Week 10 - Monday

COMP 2100

Last time

- What did we talk about last time?
- Cycle detection
- Topological sort
- Connectivity
- Minimum spanning tree

Questions?

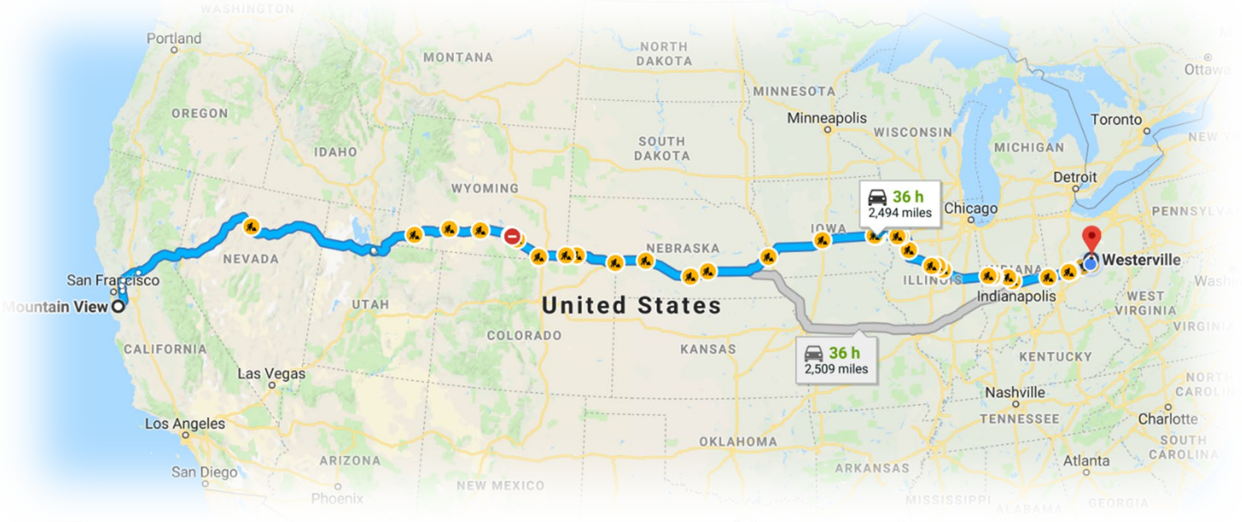
Project 3

Assignment 5

Shortest Paths

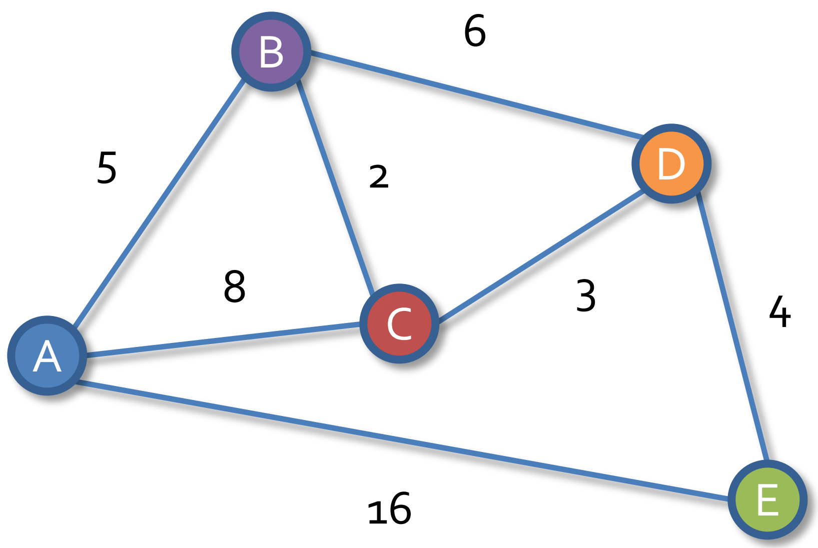
Google Maps

- How does Google Maps find the shortest route from Silicon Valley to Westerville?
- Graph theory, of course!
- It stores a very large graph where locations are nodes and streets (well, parts of streets) are edges



