Khái niệm về đồ thị

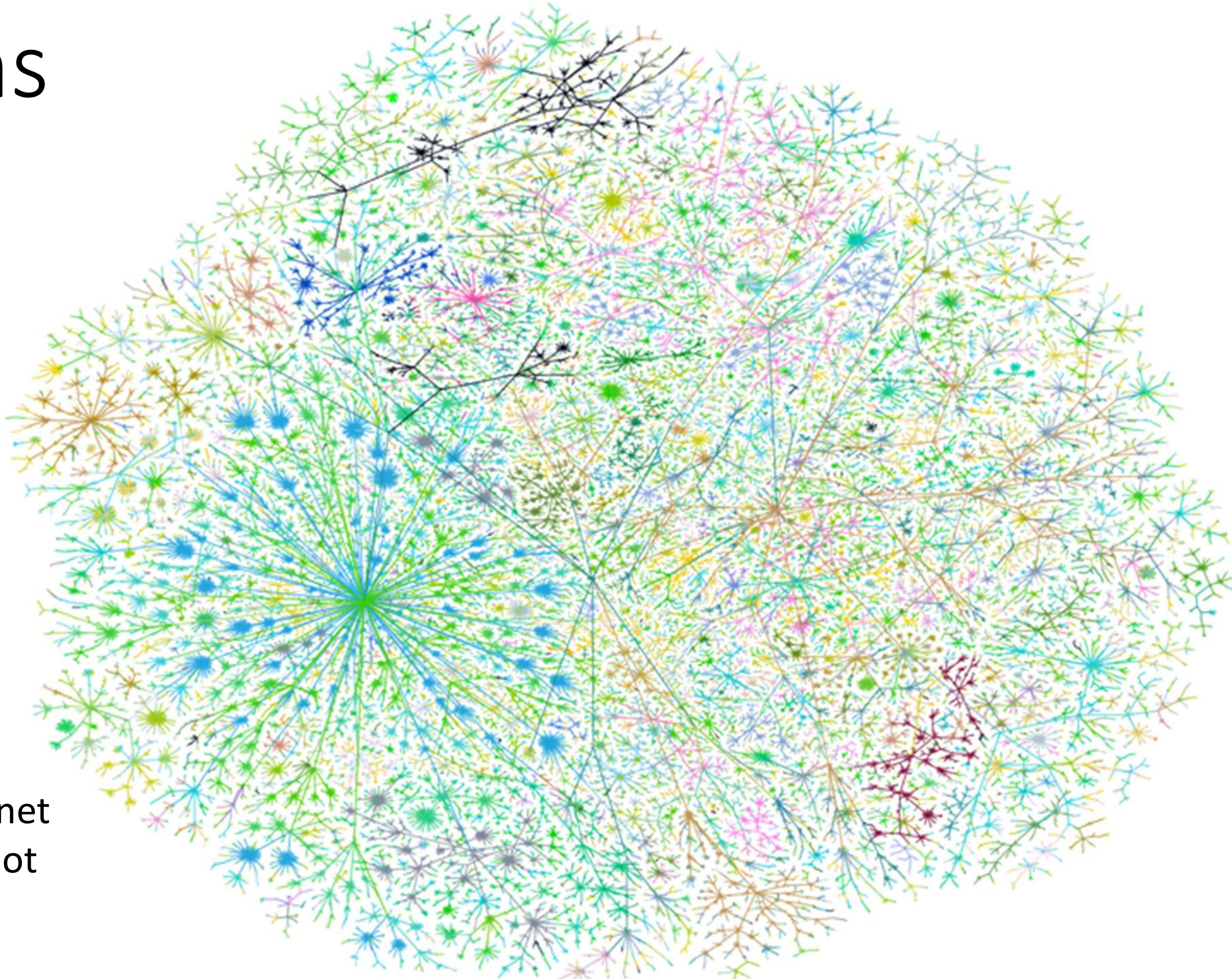
Nội dung

- Khái niệm về đồ thị (Graphs and terminology)
- Tìm kiếm theo chiều sâu (Depth-first search)
- Tìm kiếm theo chiều rộng (Breadth-first search)

Sử dụng một phần tài liệu bài giảng CS161 Stanford University

Phần 1: Khái niệm về đồ thị

Graphs



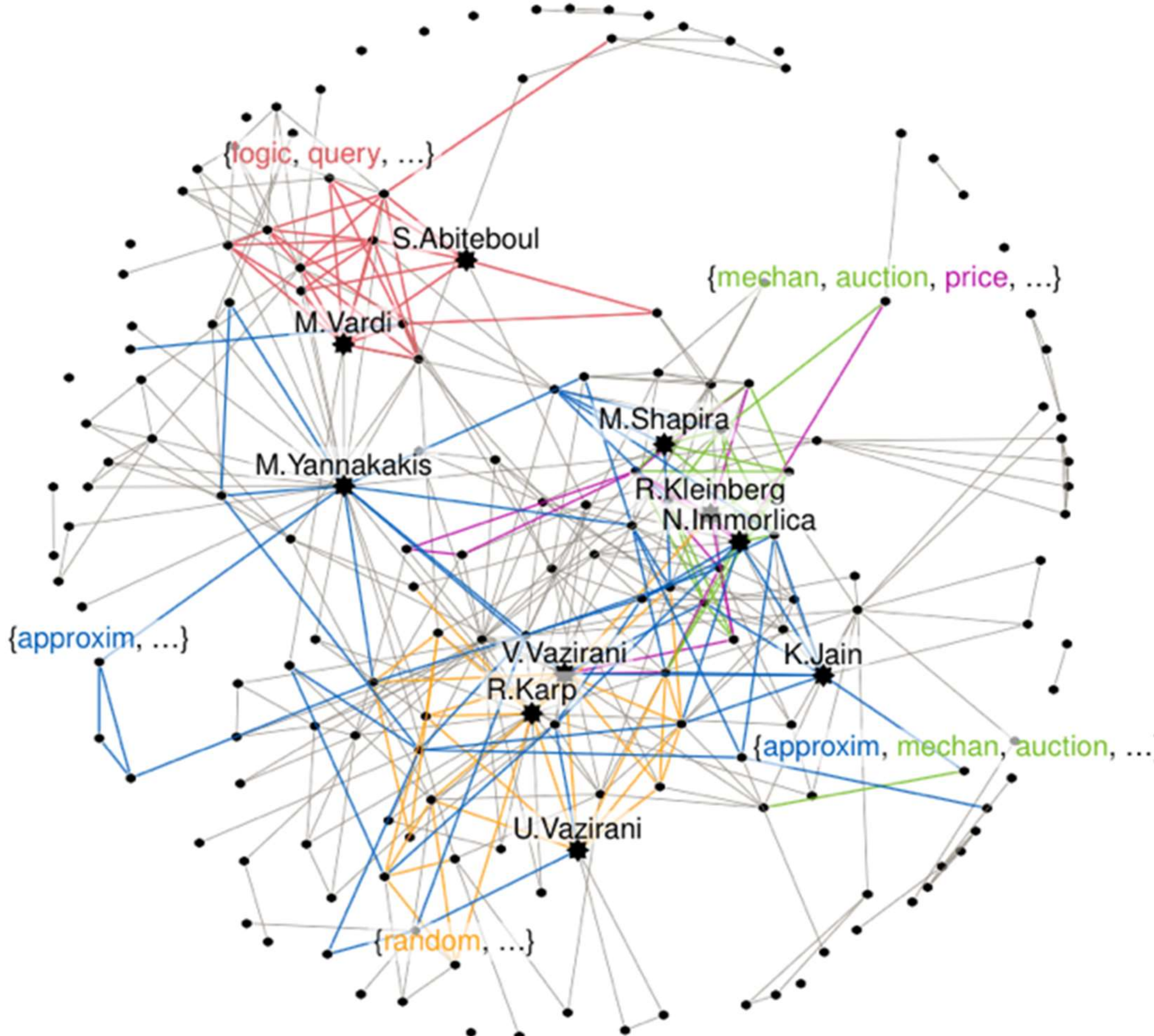
Graph of the internet
(circa 1999…it's a lot
bigger now…)

Graphs

Citation graph of literary theory academic papers

Graphs

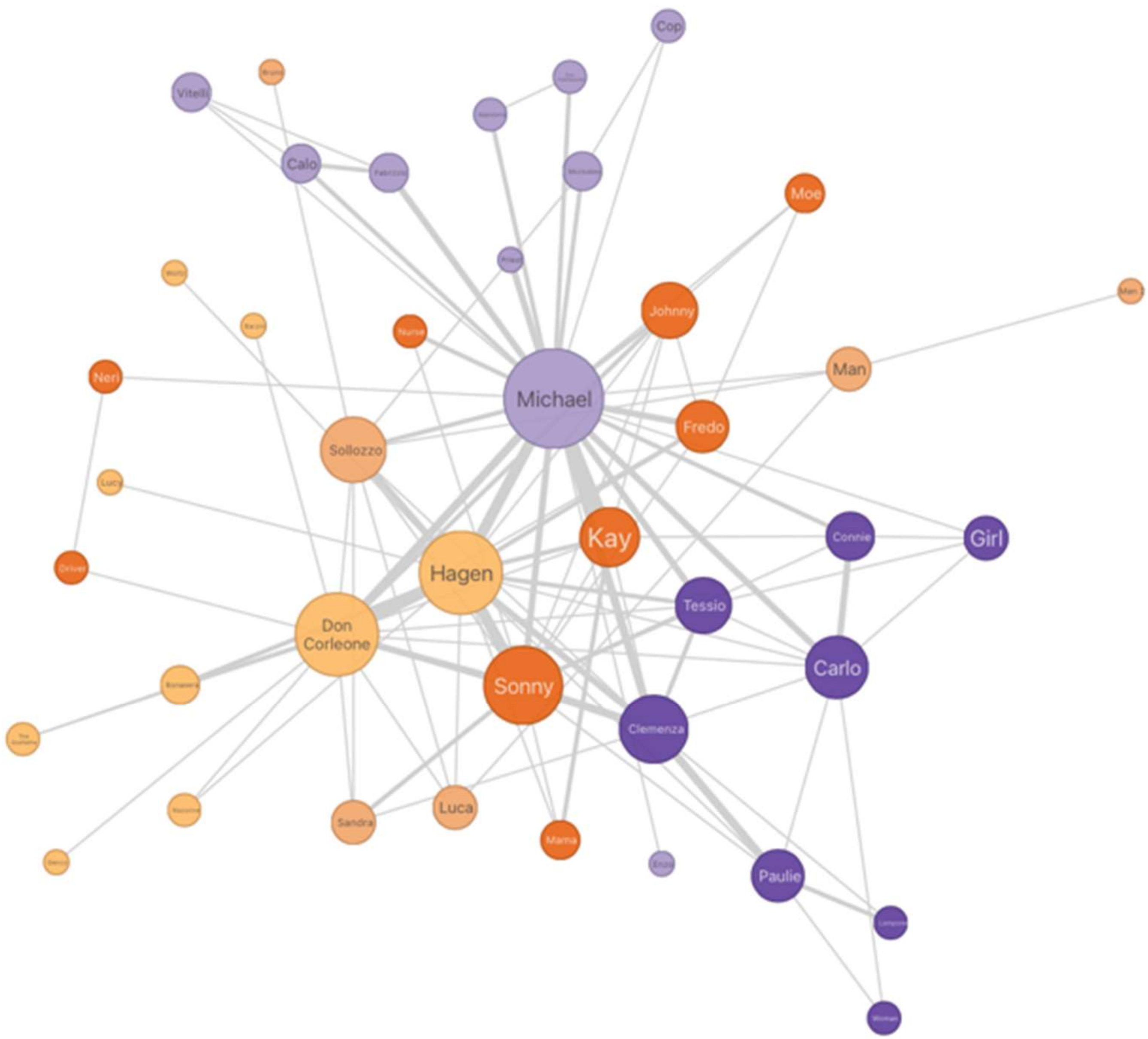
Theoretical Computer Science academic communities



Example from DBLP:
Communities within the co-authors of Christos H. Papadimitriou

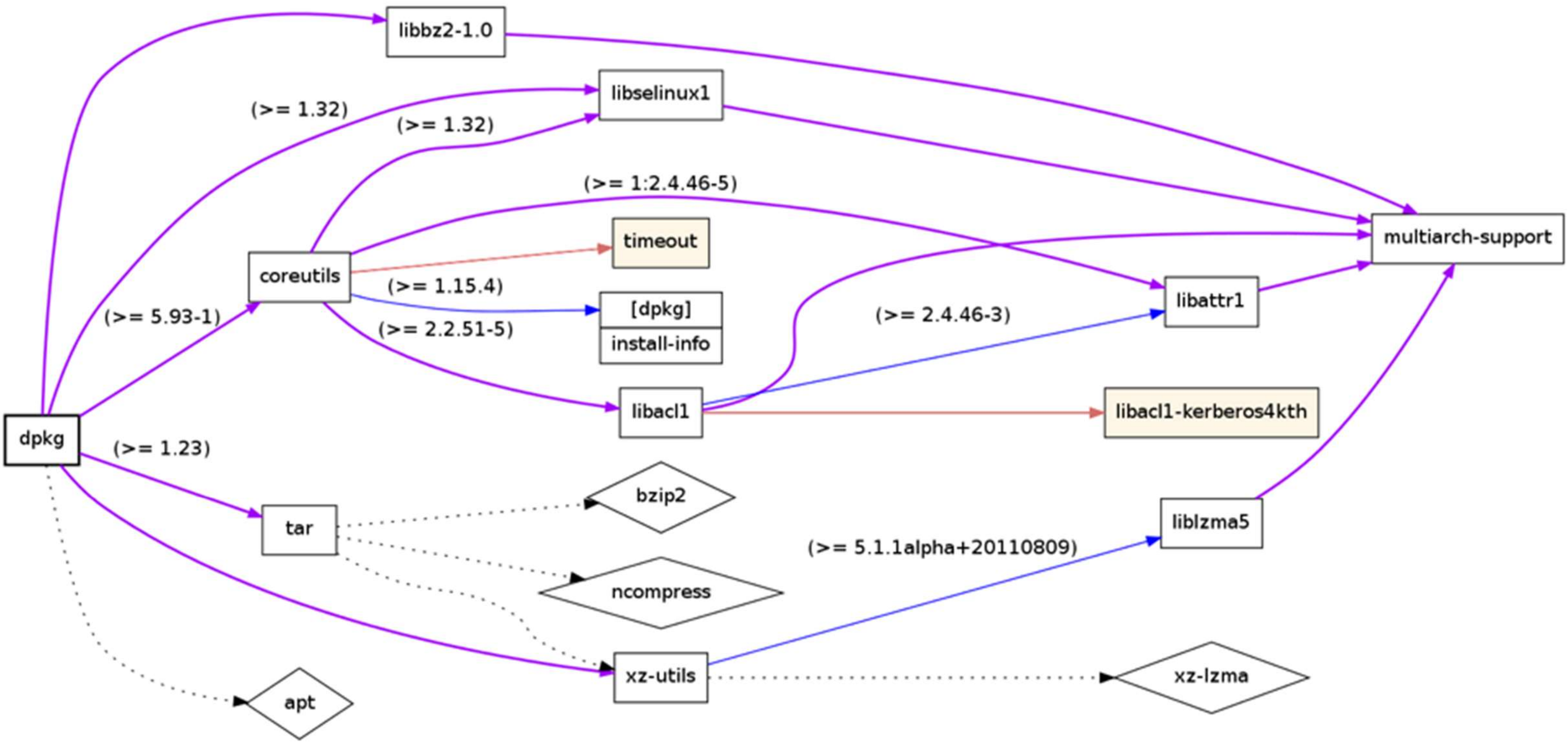
Graphs

The Godfather Characters
Interaction Network



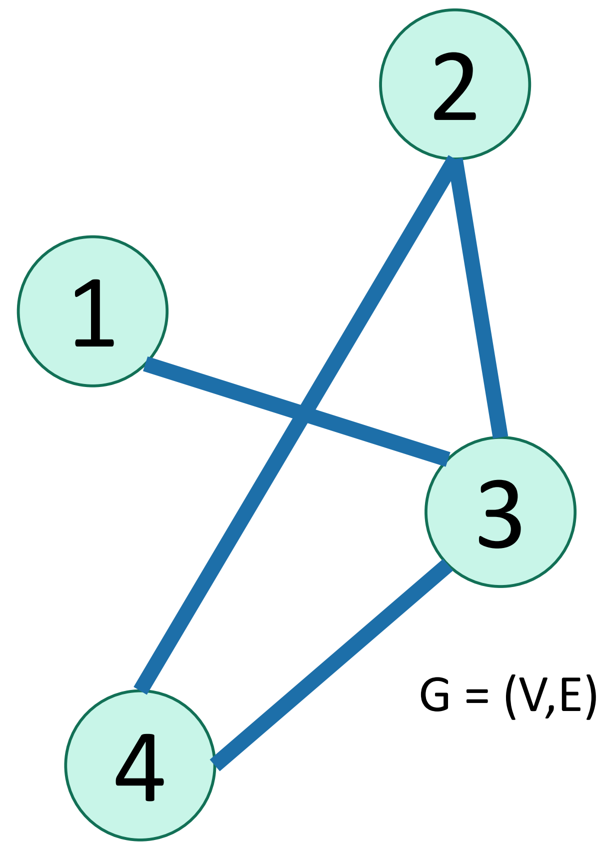
Graphs

debian dependency (sub)graph



Đồ thị vô hướng
Undirected Graphs

- Có đỉnh và cạnh
  - V is the set of vertices
  - E is the set of edges
  - Formally, a graph is G = (V,E)

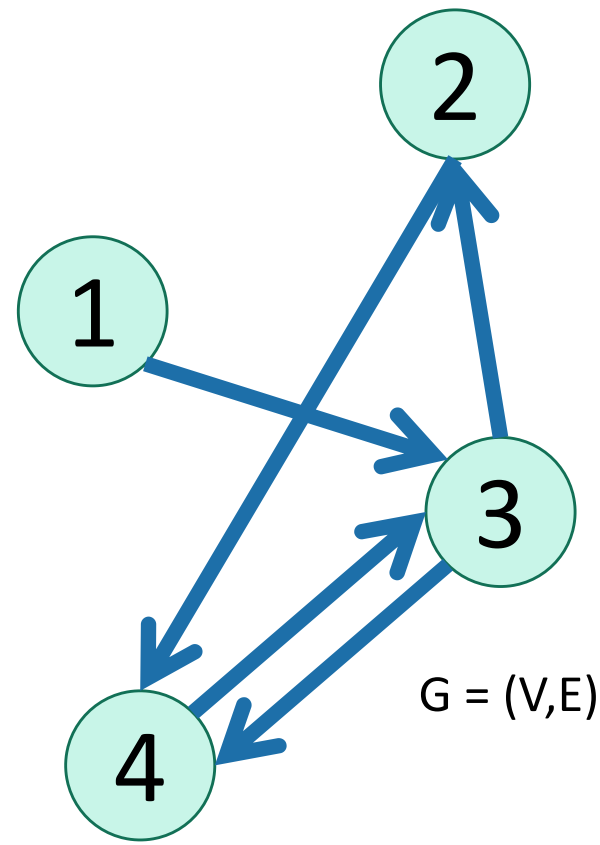


- Ví dụ
  - V = {1,2,3,4}
  - E = { {1,3}, {2,4}, {3,4}, {2,3} }

- The **degree** of vertex 4 is 2.
  - There are 2 edges coming out.
- Vertex 4's **neighbors** are 2 and 3

Đồ thị có hướng
Directed Graphs

- Có đỉnh và cạnh
  - V is the set of vertices
  - E is the set of **DIRECTED** edges
  - Formally, a graph is G = (V,E)



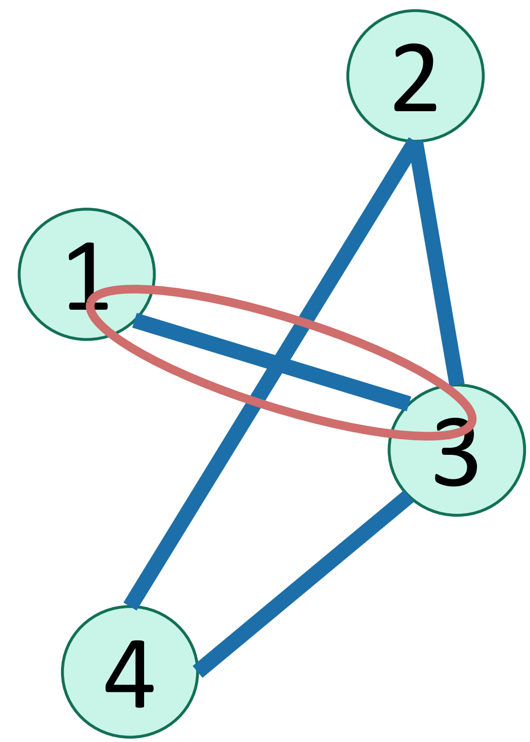
- Ví dụ
  - V = {1,2,3,4}
  - E = { (1,3), (2,4), (3,4), (4,3), (3,2) }

- The **in-degree** of vertex 4 is 2.
- The **out-degree** of vertex 4 is 1.
- Vertex 4's **incoming neighbors** are 2,3
- Vertex 4's **outgoing neighbor** is 3.

Biểu diễn đồ thị như thế nào?

- Option 1: adjacency matrix

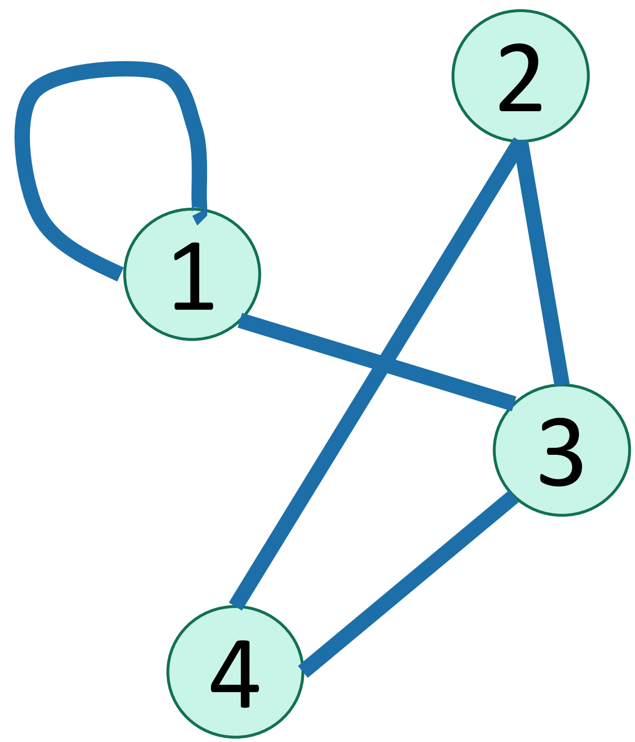
$$
\begin{array}{c c}
 & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}
\end{array}
$$



Biểu diễn đồ thị như thế nào?

- Option 1: adjacency matrix

$$\begin{array}{cc} & \begin{array}{cccc}1 & 2 & 3 & 4\end{array} \\ \begin{array}{c}1\\2\\3\\4\end{array} & \begin{bmatrix}1 & 0 & 1 & 0\\0 & 0 & 1 & 1\\1 & 1 & 0 & 1\\0 & 1 & 1 & 0\end{bmatrix}\end{array}$$

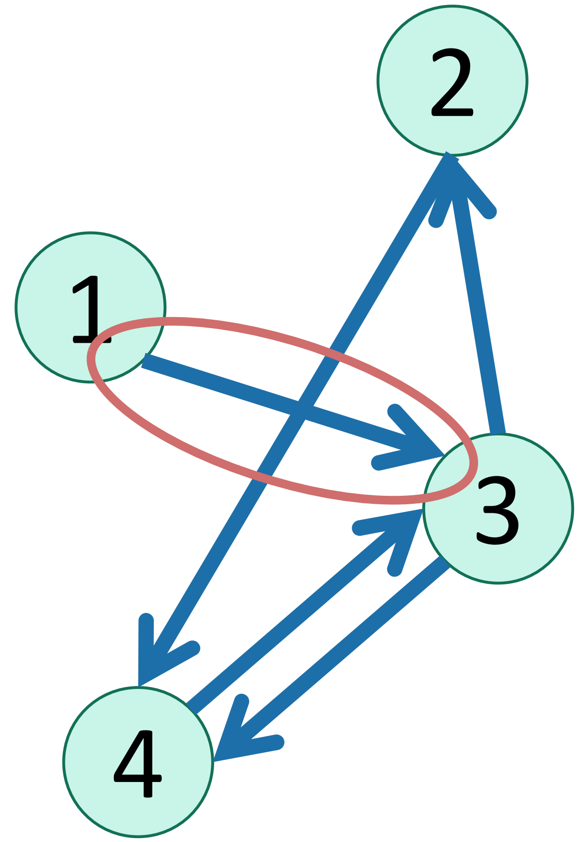


How do we represent graphs?

