# Introduction to Algorithms

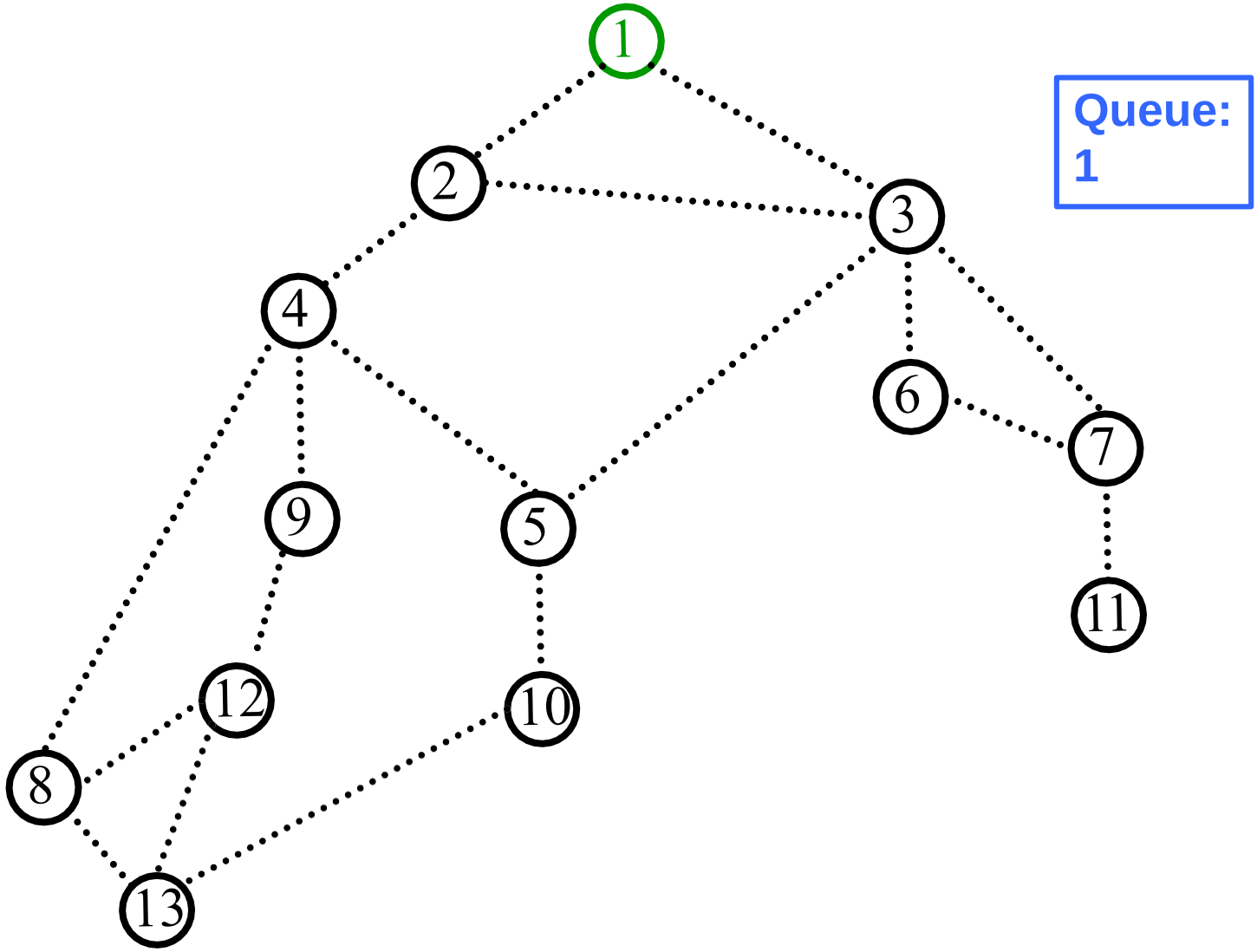
## Induction - Graphs

# BFS implementation

Global initialization: mark all vertices "undiscovered"

