Week 12 - Wednesday

COMP 4500

Last time

- What did we talk about last time?
- Exam 3
- Before that:
  - Proving problems NP-complete

Questions?

Assignment 6

Logical warmup

- A pot contains 75 white beans and 150 black ones
- Next to the pot is a large pile of black beans.
- The (insane) cook removes the beans from the pot, one at a time, according to the following strange rule:
  - He removes two beans from the pot at random
  - If at least one of the beans is black, he places it on the bean-pile and drops the other bean, no matter what color, back in the pot
  - If both beans are white, on the other hand, he discards both of them and removes one black bean from the pile and drops it in the pot
- After each step of this procedure, the pot has one fewer bean in it
- Eventually, just one bean is left in the pot
- What color is it?

Three-Sentence Summary Listing Lots of NP-Complete Problems

Showing Lots of Problems are NP-Complete

Why are we showing that lots of problems are NP-complete?

- You need to know if your boss gives you the (probably impossible) task of writing a program to solve an **NP-complete** problem
- Trying to understand these reductions will (hopefully) help you remember a number of **NP-complete** problems
- Finally, these reductions are impressive accomplishments of computer science

Sequencing problems

- We've seen **NP-complete** problems for sets and satisfying Boolean variables
- Another important category are sequencing problems where we want to find the right permutation of a collection of objects
- How many permutations are there of ***n*** objects?

