Week 10 - Wednesday

COMP 4500

Last time

- What did we talk about last time?
- Maximum flow
- Minimum cuts

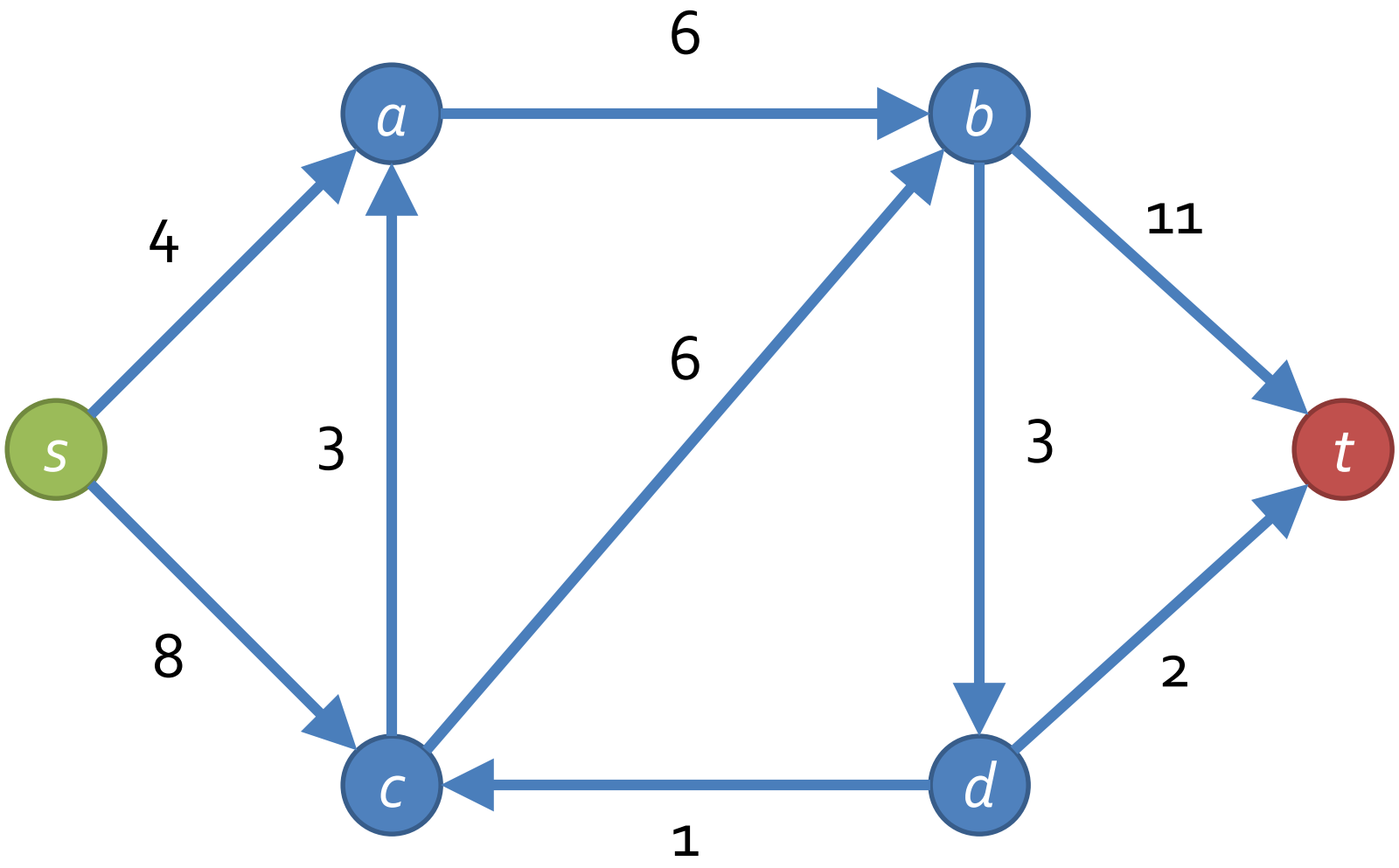
Questions?

Assignment 5

Logical warmup

- As I was going to St. Ives
- I crossed the path of seven wives
- Every wife had seven sacks
- Every sack had seven cats
- Every cat had seven kittens
- Kittens, cats, sacks, wives
- How many were going to St. Ives?

