# CS 401: Computer Algorithm I

## BFS

Xiaorui Sun

# Last Lecture: BFS algorithm

Initialization: mark all vertices "undiscovered"

BFS($s$)

mark $s$ discovered

queue = { $s$ }

while queue not empty

$u$ = remove_first(queue)

for each edge $\{u, x\}$

if ($x$ is undiscovered)

mark $x$ discovered

append $x$ on queue

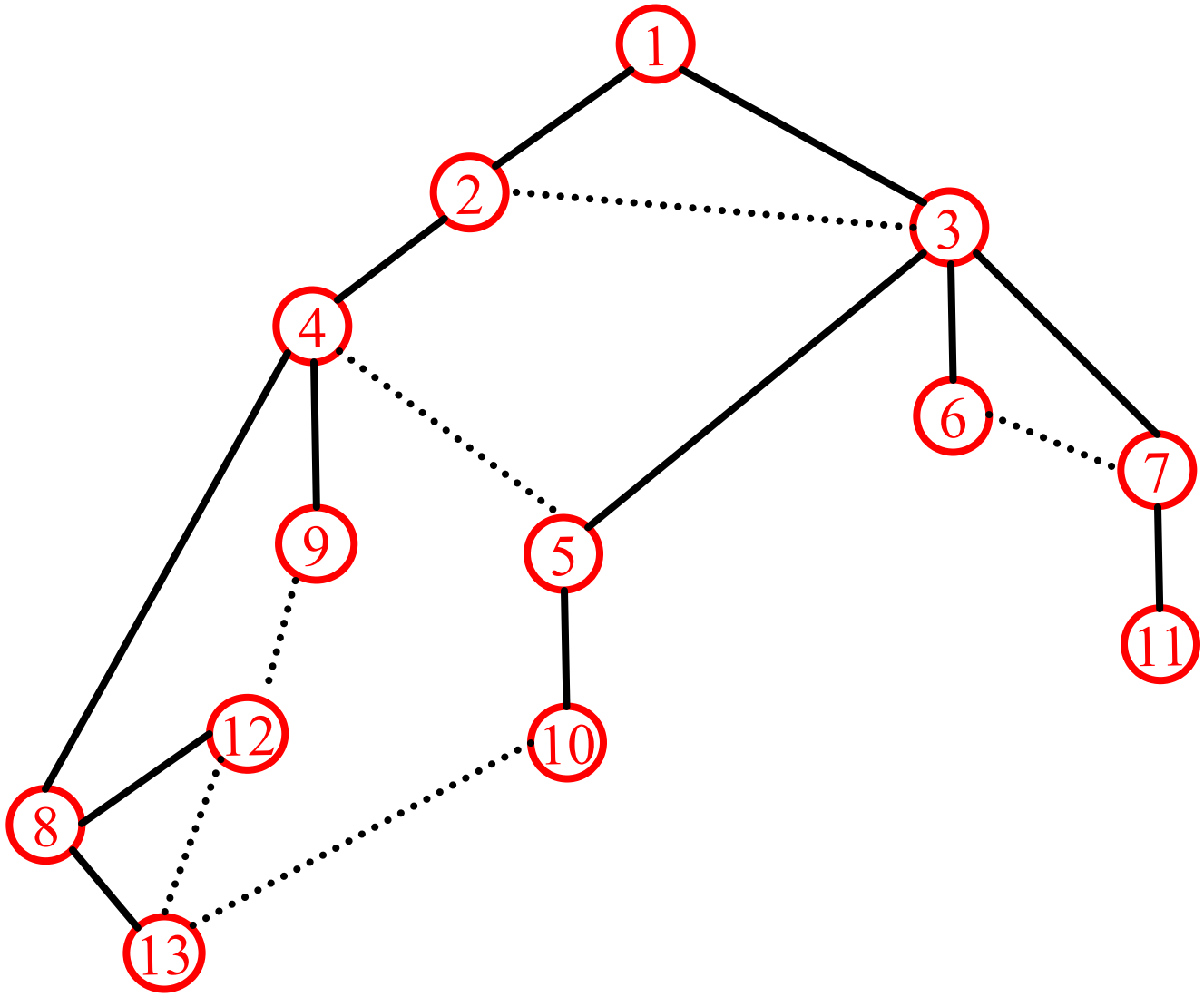
mark $u$ fully-explored

Running time
Adjacency matrix: $O(n^2)$
Adjacency list: $O(n+m)$

# Properties of BFS

- BFS($s$) visits a vertex $v$ if and only if there is a path from $s$ to $v$
- Edges into then-undiscovered vertices define a tree – the "Breadth First spanning tree" of $G$