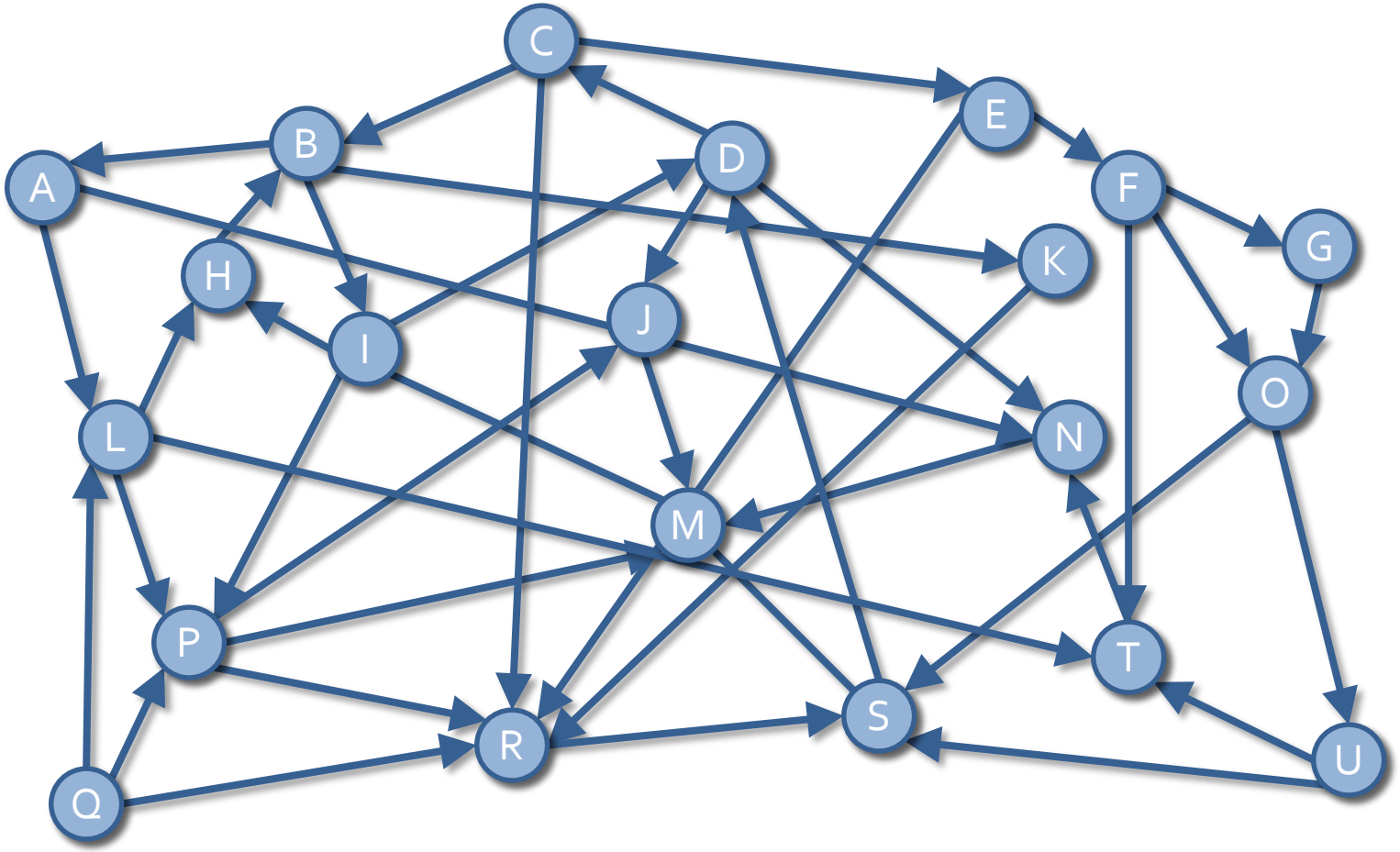
Traveling salesman problem

- The traveling salesman problem (TSP) supplies n cities $v_1, v_2, \ldots, v_n$
- All cities are connected with a directed edge (v_i, v_j) of length $d(v_i, v_j)$
- Starting at city v_1, find a **tour** of minimum distance that visits every city exactly once and returns to v_1
- Applications: routing problems, path planning, circuit layout in VLSI
- Decision version:
  - Given a set of distances on n cities and a bound D, is there a tour of length at most D?

Hamiltonian cycle problem

- Given a directed graph $G = (V,E)$, a cycle C in G is a Hamiltonian cycle if it visits each vertex exactly once
- Decision problem:
  - Given a directed graph G, does it contain a Hamiltonian cycle?