Maximum flow practice



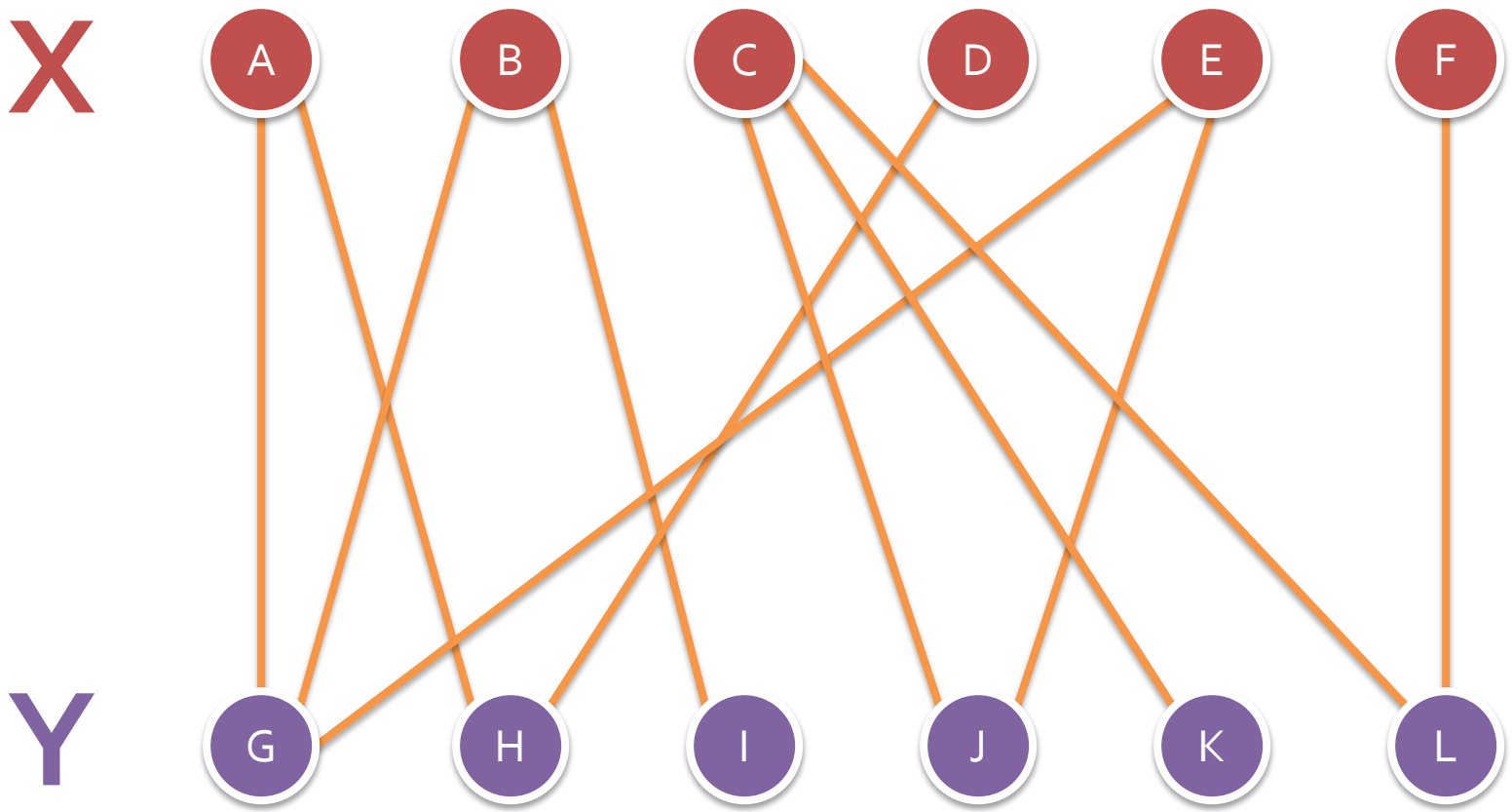
Three-sentence Summary of Bipartite Matching

Bipartite Matching

Bipartite graphs

- Recall that a bipartite graph is one whose nodes can be divided into two disjoint sets X and Y
- Every edge has one end in set X and the other in set Y
  - There are no edges from a node inside set X to another node in set X
  - There are no edges from a node inside set Y to another in set Y
- Equivalently, a graph is bipartite if and only if it contains no odd cycles

