ساختمان داده و الگوریتم ها (CE203)

جلسه پانزدهم:
جستجوی سطح اول (BFS)

سجاد شیرعلی شهرضا

پاییز 1401

دوشنبه، 14 آذر 1401

Based on Stanford CS161 Slides from Summer 2021 by Karey Shi

اطلاع رسانی

- بخش مرتبط کتاب برای این جلسه: 22

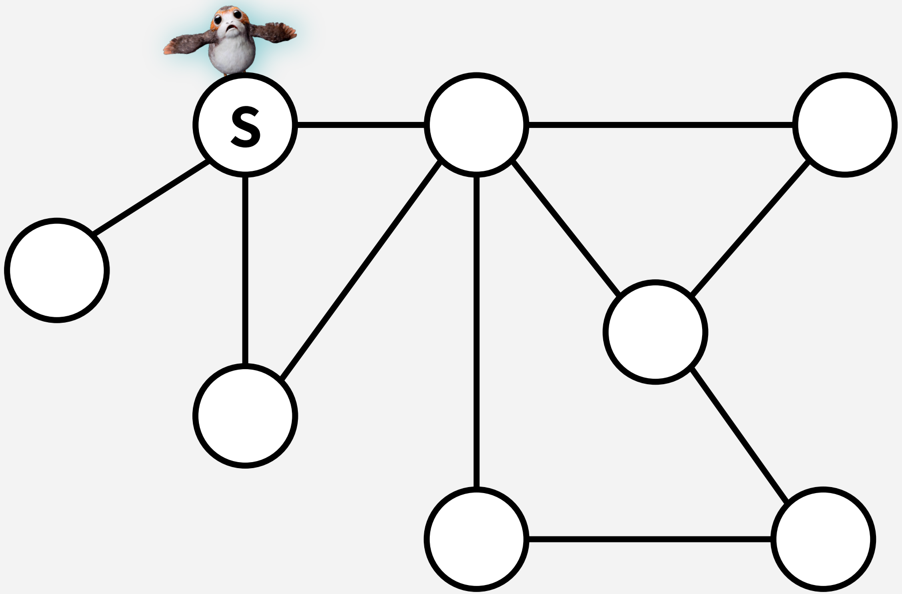
جستجوی سطح اول (BFS)

یک روش پیمایش گراف

BREADTH-FIRST SEARCH

An analogy:

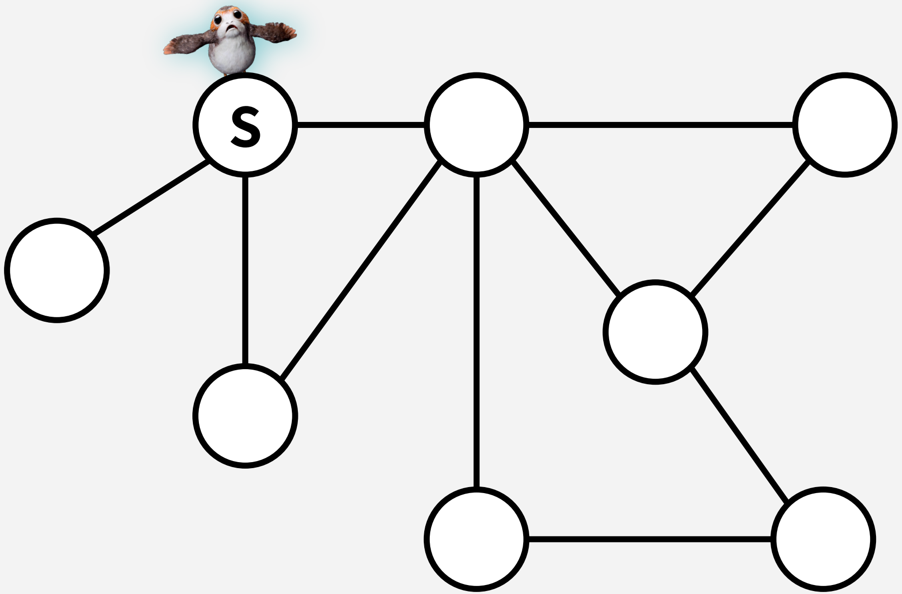
A bird is exploring a labyrinth from above (with a bird's eye view)

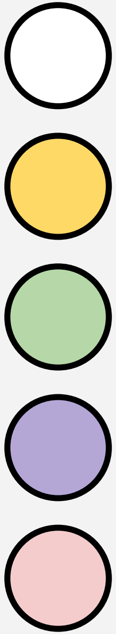


BREADTH-FIRST SEARCH

An analogy:

A bird is exploring a labyrinth from above (with a bird's eye view)





unvisited

Layer 0: reachable in 0 steps

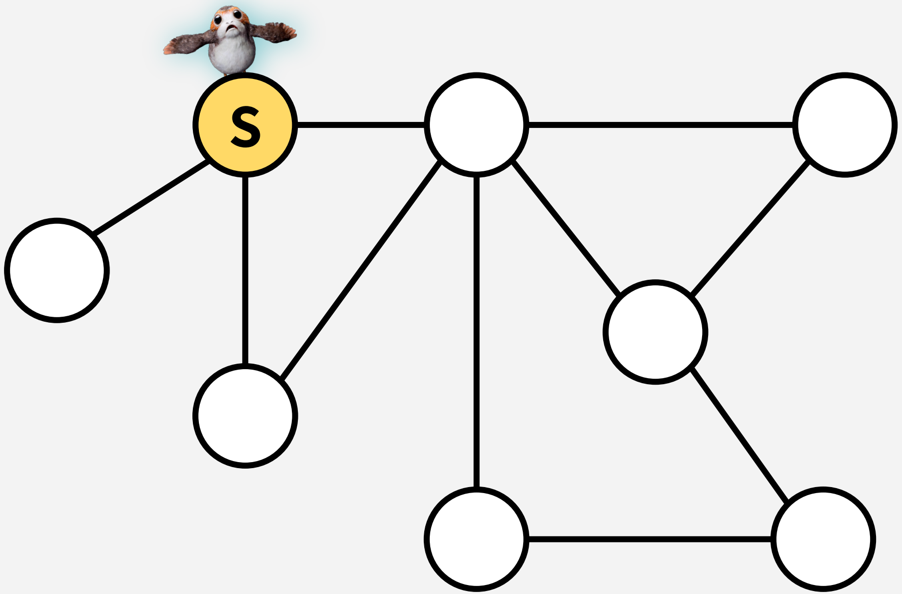
Layer 1: reachable in 1 step

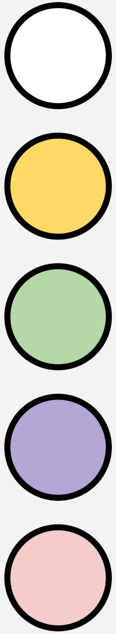
Layer 2: reachable in 2 steps

Layer 3: reachable in 3 steps

BREADTH-FIRST SEARCH

An analogy:
A bird is exploring a labyrinth from above (with a bird's eye view)

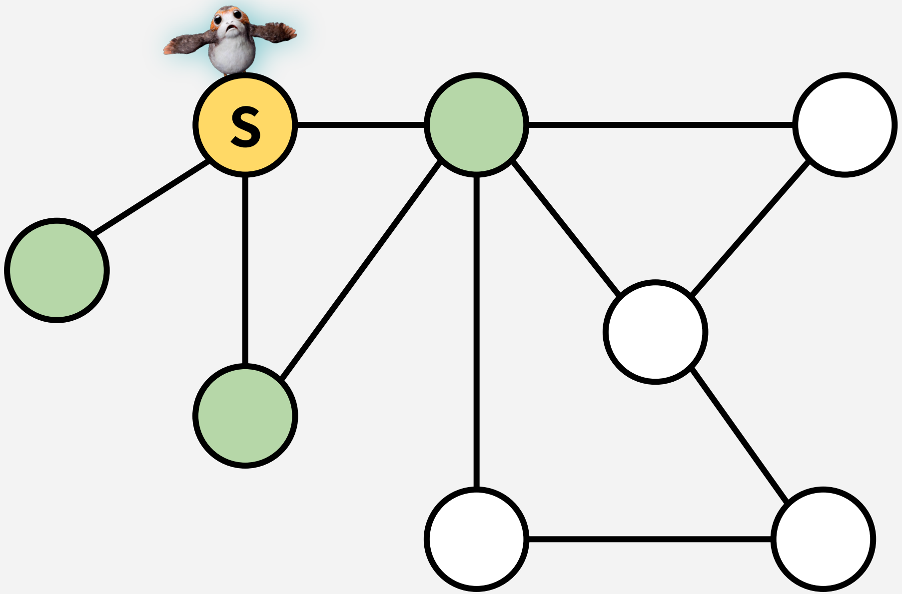




BREADTH-FIRST SEARCH

An analogy:

A bird is exploring a labyrinth from above (with a bird's eye view)



unvisited

Layer 0: reachable in 0 steps

Layer 1: reachable in 1 step

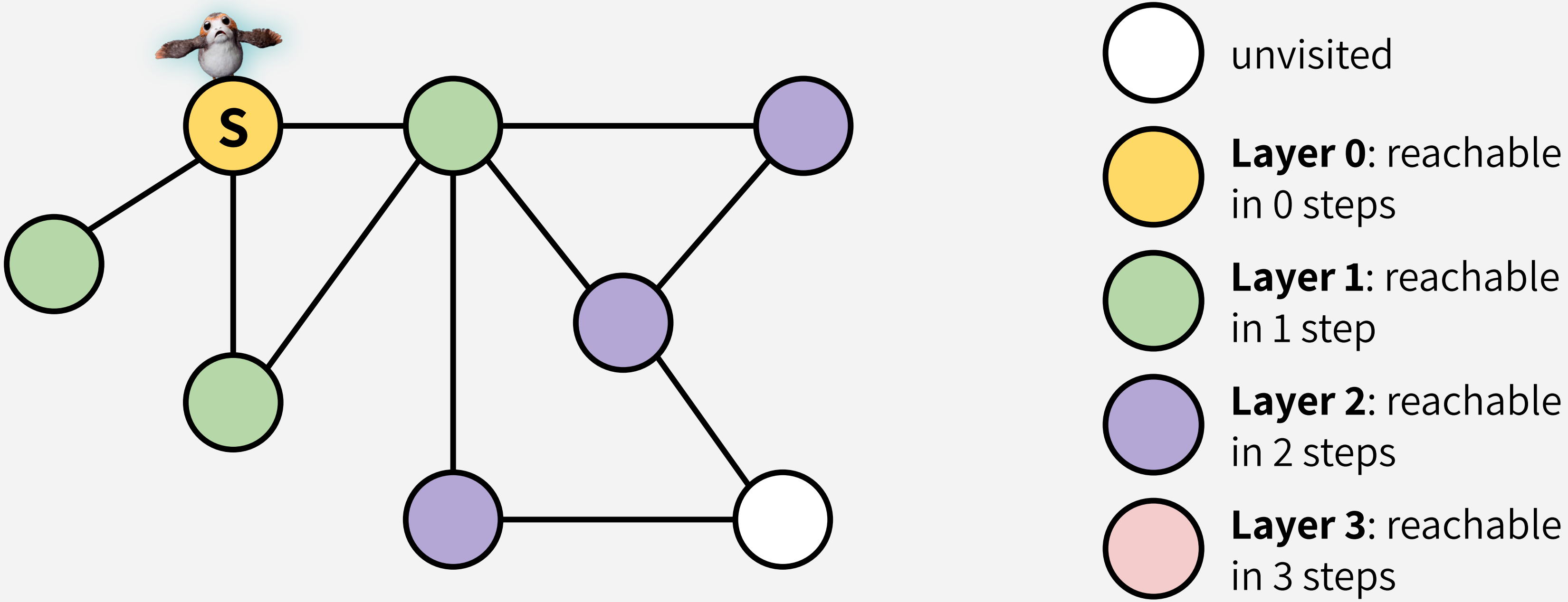
Layer 2: reachable in 2 steps

Layer 3: reachable in 3 steps

BREADTH-FIRST SEARCH

An analogy:

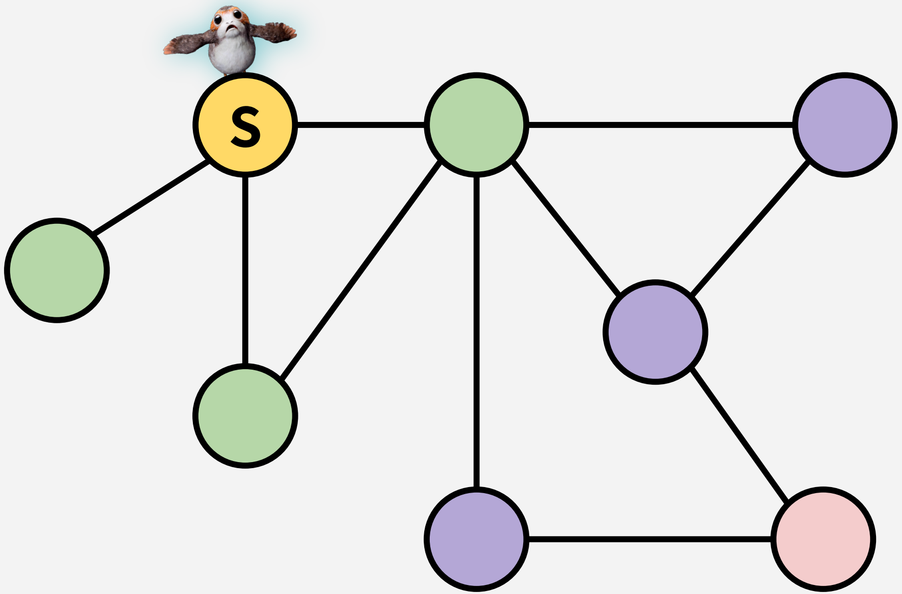
A bird is exploring a labyrinth from above (with a bird's eye view)



BREADTH-FIRST SEARCH

An analogy:

A bird is exploring a labyrinth from above (with a bird's eye view)



unvisited

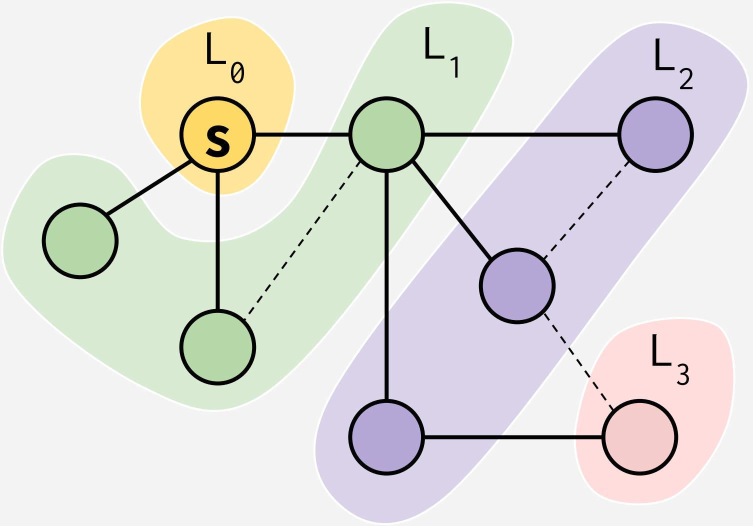
Layer 0: reachable in 0 steps

Layer 1: reachable in 1 step

Layer 2: reachable in 2 steps

Layer 3: reachable in 3 steps

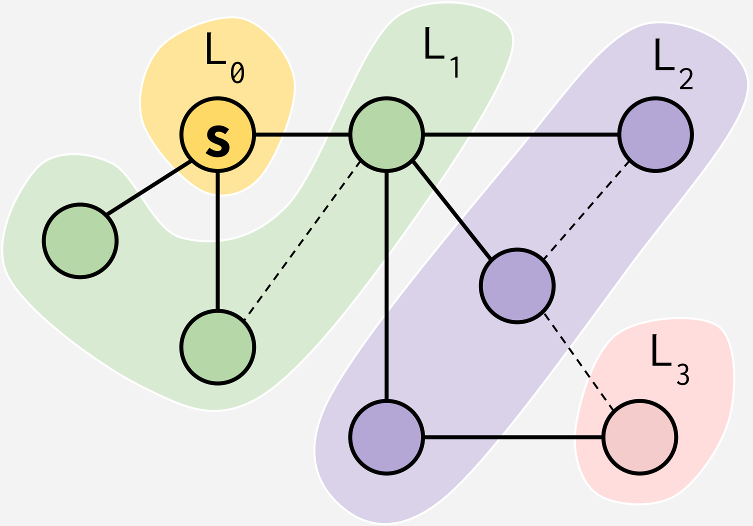
BREADTH-FIRST SEARCH




```
BFS(s):
    Set L_i = [] for i = 0, ..., n-1
    L_0 = s
    for i = 0, ..., n-1:
        for u in L_i:
            for v in u.neighbors:
                if v not yet visited:
                    mark v as visited
                    add v to L_{i+1}
```

L_i = The set of nodes we can reach in i steps from s

BREADTH-FIRST SEARCH

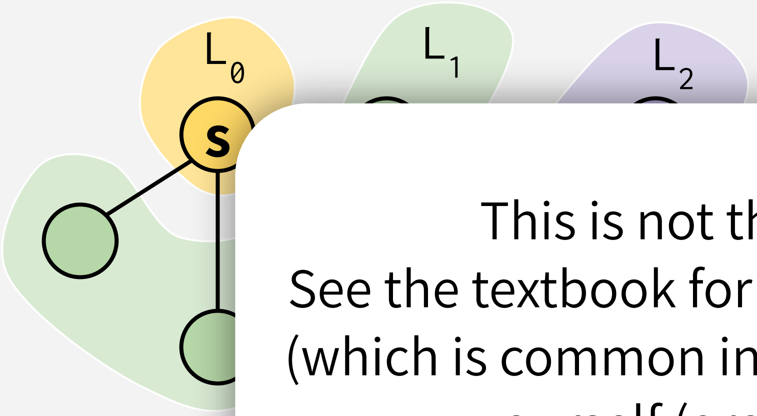


L_i = The set of nodes we can reach in i steps from s

```
BFS(s):
    Set L_i = [] for i = 0, ..., n-1
    L_0 = s
    for i = 0, ..., n-1:
        for u in L_i:
            for v in u.neighbors:
                if v not yet visited:
                    mark v as visited
                    add v to L_i+1
```

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

BREADTH-FIRST SEARCH




```
BFS(s):
    ..., n-1
    rs:
    sited:
    sited
```

This is not the only way to write BFS!
See the textbook for a version that uses a FIFO **queue** (which is common in the real world), and try writing it yourself (great for interview practice)

L_i = The set of nodes we can reach in i steps from s

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

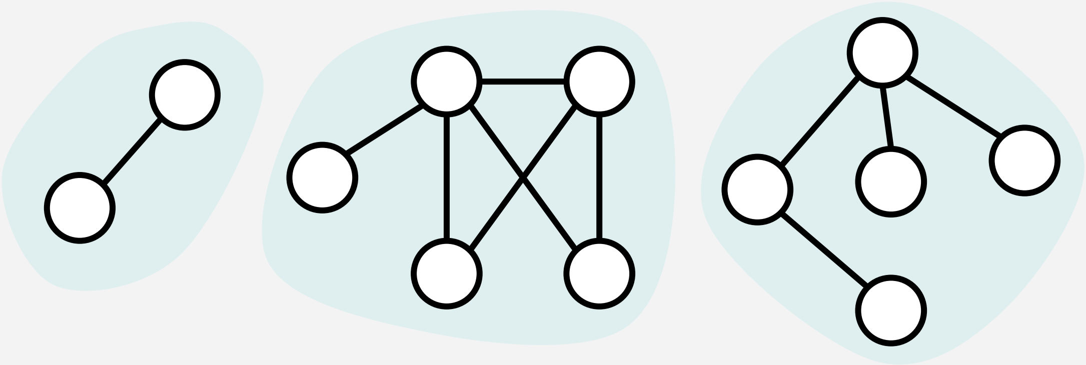
BREADTH-FIRST SEARCH

BFS finds all the nodes reachable from the starting point!

BREADTH-FIRST SEARCH

BFS finds all the nodes reachable from the starting point!

In undirected graphs, this is equivalent to finding the node's **connected component.**



BREADTH-FIRST SEARCH: RUNTIME

To explore a graph's **i^{th} connected component** (n_i nodes, m_i edges):

We visit each vertex in the CC exactly once ("visit" = grab from its L_i).

BREADTH-FIRST SEARCH: RUNTIME

To explore a graph's **i^{th} connected component** (n_i nodes, m_i edges):

We visit each vertex in the CC exactly once ("visit" = grab from its L_i).
At each vertex v, we:

- Do some bookkeeping: **O(1)**
- Loop over v's neighbors & check if they are visited (& then potentially mark the neighbor & place in L_{i+1}): O(1) per neighbor → **O(deg(v))** total.

BREADTH-FIRST SEARCH: RUNTIME

To explore a graph's $\mathbf{i^{th}}$ **connected component** (n_i nodes, m_i edges):

We visit each vertex in the CC exactly once ("visit" = grab from its L_i).
At each vertex v, we:

- Do some bookkeeping: **O(1)**
- Loop over v's neighbors & check if they are visited (& then potentially mark the neighbor & place in L_{i+1}): O(1) per neighbor → **O(deg(v))** total.

Total: $\sum_v O(\deg(v)) + \sum_v O(1) = \mathbf{O(m_i + n_i)}$

BREADTH-FIRST SEARCH: RUNTIME

To explore **the entire graph** (n nodes, m edges):

A graph might have multiple connected components! To **explore the whole graph**, we would call our BFS routine once for each connected component (note that each vertex and each edge participates in exactly one connected component). The combined running time would be:

$$O(\sum_i m_i + \sum_i n_i) = \mathbf{O(m + n)}$$

BREADTH-FIRST SEARCH

Why is it called breadth-first?

We are implicitly building a **tree**!