- Option 1: adjacency matrix

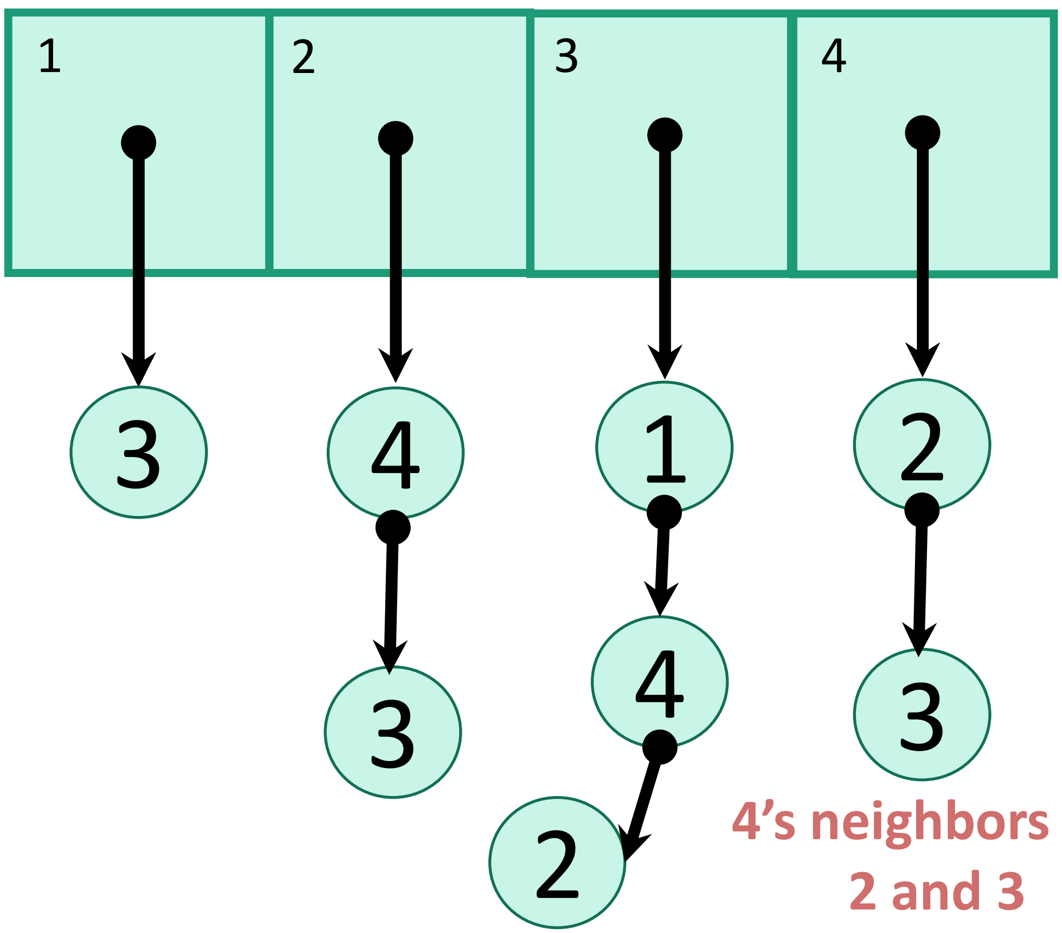
Destination

$$
\begin{array}{cc}
 & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\
\text{Source} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}
\end{array}
$$

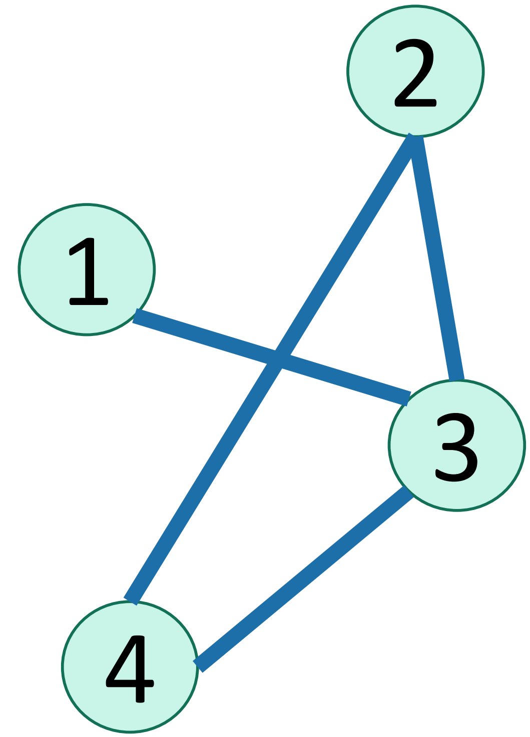


How do we represent graphs?

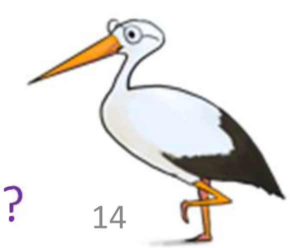
- Option 2: adjacency lists.



4’s neighbors are 2 and 3



How would you modify this for directed graphs?

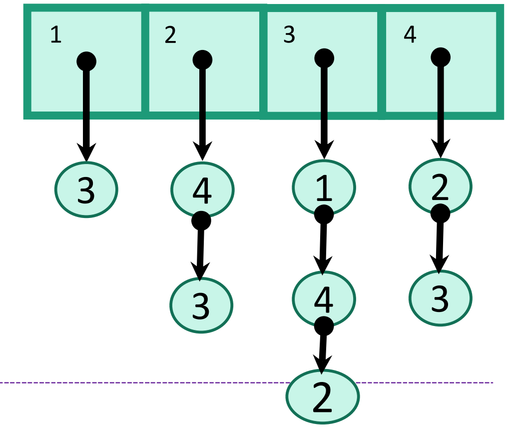


Tính chất chung

- Đỉnh có thể chứa các thông tin
  - Thuộc tính (name, IP address, …)
  - Thông tin bổ trợ cho thuật toán đồ thị (v.d: số đỉnh liền kề,…)

- Có thể thực hiện các thao tác
  - **Edge Membership**: Is edge e in E?
  - **Neighbor Query**: What are the neighbors of vertex v?

So sánh

Giả sử có n đỉnh và m cạnh	$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$	Generally better for **sparse** graphs (where $m \ll n^2$) 
Edge membership Is e = {v,w} in E?	O(1)	O(deg(v)) or O(deg(w))
Neighbor query Give me a list of v's neighbors.	O(n)	O(deg(v))
Space requirements	$O(n^2)$	O(n + m)

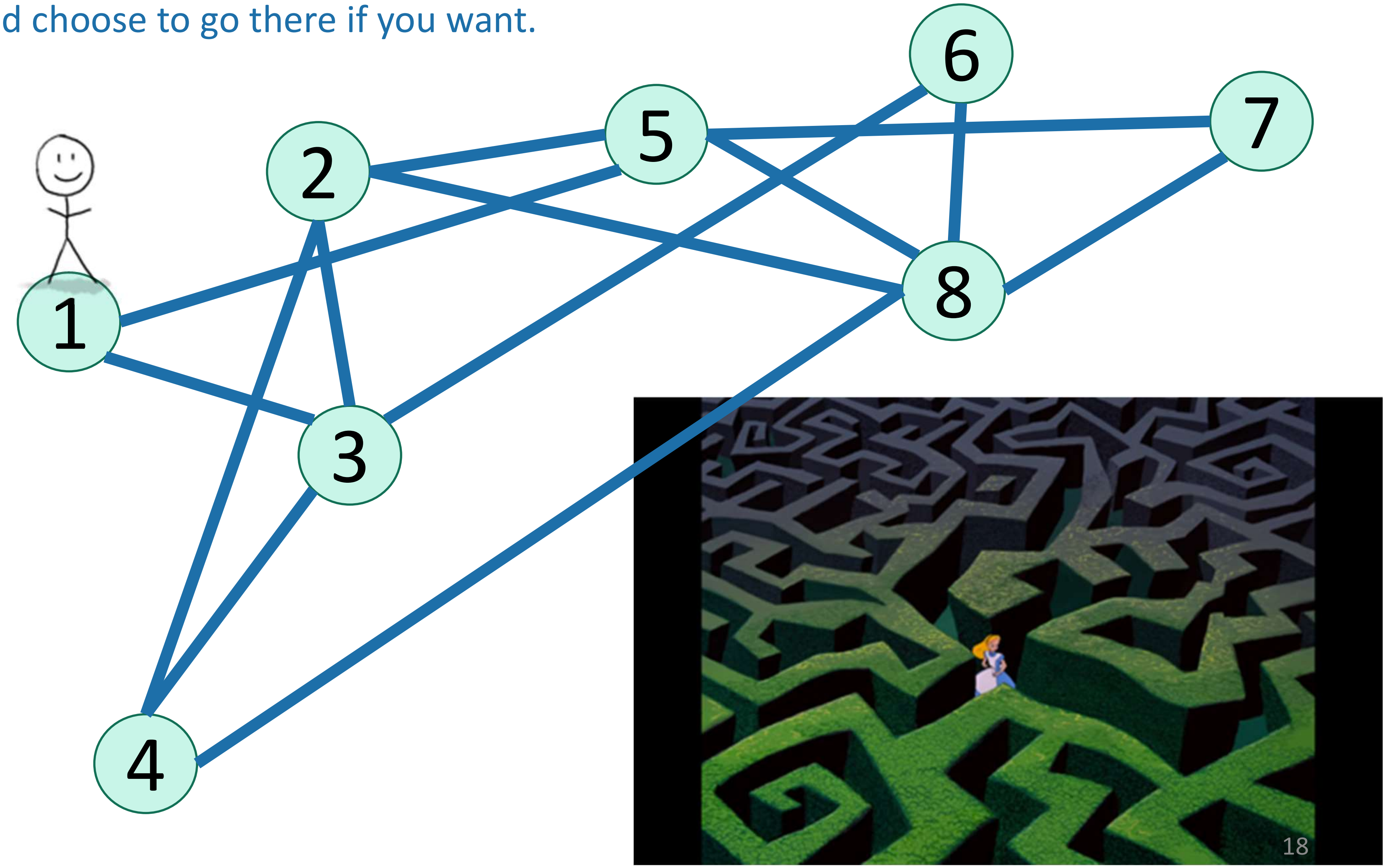
We'll assume this representation for the rest of the class

Phần 2: Depth-first search

Khám phá đồ thị như thế nào?

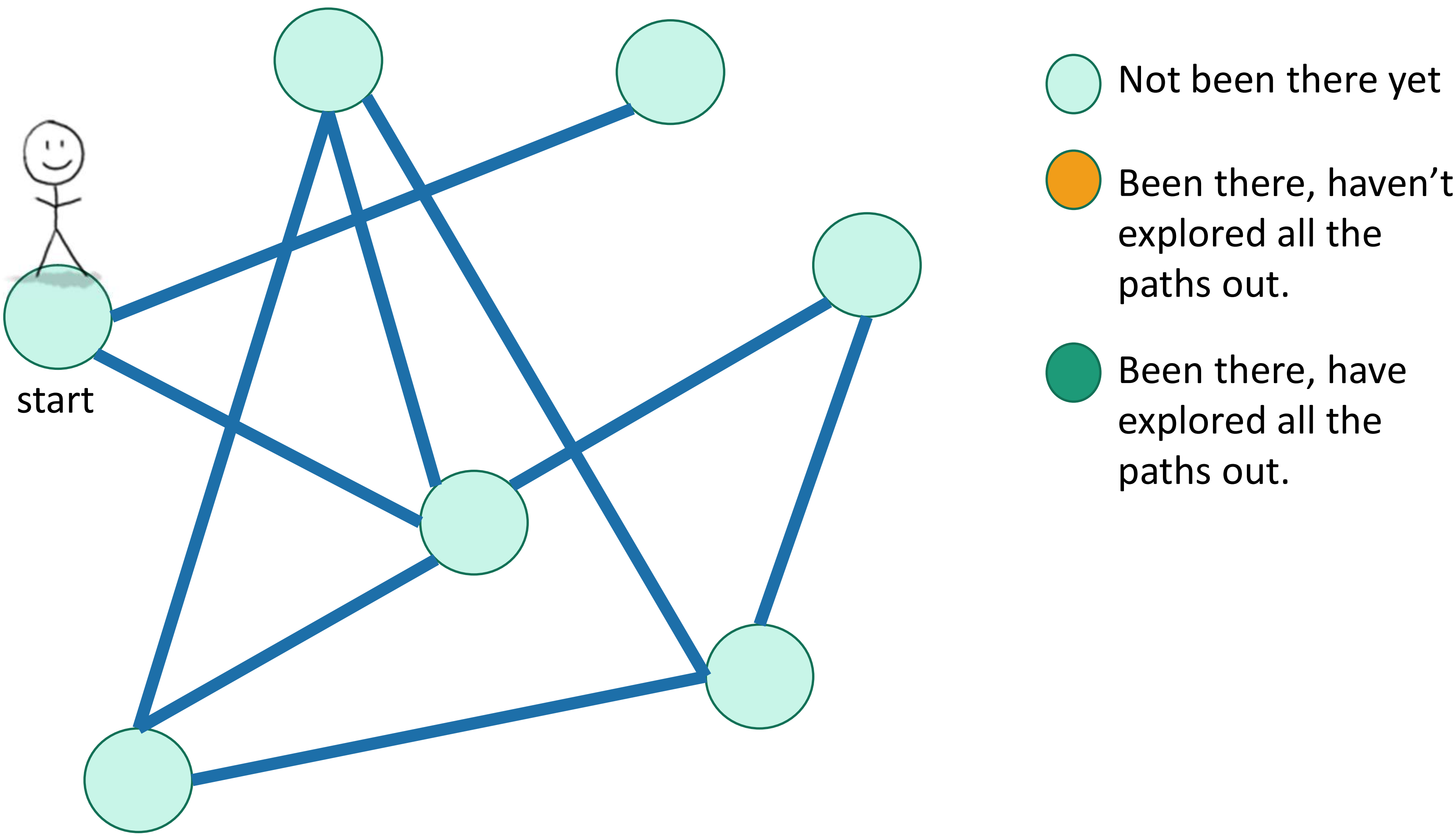
How do we explore a graph?

At each node, you can get a list of neighbors, and choose to go there if you want.



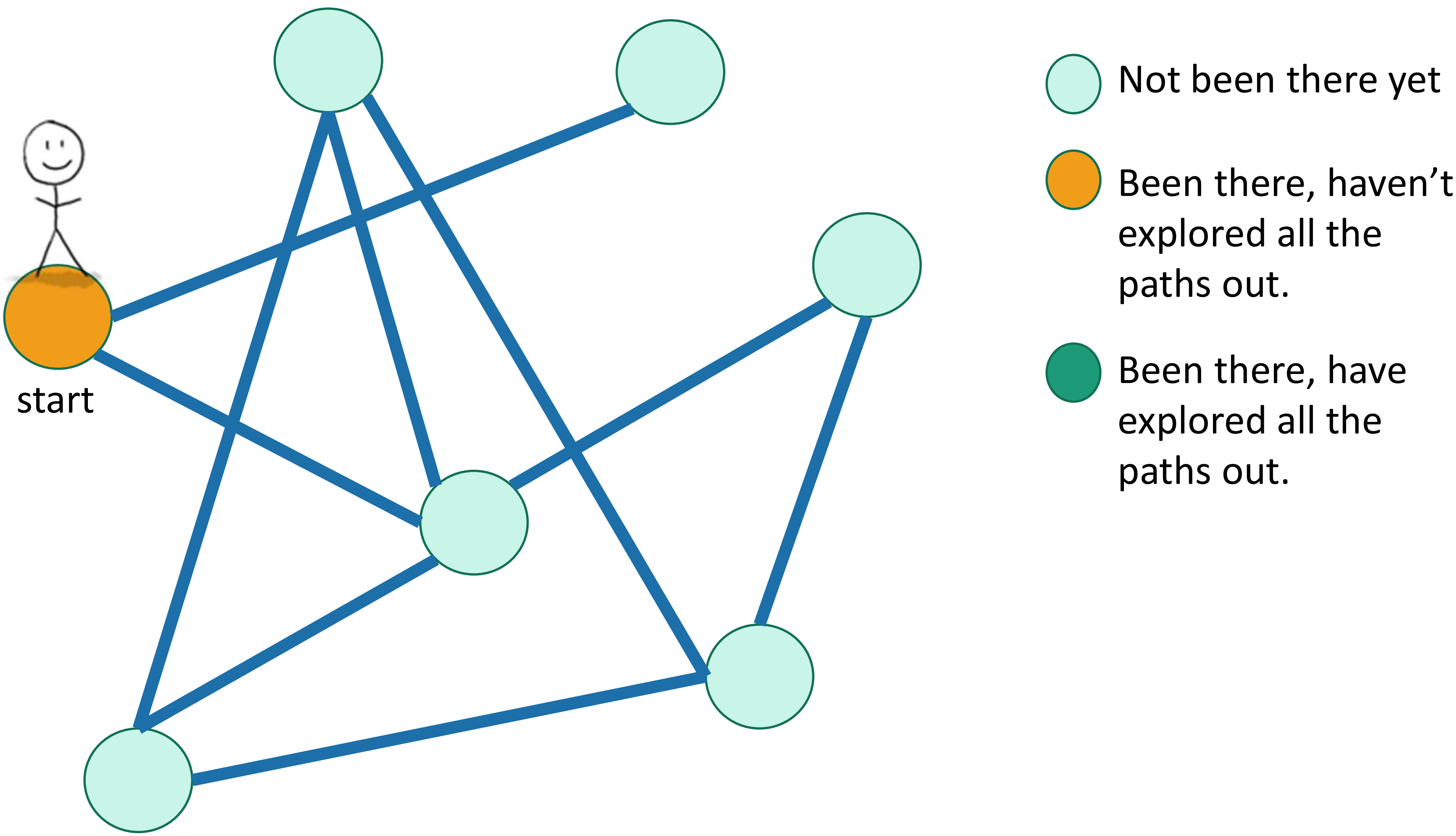
Depth First Search

Exploring a labyrinth with chalk and a piece of string



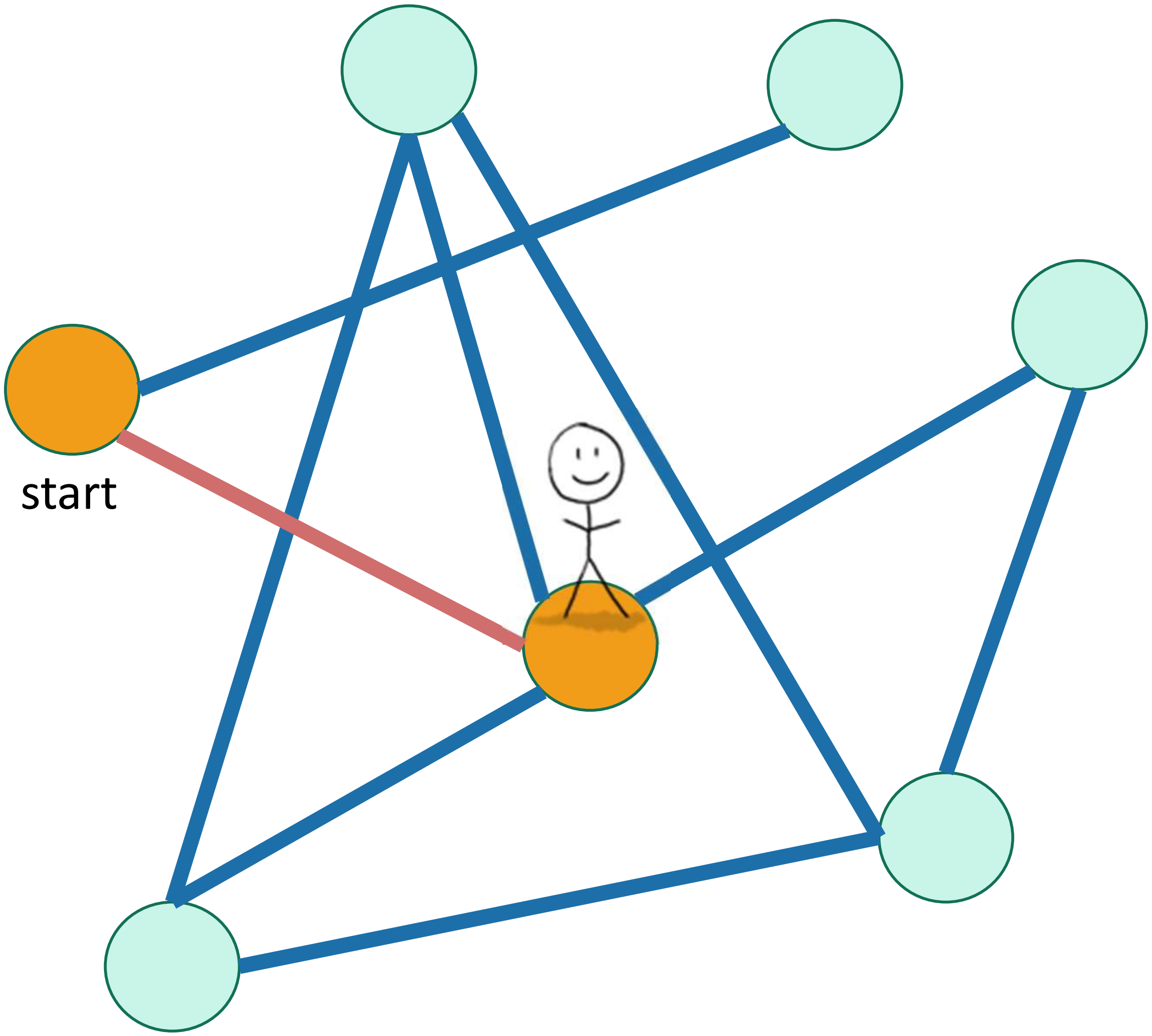
Depth First Search

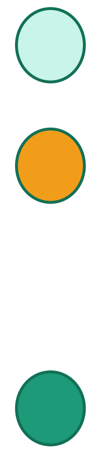
Exploring a labyrinth with chalk and a piece of string



Depth First Search

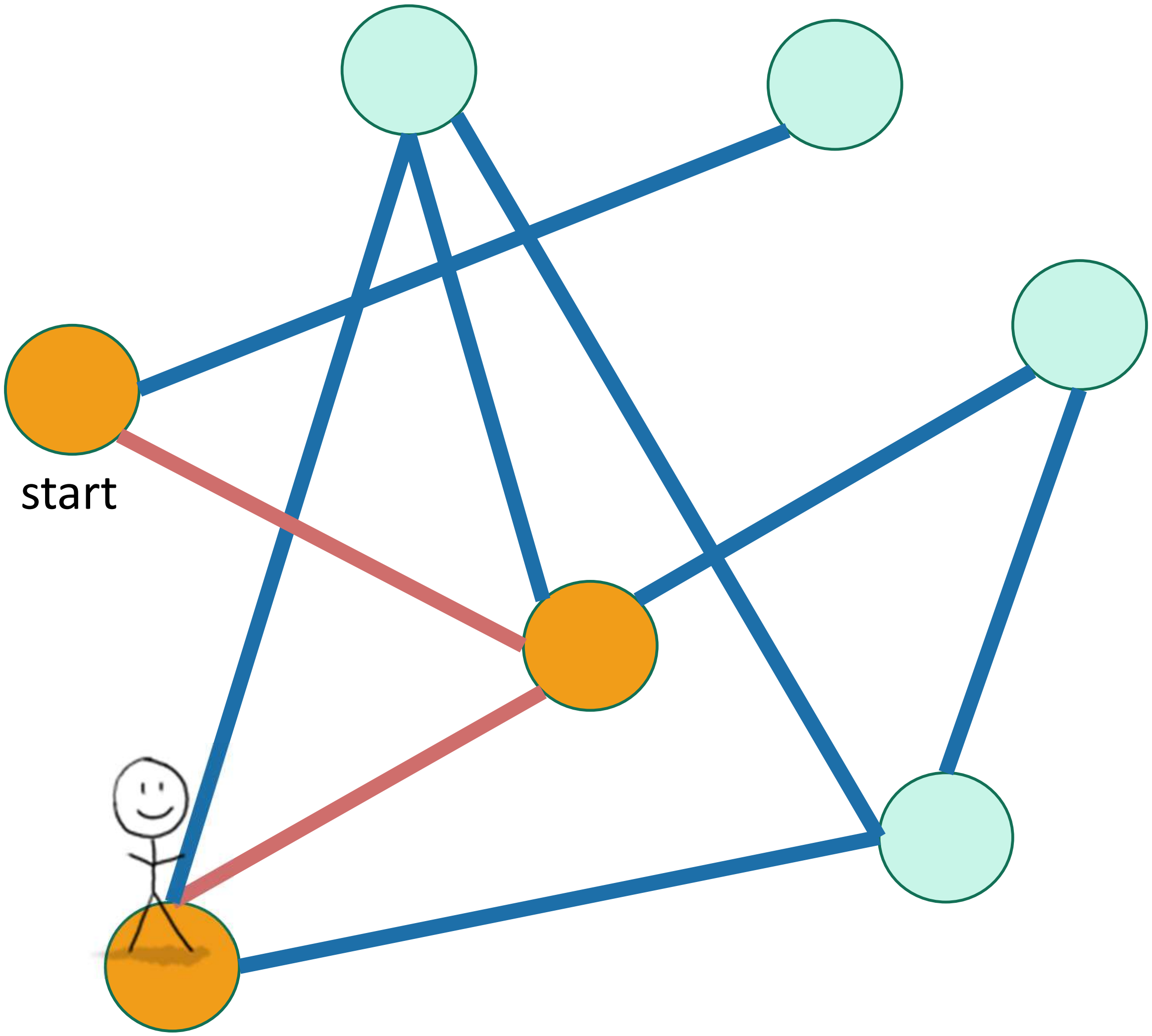
Exploring a labyrinth with chalk and a piece of string