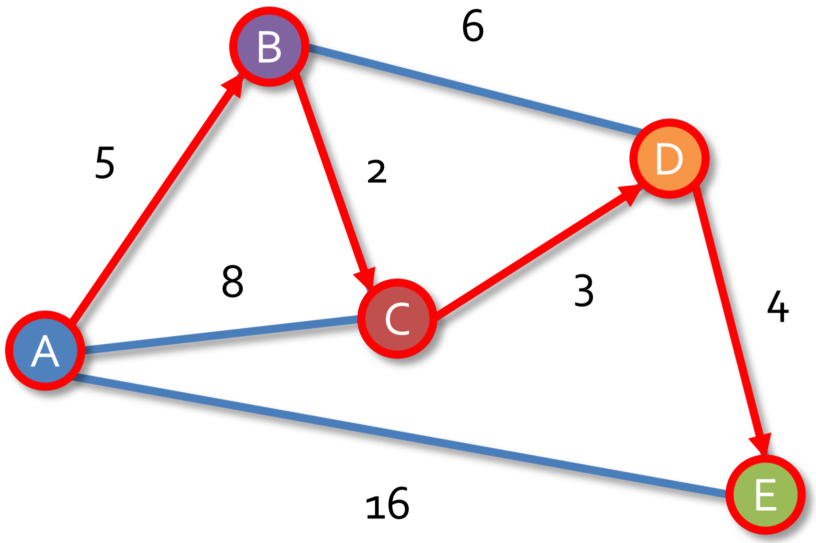
Shortest paths

- We use a weighted graph
- Weight can represent time, distance, cost: anything, really
- The shortest path (lowest total weight) is not always obvious



What's the shortest path?

- Take a moment and try to find the shortest path from A to E.
- The shortest path has cost 14



How can we always find the shortest path?

- On a graph of that size, it isn't hard to find the shortest path
- A Google Maps graph has millions and millions of nodes
- How can we come up with an algorithm that will always find the shortest path from one node to another?

Dijkstra's Algorithm

- In 1959, Edsger Dijkstra published an algorithm to find shortest paths

Notation	Meaning
s	Starting node
***d*(*v*)**	The best distance from ***s*** to ***v*** found so far
***d*(*u*, *v*)**	The direct distance between nodes ***u*** and ***v***
S	A set which contains the nodes for which we know the shortest path from ***s***
V	A set which contains the nodes for which we do not yet know the shortest path from ***s***
pred(u)	Predecessor of ***u*** in the shortest path from ***s***

Dijkstra's Algorithm

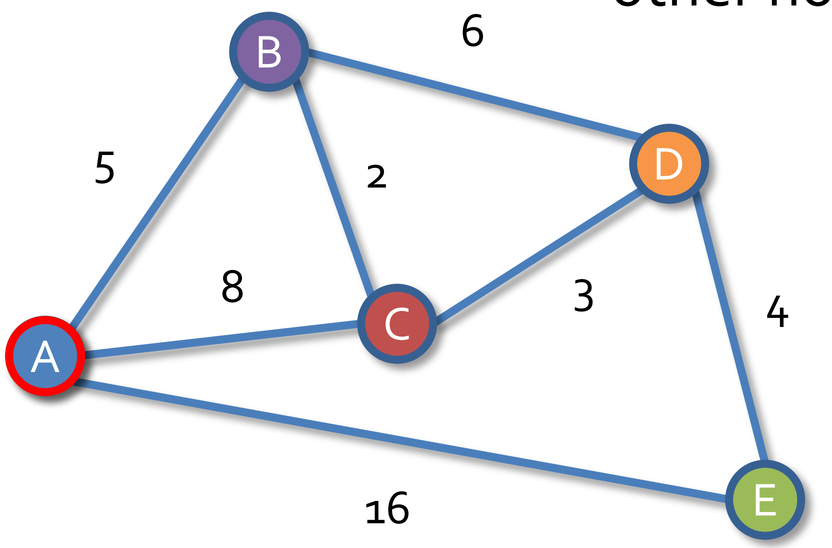
1. Start with two sets, ***S*** and ***V***:
   - ***S*** is empty
   - ***V*** has all the nodes in it
2. Set the distance to all nodes in ***V*** to ∞
3. Set the distance to the starting node ***s*** to 0
4. Find the node ***u*** in ***V*** that is closest to ***s***
5. For every neighbor ***v*** of ***u*** in ***V***
   - If ***d***(***v***) > ***d***(***u***) + ***d***(***u***,***v***)
   - Set ***d***(***v***) = ***d***(***u***) + ***d***(***u***,***v***)
   - Set ***pred***(***v***) = ***u***
6. Move ***u*** from ***V*** to ***S***
7. If ***V*** is not empty, go back to Step 4