Find a Hamiltonian Cycle



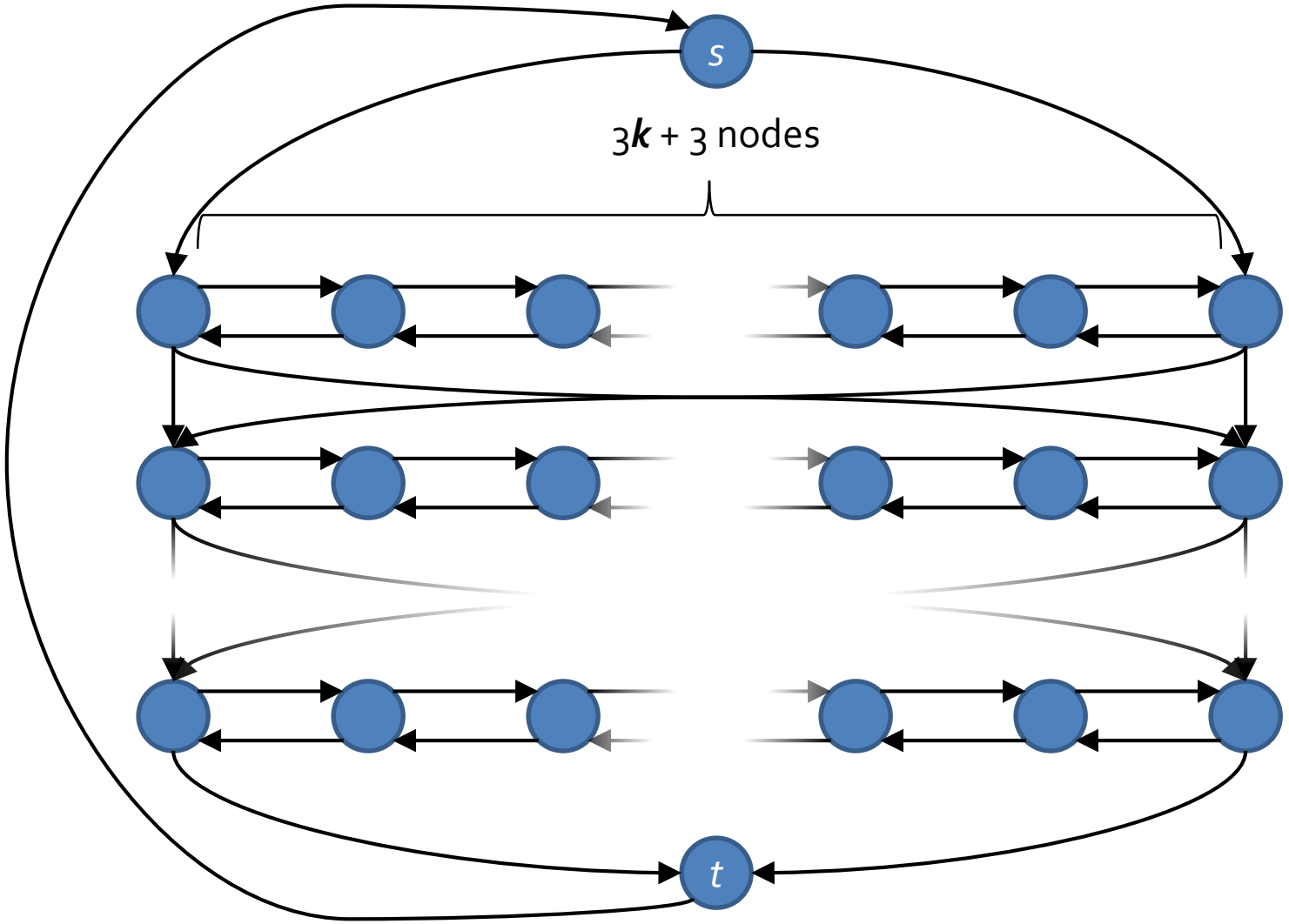
Hamiltonian cycle is NP-complete

- **Proof:**
  - A list of vertices giving such a cycle could be checked in polynomial time, showing that Hamiltonian cycle is in NP.
  - We can reduce 3-SAT to Hamiltonian cycle in the following way.
  - Consider an instance of 3-SAT with variables $x_1, x_2, \ldots, x_n$ and clauses $C_1, C_2, \ldots, C_k$
  - Imagine a graph with 2^n different Hamiltonian cycles, corresponding to the 2^n truth assignments to the variables.

Proof continued

- Specifically, imagine n paths $P_1, P_2, \ldots, P_n$
- P_i consists of nodes $v_{i1}, v_{i2}, \ldots, v_{ib}$ where $b = 3k + 3$
- There is an edge from v_{ij} to $v_{i,j+1}$ and an edge from $v_{i,j+1}$ to v_{ij}, in other words, in both directions
- We hook these paths together by putting edges from v_{i1} to $v_{i+1,1}$ and to $v_{i+1,b}$ and from v_{ib} to $v_{i+1,1}$ and to $v_{i+1,b}$
- Finally, we add two nodes s and t
- We put edges from s to v_{11} and v_{1b}, from v_{n1} and v_{nb} to t, and from t to s

What does that look like?



P_1 (nodes for x_1, first Boolean variable)

P_2 (nodes for x_2, second Boolean variable)

P_n (nodes for x_n, last Boolean variable)

