



CSCI 2200  
Foundations of Computer Science

Lecture 12:  
Graphs II – Electric  
Boogaloo



CSCI 2200  
Foundations of Computer Science

# Lecture 12: Graphs II – Electric Boogaloo

“Spring (Wild Horseradish Jam)”  
from *Stardew Valley* by Eric Barone

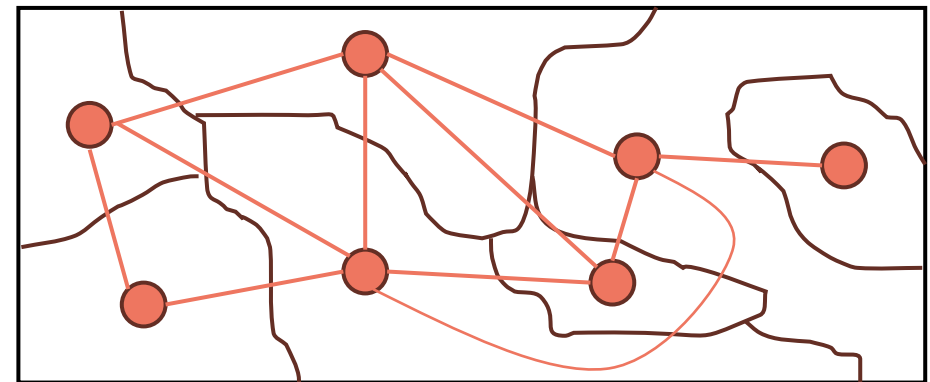


# Today's tasks

- Exam 2 next week on Wed. 2/26
  - Same format / ground rules as Exam 1
- HW questions?
- Common graph problems
  - Coloring
  - Matching
  - Other interesting problems

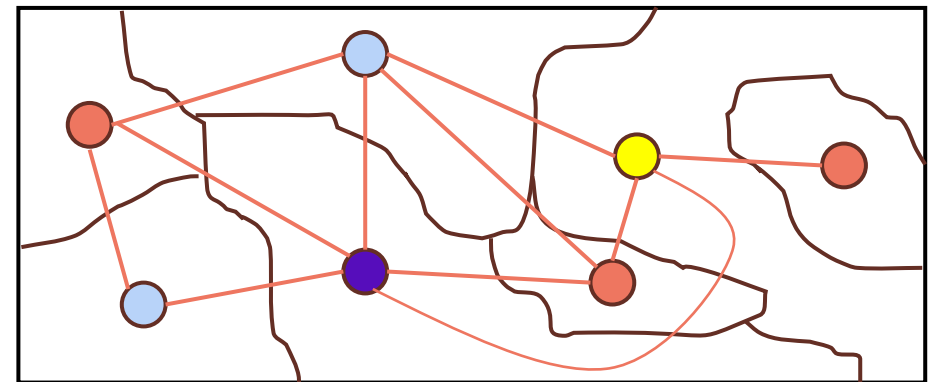
# Graph coloring

- We talked about planar graphs last class: ones that we can draw on paper without crossing lines.
  - Recall: Must not have  $K_5$  and  $K_{3,3}$  as a "minor".
  - Also: Any map of regions can be represented as a "border graph" in which nodes are connected iff regions share a border.
- Common task: Color the regions so that no regions of the same color are touching.
  - How many colors are needed here?



# Graph coloring

- We talked about planar graphs last class: ones that we can draw on paper without crossing lines.
  - Recall: Must not have  $K_5$  and  $K_{3,3}$  as a "minor".
  - Also: Any map of regions can be represented as a "border graph" in which nodes are connected iff regions share a border.
- Common task: Color the regions so that no regions of the same color are touching.
  - How many colors are needed here?



# How many colors are needed in general?





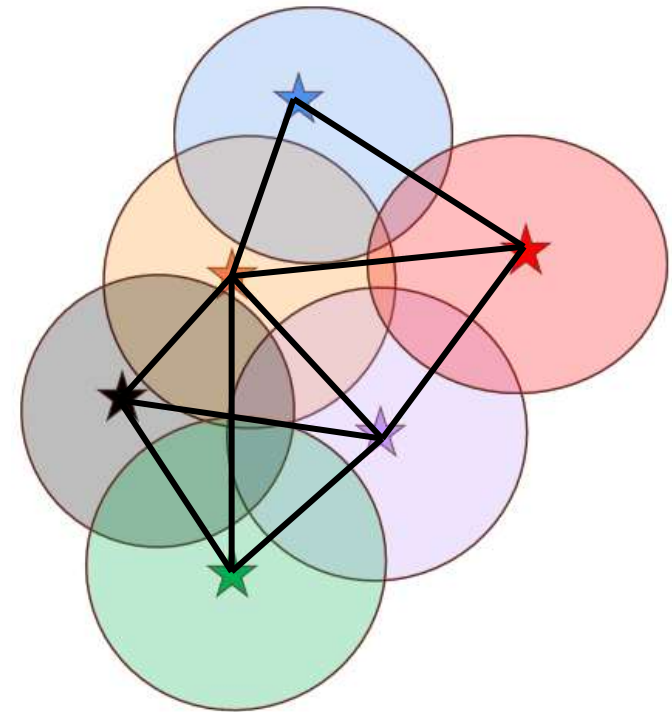
# The Four-Color Theorem

- The chromatic number of a graph is the minimum number of colors required, when assigning colors to vertices, to ensure that no two connected vertices are the same color.
- For any planar graph, the chromatic number is  $\leq 4$ .
  - Note this isn't \*quite\* the same thing as countries on a map, because countries often have exclaves (non-contiguous areas, like Alaska or Kaliningrad).
  - This result was first conjectured in the mid-1800s, but only proven in 1976. It was the first major theorem proven with substantial computer existence.