Example for Dijkstra

Node	*d(u)*	*pred(u)*
A	0	-
B	∞	
C	∞	
D	∞	
E	∞	

Sets	
S	**V**
	A, B, C, D, E

Finding the shortest distance from **A** to all other nodes



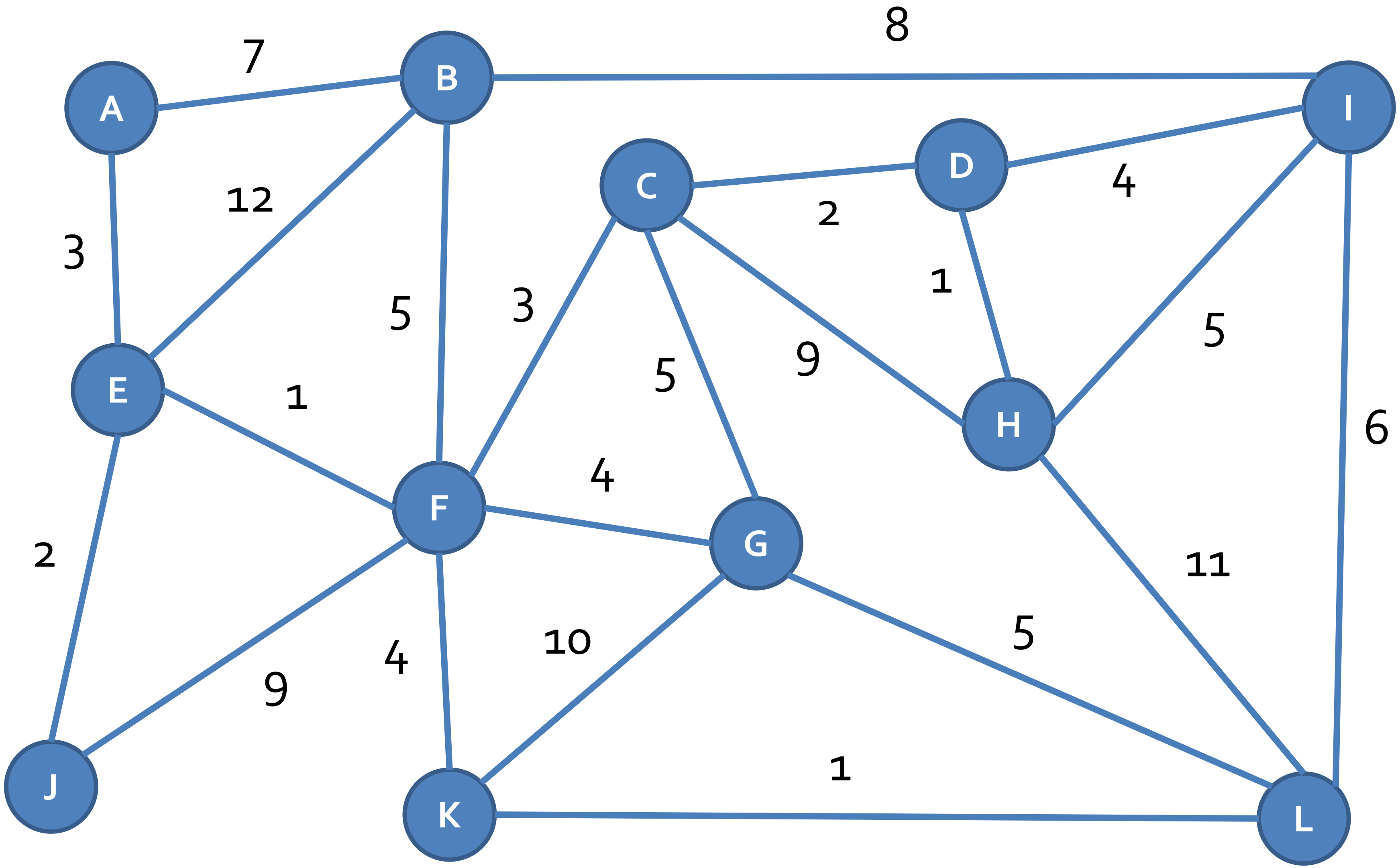
Features of Dijkstra

- Always gets the next closest node, so we know there isn’t a better way to get there
- Finds the shortest path from a starting node to **all** other nodes
- Works even for directed graphs
  - Provided that they don't have negative edge weights

