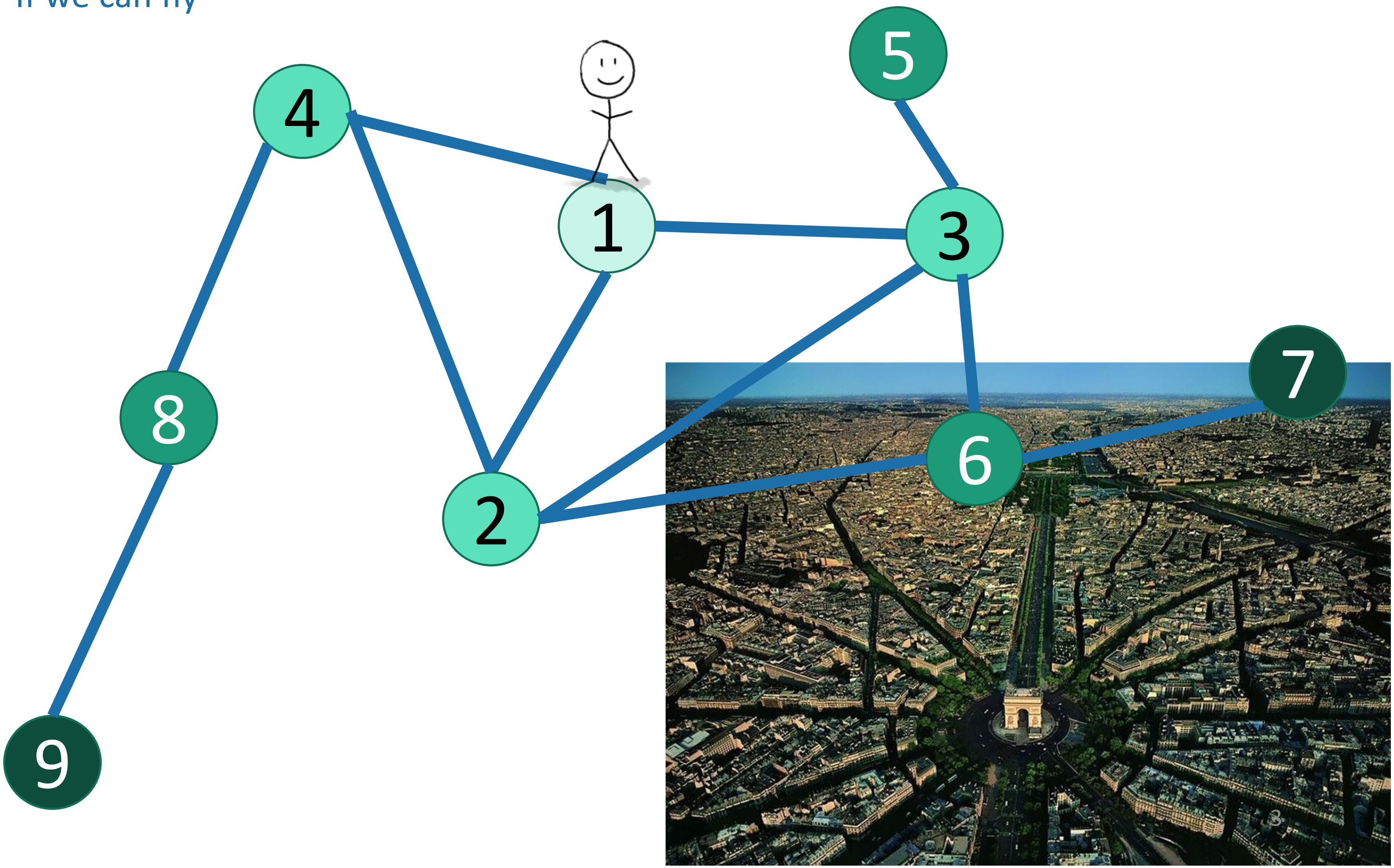
Advanced Data Structures and Algorithms

Breadth First Search (BFS)

Breadth-first search

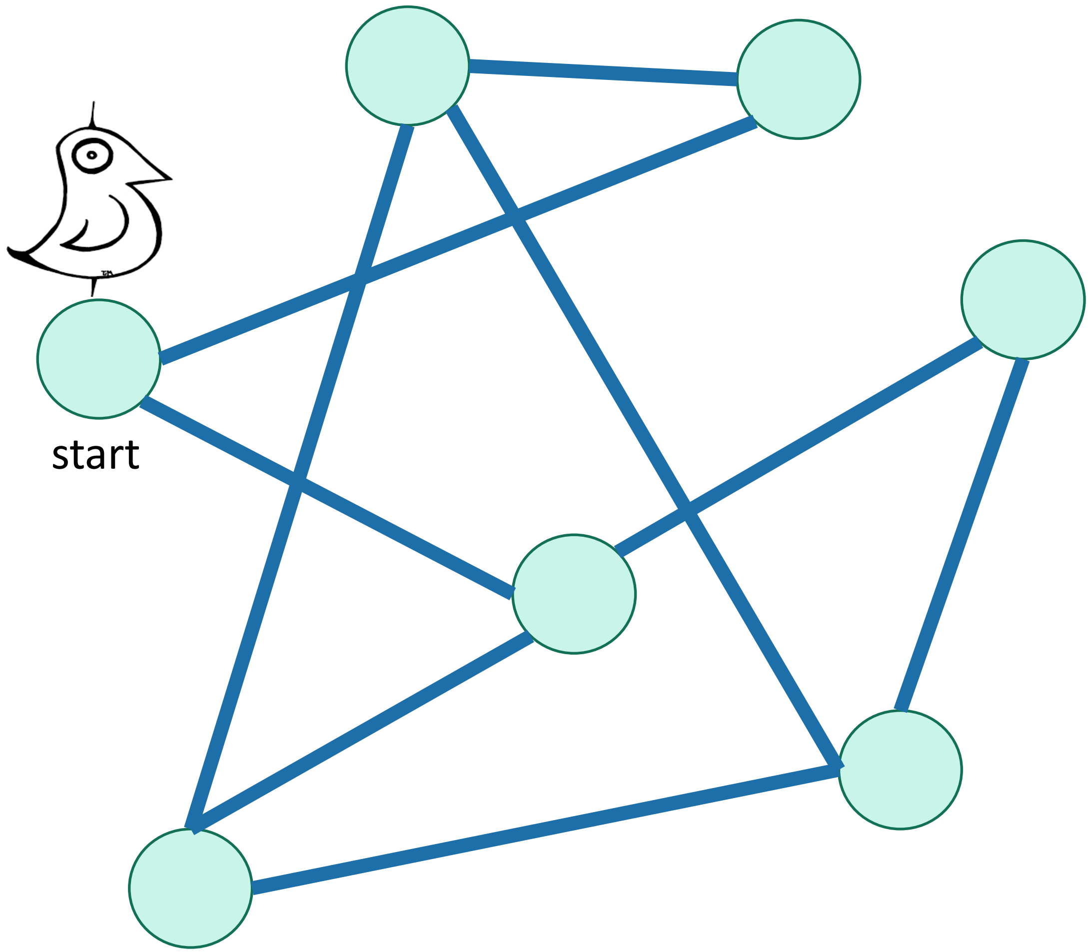
How do we explore a graph?

If we can fly



Breadth-First Search

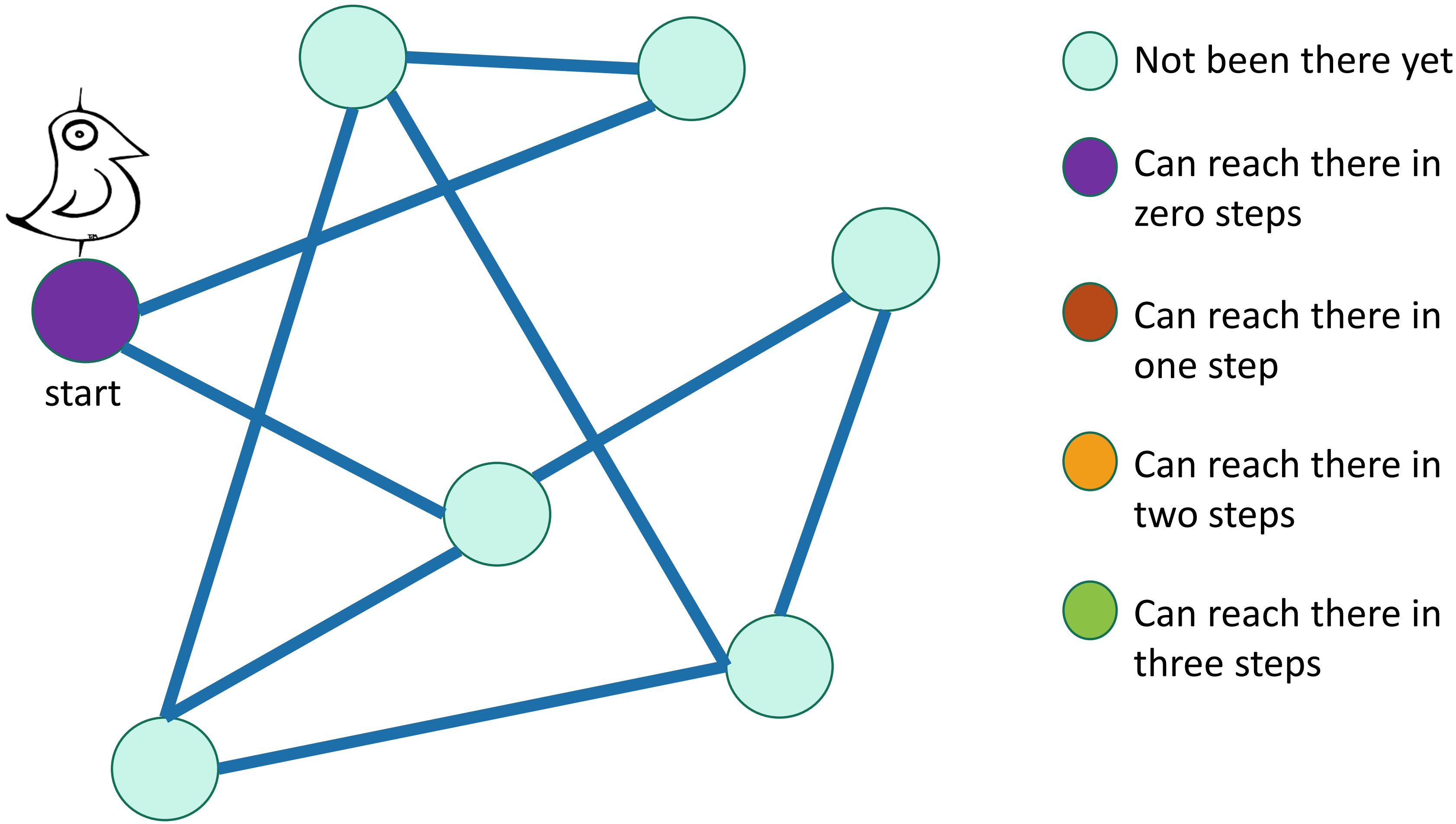
Exploring the world with a bird's-eye view



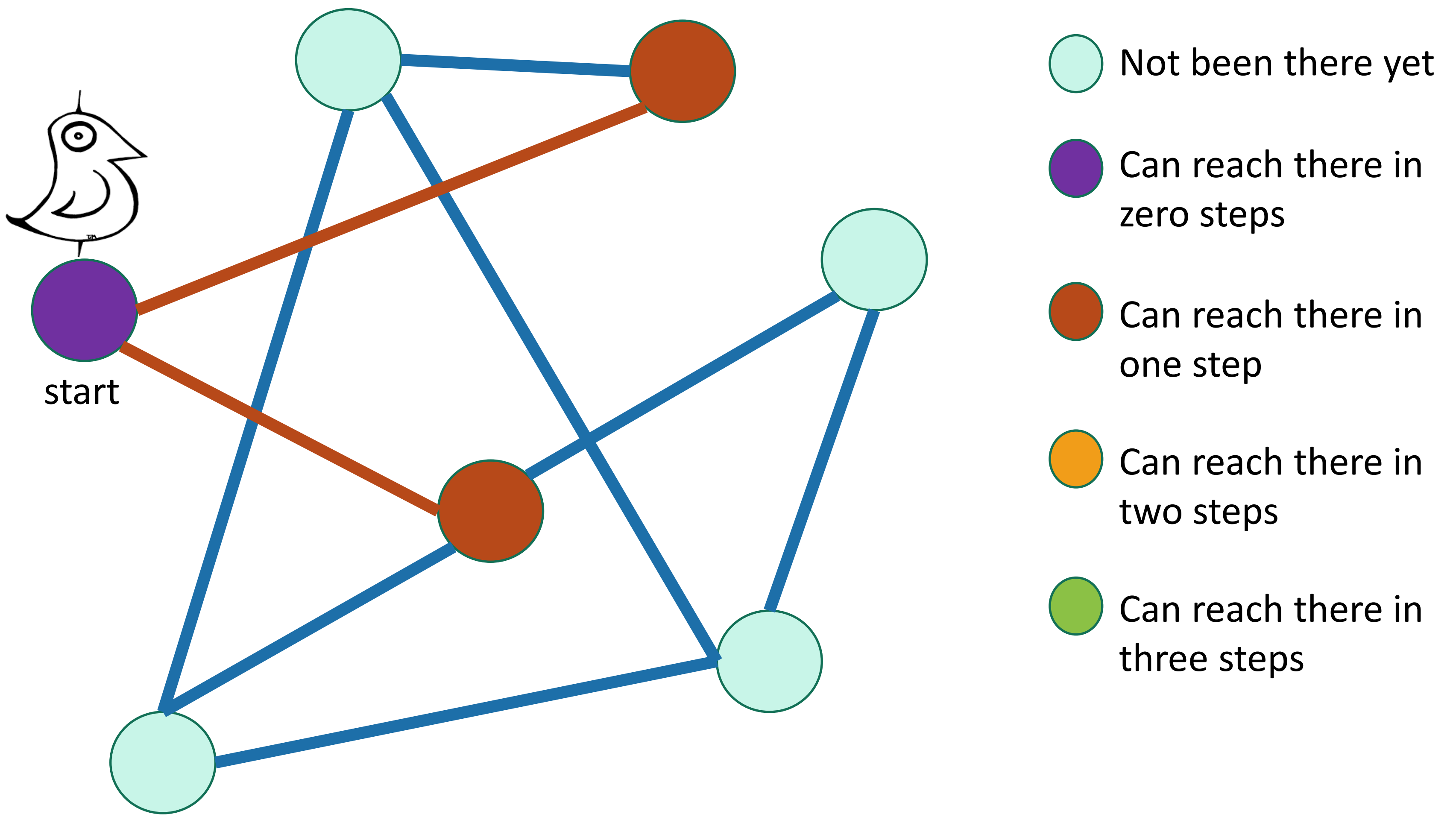
- Not been there yet
- Can reach there in zero steps
- Can reach there in one step
- Can reach there in two steps
- Can reach there in three steps