Bipartite graph



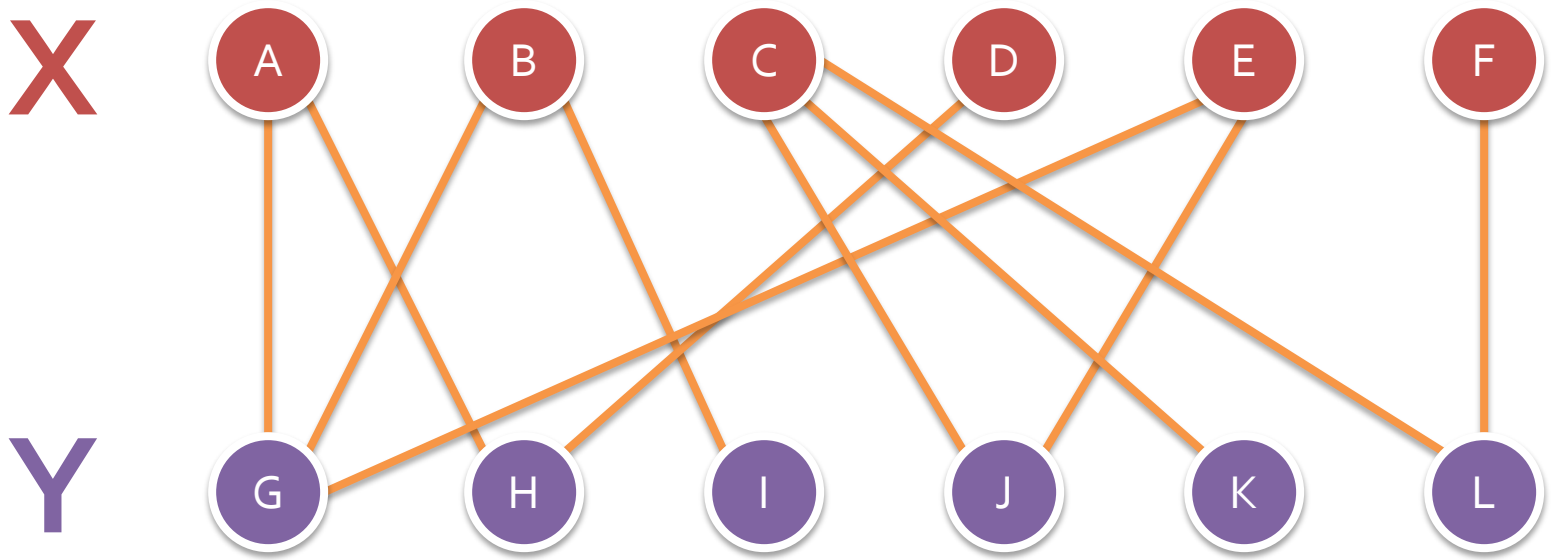
Maximum matching

- **Matching** means pairing up nodes in set X with nodes in set Y
- A node can only be in one pair
- A **perfect matching** is when every node in set X and every node in set Y is matched
- It is not always possible to have a perfect matching
- We can still try to find a **maximum matching** in which as many nodes are matched up as possible

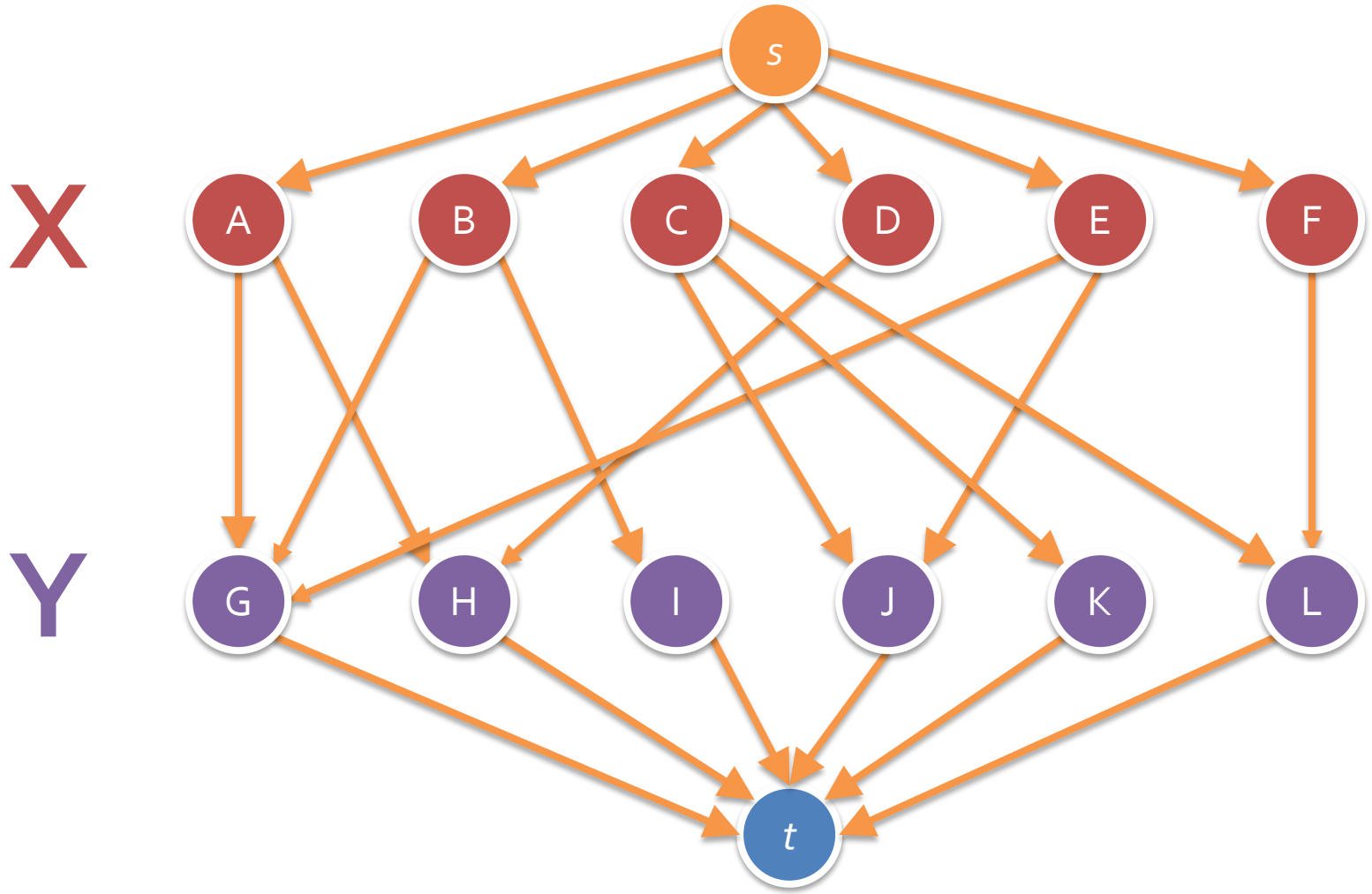
Can we use our maximum flow algorithm?

