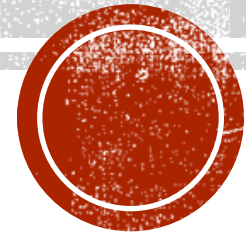




Indian Institute of Information Technology Allahabad

Data Structures and Algorithms

Breadth First Search (BFS)



Dr. Shiv Ram Dubey

Associate Professor

Department of Information Technology

Indian Institute of Information Technology, Allahabad

Email: srdubey@iiita.ac.in

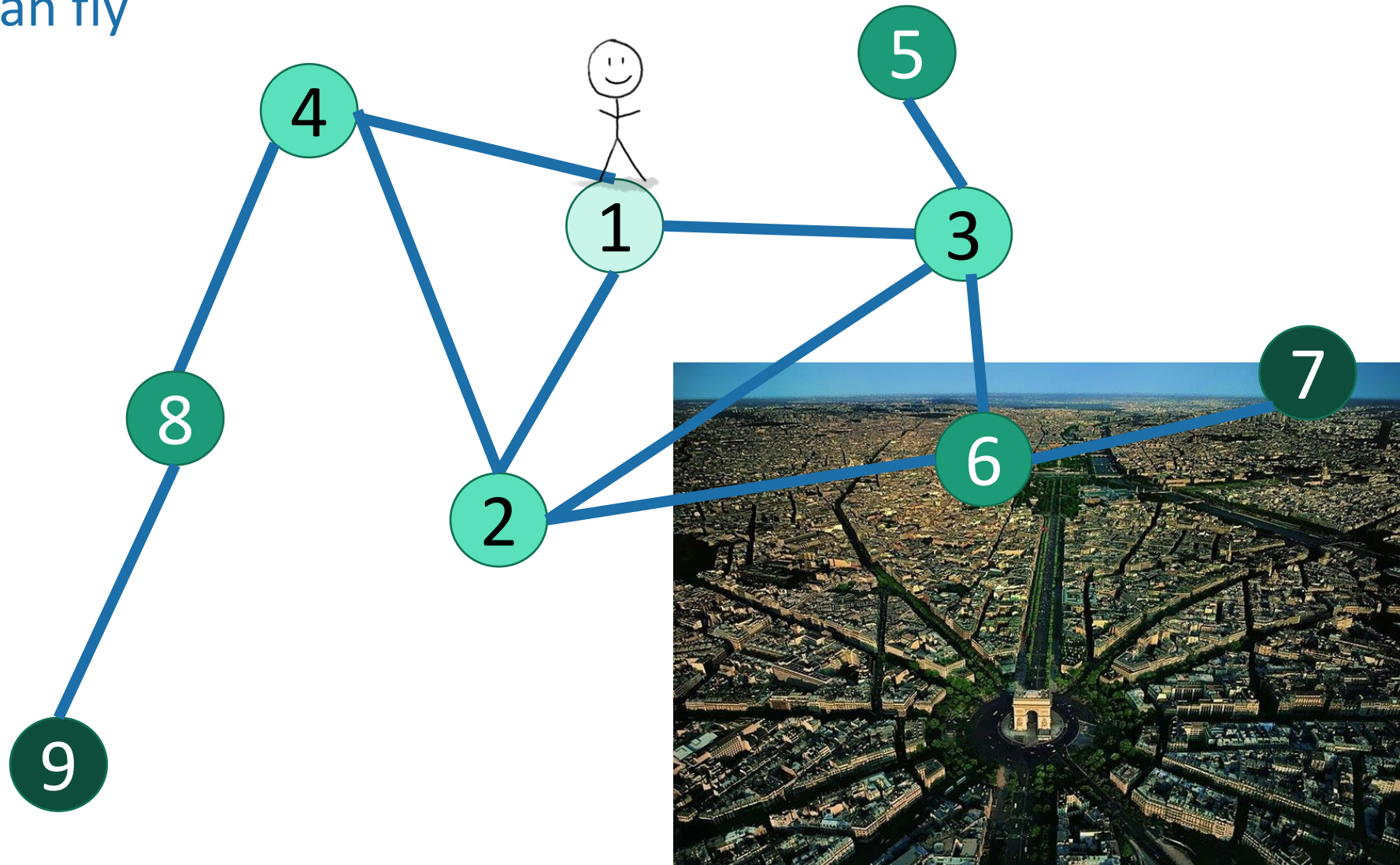
Web: <https://profile.iiita.ac.in/srdubey/>

DISCLAIMER

The content (text, image, and graphics) used in this slide are adopted from many sources for academic purposes. Broadly, the sources have been given due credit appropriately. However, there is a chance of missing out some original primary sources. The authors of this material do not claim any copyright of such material.

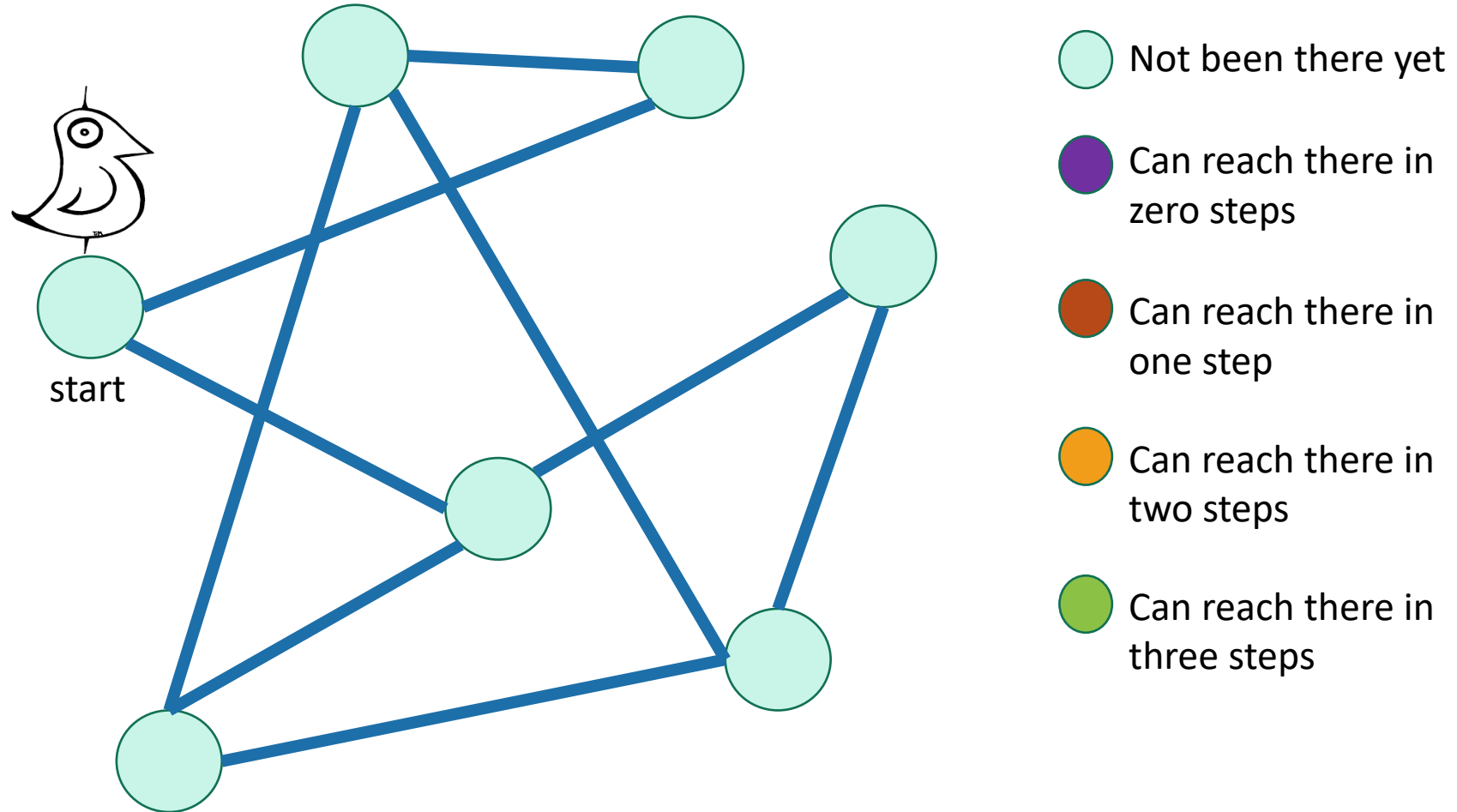
How do we explore a graph?