Breadth-First Search

Exploring the world with a bird's-eye view

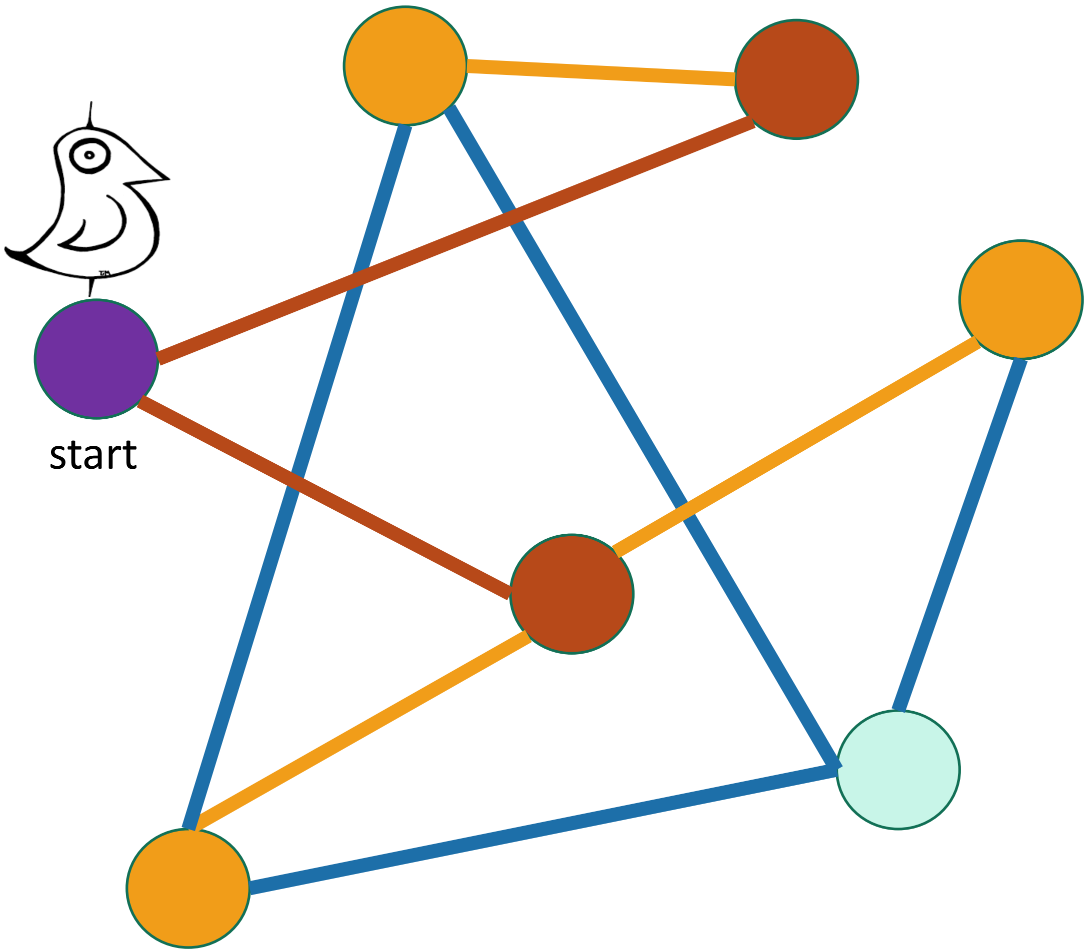


Breadth-First Search
Exploring the world with a bird's-eye view



Breadth-First Search

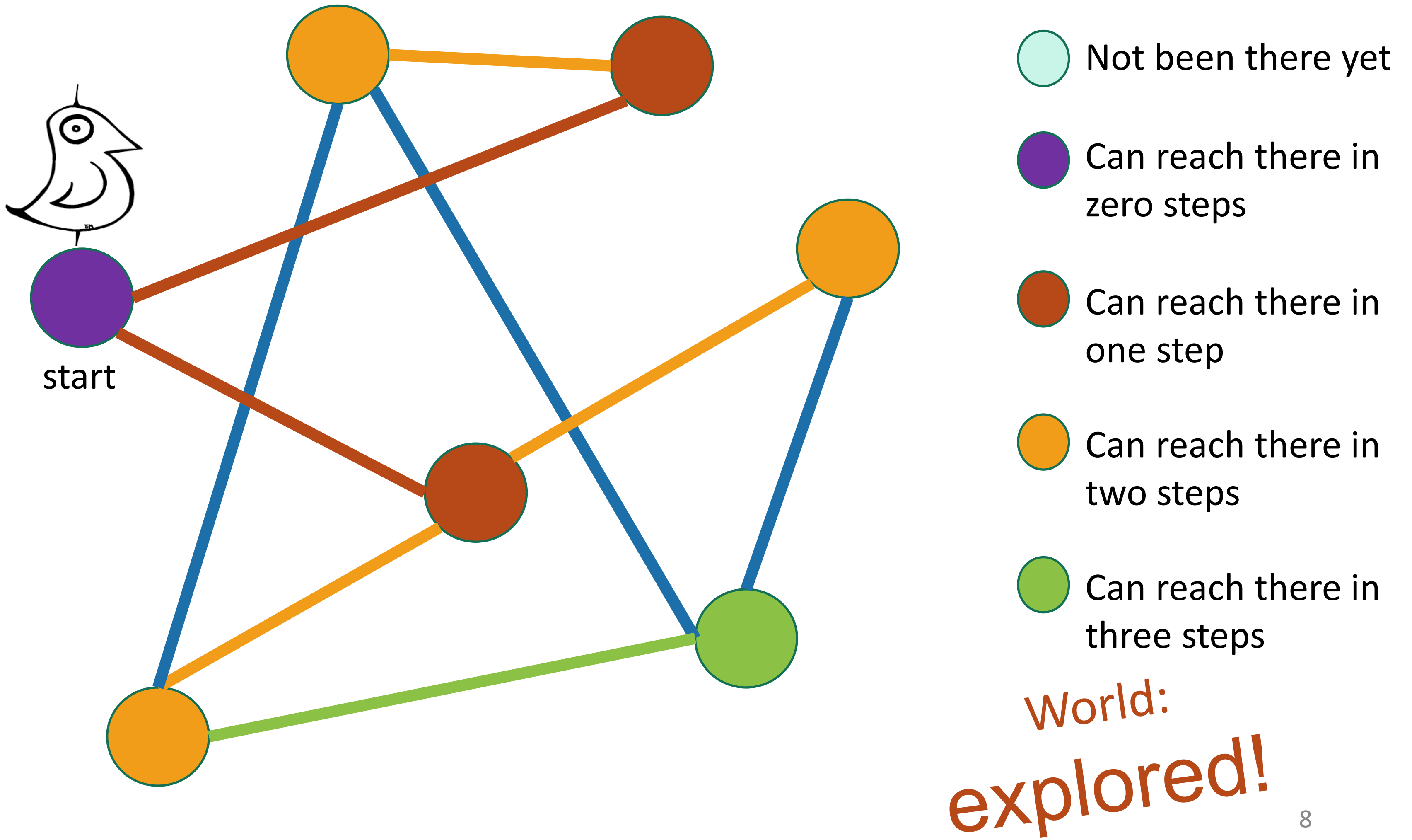
Exploring the world with a bird’s-eye view



- Not been there yet
- Can reach there in zero steps
- Can reach there in one step
- Can reach there in two steps
- Can reach there in three steps

Breadth-First Search

Exploring the world with a bird's-eye view



Same disclaimer as for DFS: you may have seen other ways to implement this, this will be convenient for us.

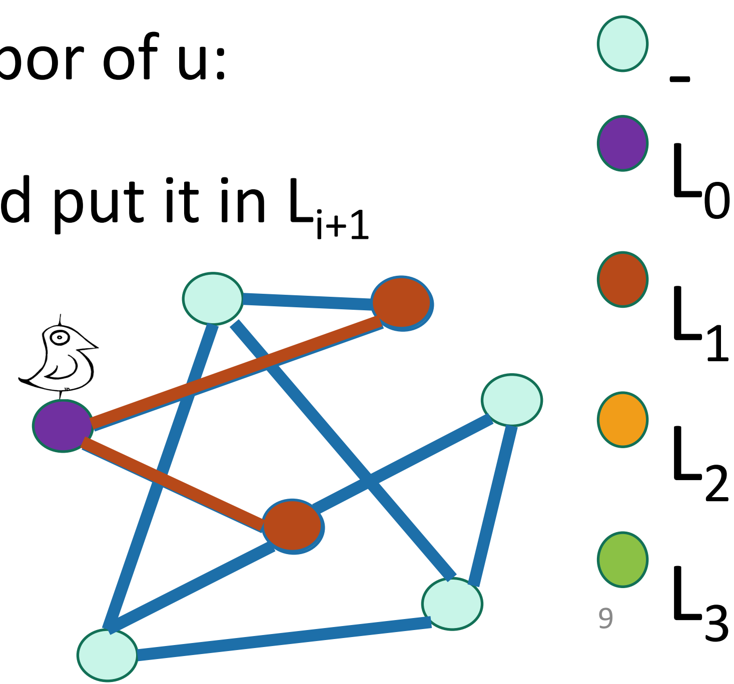
Breadth-First Search

Exploring the world with pseudocode

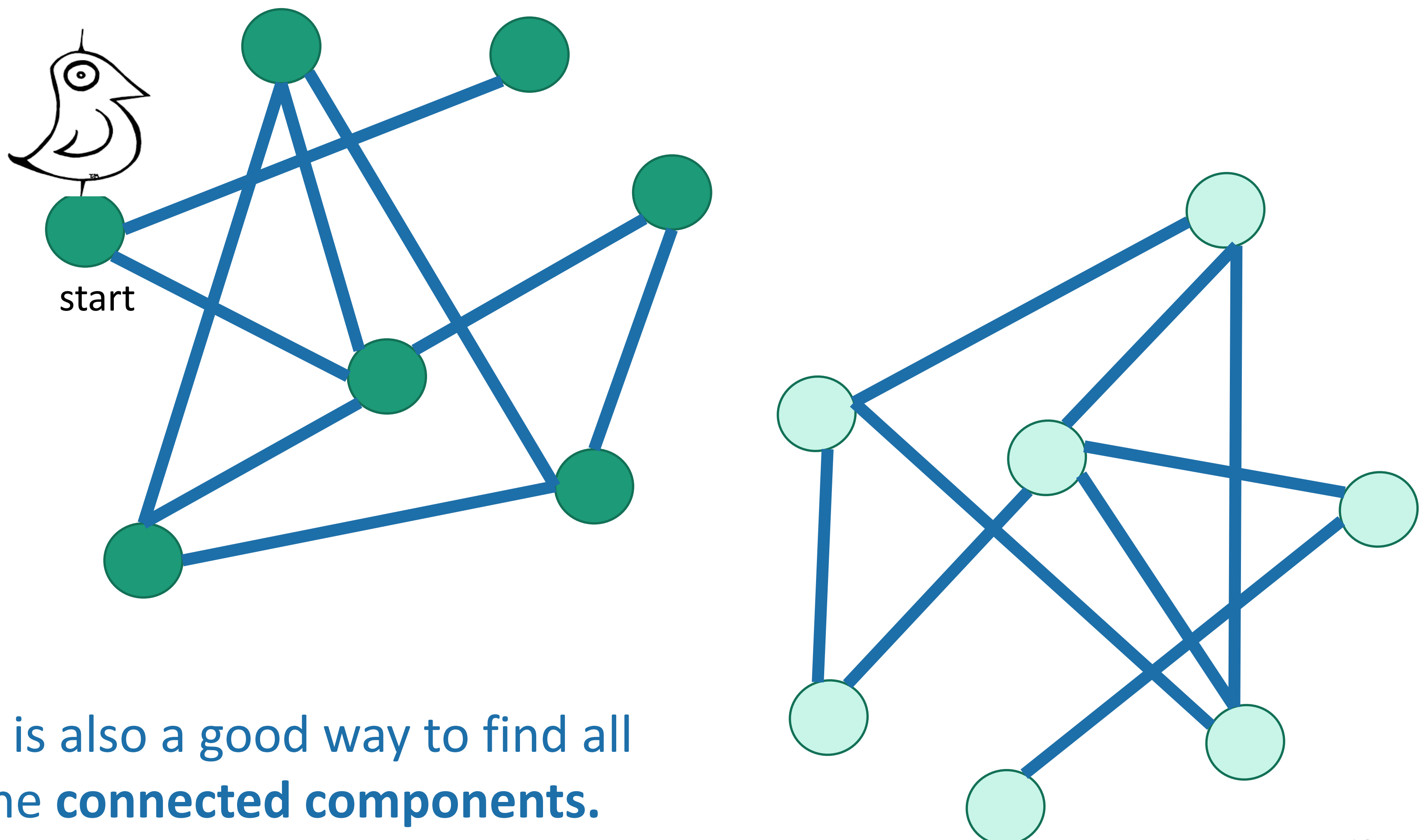
- Set L_i = [] for i=1,…,n
- L_0 = [w], where w is the start node
- Mark w as visited
- **For** i = 0, …, n-1:
  - **For** u in L_i:
    - **For** each v which is a neighbor of u:
      - **If** v isn't yet visited:
        - mark v as visited, and put it in L_{i+1}