(It's a tree because we never revisit a node)

We go as "broadly" as we can when building each layer of the tree

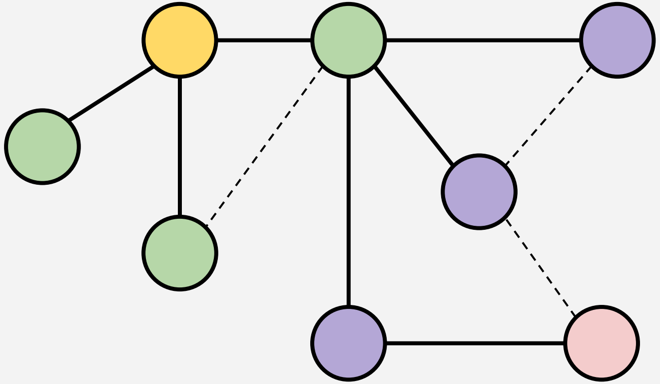
BREADTH-FIRST SEARCH

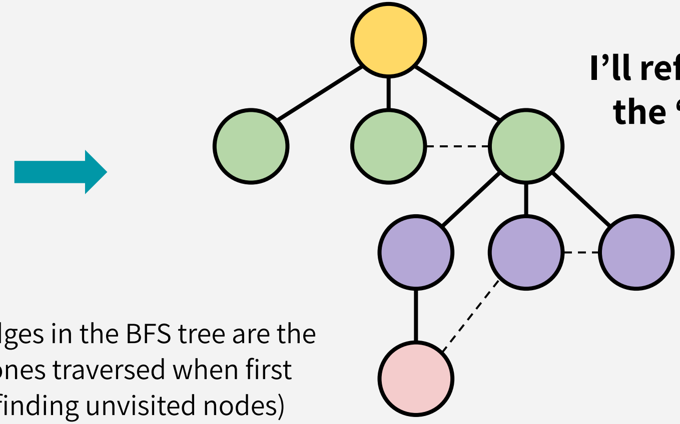
Why is it called breadth-first?

We are implicitly building a **tree**!

(It's a tree because we never revisit a node)

We go as "broadly" as we can when building each layer of the tree





I'll refer to this as the "BFS tree"

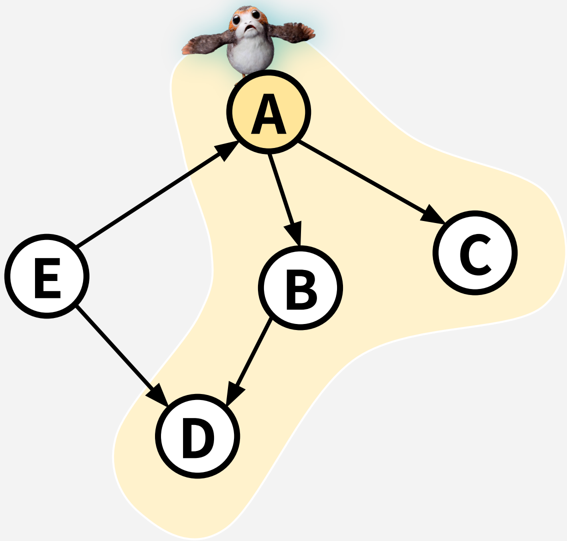
(Edges in the BFS tree are the ones traversed when first finding unvisited nodes)

BREADTH-FIRST SEARCH

BFS works fine on directed graphs too!

From a start node x, BFS would find all nodes ***reachable*** from x.

(In directed graphs, "connected component" isn't as well defined… more on that later!)



Verify this on your own: running BFS from A would still find all nodes reachable from A (E isn't reachable from A in this directed graph).

BREADTH-FIRST SEARCH

What are some applications of BFS?

Finding a node's connected component (just run BFS)!
(or in directed graphs, finding reachable nodes from a starting node)

Single-source shortest paths

Testing bipartiteness

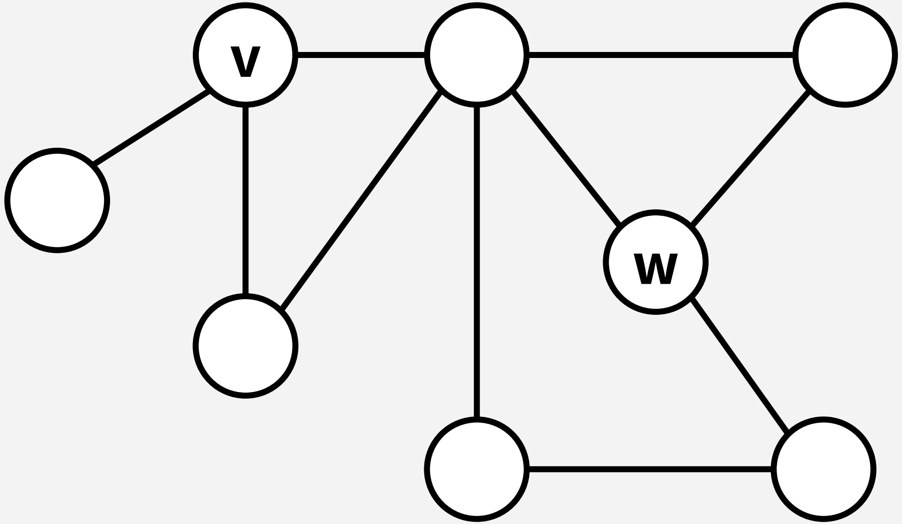
And more…

سوال؟

پیدا کردن کوتاه ترین مسیر با جستجوی سطح اول

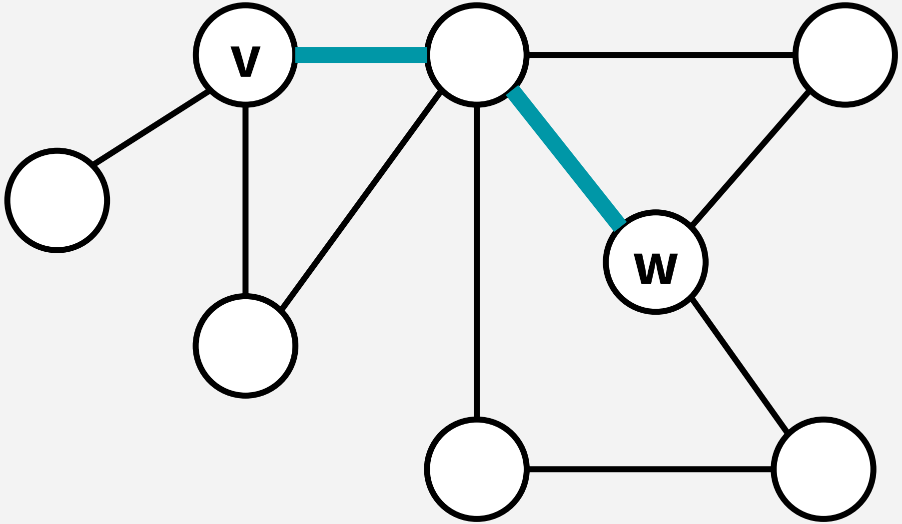
SHORTEST PATH: THE TASK

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



SHORTEST PATH: THE TASK

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

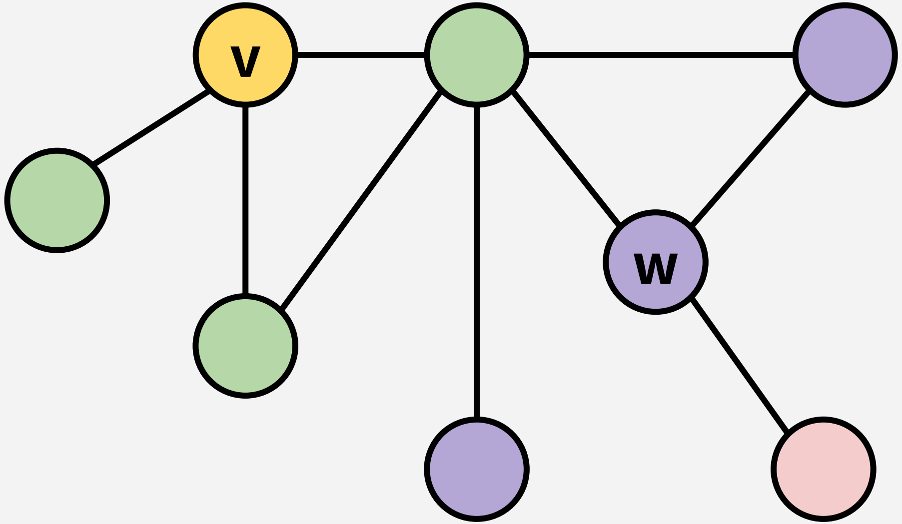


From visually inspecting the graph, we can see that the shortest path from **v** to **w** is 2 (there are 2 edges on that path)!

There are paths of length 3, 4, or 5 as well, but we can't do any better than 2.

SHORTEST PATH: THE TASK

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



unvisited

Layer 0: reachable in 0 steps

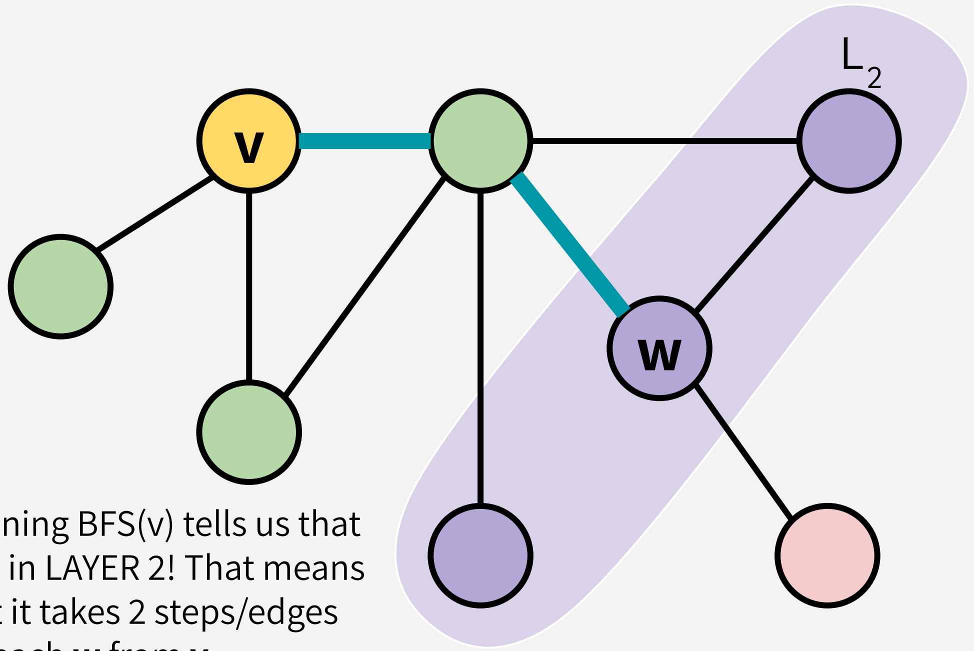
Layer 1: reachable in 1 step

Layer 2: reachable in 2 steps

Layer 3: reachable in 3 steps

SHORTEST PATH: THE TASK

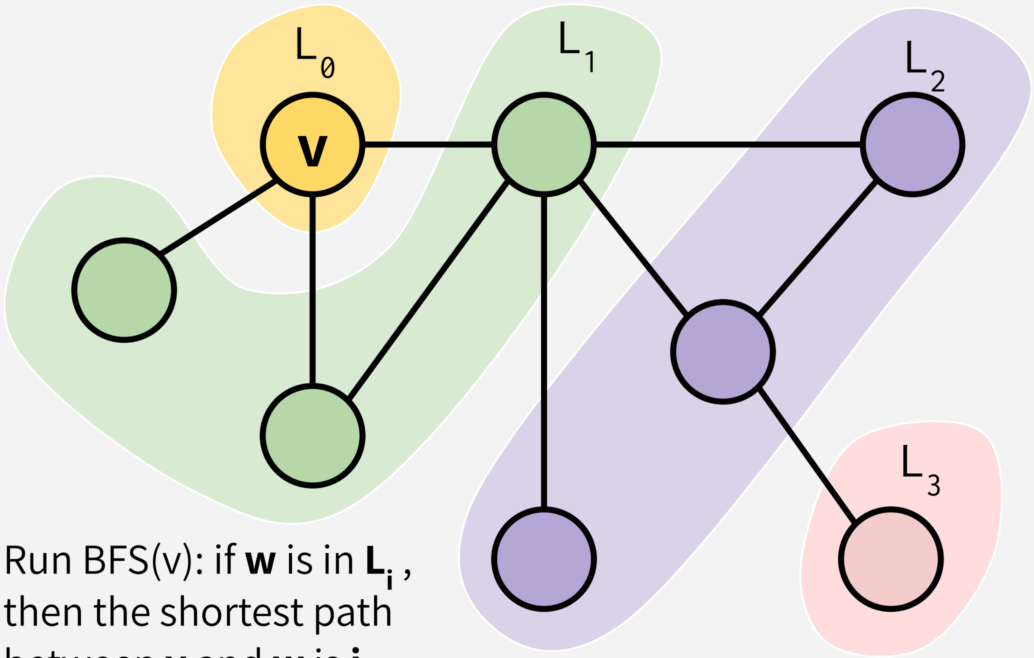
How long is the shortest path between vertices v and w?