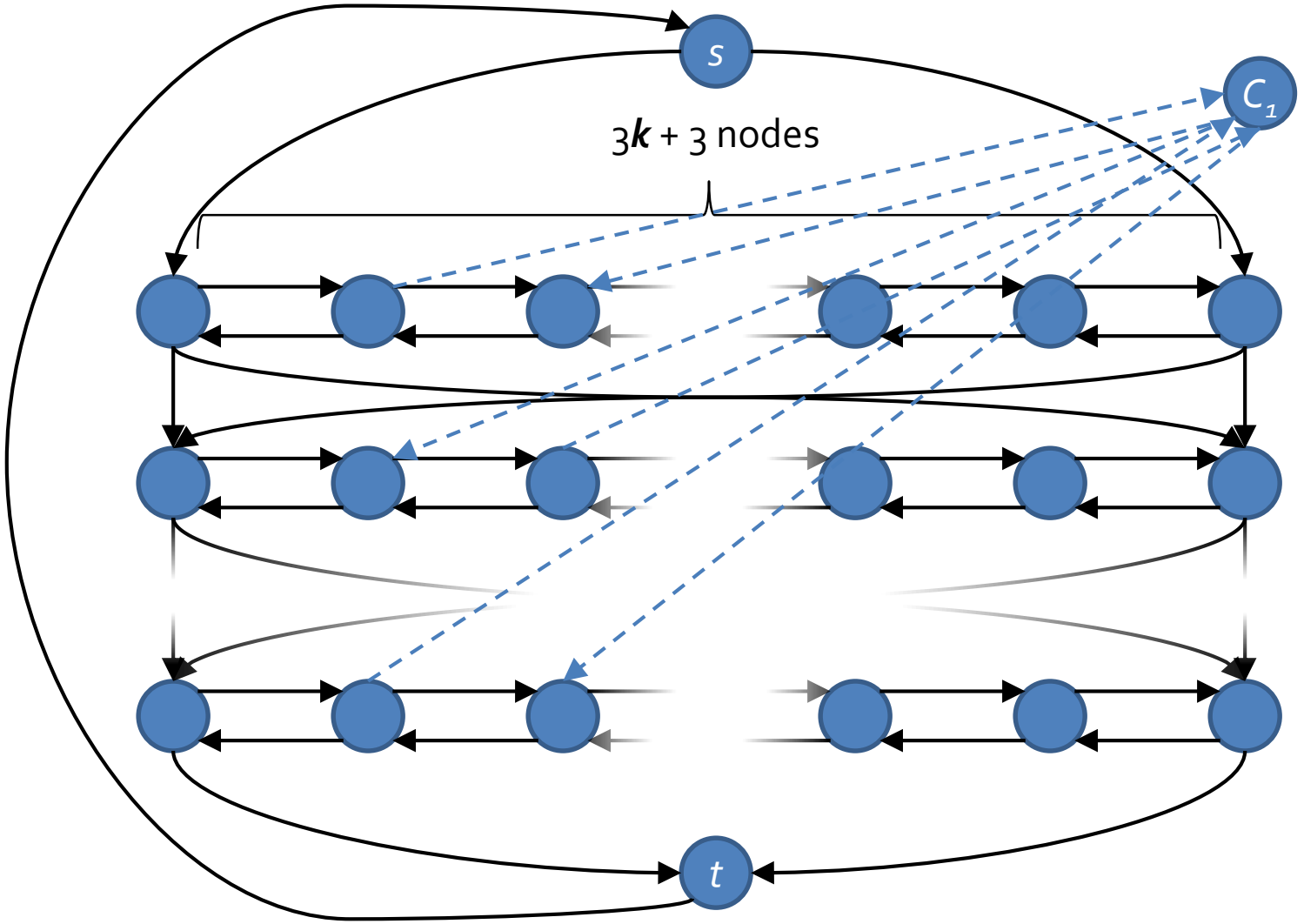
Why did we do all that?

- Only one edge leaves t, so a Hamiltonian cycle must use edge (t,s)
- From s, the cycle could travel through P_1 from the left to the right or from the right to the left
- After P_1, it could travel through P_1 from the left to the right or from the right to the left, and so on, a total of 2^n different cycles
- Each cycle maps to the n independent choices of true or false for variables $x_1, x_2, \ldots x_n$

How?!

- We want to make it so that traveling through P_i from left to right means that x_i is 1 and traveling through P_i from right to left means that x_i is 0
- Consider clause $C_1 = x_1 \vee \overline{x_2} \vee x_3$
- C_1 means that the cycle should traverse P_1 left to right or P_2 right to left or P_3 left to right
- To enforce this, we add a node c_1 that, for some value l, has edges from v_{1l}, v_{2l+1}, and v_{3l} and edges to $v_{1,l+1}$, v_{2l}, and $v_{3,l+1}$
- Then, c_1 can be spliced exactly once into any Hamiltonian path that visits the P_1, P_2, or P_3 paths in the correct directions
- In general, we make a node c_j for every clause and use node positions $3j$ and $3j + 1$ in each path P_i for variable x_i in clause C_j
  - Add edges $(v_{i,3j}, c_j)$ and $(c_j, v_{i,3j+1})$ if variable is not negated
  - Add edges $(v_{i,3j+1}, c_j)$ and $(c_j, v_{i,3j})$ if variable is negated

It gets worse!