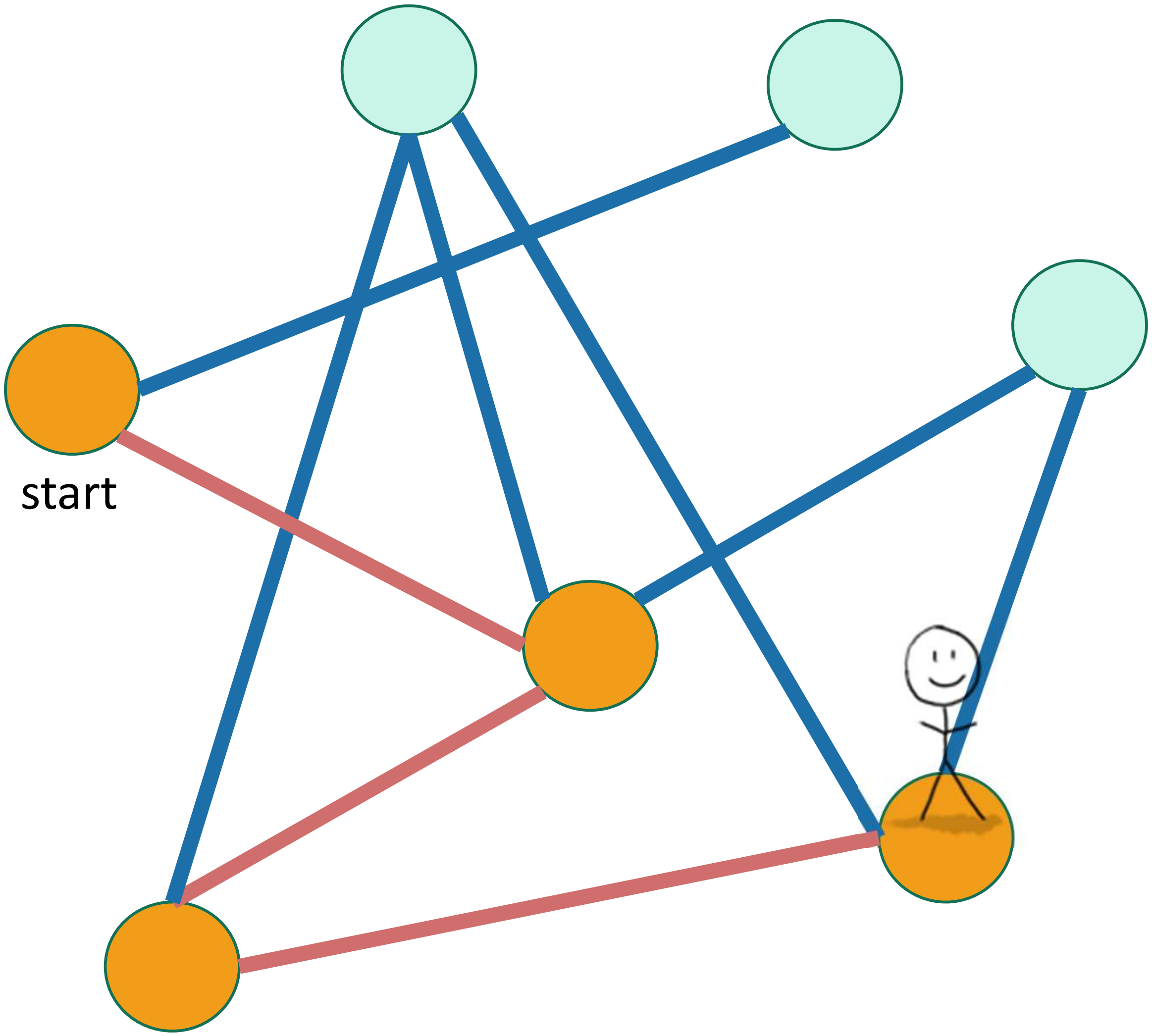
Depth First Search

Exploring a labyrinth with chalk and a piece of string



Depth First Search

Exploring a labyrinth with chalk and a piece of string



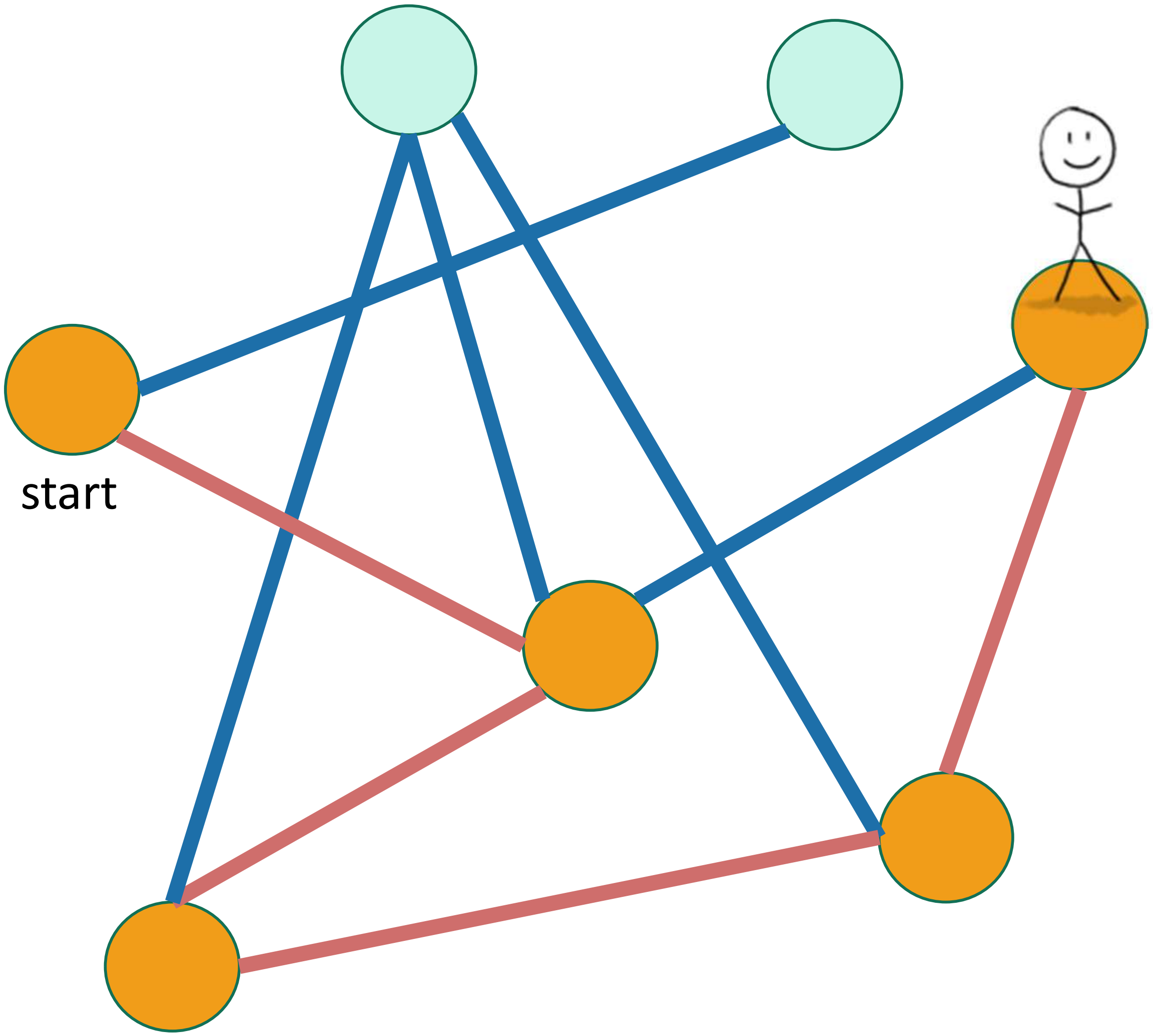
Not been there yet

Been there, haven't explored all the paths out.

Been there, have explored all the paths out.

Depth First Search

Exploring a labyrinth with chalk and a piece of string



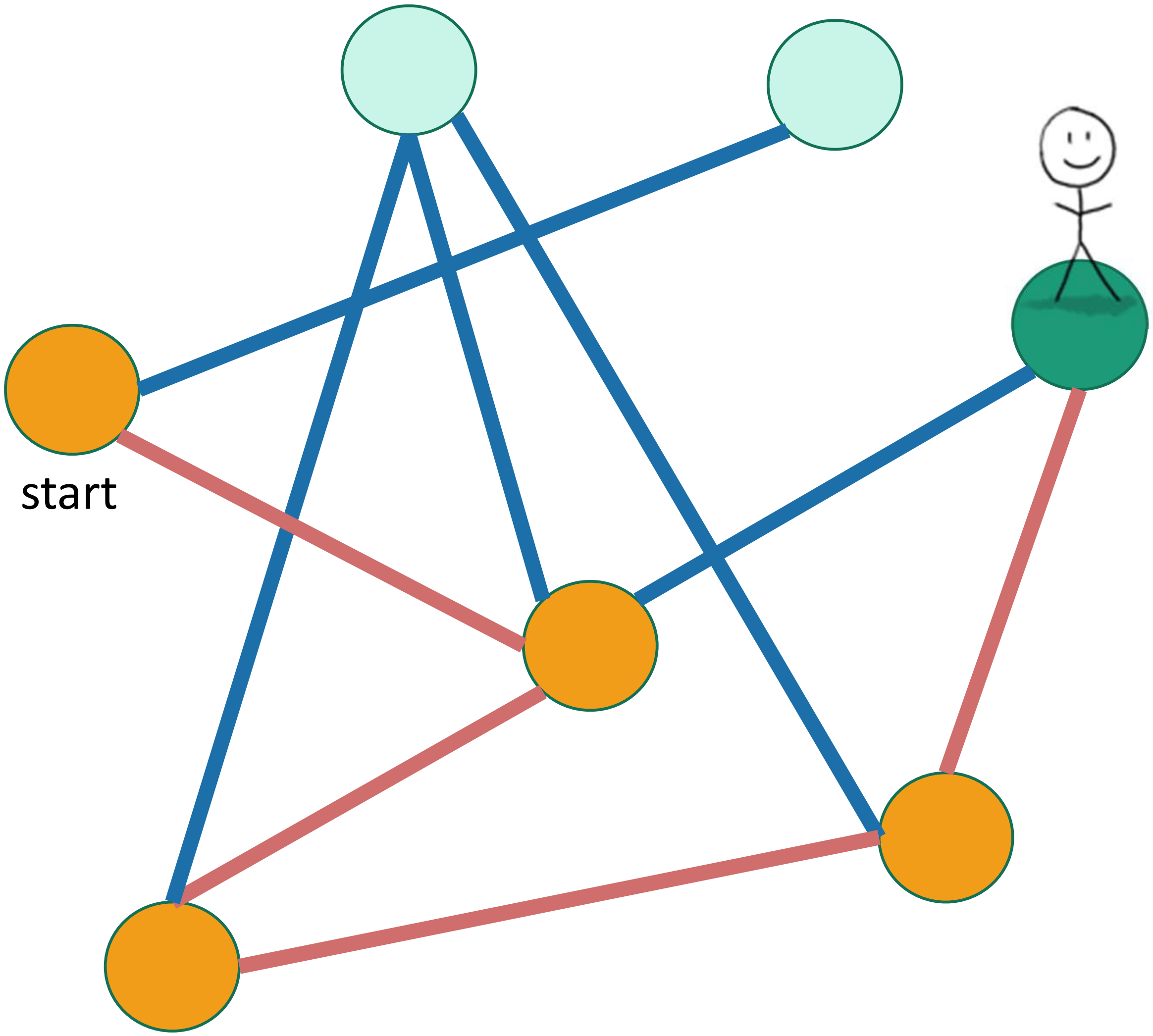
Not been there yet

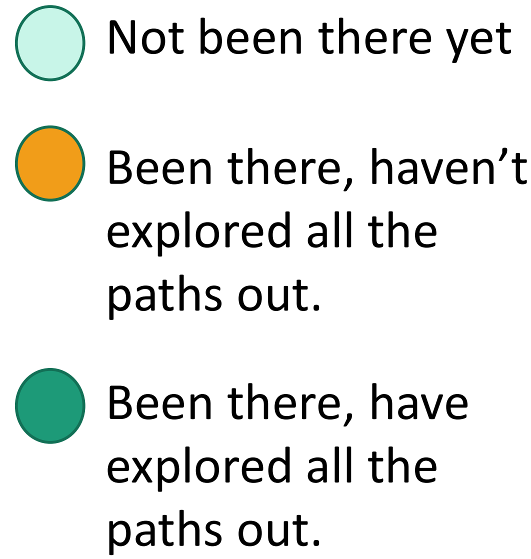
Been there, haven't explored all the paths out.

Been there, have explored all the paths out.

Depth First Search

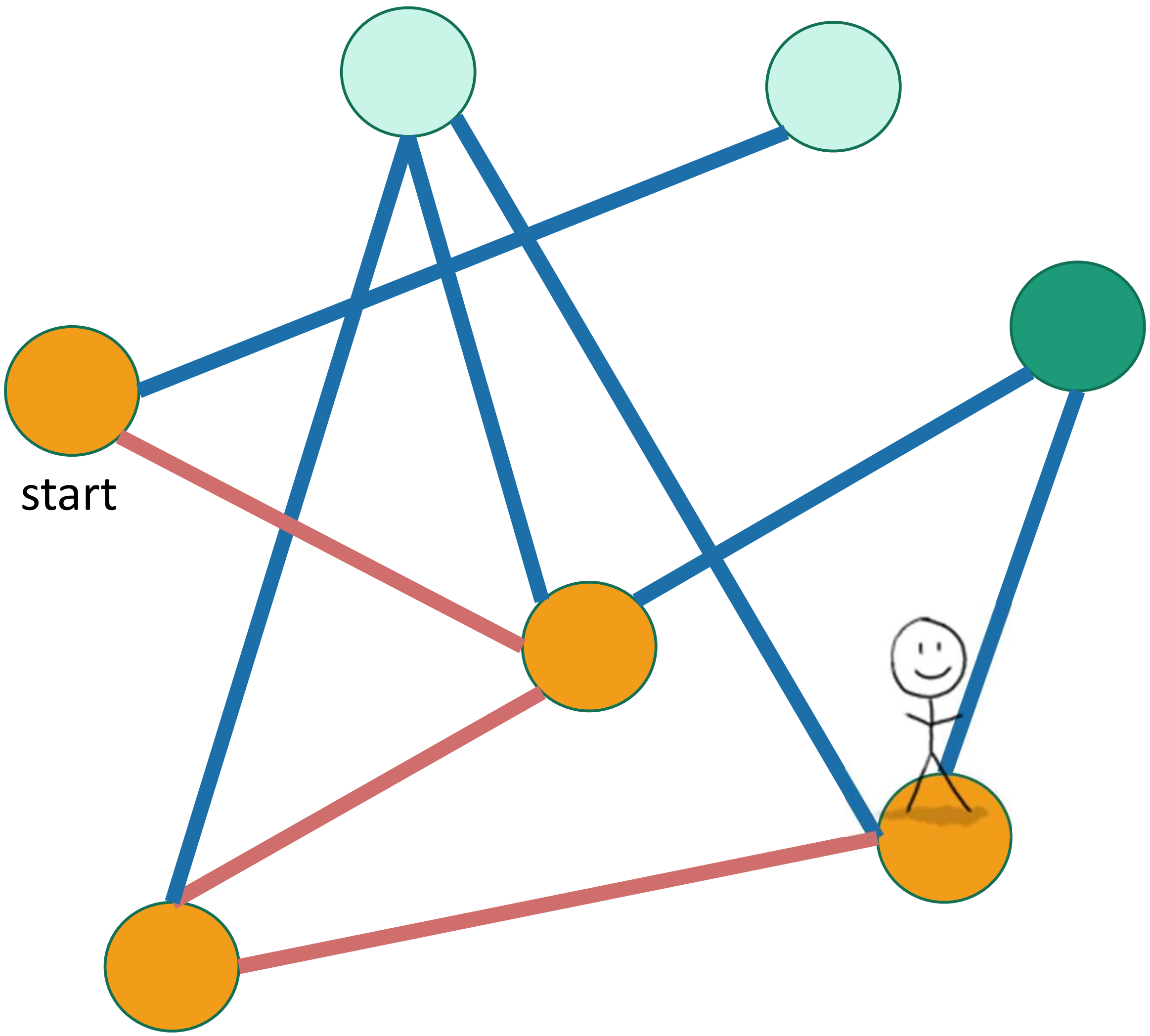
Exploring a labyrinth with chalk and a piece of string





Depth First Search

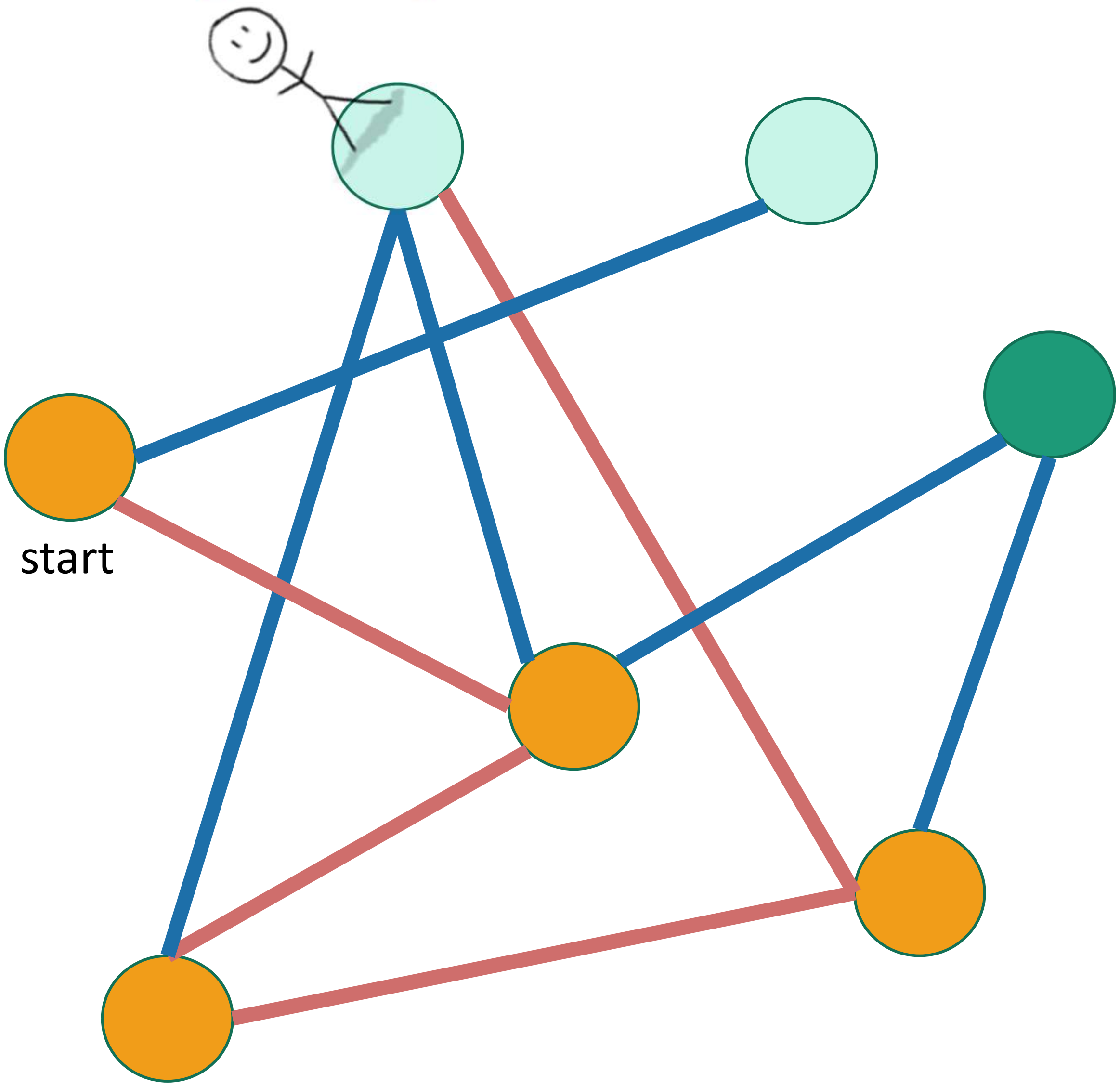
Exploring a labyrinth with chalk and a piece of string

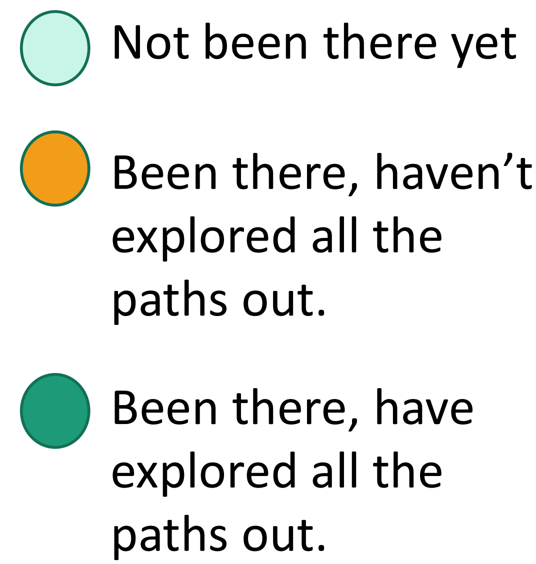


- Not been there yet
- Been there, haven't explored all the paths out.
- Been there, have explored all the paths out.

Depth First Search

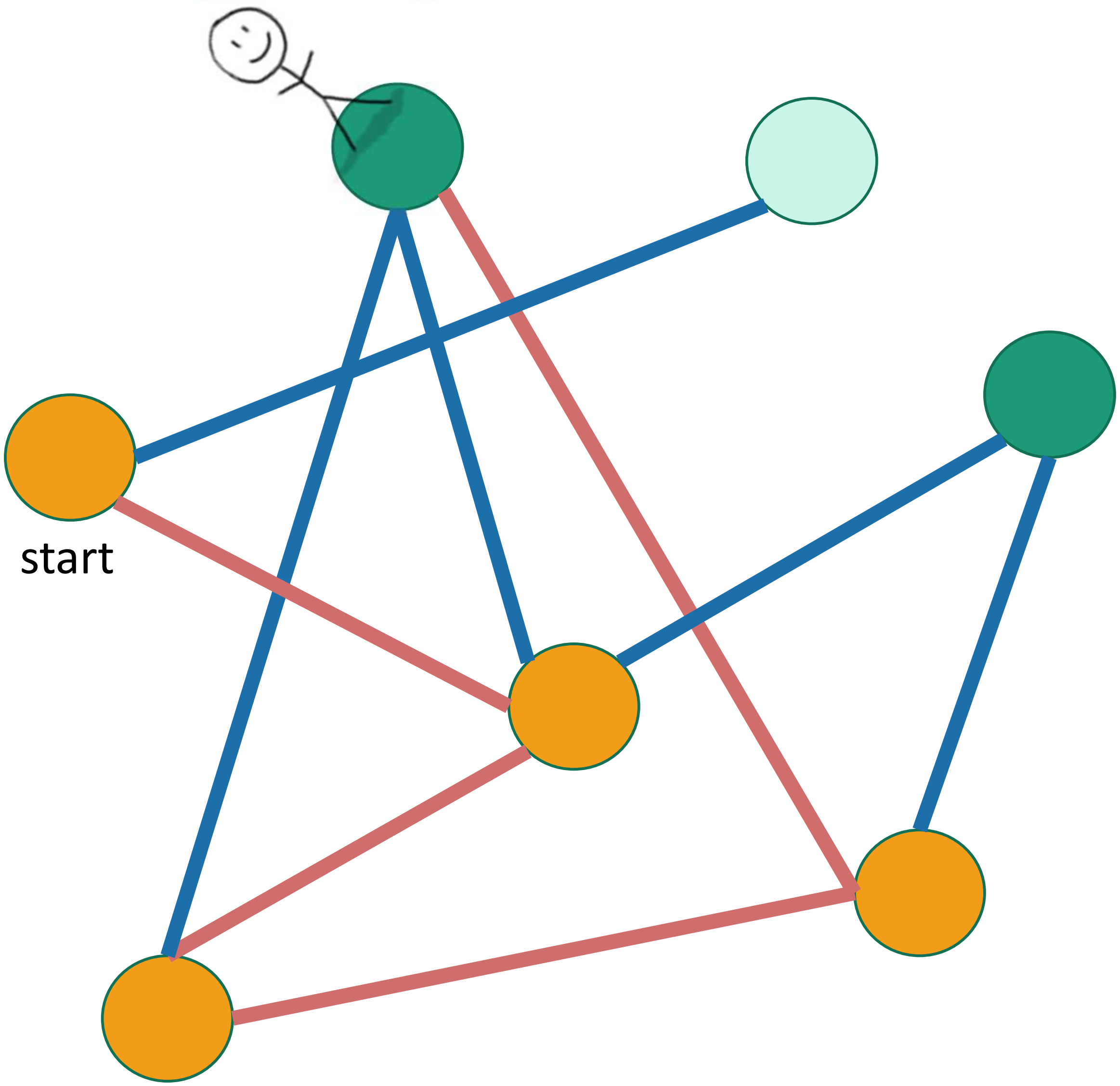
Exploring a labyrinth with chalk and a piece of string





Depth First Search

Exploring a labyrinth with chalk and a piece of string



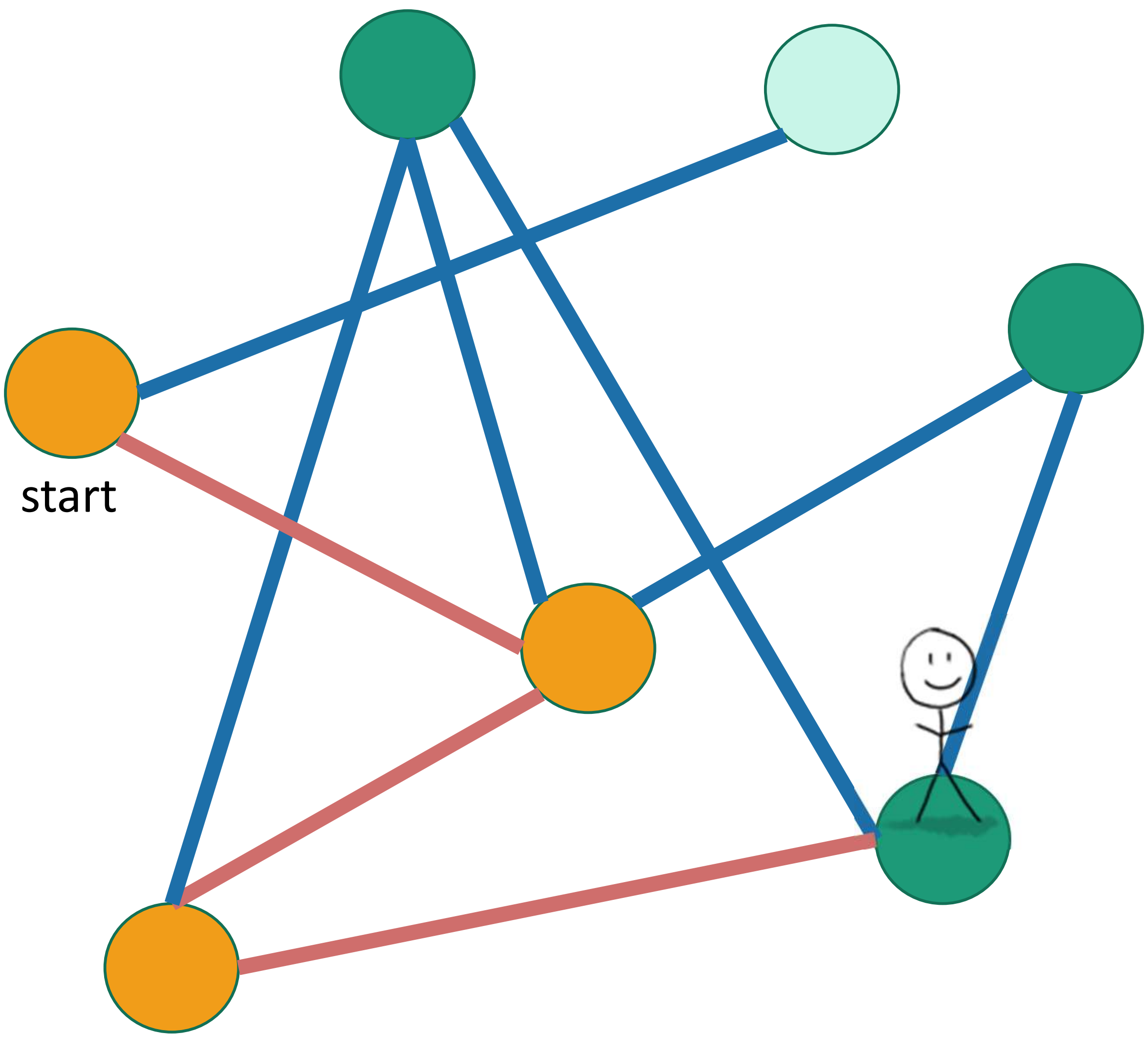
Not been there yet

Been there, haven't explored all the paths out.