Dijkstra's running time

- The normal running time for Dijkstra's is $O(|V|^2)$
  - At worst, we may have to update each node in ***V*** for each node ***v*** that we find the shortest path to
- A special data structure called a min-priority queue can implement the process of updating priorities faster
  - Total running time of $O(|E| + |V| \log |V|)$
  - Technically faster for sparse graphs
  - Algorithm wizards Fredman and Tarjan created an implementation called a Fibonacci Heap
    - Actually slow in practice

Dijkstra's practice

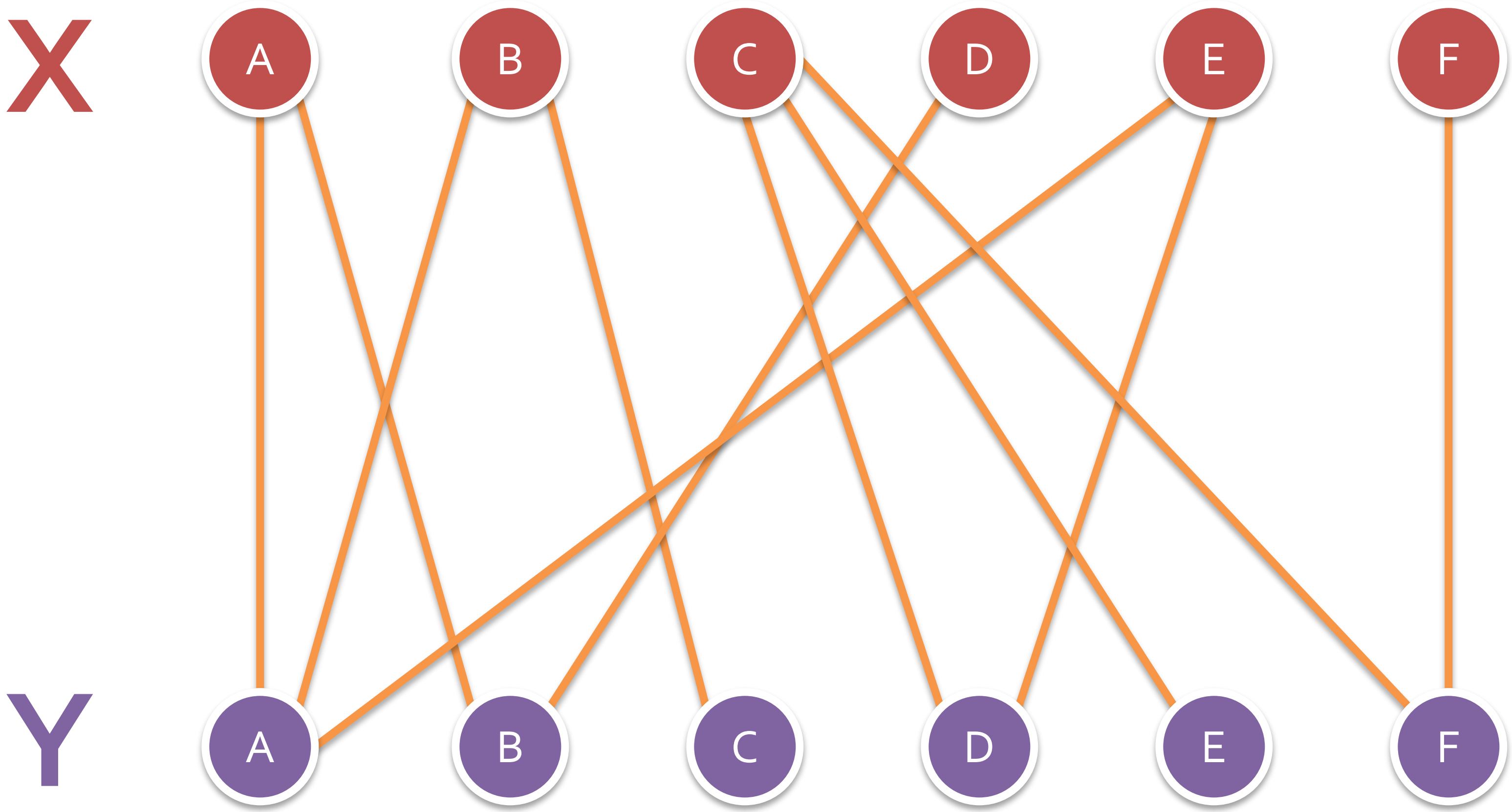


Matching

Bipartite graphs

- A bipartite graph is one whose nodes can be divided into two disjoint sets X and Y
- There can be edges between set X and set Y
- There are no edges inside set X or set Y
- A graph is bipartite if and only if it contains no odd cycles