- The goal of this class is to expose you to many algorithms
- Hopefully, an algorithm for one problem can be used for another problem, by adding a tweak
- It turns out that we can think of the bipartite matching problem as a version of the maximum flow problem
- We just need to update the graph a little

Bipartite matching problem



Maximum flow problem



An easy change

- Take a bipartite graph *G* and turn it into a directed graph *G'*
- Create a source node ***s*** and a sink node ***t***
- Connect directed edges from the source to all the nodes in set X
- Connect directed edges from all the nodes in set Y to the sink
- Change all the undirected edges from X to Y to directed edges from X to Y
- Set the capacities of all edges to 1

Algorithmic changes

- We run the Ford-Fulkerson algorithm to find the maximum flow on our new graph
- Since all edges from X to Y have capacity 1, they will either have a flow of 1 or of 0
- If they have a flow of 1, they are in the matching
- If they have a flow of 0, they aren't
- The maximum flow value tells us how many nodes are matched

Why does it work?

- Every node in X only has a single incoming edge from *s*
- Since it has a maximum of an incoming flow of 1, it has a maximum outgoing flow of 1 as well
- Each node in X can thus only be matched with one node in Y

Minor variation

- If you don't want to, you don't have to make the flow network
- You can apply the same idea directly to the bipartite graph
- To be parallel, an augmenting path will start in X and end in Y
- It will always start at an unmatched node in X and end at an unmatched node in Y
- Crossing an unmatched edge (one with 0 flow) will change it to a matched edge (one with 1 flow)
- Crossing a matched edge (one with 1 flow) is crossing it backwards, changing it to an unmatched edge (one with 0 flow)