Been there, have explored all the paths out.

Depth First Search

Exploring a labyrinth with chalk and a piece of string



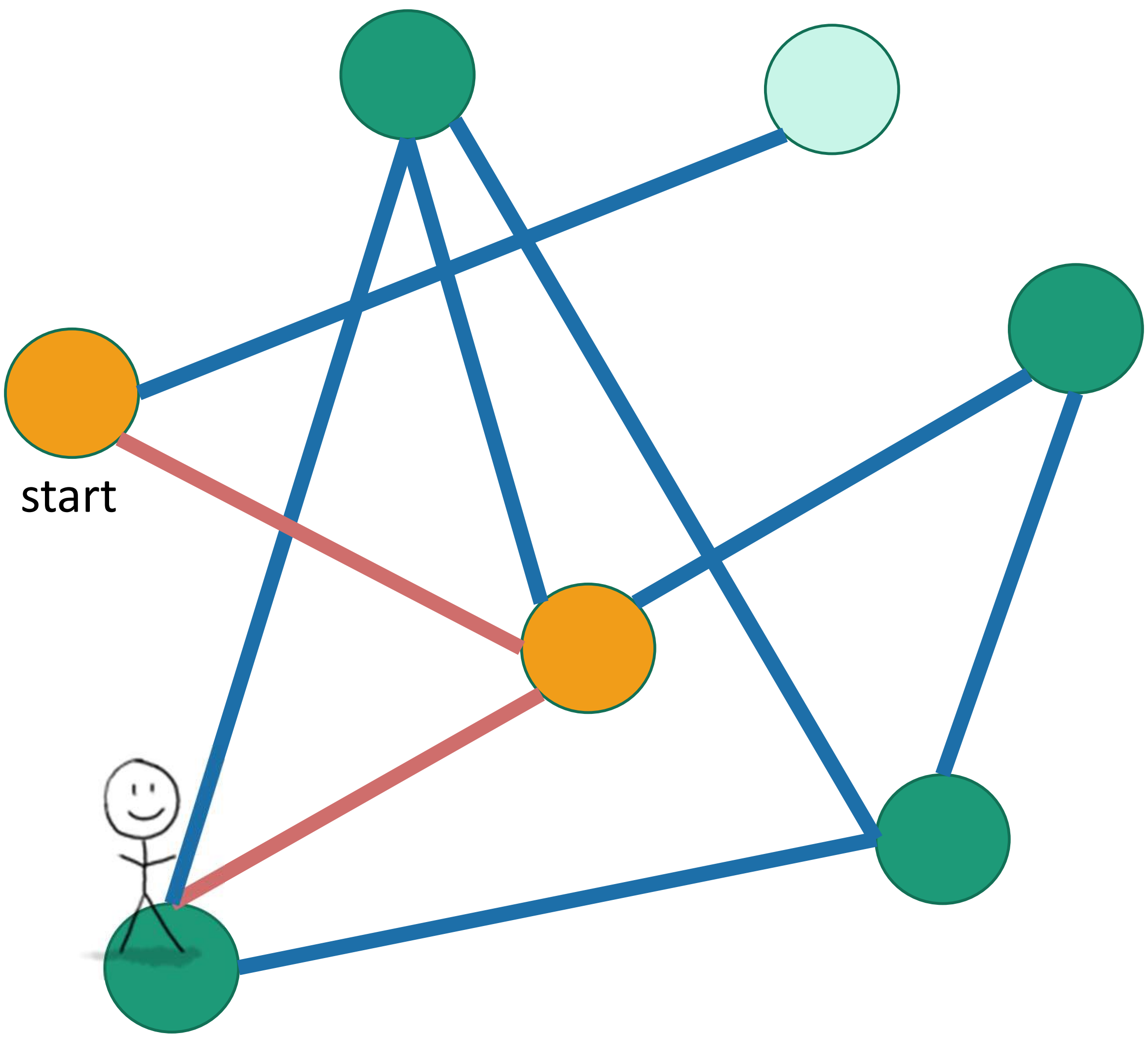
Not been there yet

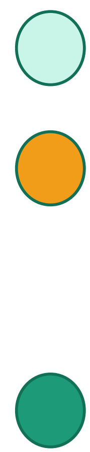
Been there, haven't explored all the paths out.

Been there, have explored all the paths out.

Depth First Search

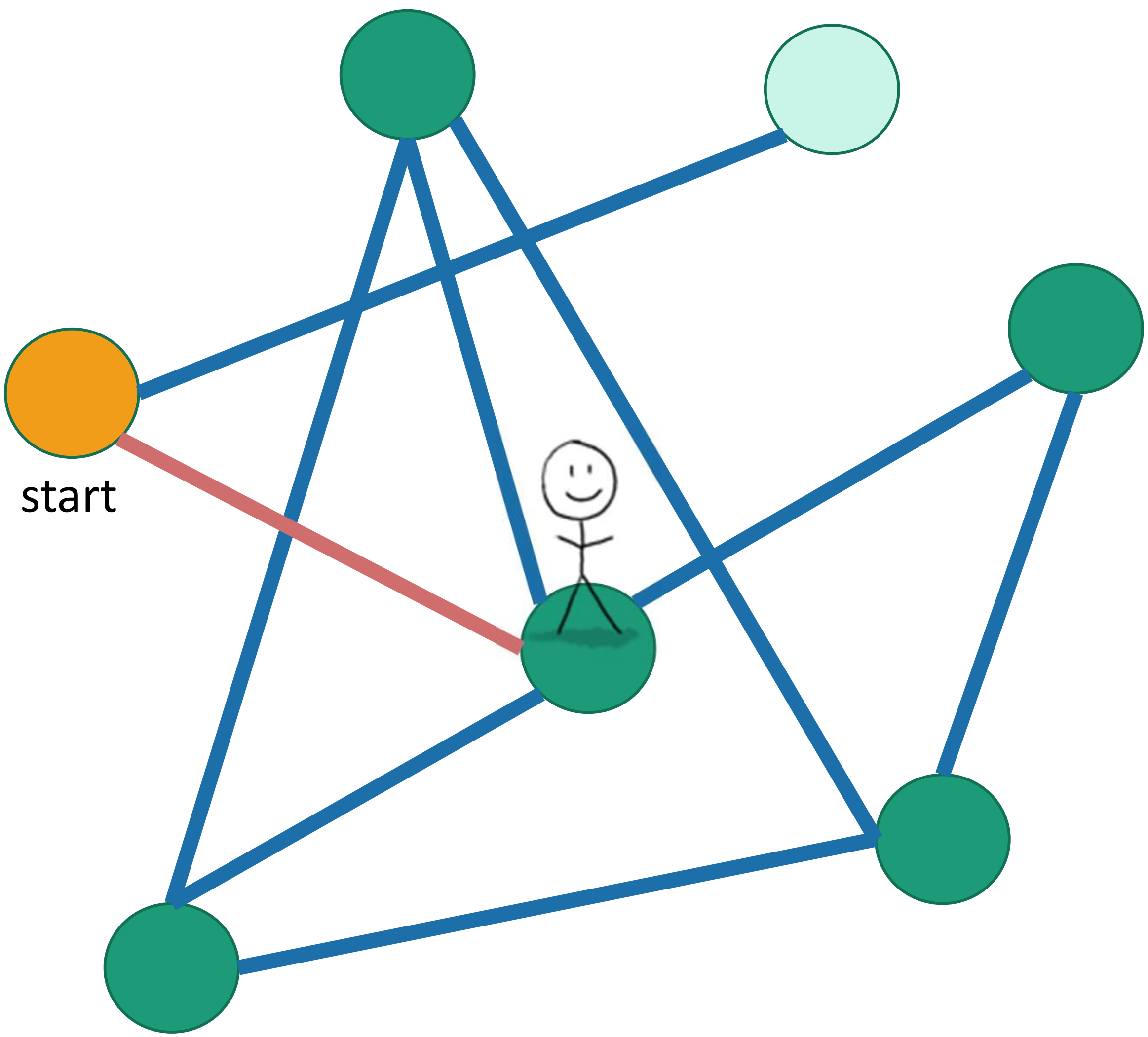
Exploring a labyrinth with chalk and a piece of string





Depth First Search

Exploring a labyrinth with chalk and a piece of string



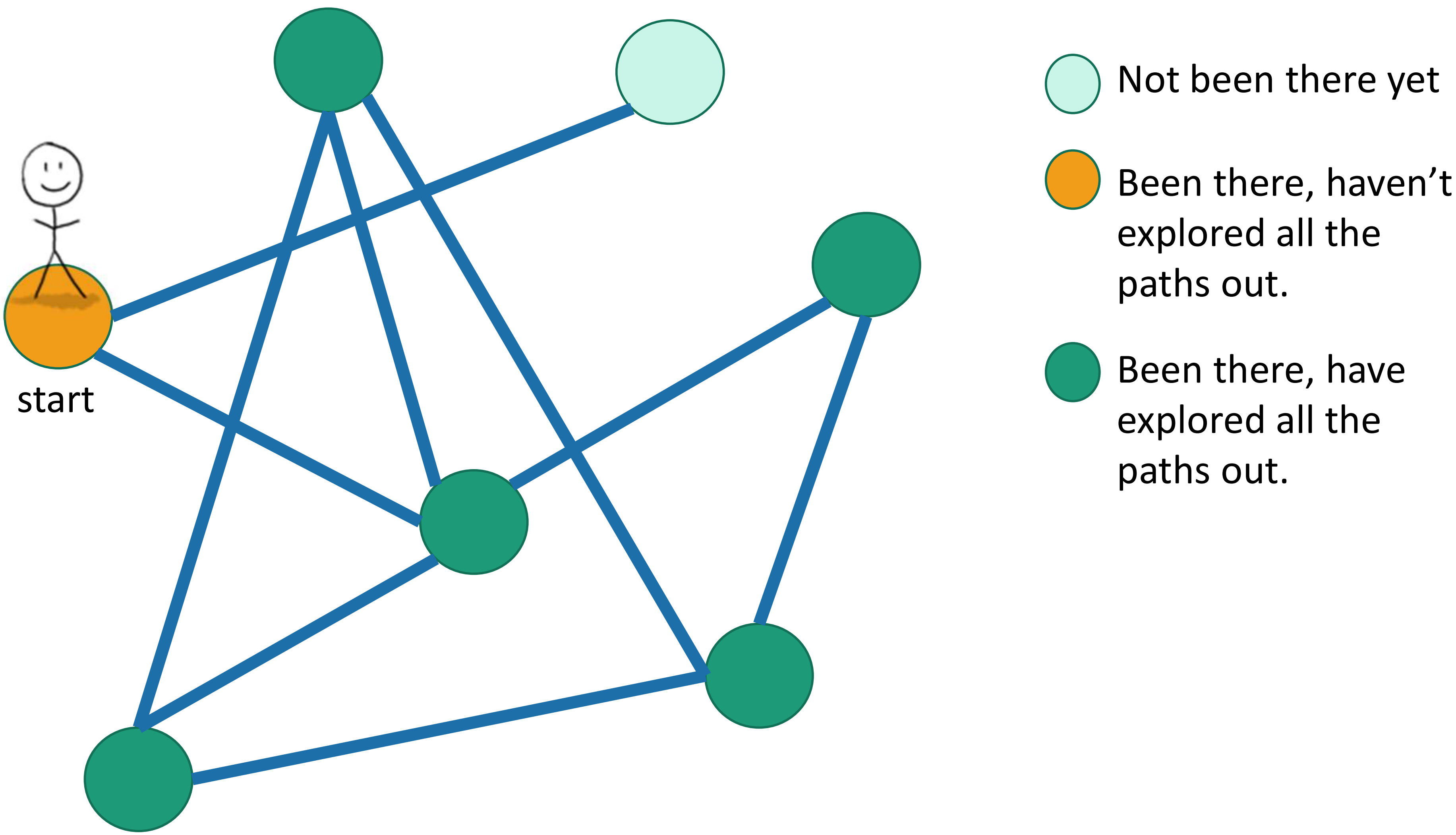
Not been there yet

Been there, haven't explored all the paths out.

Been there, have explored all the paths out.

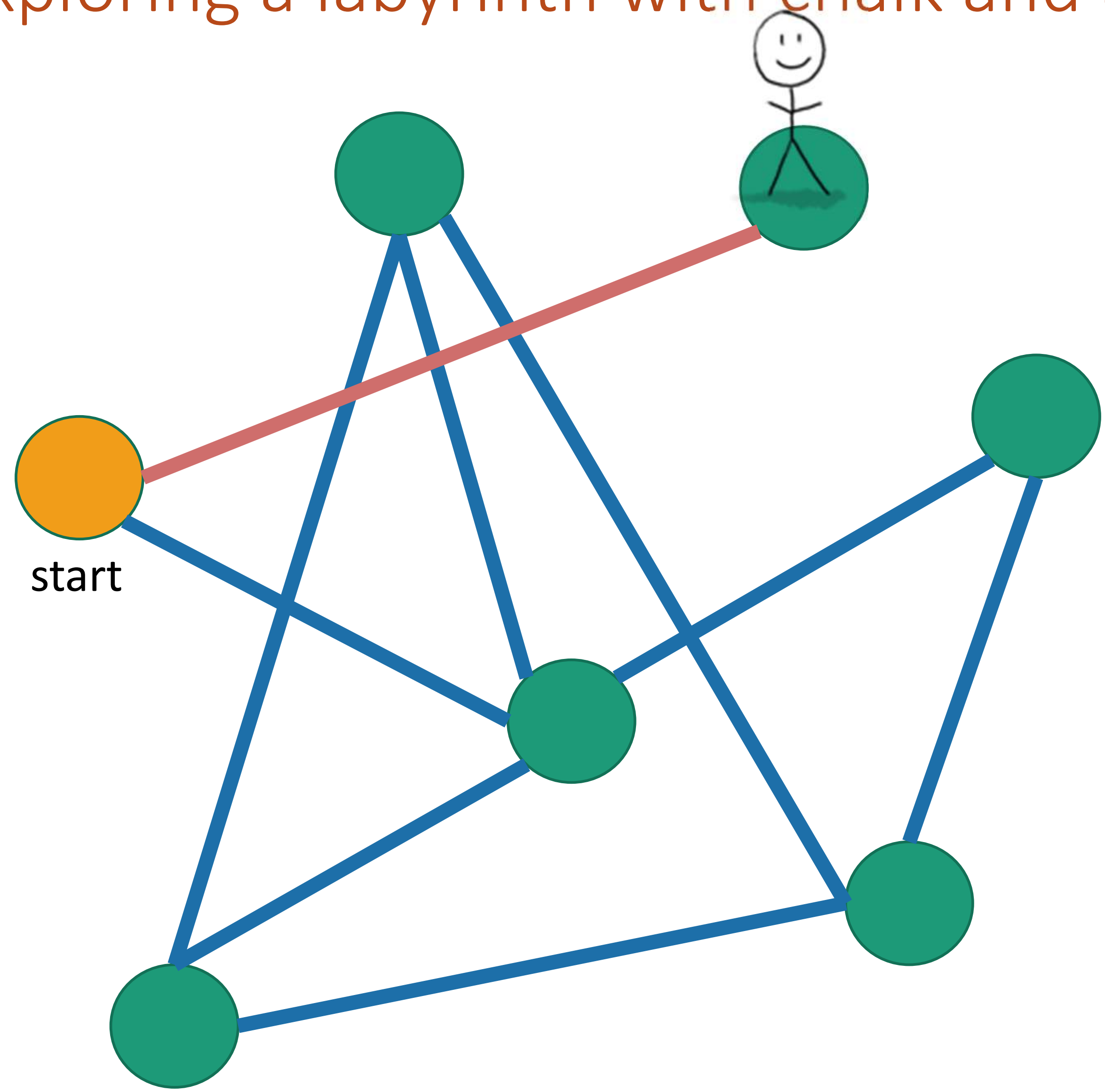
Depth First Search

Exploring a labyrinth with chalk and a piece of string



Depth First Search

Exploring a labyrinth with chalk and a piece of string



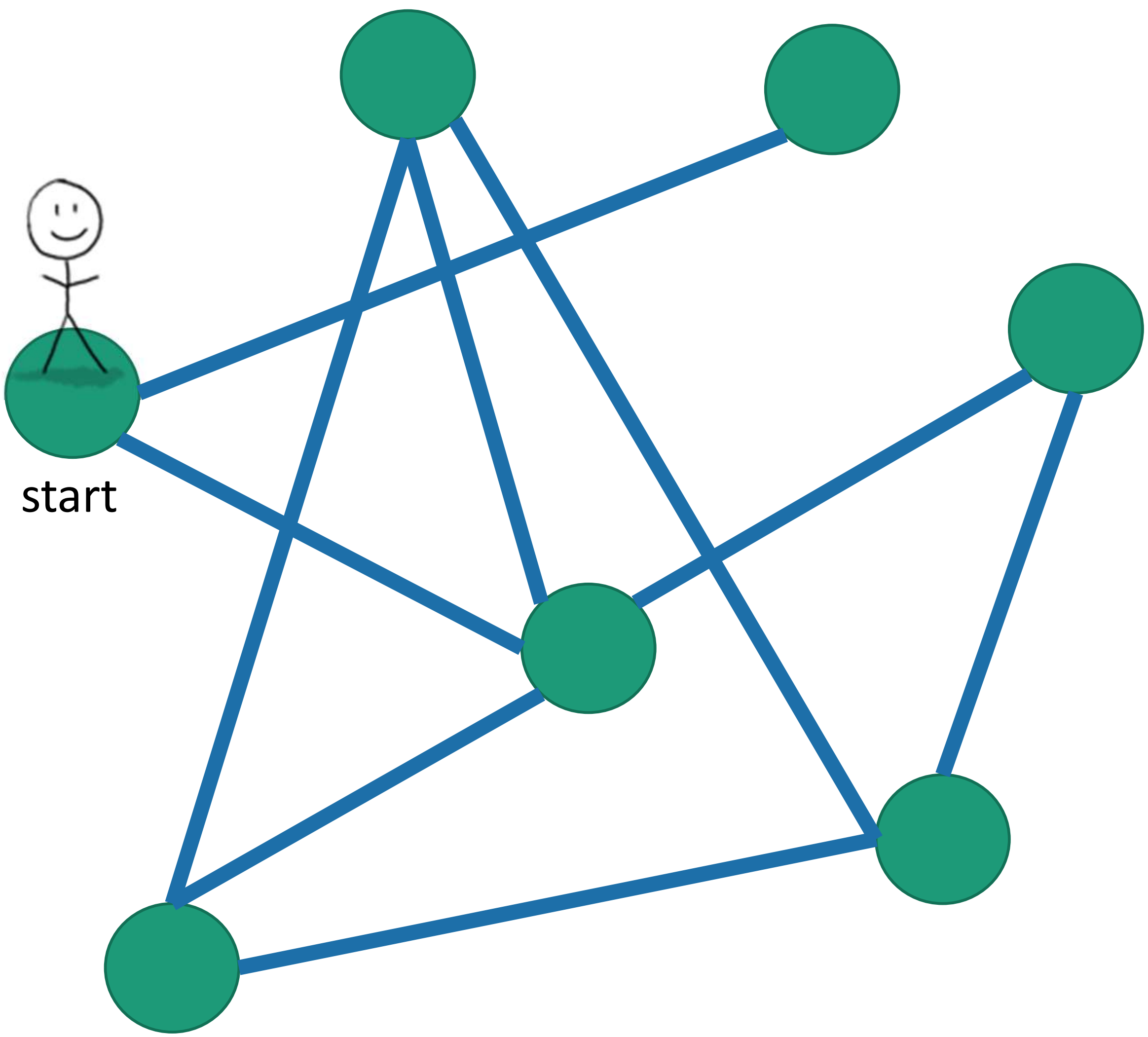
Not been there yet

Been there, haven't explored all the paths out.

Been there, have explored all the paths out.

Depth First Search

Exploring a labyrinth with chalk and a piece of string



Not been there yet

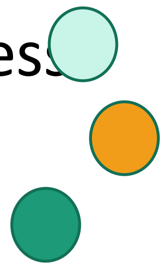
Been there, haven't explored all the paths out.

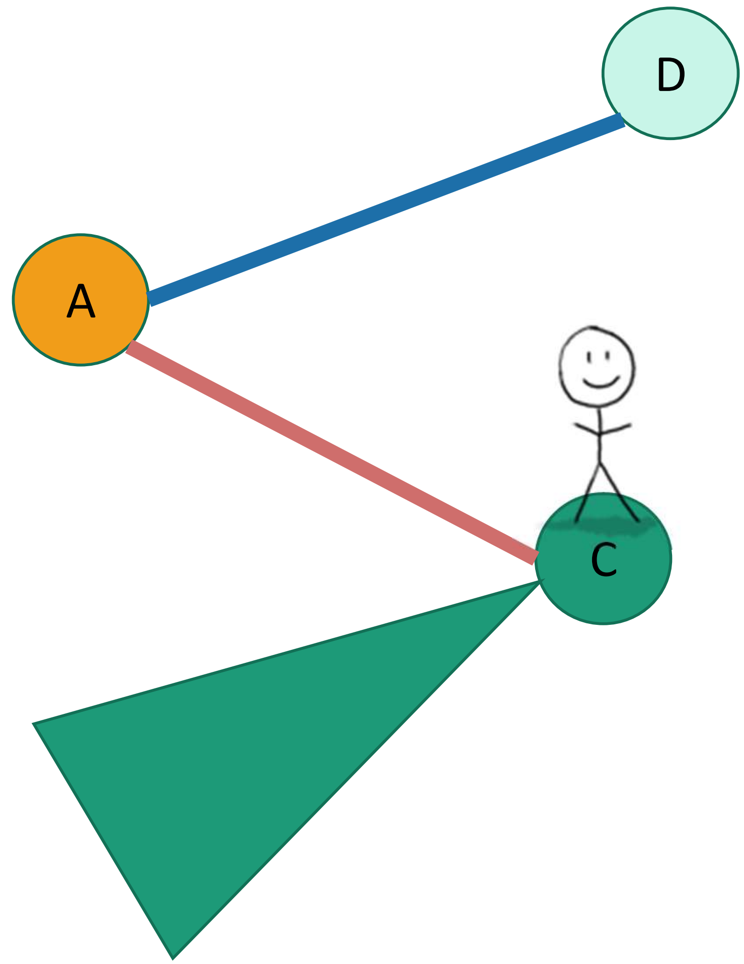
Been there, have explored all the paths out.

Labyrinth: explored!