# BFS Tree

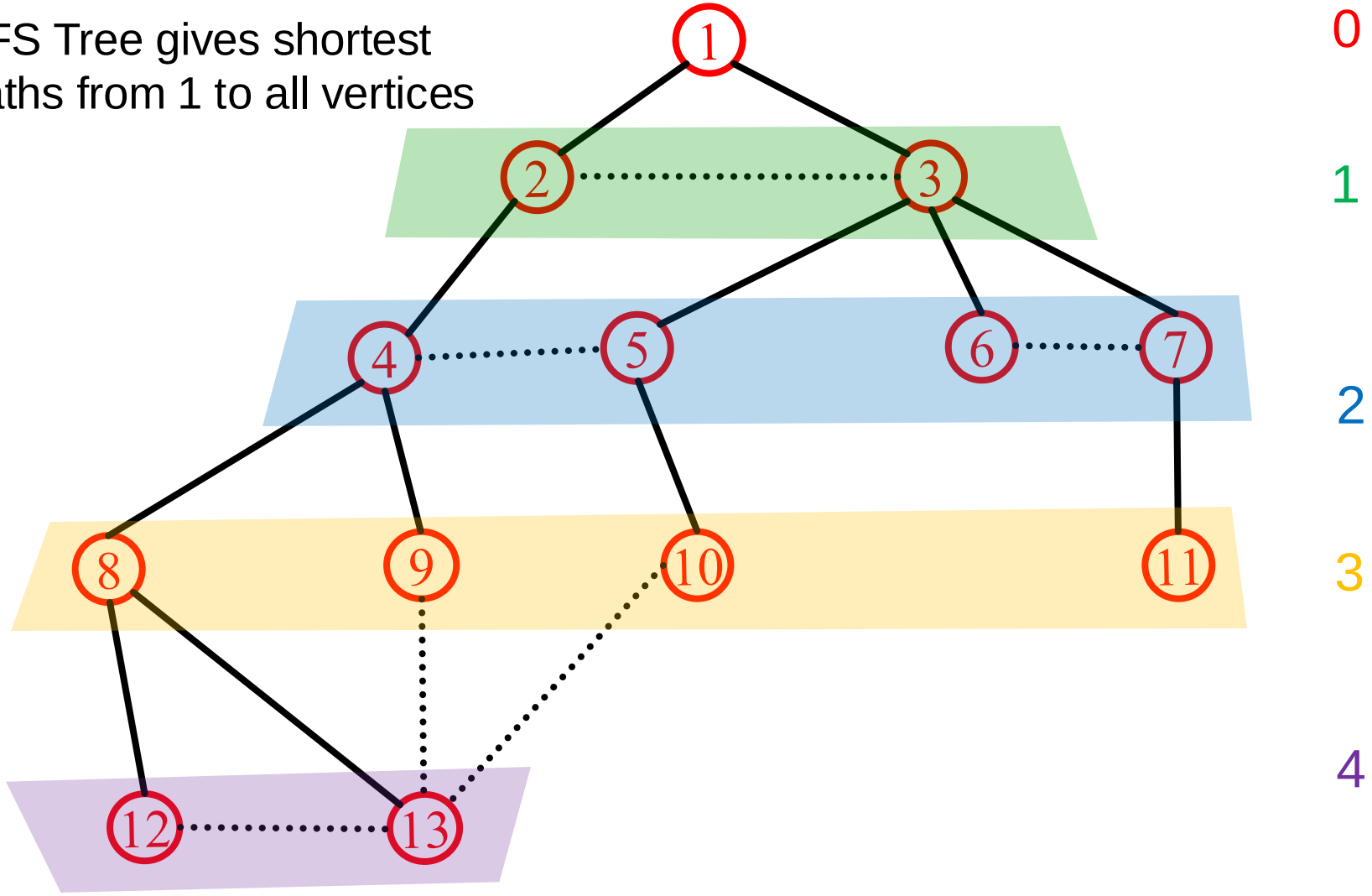


# Properties of BFS

- BFS($s$) visits a vertex $v$ if and only if there is a path from $s$ to $v$
- Edges into then-undiscovered vertices define a tree – the "Breadth First spanning tree" of $G$
- Level $i$ in the tree are exactly all vertices $v$ s.t., the shortest path (in $G$) from the root $s$ to $v$ is of length $i$

