Graph Traversals: BFS & DFS

Story So Far

- Breadth-first search and breadth-first search tree
- Analysis: $O(n+m)$ time

```
BFS (G, s):
 Put s in the queue Q
 While Q is not empty
   Extract v from Q
     If v is unmarked
       Mark v
       For each edge (v, w):
          Put w into the queue Q
```

We can optimize this algorithm by checking whether the node w is marked before we place it the bag.

Story So Far

- Breadth-first search and breadth-first search tree
- Analysis: $O(n + m)$ time

```
BFS (G, s):
 Mark s and put in the queue Q
 While Q is not empty
   Extract v from Q
       For each edge (v, w):
          If w is unmarked
              Mark w
              Put w into the queue Q
```

BFS: Computing Levels

```
BFS (G, s):
 for all v ≠ s: set L[v] = ∞
 Set L[s] = 0
 Put s in the queue Q
 While Q is not empty
   Extract v from Q
       For each edge (v, w):
          if L[w] = ∞ then        // w is unmarked
             Put w into the queue Q
             L[w] = L[v]+1
```

BFS Levels

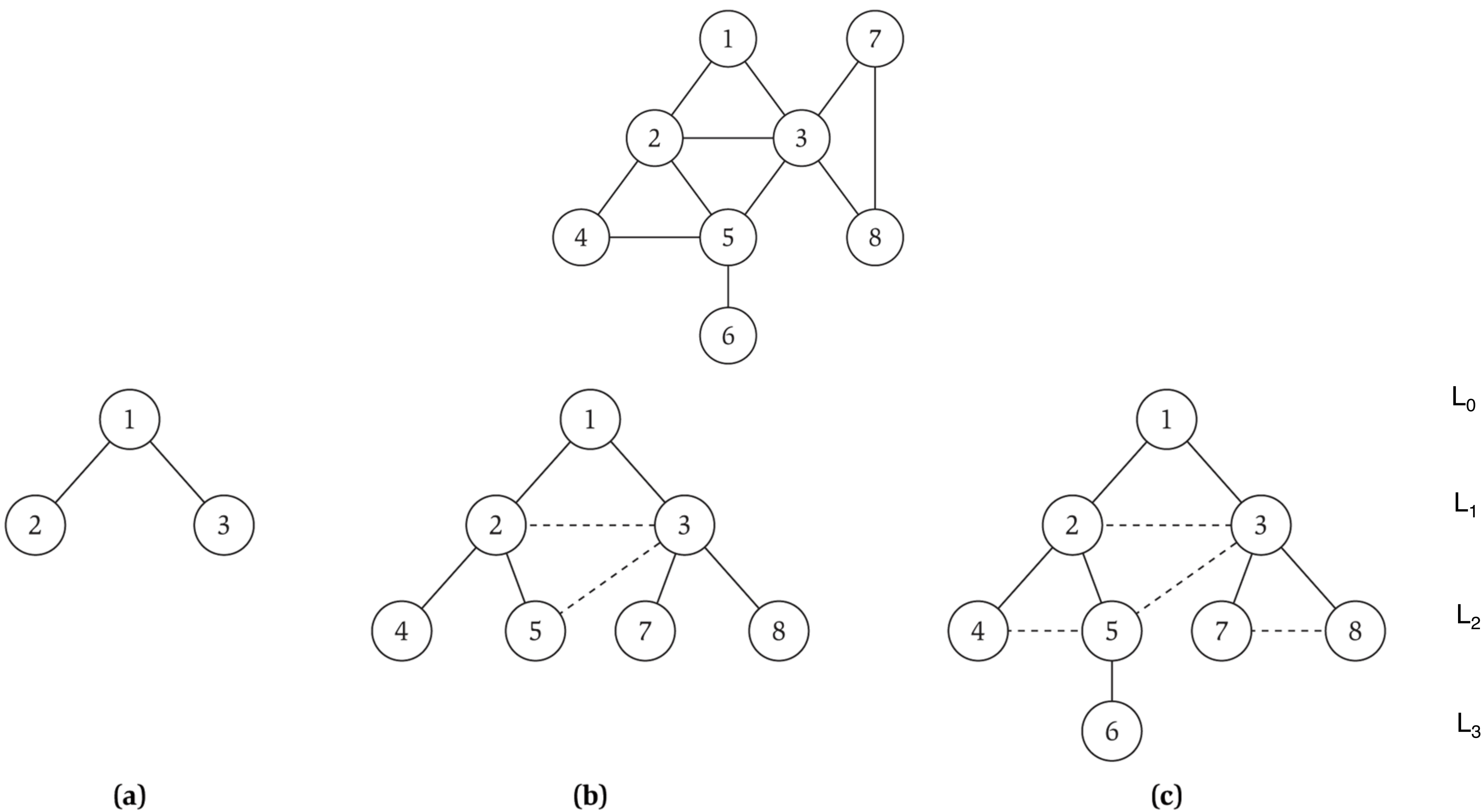
- **BFS Property.** All nodes in level L_i are explored (removed from the queue) before any node in level L_{i+1}.
- Can show this by observing the following invariant:
- Let the queue contain vertices $u_1, u_2, \ldots, u_k$ in order at any time, then
    - $L[u_1] \leq L[u_2] \leq \ldots L[u_k] \leq L[u_1] + 1$
    - Proof by induction