Depth First Search

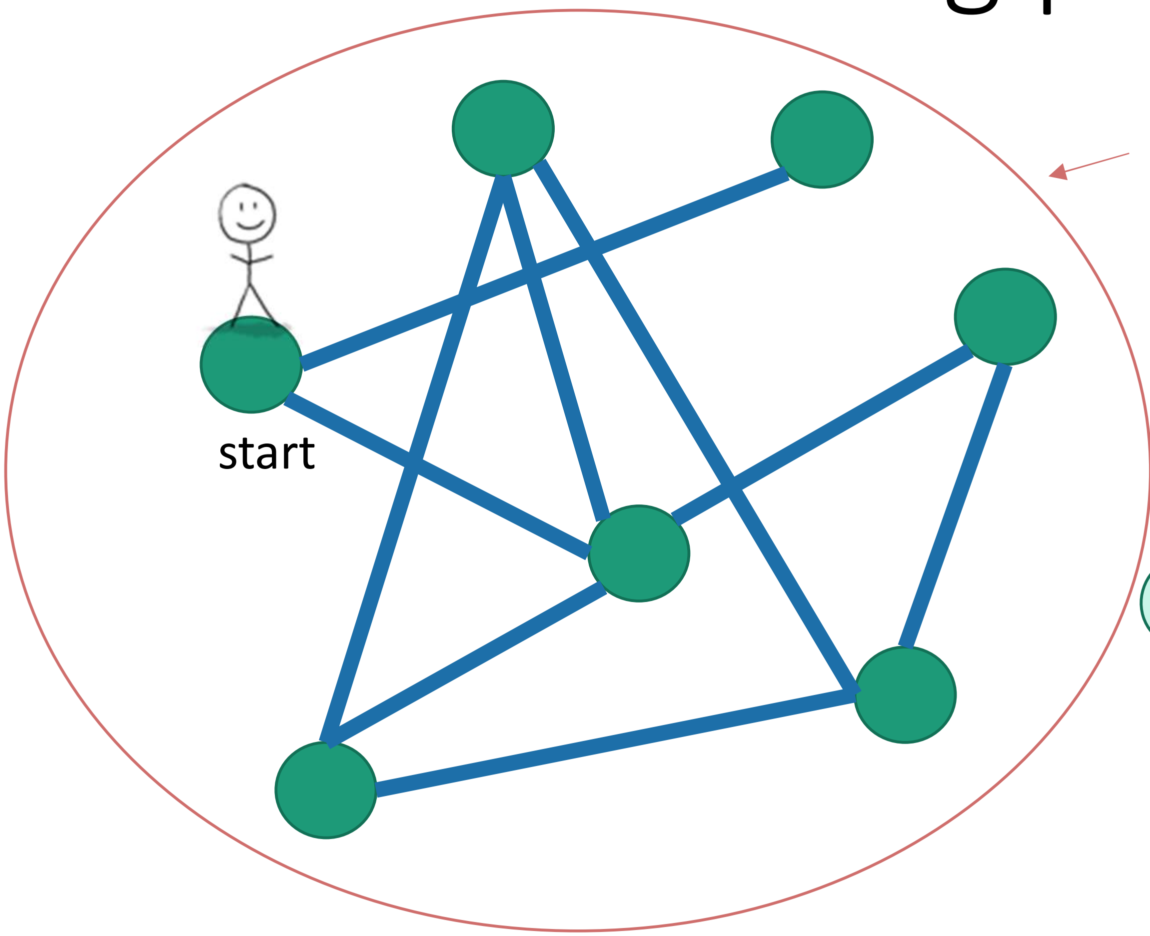
- Each vertex keeps track of whether it is:
  - Unvisited
  - In progress
  - All done



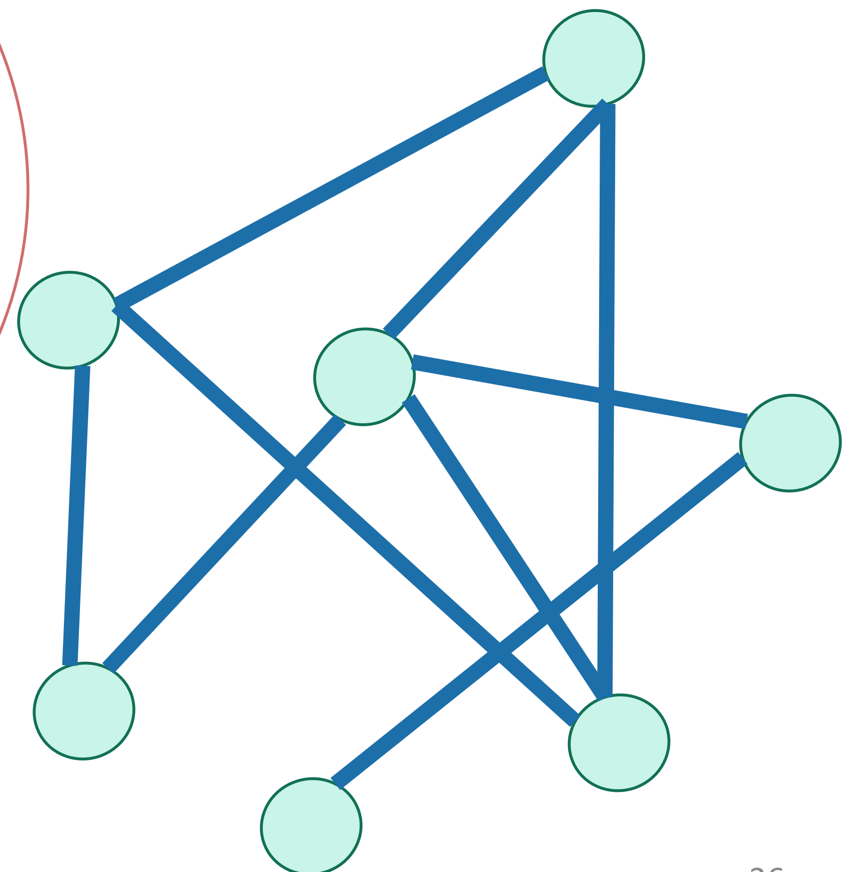


- **DFS**(w):
  - Mark w as **in progress**.
  - **for** v in w.neighbors:
    - **if** v is **unvisited**: **DFS**(v)
  - Mark w as **all done**

DFS finds all the nodes reachable from the starting point



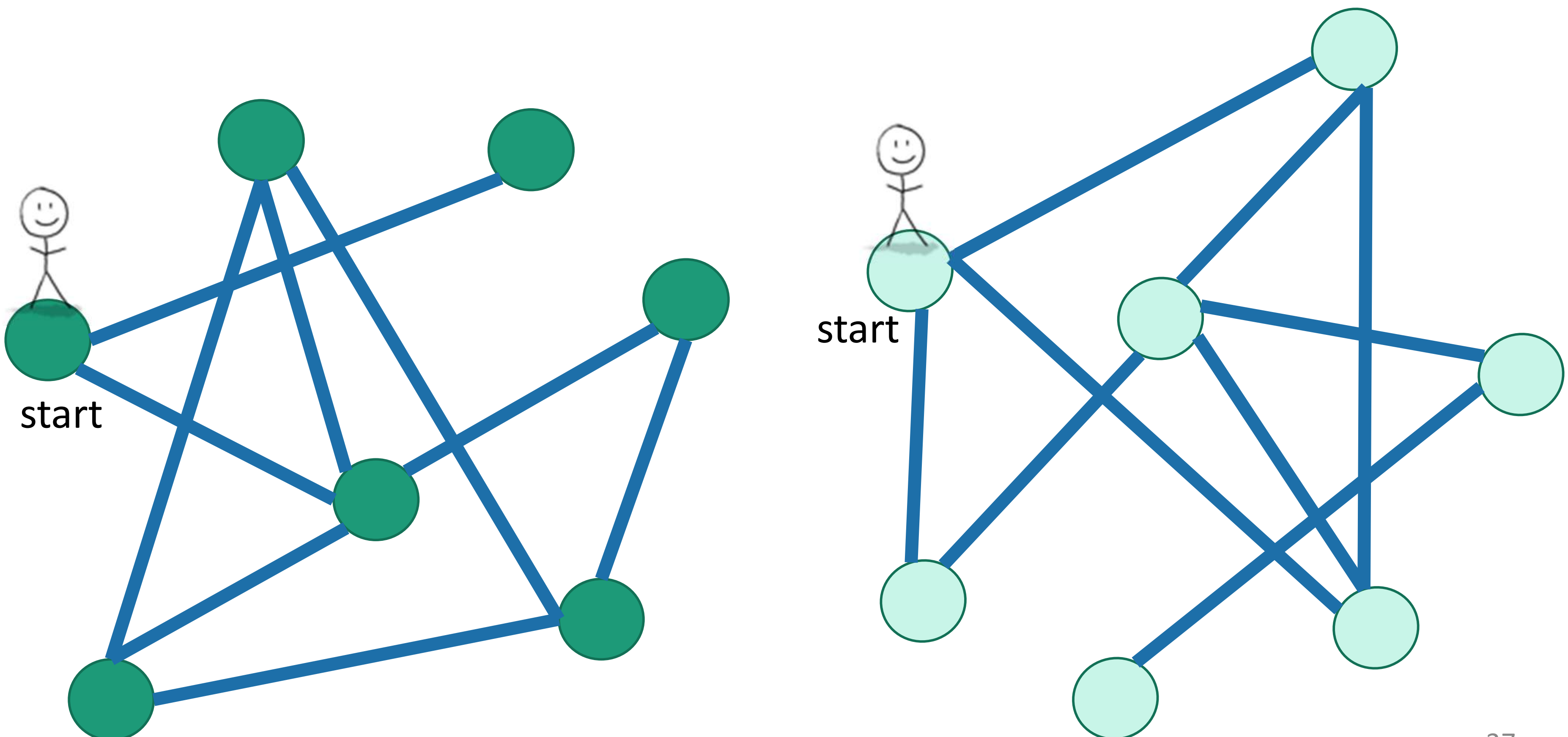
In an undirected graph, this is called a **connected component.**



One application of DFS: finding connected components.

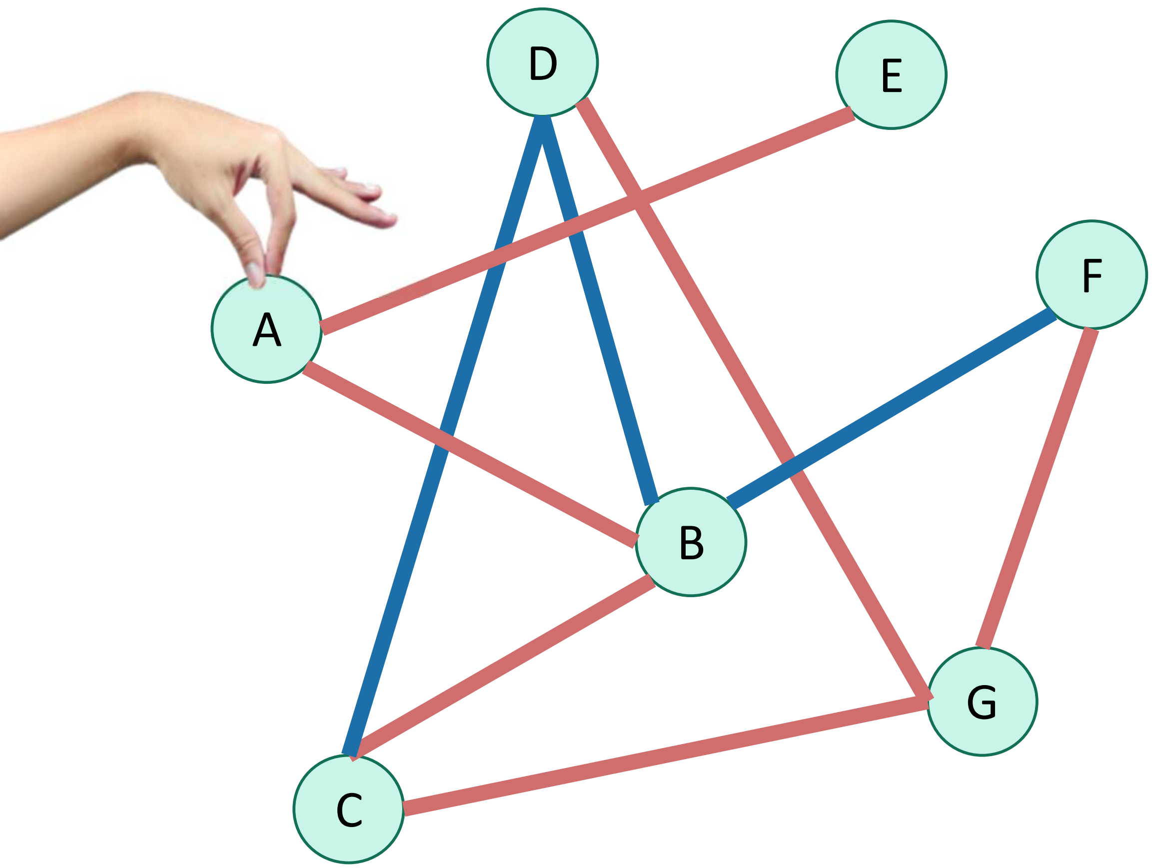
To explore the whole graph

- Do it repeatedly!

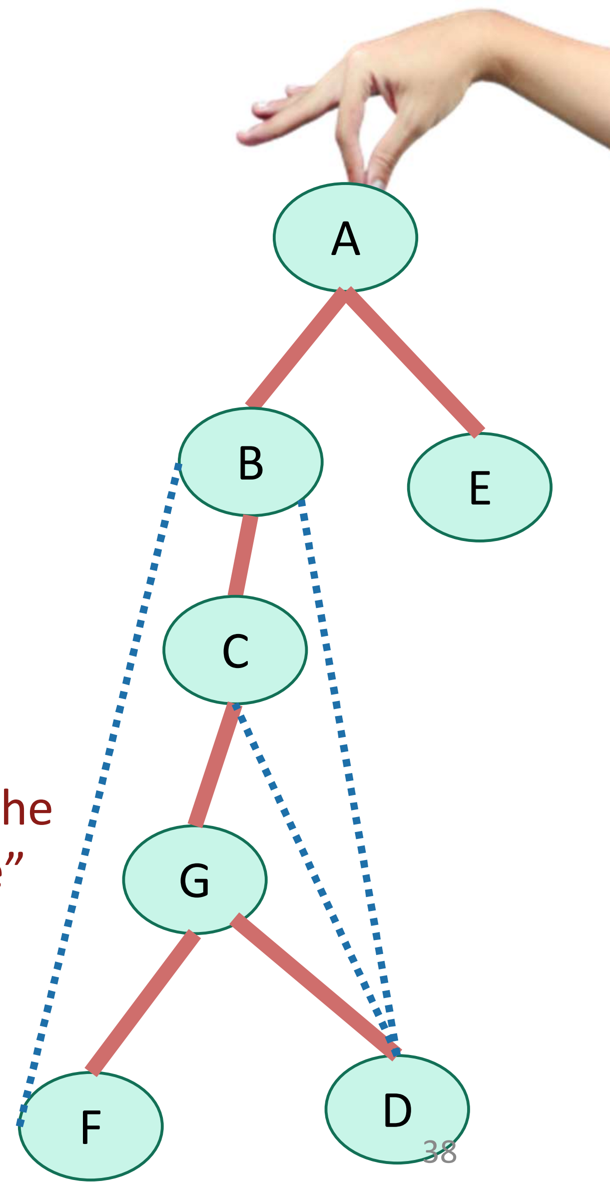


Why is it called depth-first?

- We are implicitly building a tree:



Call this the "DFS tree"



- First, we go as deep as we can.

Running time

To explore just the connected component we started in

- We look at each edge at most twice.
  - Once from each of its endpoints
- And basically, we don't do anything else.
- So…

$$O(m)$$

Running time

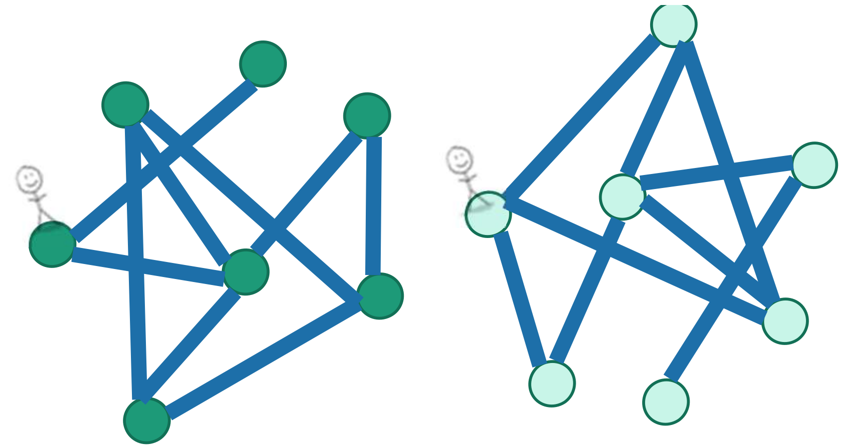
To explore just the connected component we started in

- Assume we are using the linked-list format for G.
- Say C = (V', E') is a connected component.
- We visit each vertex in V' exactly once.
  - Here, "visit" means "call DFS on"
- At each vertex w, we:
  - Do some book-keeping: O(1)
  - Loop over w's neighbors and check if they are visited (and then potentially make a recursive call): O(1) per neighbor or O(deg(w)) total.
- Total time:
  - $\sum_{w \in V'} (O(\deg(w)) + O(1))$
  - $= O(|E'| + |V'|)$
  - $= O(|E'|)$

In a connected graph, $|V'| \leq |E'| + 1$.

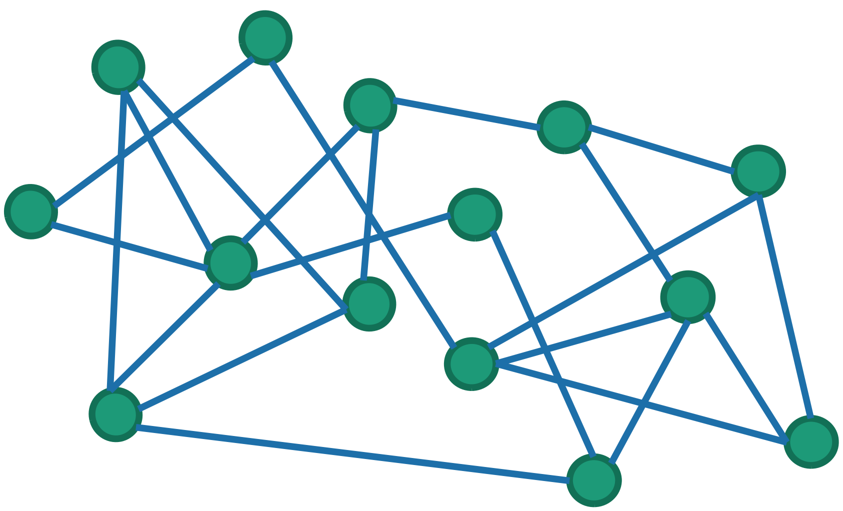
Running time

To explore the whole graph



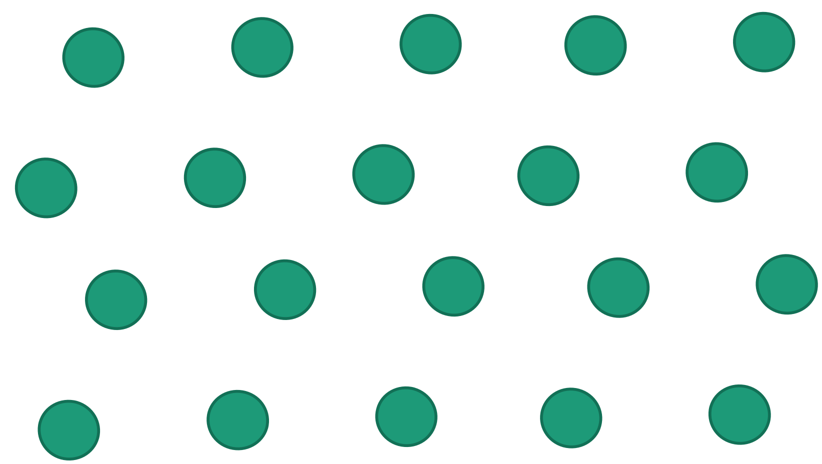
- Explore the connected components one-by-one.
- This takes time O(n + m)
  - Same computation as before:

$$\sum_{w \in V} (O(\deg(w)) + O(1)) = O(|E| + |V|) = O(n + m)$$



Here the running time is O(m) like before

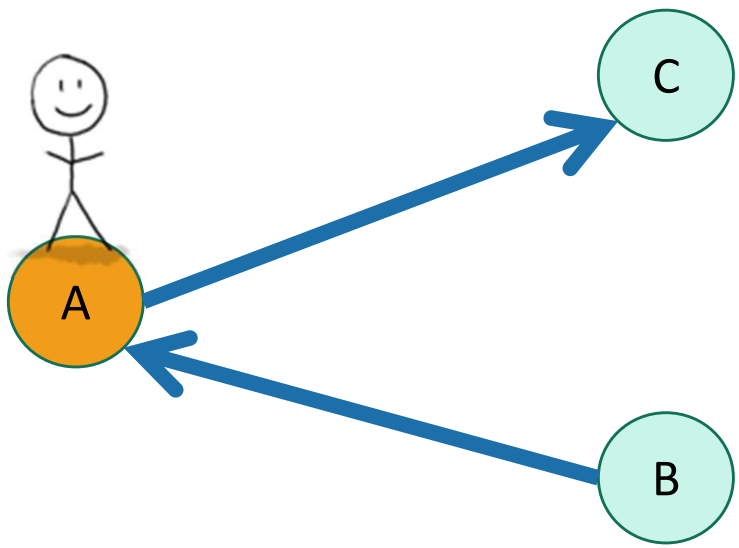
or



Here m=0 but it still takes time O(n) to explore the graph.

You check:

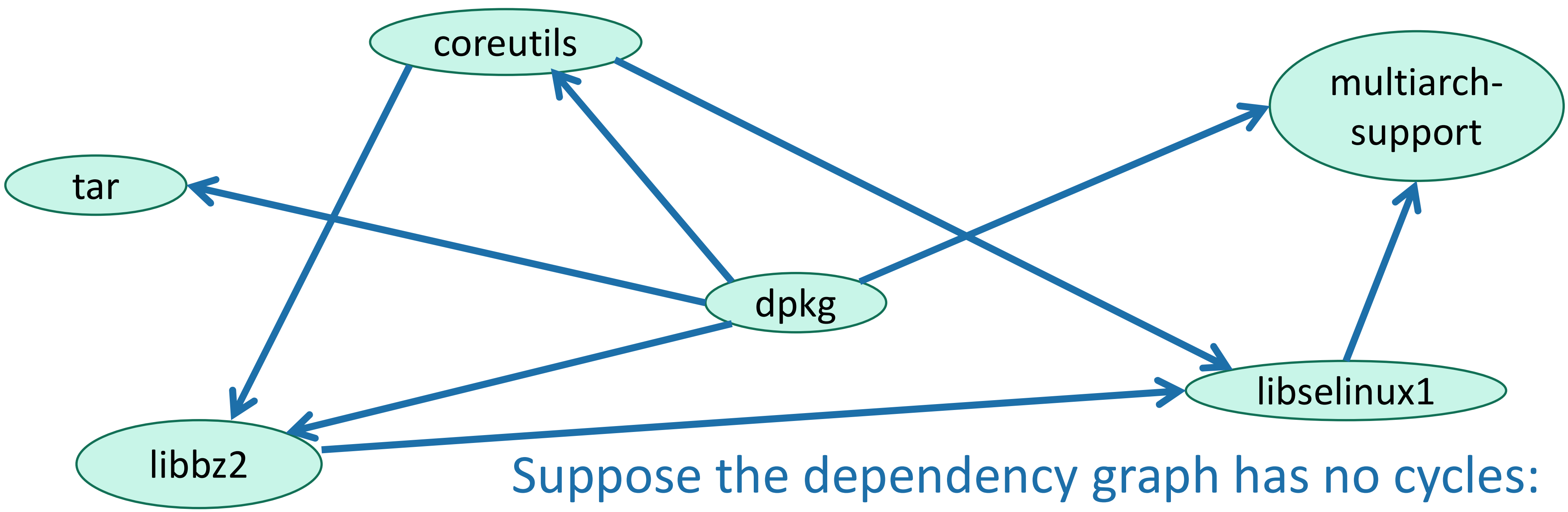
DFS works fine on directed graphs too!



Only walk to C, not to B.

Ví dụ DFS: sắp xếp topo
(topological sorting)

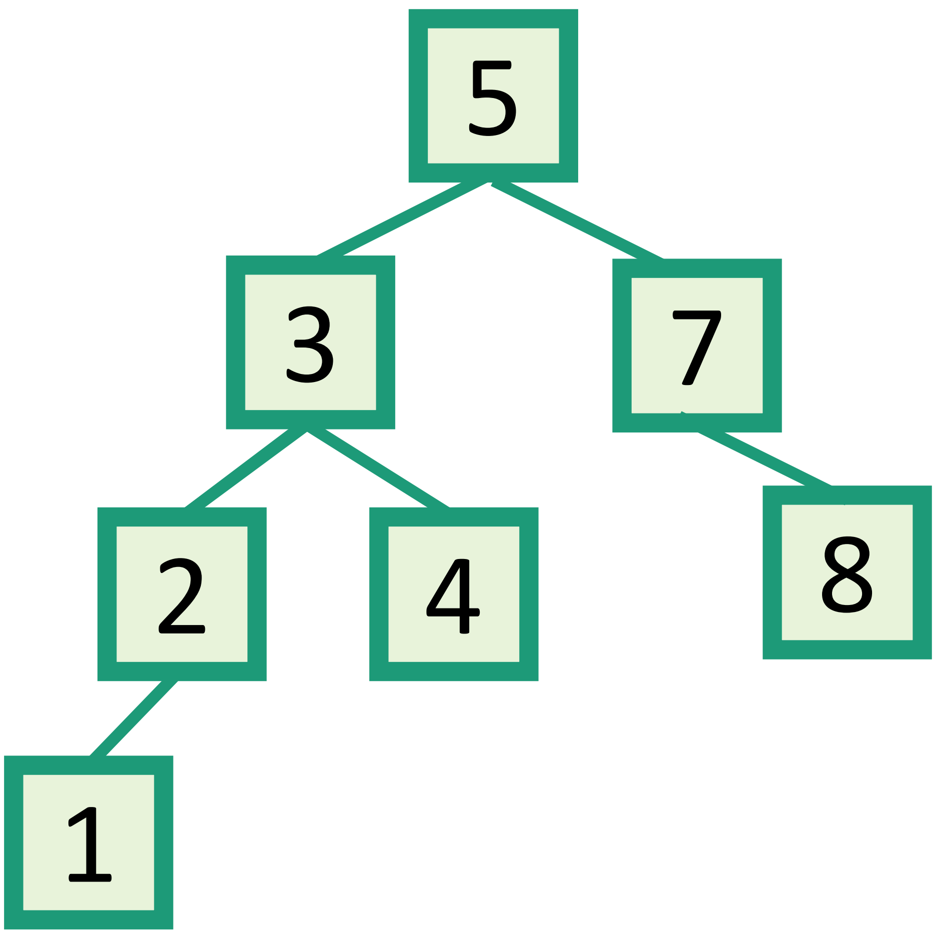
- Tìm thứ tự các đỉnh đảm bảo thỏa mãn quan hệ phụ thuộc.
  - Aka, if v comes before w in the ordering, there is not an edge from w to v.



Suppose the dependency graph has no cycles: it is a **Directed Acyclic Graph (DAG)**

Ví dụ DFS: duyệt cây nhị phân

- Duyệt cây nhị phân theo chiều sâu (in-order)

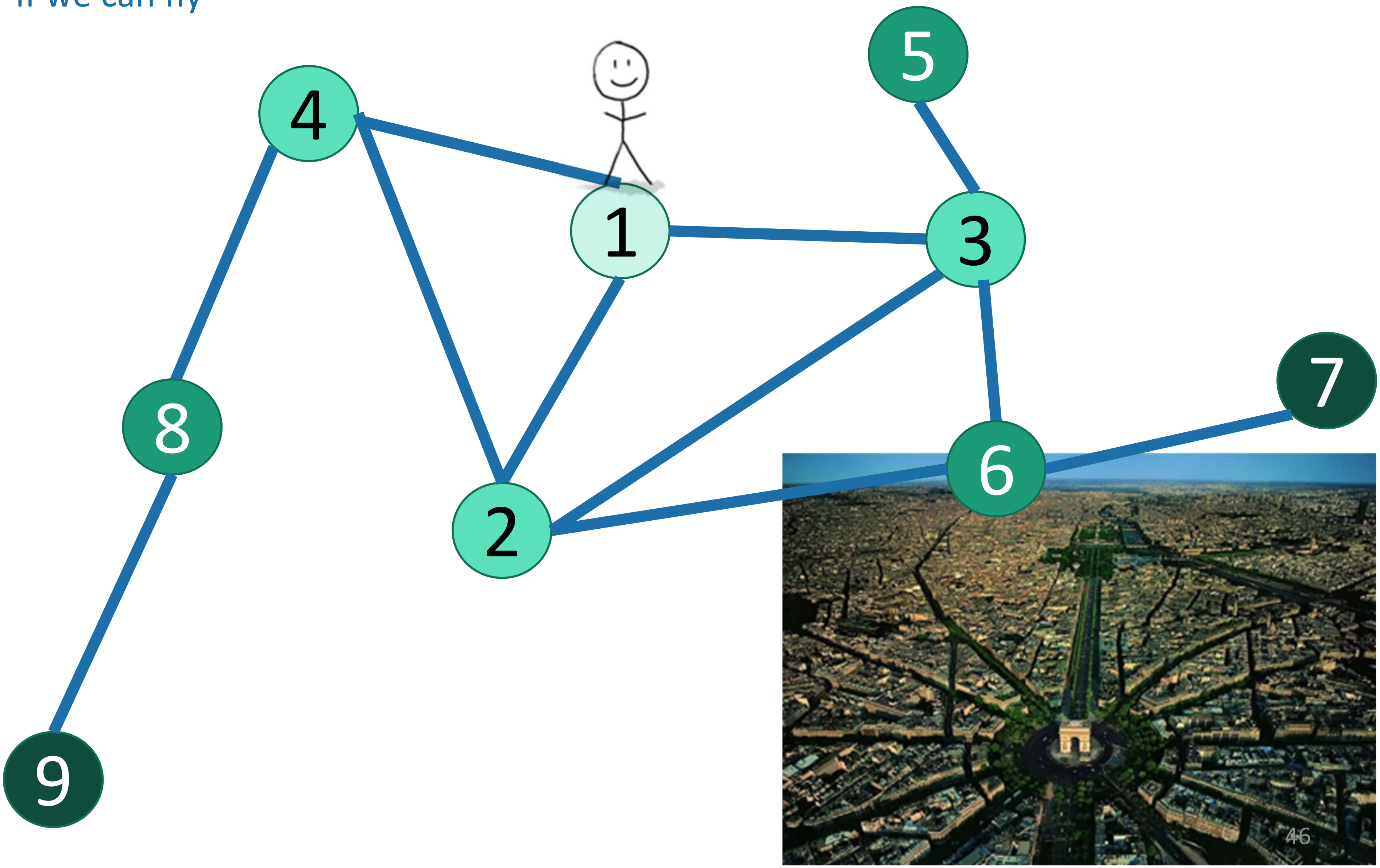


Do DFS and print a node's label when you are done with the left child and before you begin the right child.