# Properties of BFS

BFS Tree gives shortest paths from 1 to all vertices



# Properties of BFS

- BFS($s$) visits a vertex $v$ if and only if there is a path from $s$ to $v$
- Edges into then-undiscovered vertices define a tree – the "Breadth First spanning tree" of $G$
- Level $i$ in the tree are exactly all vertices $v$ s.t., the shortest path (in $G$) from the root $s$ to $v$ is of length $i$
- All nontree edges join vertices on the same or adjacent levels of the tree

# Why Trees?

Trees are simpler than graphs

Many statements can be proved on trees by induction

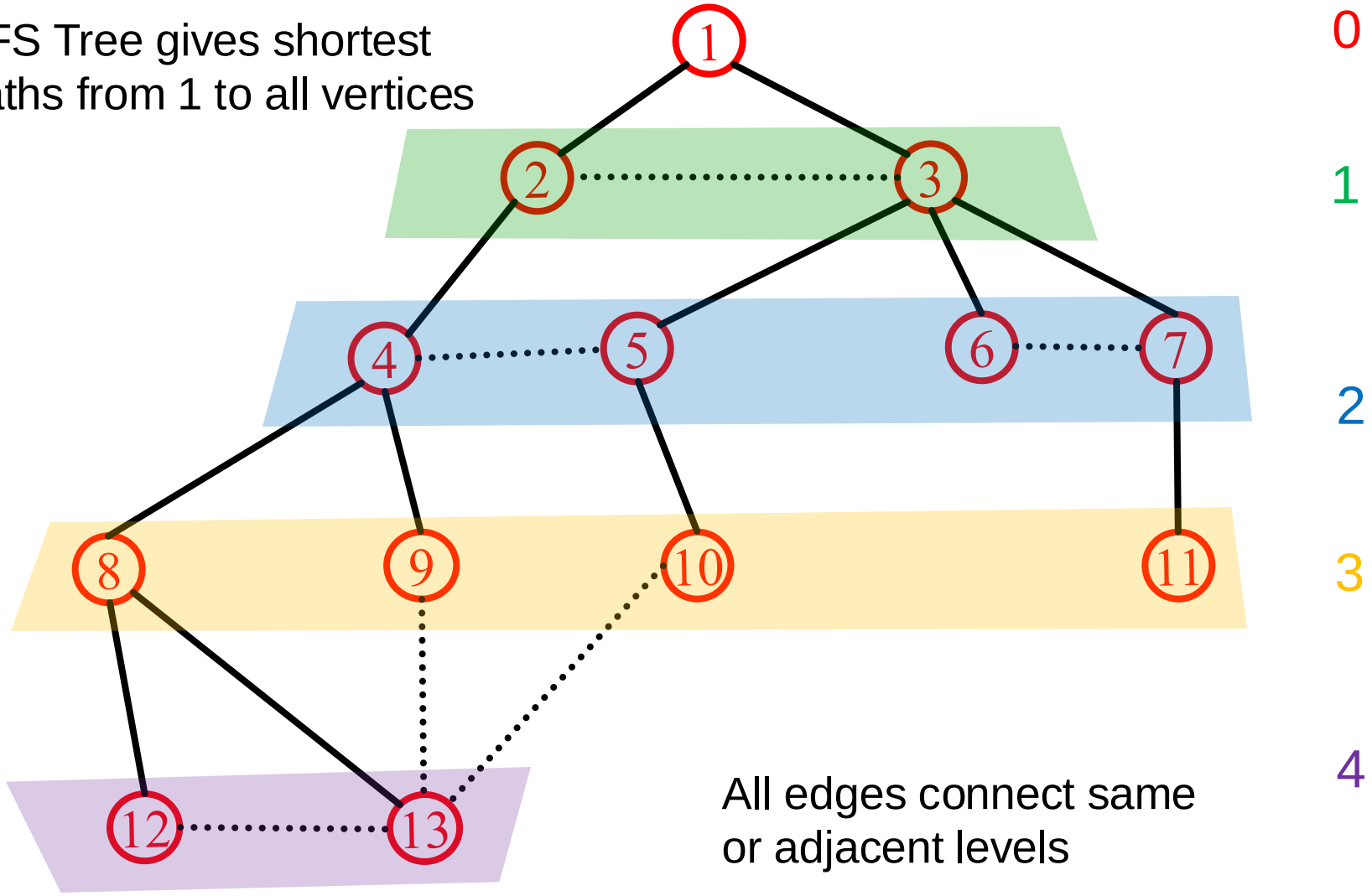
So, computational problems on trees are simpler than general graphs