L_i is the set of nodes we can reach in i steps from w

Go through all the nodes in L_i and add their unvisited neighbors to L_{i+1}



BFS also finds all the nodes reachable from the starting point



It is also a good way to find all the **connected components.**

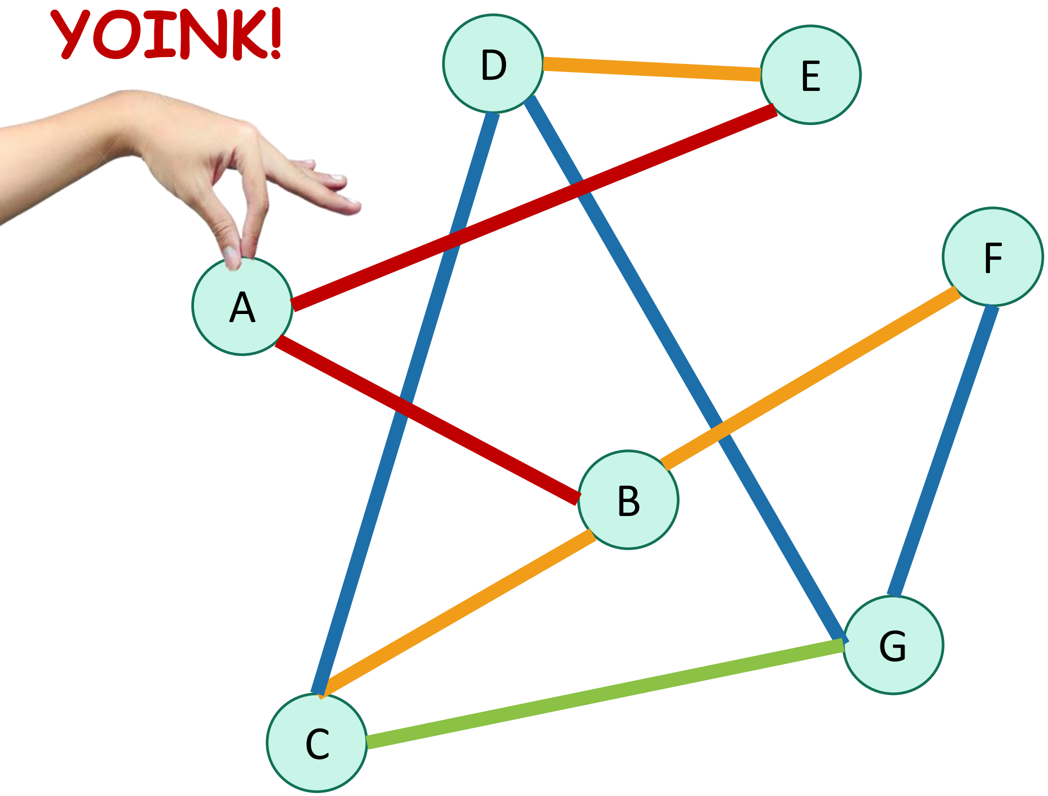
Running time and extension to directed graphs

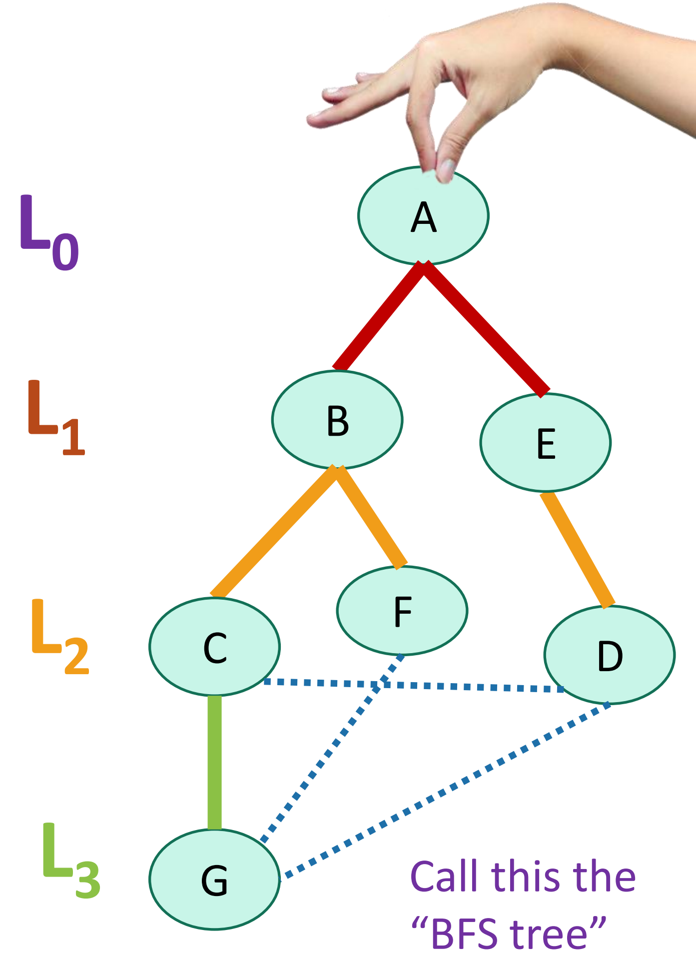
- To explore the whole graph, explore the connected components one-by-one.
  - Same argument as DFS: BFS running time is O(n + m)
- Like DFS, BFS also works fine on directed graphs.

Verify these!

Why is it called breadth-first?

- We are implicitly building a tree:

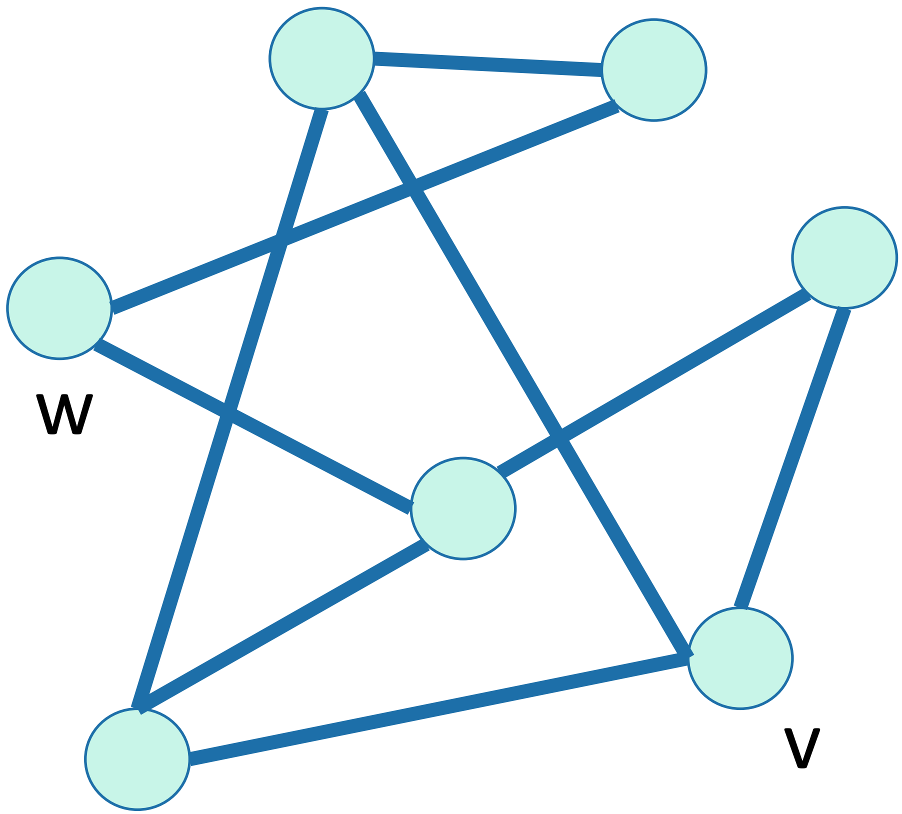




- First we go as broadly as we can.

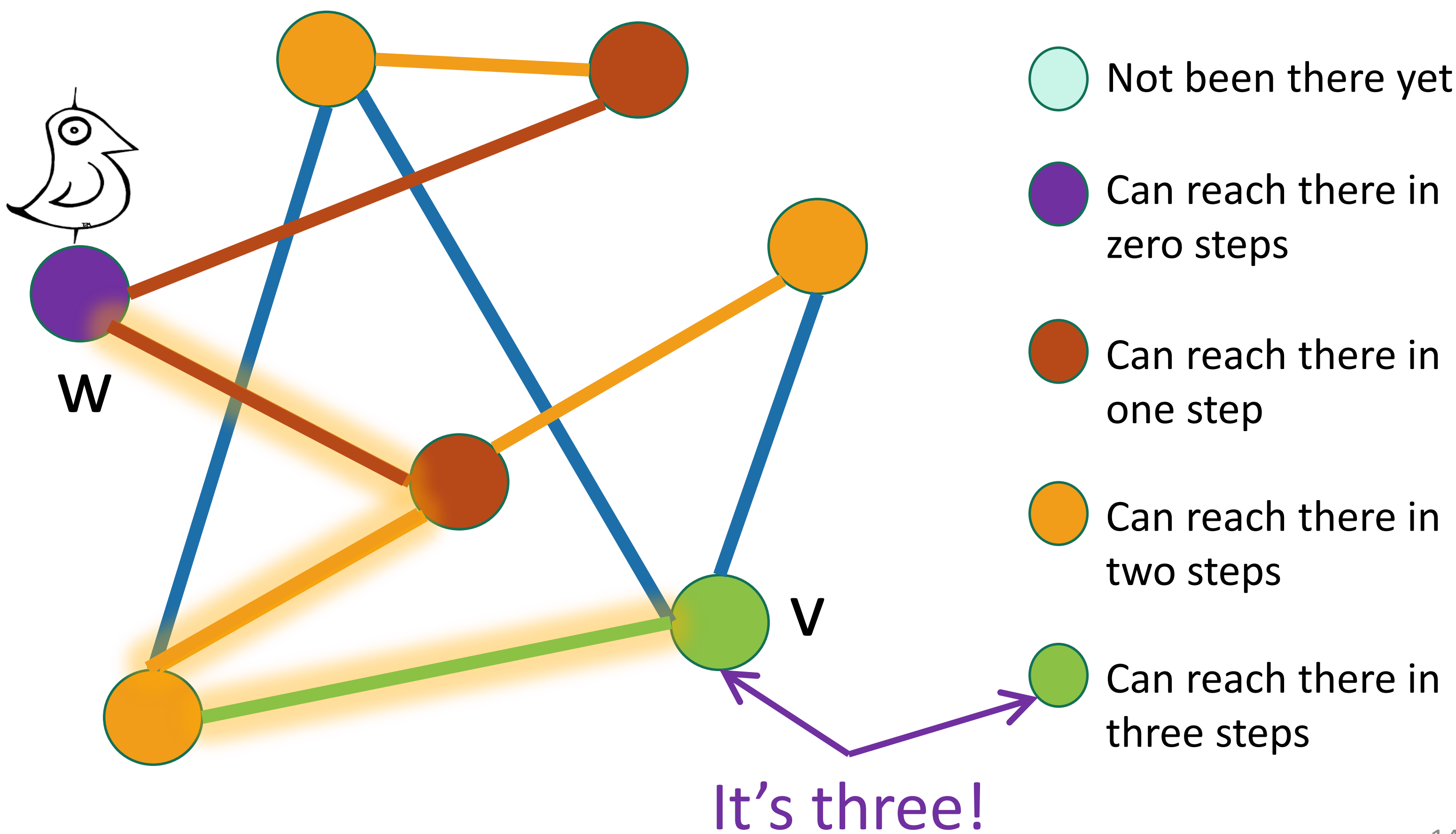
Application of BFS: shortest path

- How long is the shortest path between w and v?



Application of BFS: shortest path

- How long is the shortest path between w and v?

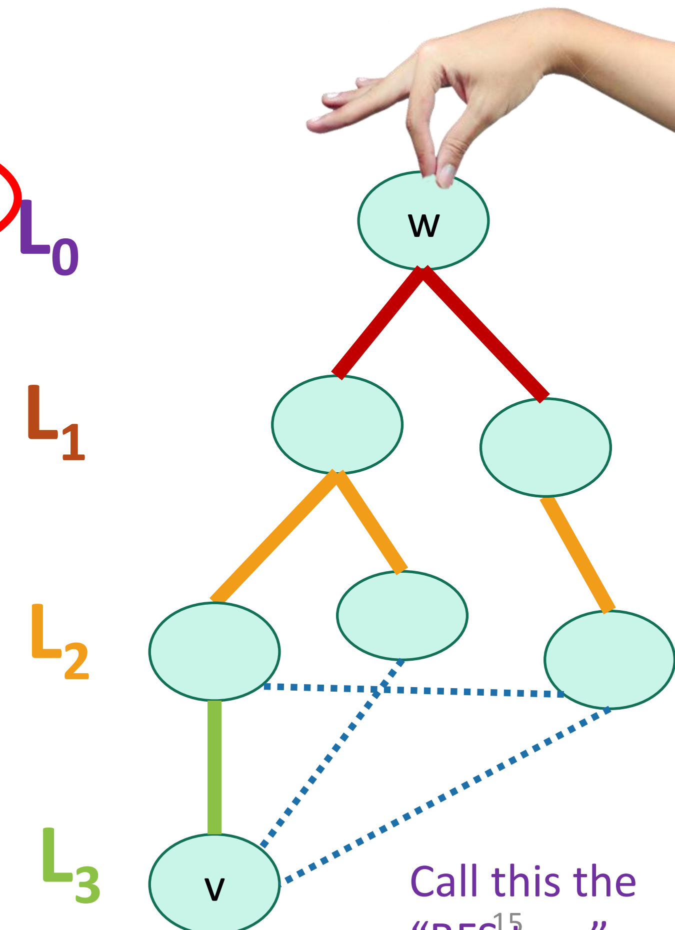


To find the distance between w and all other vertices v

The **distance** between two vertices is the number of edges in the shortest path between them.