Bipartite graph



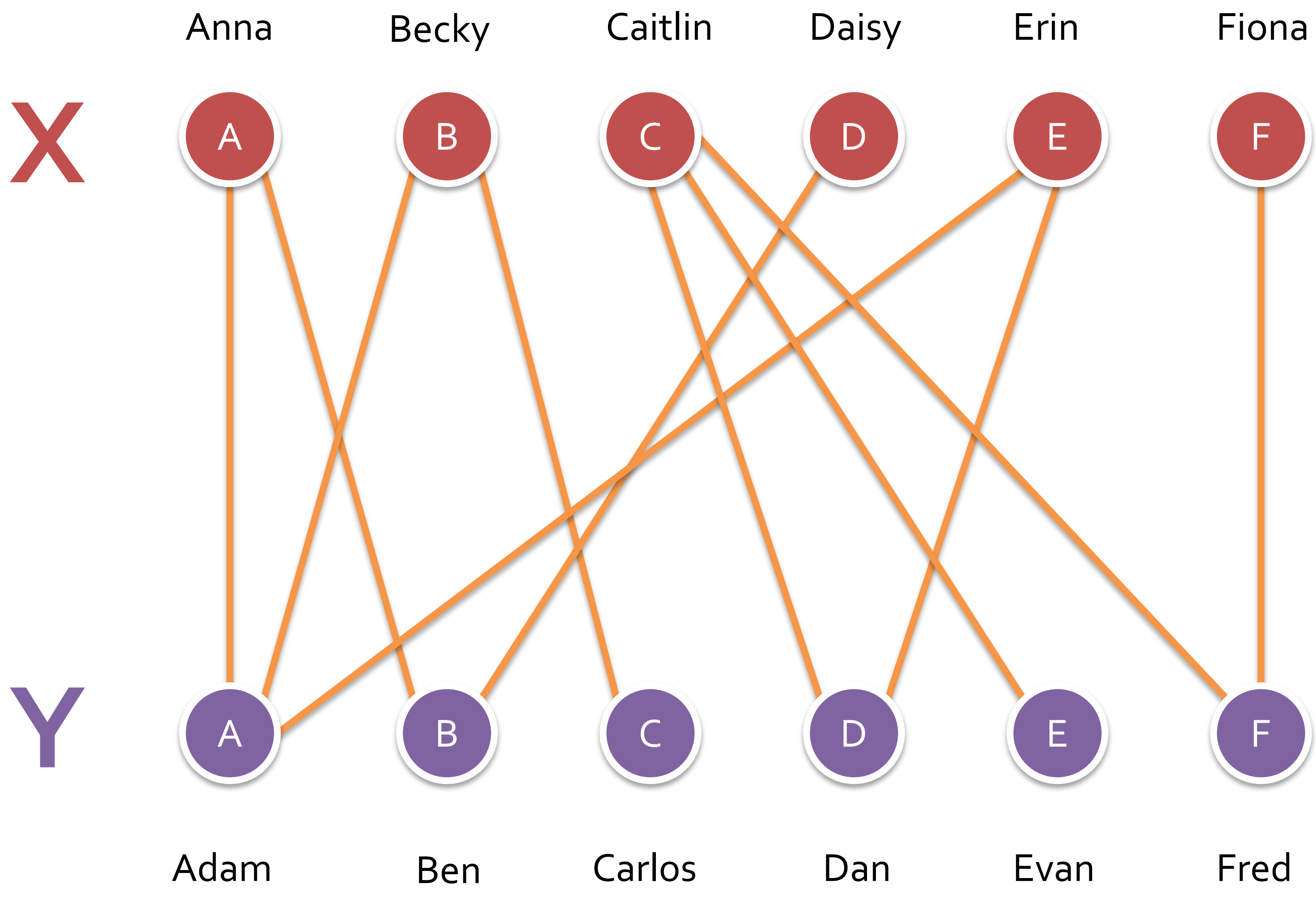
Maximum matching

- 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

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, alternating the kind of edges it cross (first unmatched, then matched, then unmatched, etc.)
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's another augmenting path, go back to Step 2

Match the graph



Upcoming

Next time…

- Finish matching
- Stable marriage
- Euler paths and tours

Reminders

- Keep working on Project 3
- Start Assignment 5
- Read sections 6.2 and 6.4