Running BFS(v) tells us that **w** is in LAYER 2! That means that it takes 2 steps/edges to reach **w** from **v**.

SINGLE-SOURCE SHORTEST PATH

How long is the shortest path between vertices v & *all other vertices* w?



Run BFS(v): if **w** is in $\mathbf{L_i}$, then the shortest path between **v** and **w** is **i**

unvisited

Layer 0: reachable in 0 steps

Layer 1: reachable in 1 step

Layer 2: reachable in 2 steps

Layer 3: reachable in 3 steps

SINGLE-SOURCE SHORTEST PATH

How long is the shortest path between vertices v & *all other vertices* w?

```
findAllDistances(v):
    perform BFS(v) → gives us all L_i
    for all w in V:
        d[w] = ∞
    for each L_i:
        for all w in L_i:
            d[w] = i
```

SINGLE-SOURCE SHORTEST PATH

How long is the shortest path between vertices v & *all other vertices* w?

```
findAllDistances(v):
    perform BFS(v) → gives us all L_i
    for all w in V:
        d[w] = ∞
    for each L_i:
        for all w in L_i:
            d[w] = i
```

This is something you can repurpose on your problem set or quiz questions: **you may assume that running BFS(v) also provides you all the levels L_i**

Runtime: O(m+n)

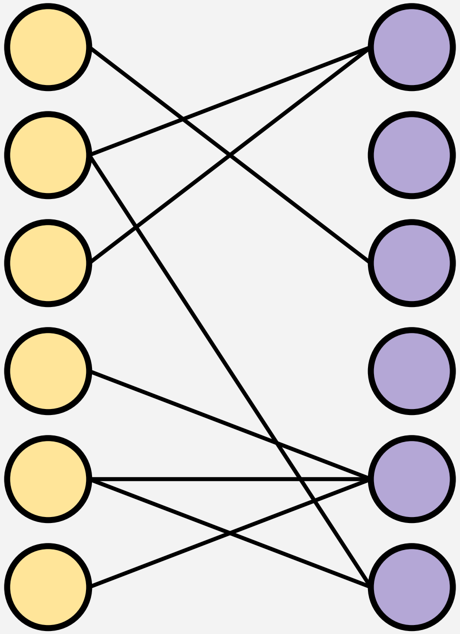
سوال؟

آزمایش دوبخشی بودن گراف

استفاده از جستجوی سطح اول برای آزمایش دوبخشی بودن گراف

BIPARTITE GRAPHS

A graph is **bipartite** iff there exists a 2-coloring such that there are no edges between same-colored vertices

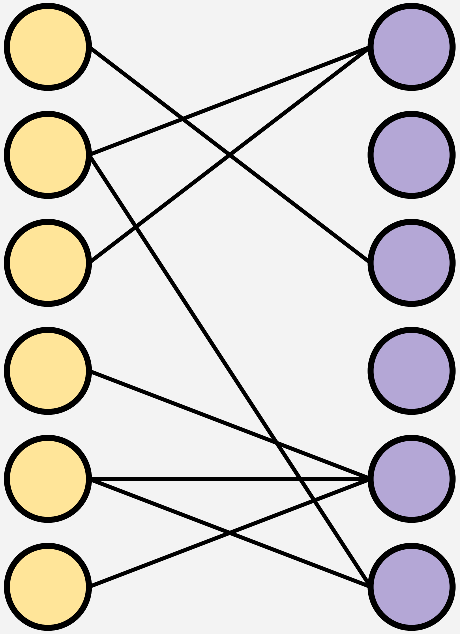


BIPARTITE GRAPHS

A graph is **bipartite** iff there exists a 2-coloring such that there are no edges between same-colored vertices

Example 1:

You're planning a cross-grade buddy system for 3rd and 4th graders, and you polled everyone's preferences for buddies. Can you verify that no students were listing someone from their same grade as one of their top choices?

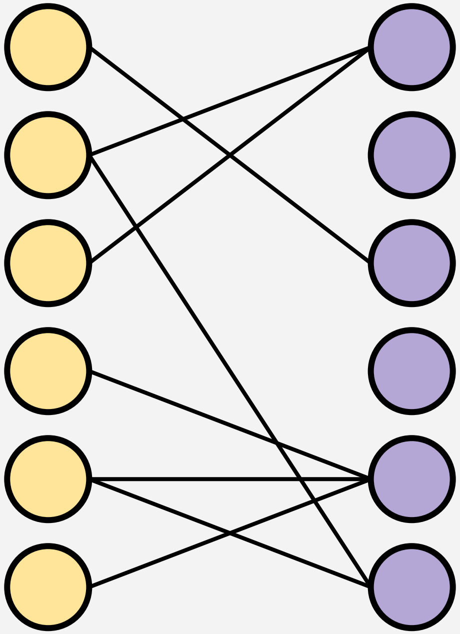


BIPARTITE GRAPHS

A graph is **bipartite** iff there exists a 2-coloring such that there are no edges between same-colored vertices

Example 1:

You're planning a cross-grade buddy system for 3rd and 4th graders, and you polled everyone's preferences for buddies. Can you verify that no students were listing someone from their same grade as one of their top choices?

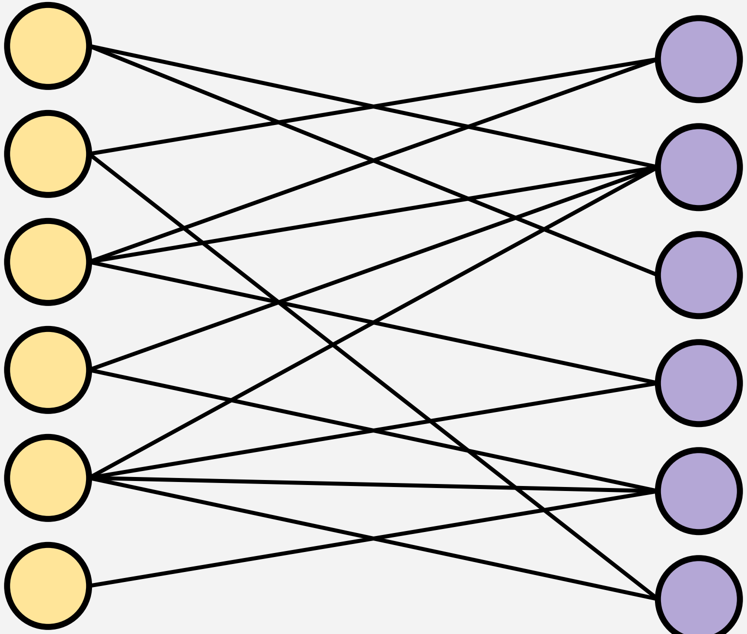


Example 2:

You have a bunch of fish and two fish tanks; some pairs of fish will fight if they're in the same tank. Can you separate the fish so that there's no fighting?

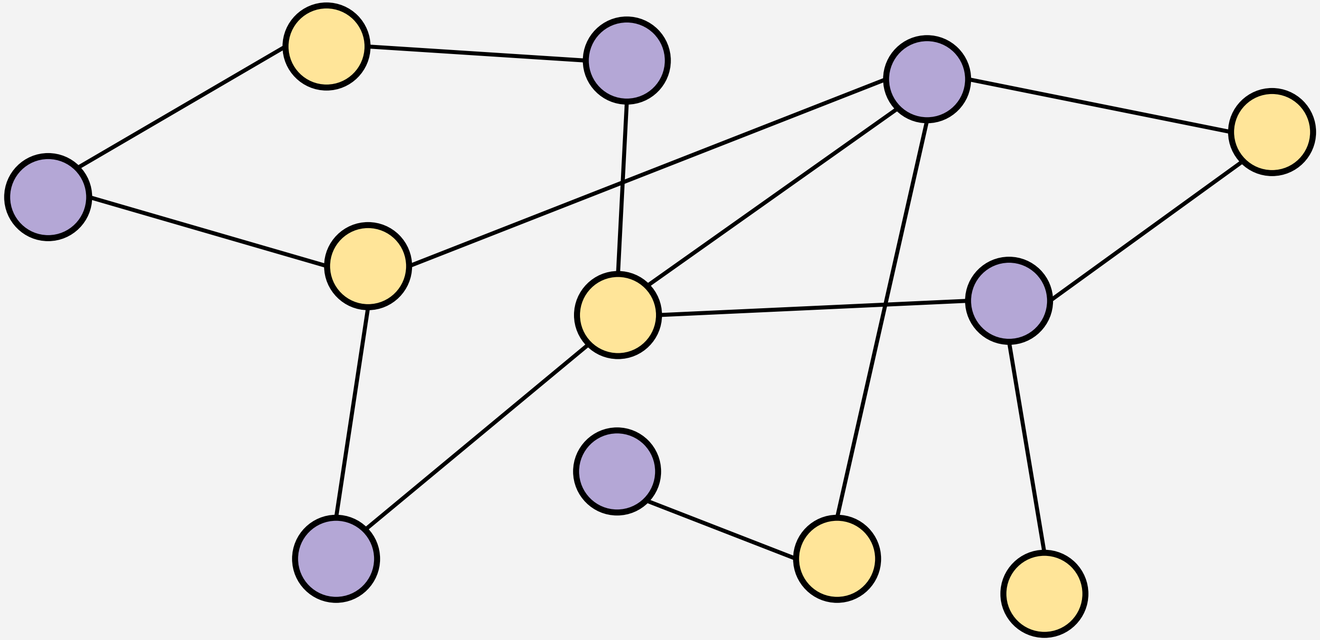
BIPARTITE GRAPHS

Is this graph bipartite?