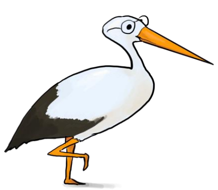
- Do a BFS starting at w
- For all v in L_i
  - The shortest path between w and v has length i
  - A shortest path between w and v is given by the path in the BFS tree.
- If we never found v, the distance is infinite.

This requires some proof!



Call this the "BFS tree"

Modify the BFS pseudocode to return shortest paths!



Proof overview
that the BFS tree behaves like it should

- Proof by induction.
- Inductive hypothesis for j:
  - For all i<j the vertices in L_i have distance i from v.
- Base case:
  - L_0 = {v}, so we're good.
- Inductive step:
  - Let w be in L_j. Want to show dist(v,w) = j.
  - We know dist(v,w) $\leq$ j, since dist(v, w's parent in L_{j-1}) = j-1 by induction, so that gives a path of length j from v to w.
  - On the other hand, dist(v,w) $\geq$ j, since if dist(v,w) < j, w would have shown up in an earlier layer.
  - Thus, dist(v,w) = j.
- Conclusion:
  - For each vertex w in V, if w is in L_j, then dist(v,w) = j.

What have we learned?

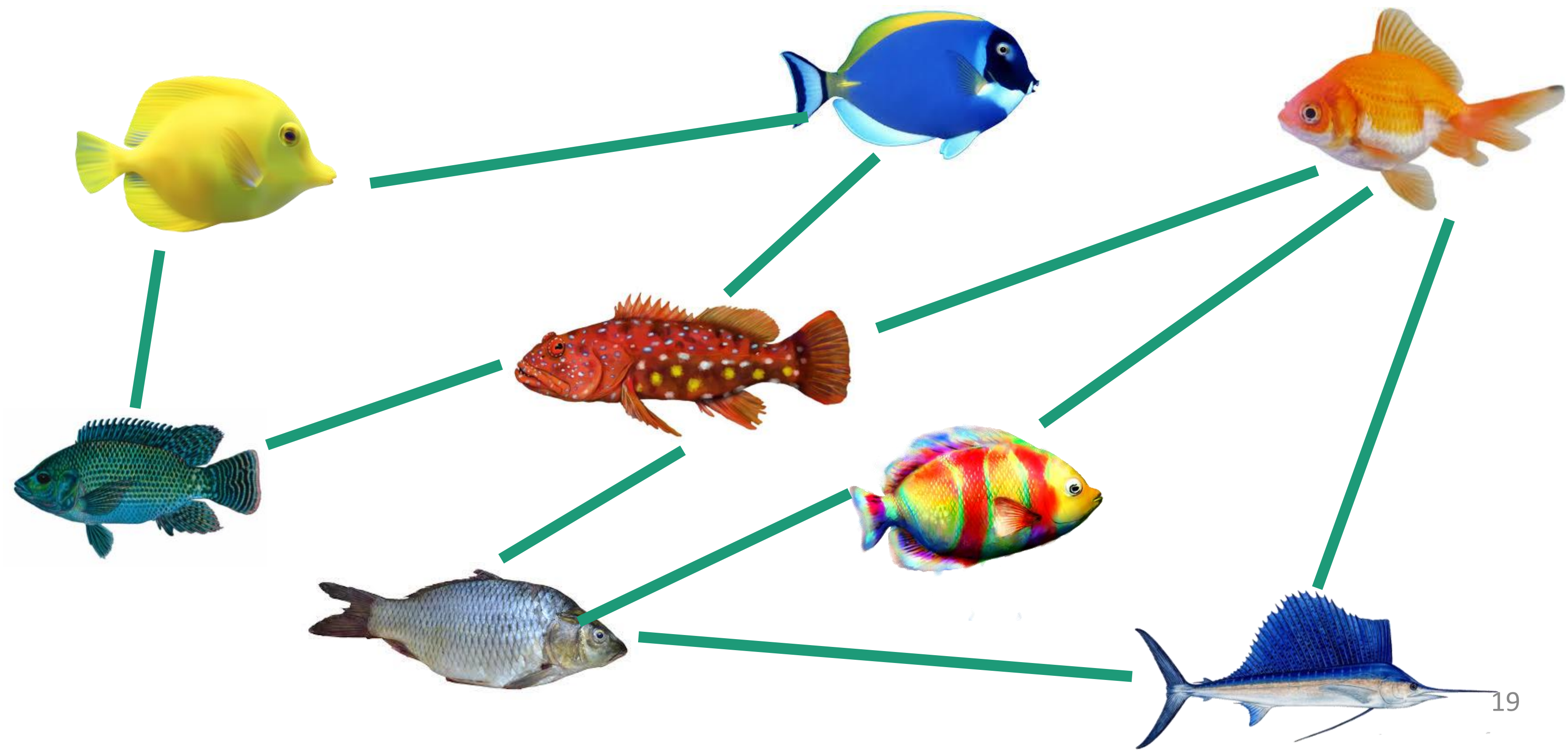
- The BFS tree is useful for computing distances between pairs of vertices.
- We can find the shortest path between u and v in time O(m).

Another application of BFS

- Testing bipartite-ness

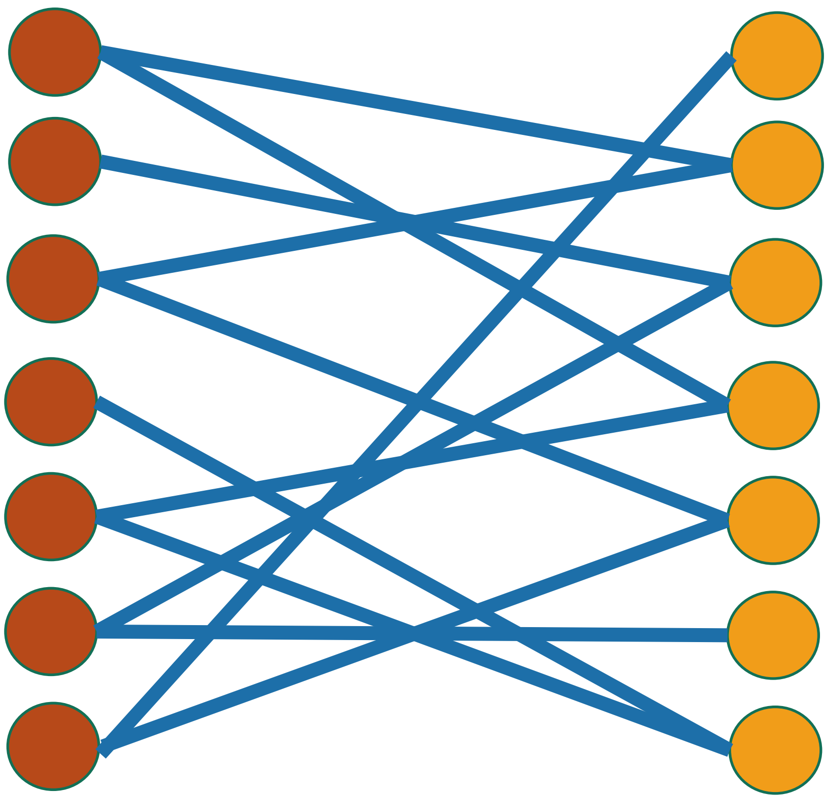
Exercise: fish

- You have a bunch of fish and two fish tanks.
- Some pairs of fish will fight if put in the same tank.
  - Model this as a graph: connected fish will fight.
- Can you put the fish in the two tanks so that there is no fighting?



Bipartite graphs

- A bipartite graph looks like this:

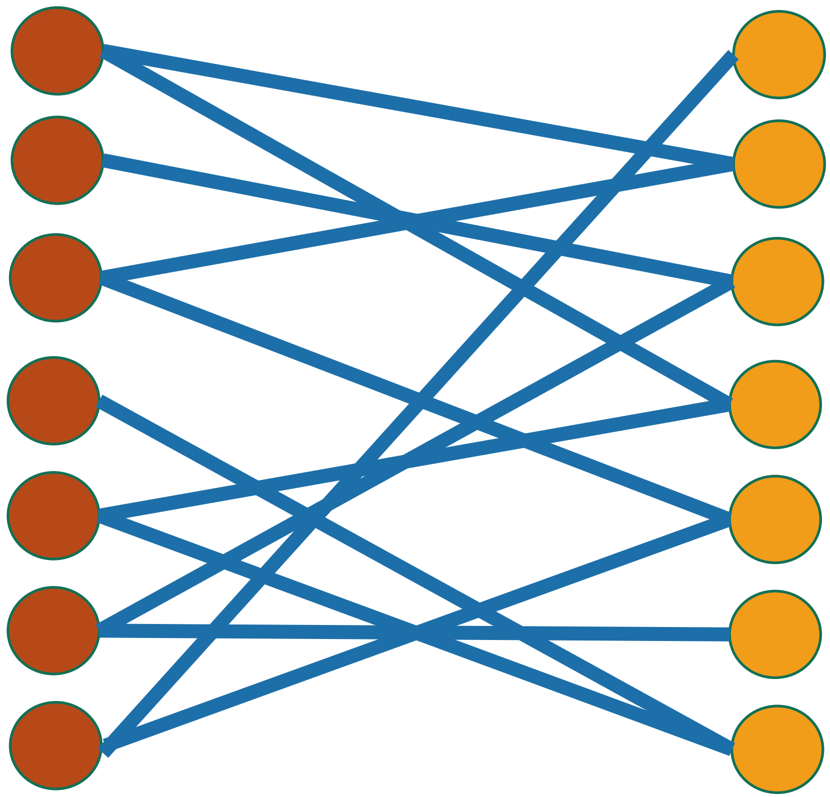


Can color the vertices red and orange so that there are no edges between any same-colored vertices

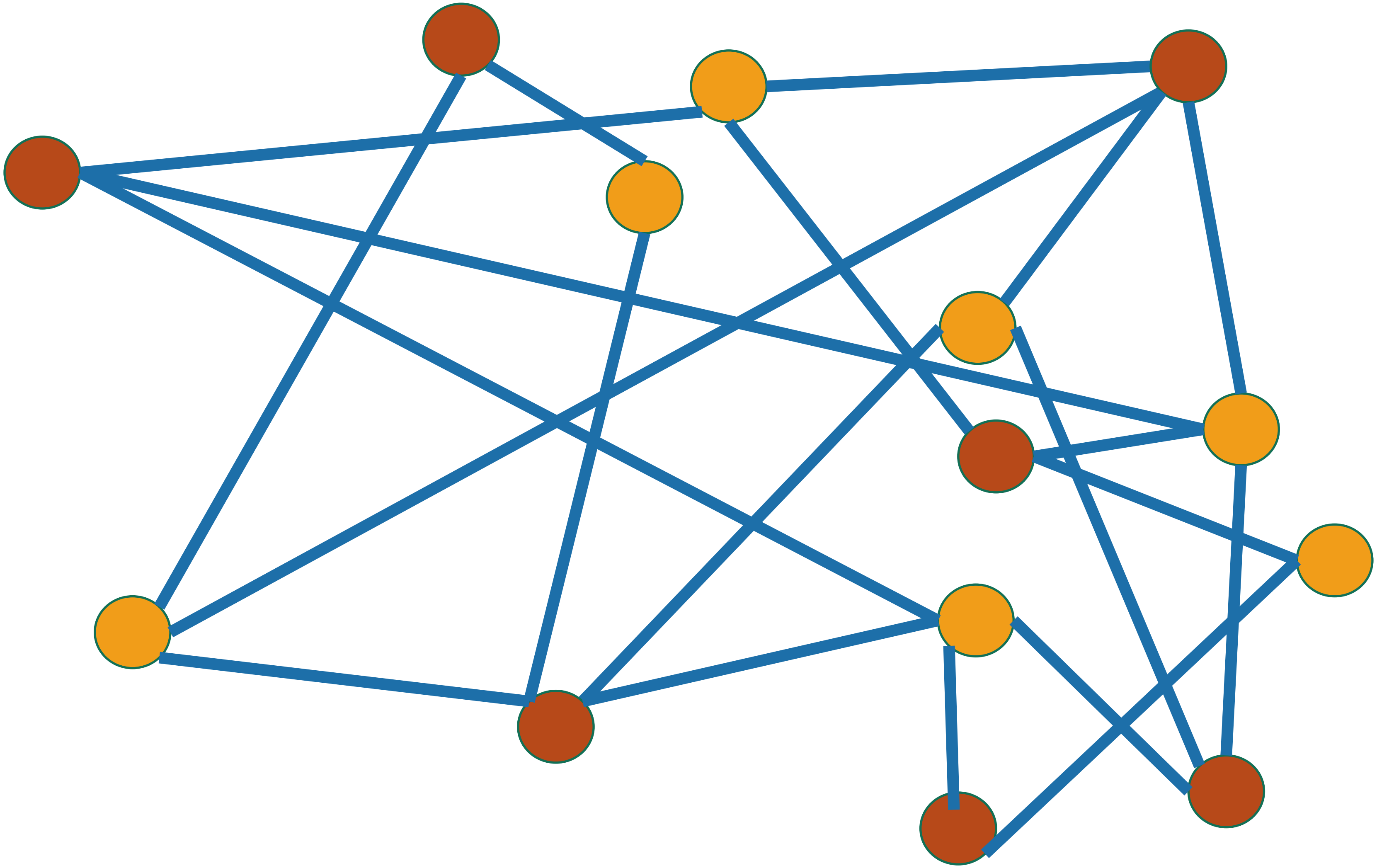
Example:
are in tank A
are in tank B
if the fish fight