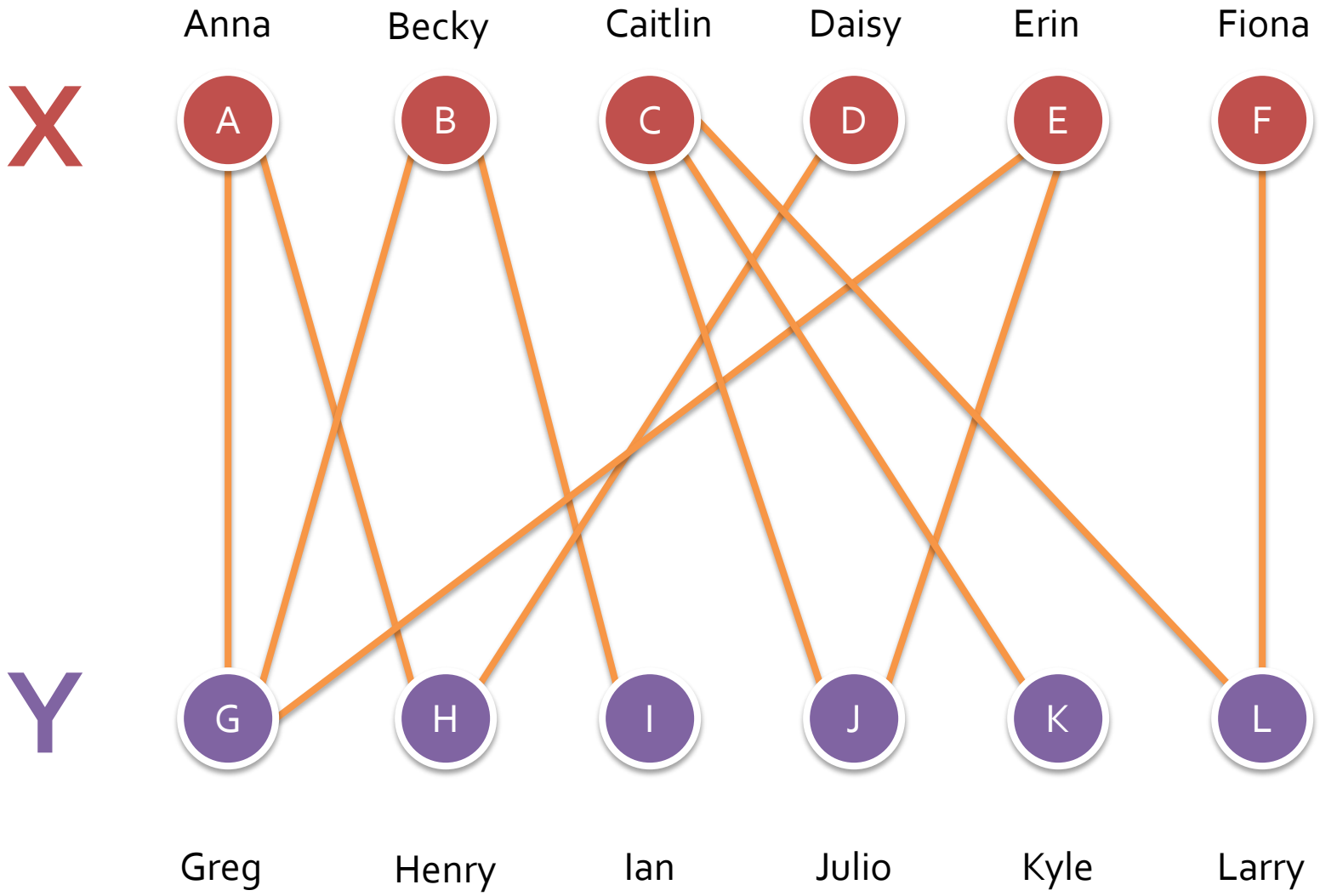
Maximal matching

- To make the algorithm go faster, we can start with a **maximal matching**
- A maximal matching is not necessarily maximum, but you can't add edges to it directly without removing other edges
- In essence, arbitrarily match unmatched nodes until you can't anymore
- Then start the process of looking for augmenting paths