BFS Levels

- **Claim**. Let the BFS queue contain vertices $u_1, u_2, \ldots, u_k$ in order at any time, then $L[u_1] \leq L[u_2] \leq \ldots L[u_k] \leq L[u_1] + 1$
- **Proof**. Base case: only s in queue, condition holds trivially
- Suppose the condition holds until the current step
- Next step: u_1 is removed from queue, and its neighbors $v_1, v_1, \ldots, v_r$ are added to the queue with $L[v_1] = L[v_2] = \cdots = L[v_r] = L[u_1] + 1$
- By the induction hypothesis $L[u_2] \leq L[u_3] \leq \ldots L[u_k] \leq L[u_1] + 1$
- Thus, $L[u_2] \leq L[u_3] \leq \ldots L[u_k] \leq L[v_1] \leq \ldots \leq L[v_r] = L[u_1] + 1$
- Since $L[u_1] \leq L[u_2]$ from induction hypothesis, $L[u_2] \leq L[u_3] \leq \ldots L[u_k] \leq L[v_1] \leq \ldots \leq L[v_r] \leq L[u_2] + 1$ ∎

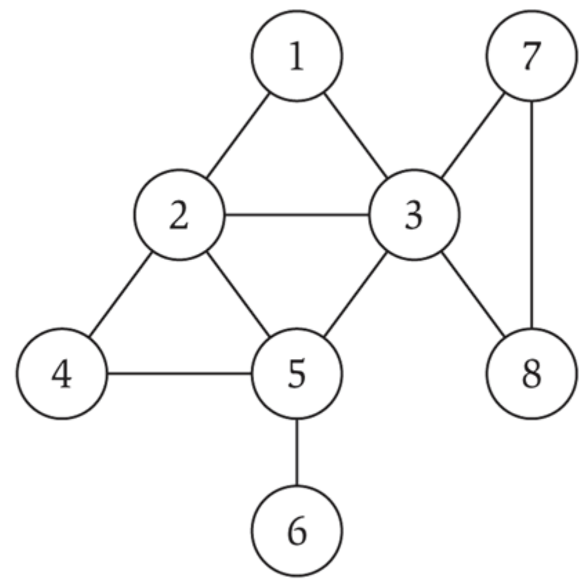
BFS Tree Structure

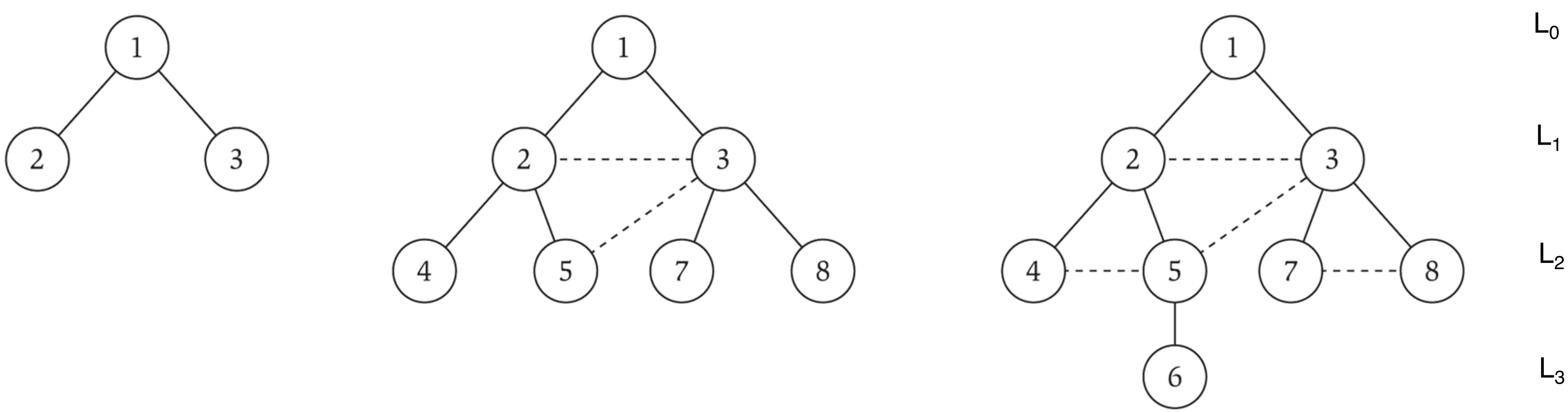
- Property. Let T be a BFS tree of $G = (V, E)$, and let (x, y) be an edge of G. Then, the levels of x and y differ by at most 1.



BFS Tree Structure

- Property. Let T be a BFS tree rooted at r of a connected unweighted graph, then the path from r to any node $u \in V$ in T is **the shortest path** from r to u (path with fewest edges)