Example:
are students
are classes
if the student is enrolled in the class

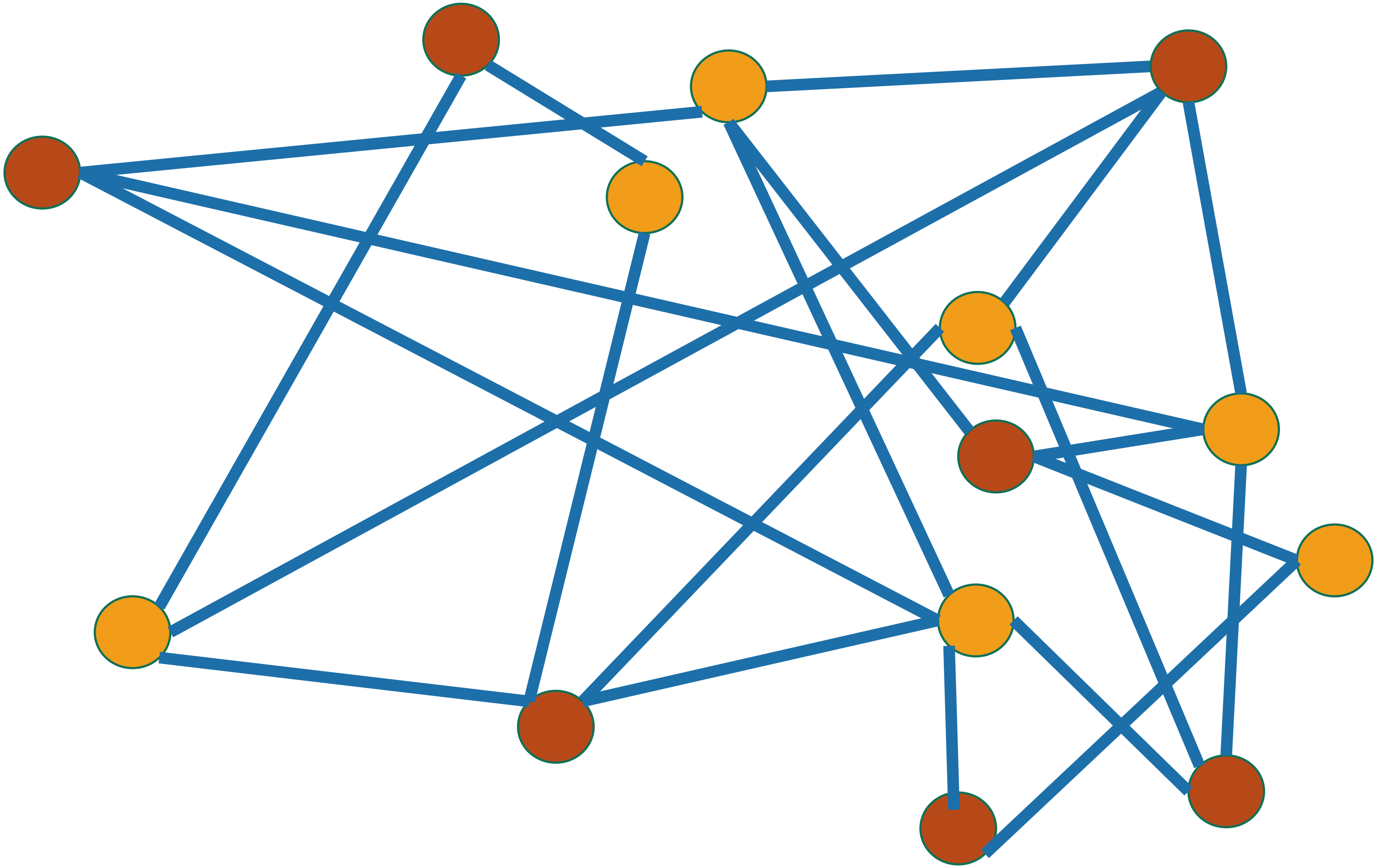
Is this graph bipartite?



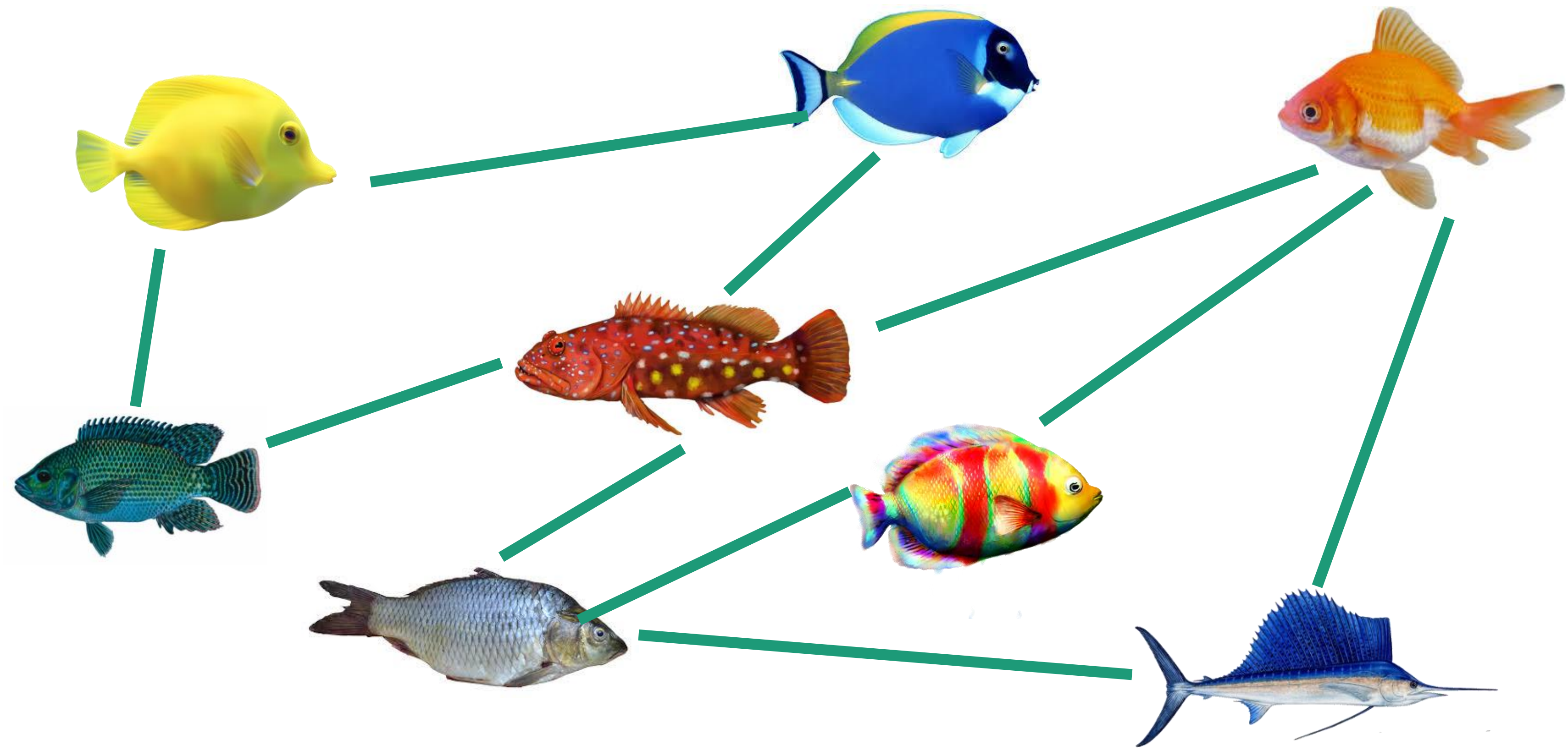
How about this one?



How about this one?

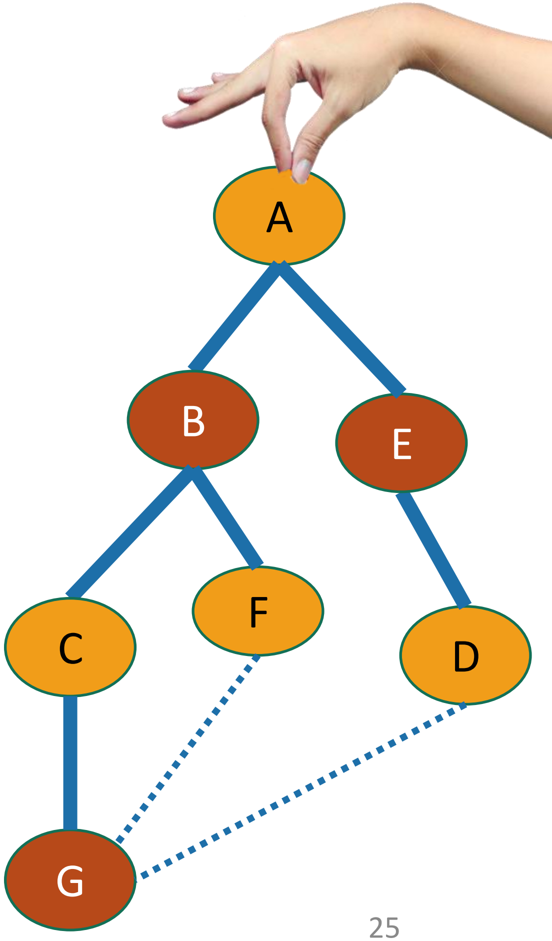


This one?



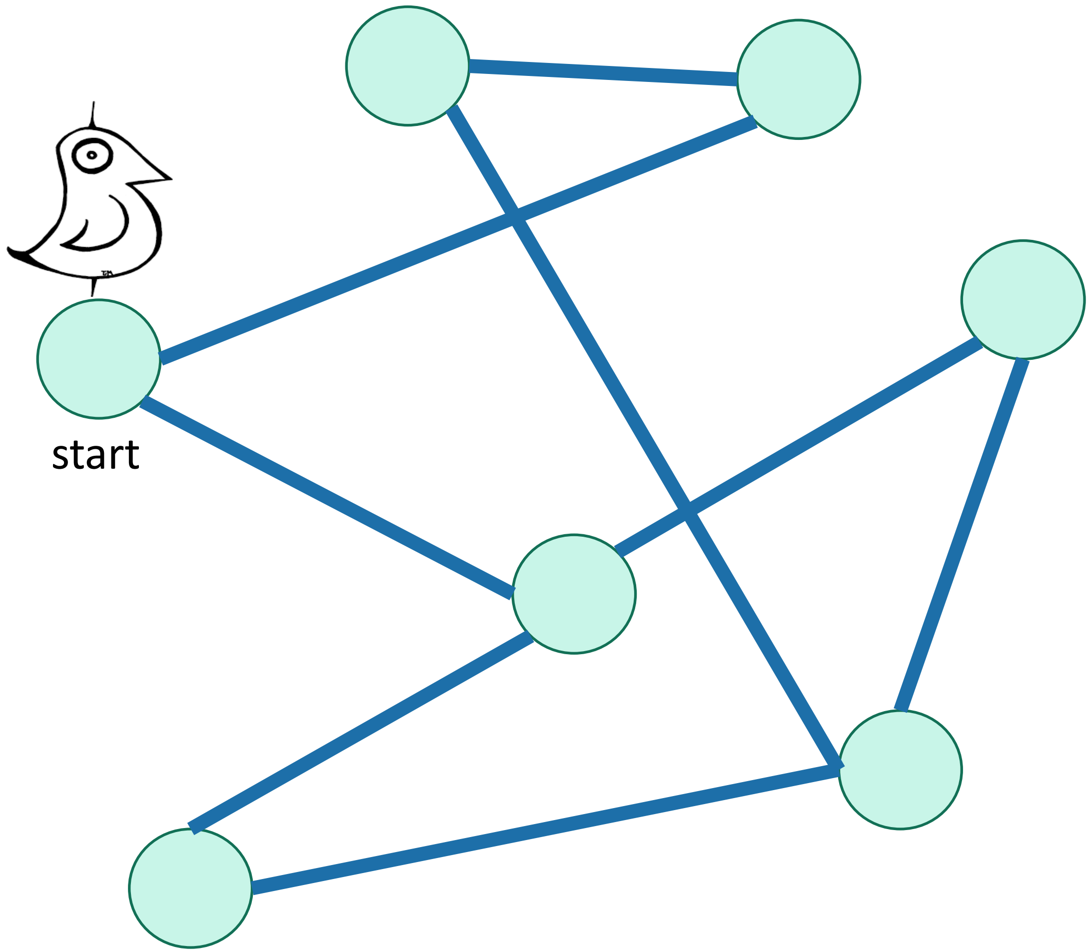
Application of BFS:
Testing Bipartiteness

- Color the levels of the BFS tree in alternating colors.
- If you never color two connected nodes the same color, then it is bipartite.
- Otherwise, it's not.