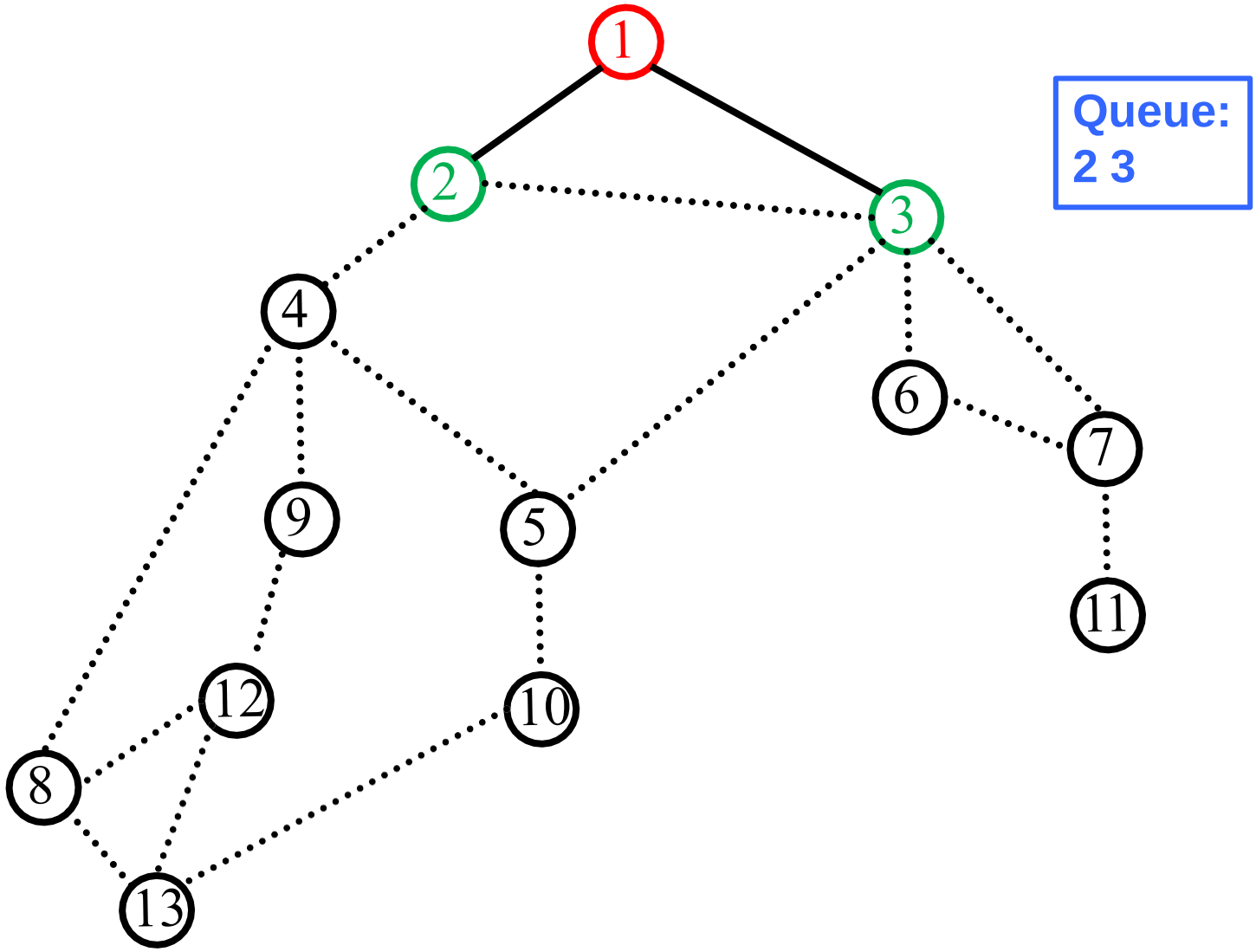
```
BFS(s)
    mark  s "discovered"
    queue = { s }
    while queue not empty
        u = remove_first(queue)
        for each edge {u,x}
            if (x is undiscovered)
                mark x discovered
                append x on queue
        mark u fully-explored
```

