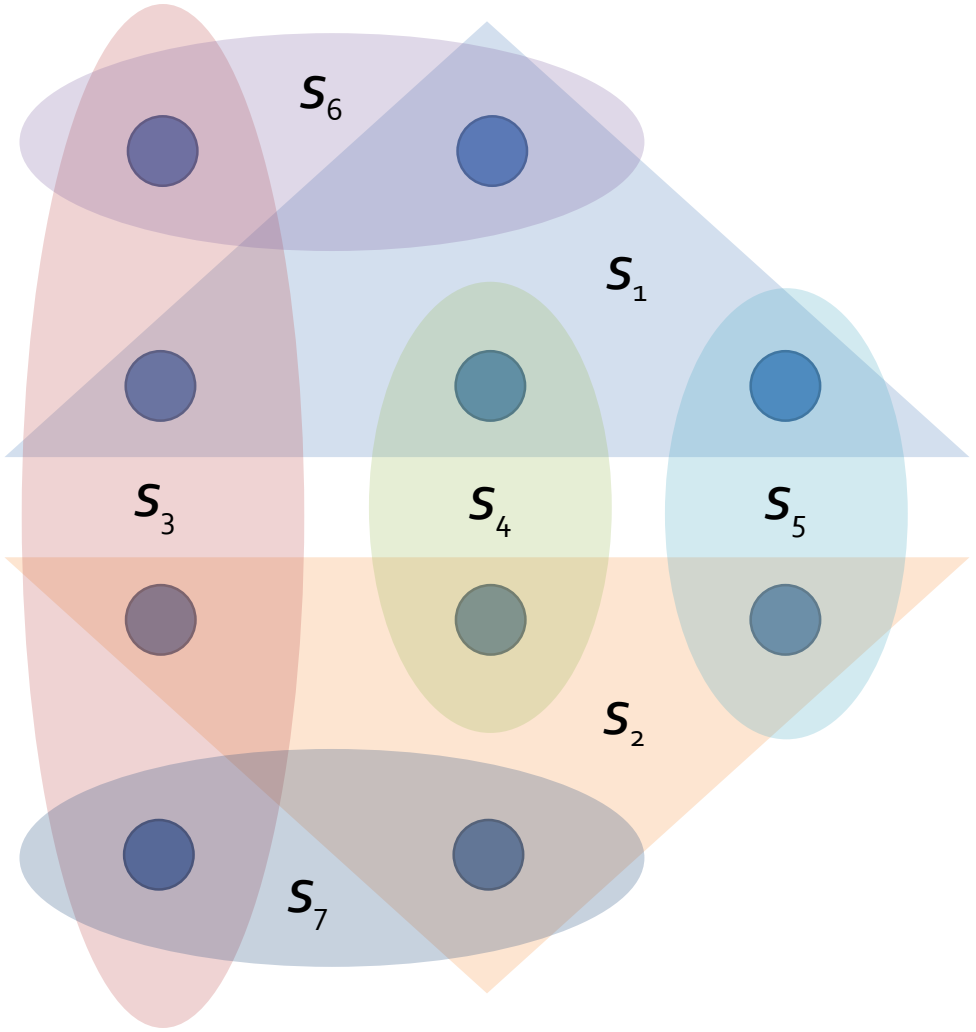
Packing and covering

- Independent set is a **packing problem**
  - We want to pack in as many things as possible, subject to constraints
- Vertex cover is a **covering problem**
  - We want to cover everything (edges, in this case) with the smallest number of things (vertices in this case)
- There are many NP-complete problems that fall into categories of packing and covering problems

Set cover

- Given:
  - Set U of n elements
  - Collection of sets $S_1, S_2, \ldots, S_m$ of subsets of U
  - A number k
- Is there a collection of at most k subsets whose union is equal to all of U?

Set cover example



Vertex cover $\leq_P$ set cover

- Proof:
  - Suppose we have a black box that can solve set cover.
  - Consider an instance of vertex cover on graph $G = (V, E)$ with number k.
  - We want to cover all the edges in E, so we create a set cover problem in which the universe set U is E.
  - Each vertex in V covers some set of edges, so for every vertex $i \in V$, we add a set $S_i \subseteq U$ where the elements of S_i are all the edges incident on i.

Proof continued

- We claim that U can be covered with at most k of the sets $S_1, S_2, \ldots, S_n$ if and only if G has a vertex cover of size at most k.
- If $S_{i_1}, S_{i_2}, \ldots S_{i_l}$ are $l \leq k$ sets that cover U, then every edge in G is incident to one of the vertices $i_1, i_2, \ldots, i_l$. Thus, the set $\{i_1, i_2, \ldots, i_l\}$ is a vertex cover in G of size $l \leq k$.
- Conversely, if $\{i_1, i_2, \ldots, i_l\}$ is a vertex cover in G of size $l \leq k$, then the sets $S_{i_1}, S_{i_2}, \ldots S_{i_l}$ cover U.
- For any vertex cover problem, we make an instance of set cover as described, pass it to our black box, and answer yes if and only if the black box answers yes.

■

Upcoming

Next time…

- Reductions via gadgets
- Certificates and the definition of NP

Reminders

- Finish Homework 5
  - **Due tonight by midnight!**
- Read 8.2 and 8.3
- Study for Exam 3
  - Monday after next!