Breadth-First Search

For testing bipartite-ness



Not been there yet

Can reach there in zero steps

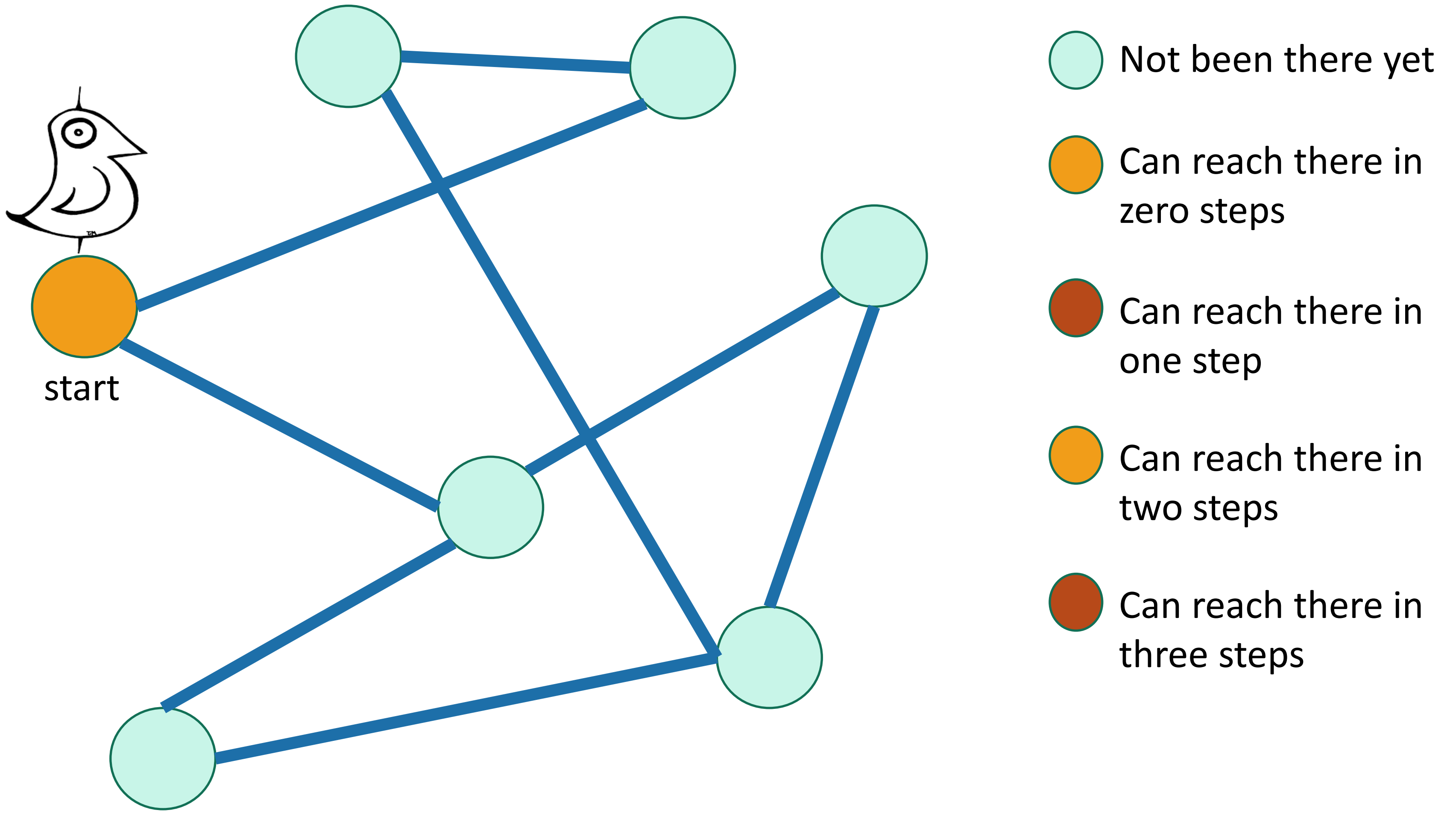
Can reach there in one step

Can reach there in two steps

Can reach there in three steps

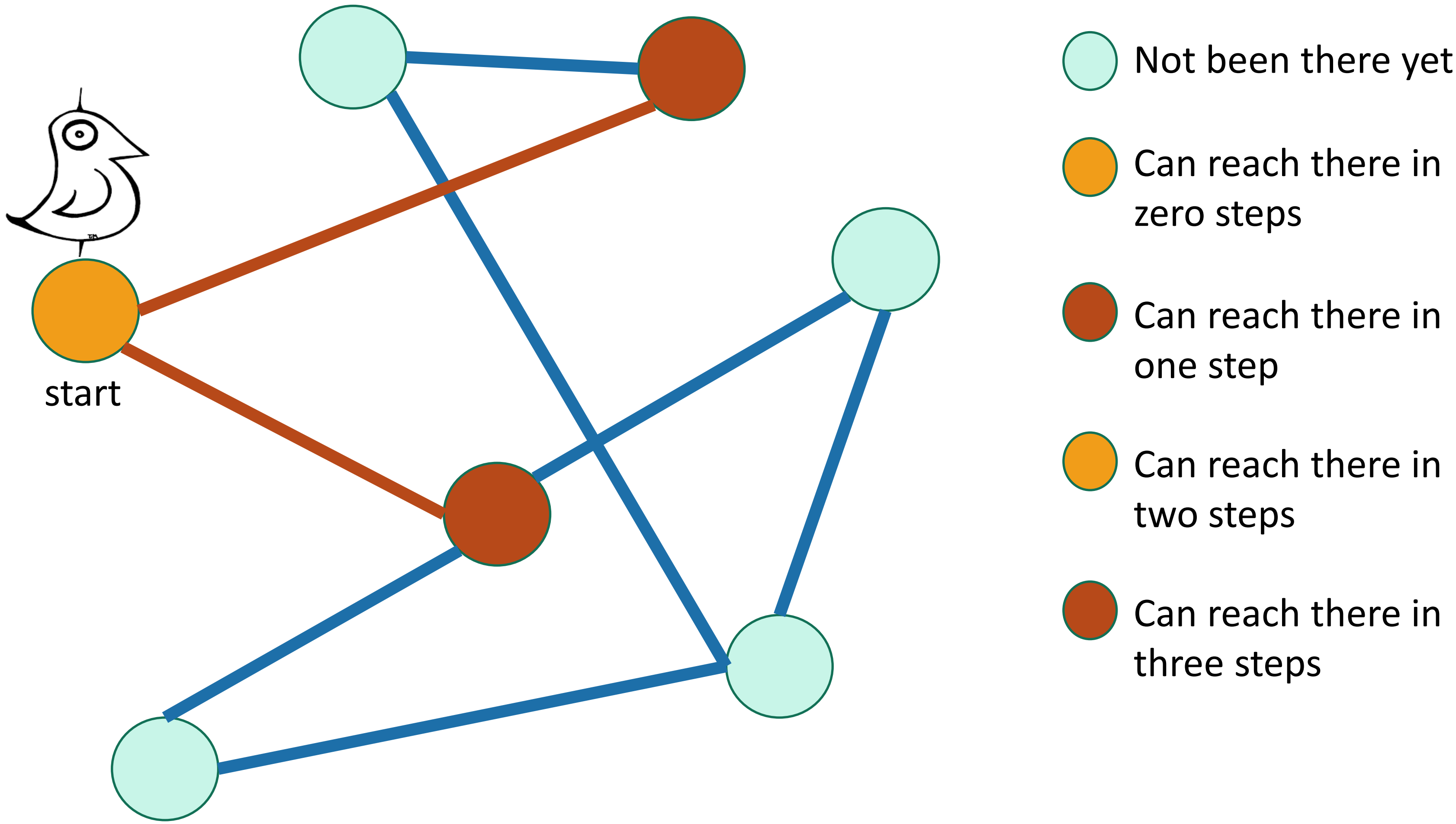
Breadth-First Search

For testing bipartite-ness



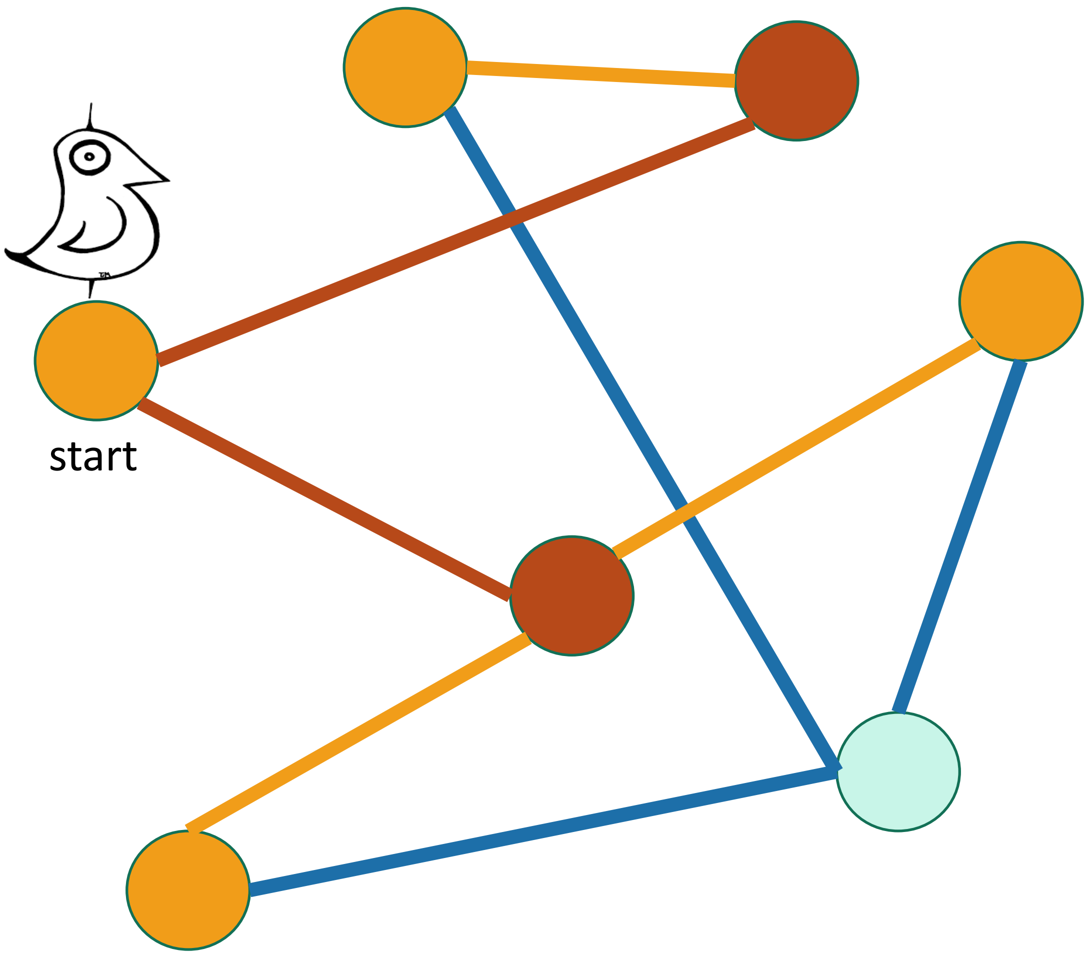
Breadth-First Search

For testing bipartite-ness



Breadth-First Search

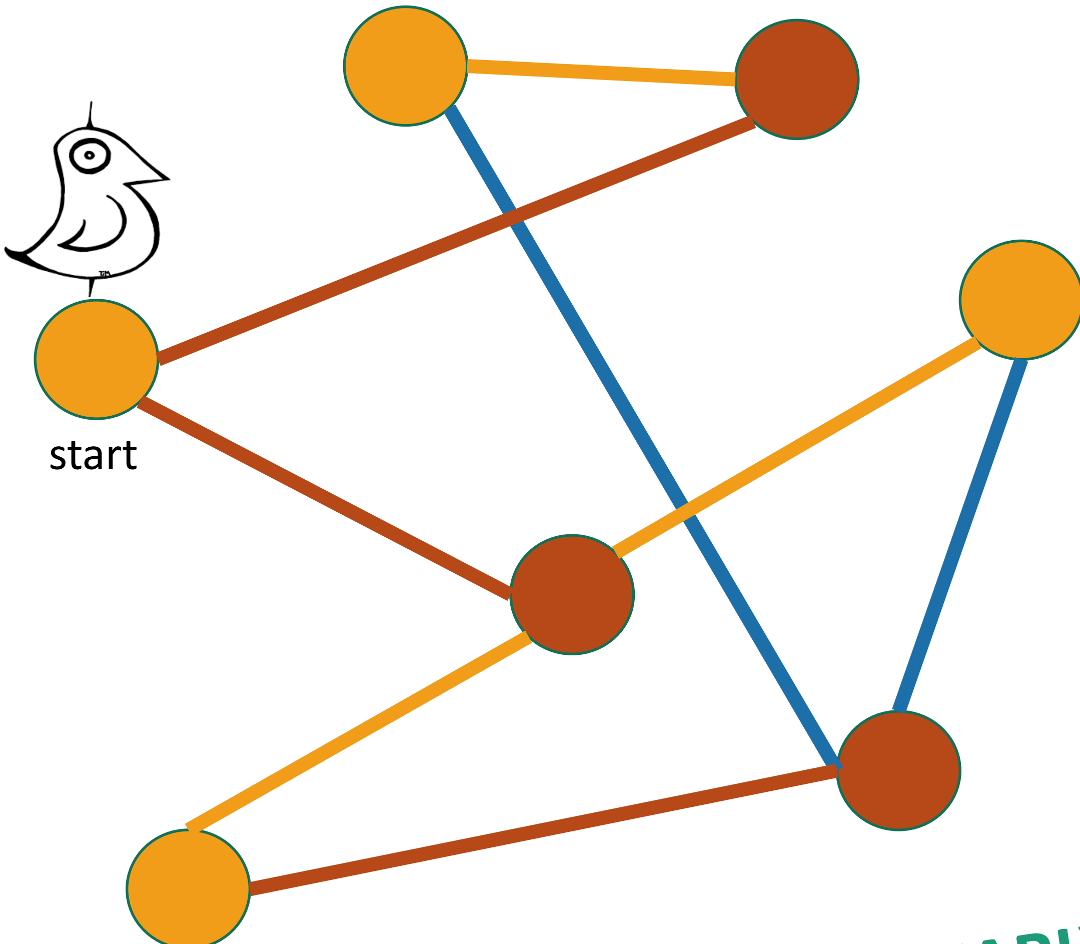
For testing bipartite-ness



- Not been there yet
- Can reach there in zero steps
- Can reach there in one step
- Can reach there in two steps
- Can reach there in three steps