C_1 (node for first clause)

- Connected to the three Boolean variable paths
- Left-to-right for variables that are not negated
- Right-to-left for variables that are

P_1 (nodes for x_1, first Boolean variable)

P_2 (nodes for x_2, second Boolean variable)

P_n (nodes for x_n, last Boolean variable)

Does that work?

- If there is a satisfying assignment for 3-SAT, there will be a Hamiltonian path:
  - If x_i is 1 in the satisfying assignment, we traverse path P_i left to right
  - Otherwise, we traverse P_i right to left
  - For each clause C_j, since it's satisfied, there will be at least one path P_i going the correct direction relative to c_j, and it will get spliced in there

Does that work? (continued)

- If there is a Hamiltonian cycle C in G, there will be a satisfying assignment
  - If C enters a node c_j on an edge from $v_{i,3j}$ it must depart on an edge to $v_{i,3j+1}$
  - Otherwise, if C enters a node c_j on an edge from $v_{i,3j+1}$ it must depart on an edge to $v_{i,3j}$
  - If cycle C visits P_i left to right (ignoring any c_j nodes), we set x_i to 1
  - Otherwise we set x_i to 0
  - All clauses will be satisfied

■

Traveling salesman is NP-complete

- **Proof:**
  - This one is easy. Obviously, it's in **NP**.
  - Now, we reduce Hamiltonian cycle to TSP.
  - Given a directed graph $G = (V,E)$ we define an instance of TSP:
    - Create a city v_i for every node v_i in the graph.
    - If there's an edge from v_i to v_j in the graph, then distance $d(v_i,v_j) = 1$
    - Otherwise, distance $d(v_i,v_j) = 2$

Proof continued

- ***G*** has a Hamiltonian cycle if and only if there is a tour of length at most ***n*** in the TSP instance:
  - If ***G*** has a Hamiltonian cycle, then following that sequence of cities will have ***n*** hops of length 1, allowing a TSP tour of length ***n***.
  - If TSP has a tour of at most ***n***, it's a sum of ***n*** terms which are at least 1, thus all terms are equal to 1. Each pair of connected cities must have had an edge in the original graph, and following that ordering must form a Hamiltonian cycle.

■

Hamiltonian path is NP-complete

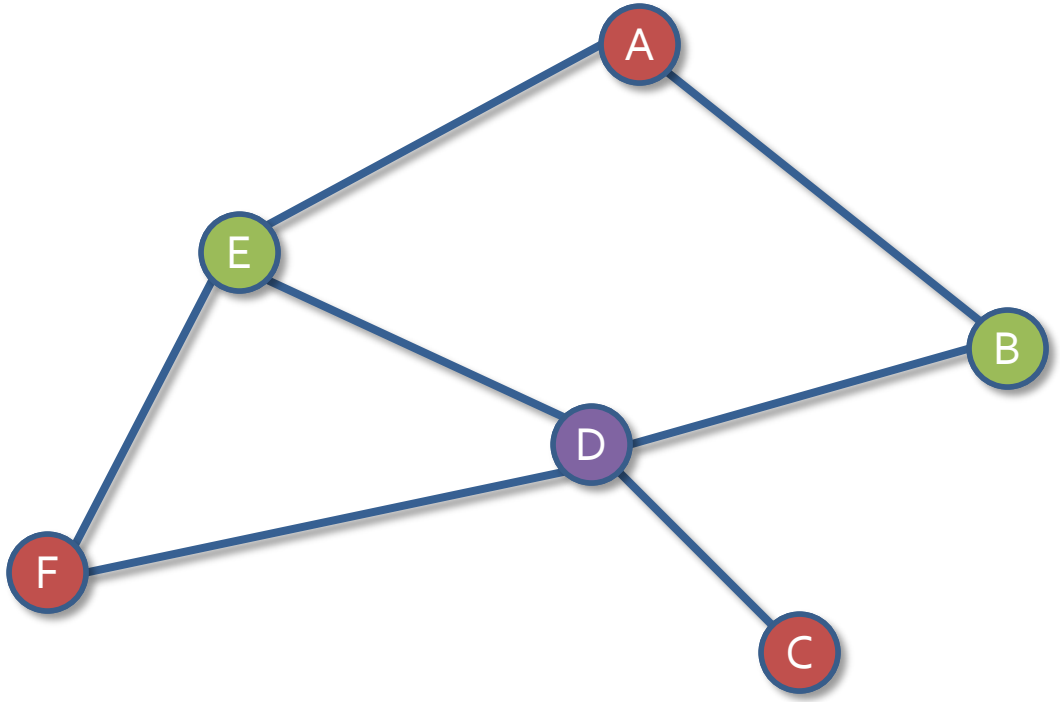
- Hamiltonian path is like Hamiltonian tour except that you don't have to return to the starting point
- We can do an easy reduction from Hamiltonian cycle to Hamiltonian path by splitting an arbitrary node ***v*** from ***V*** into ***v′*** and ***v″***
  - ***v′*** has all the outgoing edges of ***v***
  - ***v″*** has all the incoming edges of ***v***
  - Any tour that once went through ***v*** must now start at ***v′*** and end at ***v″***