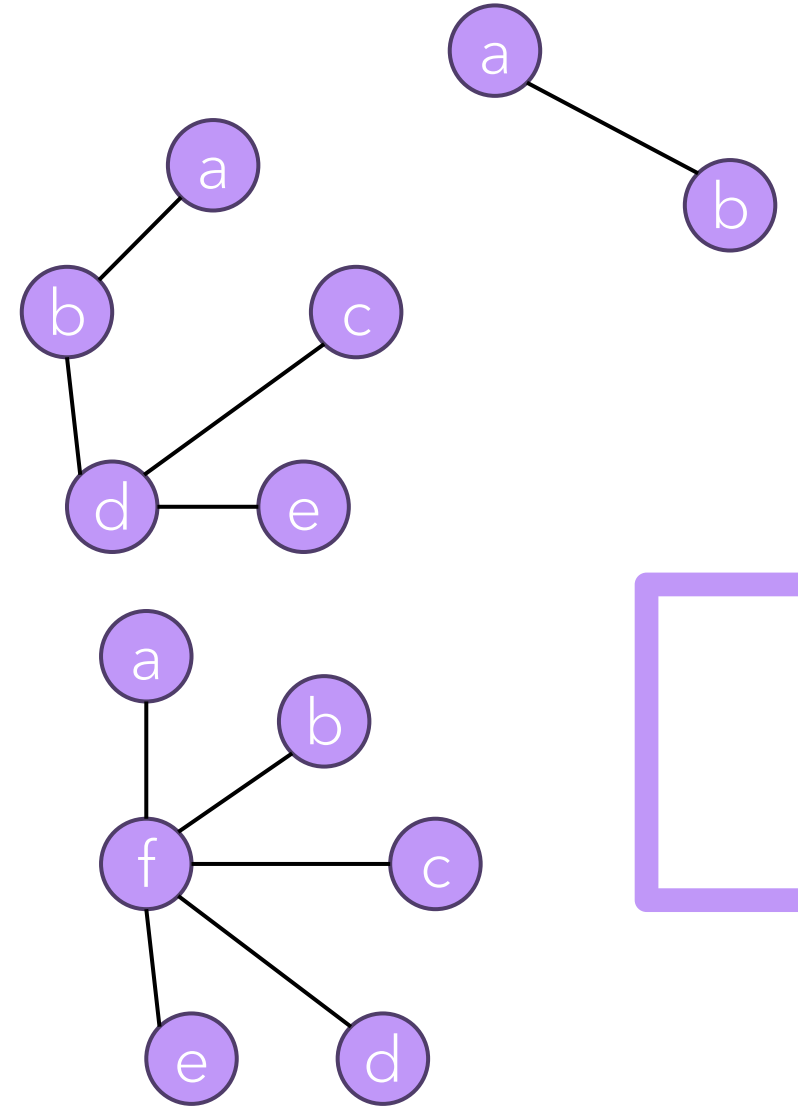
# Coloring & conflict graphs

- Remember this problem?
  - Six cell towers have overlapping coverage areas, as shown in the diagram at right. Towers whose areas overlap must use different frequency bands to communicate with phones in their area. What is the minimum number of frequency bands needed for this set of cell towers?
- This is a graph coloring problem!
- Draw the overlap graph – connected nodes cannot share a frequency.