If we can fly



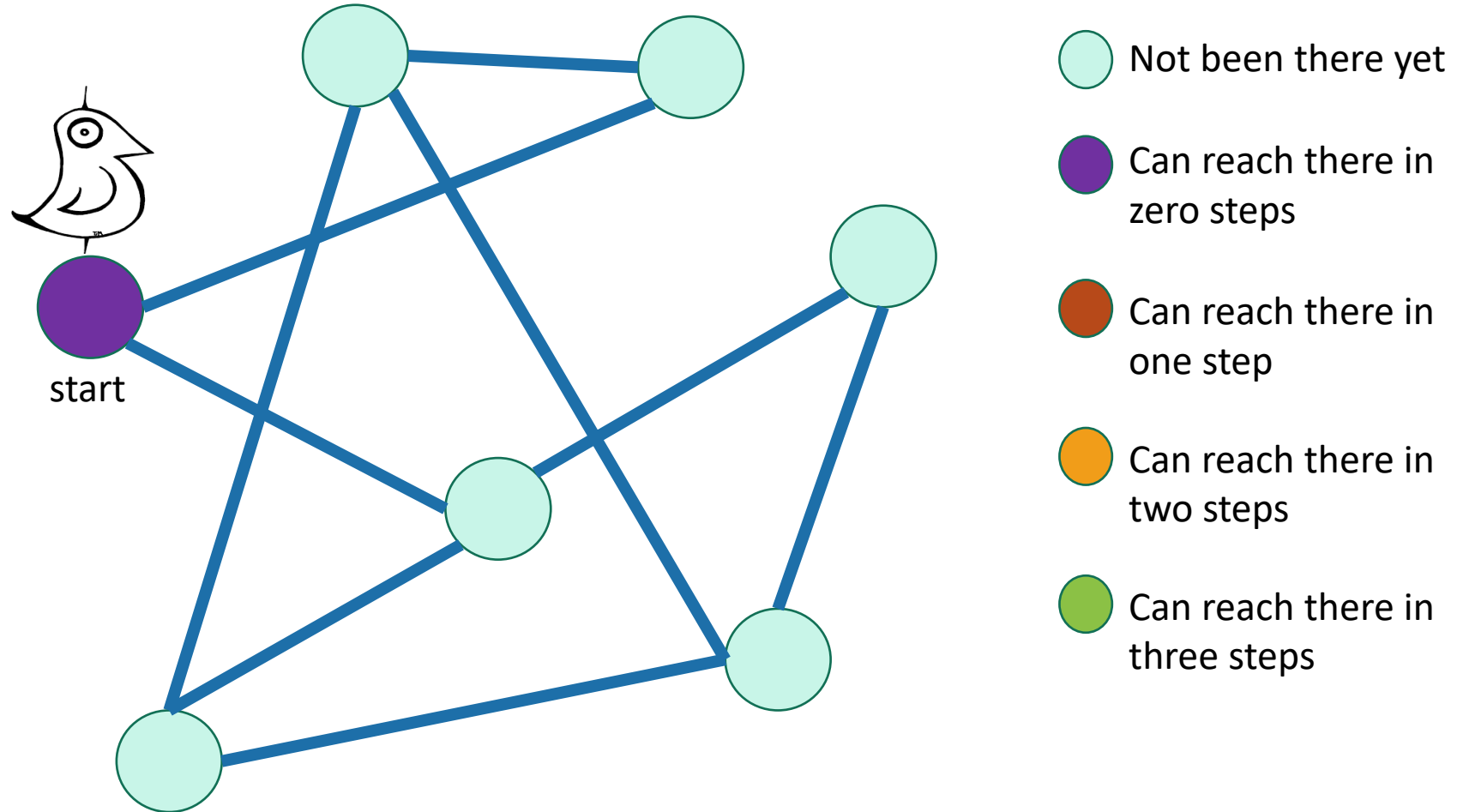
Breadth-First Search

Exploring the world with a bird's-eye view



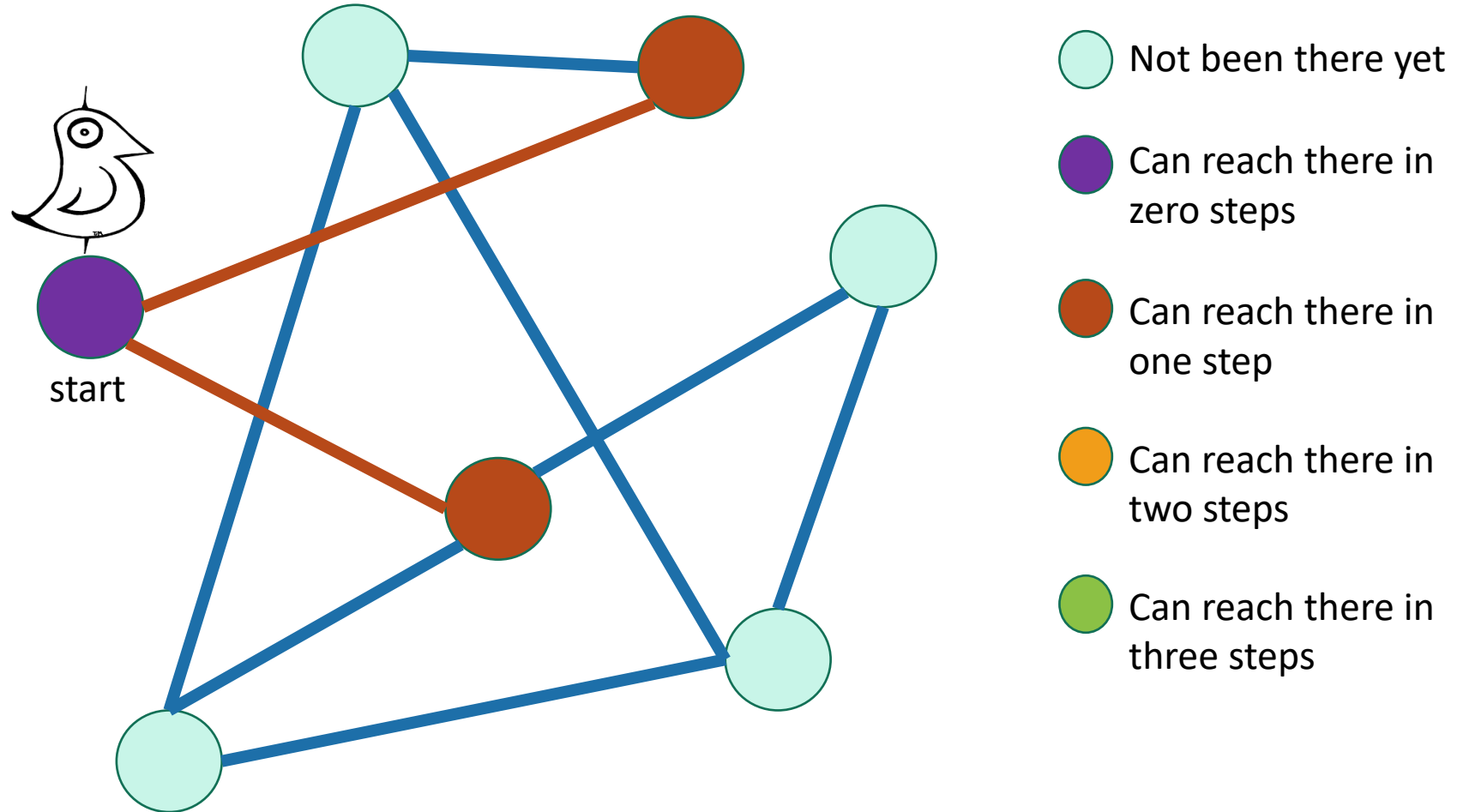
Breadth-First Search

Exploring the world with a bird's-eye view



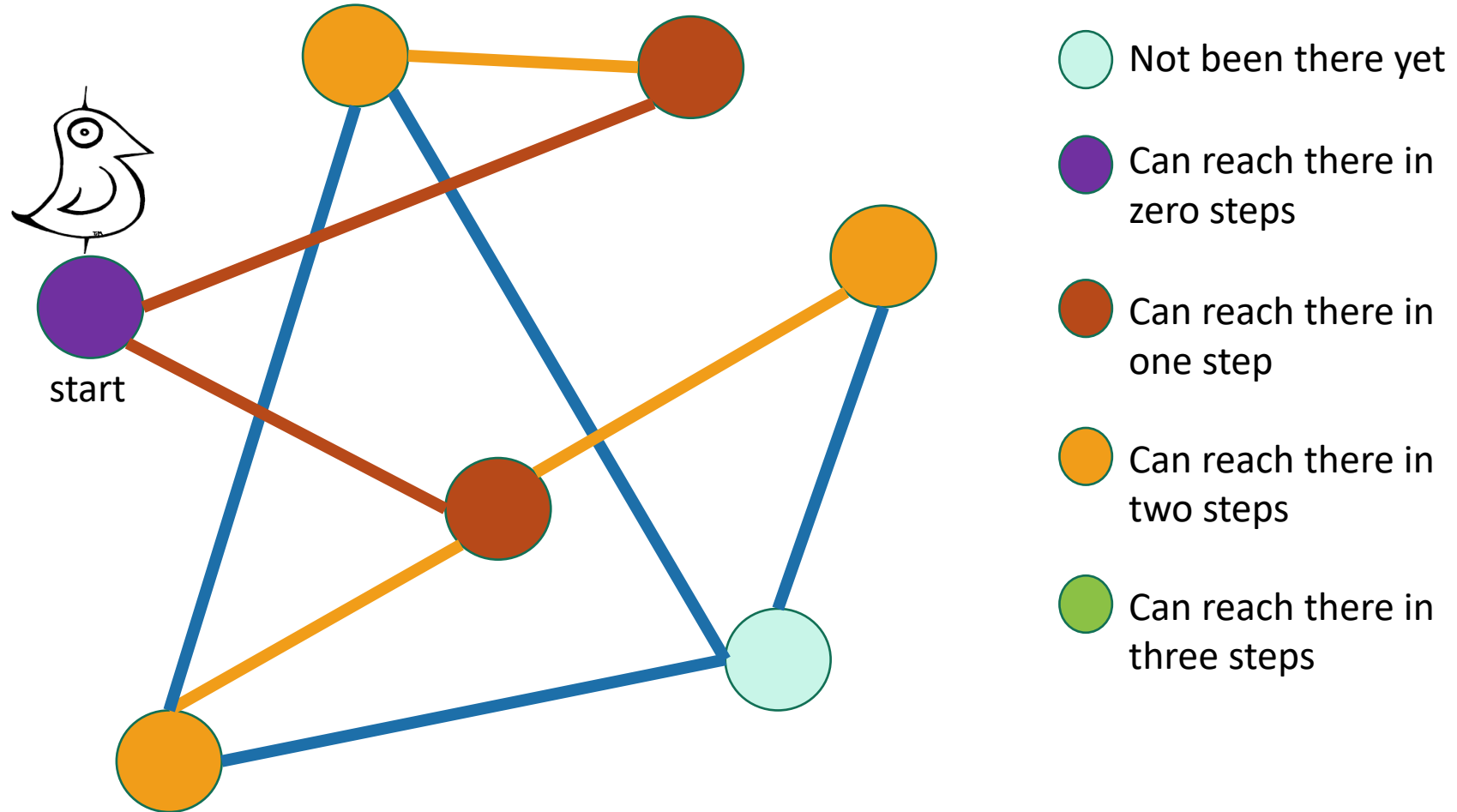
Breadth-First Search

Exploring the world with a bird's-eye view



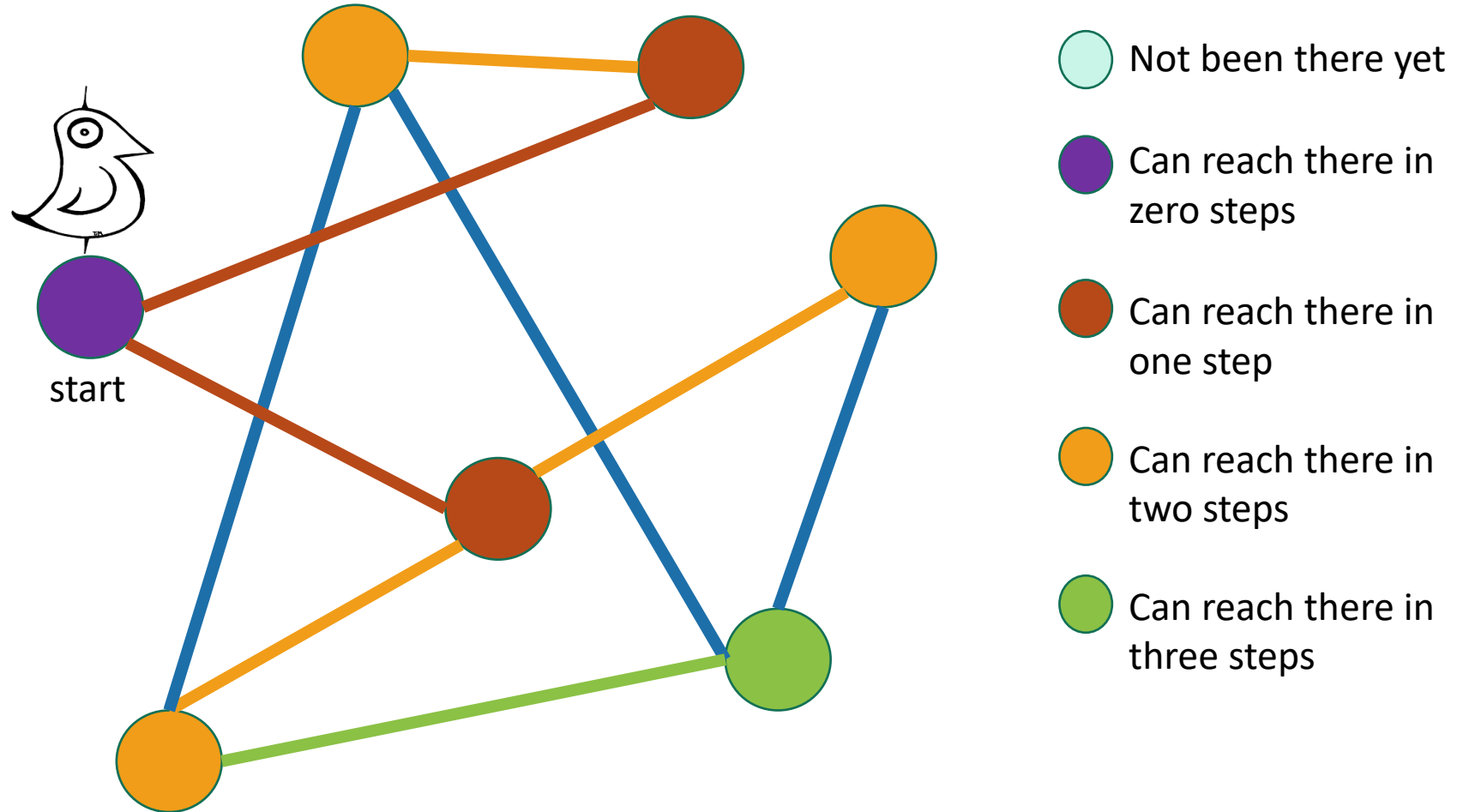
Breadth-First Search

Exploring the world with a bird's-eye view



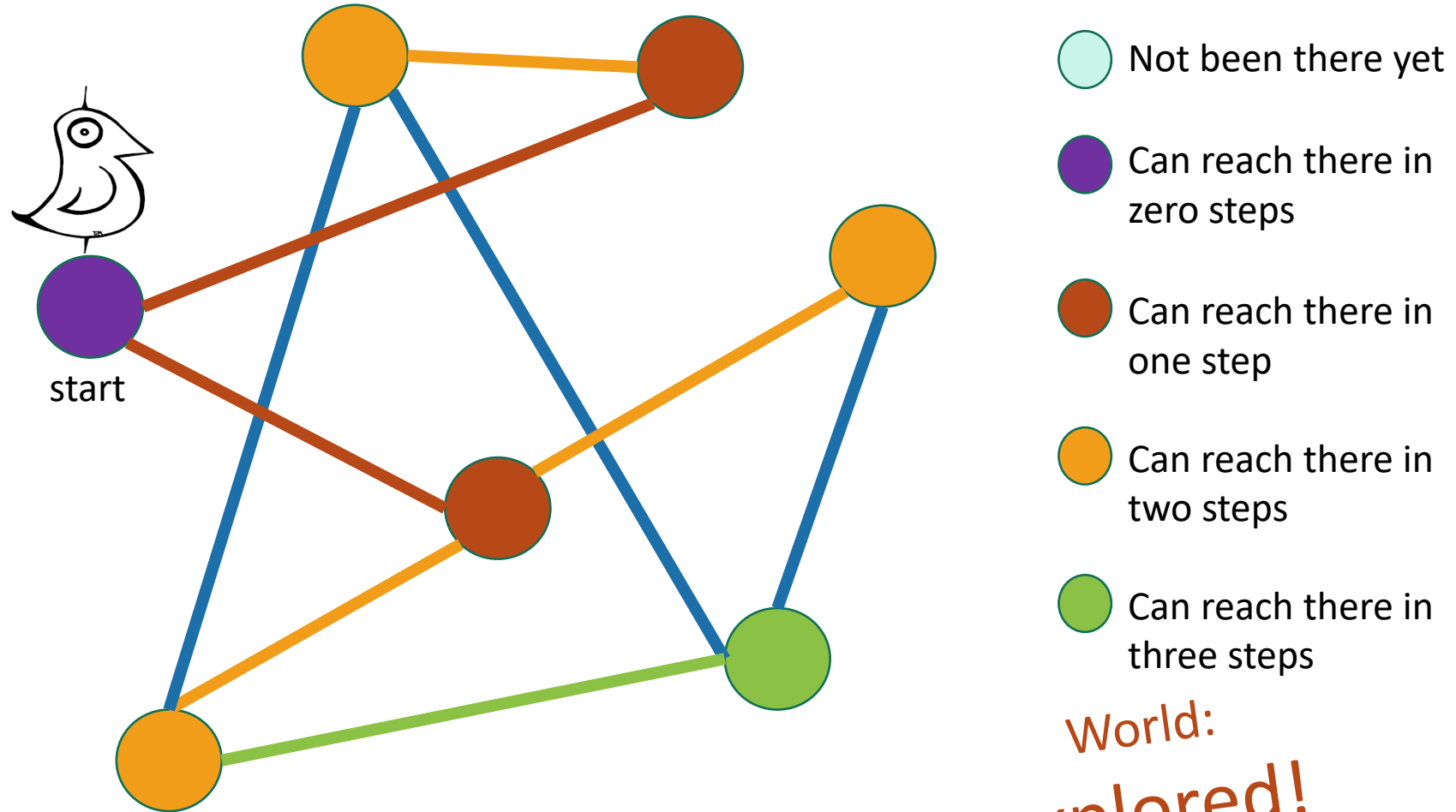
Breadth-First Search

Exploring the world with a bird's-eye view



Breadth-First Search

Exploring the world with a bird's-eye view

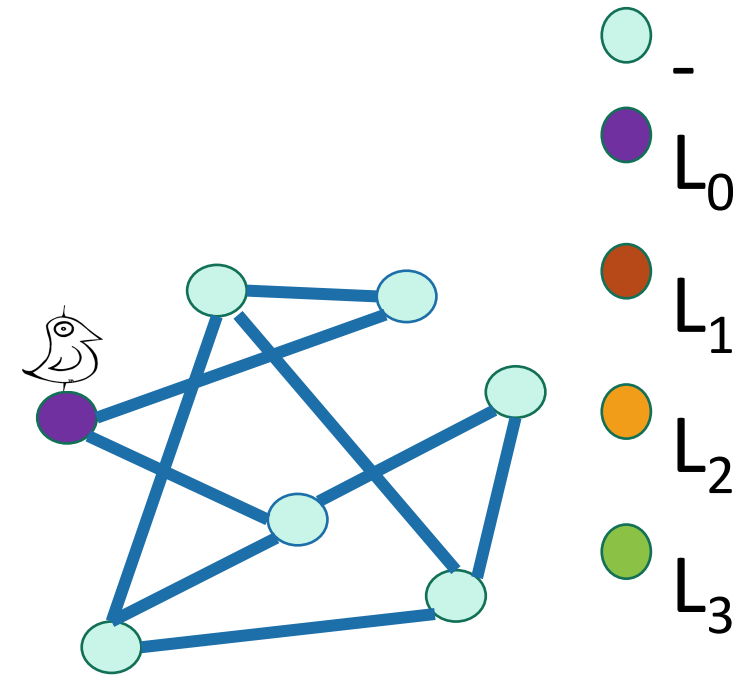


World:
explored!

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



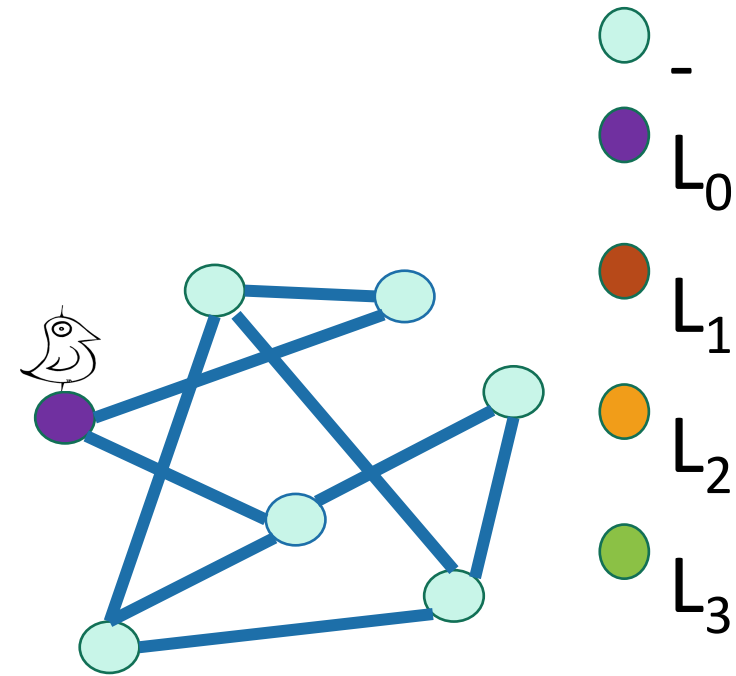
Same disclaimer as for DFS: you may have seen other ways to implement this, this will be convenient for us.

Breadth-First Search

Exploring the world with pseudocode

- Set $L_i = []$ for $i=1, \dots, n$
- $L_0 = [w]$, where w is the start node

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



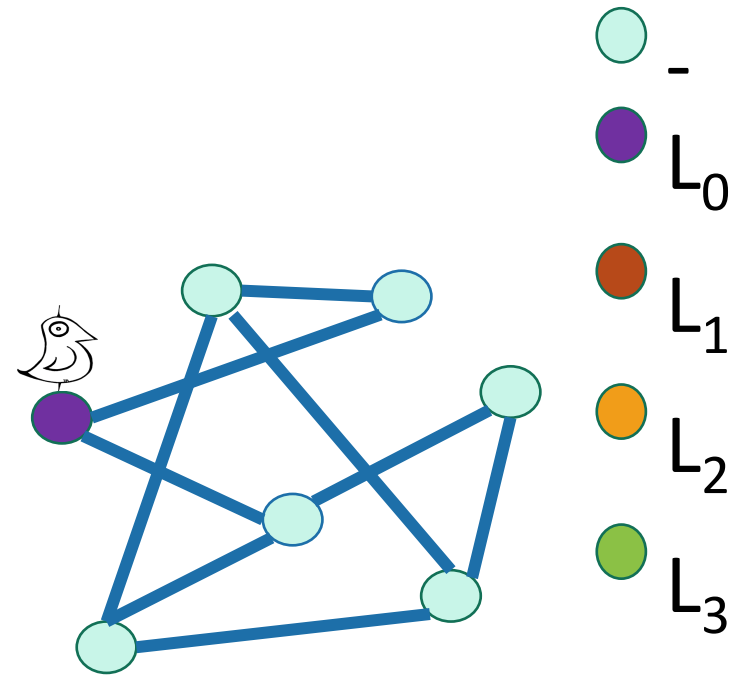
Same disclaimer as for DFS: you may have seen other ways to implement this, this will be convenient for us.

Breadth-First Search

Exploring the world with pseudocode

- Set $L_i = []$ for $i=1, \dots, n$
- $L_0 = [w]$, where w is the start node
- Mark w as visited

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



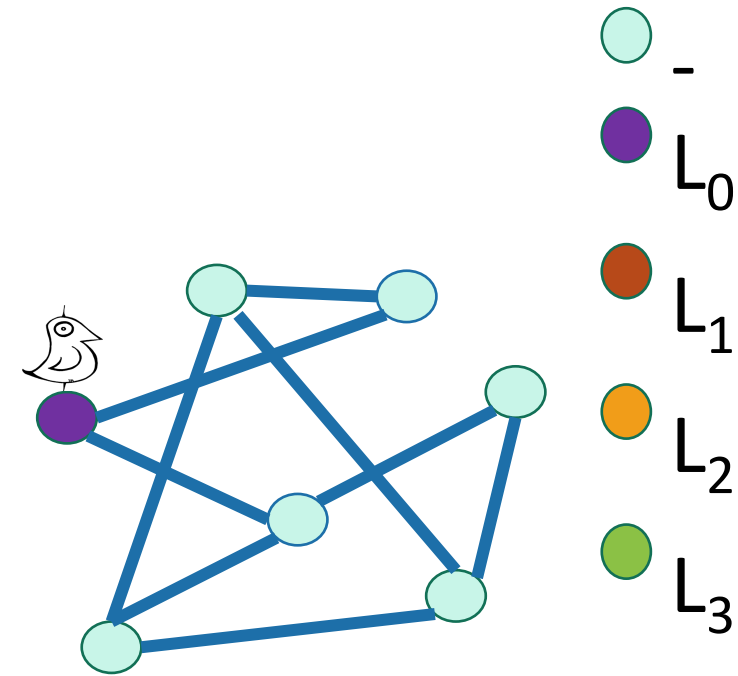
Same disclaimer as for DFS: you may have seen other ways to implement this, this will be convenient for us.

Breadth-First Search

Exploring the world with pseudocode

- Set $L_i = []$ for $i=1, \dots, n$
- $L_0 = [w]$, where w is the start node
- Mark w as visited
- **For** $i = 0, \dots, n-1$:

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



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

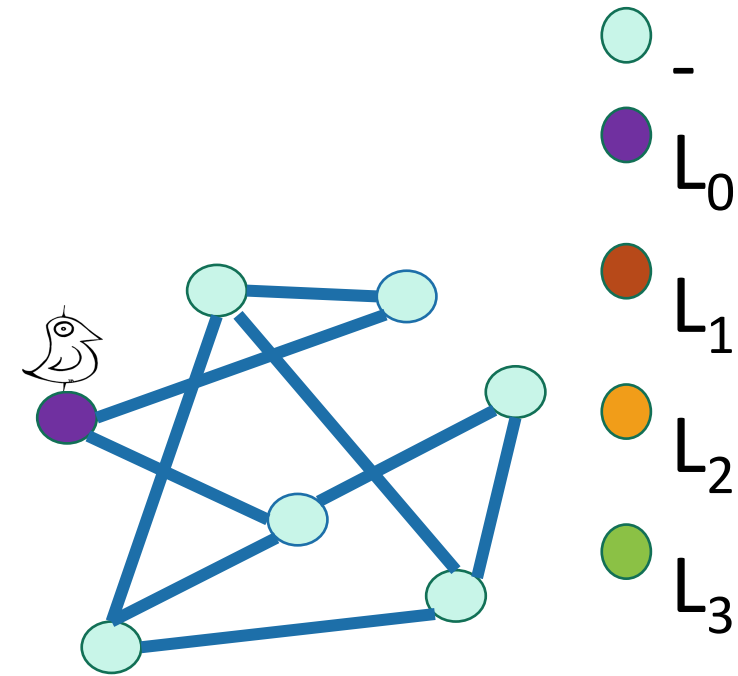
Breadth-First Search

Exploring the world with pseudocode

- Set $L_i = []$ for $i=1, \dots, n$
- $L_0 = [w]$, where w is the start node
- Mark w as visited
- **For** $i = 0, \dots, n-1$:

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}



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

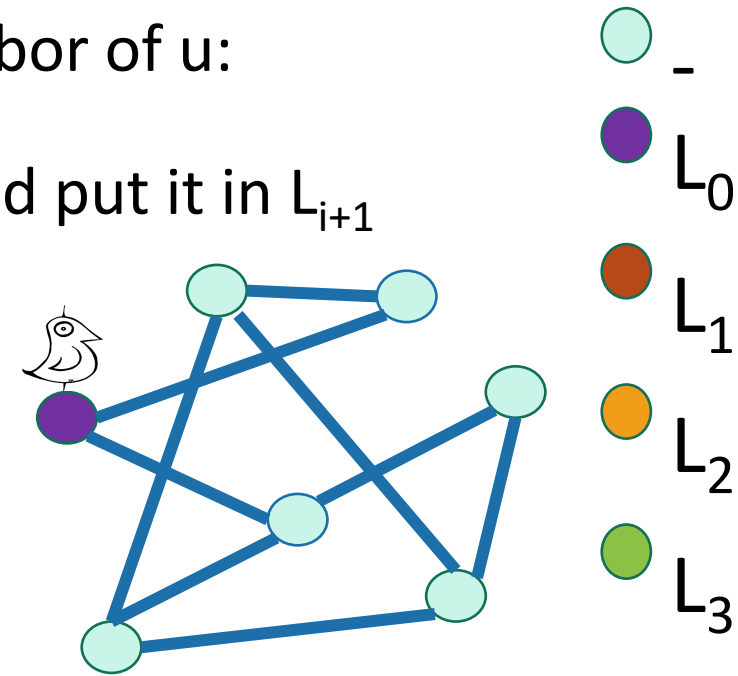
Breadth-First Search

Exploring the world with pseudocode

- Set $L_i = []$ for $i=1, \dots, n$
- $L_0 = [w]$, where w is the start node
- Mark w as visited
- **For** $i = 0, \dots, n-1$:
 - **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}



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

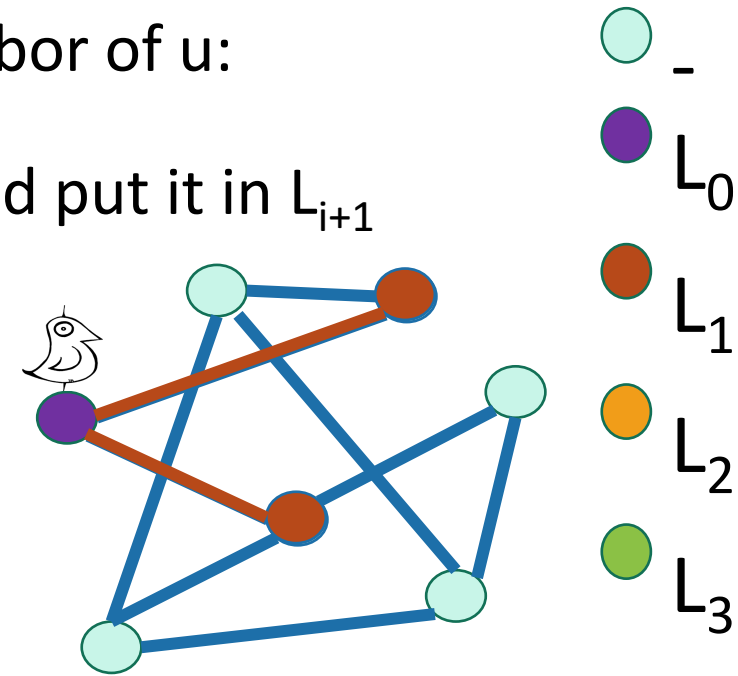
Breadth-First Search

Exploring the world with pseudocode

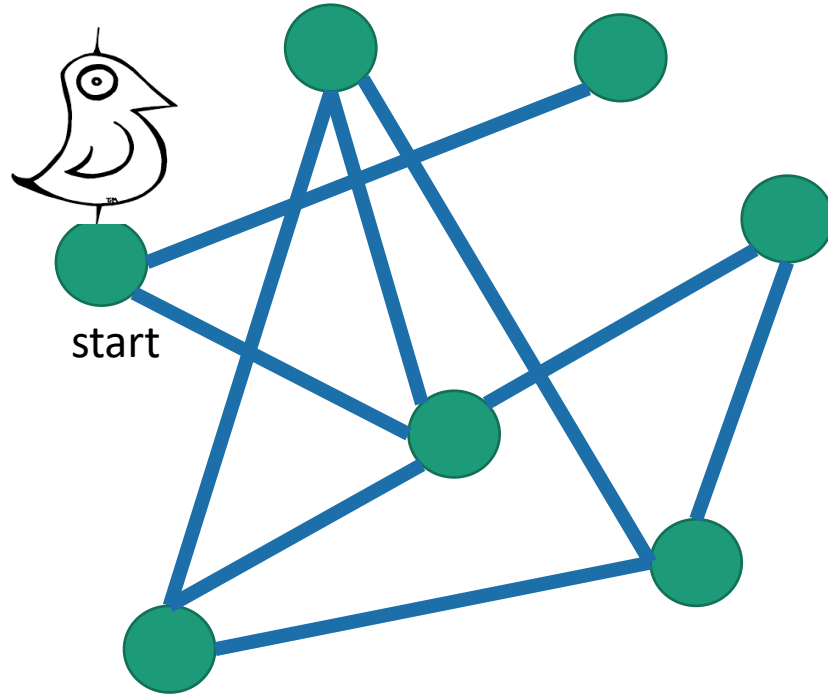
- Set $L_i = []$ for $i=1, \dots, n$
- $L_0 = [w]$, where w is the start node
- Mark w as visited
- **For** $i = 0, \dots, n-1$:
 - **For** u in L_i :
 - **For** each v which is a neighbor of u :
 - **If** v isn't yet visited:
 - mark v as visited, and put it in L_{i+1}