# BFS(1)



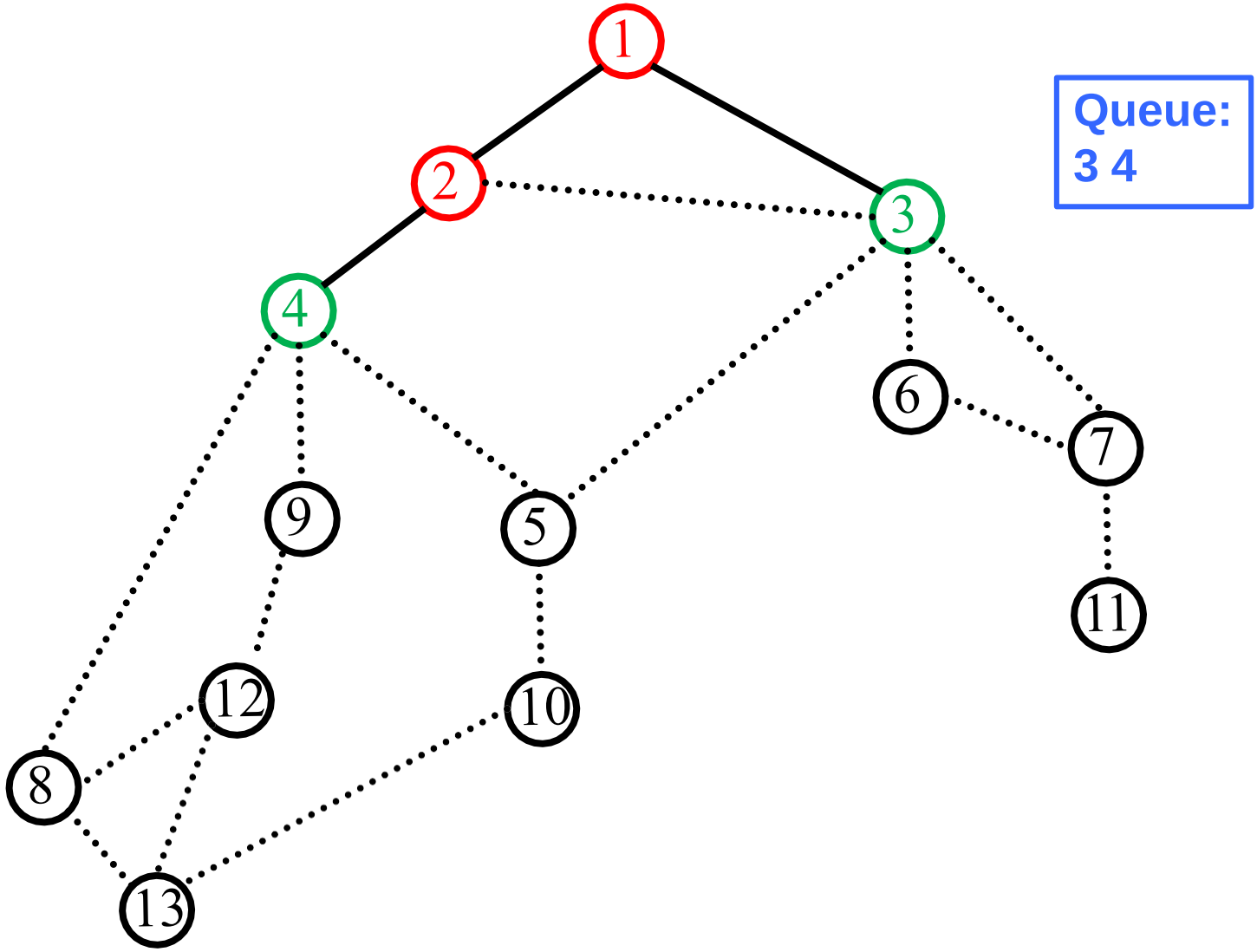

**Queue:**
**1**