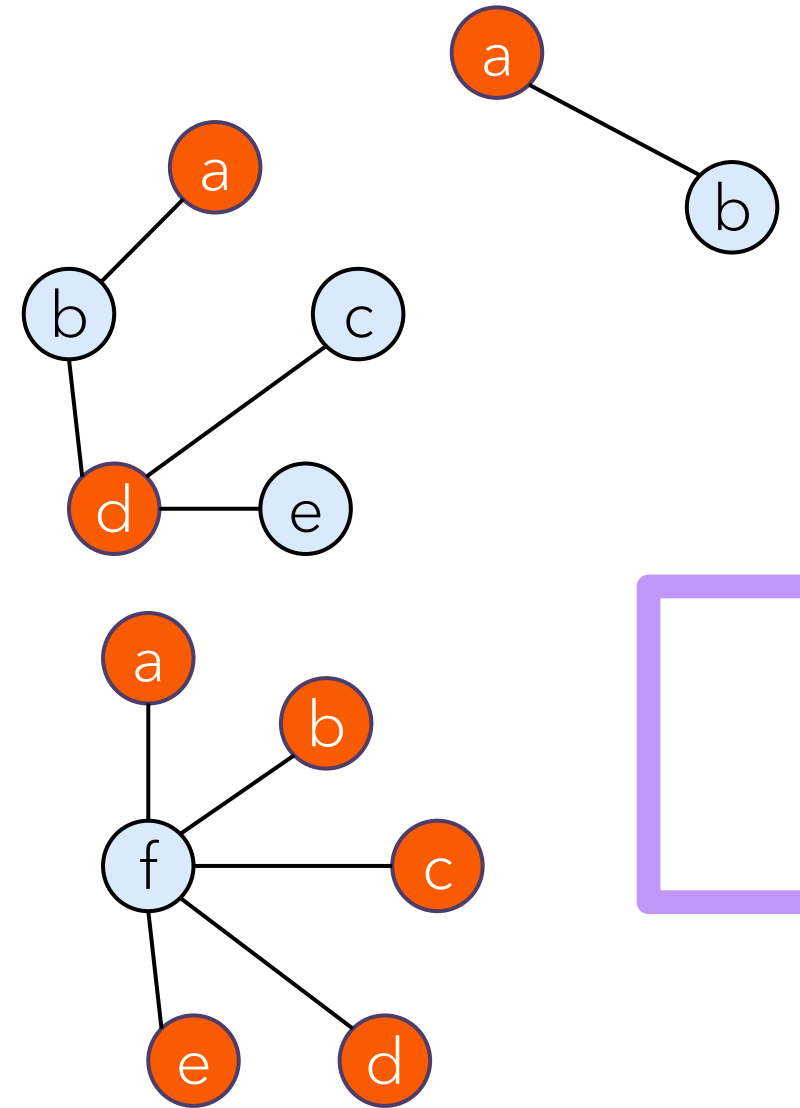
# Trees

- A tree is a connected graph with no cycles.
- Theorem: Every tree has a chromatic number of 2 or less.



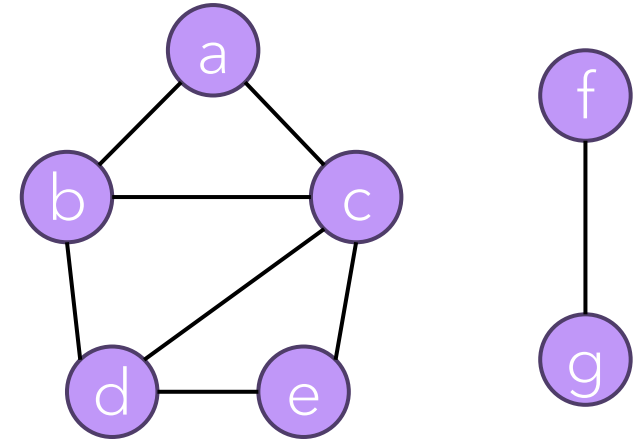
# Trees

- A tree is a connected graph with no cycles.
- Theorem: Every tree has a chromatic number of 2 or less.
  - Proof sketch: Pick a node and color it red. Color all of its neighbors blue. Color all of THEIR neighbors red, and continue alternating colors like this. Observe that at no point can we hit a node that was previously colored, because there are no cycles.



# Matching

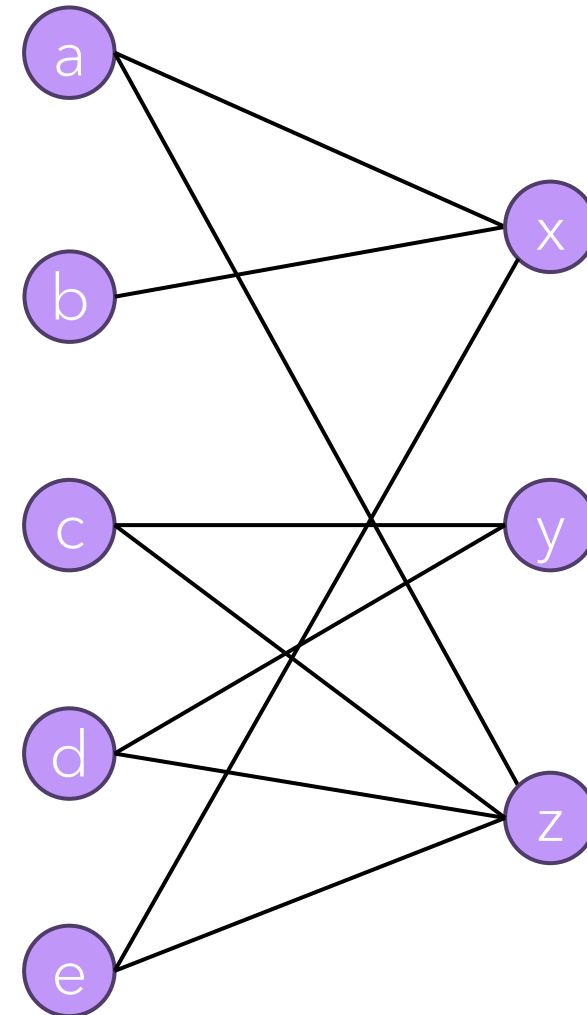
- There are two primary methods for representing a graph within a computer program.
- If you have a *dense* graph (i.e. lots of edges), an adjacency matrix can make sense.
  - Symmetric in undirected graphs.
  - Observe that it requires  $O(|V|^2)$  memory, and so it only makes sense when you have  $O(|V|^2)$  edges. But access  $O(1)$ !