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

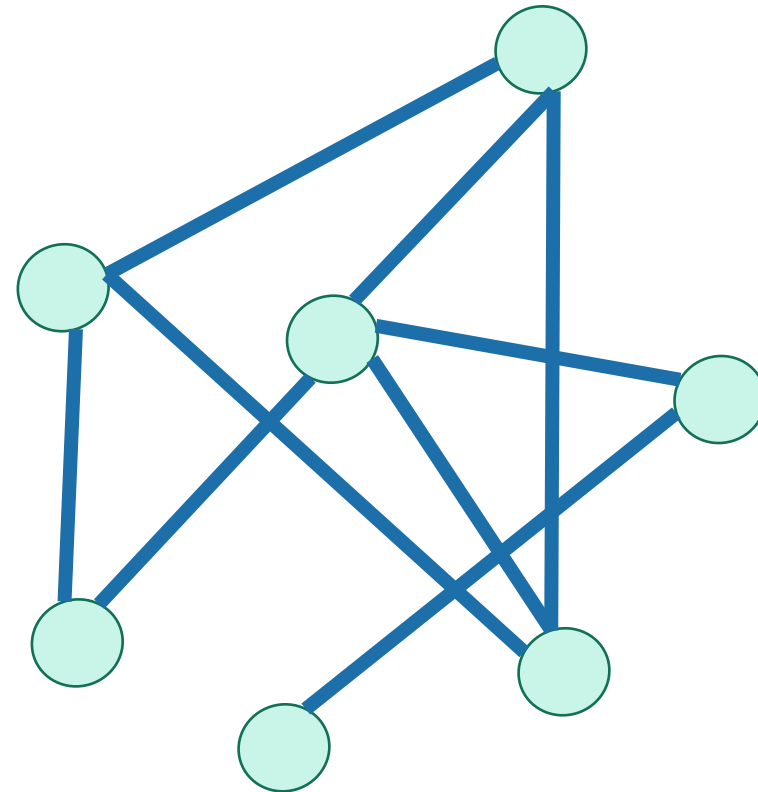
Go through all the nodes in L_i and add their unvisited neighbors to L_{i+1}



BFS also finds all the nodes reachable from the starting point



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



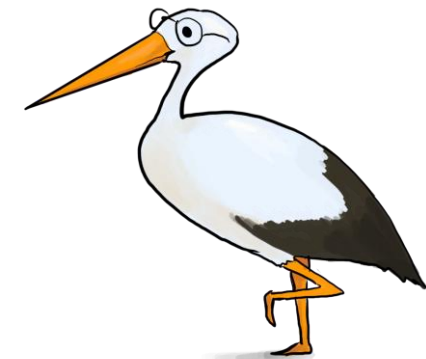
Running time and extension to directed graphs

- To explore the whole graph, explore the connected components one-by-one.
 - Same argument as DFS: BFS running time is $O(n + m)$

Running time and extension to directed graphs

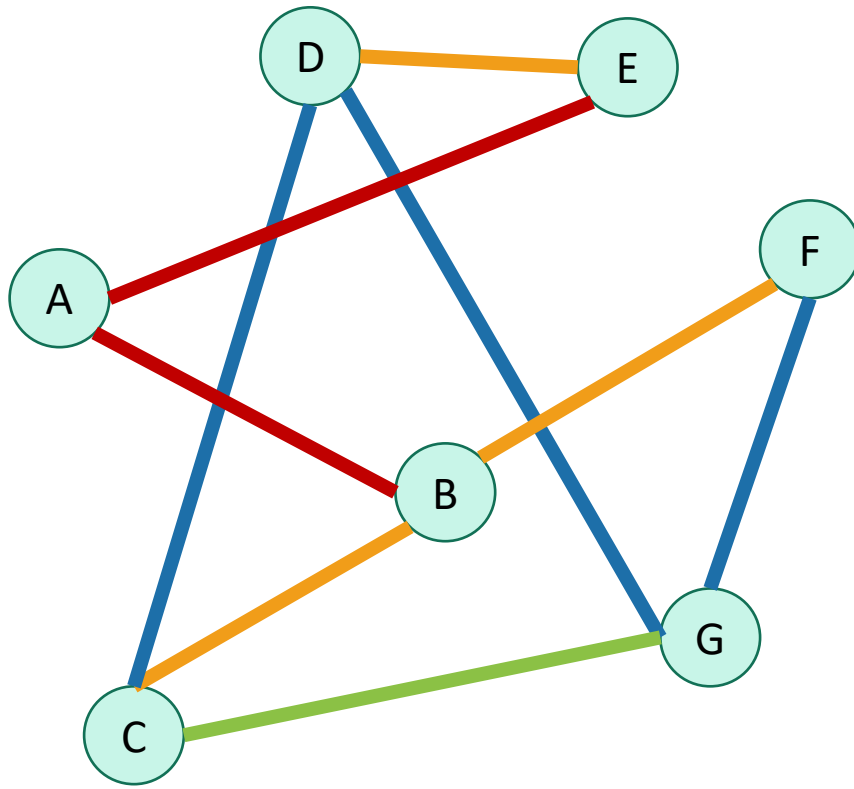
- To explore the whole graph, explore the connected components one-by-one.
 - Same argument as DFS: BFS running time is $O(n + m)$
- Like DFS, BFS also works fine on directed graphs.

Verify these!



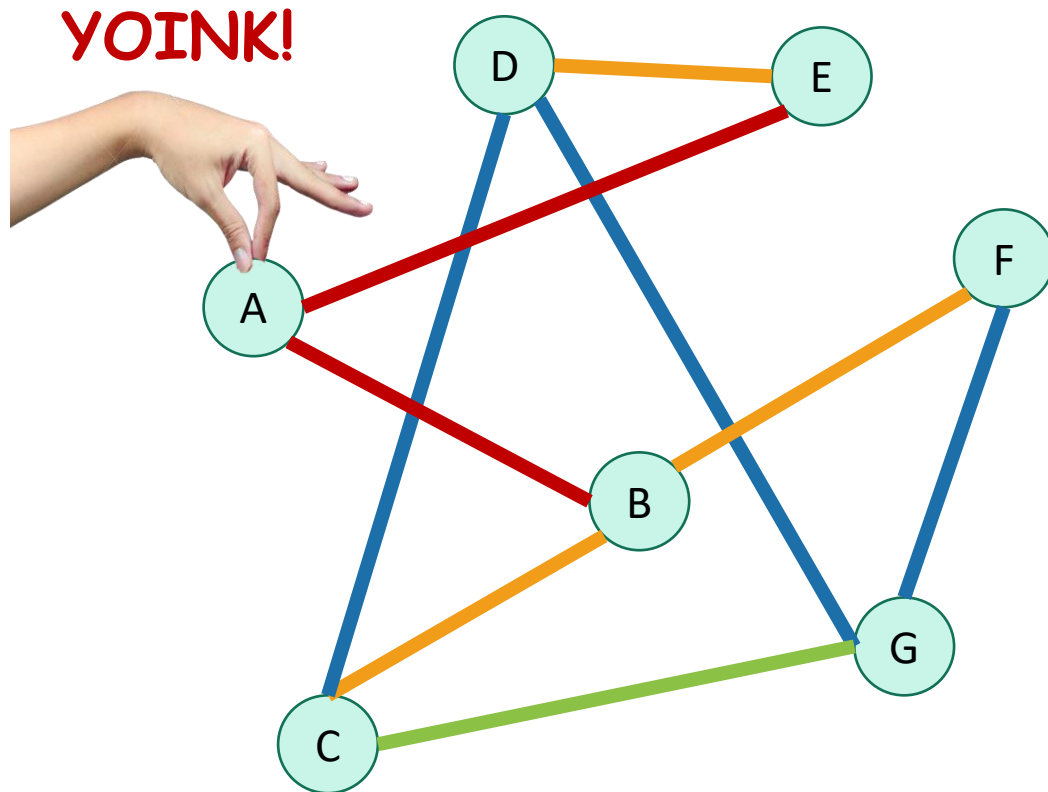
Why is it called breadth-first?

- We are implicitly building a tree:



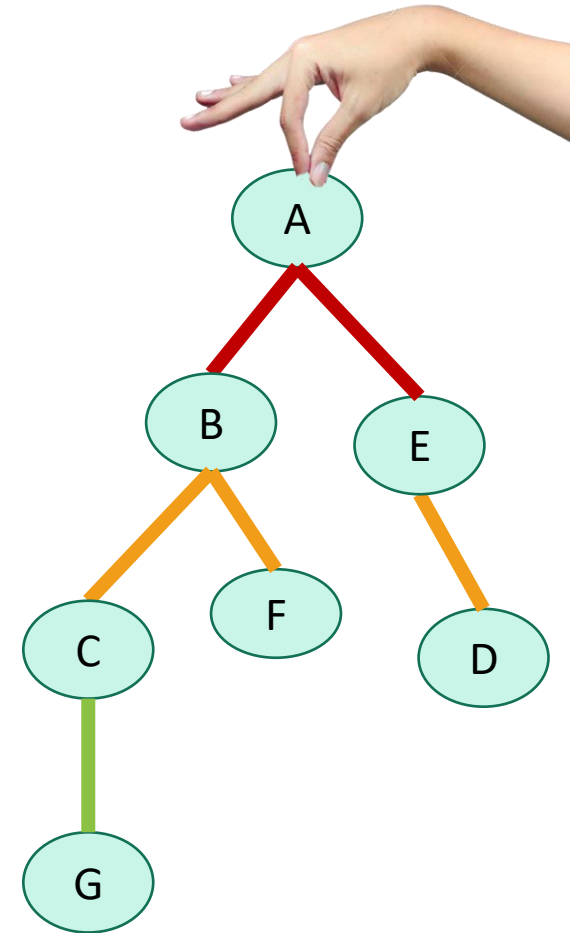
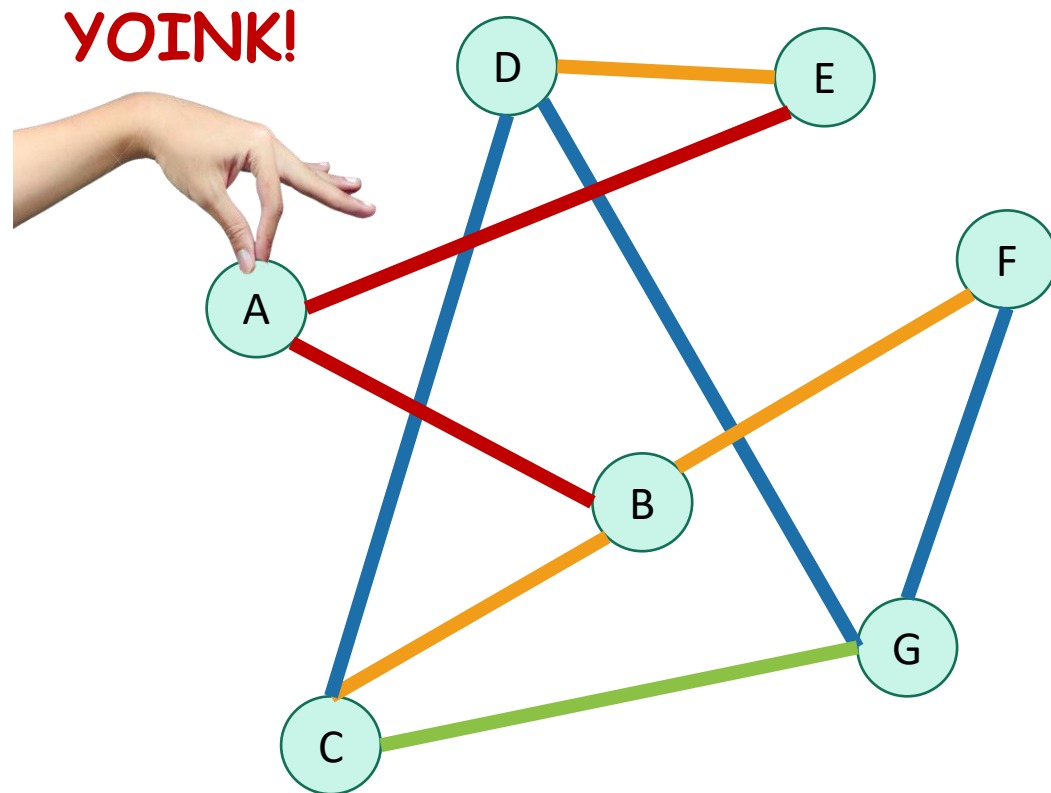
Why is it called breadth-first?

- We are implicitly building a tree:



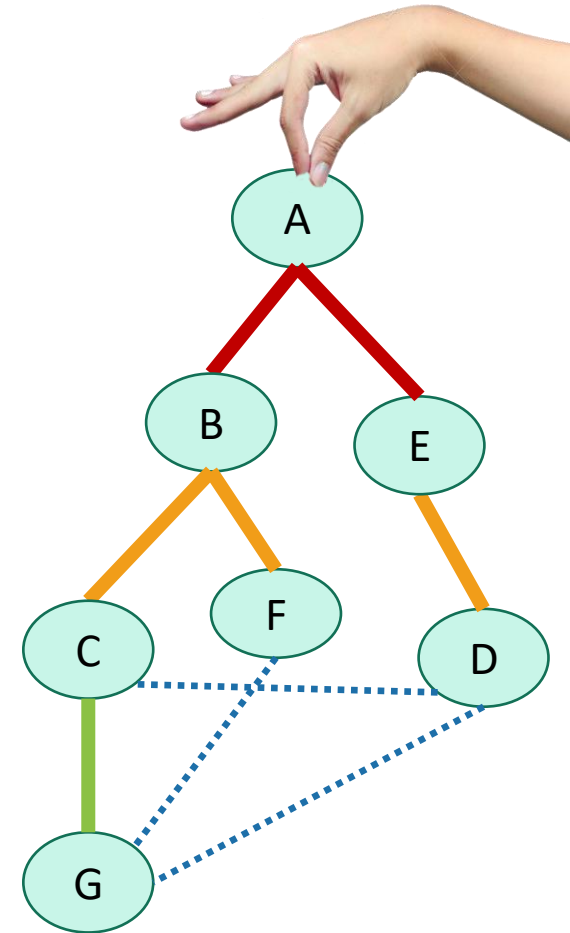
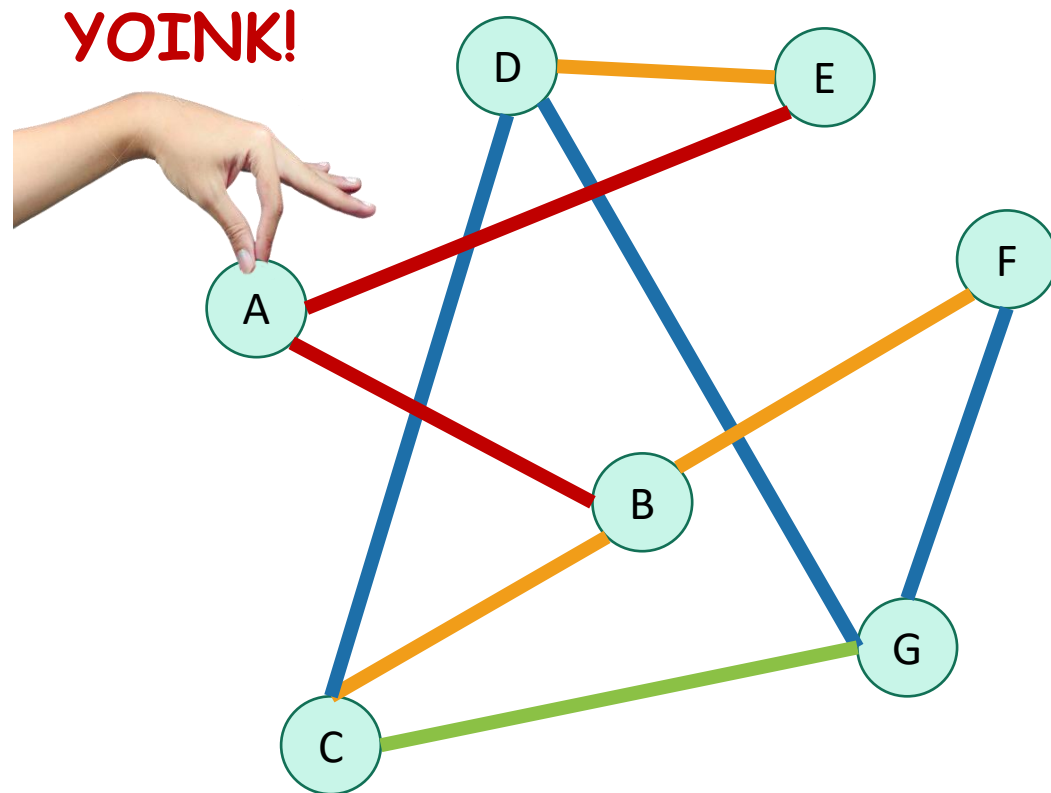
Why is it called breadth-first?

- We are implicitly building a tree:



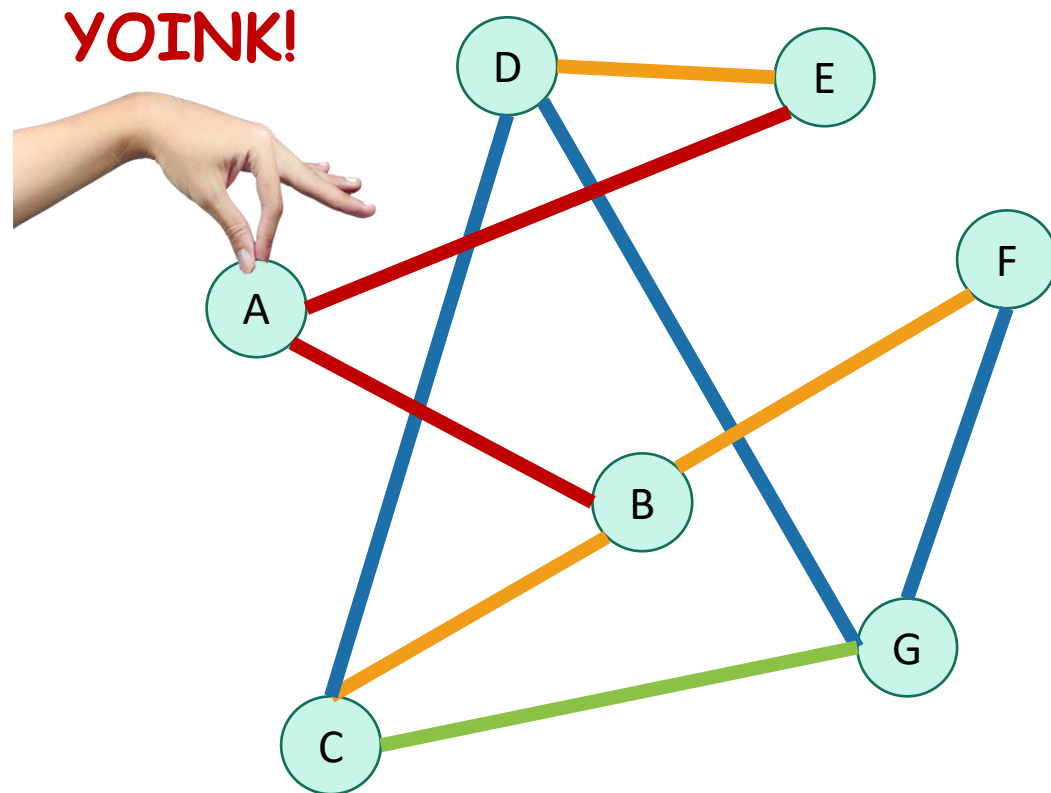
Why is it called breadth-first?