Phần 3: Breadth-first search

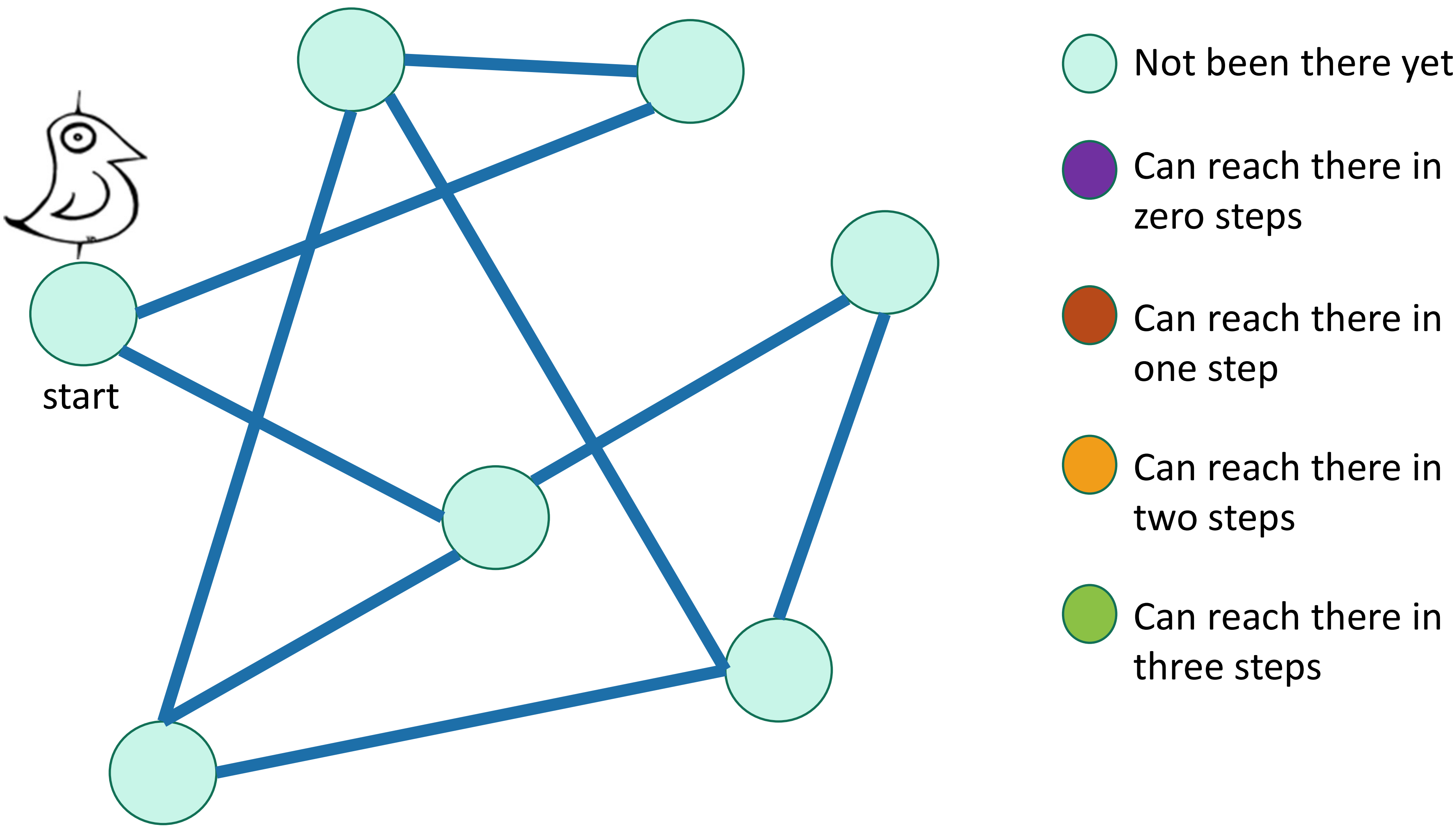
How do we explore a graph?

If we can fly



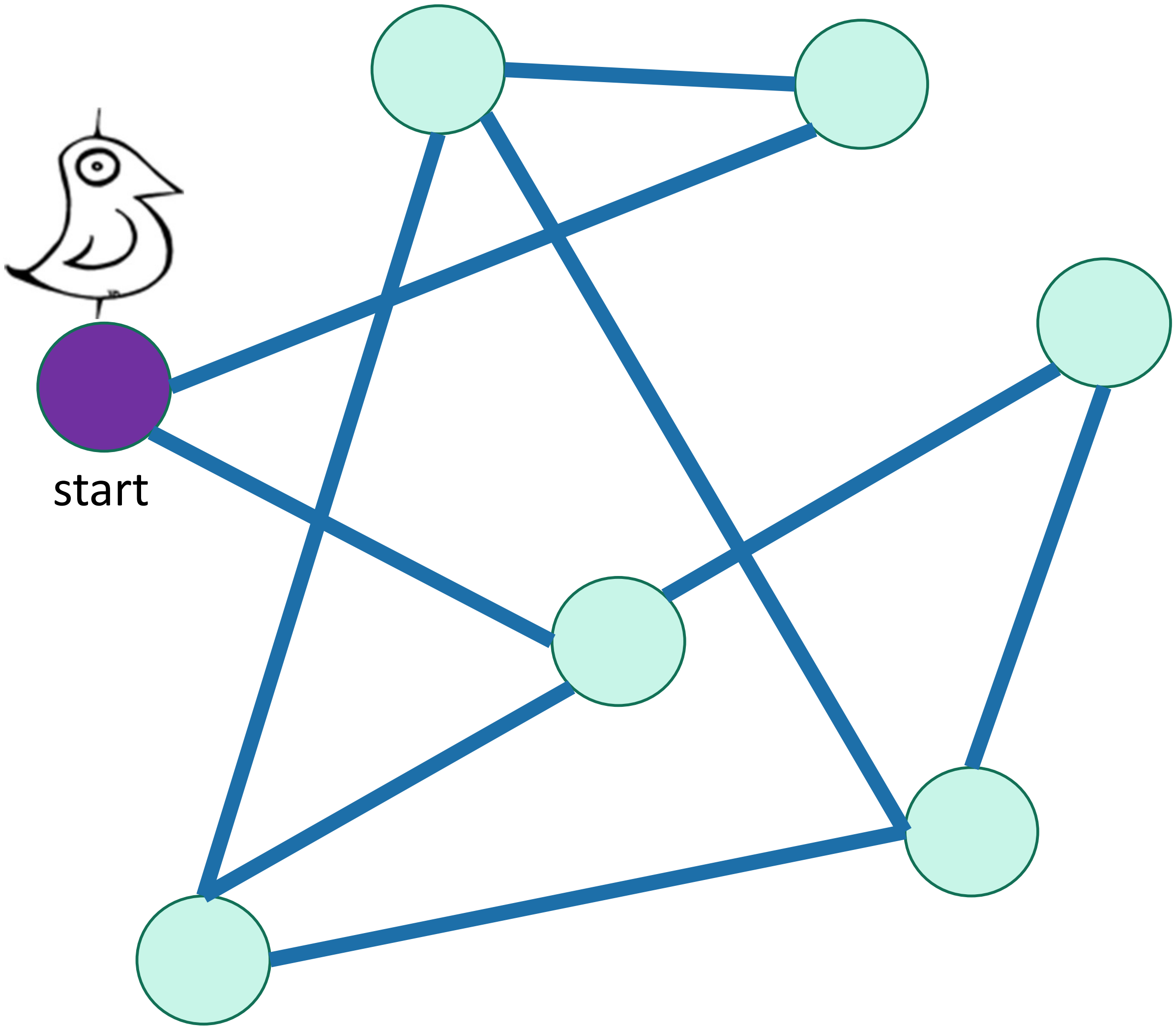
Breadth-First Search

Exploring the world with a bird's-eye view



Breadth-First Search

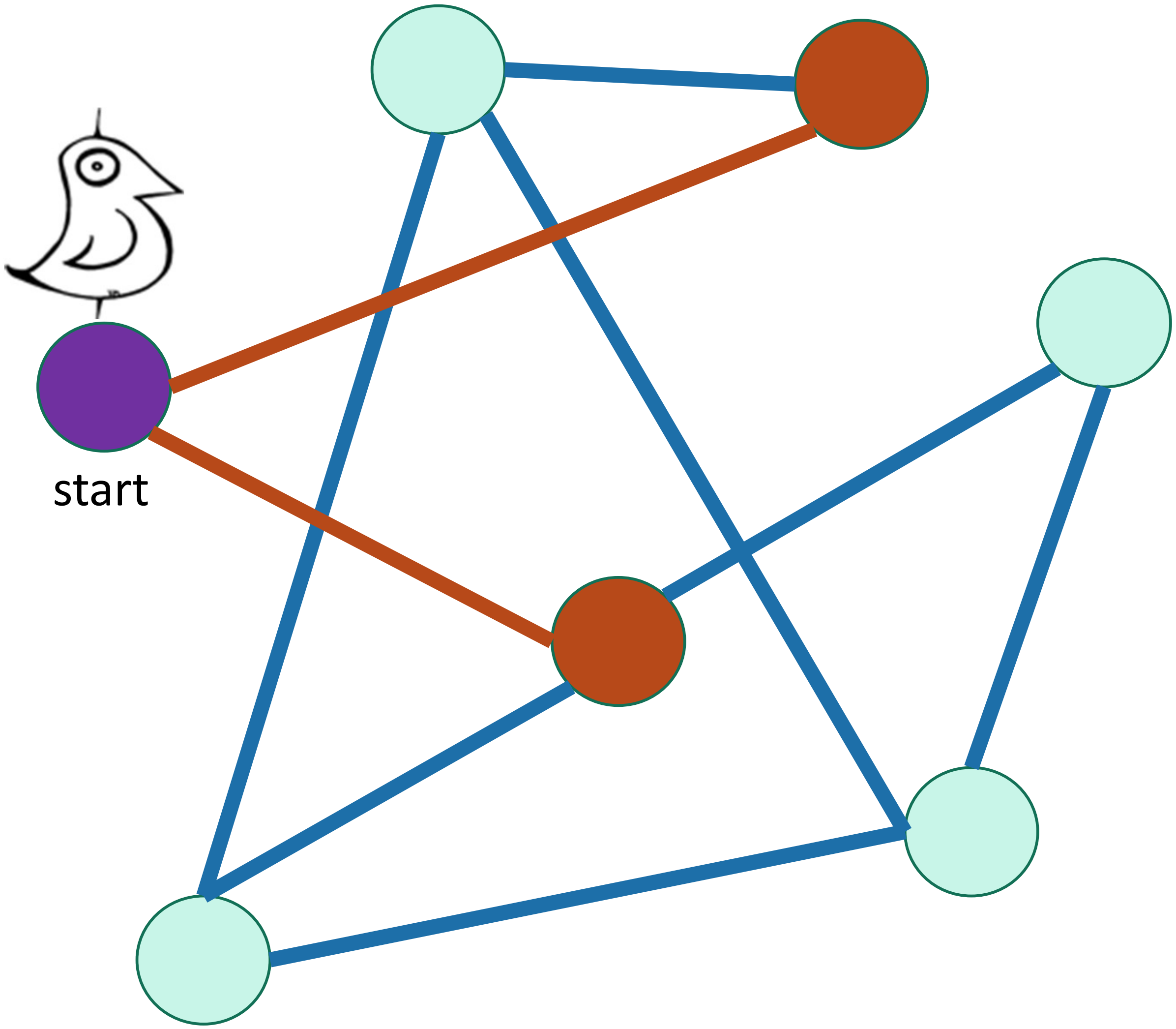
Exploring the world with a bird's-eye view



- 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

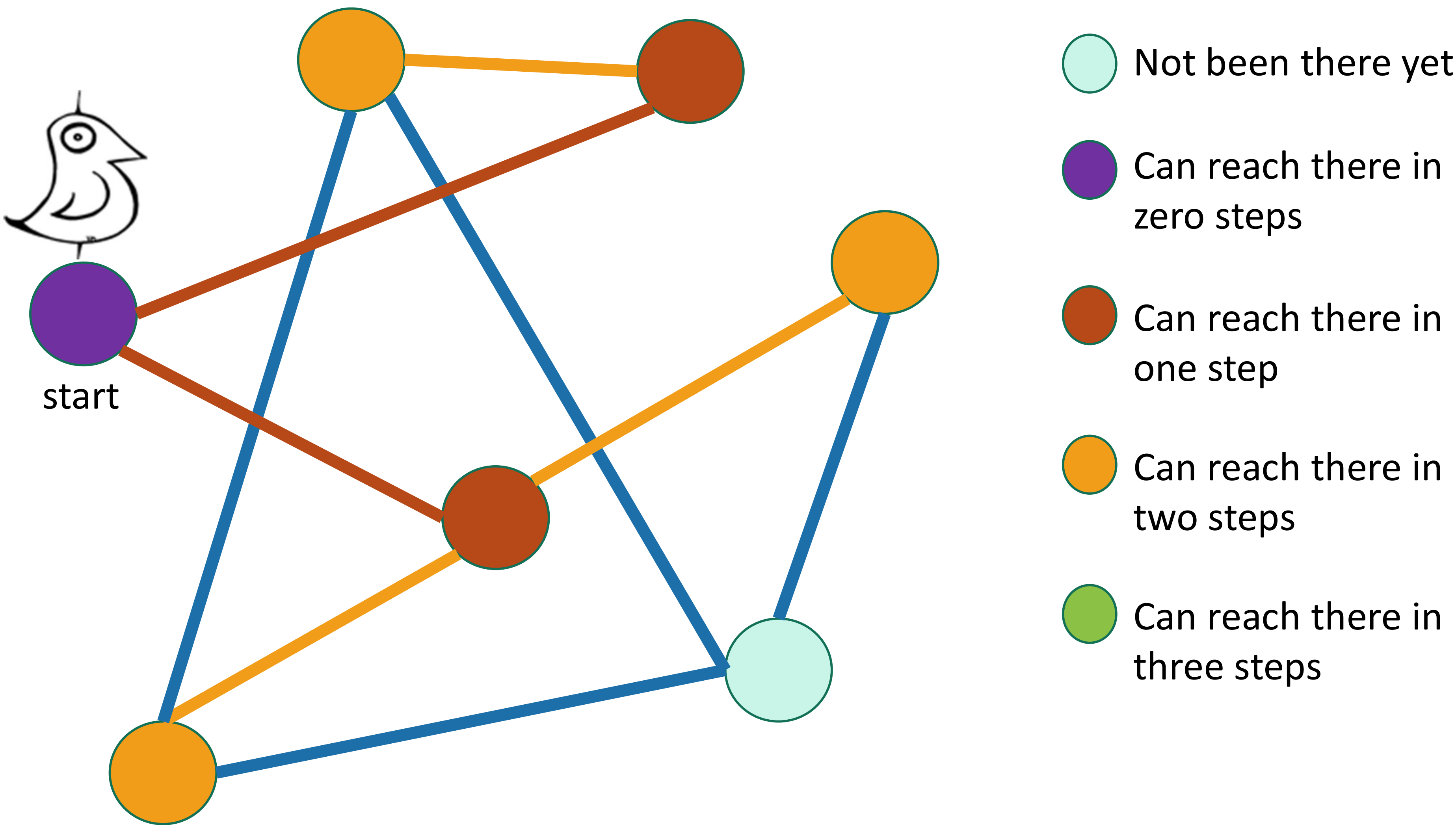
Exploring the world with a bird's-eye view



- 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

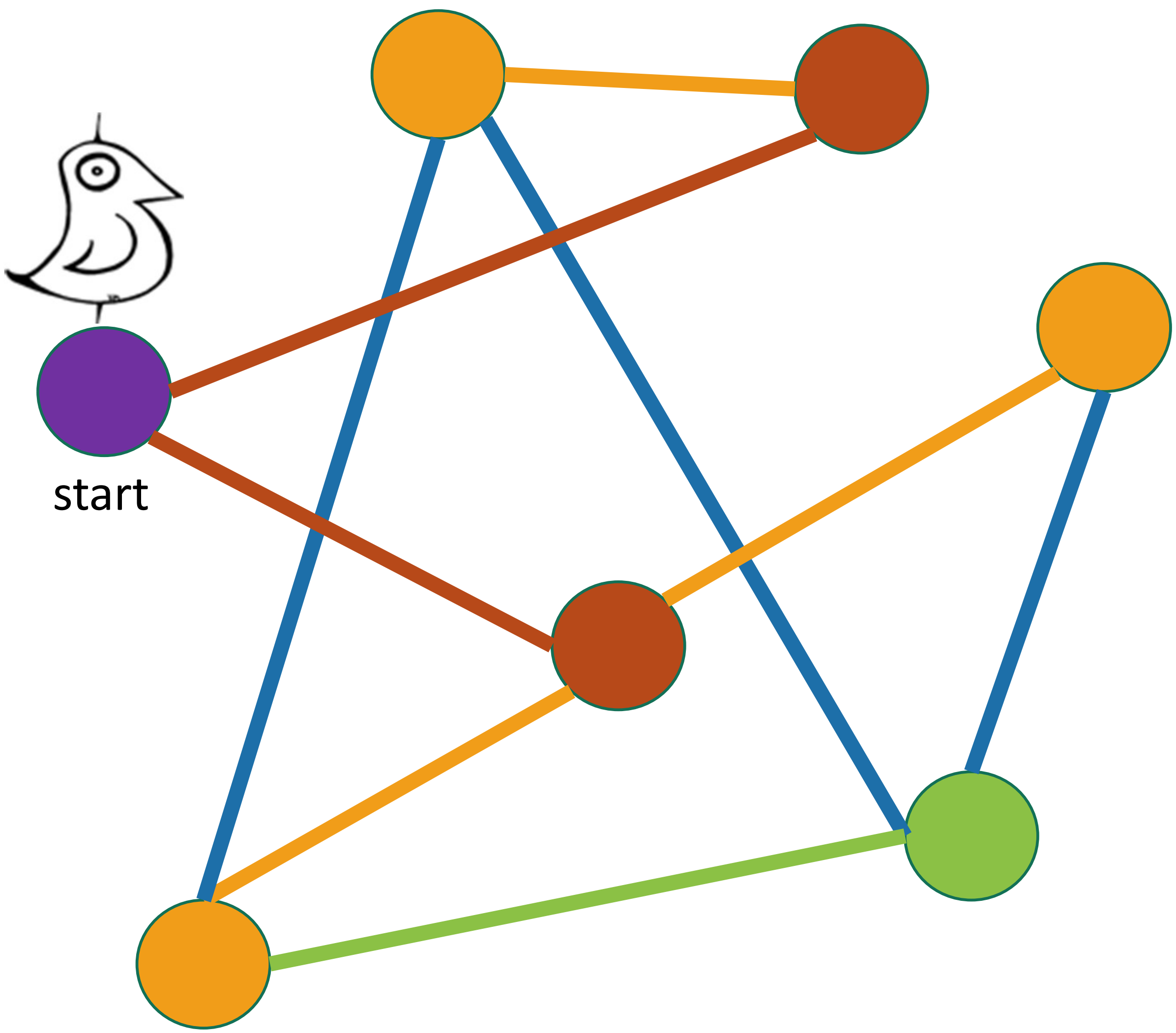
Breadth-First Search

Exploring the world with a bird’s-eye view



Breadth-First Search

Exploring the world with a bird's-eye view



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

World: explored!

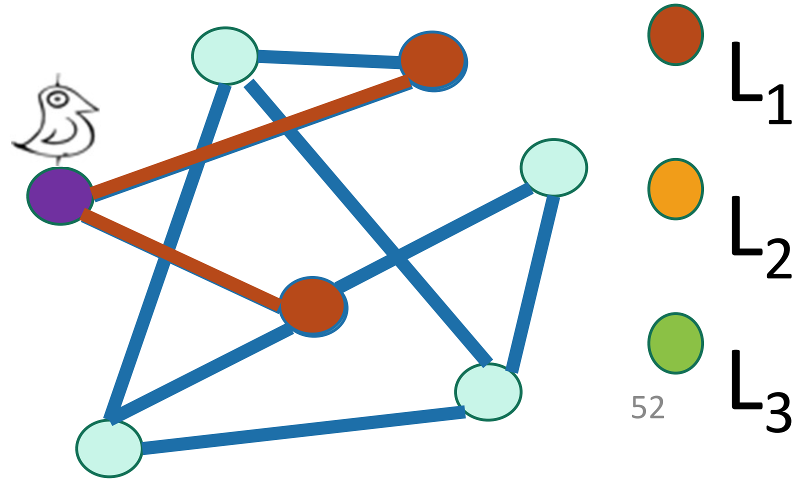
Breadth-First Search

Exploring the world with pseudocode

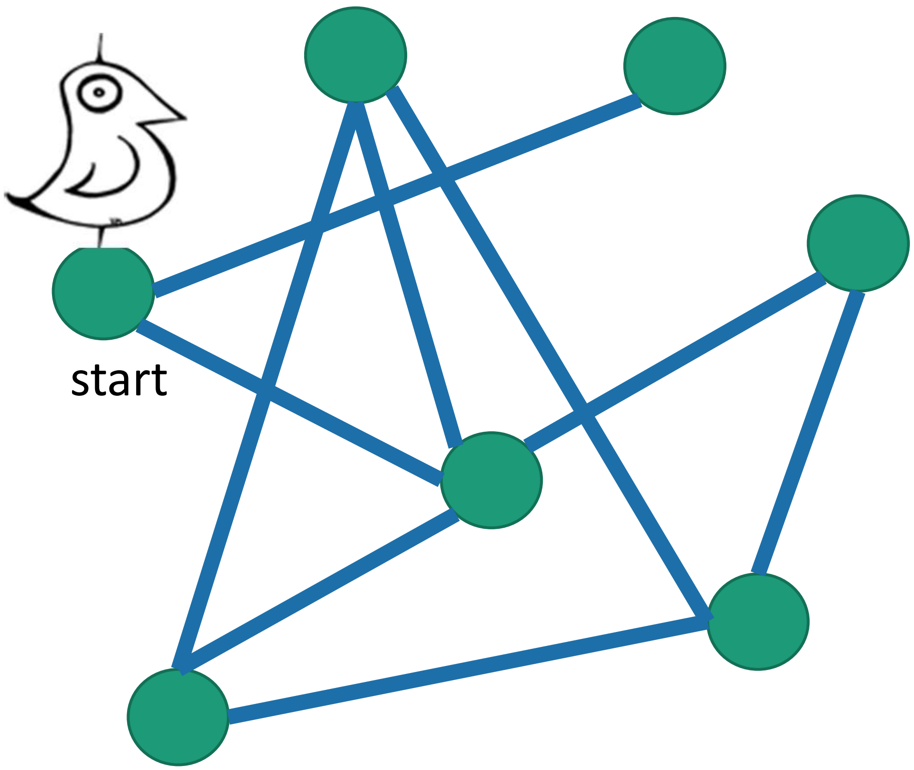
L_i is the set of nodes we can reach in i steps from w

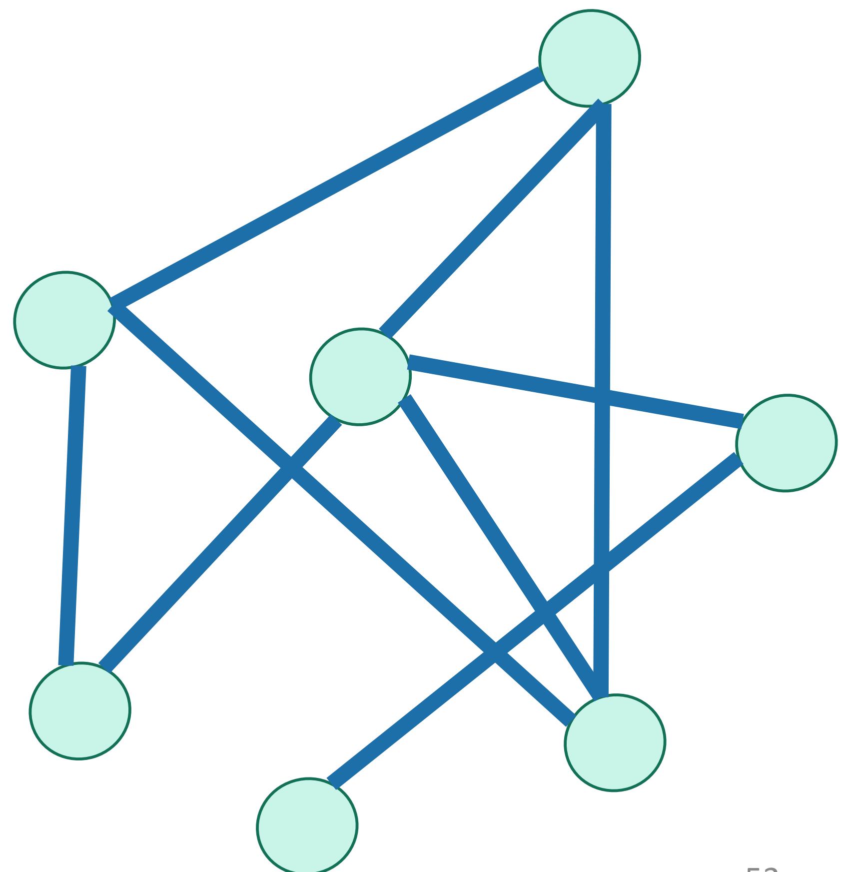
- Set L_i = [] for i=1,…,n
- L_0 = [w], where w is the start node
- Mark w as visited
- **For** i = 0, …, n-1:
  - **For** u in L_i:
    - **For** each v which is a neighbor of u:
      - **If** v isn't yet visited:
        - mark v as visited, and put it in L_{i+1}