	a	b	c	d	e	f	g
a	0	1	1	0	0	0	0
b	1	0	1	1	0	0	0
c	1	1	0	1	1	0	0
d	0	1	1	0	1	0	0
e	0	0	1	1	0	0	0
f	0	0	0	0	0	0	1
g	0	0	0	0	0	1	0

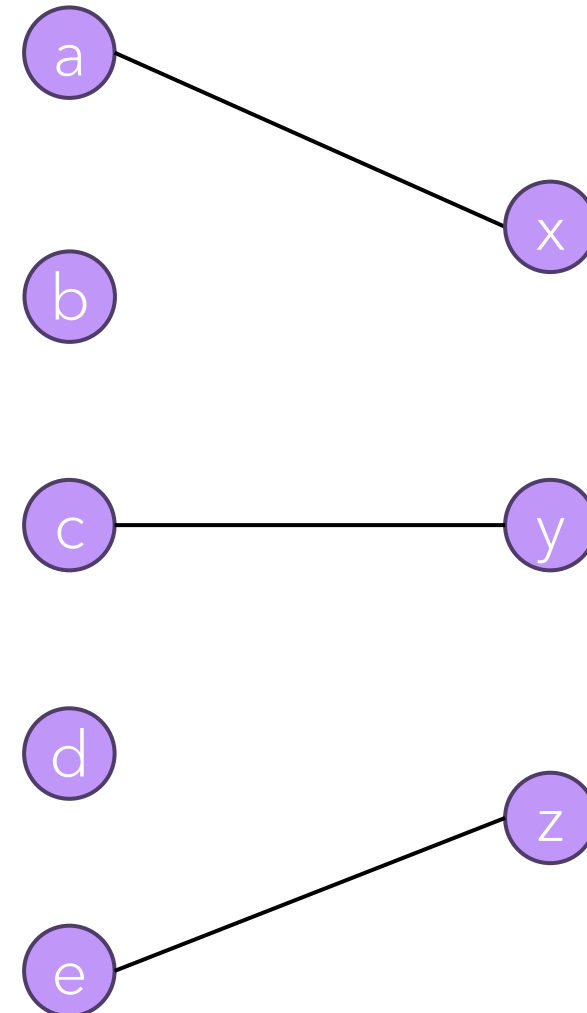
# Matching

- Recall that a bipartite graph is one in which we can separate our vertices into two groups, and that edges are only permitted from one group to the other.
- A matching is a subgraph which contains edges that cover all of the vertices in one of the sets.



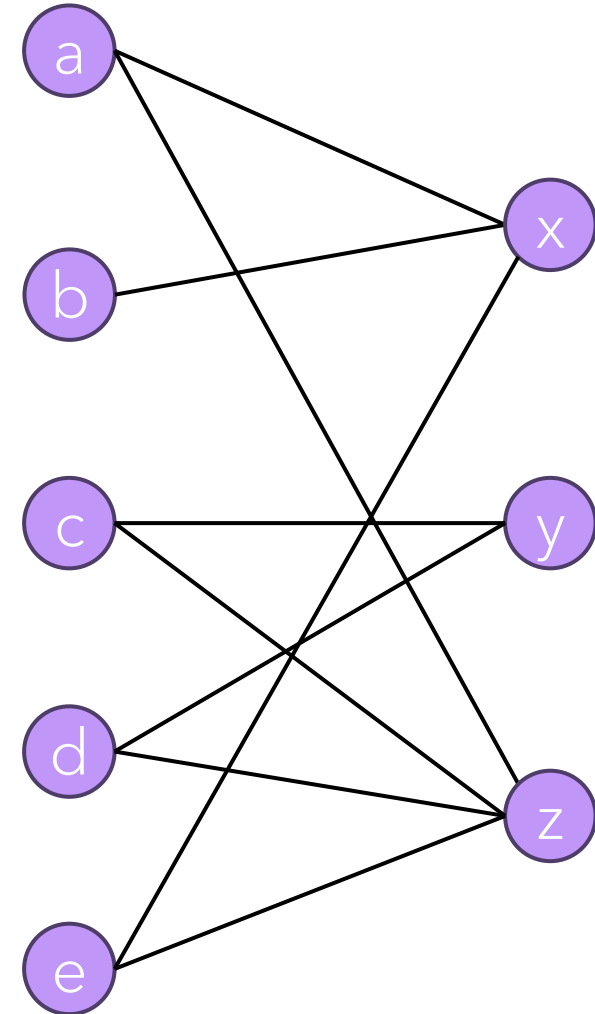
# Matching

- Recall that a bipartite graph is one in which we can separate our vertices into two groups, and that edges are only permitted from one group to the other.
- A matching is a subgraph which contains edges that cover all of the vertices in one of the sets.
  - A perfect matching (not possible here; requires both sets to be the same size) is when **both** sets are covered exactly once.