- We are implicitly building a tree:

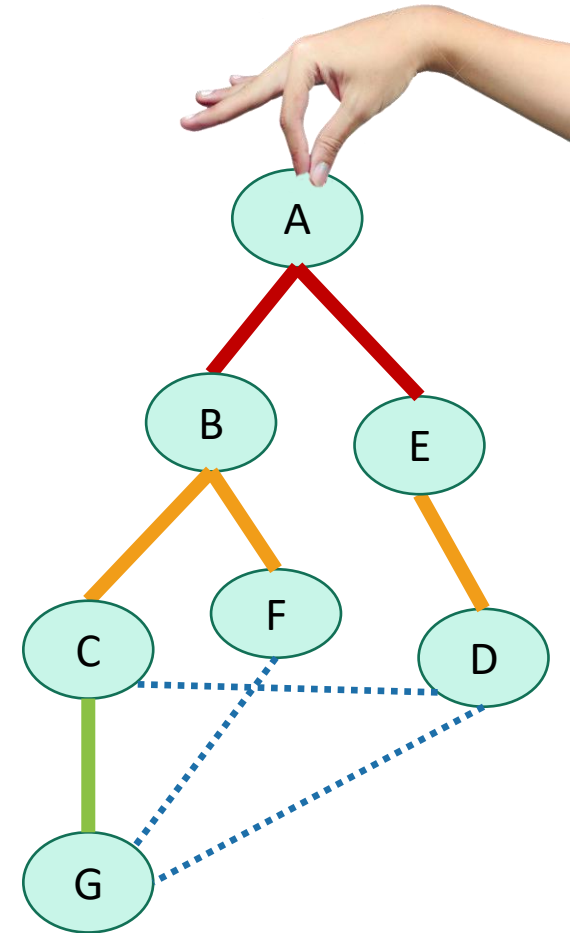


Why is it called breadth-first?

- We are implicitly building a tree:

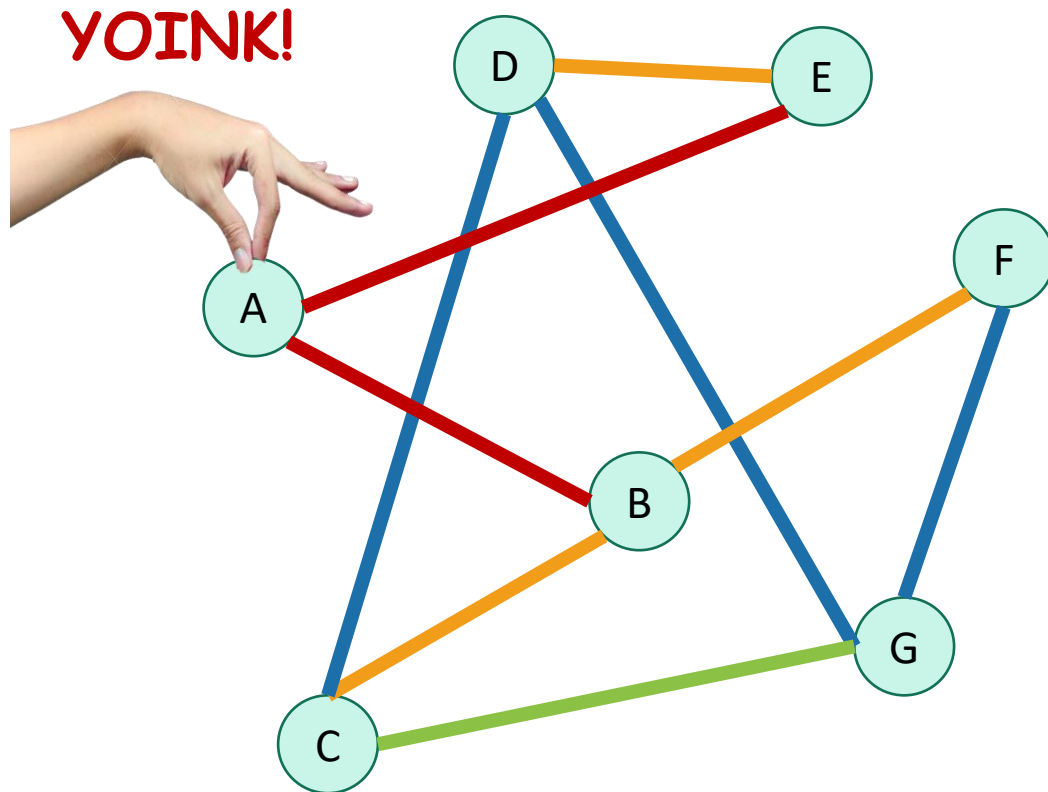


- First we go as broadly as we can.

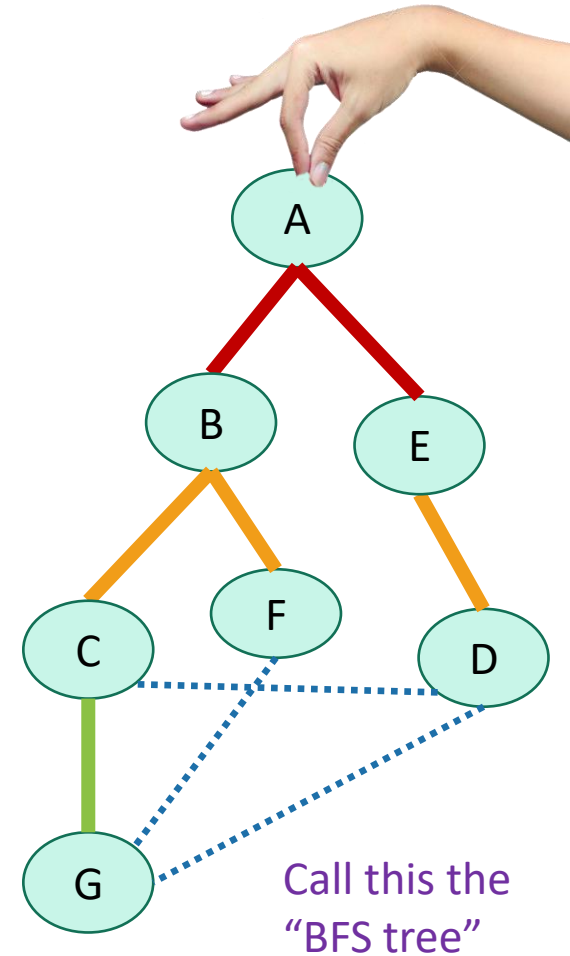


Why is it called breadth-first?

- We are implicitly building a tree:

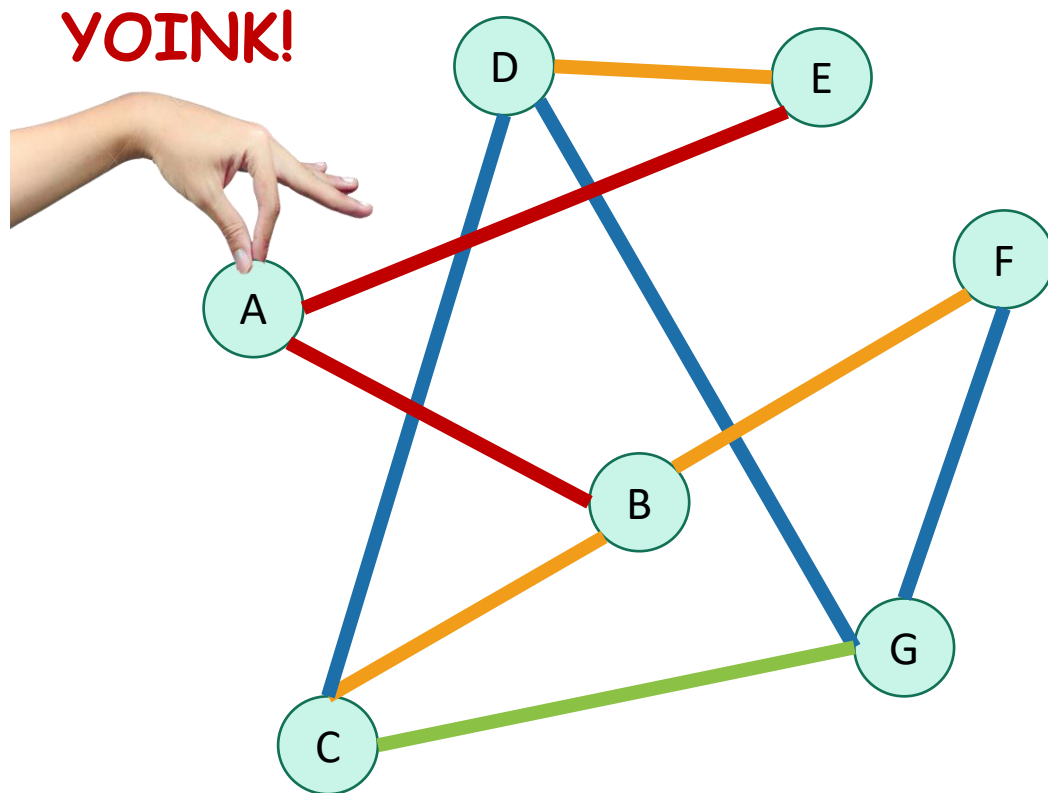


- First we go as broadly as we can.

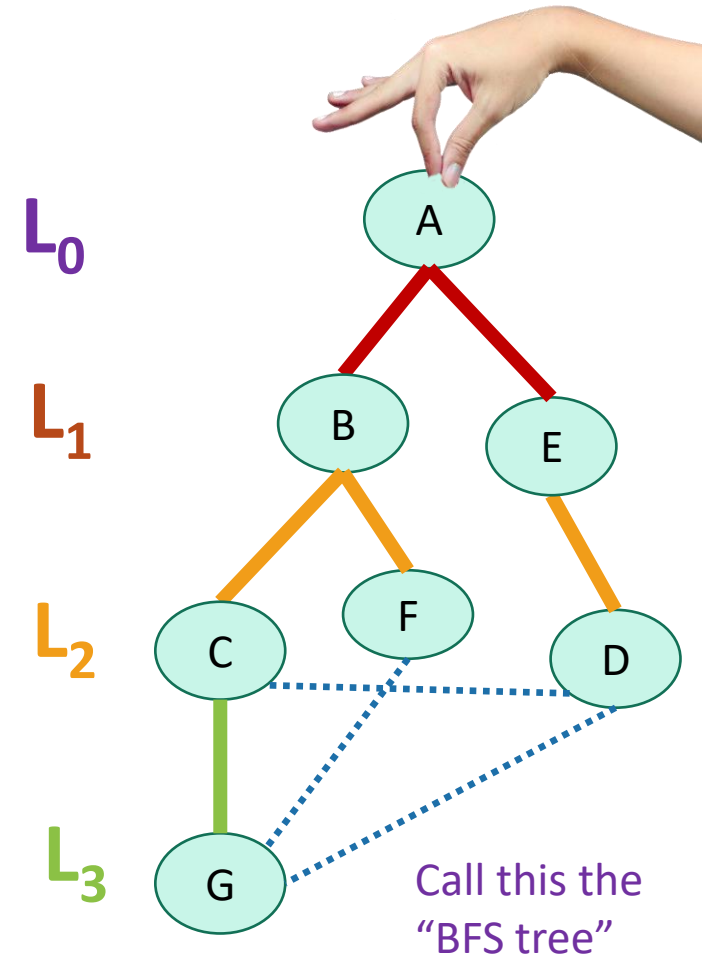


Why is it called breadth-first?

- We are implicitly building a tree:

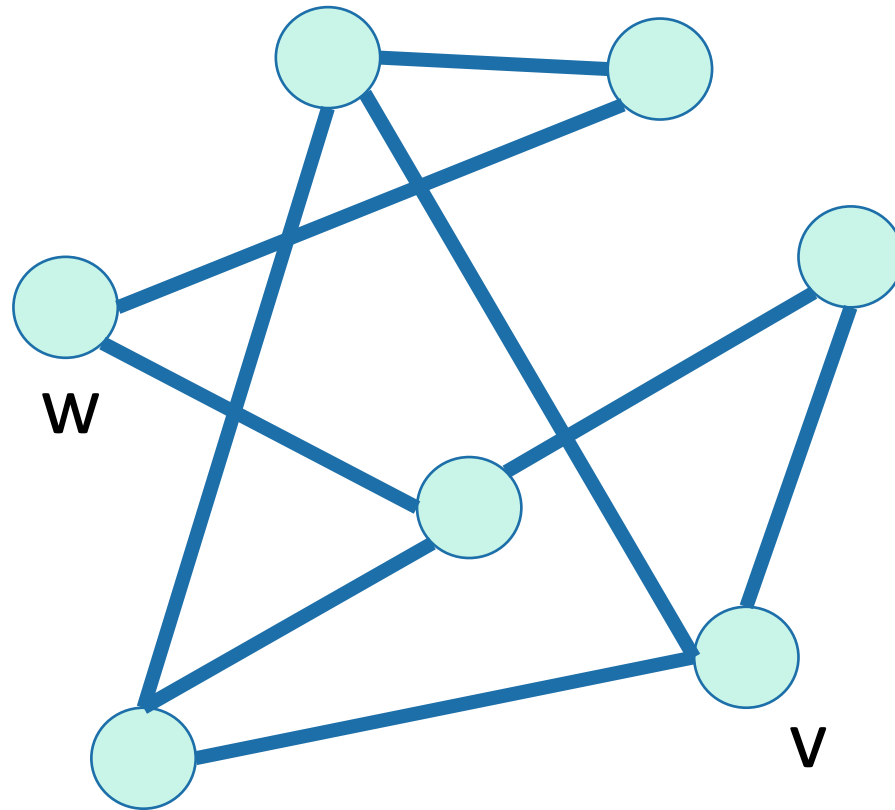


- First we go as broadly as we can.