This is often a good way to approach a graph problem:

- Find a "nice" tree in the graph, i.e., one such that non-tree edges have some simplifying structure
- Solve the problem on the tree
- Use the solution on the tree to find a “good” solution on the graph

# BFS Application: Shortest Paths

BFS Tree gives shortest paths from 1 to all vertices




All edges connect same or adjacent levels

# BFS Application: Connected Component

We want to answer the following type questions (fast):
Given vertices $u, v$ is there a path from $u$ to $v$ in $G$?

Idea: Create an array $A$ such that
For all $u$ in the same connected component, $A[u]$ is same.

Therefore, question reduces to

If A[u] = A[v]?

# BFS Application: Connected Component

Initial State: All vertices undiscovered, $c = 0$

For $v$ = 1 to $n$ do
    If state($v$) != fully-explored then
        Run BFS($v$)
        Set $A[u] = c$ for each $u$ found in BFS($v$)
        $c = c + 1$

Note: We no longer initialize to undiscovered in the BFS subroutine

Total Cost: $O(m + n)$

In every connected component with $n_i$ vertices and $m_i$ edges BFS takes time $O(m_i + n_i)$.

Note: one can use DFS instead of BFS.

# Connected Components

Lesson: We can execute any algorithm on disconnected graphs by running it on each connected component.

We can use the previous algorithm to detect connected components.

There is no overhead, because the algorithm runs in time $O(m+n)$.

So, from now on, we can (almost) always assume the input graph is connected.

# Bipartite Graphs

Definition: An undirected graph $G = (V, E)$ is bipartite

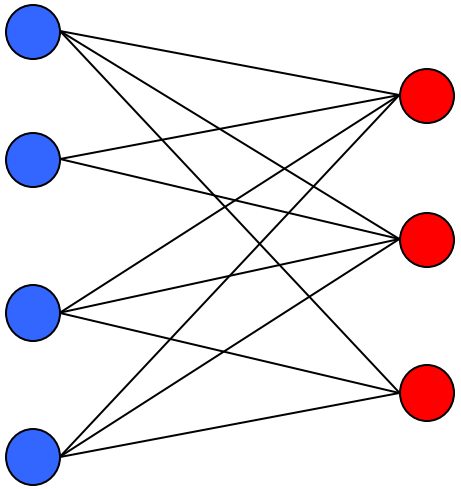
if you can partition the node set into 2 parts (say, blue/red or left/right) so that

all edges join nodes in different parts

i.e., no edge has both ends in the same part.

Application:

- Scheduling: machine=red, jobs=blue
- Stable Matching: men=blue, wom=red



*a bipartite graph*

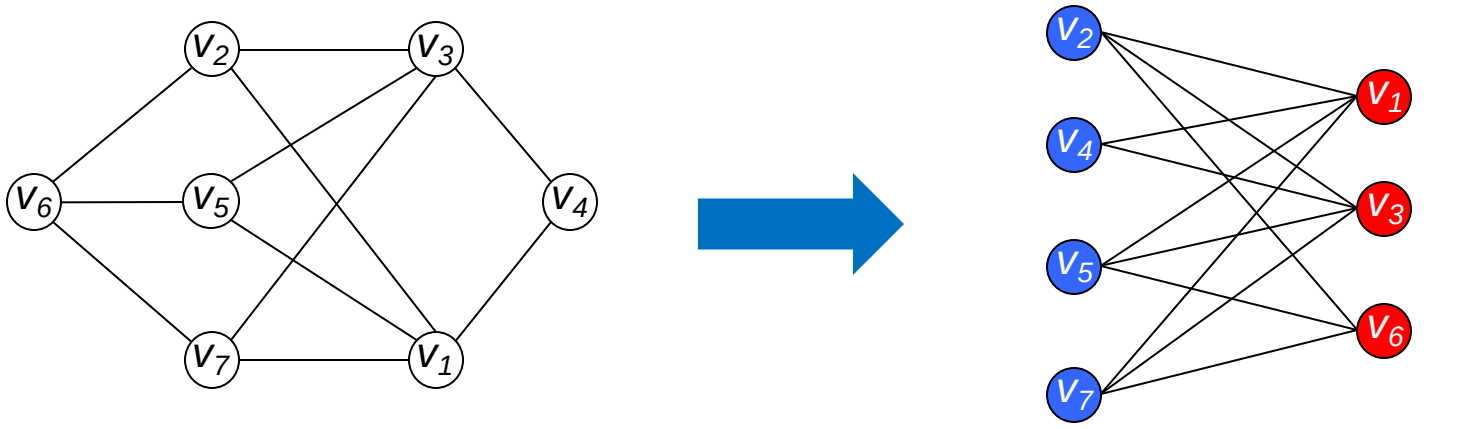
# Testing Bipartiteness

Problem: Given a graph $G$, is it bipartite?