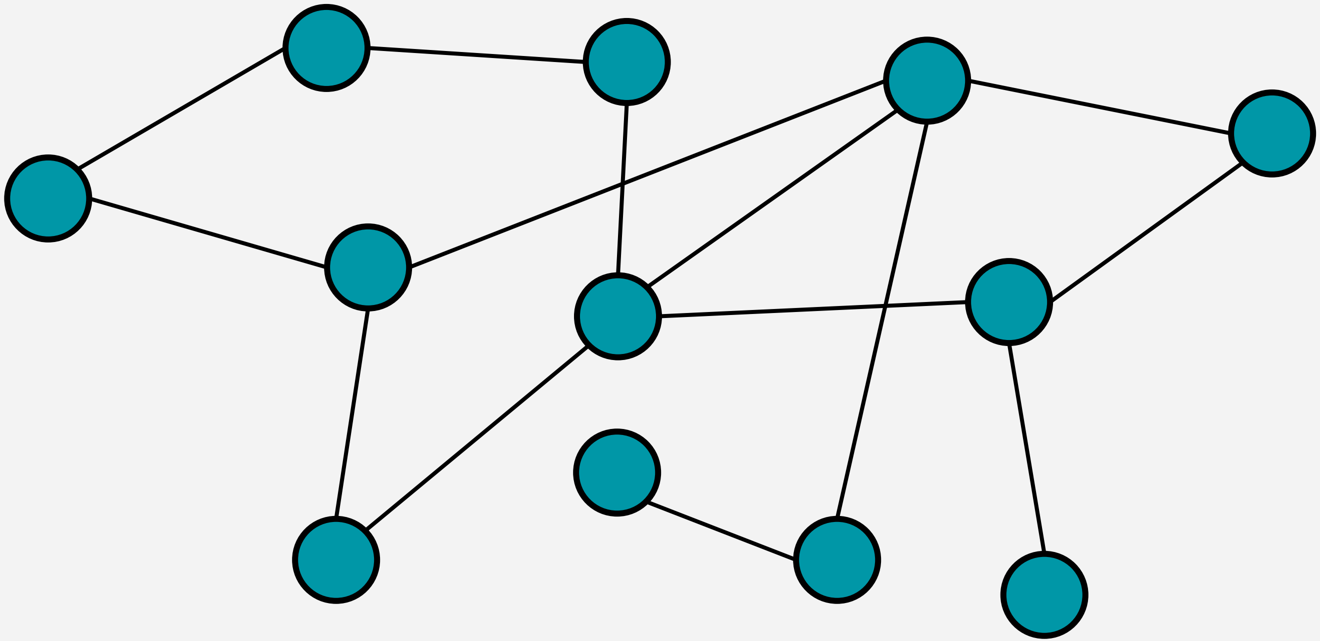
BIPARTITE GRAPHS

How about this one?



BIPARTITE GRAPHS

How about this one?



BIPARTITE GRAPHS

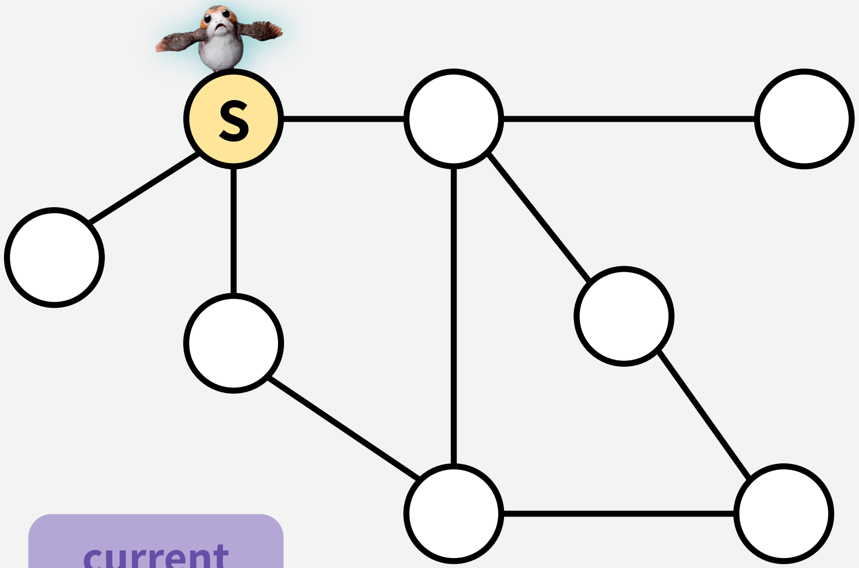
Application of BFS:

- Color the levels of the BFS tree in alternating colors (i.e. run BFS from any vertex, and alternate colors for each layer)
- If you attempt to color the same vertex different colors (i.e. revisit a node that's a different color than what you would have colored it), then the graph isn't bipartite!
- If you successfully color the whole graph without conflicts, then it is bipartite!

BIPARTITE GRAPHS

Application of BFS:

- **Color the levels of the BFS tree in alternating colors** (i.e. run BFS from any vertex, and alternate colors for each layer)
- **If you attempt to color the same vertex different colors** (i.e. revisit a node that's a different color than what you would have colored it), **then the graph isn't bipartite!**
- **If you successfully color the whole graph without conflicts, then it is bipartite!**

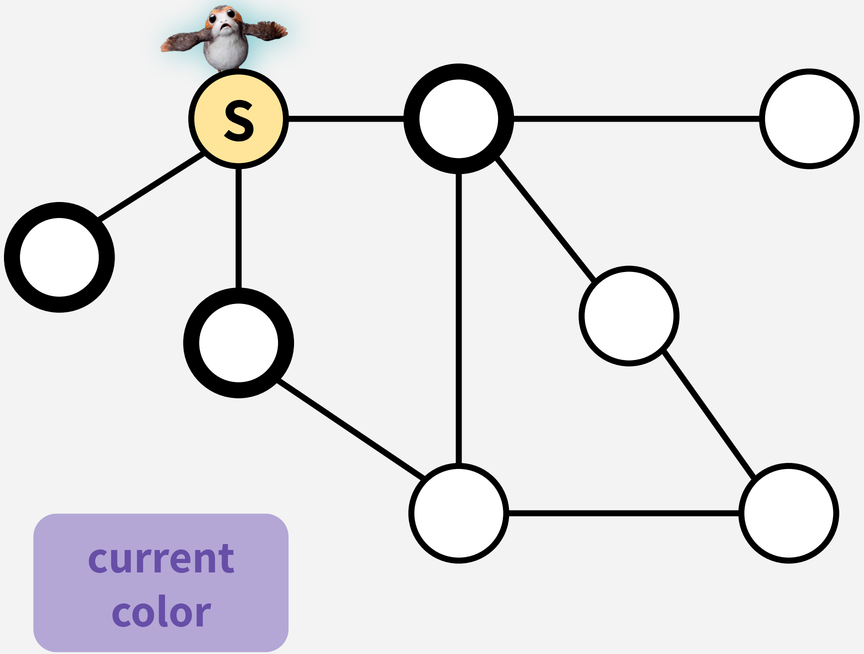


current color

BIPARTITE GRAPHS

Application of BFS:

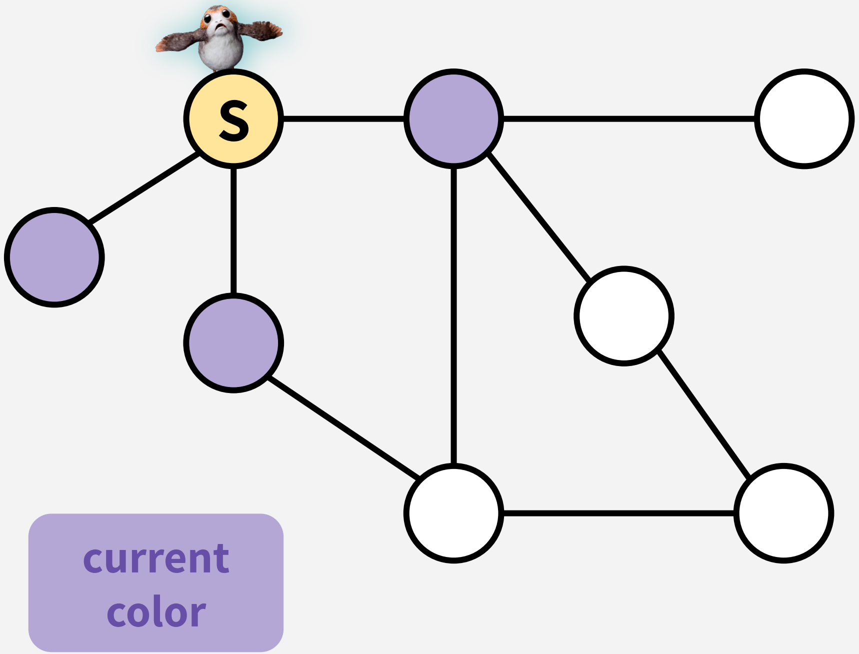
- Color the levels of the BFS tree in alternating colors (i.e. run BFS from any vertex, and alternate colors for each layer)
- If you attempt to color the same vertex different colors (i.e. revisit a node that's a different color than what you would have colored it), then the graph isn't bipartite!
- If you successfully color the whole graph without conflicts, then it is bipartite!



BIPARTITE GRAPHS

Application of BFS:

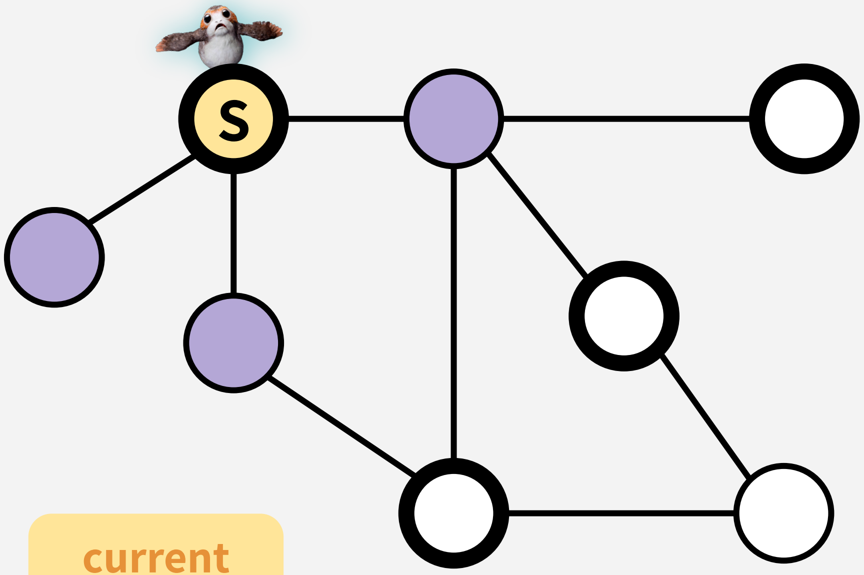
- **Color the levels of the BFS tree in alternating colors** (i.e. run BFS from any vertex, and alternate colors for each layer)
- **If you attempt to color the same vertex different colors** (i.e. revisit a node that's a different color than what you would have colored it), **then the graph isn't bipartite!**
- **If you successfully color the whole graph without conflicts, then it is bipartite!**



BIPARTITE GRAPHS

Application of BFS:

- **Color the levels of the BFS tree in alternating colors** (i.e. run BFS from any vertex, and alternate colors for each layer)
- **If you attempt to color the same vertex different colors** (i.e. revisit a node that's a different color than what you would have colored it), **then the graph isn't bipartite!**
- **If you successfully color the whole graph without conflicts, then it is bipartite!**