(a) (b) (c)

Shortest Path Proof

- **Lemma**. Let L_i denote the set of nodes at level i in the BFS tree of G with root s, and let $d(s, u)$ denote the length of the shortest path from s to u in G, then $u \in L_i$ iff $d(s, u) = i$
- **Proof**. (Strong induction on i). Base case: $i = 0$, $L_0 = \{s\}$
- Suppose $u \in L_j$ iff $d(s, u) = j$ for all $j \leq i$
- $(\Rightarrow)$ Consider $y \in L_{i+1}$, then it was added through some edge (x, y) where $x \in L_i$, so there is a path from s to y through x
    - $d(s, y) \leq d(s, x) + 1$
    - By the induction hypothesis $d(s, x) = i$
    - Thus $d(s, y) \leq i + 1$

Shortest Path Proof

- **Proof**. (Strong induction on i). Base case: $i = 0$, $L_0 = \{s\}$
- Suppose $u \in L_j$ iff $d(s, u) = j$ for all $j \leq i$
- $(\Rightarrow)$ Consider $y \in L_{i+1}$, then it was added through some edge (x, y) where $x \in L_i$, so there is a path from s to y through x
    - $d(s, y) \leq d(s, x) + 1$
    - By the induction hypothesis $d(s, x) = i$
    - Thus $d(x, y) \leq i + 1$
- Since $y \notin L_j$ for any $j \leq i$, by the induction hypothesis $d(s, y) > i$
- Thus, $d(s, y) = i + 1$