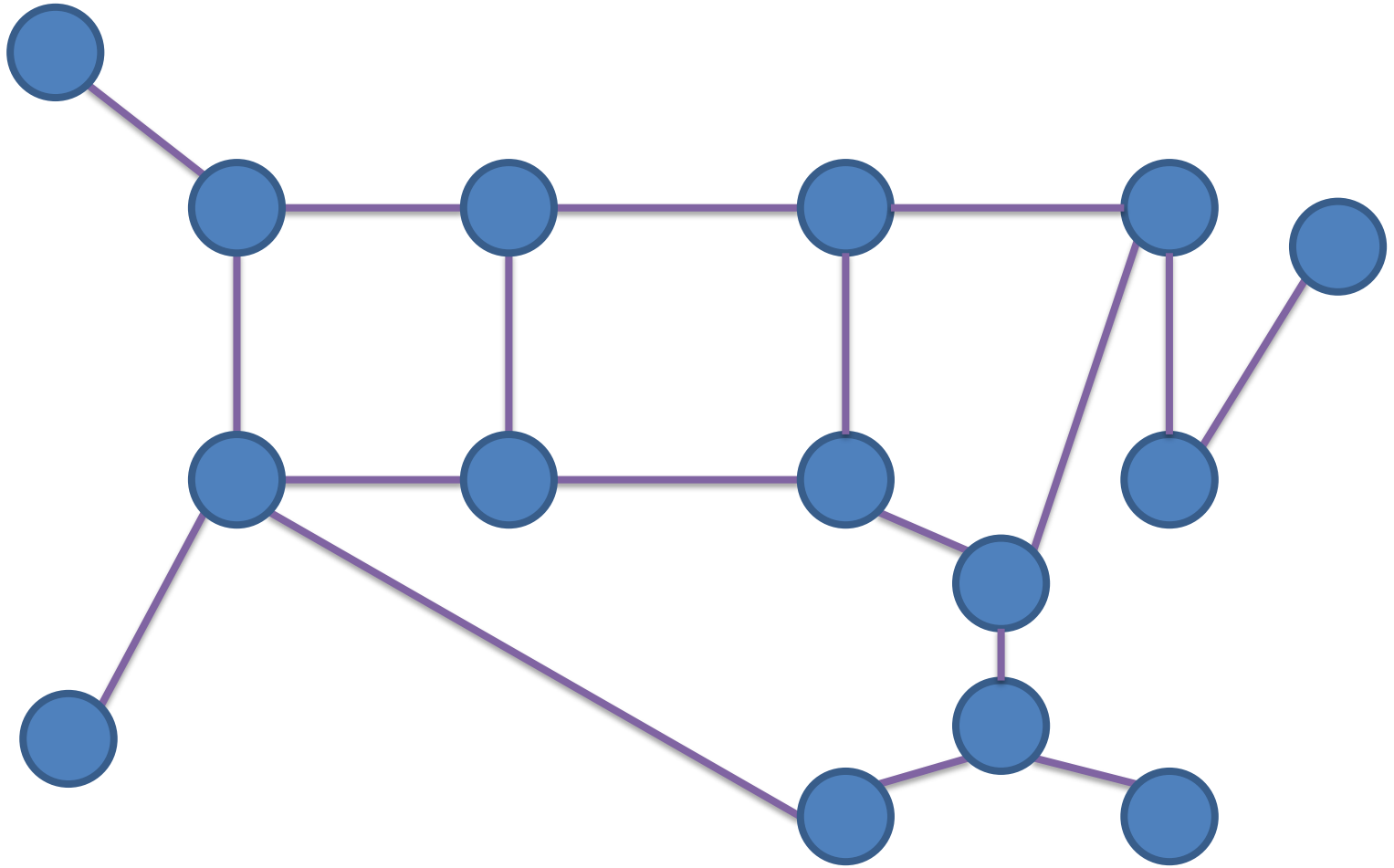
Matching algorithm

1. Come up with a legal, maximal matching
2. Take an **augmenting path** that starts at an unmatched node in X and ends at an unmatched node in Y
3. If there is such a path, switch all the edges along the path from being in the matching to being out and vice versa
4. If there is another augmenting path, go back to Step 2