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.**

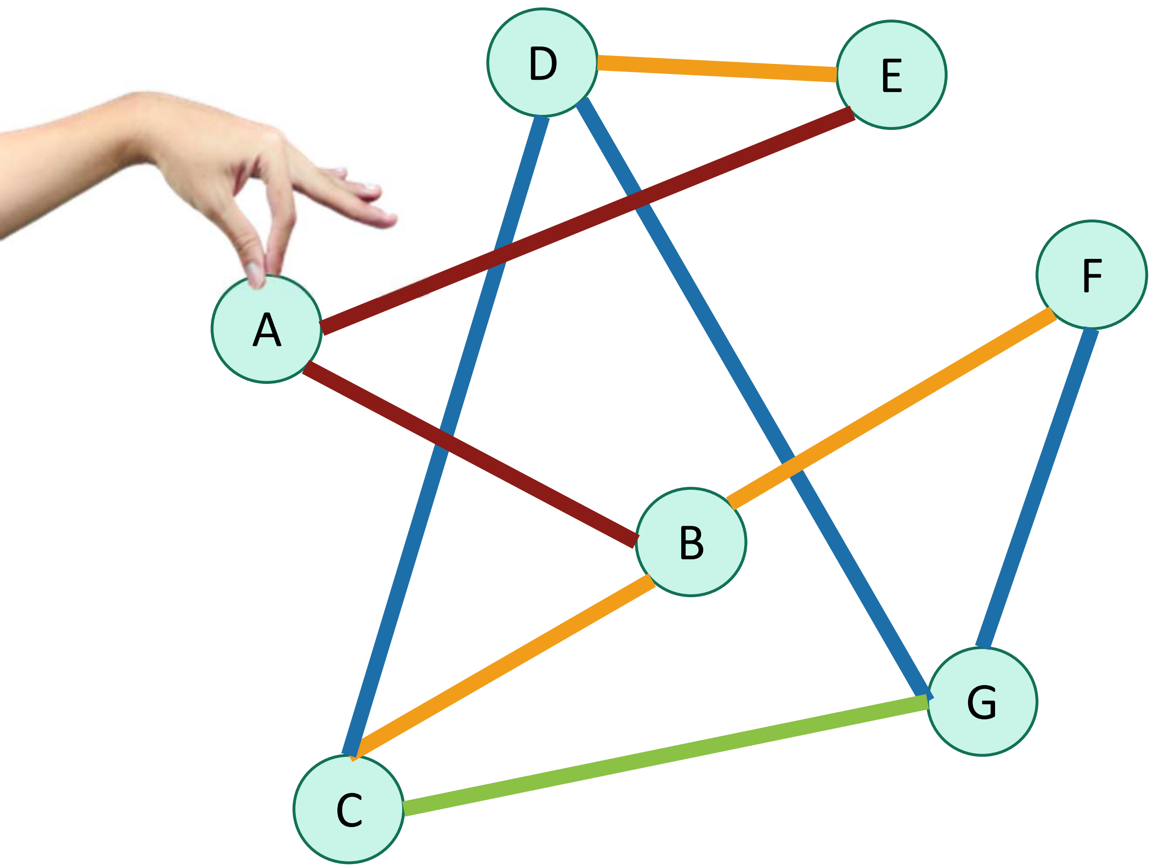
Running time and extension to directed graphs

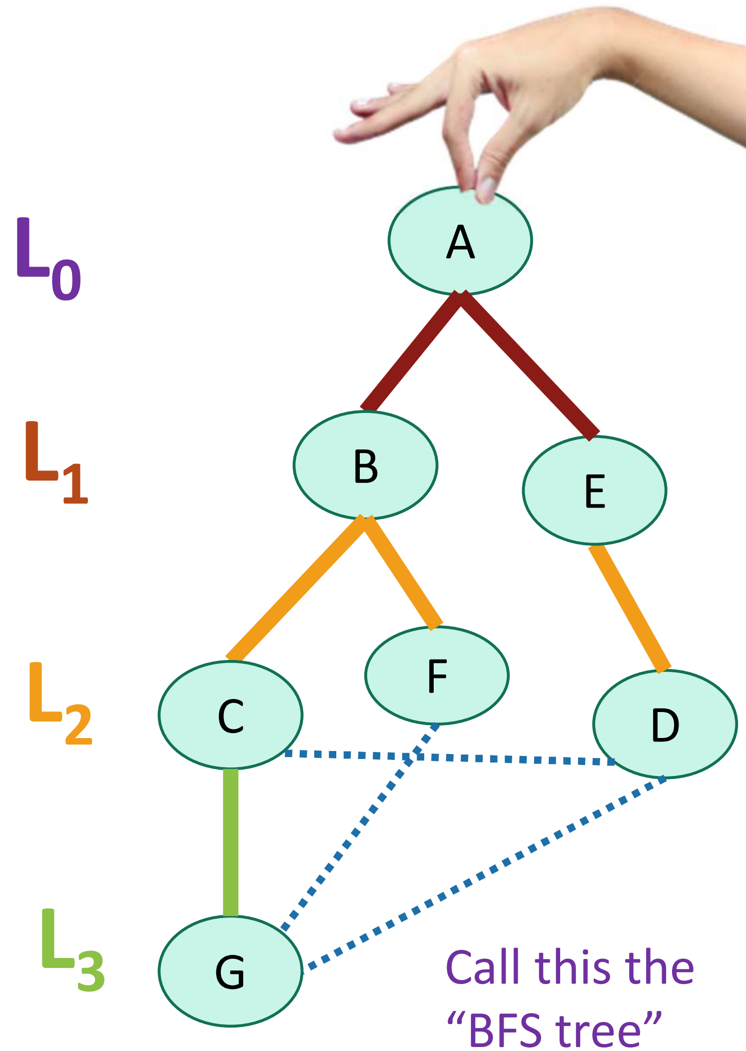
- To explore the whole graph, explore the connected components one-by-one.
  - Same argument as DFS: BFS running time is O(n + m)
- Like DFS, BFS also works fine on directed graphs.

Verify these!

Why is it called breadth-first?

- We are implicitly building a tree:

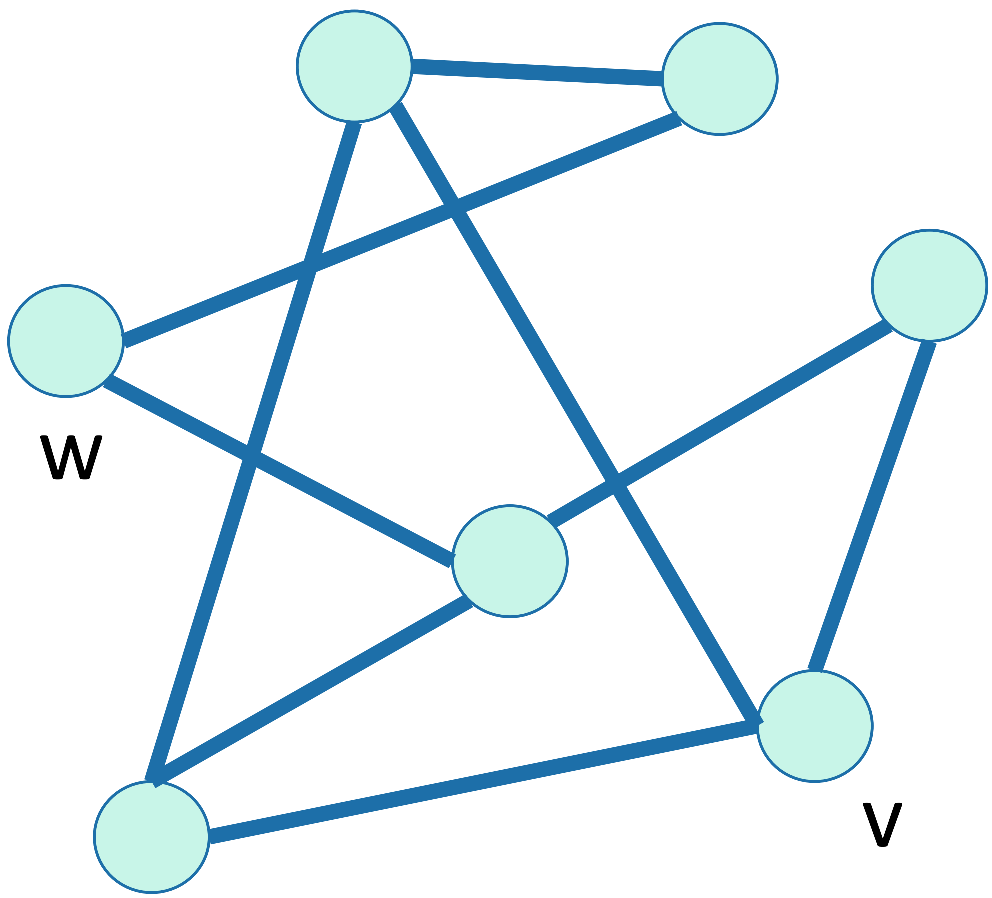




- First we go as broadly as we can.

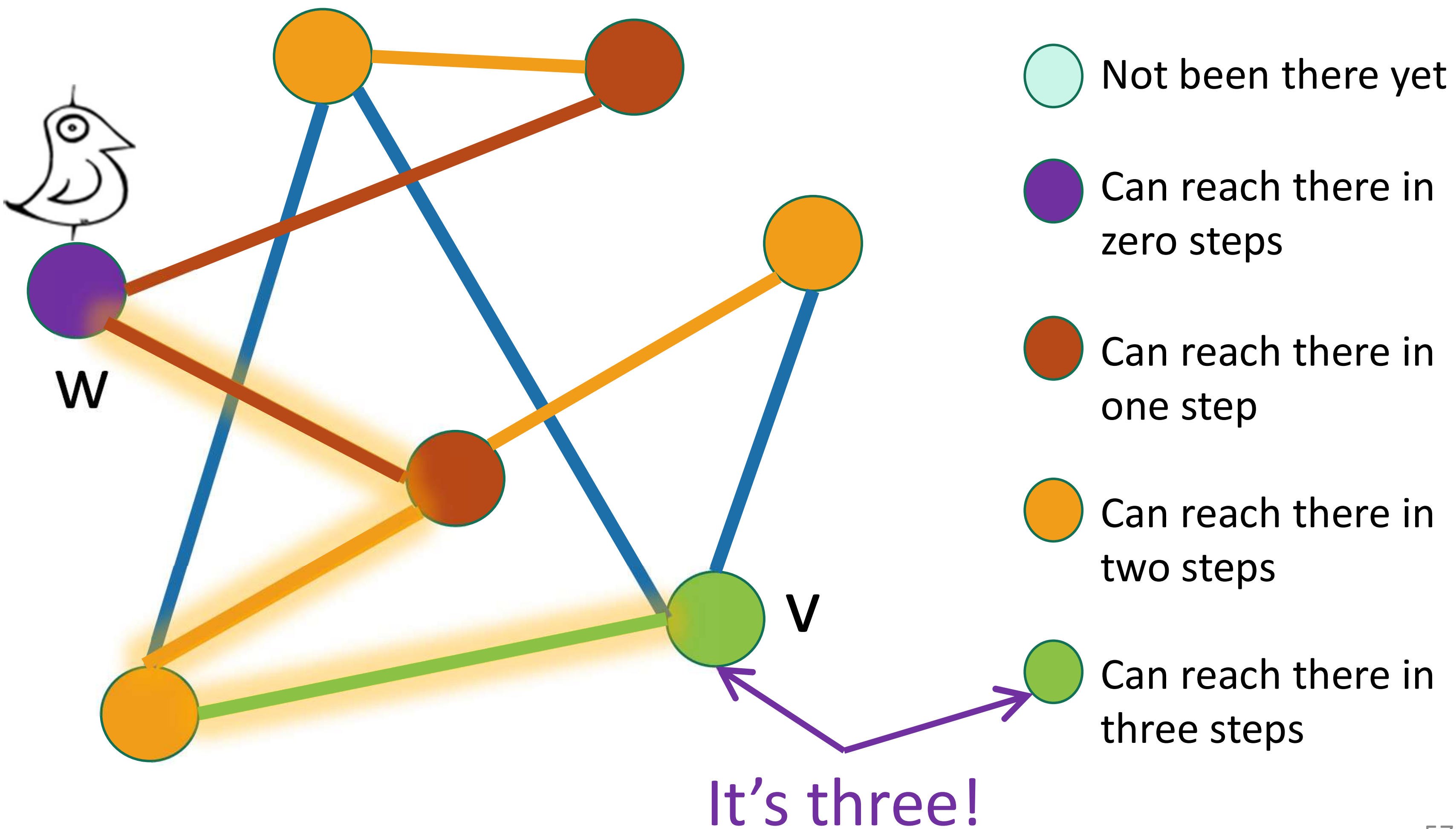
Application of BFS: shortest path

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



Application of BFS: shortest path

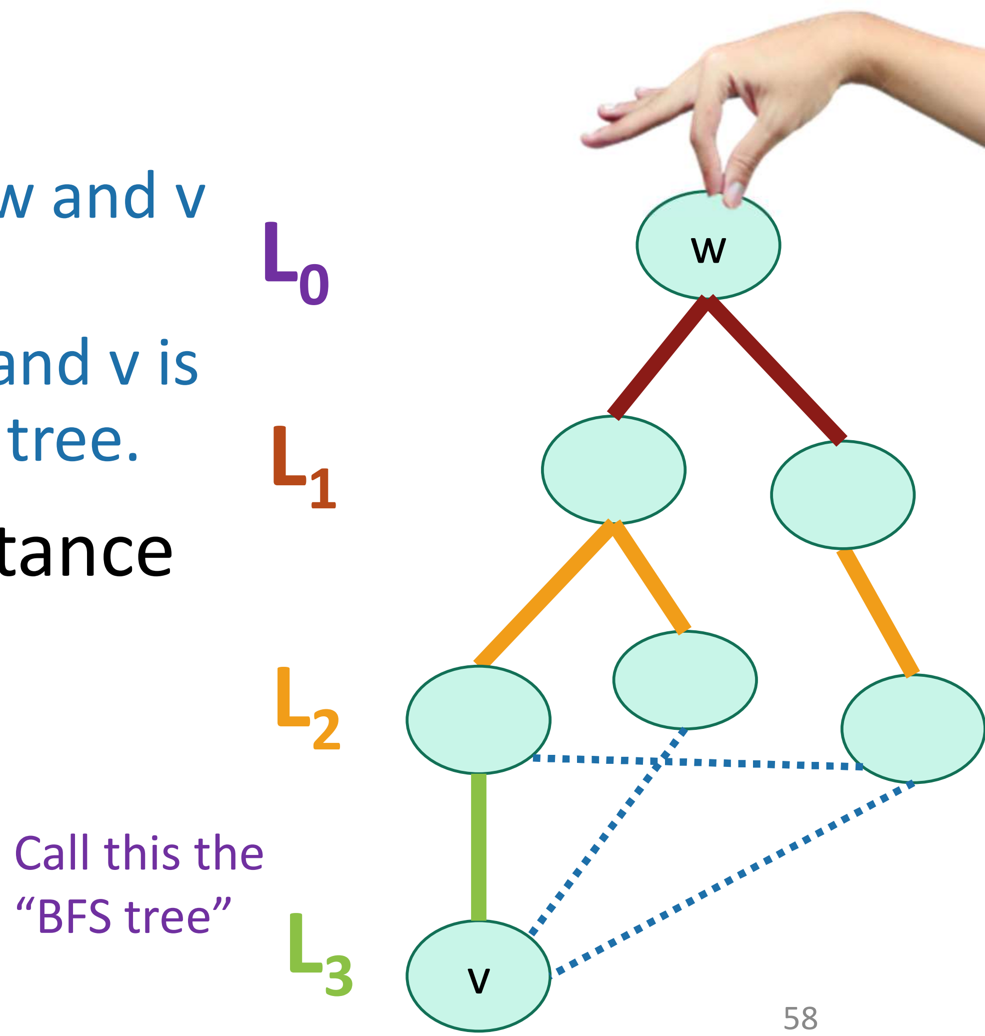
- How long is the shortest path between w and v?



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.



What have we learned?

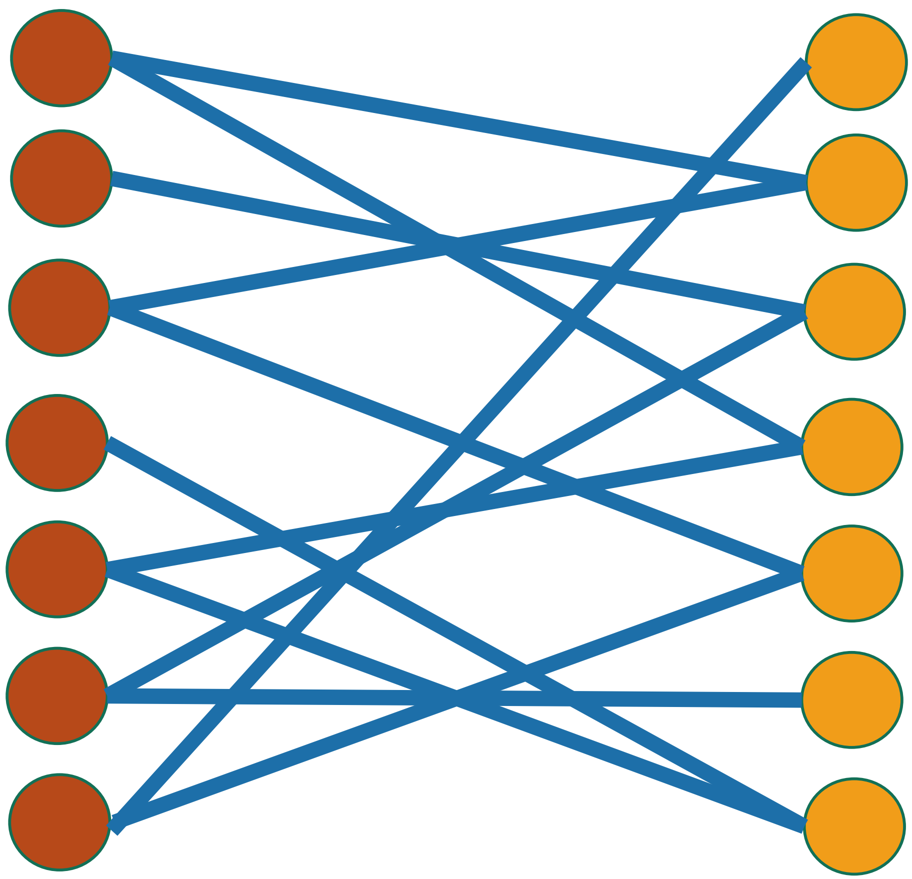
- The BFS tree is useful for computing distances between pairs of vertices.
- We can find the shortest path between u and v in time O(m).

Another application of BFS

- Kiểm tra tính chất lưỡng phân của đồ thị (Testing bipartite-ness)

Đồ thị lưỡng phân
Bipartite graphs

- A bipartite graph looks like this:

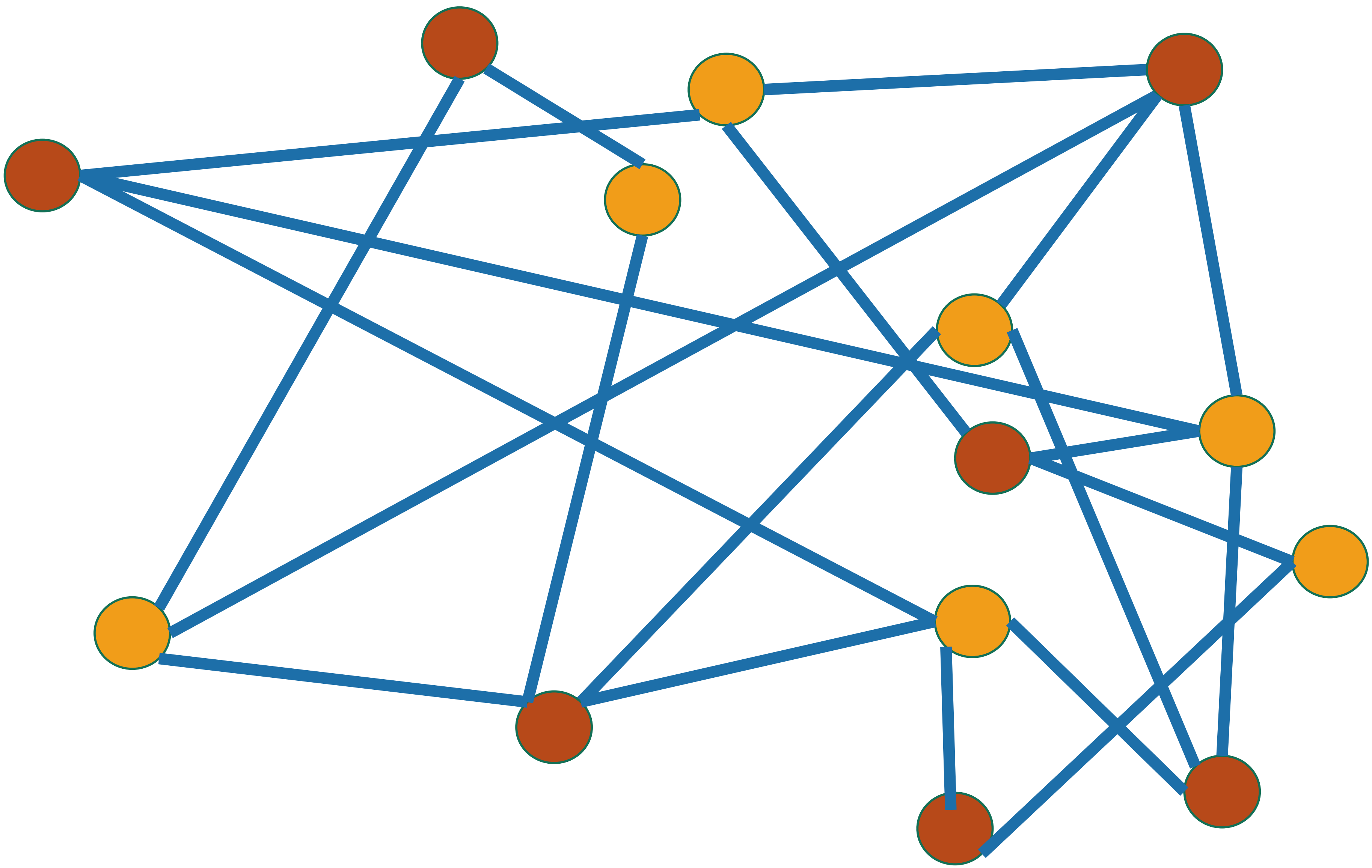


Can color the vertices red and orange so that there are no edges between any same-colored vertices

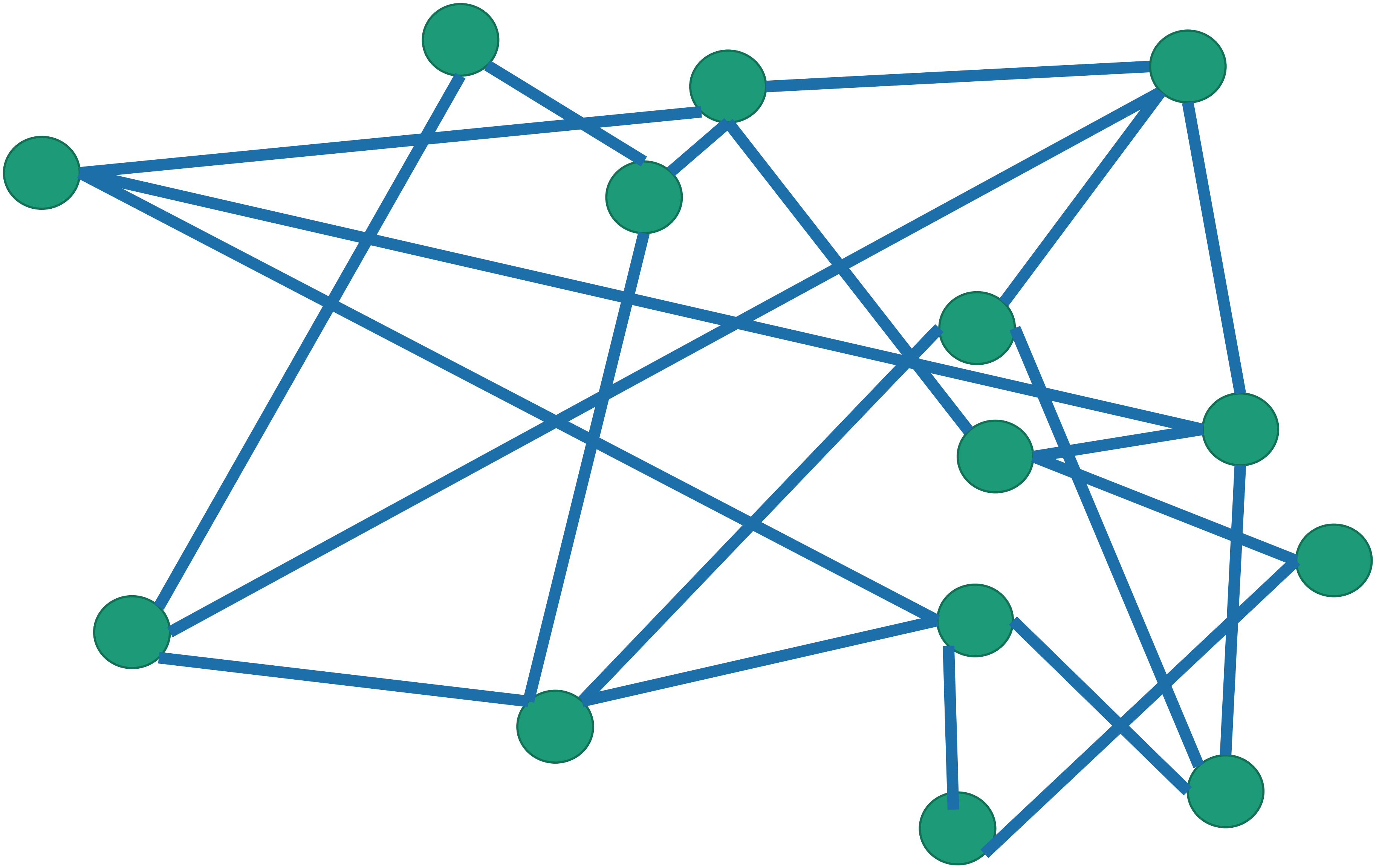
Example:
are in tank A
are in tank B
if the fish fight

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

How about this one?