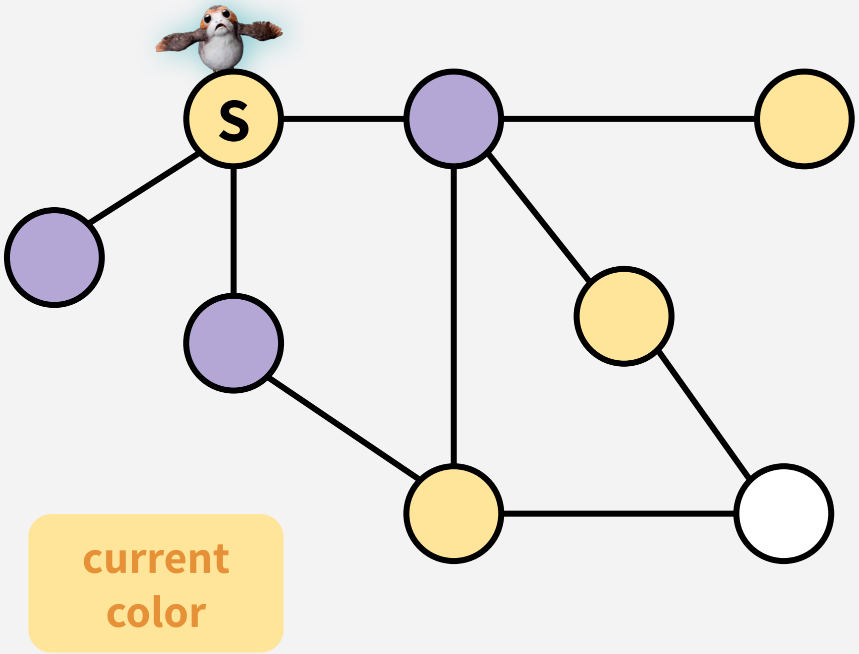


current color

BIPARTITE GRAPHS

Application of BFS:

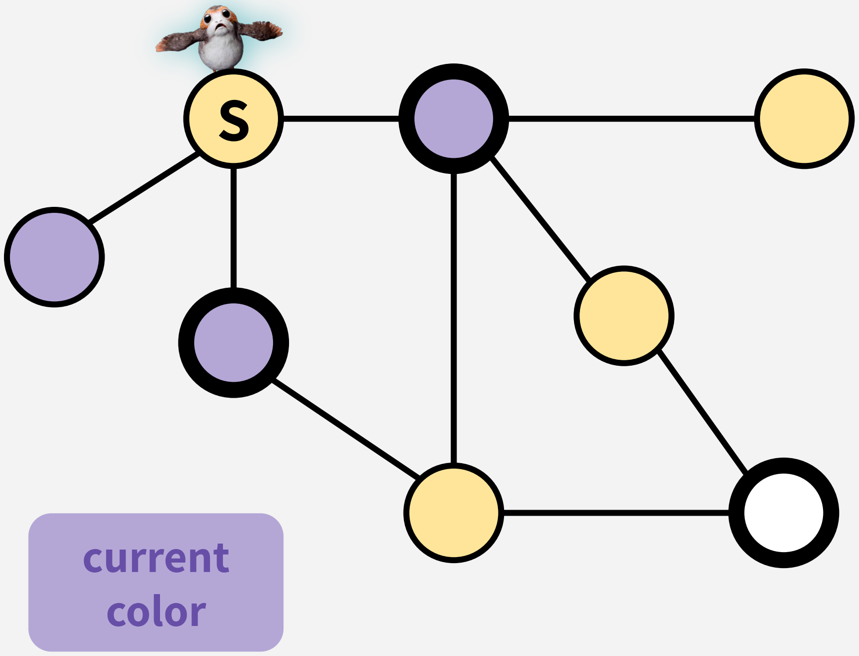
- **Color the levels of the BFS tree in alternating colors** (i.e. run BFS from any vertex, and alternate colors for each layer)
- **If you attempt to color the same vertex different colors** (i.e. revisit a node that's a different color than what you would have colored it), **then the graph isn't bipartite!**
- **If you successfully color the whole graph without conflicts, then it is bipartite!**



BIPARTITE GRAPHS

Application of BFS:

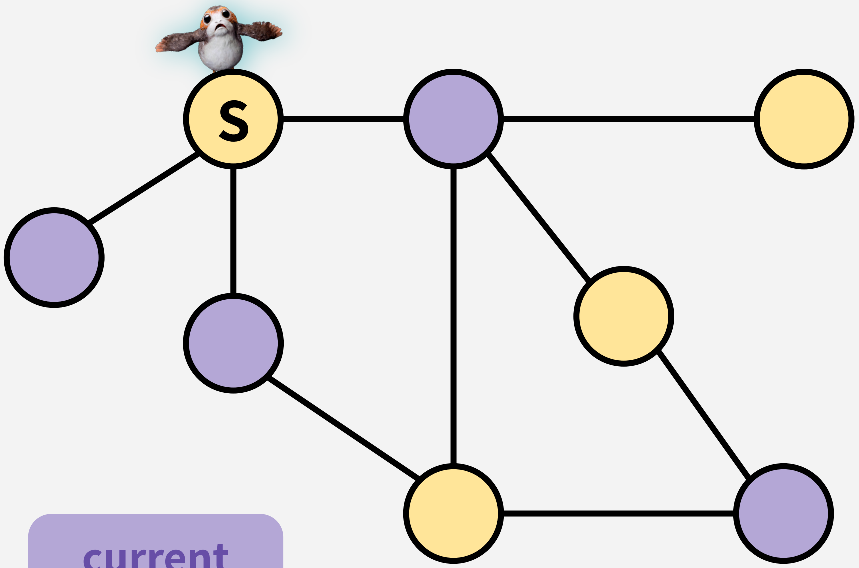
- **Color the levels of the BFS tree in alternating colors** (i.e. run BFS from any vertex, and alternate colors for each layer)
- **If you attempt to color the same vertex different colors** (i.e. revisit a node that's a different color than what you would have colored it), **then the graph isn't bipartite!**
- **If you successfully color the whole graph without conflicts, then it is bipartite!**



BIPARTITE GRAPHS

Application of BFS:

- **Color the levels of the BFS tree in alternating colors** (i.e. run BFS from any vertex, and alternate colors for each layer)
- **If you attempt to color the same vertex different colors** (i.e. revisit a node that's a different color than what you would have colored it)**, then the graph isn't bipartite!**
- **If you successfully color the whole graph without conflicts, then it is bipartite!**

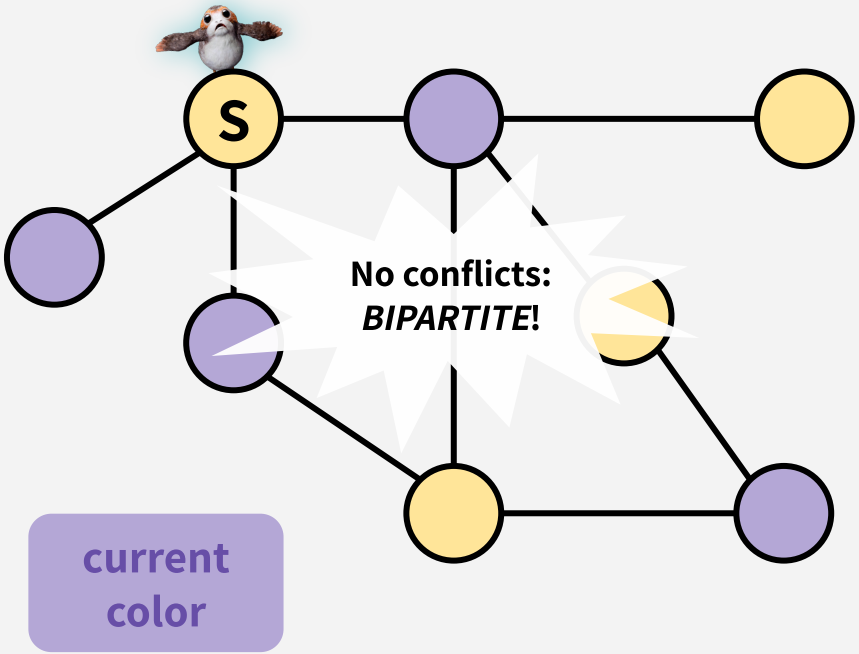


current color

BIPARTITE GRAPHS

Application of BFS:

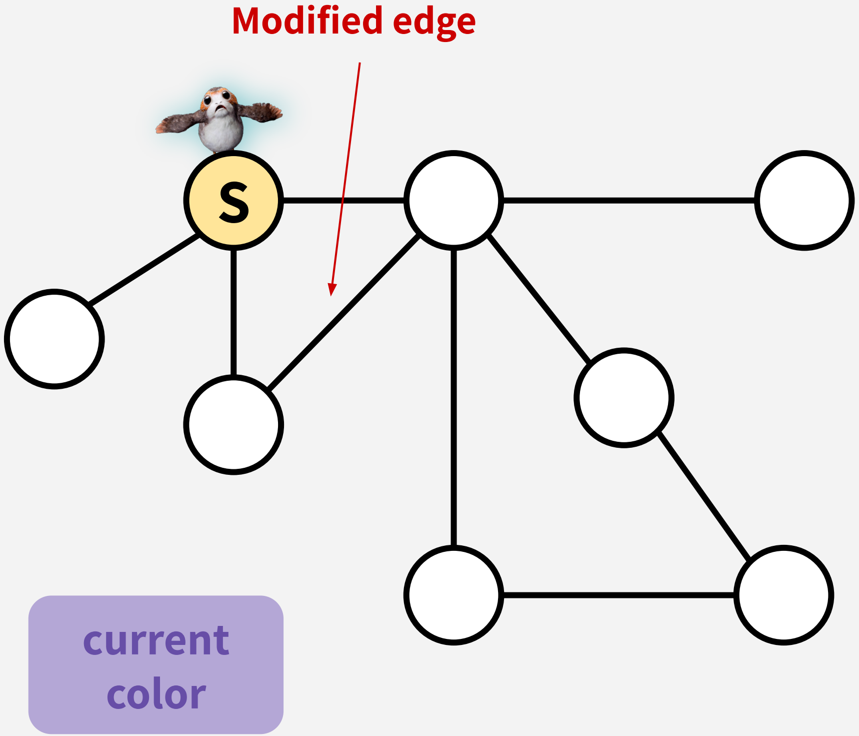
- **Color the levels of the BFS tree in alternating colors** (i.e. run BFS from any vertex, and alternate colors for each layer)
- **If you attempt to color the same vertex different colors** (i.e. revisit a node that's a different color than what you would have colored it), **then the graph isn't bipartite!**
- **If you successfully color the whole graph without conflicts, then it is bipartite!**



BIPARTITE GRAPHS

Application of BFS:

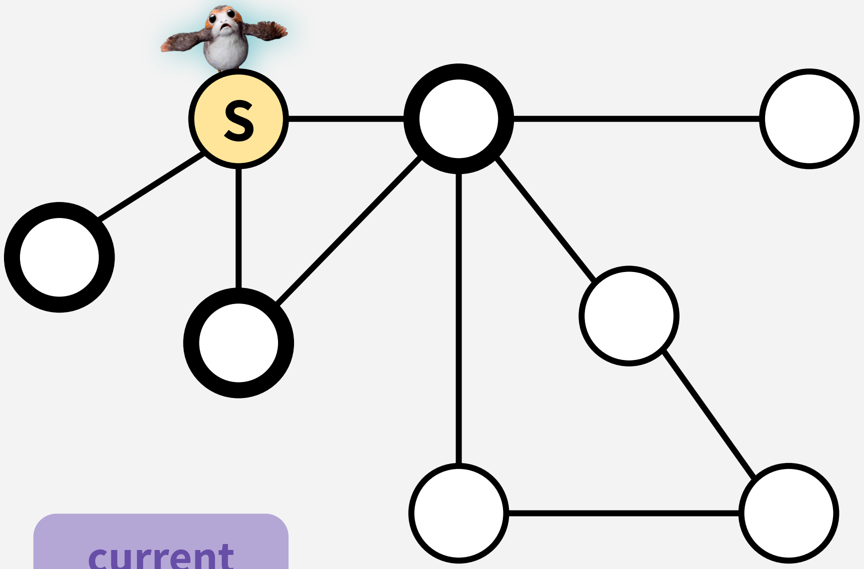
- Color the levels of the BFS tree in alternating colors (i.e. run BFS from any vertex, and alternate colors for each layer)
- If you attempt to color the same vertex different colors (i.e. revisit a node that's a different color than what you would have colored it), then the graph isn't bipartite!
- If you successfully color the whole graph without conflicts, then it is bipartite!