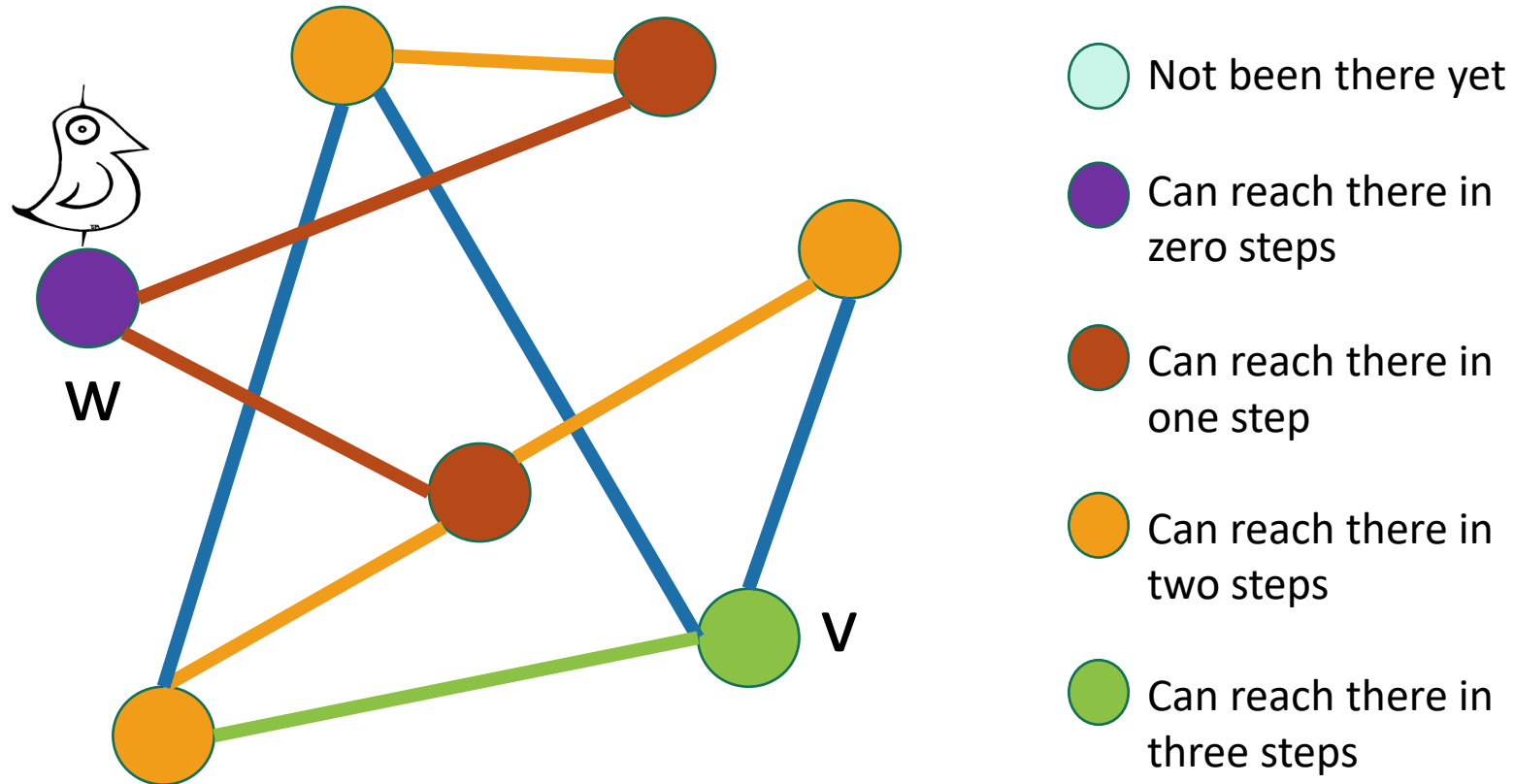
Application of BFS: shortest path

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



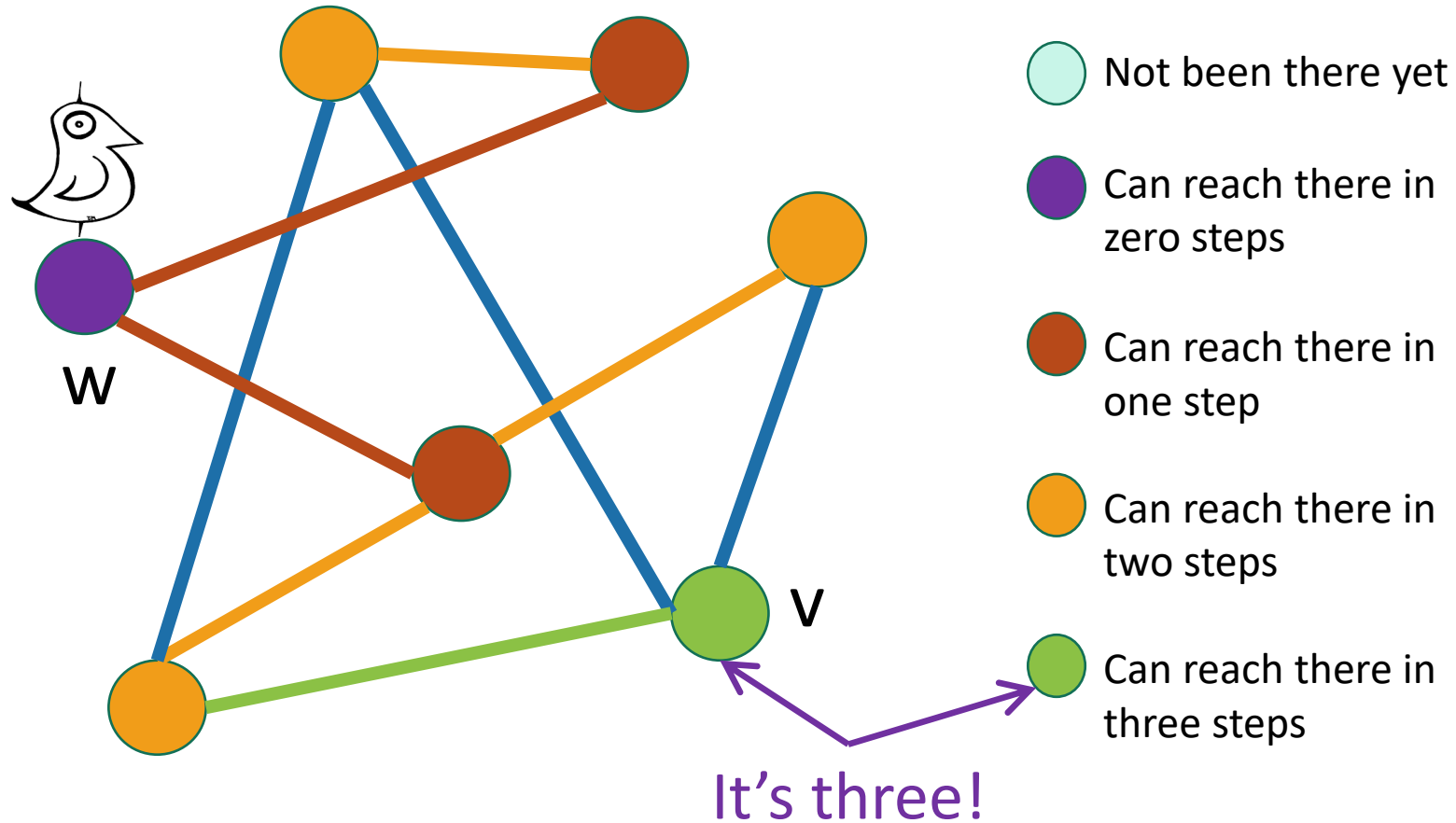
Application of BFS: shortest path

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



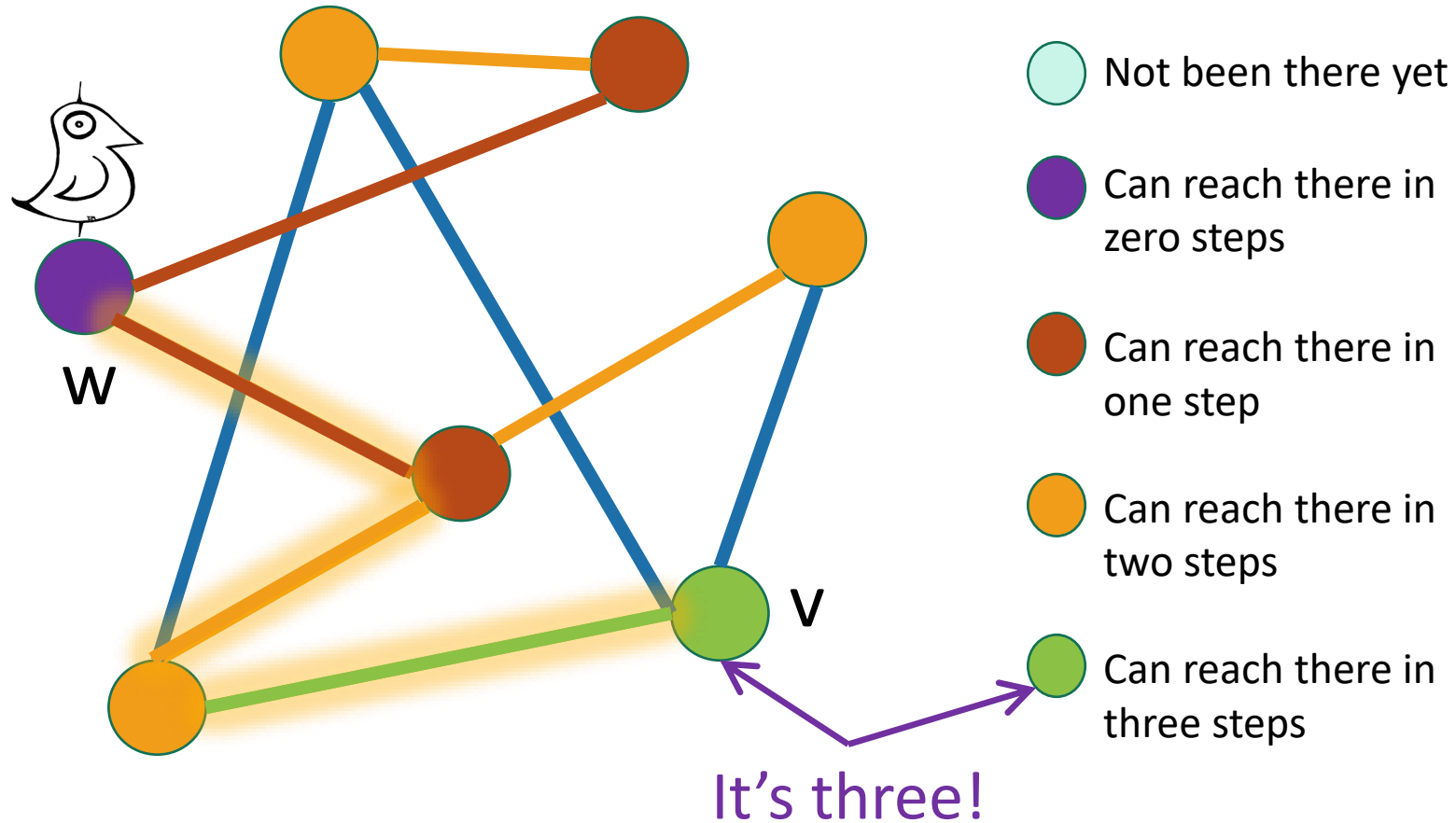
Application of BFS: shortest path

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



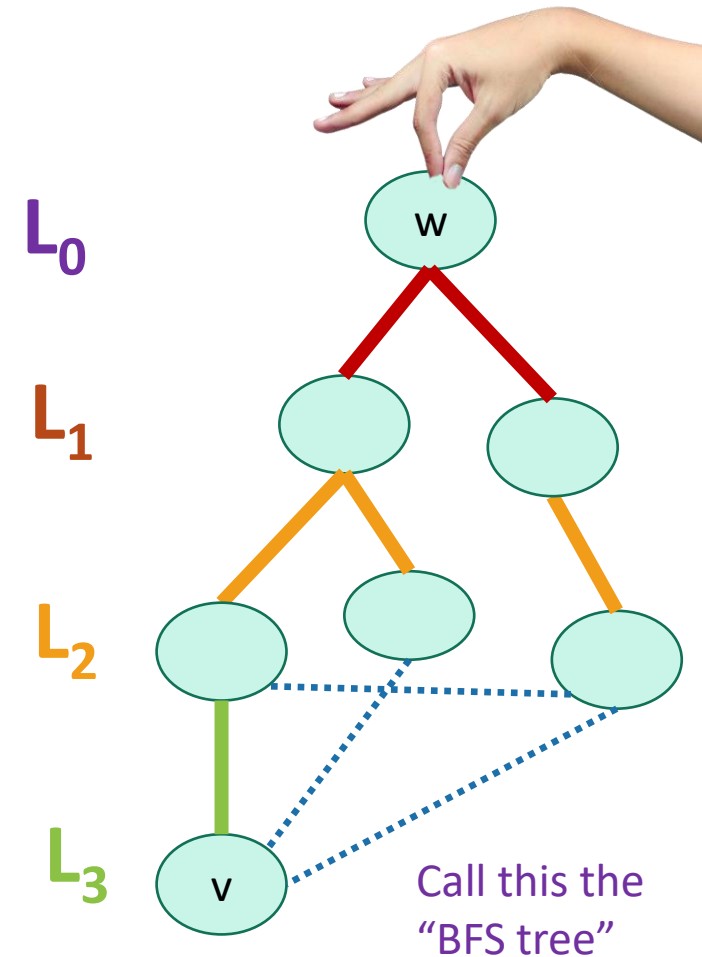
Application of BFS: shortest path

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



To find the **distance** between w and all other vertices v

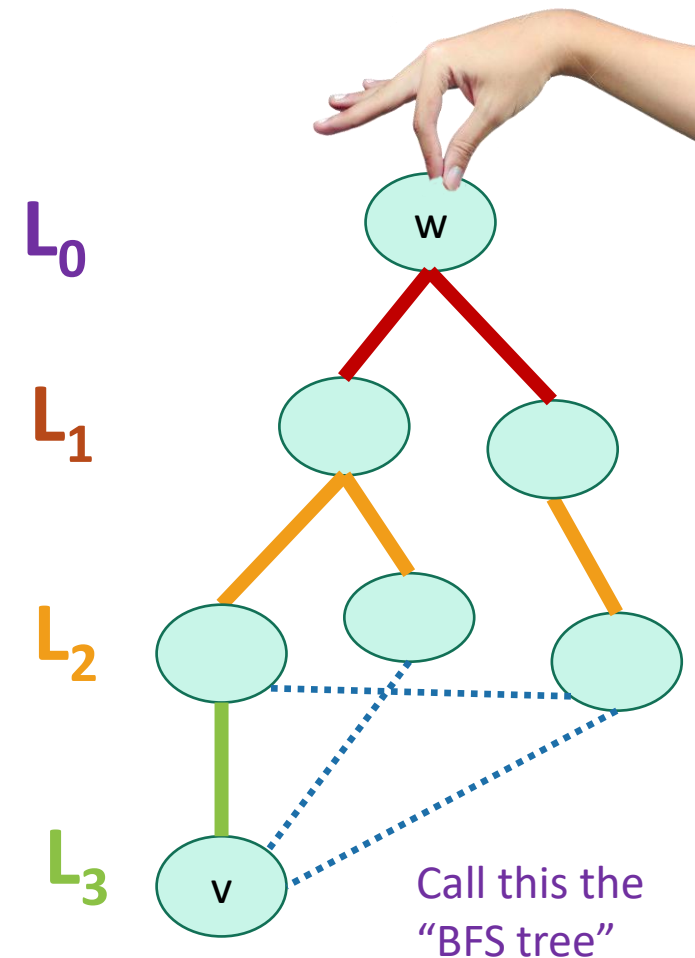
The **distance** between two vertices is the number of edges in the shortest path between them.



To find the **distance** between w and all other vertices v

- 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.

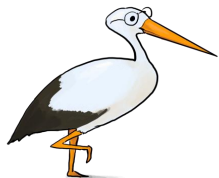
The **distance** between two vertices is the number of edges in the shortest path between them.



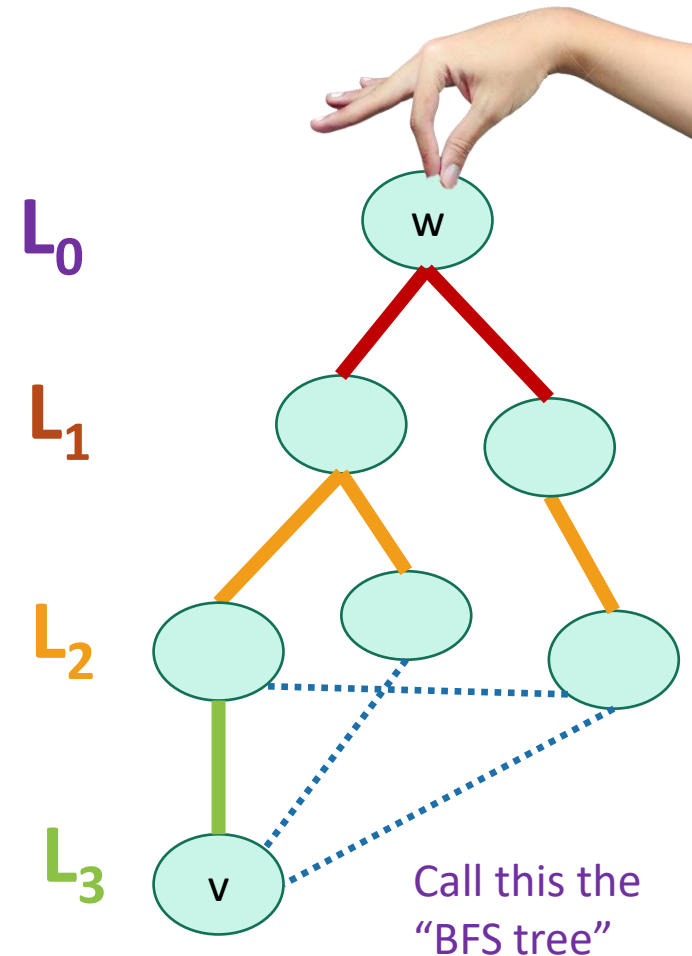
To find the **distance** between w and all other vertices v

- 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.

Modify the BFS pseudocode to return shortest paths!



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



What have we learned?

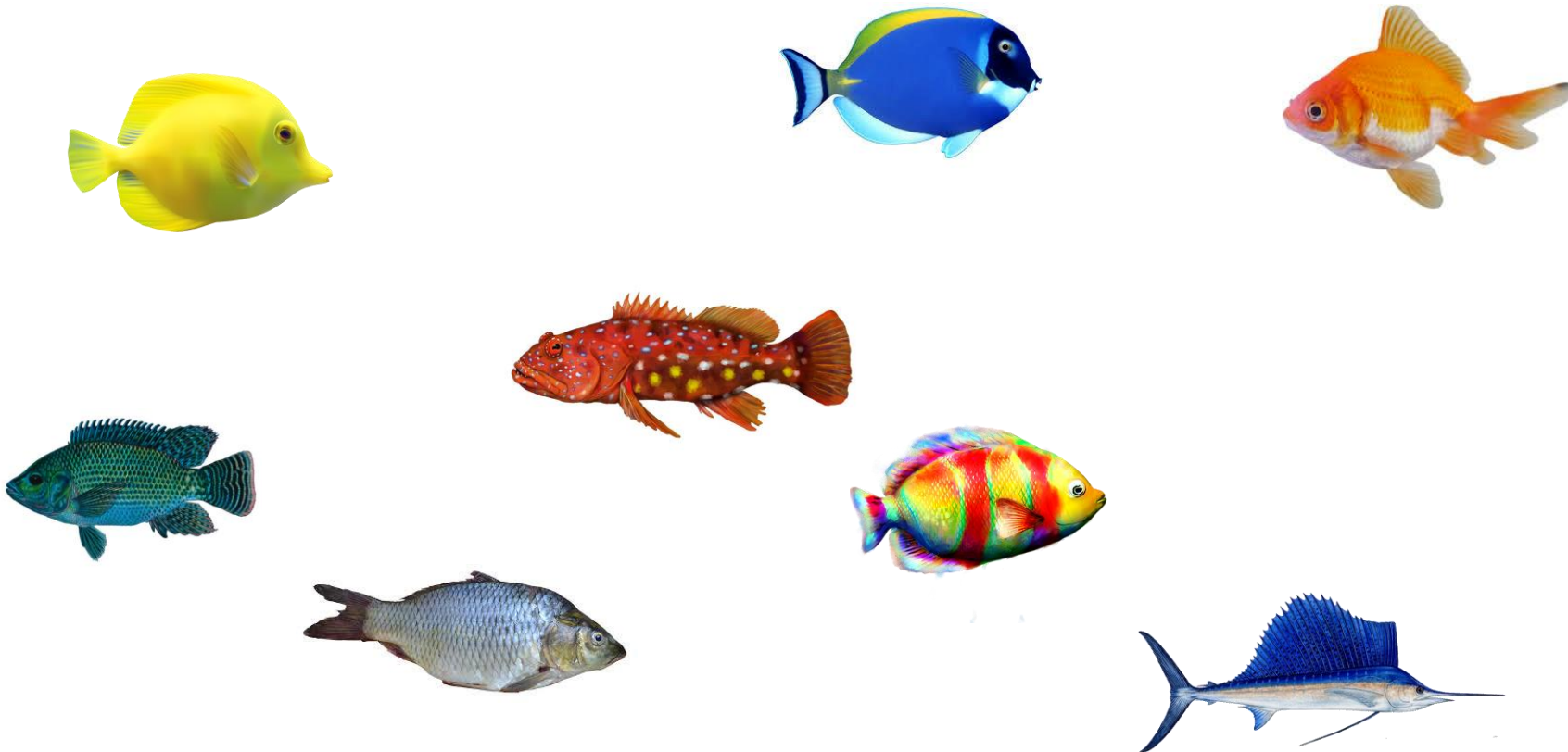
- The BFS tree is useful for computing distances between pairs of vertices.
- We can find the shortest path between u and v in time $O(m)$.

Another application of BFS

- Testing bipartite-ness

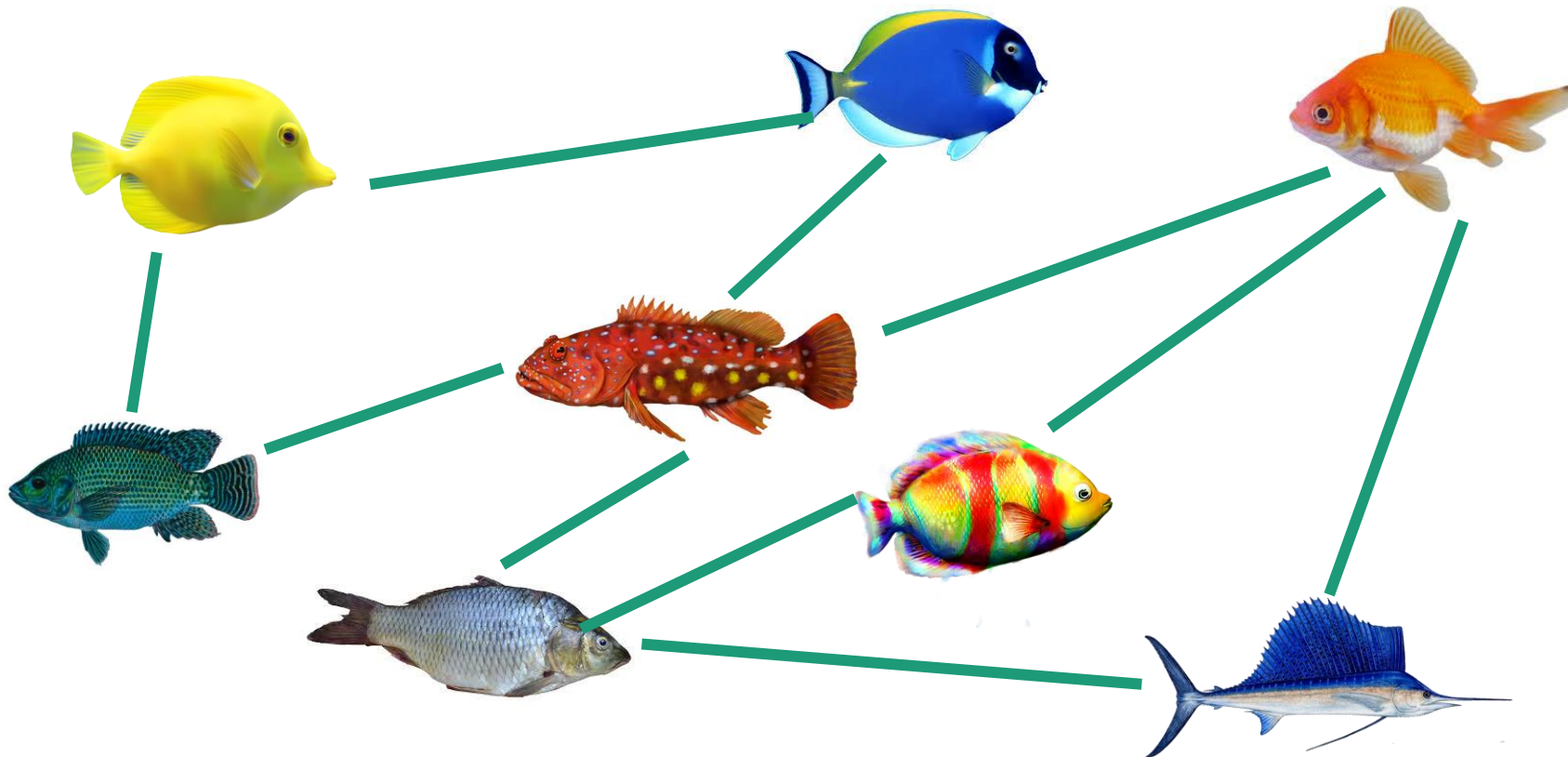
Exercise: fish

- 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.