Breadth-First Search

For testing bipartite-ness



Not been there yet

Can reach there in zero steps

Can reach there in one step

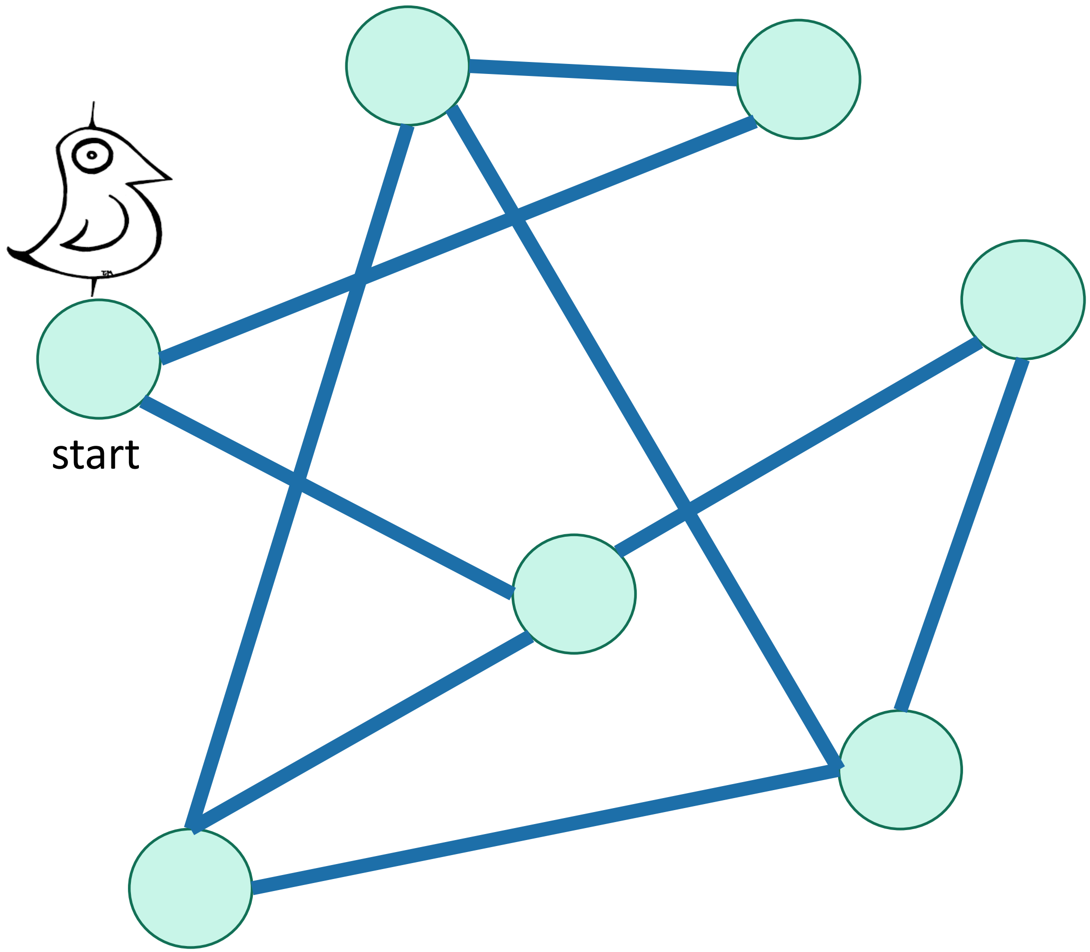
Can reach there in two steps

Can reach there in three steps

CLEARLY BIPARTITE!

Breadth-First Search

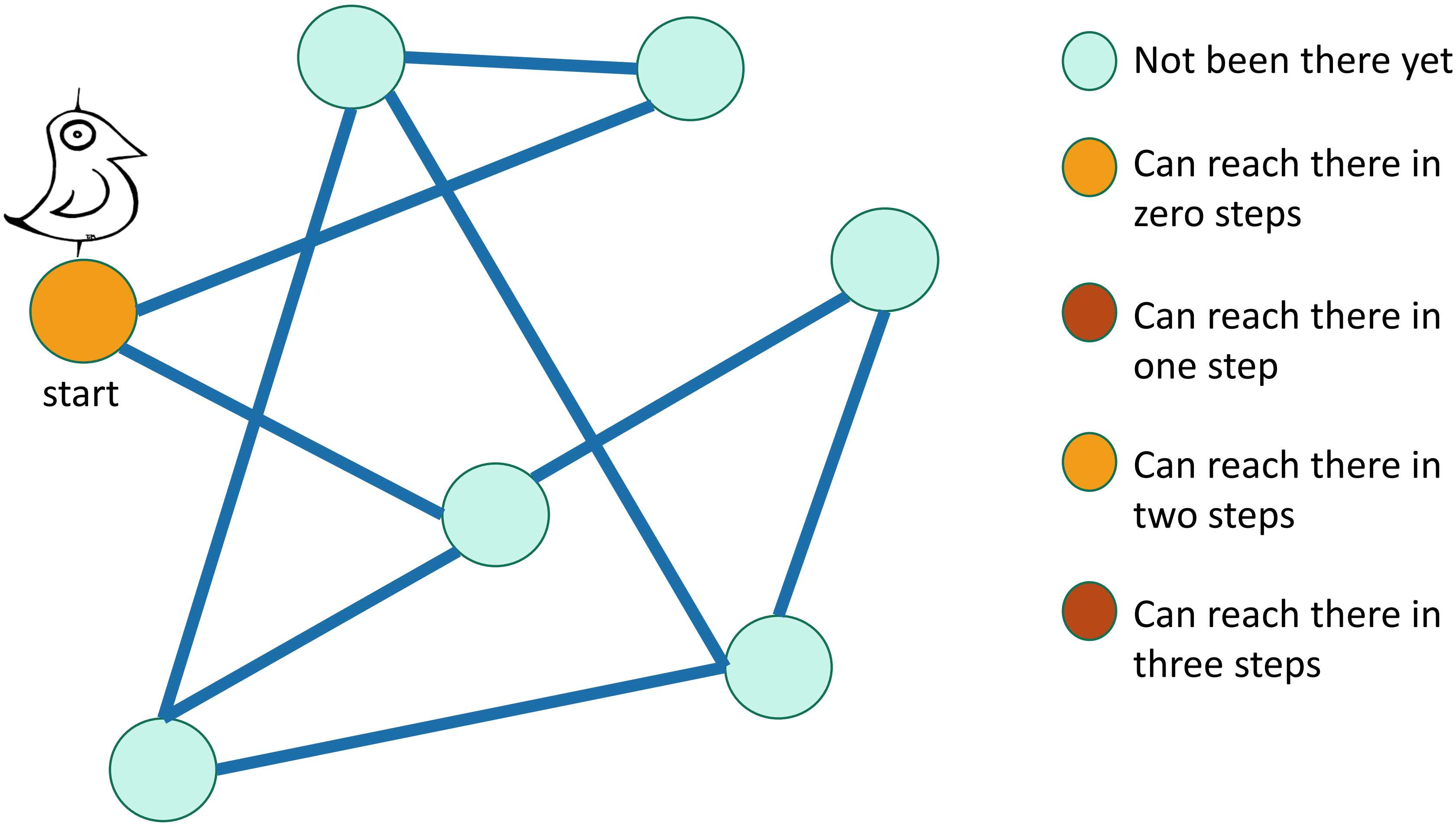
For testing bipartite-ness



- Not been there yet
- Can reach there in zero steps
- Can reach there in one step
- Can reach there in two steps
- Can reach there in three steps

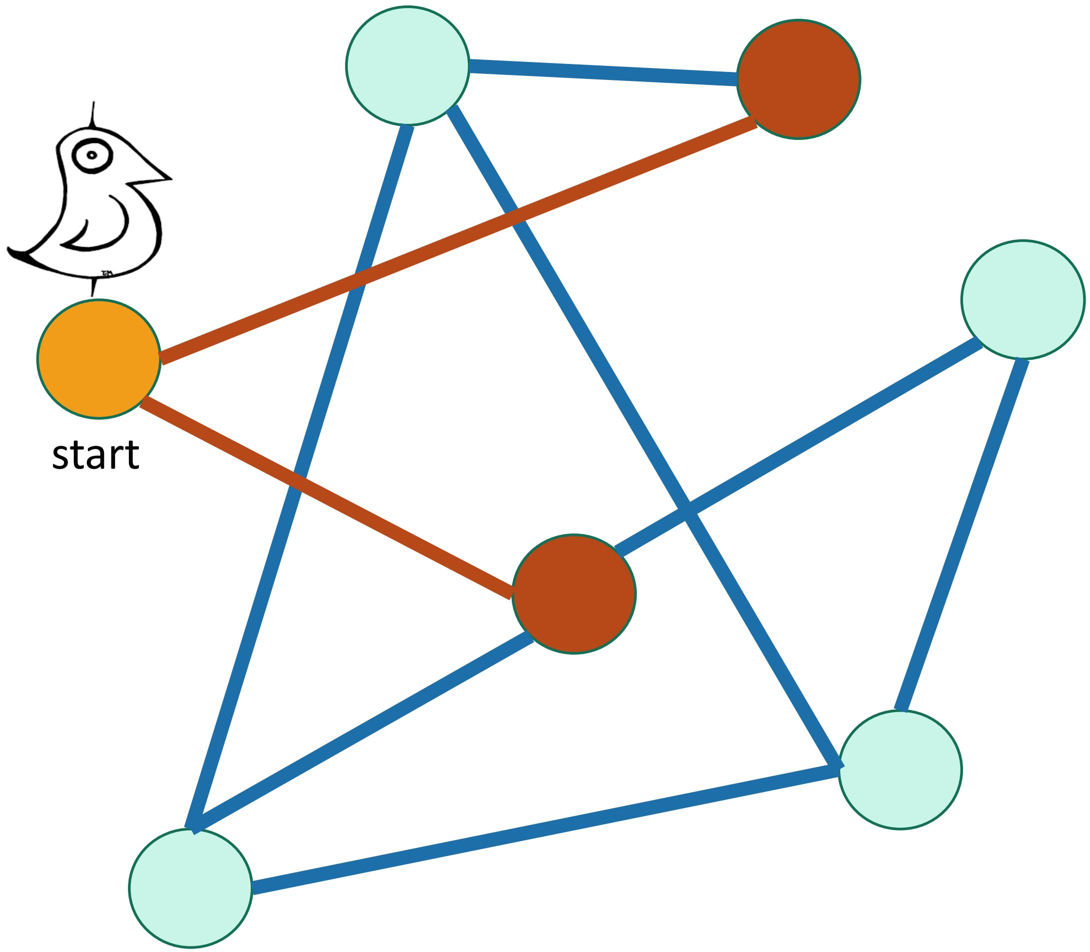
Breadth-First Search

For testing bipartite-ness



Breadth-First Search

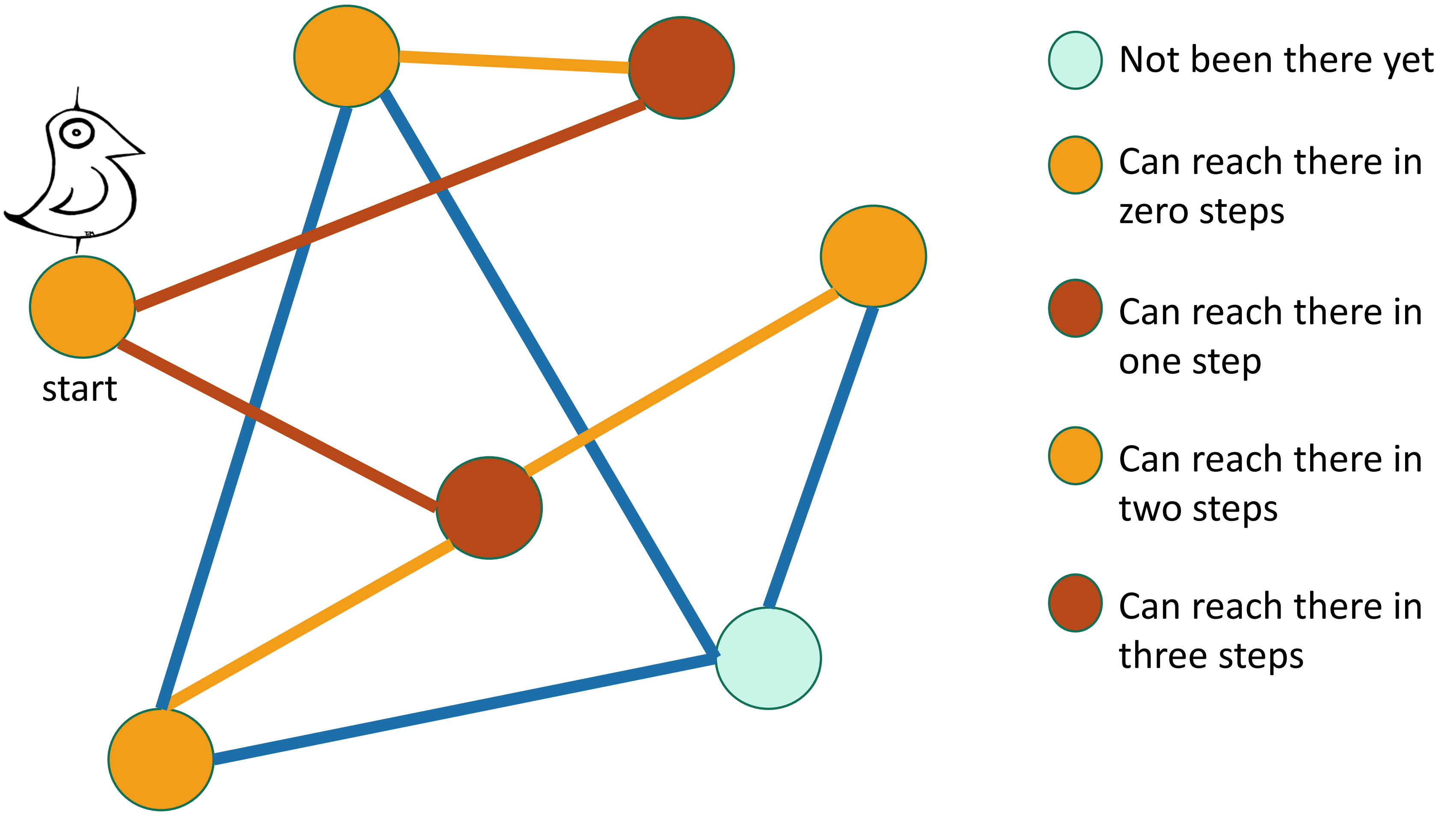
For testing bipartite-ness



- Not been there yet
- Can reach there in zero steps
- Can reach there in one step
- Can reach there in two steps
- Can reach there in three steps