# When is a matching possible?

- To state the obvious, every subset of the set we want to cover needs at least as many neighbors as it has nodes. (In this example:  $\{x, y, z\}$  needs to have at least 3 neighbors;  $\{x, y\}$  must have at least two neighbors, etc.)
- Hall's Theorem: The above condition is **sufficient** to guarantee a matching.
  - Proof is by induction, given in the textbook.



# Matching problems

- Less obvious: Sudoku / Latin Squares
  - Start with  $K_{9,9}$  - each time a row is filled, some of the edges are removed.
- More obvious: Jobs / applicants

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


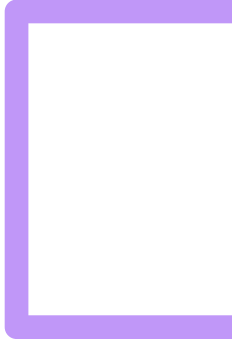


# The Stable Matching (Marriage) Problem

- Consider a group of job applicants, each applying for the same set of jobs. Happily, there are precisely the same number of positions as applicants, and so a *perfect matching* is possible.
- As you might imagine, each applicant has a ranking of which jobs they would prefer to land, though they will take what they can. Similarly, after looking at applications, each employer has a ranking of the candidates.
- Our matching is *unstable* if there are two candidates who land in jobs where each would prefer the others' job and the employers would prefer to switch, too.


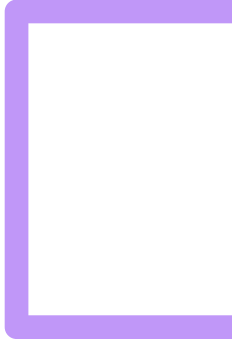


# Instability

- Employer A has hired candidate X, and employer B has hired candidate Y.
  - A ranked Y higher than X, and B ranked X higher than Y.
  - X ranked B higher than A, and Y ranked A higher than B.
  - Why wouldn't they switch?
  - But we'd like an assignment that was stable, rather than folks switching jobs a bunch.
- 
- 


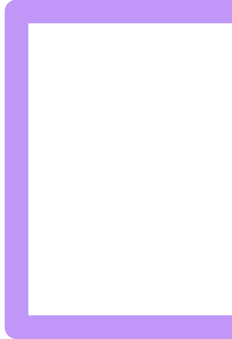


# The Stable Matching (Marriage) Problem

- Given the preference lists, how can we find a stable matching?
  - Well, it starts when an employer makes someone an offer.
    - Presumably, they'll start with their first-choice candidate.
    - Since that person hasn't had any offers yet, they will tentatively say yes.
  - What happens next?
- 
- 