Graph coloring

- Given a graph $G = (V,E)$ and a bound k, is there a way to color each node such that no two adjacent nodes have the same color, using no more than k colors?
- Applications:
  - Register allocation
  - Assigning campers to tents
  - Scheduling jobs that need the same resources
  - Assigning wavelengths to communication devices

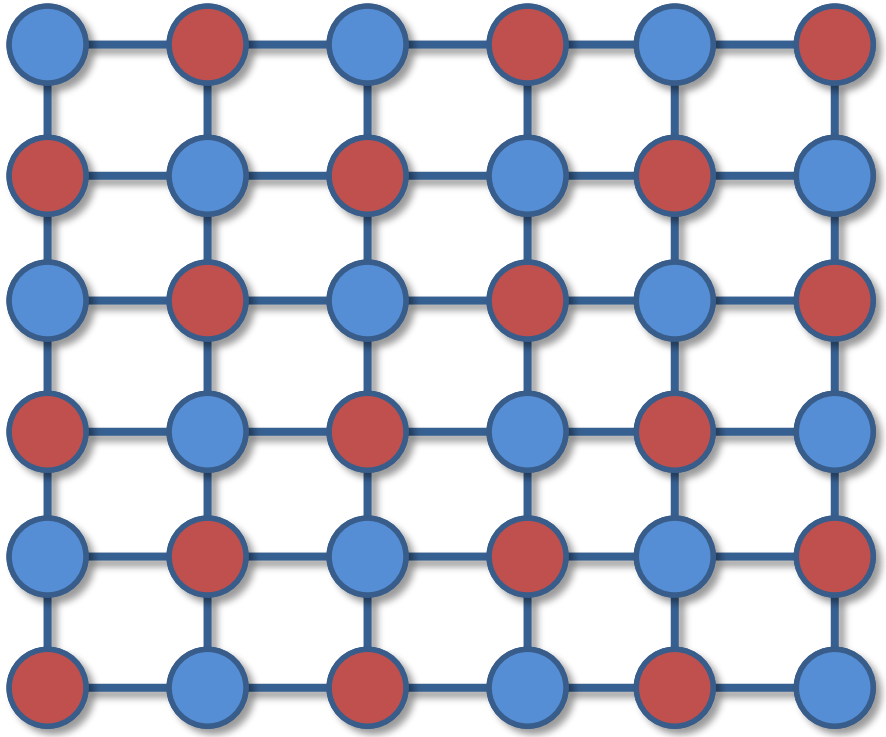
Graph coloring

- Find the smallest number of colors for coloring nodes such that no two adjacent nodes have the same color



Graph coloring

- What about this graph?