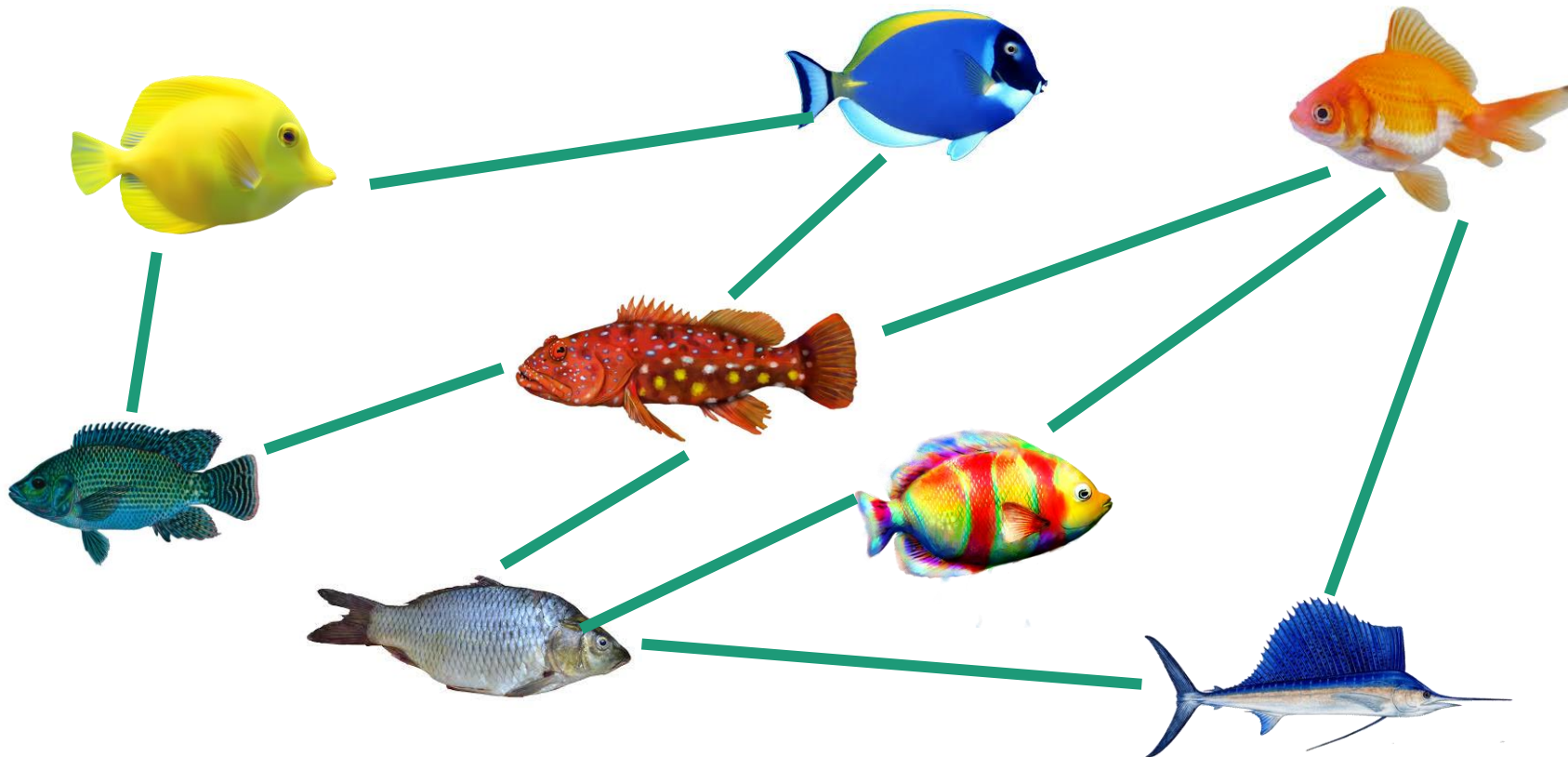
Exercise: fish

- 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.



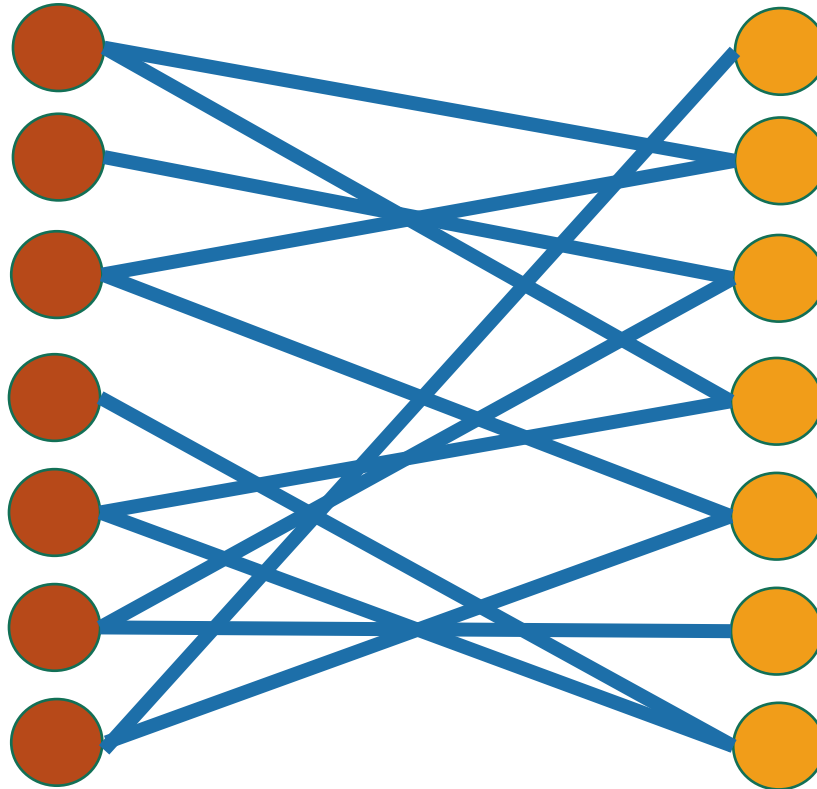
Exercise: fish

- 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?



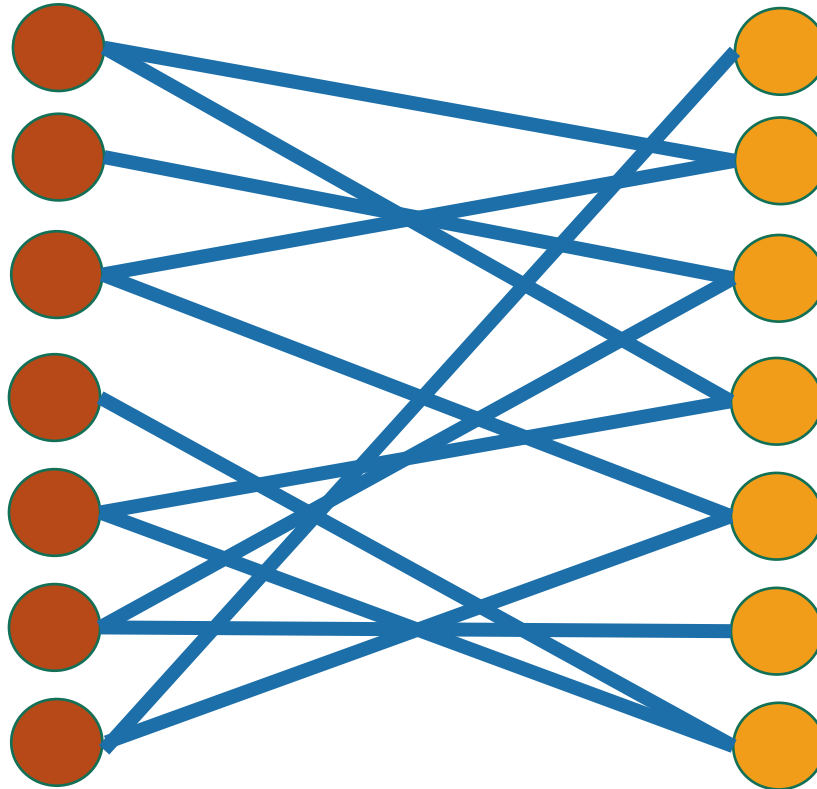
Bipartite graphs

- A bipartite graph looks like this:



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

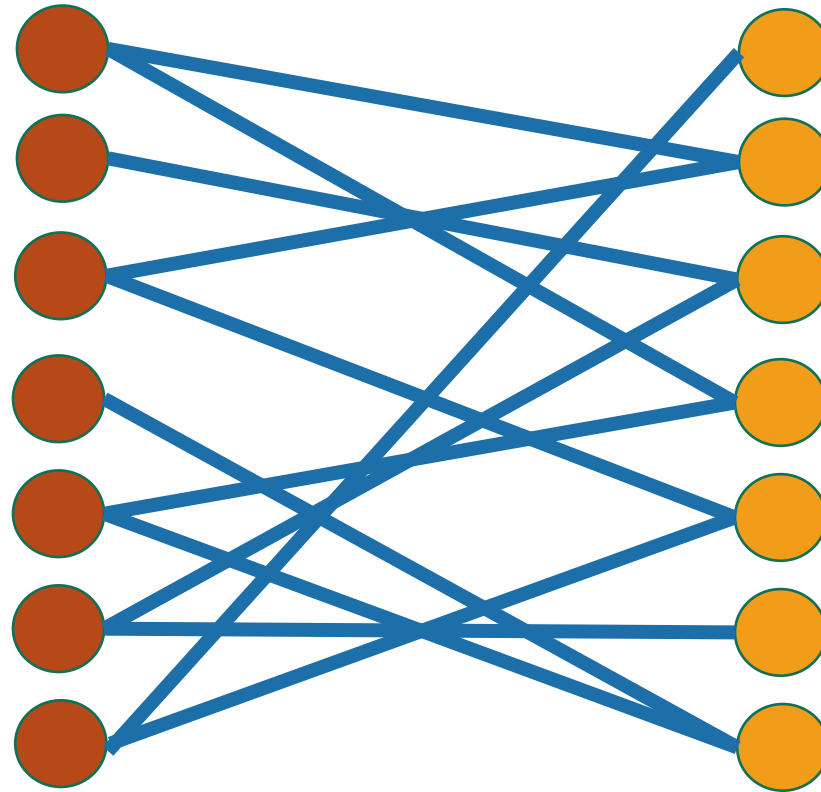
Example:

- are in tank A
- are in tank B
- if the fish fight

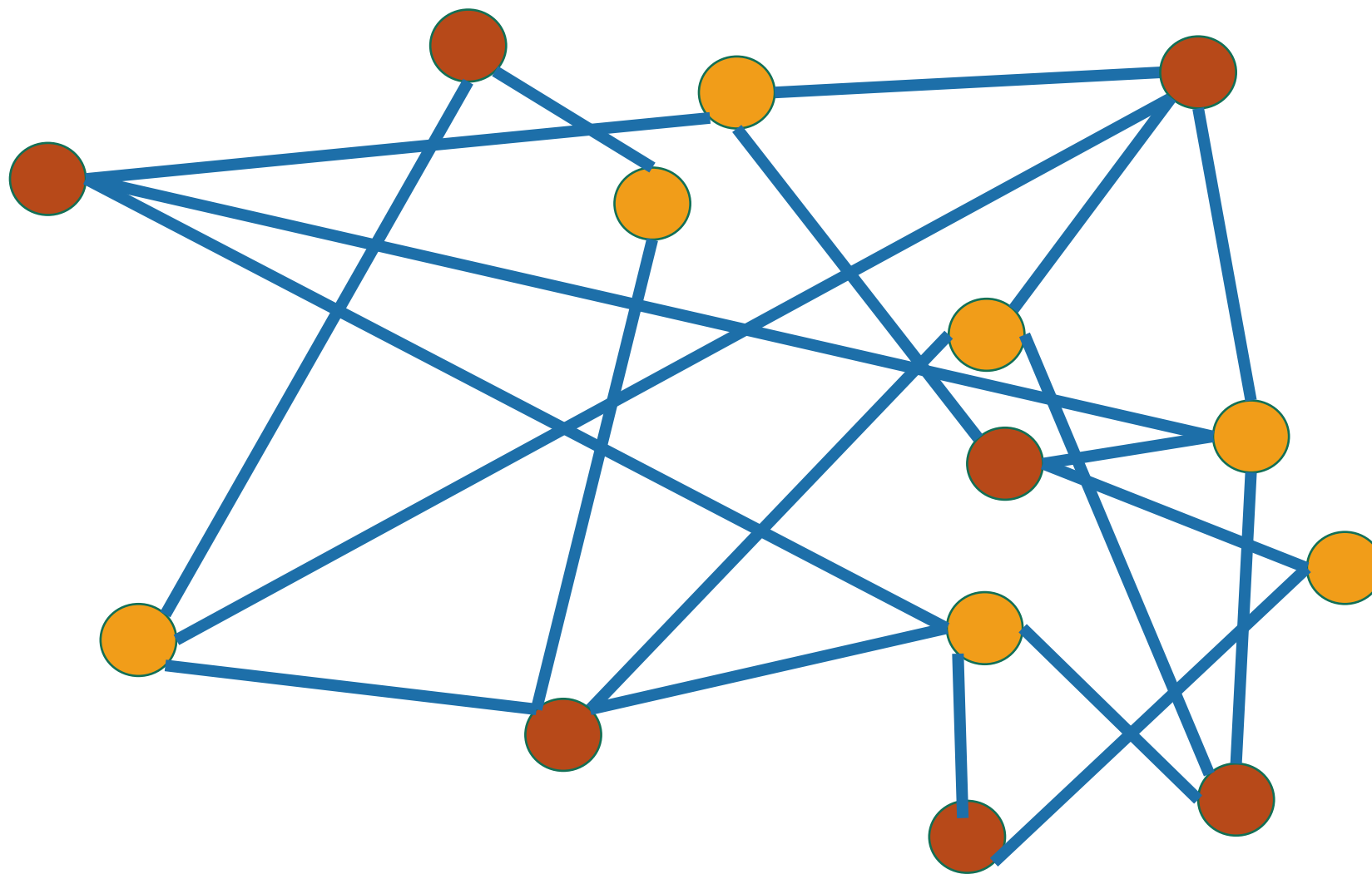
Example:

- are students
- are classes
- if the student is enrolled in the class

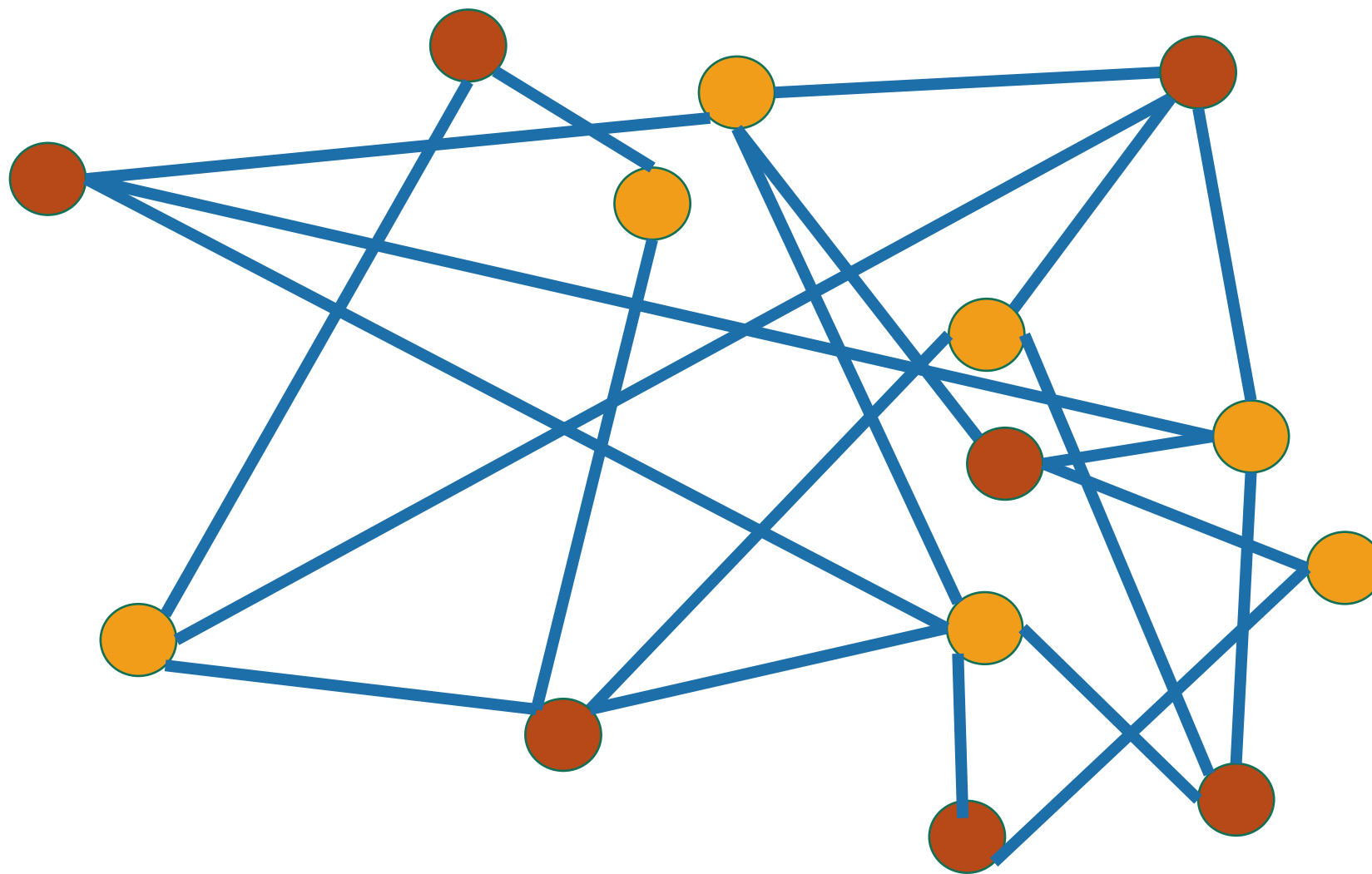
Is this graph bipartite?



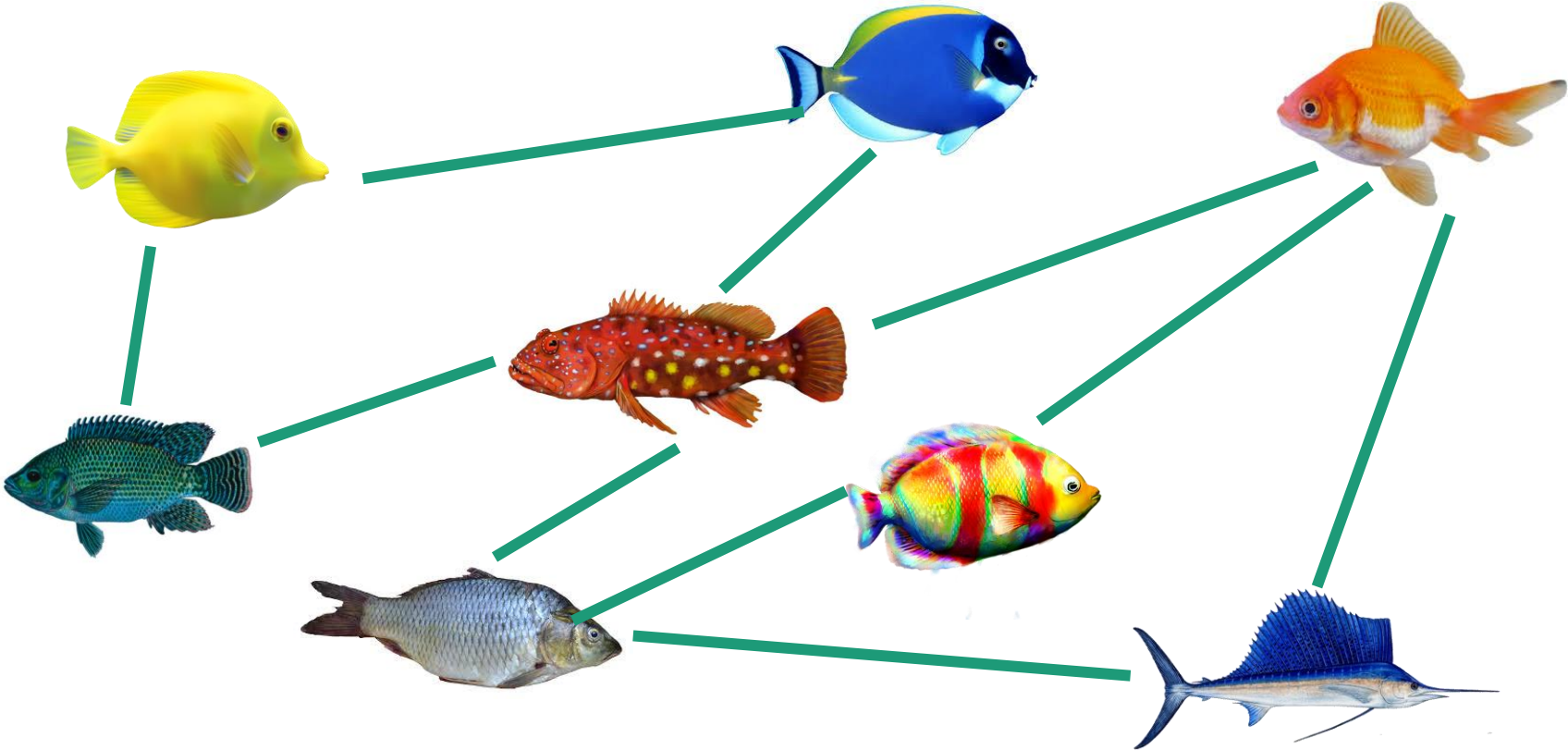
How about this one?



How about this one?