BIPARTITE GRAPHS

Application of BFS:

- **Color the levels of the BFS tree in alternating colors** (i.e. run BFS from any vertex, and alternate colors for each layer)
- **If you attempt to color the same vertex different colors** (i.e. revisit a node that's a different color than what you would have colored it), **then the graph isn't bipartite!**
- **If you successfully color the whole graph without conflicts, then it is bipartite!**

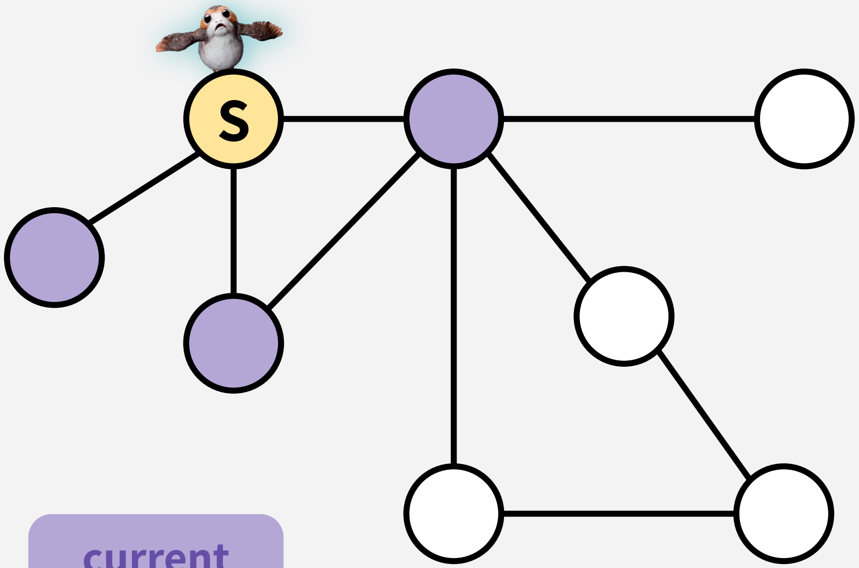


current color

BIPARTITE GRAPHS

Application of BFS:

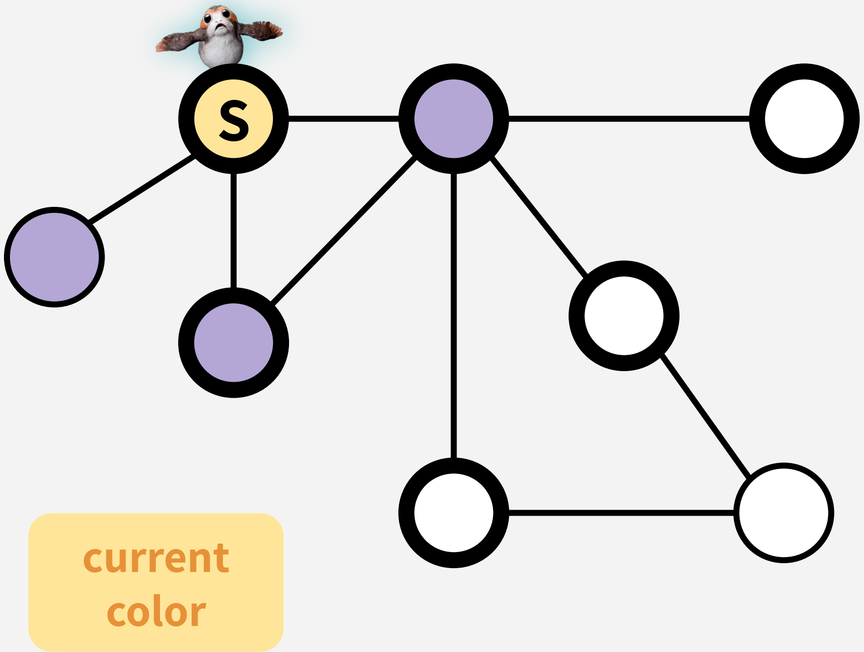
- Color the levels of the BFS tree in **alternating colors** (i.e. run BFS from any vertex, and alternate colors for each layer)
- **If you attempt to color the same vertex different colors** (i.e. revisit a node that's a different color than what you would have colored it), **then the graph isn't bipartite!**
- **If you successfully color the whole graph without conflicts, then it is bipartite!**



BIPARTITE GRAPHS

Application of BFS:

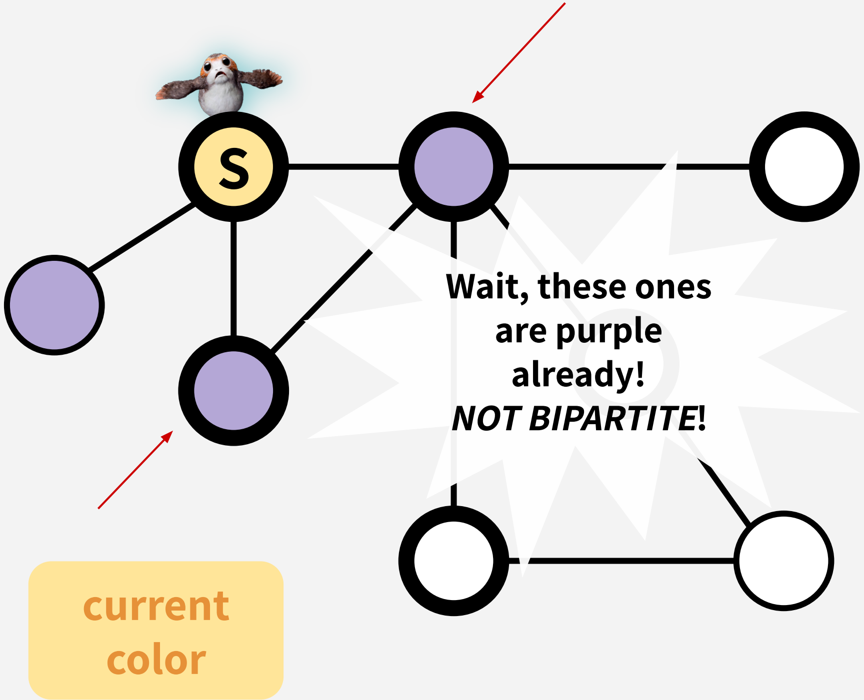
- Color the levels of the BFS tree in alternating colors (i.e. run BFS from any vertex, and alternate colors for each layer)
- If you attempt to color the same vertex different colors (i.e. revisit a node that's a different color than what you would have colored it), then the graph isn't bipartite!
- If you successfully color the whole graph without conflicts, then it is bipartite!



BIPARTITE GRAPHS

Application of BFS:

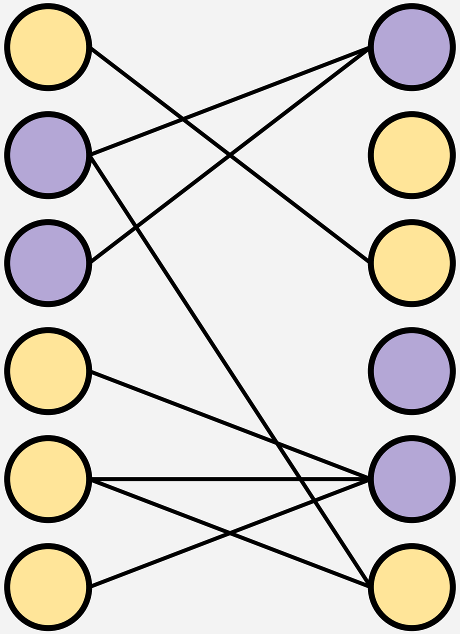
- **Color the levels of the BFS tree in alternating colors** (i.e. run BFS from any vertex, and alternate colors for each layer)
- **If you attempt to color the same vertex different colors** (i.e. revisit a node that's a different color than what you would have colored it), **then the graph isn't bipartite!**
- **If you successfully color the whole graph without conflicts, then it is bipartite!**



BIPARTITE GRAPHS

But wait… there exists many poor colorings on legitimate bipartite graphs.

Just because **the BFS coloring technique** doesn't work, why do we just throw up our hands and say **no** coloring works?

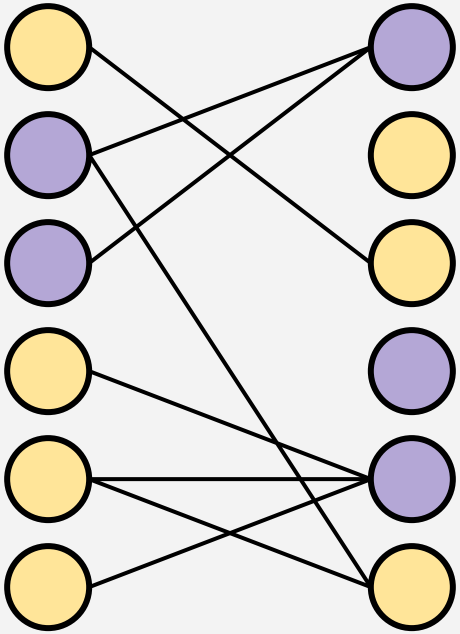


BIPARTITE GRAPHS

But wait… there exists many poor colorings on legitimate bipartite graphs.

Just because **the BFS coloring technique** doesn't work, why do we just throw up our hands and say **no** coloring works?

We can come up with plenty of bad coloring on this legitimate bipartite graph...

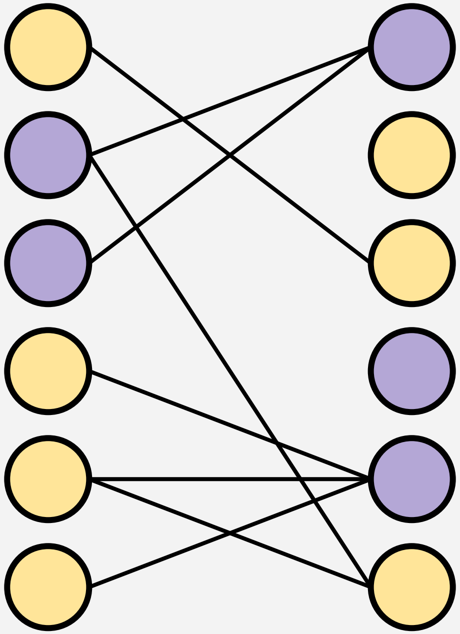


BIPARTITE GRAPHS

But wait… there exists many poor colorings on legitimate bipartite graphs.

Just because **the BFS coloring technique** doesn't work, why do we just throw up our hands and say **no** coloring works?

We can come up with plenty of bad coloring on this legitimate bipartite graph...