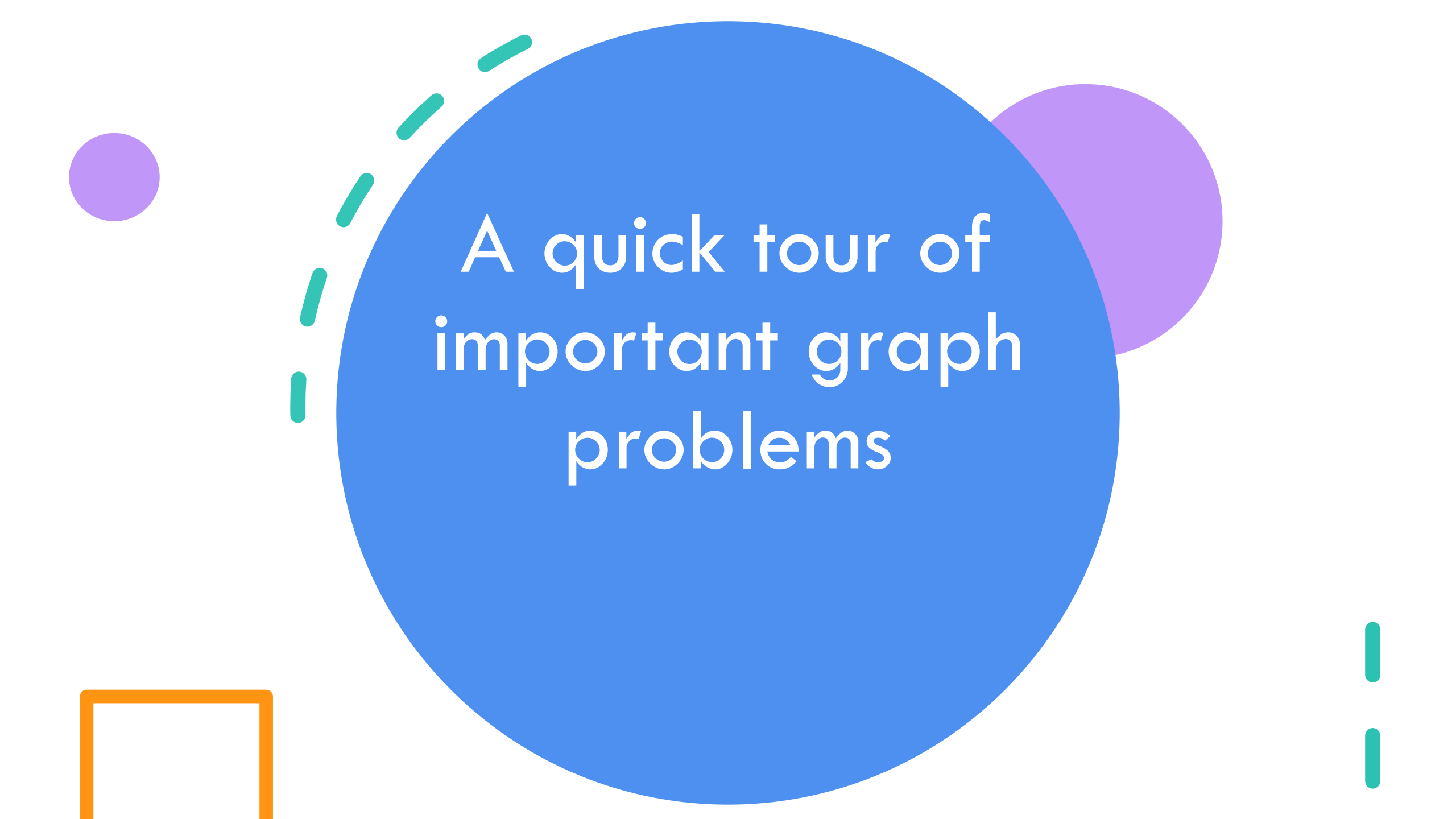


# The Stable Matching (Marriage) Problem

- The next employer makes *their* top choice an offer.
    - If they haven't had an offer yet, they will tentatively accept.
    - But if they HAVE had one, now they will look at their own preference list and either keep what they have, or trade up.
  - And this process of employers making offers repeats, with the only condition that an employer will never make someone a second offer.
    - When you're done, you will have a stable matching!
- 
- 

# The Gale-Shapley Algorithm

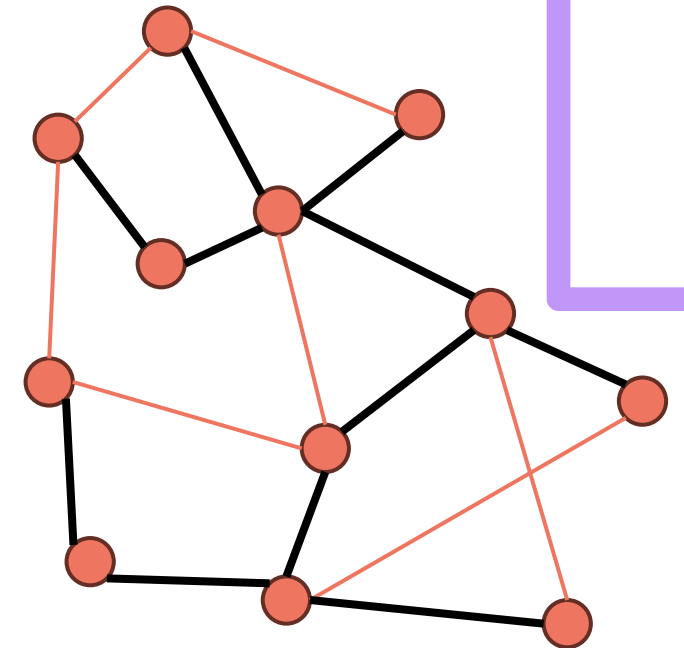
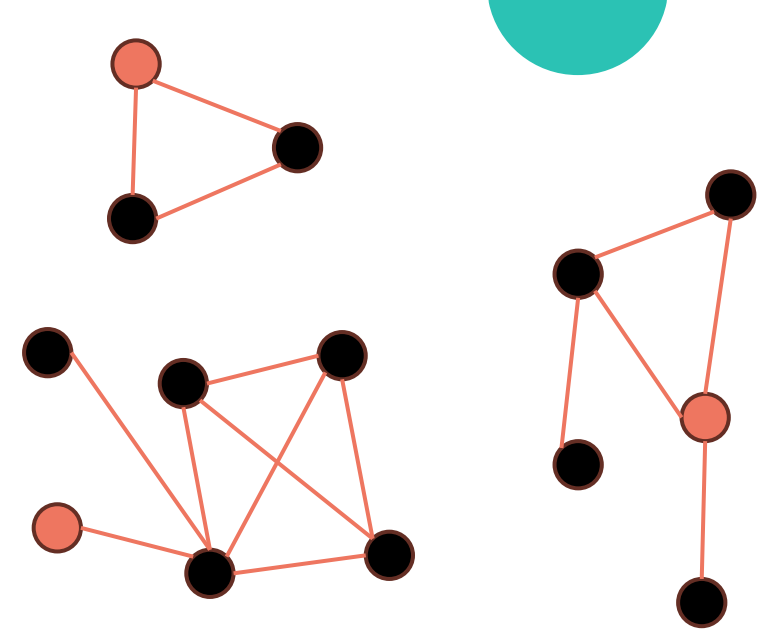
- The process previously described (and the proof of its correctness) was published by David Gale and Lloyd Shapley in 1962...
  - ... But the National Resident Matching Program (known simply as “The Match” to med students) had been using it for a decade before that!
  - It is also used in Web traffic regulation and a variety of other applications.
  - Shapley (along with Alvin Roth) won the Nobel Prize in economics for work on this and related problems in 2012.