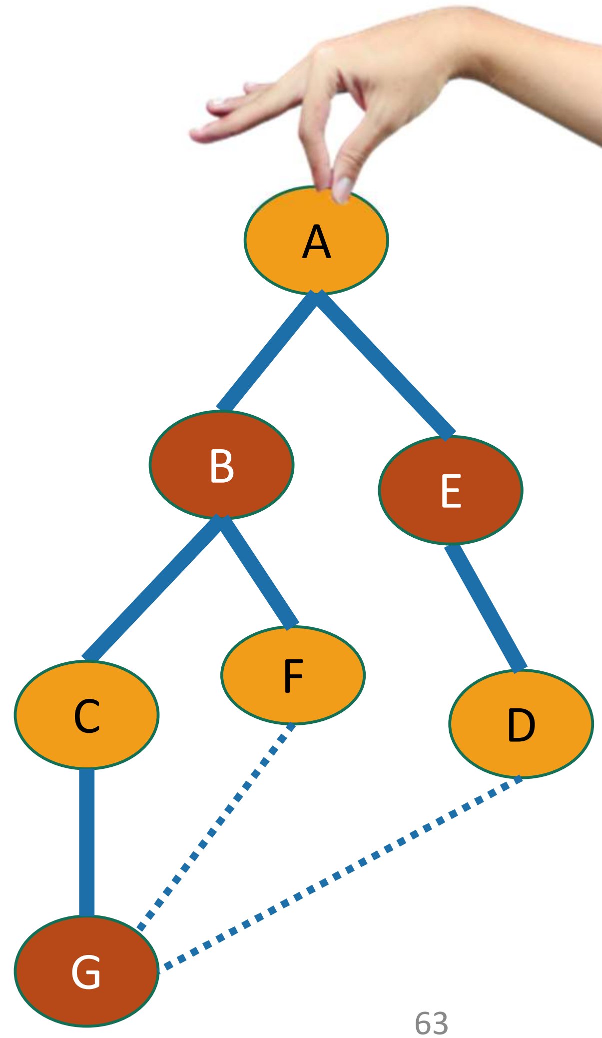
How about this one?



Does DFS work here too?

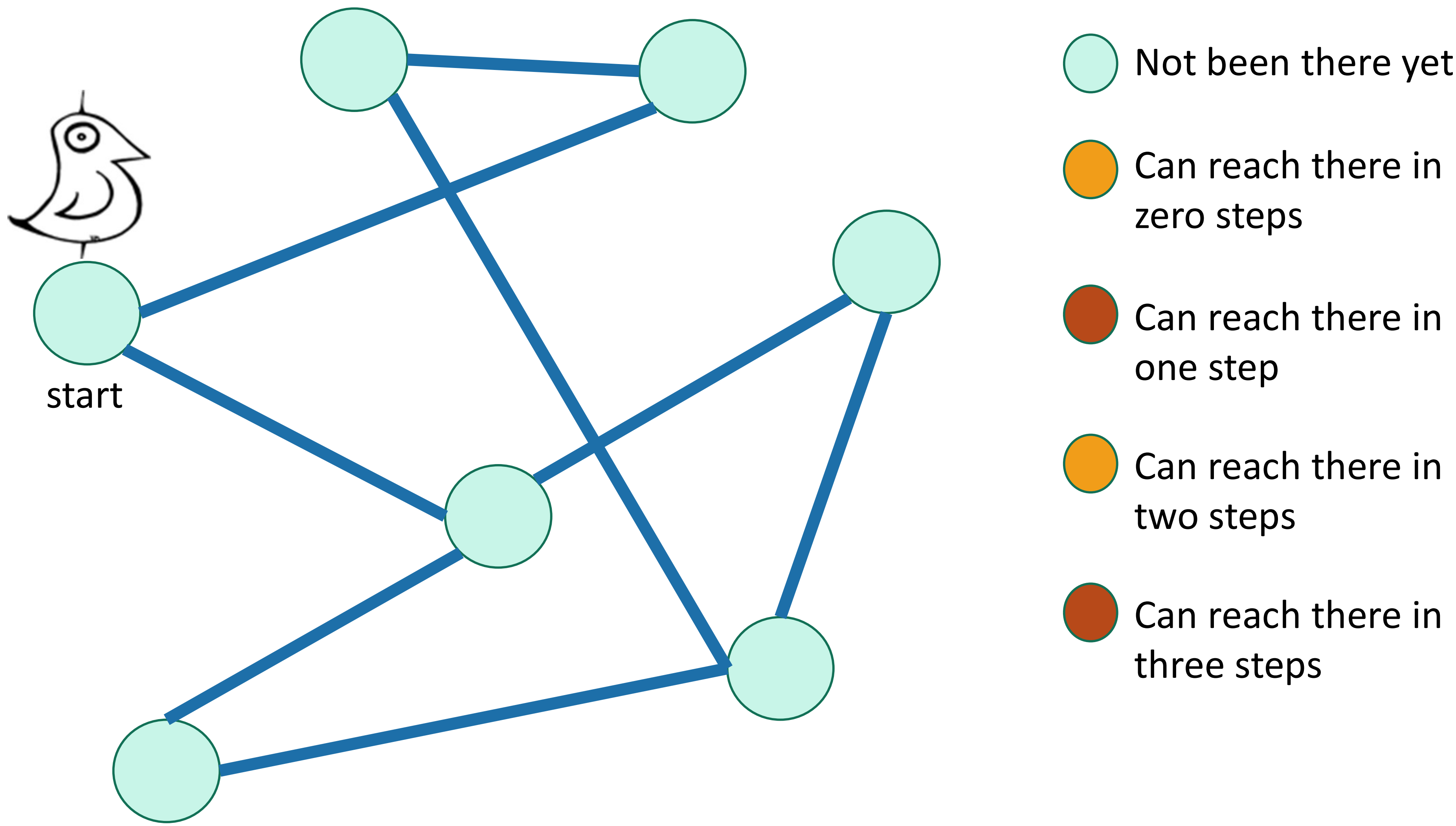
Application of BFS:
Testing Bipartiteness

- 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.



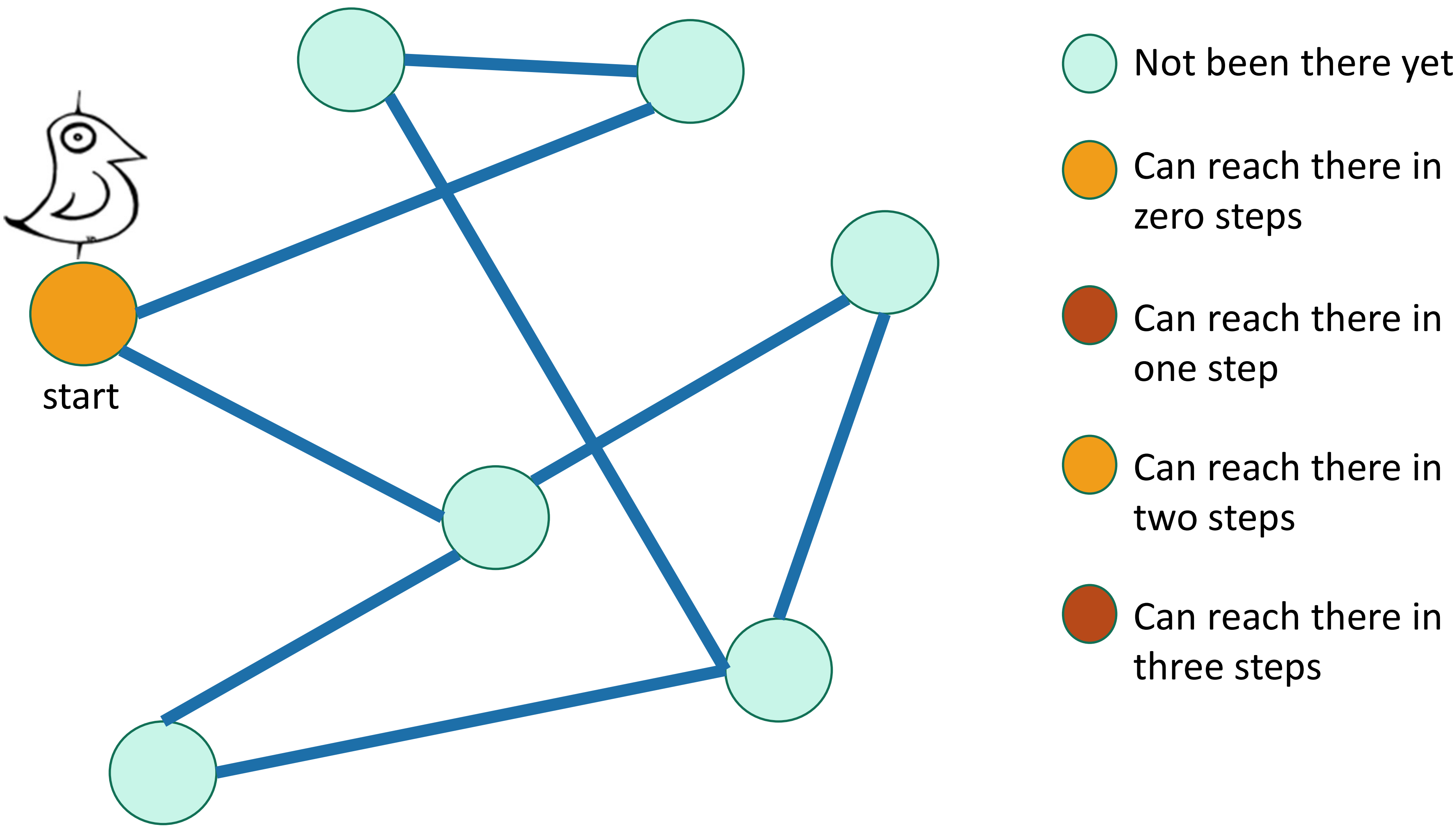
Breadth-First Search

For testing bipartite-ness



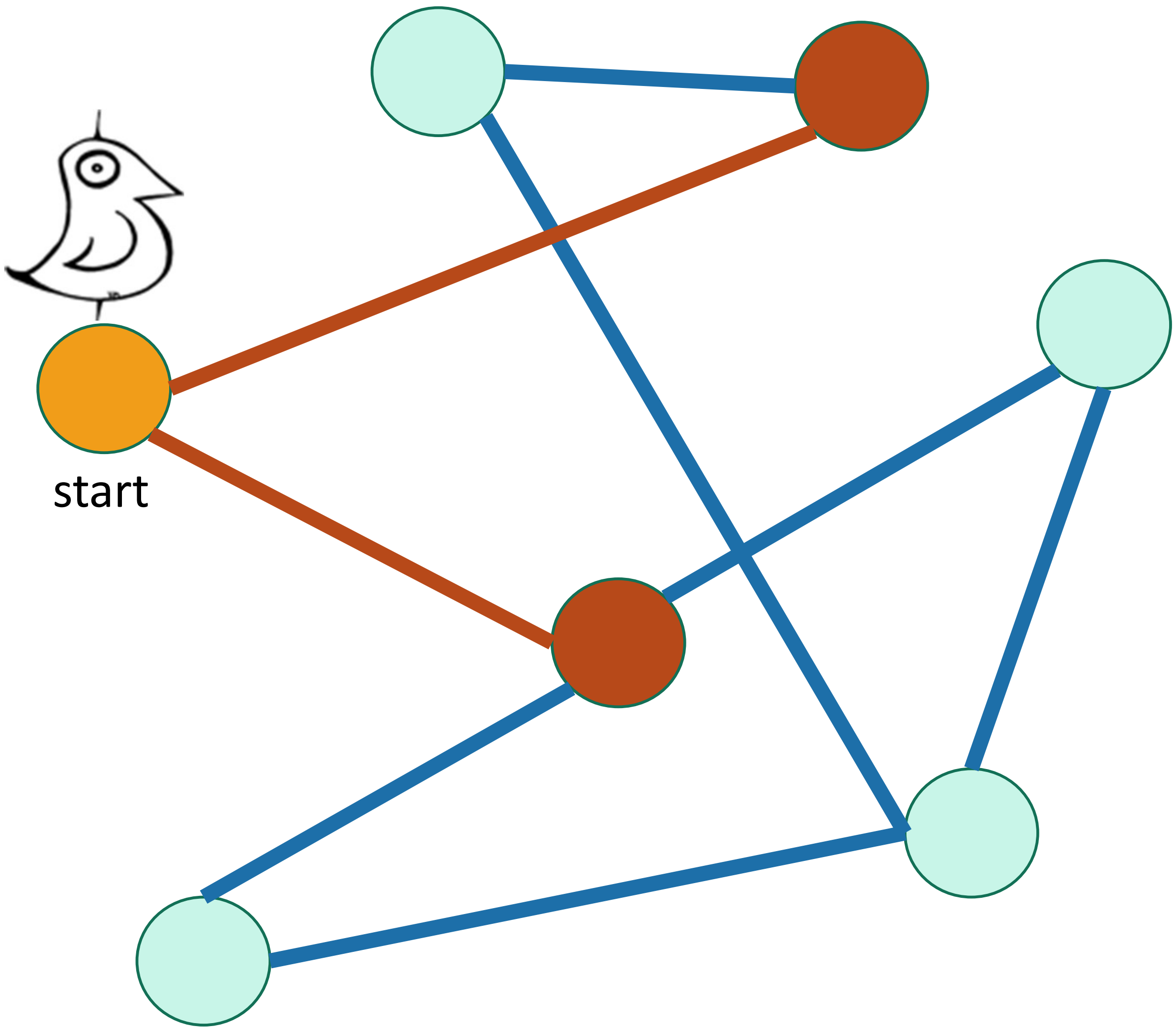
Breadth-First Search

For testing bipartite-ness



Breadth-First Search

For testing bipartite-ness



Not been there yet

Can reach there in zero steps

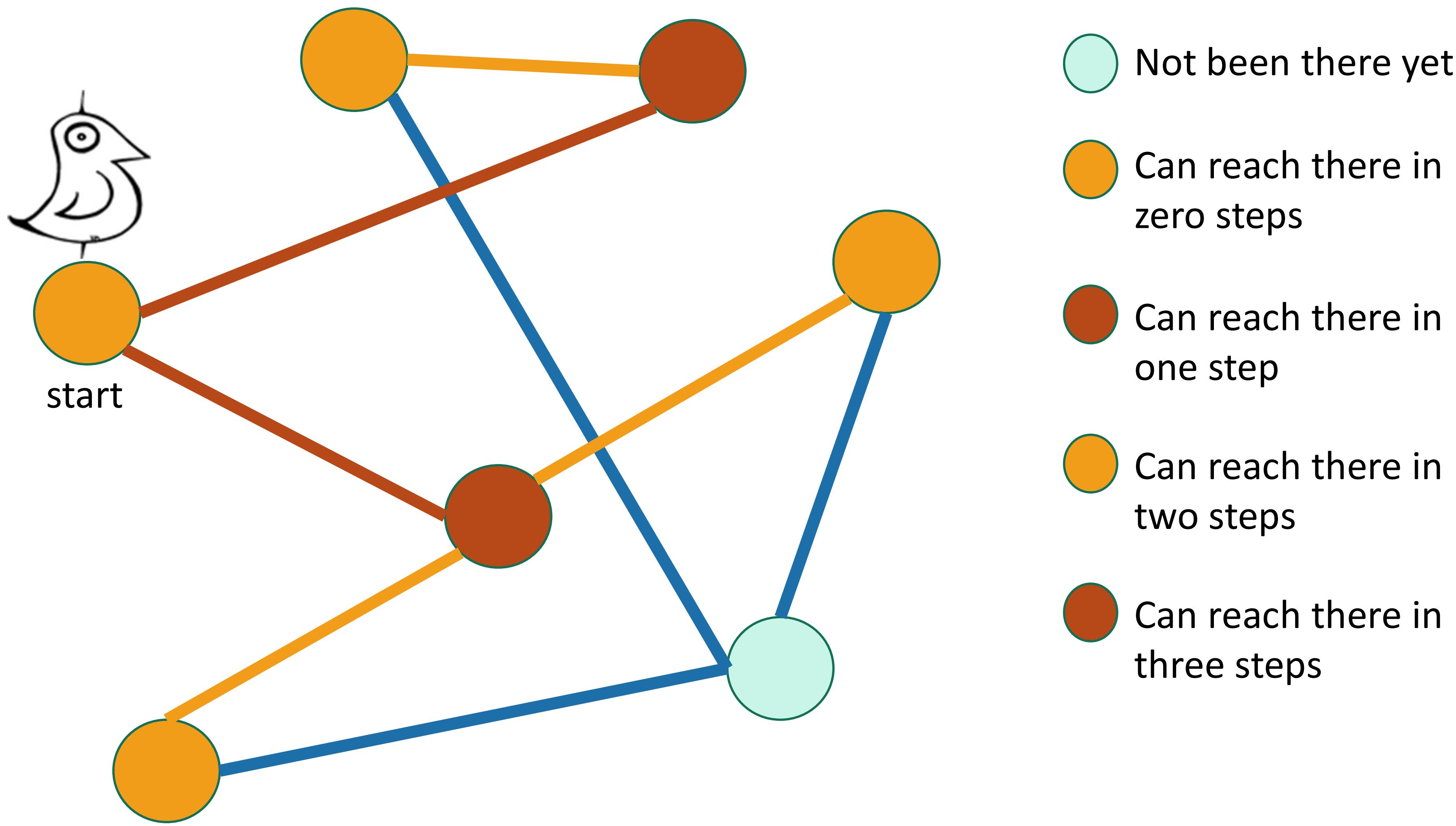
Can reach there in one step

Can reach there in two steps

Can reach there in three steps

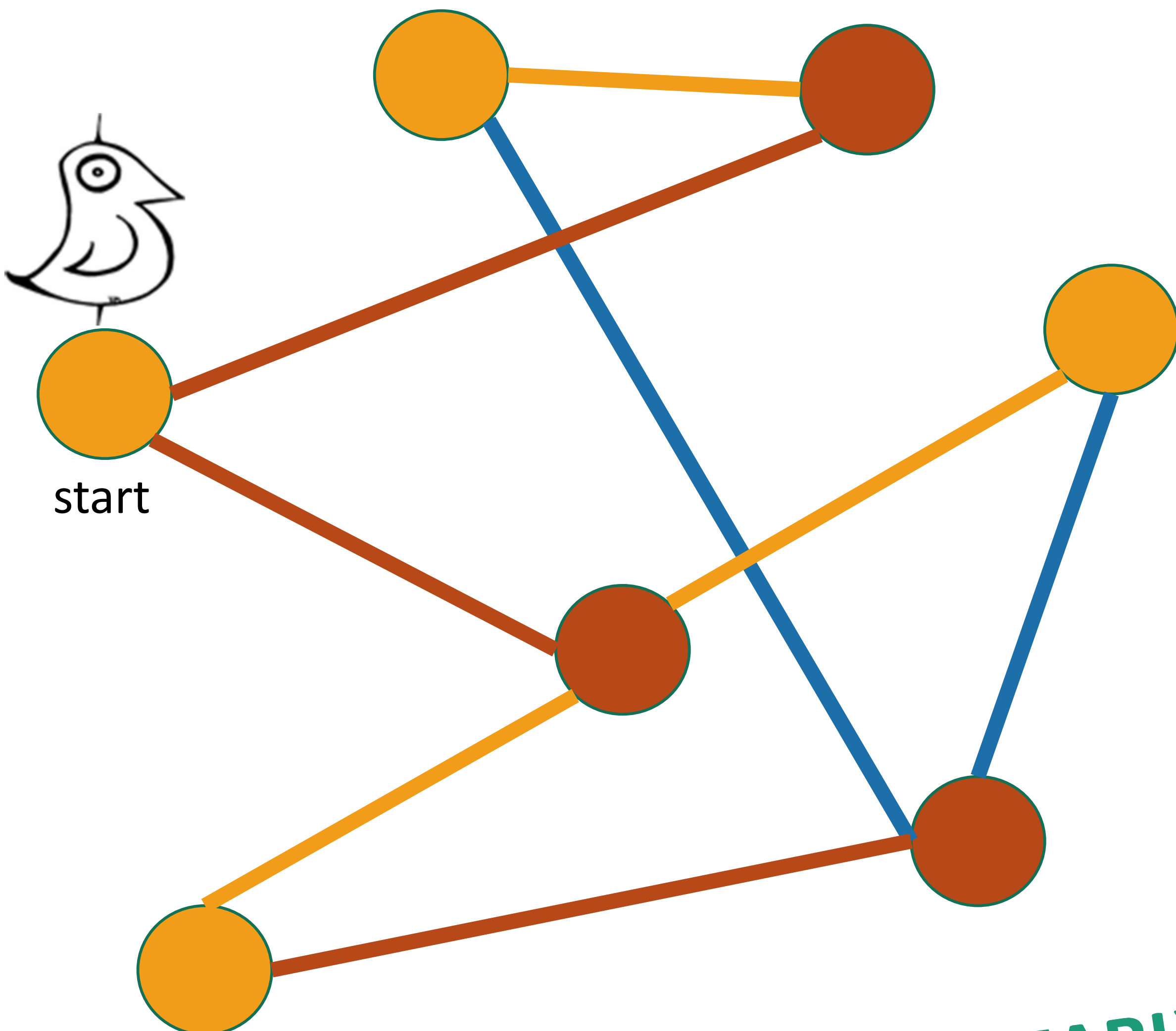
Breadth-First Search

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

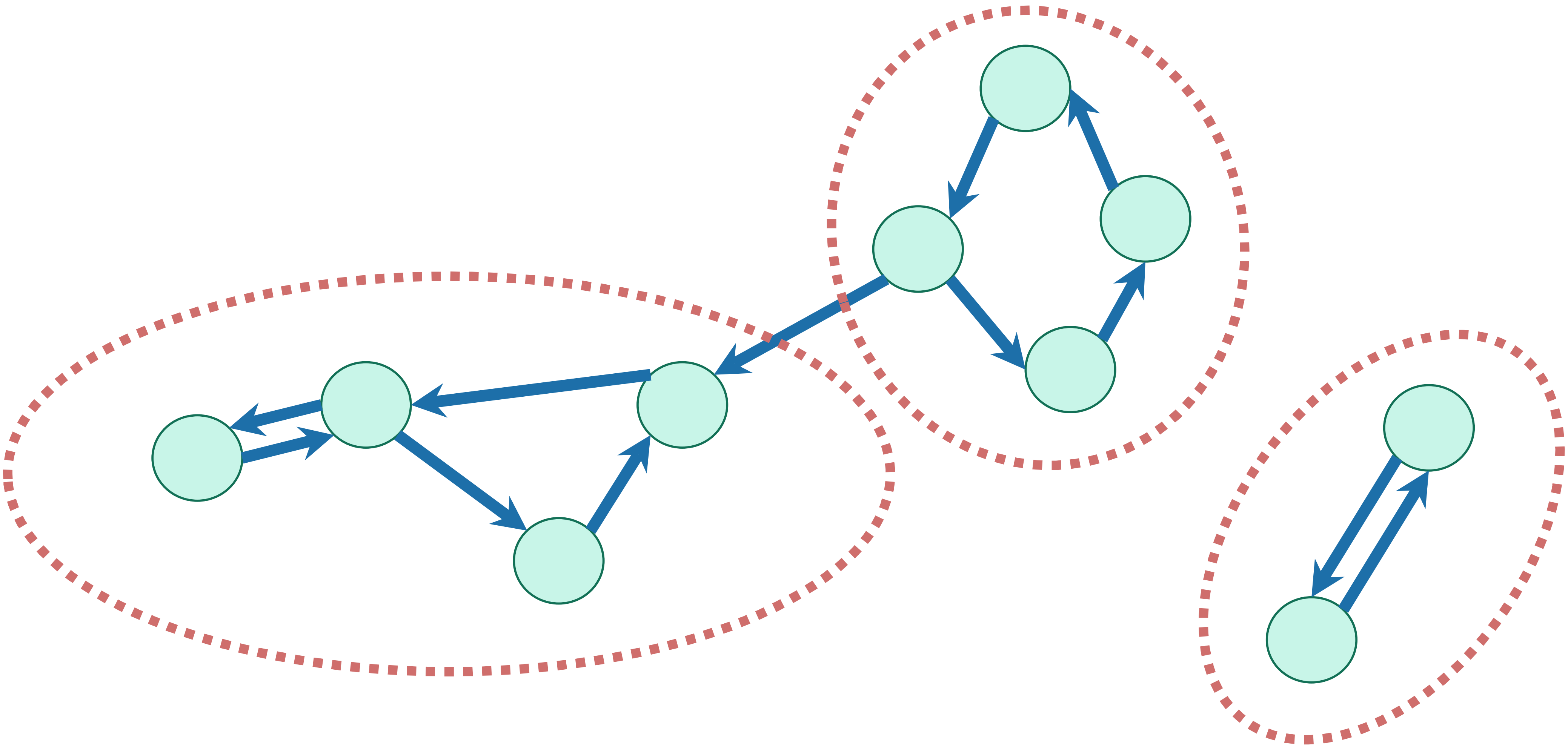
Can reach there in three steps

CLEARLY BIPARTITE!

Tổng kết

- Depth-first search
  - Sắp xếp topo (topological sorting)
  - Duyệt cây nhị phân (in-order)
- Breadth-first search
  - Tìm đường đi ngắn nhất trên đồ thị không trọng số
  - Kiểm tra đồ thị lưỡng phân
- Both DFS, BFS:
  - Khám phá đồ thị, tìm thành phần liên thông,...

Next time:
strongly connected components (SCCs)



Definition by definition: The SCCs are the equivalence classes under the “are mutually reachable” equivalence relation.