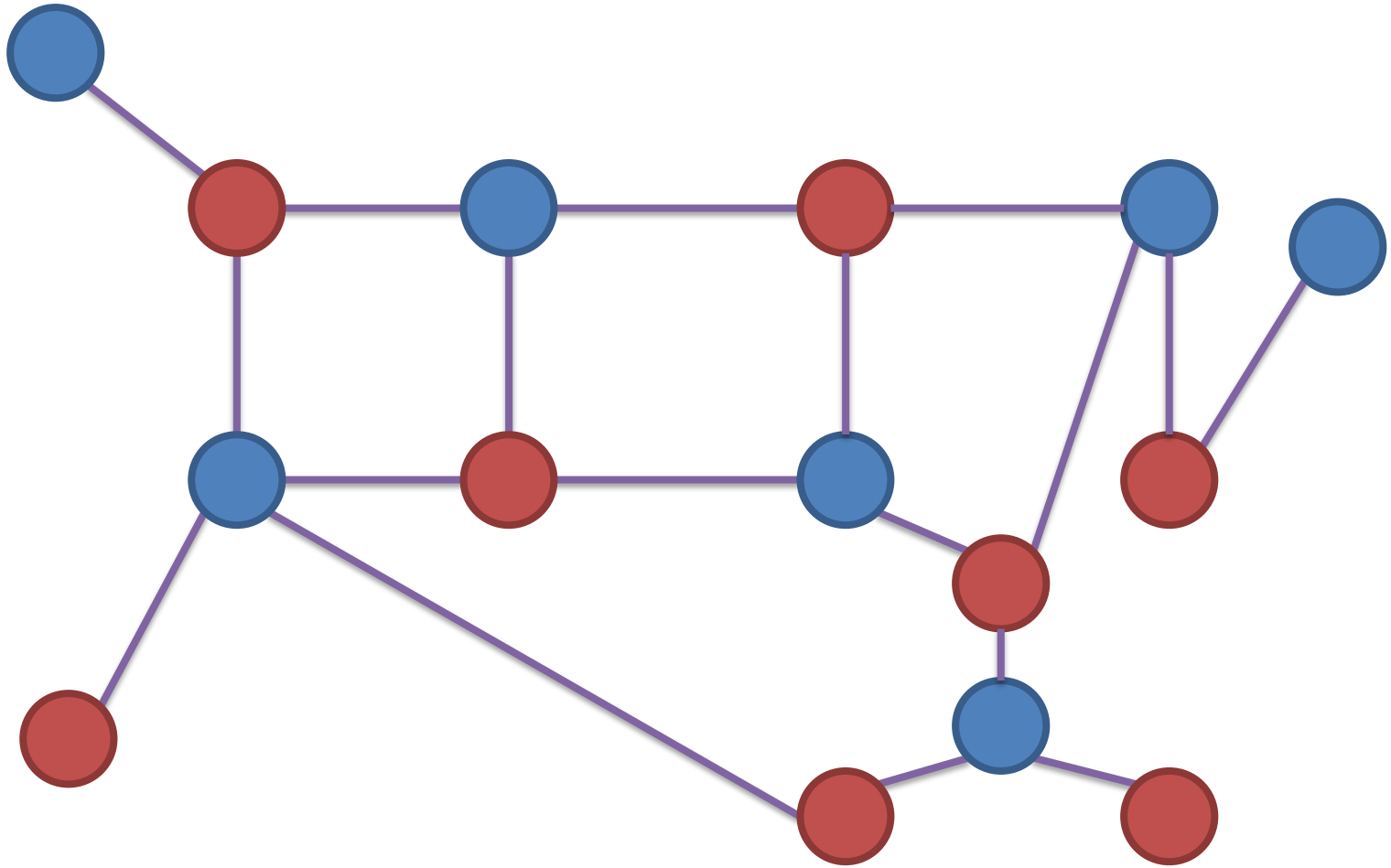
Match the graph

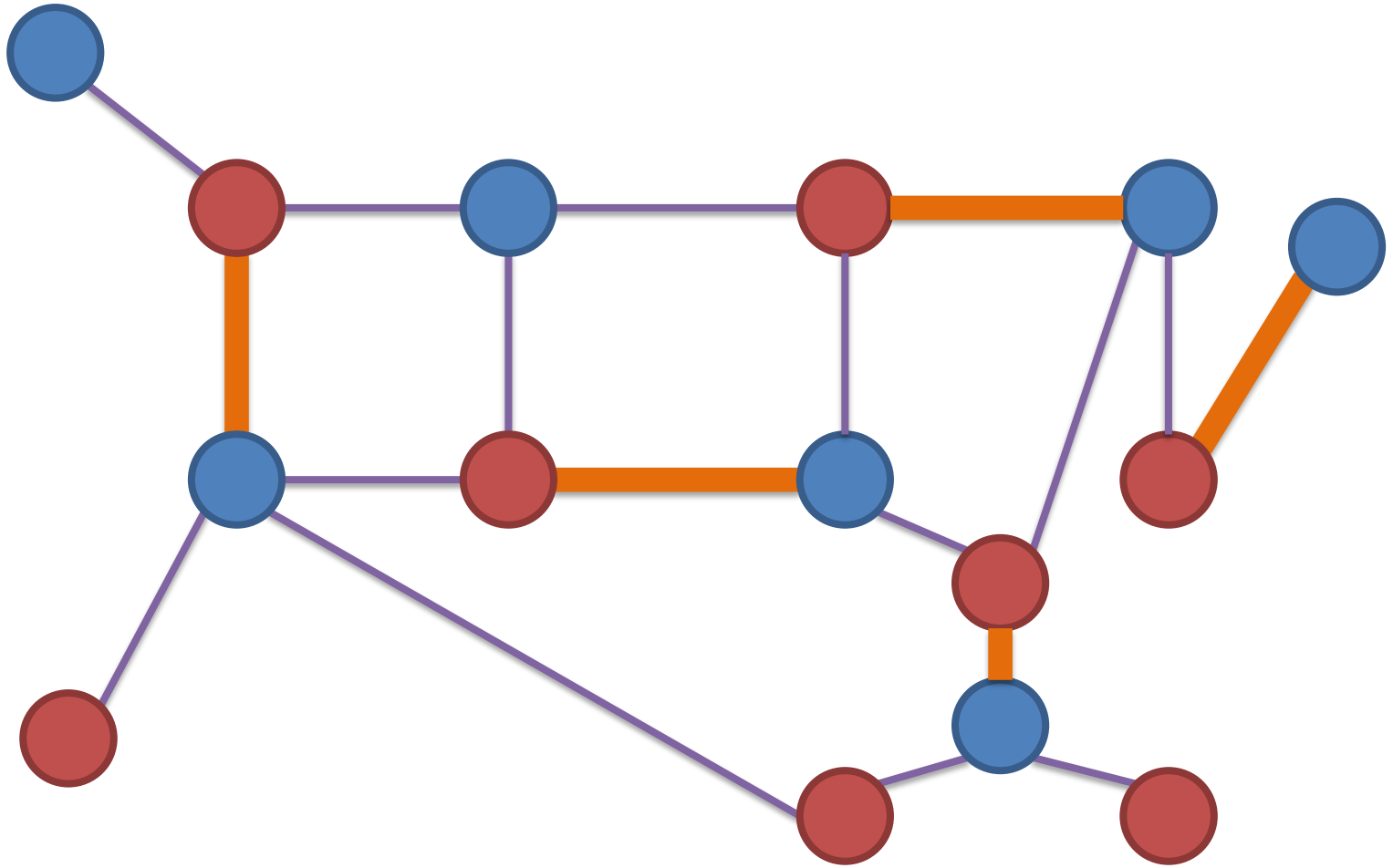


Is this graph bipartite?