Shortest Path Proof

- **Proof**. (Strong induction on i). Base case: $i = 0$, $L_0 = \{s\}$
- Suppose $u \in L_j$ iff $d(s, u) = j$ for all $j \leq i$
- $(\Leftarrow)$ Suppose $d(s, y) = i + 1$, then by the inductive hypothesis, $y \notin L_j$ for any $j \leq i$
- Let (x, y) be the last edge on the shortest path P from s to y
- Then, P must also contain the shortest from s to x: $d(s, x) = i$
- By the induction hypothesis, $x \in L_i$
- We claim that $y \in L_{i+1}$
  - Since y is not in an earlier level, and (x, y) exists, it is either added to L_{i+1} when (x, y) is scanned (or earlier when another edge out of L_i is being scanned)

Applications of BFS

Spanning Trees & Components

- **Definition.** A spanning tree of an undirected graph G is a connected acyclic subgraph of G that contains every node of G.
- The tree produced by the BFS algorithm (with $(u, \text{parent}(u))$ as edges) is a spanning tree of the component containing s.
- **Connected component of s**: all nodes reachable from s
- In an undirected graph, a BFS spanning tree gives the shortest path from s to every other vertex in its component

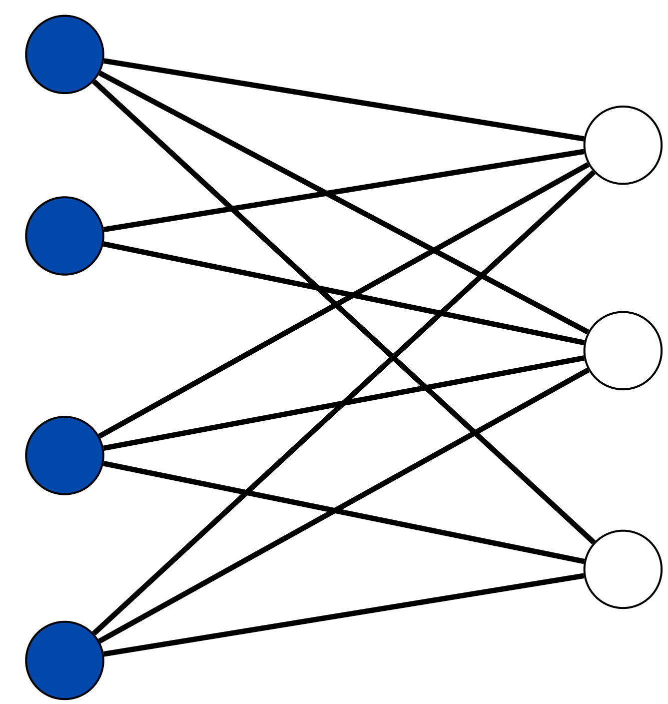
BFS Application: Connectivity

- How to whether a graph is connected using traversals?
    - If the BFS spanning tree contains all nodes of the graph, then the graph is connected
- Suppose the graph is not connected
- How can we find all connected components?
    - Start BFS with any node s, when its done, all nodes in the BFS tree of s are one component
    - Pick another node that is not visited and repeat
    - Number of trees in resulting **forest** is the number of components of the graph

BFS Application: Bipartite Testing

- **Bipartite graph.**
  - An undirected graph is **bipartite** if its nodes can be portioned into two sets S_1, S_2 such that all edges have endpoint in both sets