3-coloring a graph

- 2-coloring is easy: it's the same problem as whether or not a graph is bipartite
- It turns out that seeing if a graph can be colored with only 3 colors is, in fact, **NP-complete**
- It's interesting (but difficult to prove) that any map where a country is contiguous can be colored with 4 colors

3-coloring is NP-complete

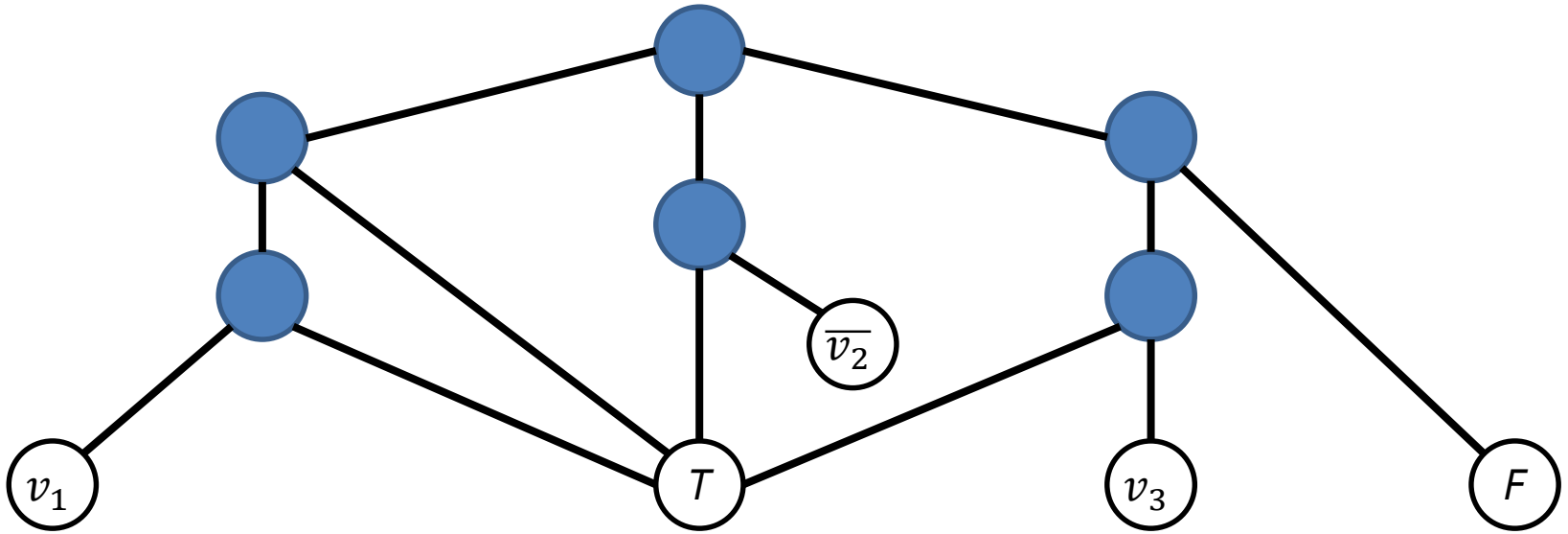
- **Proof:**
  - 3-color is in NP.
  - We can reduce 3-SAT to 3-coloring.
  - Recall that 3-SAT has variables $\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}$ and clauses $\boldsymbol{C_1}, \boldsymbol{C_2}, \ldots, \boldsymbol{C_k}$.
  - Create graph nodes v_i and $\overline{v_i}$ for variable x_i and its negation $\overline{x_i}$.
  - We create three special nodes ***T***, ***F***, and ***B*** for true, false, and base.

Proof continued

- Join every pair of nodes v_i and $\overline{v_i}$ with an edge and join both to ***B***, forming triangles.
- Join ***T***, ***F***, and ***B*** with edges, forming another triangle.
- Note that in any 3-coloring of ***G***, nodes v_i and $\overline{v_i}$ must get different colors and must be different from ***B***.
- Also, ***T***, ***F***, and ***B*** must get all three colors in some permutation.