# A quick tour of important graph problems

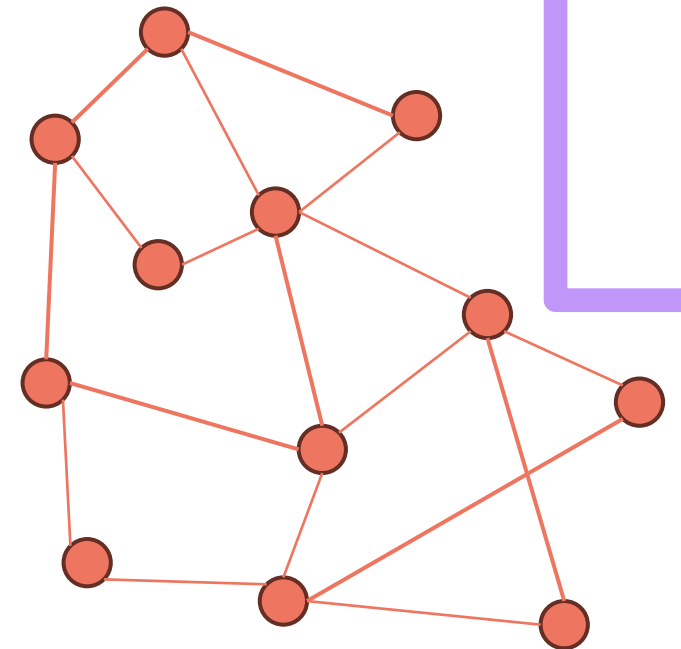
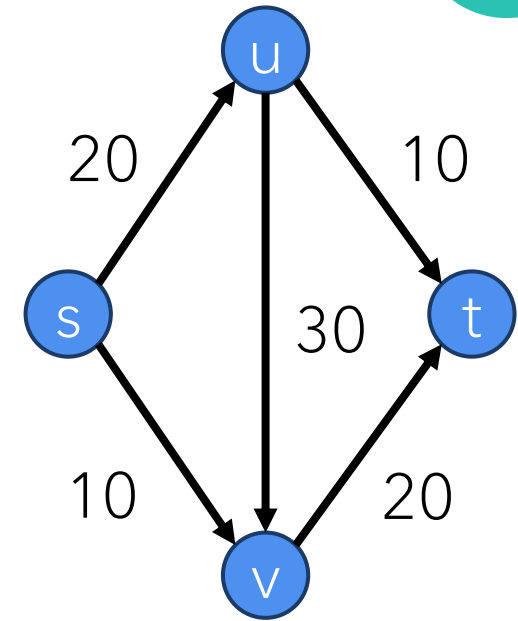
# First the easy ones...

- Representatives: Given an arbitrary (possibly disconnected) graph, select one vertex from each connected component.
- Spanning Tree: Given a graph (usually weighted), find the smallest set of edges that keeps the graph connected.



# Still easy...

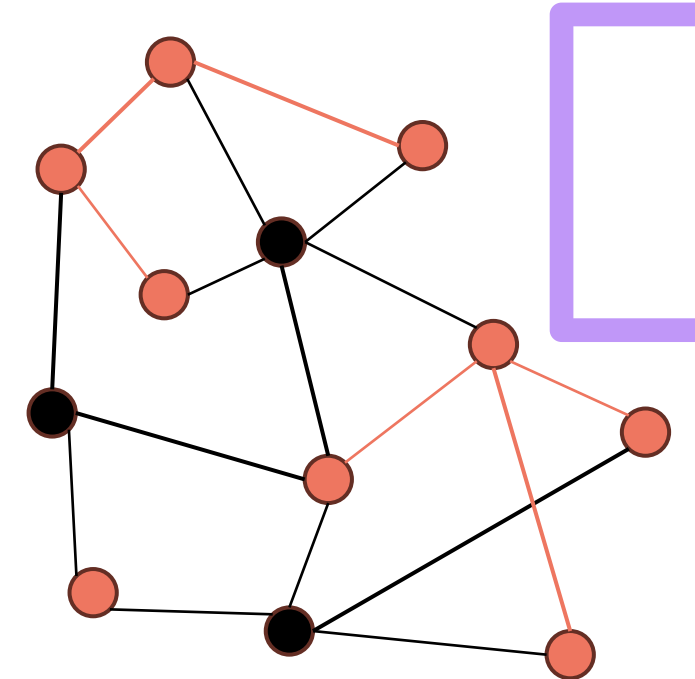
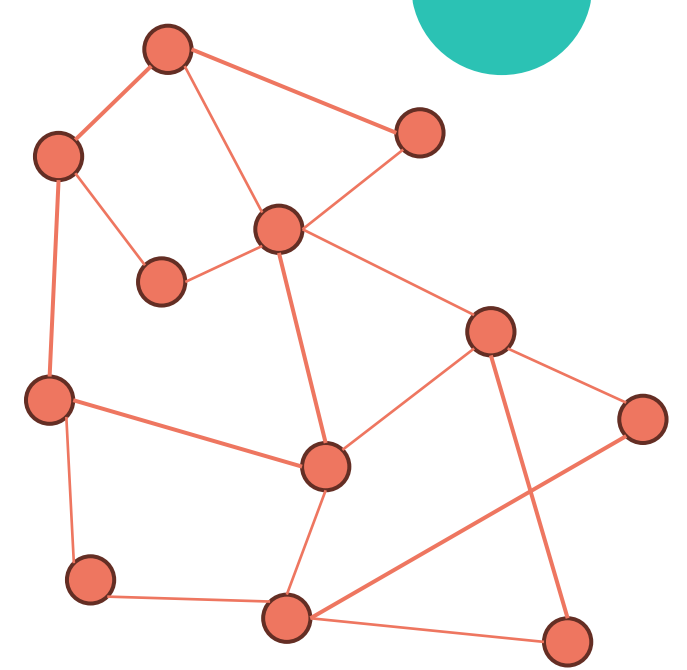
- Network Flow: In a directed graph with capacities on each edge, determine the maximum amount of flow from one node to another.
- Euler Path/Cycle: Find a path/cycle that walks every edge exactly one time. (snowplows)