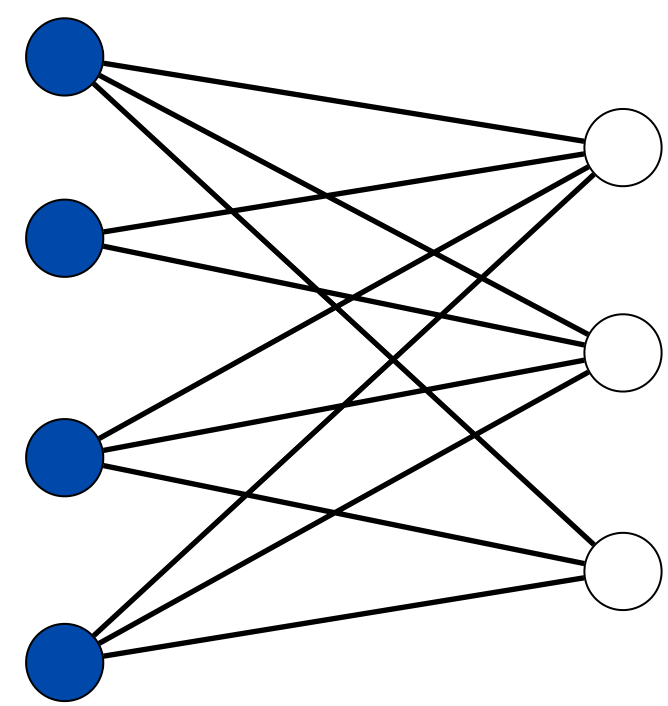
- Models many settings
  - We already encountered an application, which is…?
  - Common in scheduling, one set is machine, other set is jobs



a bipartite graph

BFS Application: Bipartite Testing

- Given a graph $G = (V, E)$ verify if it is bipartite
- Hint: need to use traversals
- But first need to understand structure of bipartite graphs
- **Question:** Can a bipartite graph contain an odd-length cycle?
- How do we prove this?
- In fact, a graph is bipartite if and only if it does not have an odd length cycle
- One direction bipartite implies no odd length cycle is simple
- Will prove the other direction constructively



a bipartite graph

Bipartite Testing: Using BFS

Theorem. The following statements are **equivalent** for a connected graph G :

(a) G is bipartite

(b) G has no odd-length cycle

(c) No BFS tree has edges between vertices at same level

(d) Some BFS tree has no edges between 2 vertices at same level

Note: Conditions (a) and (b) seem hard to check directly; but conditions (c) and (d) allow an easy check!

Bipartite Testing: Using BFS

Theorem. The following statements are equivalent for a connected graph G :

(a) G is bipartite

(b) G has no odd-length cycle

(c) No BFS tree has edges between vertices at same level

(d) Some BFS tree has no edges between 2 vertices at same level

Proof. (a) ⇒ (b)

Vertices must alternate between V_1 and V_2.

Bipartite Testing: Using BFS

Theorem. The following statements are equivalent for a connected graph G :

(a) G is bipartite

(b) G has no odd-length cycle

(c) No BFS tree has edges between vertices at same level

(d) Some BFS tree has no edges between 2 vertices at same level

Proof. (b) $\Rightarrow$ (c)

Contradiction: Such an edge implies an odd cycle

Bipartite Testing: Using BFS

Theorem. The following statements are equivalent for a connected graph G :

(a) G is bipartite

(b) G has no odd-length cycle

(c) No BFS tree has edges between vertices at same level

(d) Some BFS tree has no edges between 2 vertices at same level

Proof. (c) $\Rightarrow$ (d)

If all BFS trees have a property then some do as well

Bipartite Testing: Using BFS

Theorem. The following statements are equivalent for a connected graph G :

(a) G is bipartite

(b) G has no odd-length cycle

(c) No BFS tree has edges between vertices at same level

(d) Some BFS tree has no edges between 2 vertices at same level

Proof. (d) ⇒ (a)

Edges must span consecutive levels: levels provide bipartition of G

Implications of the Theorem

How to check if a graph is bipartite?

- When we visit an edge during BFS, we know the level of both of its endpoints
- So if both ends have the same level, then we can stop ! (G is not bipartite)
- If no such edge is found during traversal, G is bipartite
- Alternate levels give the bipartition

Running time?

- Still $O(n+m)$
- **Certificate.** If G is not bipartite this algorithm gives us a proof of it (the odd cycle that is found)!