Proof continued

- Consider a clause like $x_1 \vee \overline{x_2} \vee x_3$
- We want this to mean, at least one of v_1, $\overline{v_2}$, and v_3 gets the same color as ***T***
- For every clause, we attach a special six node subgraph that forces the top node to be none of the three colors unless one of the variables is true



Proof continued

- A 3-SAT instance is satisfiable if and only if ***G*** has a 3-coloring.
  - Suppose that the 3-SAT instance is satisfiable. Color ***T***, ***F***, and ***B*** arbitrarily with the three colors. For every ***i***, color v_i the ***T*** color if x_i is true and the ***F*** color if $\overline{x_i}$ is true. Color $\overline{v_i}$ the only available color. Since at least one term in each clause is true, we can color the six-node clause graph.
  - Suppose that the graph has a 3-coloring. Each node v_i is assigned either the ***T*** color or the ***F*** color. Set the x_i variable accordingly. It must be the case that at least one term in each clause has the value 1. Otherwise, all of the corresponding nodes in the clause subgraph will have the ***F*** color, which precludes a 3-coloring.

■

k-coloring is NP-complete

- It's an easy reduction from 3-coloring to ***k***-coloring.
- Just take a graph and add ***k*** – 3 nodes.
- Connect them to **all** other nodes (including each other).
- The graph will be ***k***-colorable if and only if the original graph was 3-colorable.

Subset sum is NP-complete

- Given natural numbers $w_1, w_2, \ldots, w_n$ and a target W, can you find a subset of your numbers that adds up to exactly W?
- We're not going to do the reduction, but we could.

Scheduling with release times and deadlines

- Consider ***n*** jobs that we want to run on a machine.
- Each job ***i*** has:
  - A release time r_i which is the earliest it could start
  - A deadline d_i which is when it must be finished by
  - A time t_i which is the total amount of time it takes to do
- Can we schedule all jobs so that they start at or after their release times and end at or before their deadlines?

Scheduling with release times and deadlines is NP-complete

- We can do a reduction from subset sum to scheduling with release times and deadlines.
- Let ***S*** be the sum of **all** the weights.
- For jobs 1, 2,…***n***, we let them all have a release time of 0, a deadline of ***S*** + 1, and a duration of w_i.
  - All jobs can finish, scheduled in any order!
- We add one more job with a release time of ***W***, a deadline of ***W*** + 1, and a duration of 1.
- To make everything fit, we have to schedule a subset of the jobs before ***W***, filling it exactly, and exactly filling ***S*** – ***W***, with the extra job running from ***W*** to ***W*** + 1.

Co-NP

Asymmetric certification

- Efficient certification is **asymmetric**
- A problem is in NP if and only if there is a certificate that can be checked in polynomial time for a "yes" answer
- If the answer is "no," there's no requirement for a certificate
- How would you give a short certificate that there's **no** satisfying assignment for 3-SAT?

Co-NP

- But there is an alternative definition
- Co-NP is the set of all problems for which there is a polynomial-size certificate if the answer is "no"
- Both Co-NP and NP problems can certainly be solved with exponential work
- Is Co-NP = NP?
- No one knows!

What if NP ≠ Co-NP?

- Then we would know that **P ≠ NP**
- **Proof:**
  - Consider the contrapositive: (**P** = **NP**) → (**NP** = **Co-NP**).
  - **P** is closed under complementation. That means that the negated problem (all the strings that would give a "no") is also in **P**.
  - $X \in \mathbf{NP} \rightarrow X \in \mathbf{P} \rightarrow \bar{X} \in \mathbf{P} \rightarrow \bar{X} \in \mathbf{NP} \rightarrow X \in \mathbf{Co\text{—}NP}$
  - $X \in \mathbf{co\text{—}NP} \rightarrow \bar{X} \in \mathbf{NP} \rightarrow \bar{X} \in \mathbf{P} \rightarrow X \in \mathbf{P} \rightarrow X \in \mathbf{NP}$
  - Thus, **NP** = **co-NP**.
  - But, because the contrapositive is logically equivalent, that means that (**NP ≠ Co-NP**) → (**P ≠ NP**).

■

Is P = (NP ∩ Co-NP)?

- It's true that **P** ⊆ (**NP** ∩ **Co-NP**)
- But opinions are mixed as to whether there might be problems in **NP** ∩ **Co-NP** that cannot be solved in polynomial time
- Maybe you can find the answer!

Upcoming

Next time…

- A little bit of theory of computing
- Approximation algorithms
  - Load balancing
  - Center selection
- Read sections 11.1 and 11.2

Reminders

- Assignment 6 due Friday!