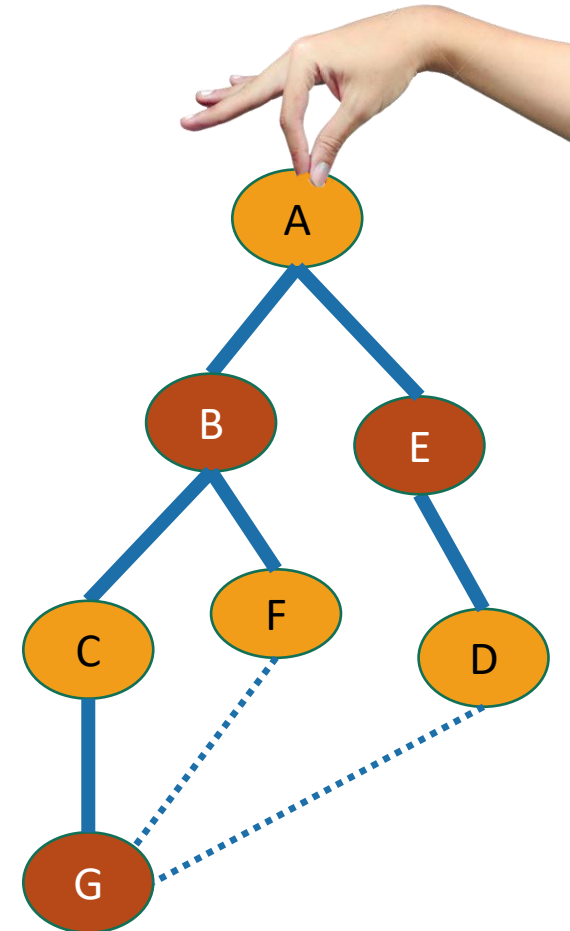


This one?



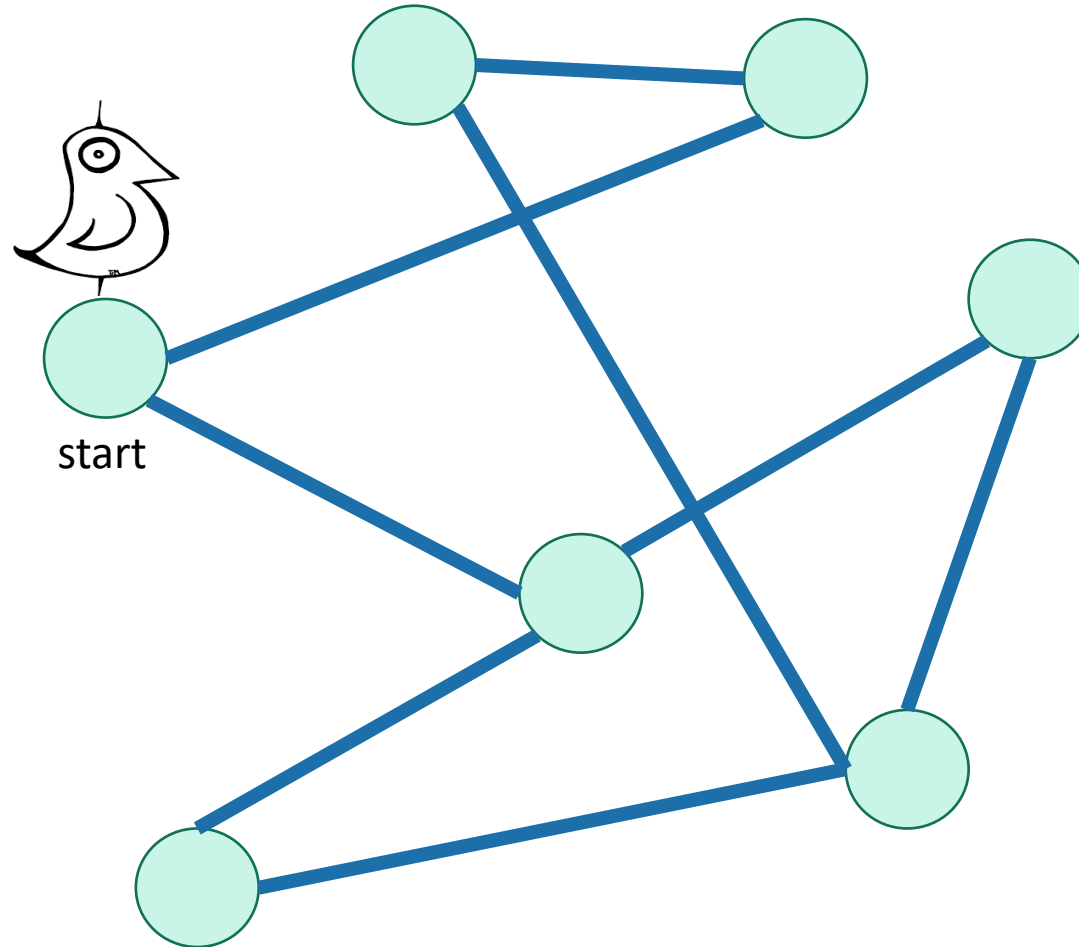
Application of BFS: Testing Bipartiteness