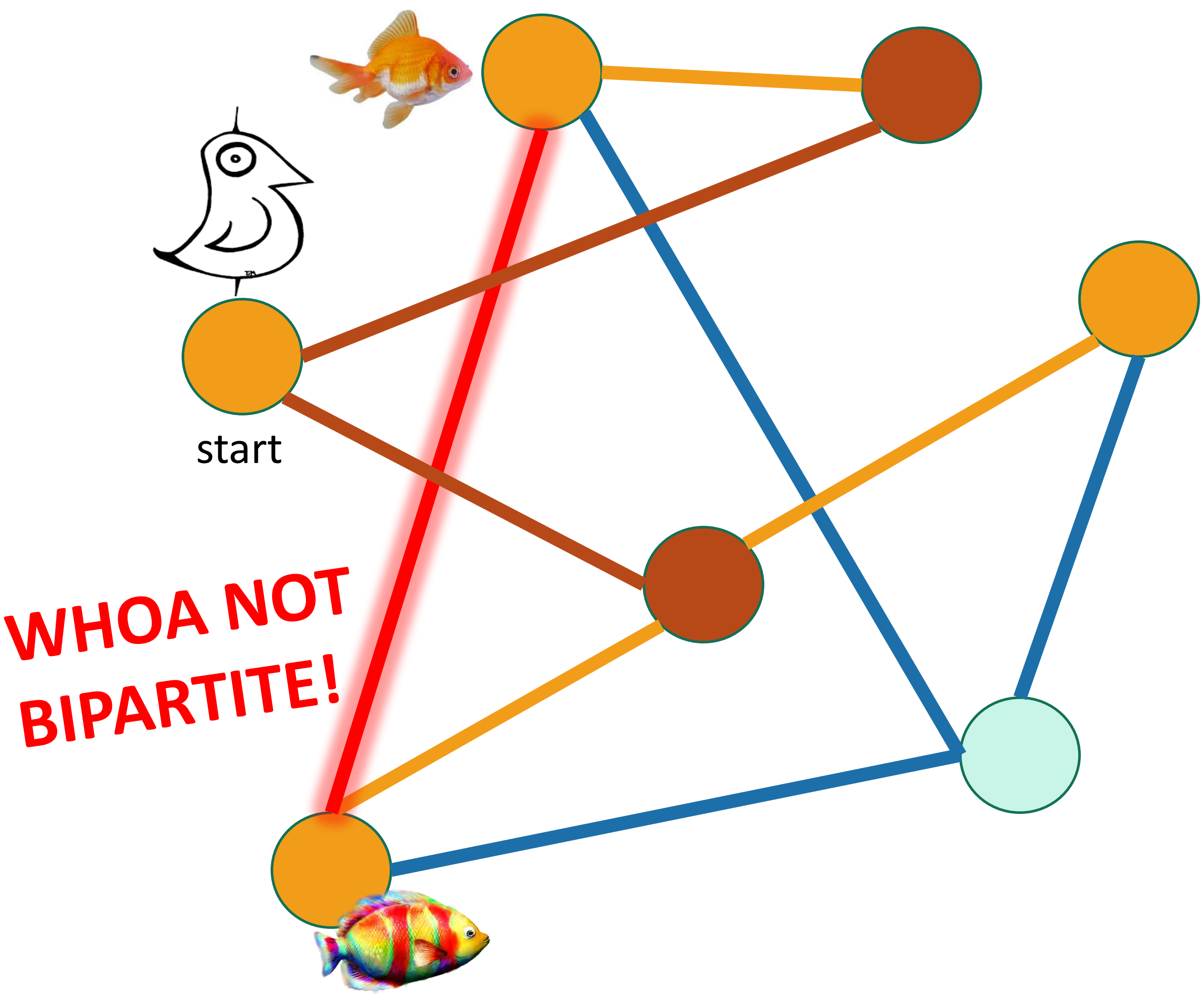
Breadth-First Search

For testing bipartite-ness



Breadth-First Search

For testing bipartite-ness



Not been there yet

Can reach there in zero steps

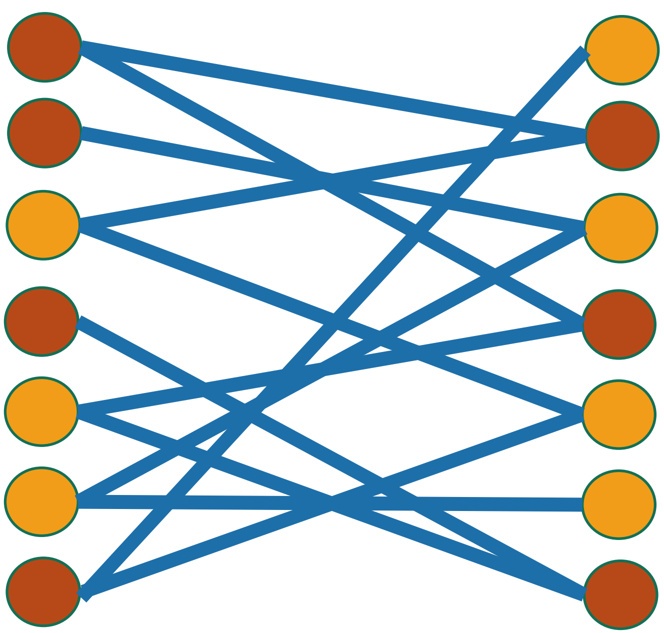
Can reach there in one step

Can reach there in two steps

Can reach there in three steps

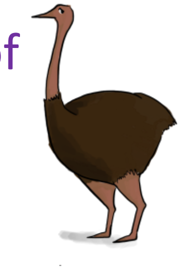
Hang on now.

- Just because **this** coloring doesn't work, why does that mean that there is **no** coloring that works?

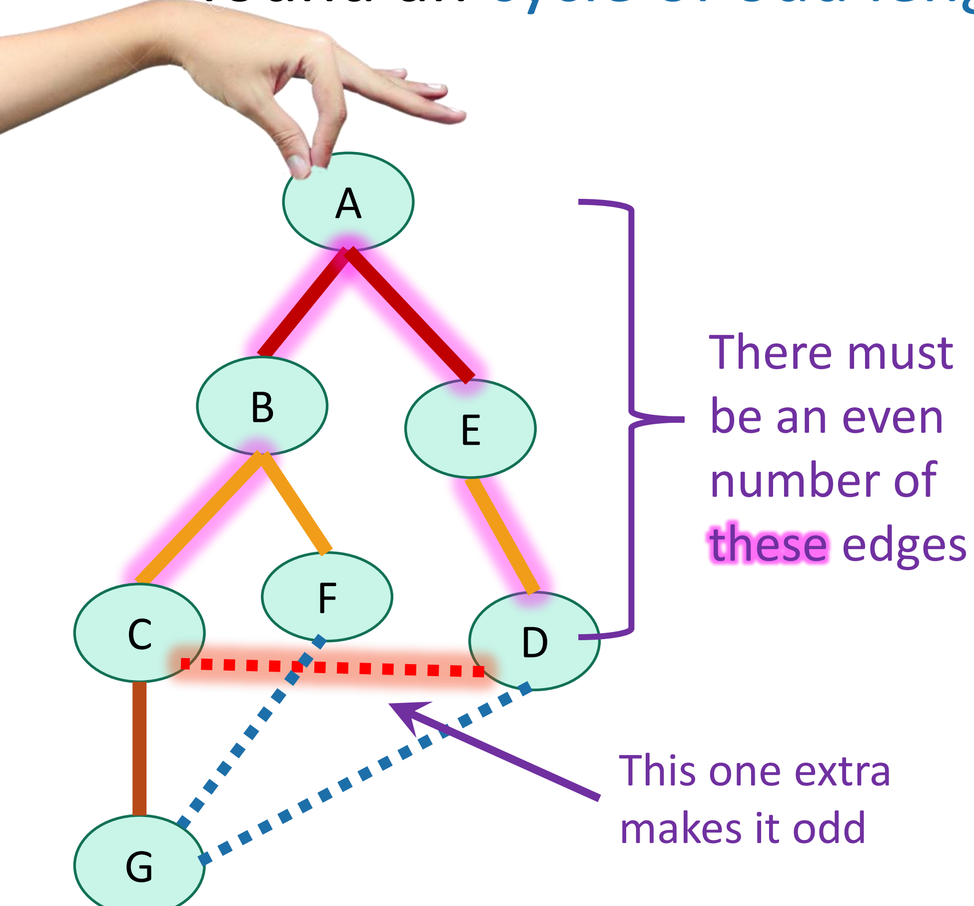


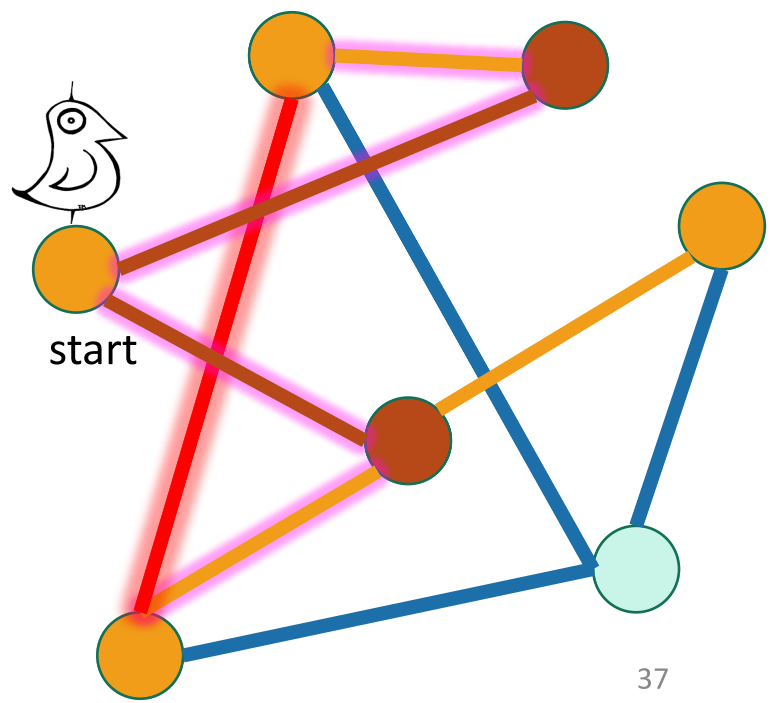
I can come up with plenty of bad colorings on this legitimately bipartite graph…

Some proof required



- If BFS colors two neighbors the same color, then it's found an cycle of odd length in the graph.

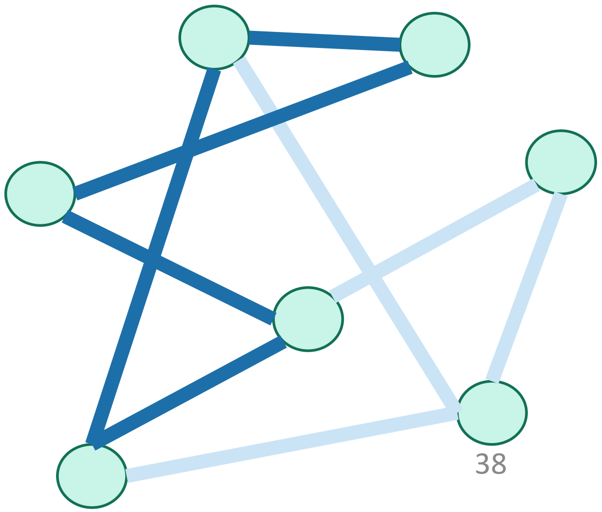




Make this proof sketch formal!

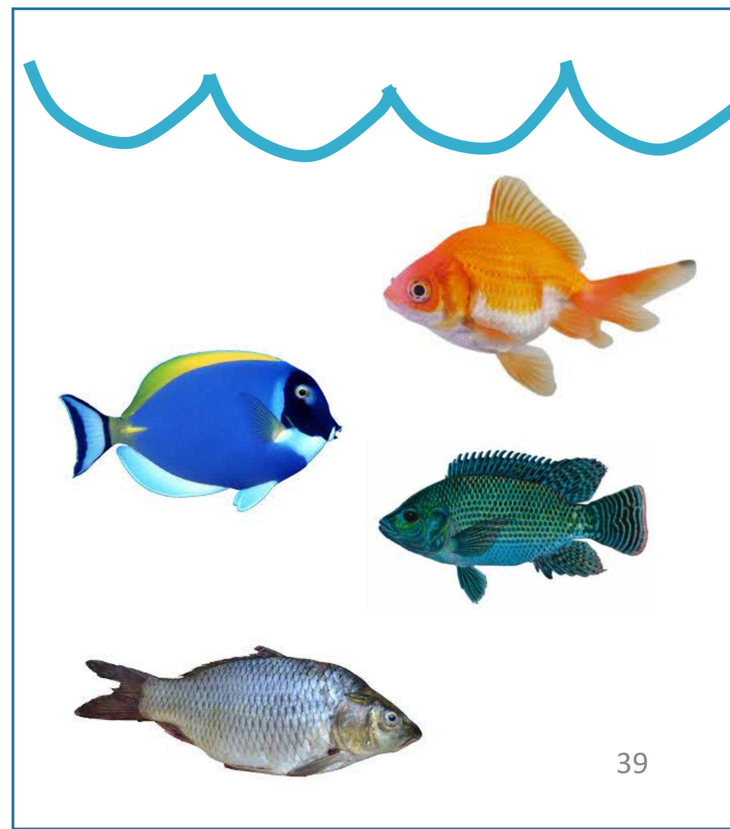
Some proof required

- If BFS colors two neighbors the same color, then it's found an cycle of odd length in the graph.
- But you can **never** color an odd cycle with two colors so that no two neighbors have the same color.
  - [Fun exercise!]
- So you can't legitimately color the whole graph either.
- **Thus it's not bipartite.**



What have we learned?

BFS can be used to detect bipartite-ness in time $O(n + m)$.



Acknowledgement

- Stanford University