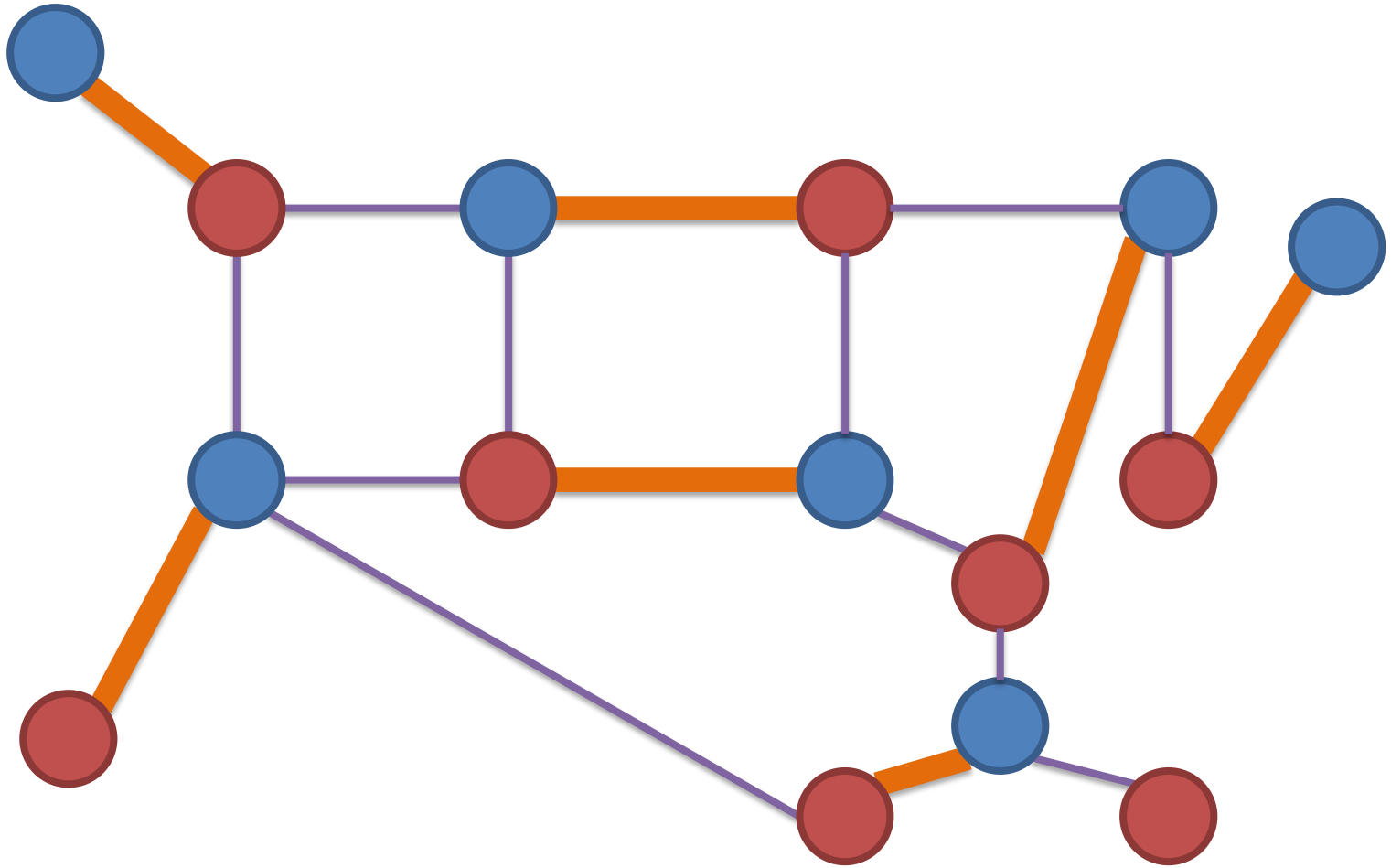


Let's make a maximal matching





Maximum matching! (one of them)



Quiz

Upcoming

Next time…

- NP-completeness
- Polynomial-time reductions

Reminders

- Finish Assignment 5
  - **Due Friday by midnight!**
- Read section 8.1