Many graph problems become:

- Easier/Tractable if the underlying graph is bipartite (matching)

Before attempting to design an algorithm, we need to understand structure of bipartite graphs.




*a bipartite graph G*

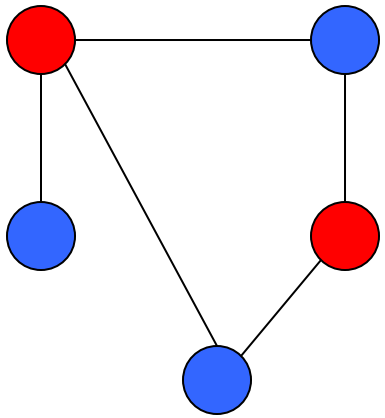
*another drawing of G*

# An Obstruction to Bipartiteness

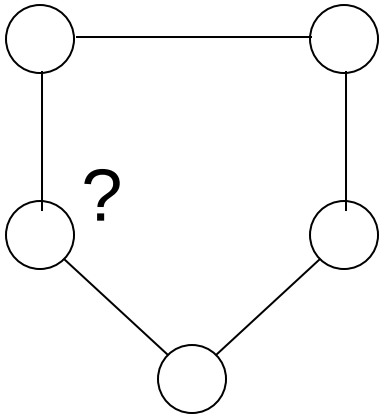
Lemma: If a graph contains an odd length cycle, then the graph is not bipartite.

(If $G$ is bipartite, then it does not contain an odd length cycle.)

Proof: We cannot 2-color an odd cycle, let alone $G$.



*bipartite*
*(2-colorable)*



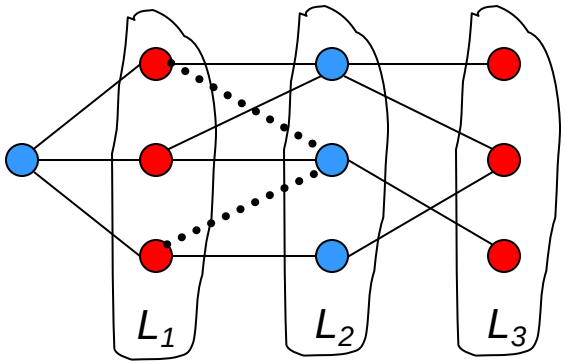

*not bipartite*
*(not 2-colorable)*

# A Characterization of Bipartite Graphs

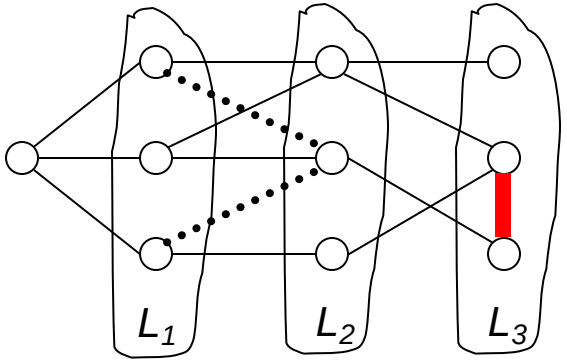
Lemma: Let $G$ be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS($s$). Exactly one of the following holds.

(i) No edge of $G$ joins two nodes of the same layer, and $G$ is bipartite.

(ii) An edge of $G$ joins two nodes of the same layer, and $G$ contains an odd-length cycle (and hence is not bipartite).




*Case (i)*




*Case (ii)*

# A Characterization of Bipartite Graphs

Lemma: Let $G$ be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS($s$). Exactly one of the following holds.