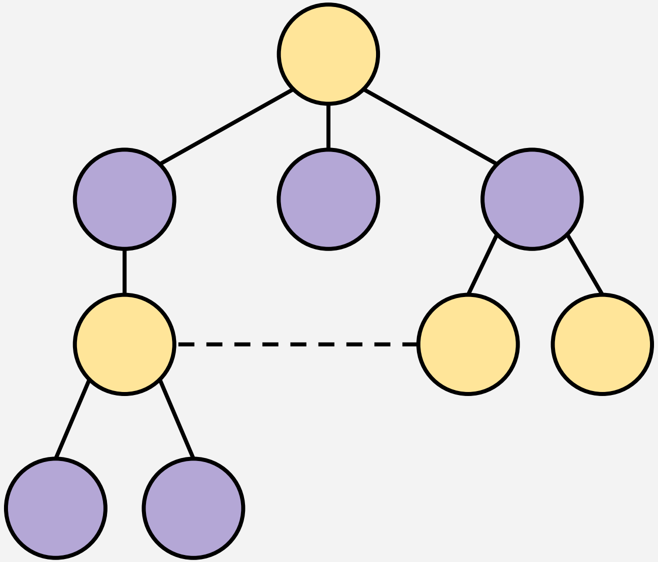


We need to prove that if BFS encounters a conflict (tries to color two neighbors the same color!), then there's no way the graph could be bipartite.

BIPARTITE GRAPHS

If BFS tries to color two neighbors the same color,
then it's found a **cycle of odd length** in the graph

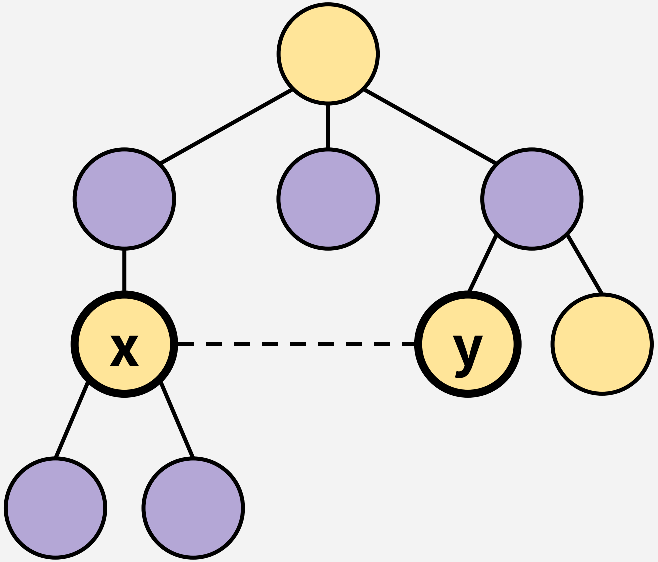
This is the BFS tree. Each level in this tree corresponds to each "BFS level". Our BFS coloring technique basically tries to alternate colors across levels.



BIPARTITE GRAPHS

If BFS tries to color two neighbors the same color, then it's found a **cycle of odd length** in the graph

This is the BFS tree. Each level in this tree corresponds to each "BFS level". Our BFS coloring technique basically tries to alternate colors across levels.

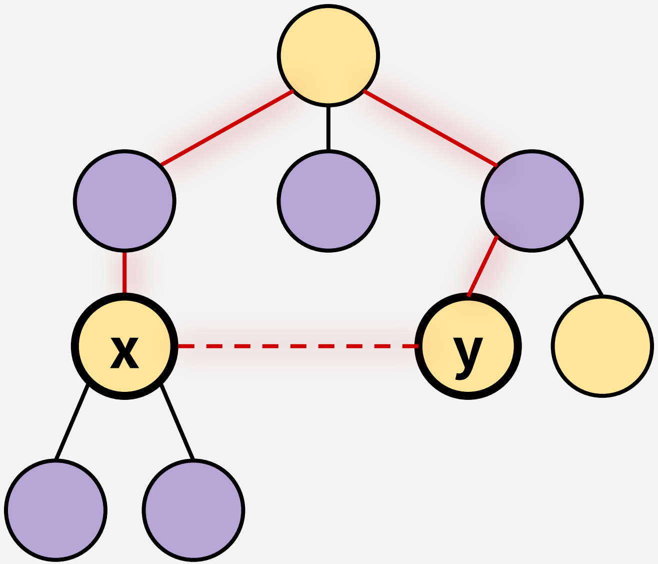


These neighbors are the conflict! BFS will try to color one of **x** or **y** purple, but it's already been colored yellow.

BIPARTITE GRAPHS

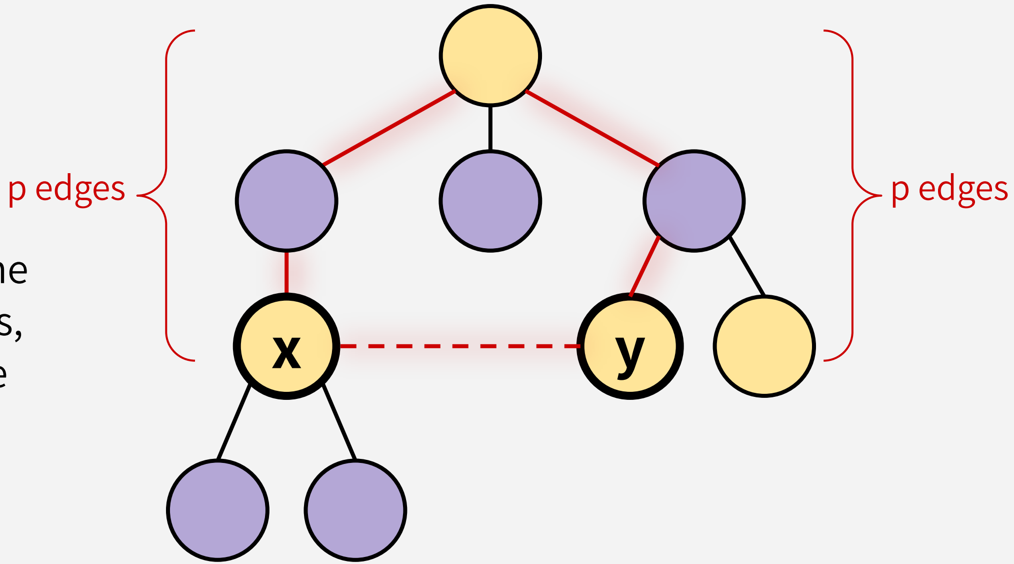
If BFS tries to color two neighbors the same color, then it's found a **cycle of odd length** in the graph

If **x** and **y** are the same color & are neighbors, then they are on the same level.



BIPARTITE GRAPHS

If BFS tries to color two neighbors the same color, then it's found a **cycle of odd length** in the graph

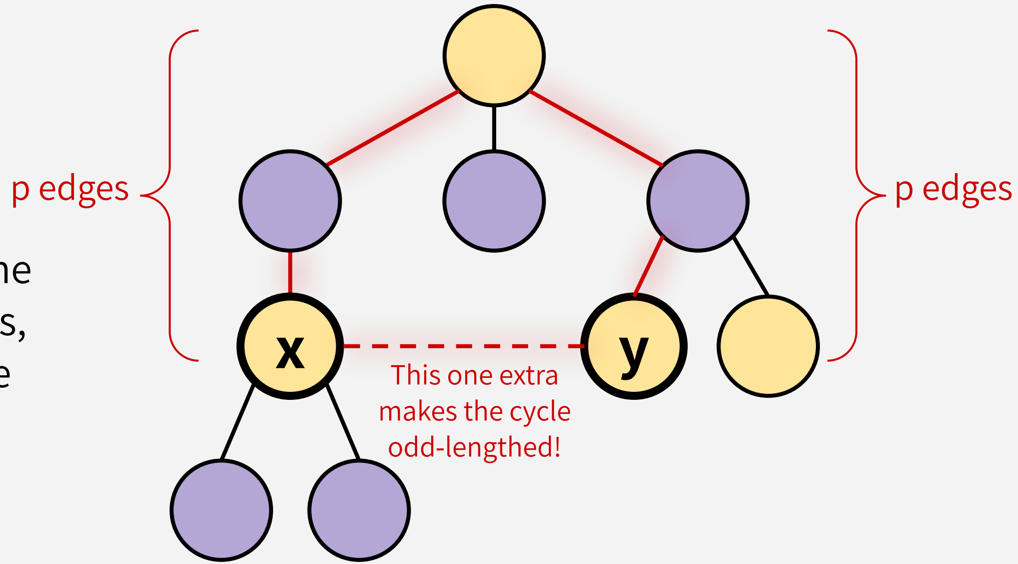


If **x** and **y** are the same color & are neighbors, then they are on the same level.

BIPARTITE GRAPHS

If BFS tries to color two neighbors the same color,
then it's found a **cycle of odd length** in the graph

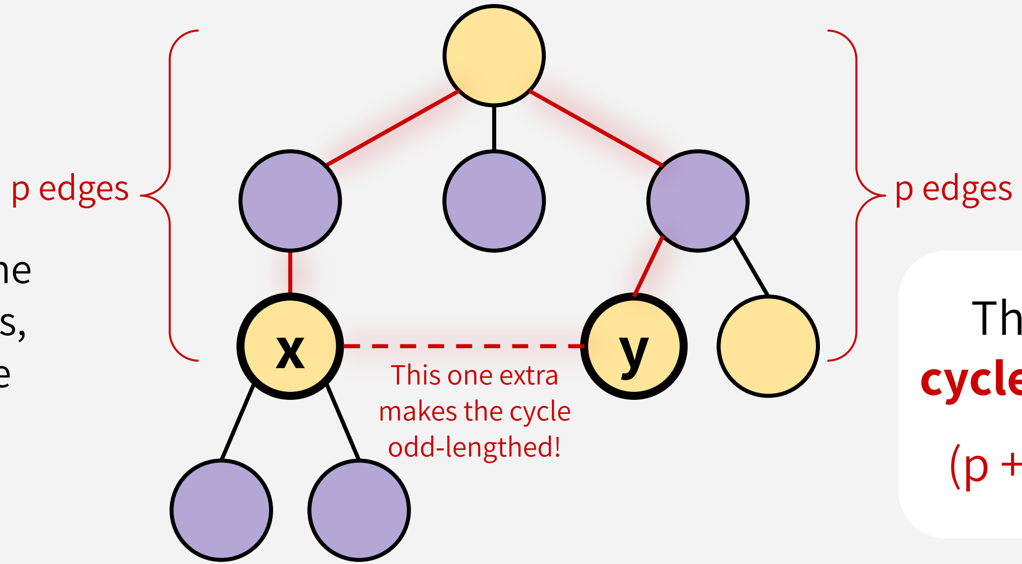
If **x** and **y** are the same color & are neighbors, then they are on the same level.



BIPARTITE GRAPHS

If BFS tries to color two neighbors the same color, then it's found a **cycle of odd length** in the graph

If **x** and **y** are the same color & are neighbors, then they are on the same level.



Thus, there is a **cycle of odd length**

(p + p + 1) = odd #

BIPARTITE GRAPHS

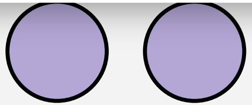
If BFS tries to color two neighbors the same color,

It's impossible to color a cycle of odd length with two colors such that no two neighbors have the same color. Therefore, it's impossible to two-color the graph such that no adjacent vertices are colored the same.

So, BFS colors two neighbors the same color iff the graph is not bipartite.

If **x** and
color &
then t
sa

s a
ngth
(p + p + 1) = odd #



BFS & BIPARTITE GRAPHS RECAP

BFS can be used to detect bipartite-ness of a graph in time O(n + m), since all that coloring business is just O(1) extra work per node or edge.

This is one example of how you can take advantage of the "layers" that BFS constructs to reason about how to accomplish a task that might not seem like a "classic" BFS-shortest-path task (which you might be more familiar with).

سوال؟