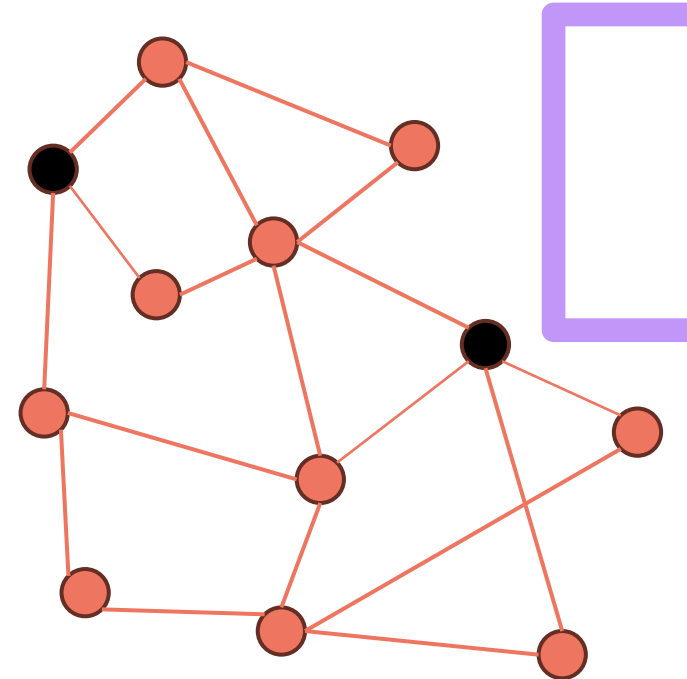
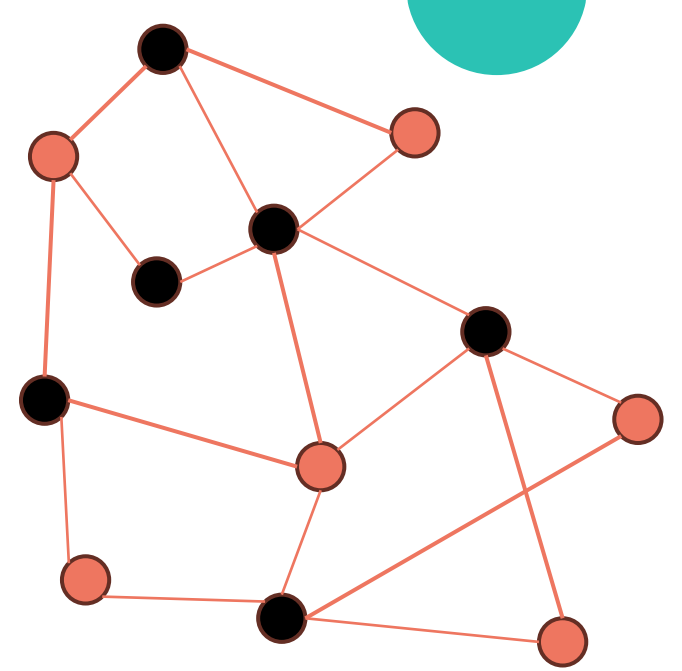
# Not easy.

- Hamiltonian Path/Cycle: Find a cycle that visits every vertex exactly once. (mailman)
- Dominating Set: Select the smallest set of vertices whose edges cover all vertices in the graph.



## Also not easy.

- Vertex Cover: Select the smallest set of vertices that neighbor every edge.
- K-center: Select the set of  $k$  vertices such that every vertex in the graph is as close as possible to one of them (facility location).



# Questions?