- 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

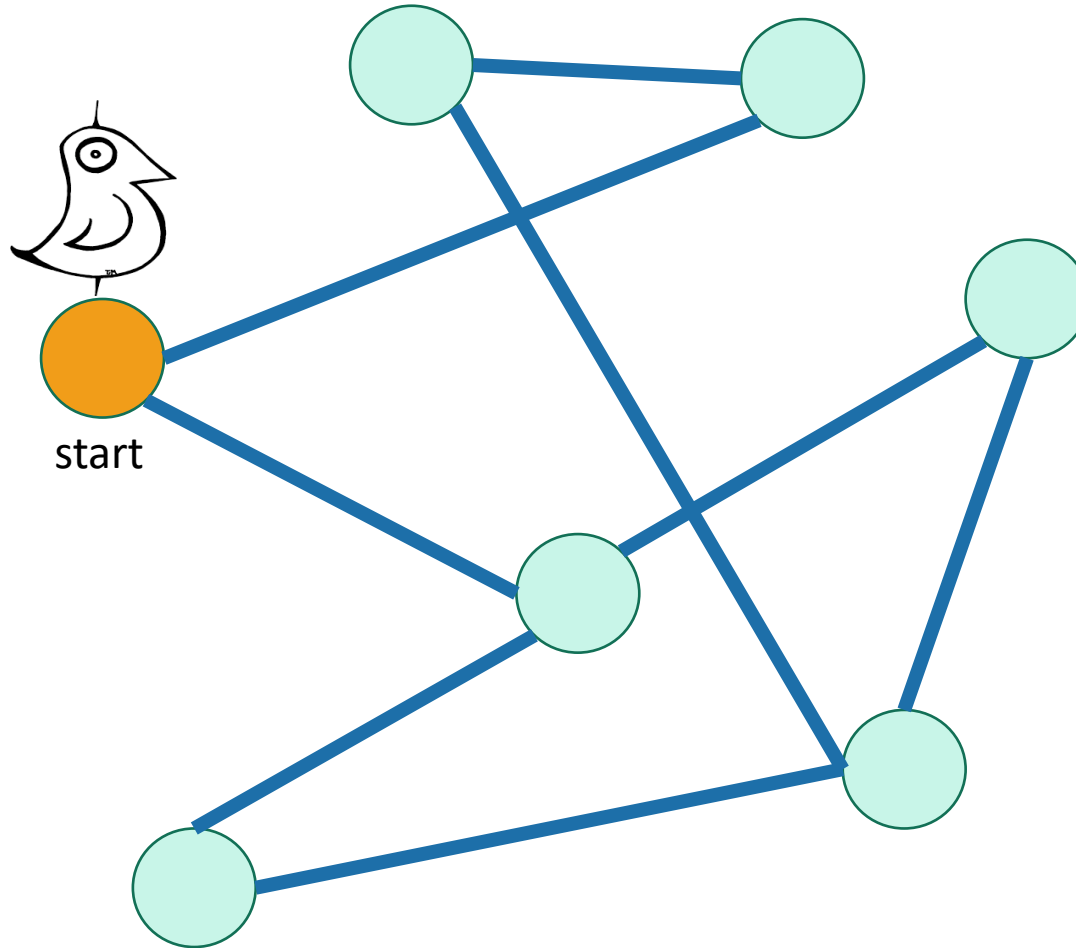
For testing bipartite-ness



-  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

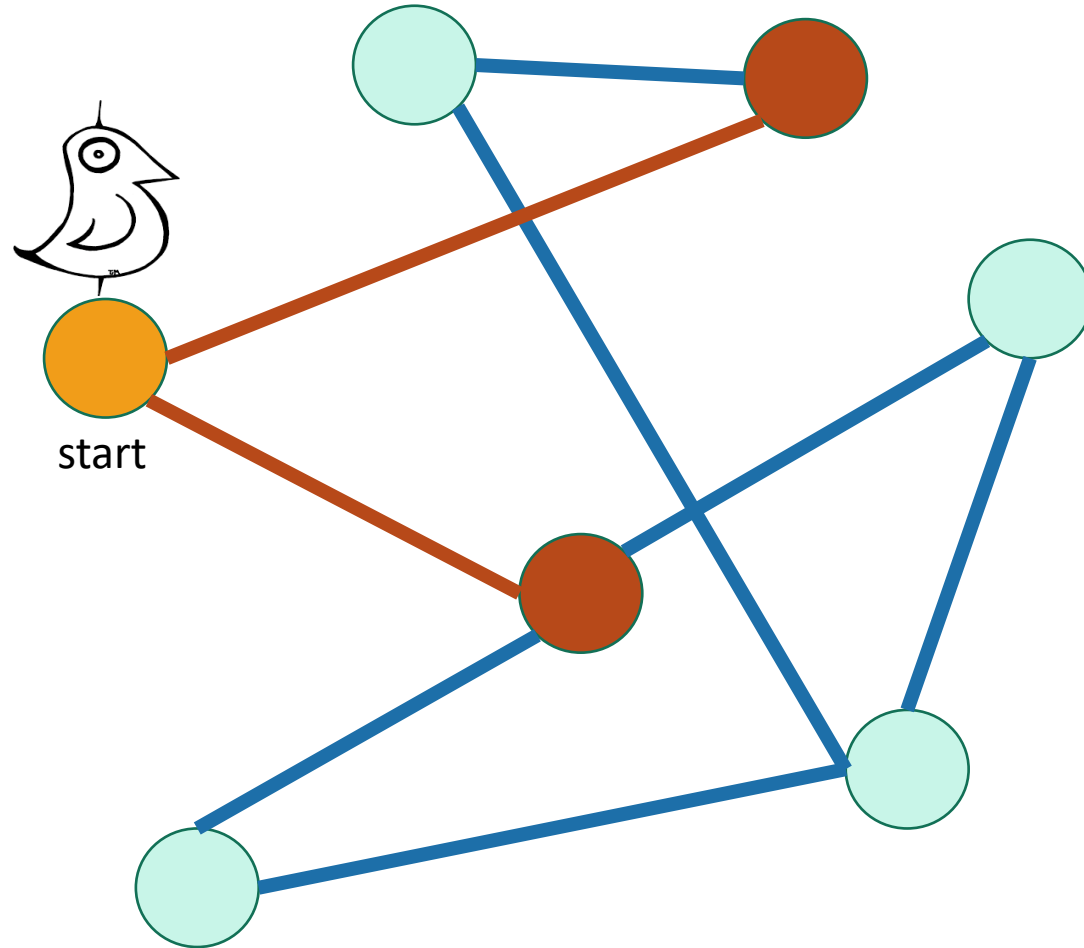
For testing bipartite-ness








- 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

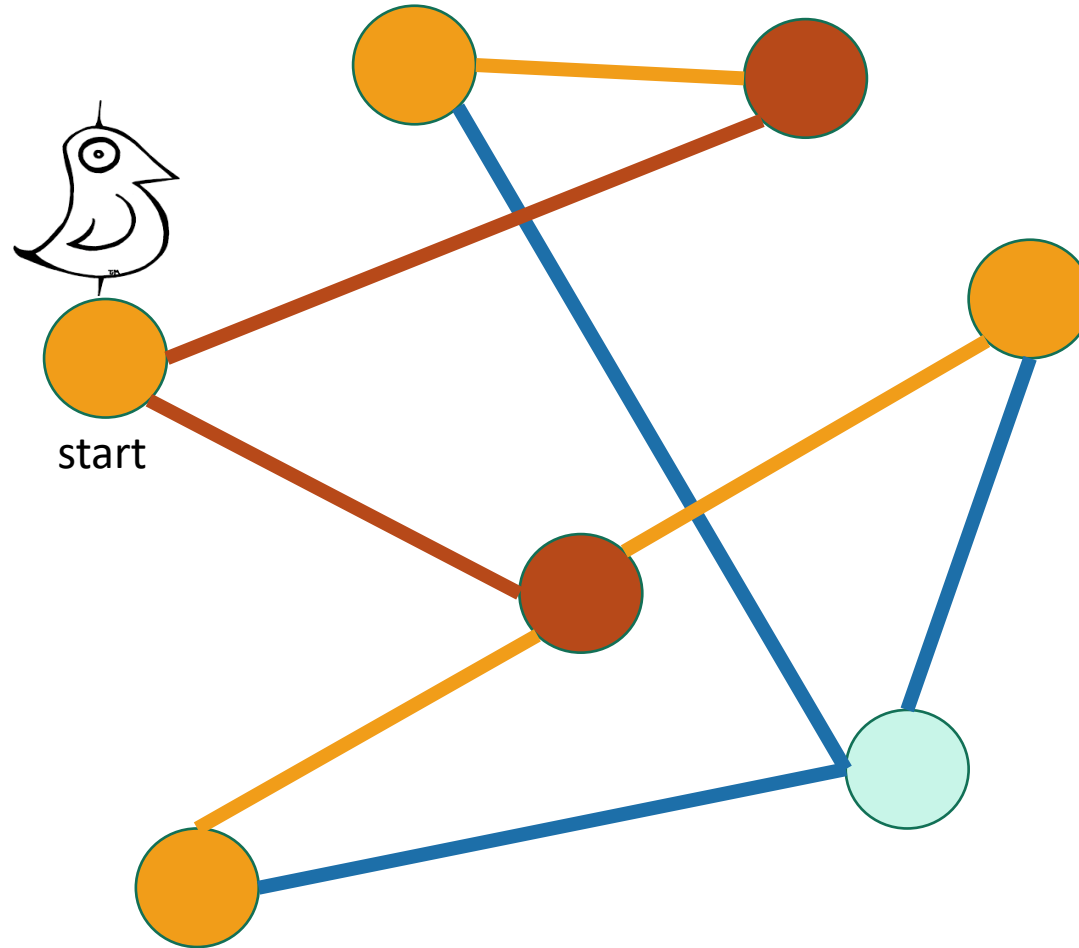
For testing bipartite-ness



-  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

Breadth-First Search

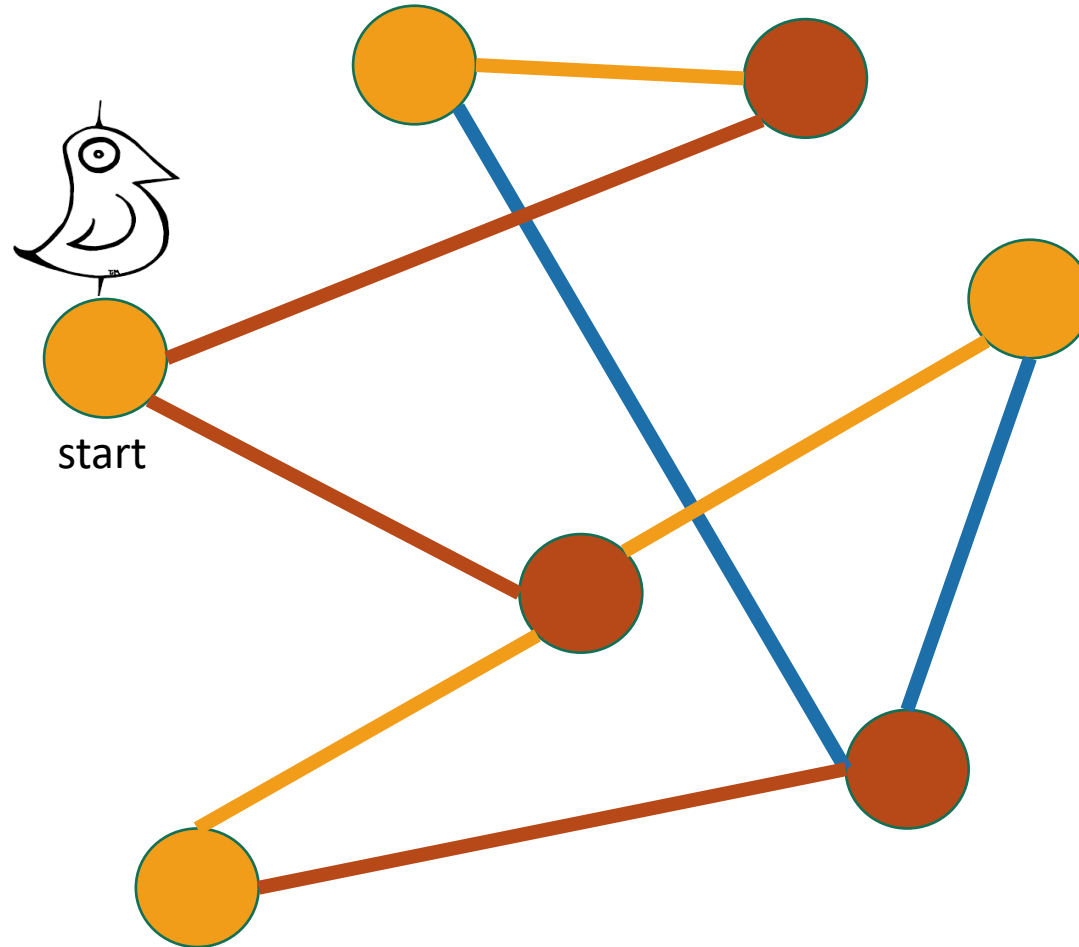
For testing bipartite-ness



- 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

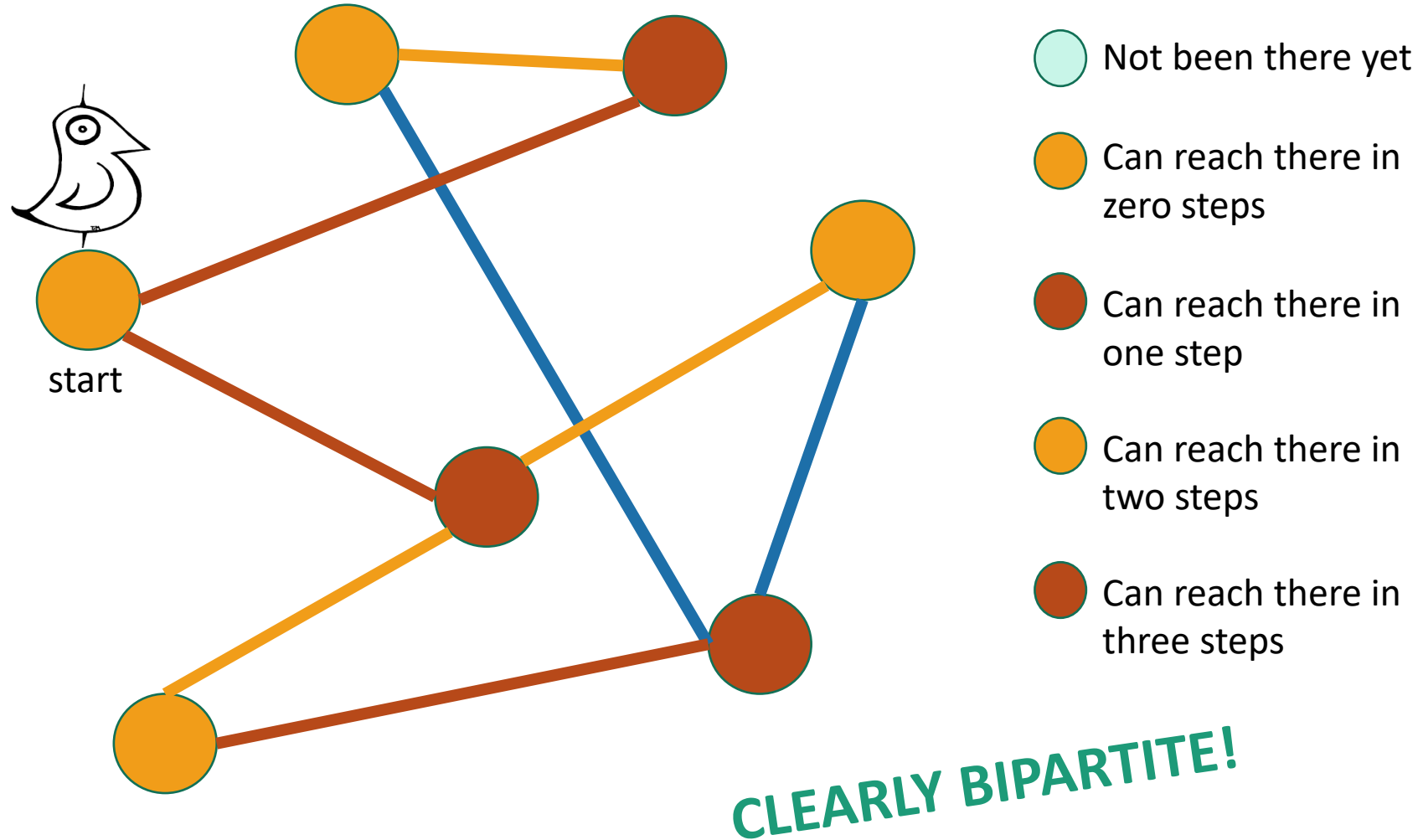
For testing bipartite-ness



- 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

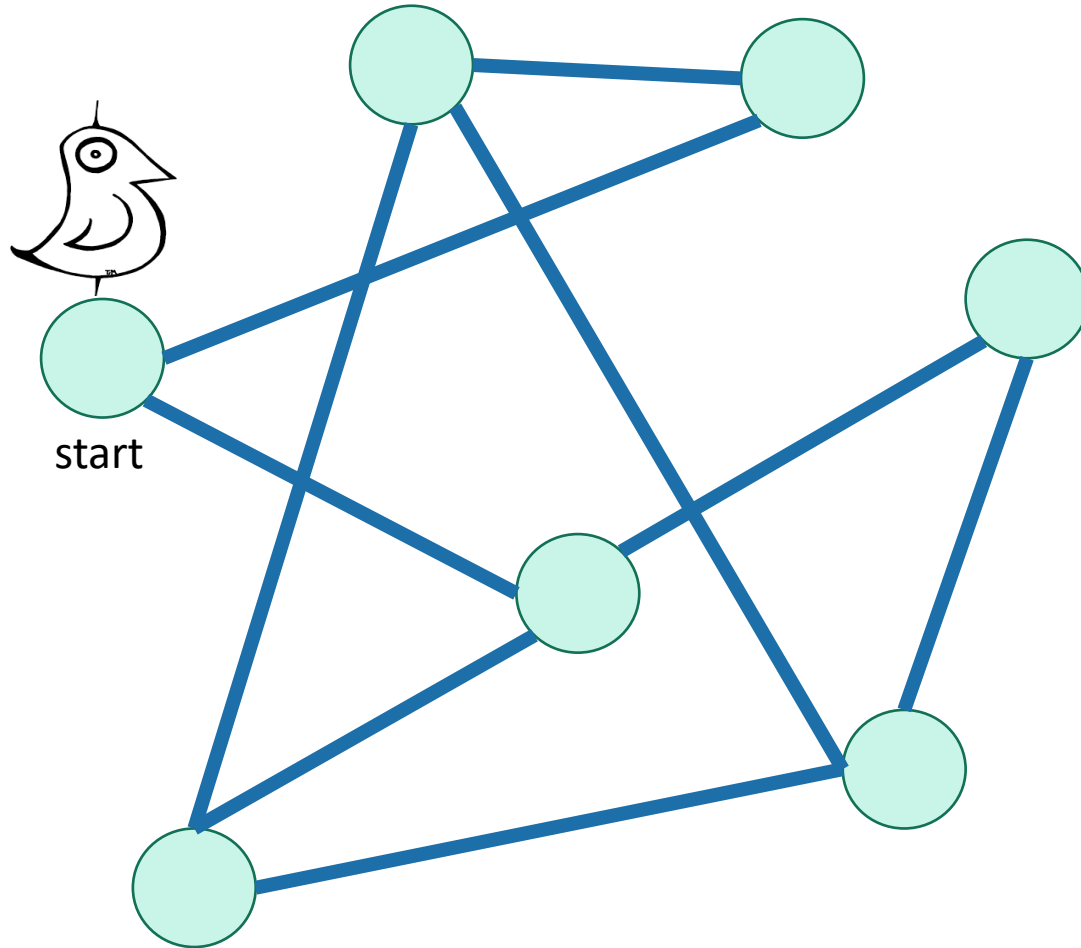
Breadth-First Search






For testing bipartite-ness



Breadth-First Search

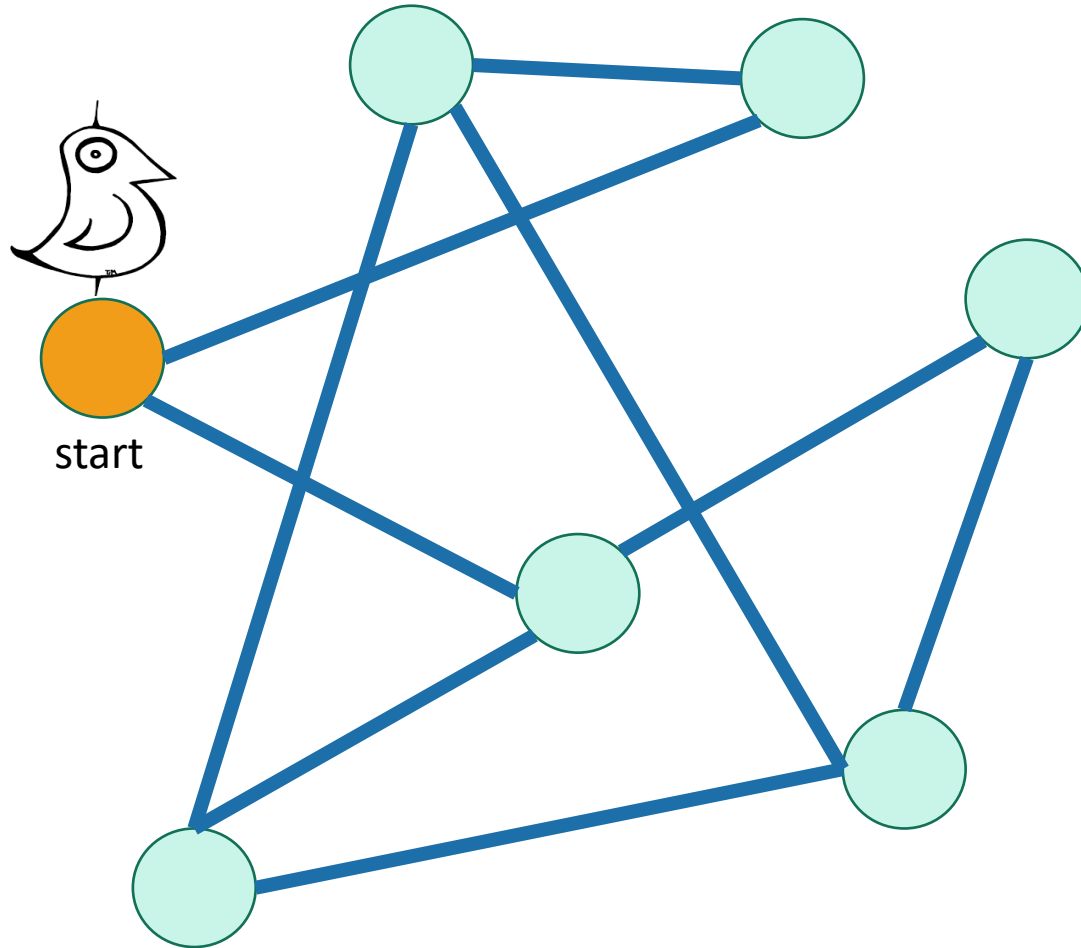
For testing bipartite-ness



-  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

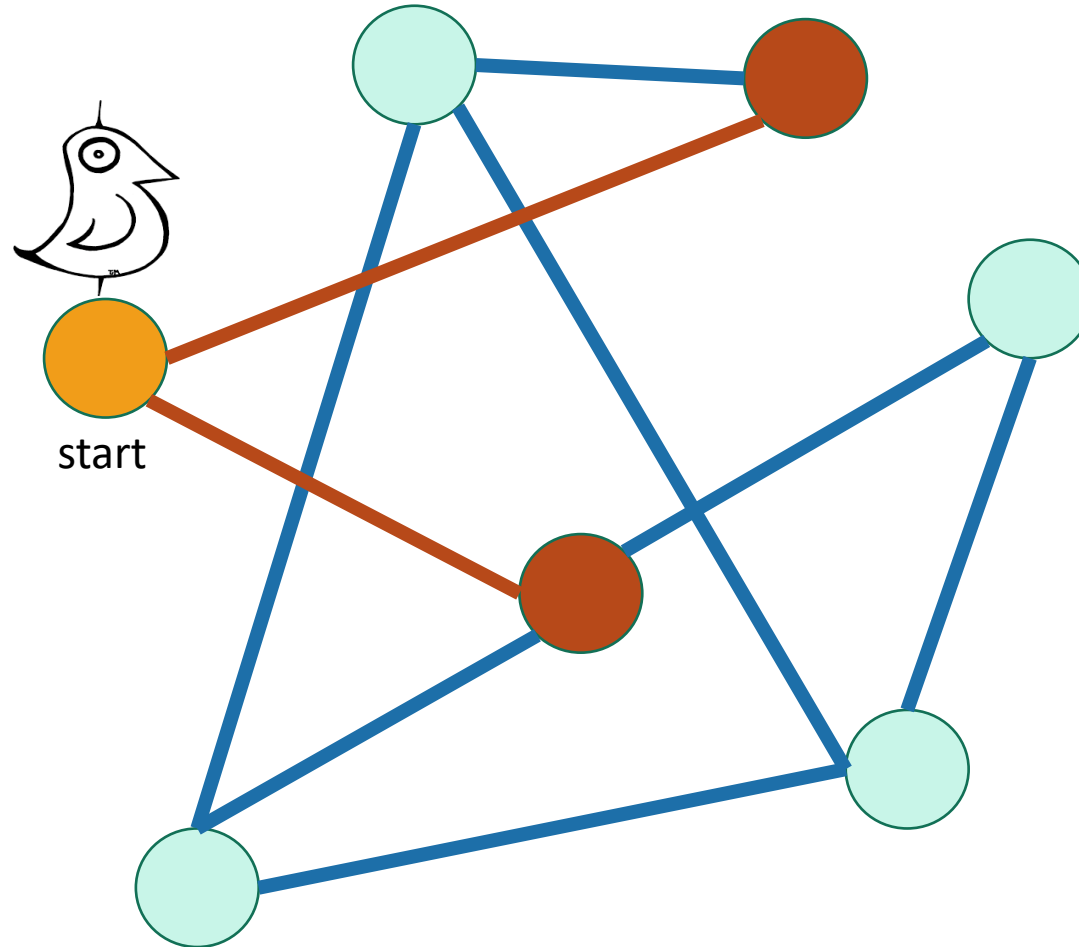
For testing bipartite-ness



- 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

Breadth-First Search

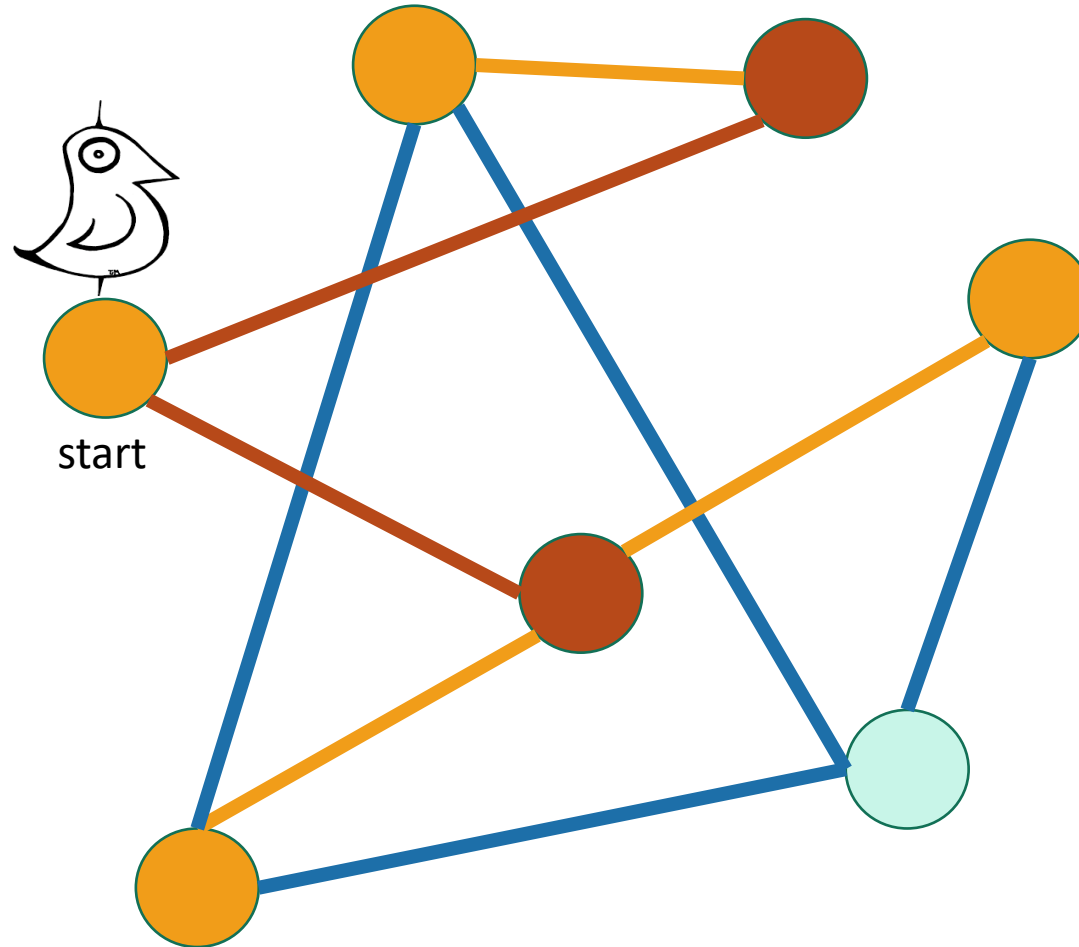
For testing bipartite-ness



- 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

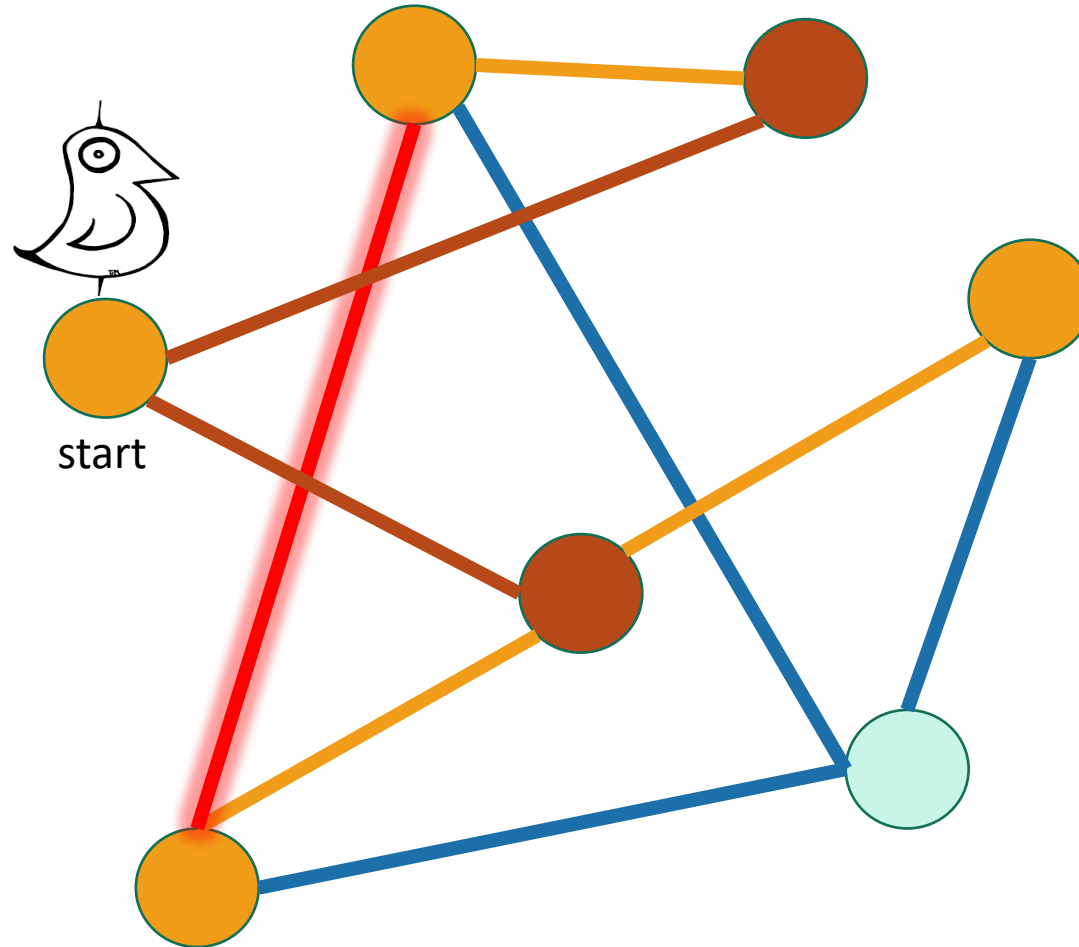
For testing bipartite-ness








- 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

Breadth-First Search

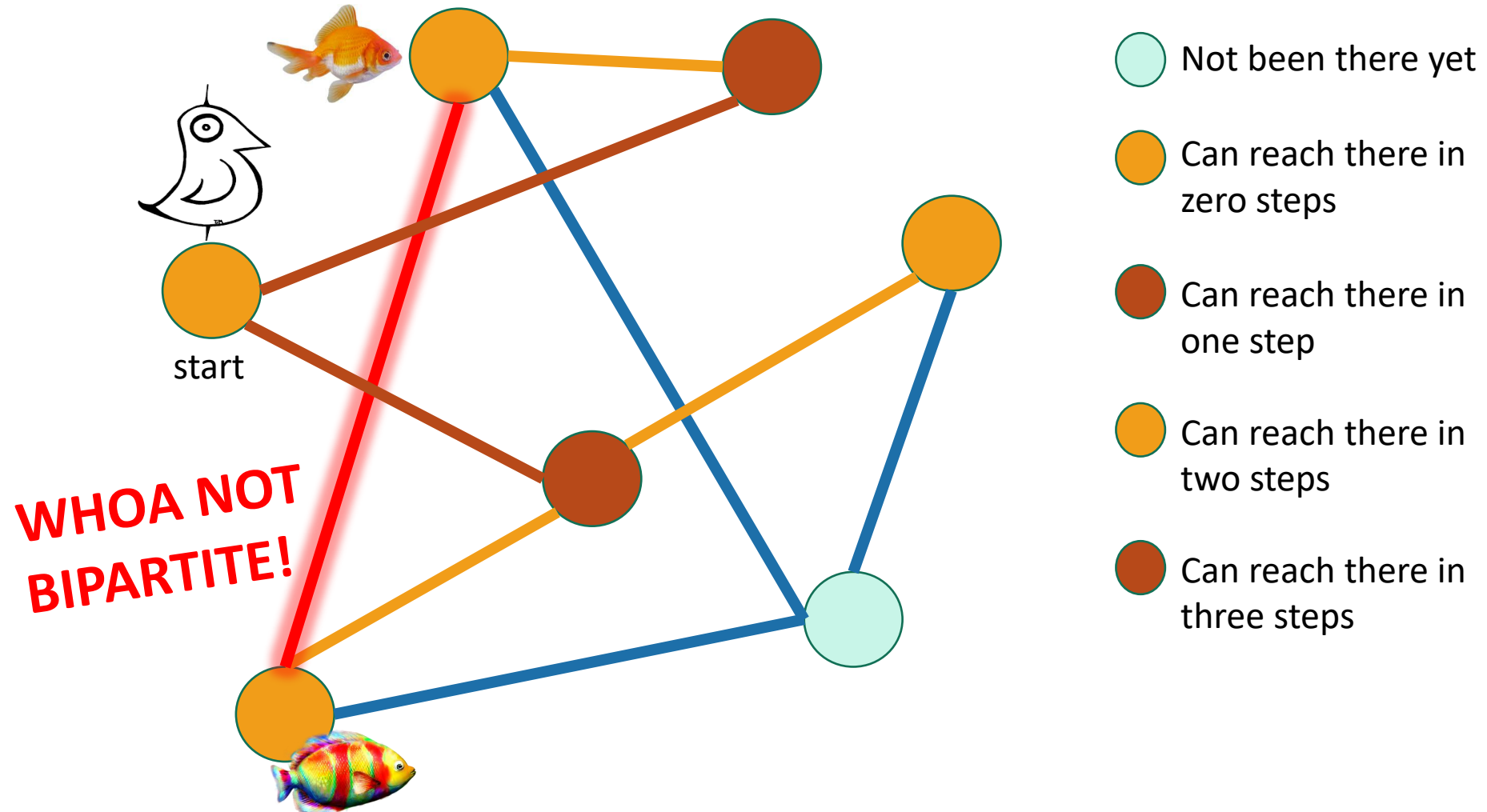
For testing bipartite-ness



-  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



What have we learned?

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



Acknowledgement

- Stanford University

Thank You