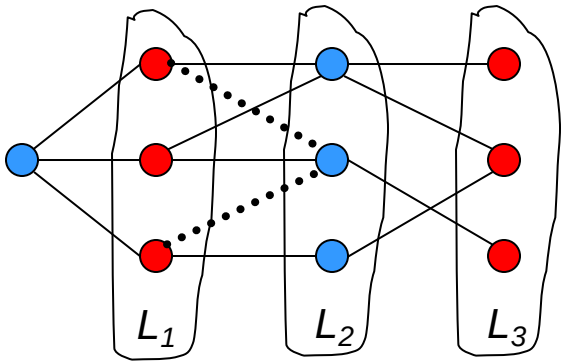
(i) No edge of $G$ joins two nodes of the same layer, and $G$ is bipartite.

(ii) An edge of $G$ joins two nodes of the same layer, and $G$ contains an odd-length cycle (and hence is not bipartite).

Proof. (i)

Suppose no edge joins two nodes in the same layer.

All edges join nodes on adjacent levels.




*Case (i)*

Bipartition:

blue = nodes on odd levels,

red = nodes on even levels.

# A Characterization of Bipartite Graphs

Lemma: Let $G$ be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS($s$). Exactly one of the following holds.

(i) No edge of $G$ joins two nodes of the same layer, and $G$ is bipartite.

(ii) An edge of $G$ joins two nodes of the same layer, and $G$ contains an odd-length cycle (and hence is not bipartite).

Proof. (ii)

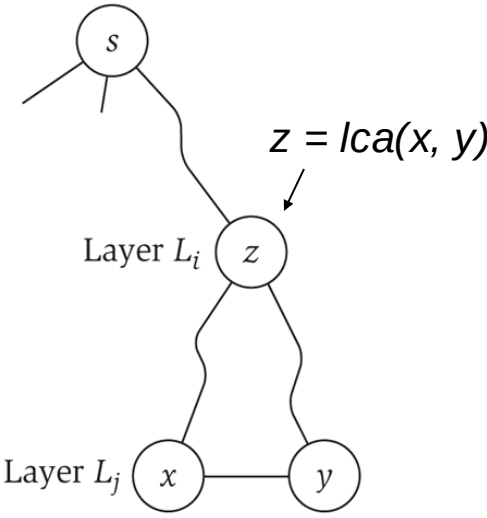
Suppose $\{x, y\}$ is an edge & $x, y$ in same level $L_j$.

Let $z =$ their lowest common ancestor in BFS tree.

Let $L_i$ be level containing $z$.

Consider cycle that takes edge from $x$ to $y$, then tree from $y$ to $z$, then tree from $z$ to $x$.

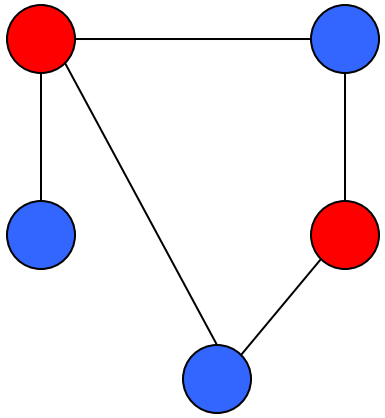
Its length is $1 + (j - i) + (j - i)$, which is odd.



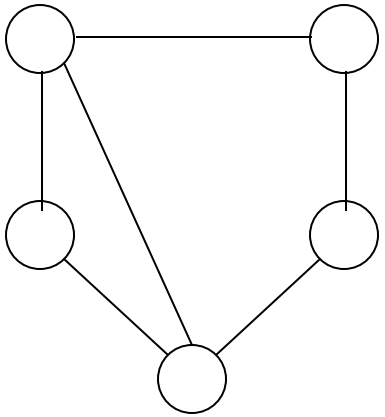
# Obstruction to Bipartiteness

Corollary: A graph $G$ is bipartite if and only if it contains no odd length cycles.

Furthermore, one can test bipartiteness using BFS.



*bipartite*
*(2-colorable)*



*not bipartite*
*(not 2-colorable)*

# Obstruction to Bipartiteness

Corollary: A graph $G$ is bipartite if and only if it contains no odd length cycles.

Furthermore, one can test bipartiteness using BFS.

Bipartiteness testing algorithm:

- Run BFS with an arbitrary start vertex
- Construct the BFS tree with the start vertex as root
- If each non-tree edge of G connects two vertices at different levels in the rooted BFS tree, then output yes.
- Otherwise, output no.

# BFS Summary

Breadth First Search (BFS): Explore vertices according to the order of the discovery of vertices

Property:

- BFS tree
- Level = distance (length of shortest path) from the initial vertex
- Every edge connect two vertices at the same or adjacent levels

Applications of BFS:

- Finding connected components of a graph
- Finding shortest path for unit-length graphs
- Testing bipartiteness