# BFS(1)



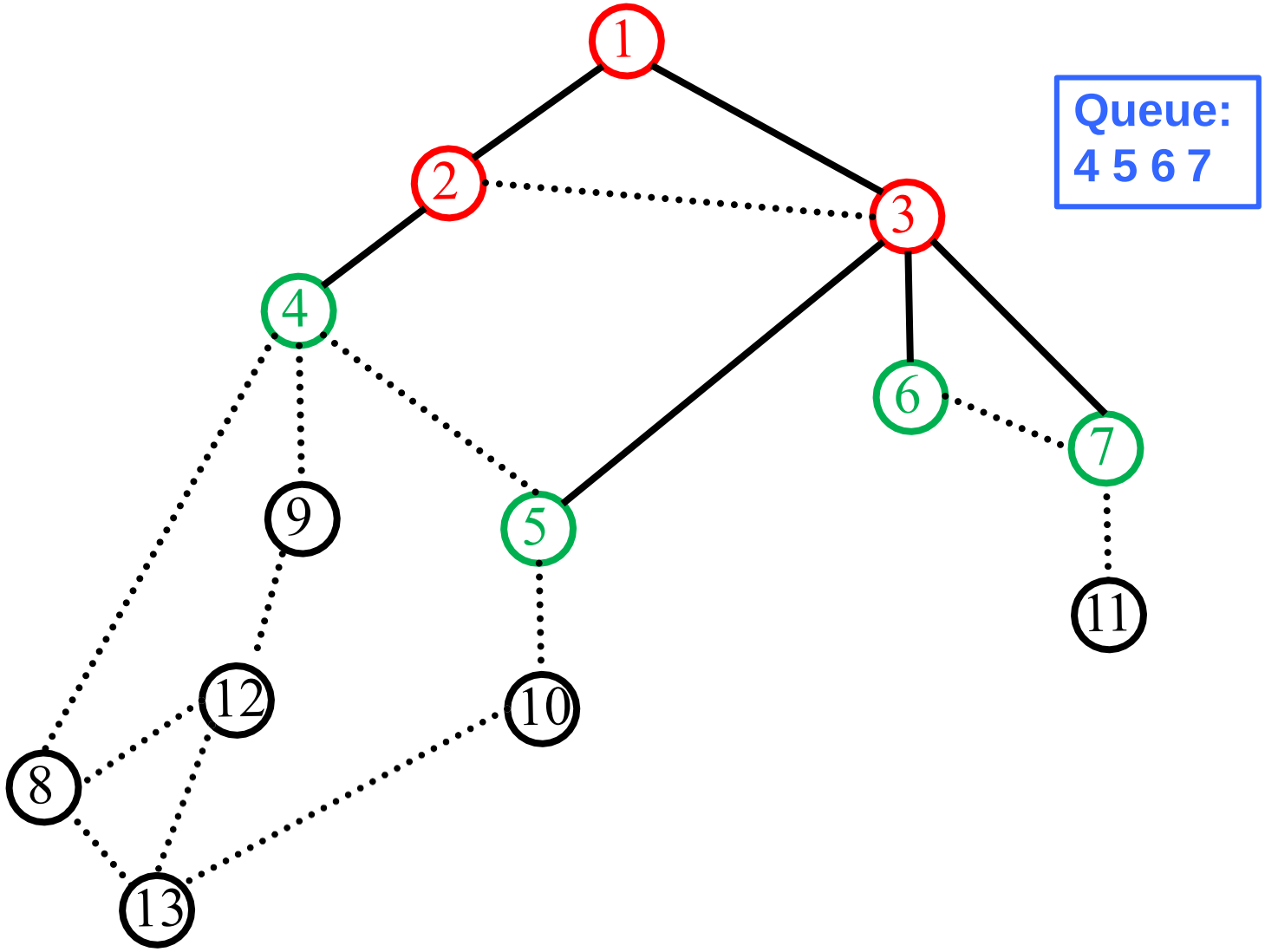

**Queue:**
**2 3**

# BFS(1)

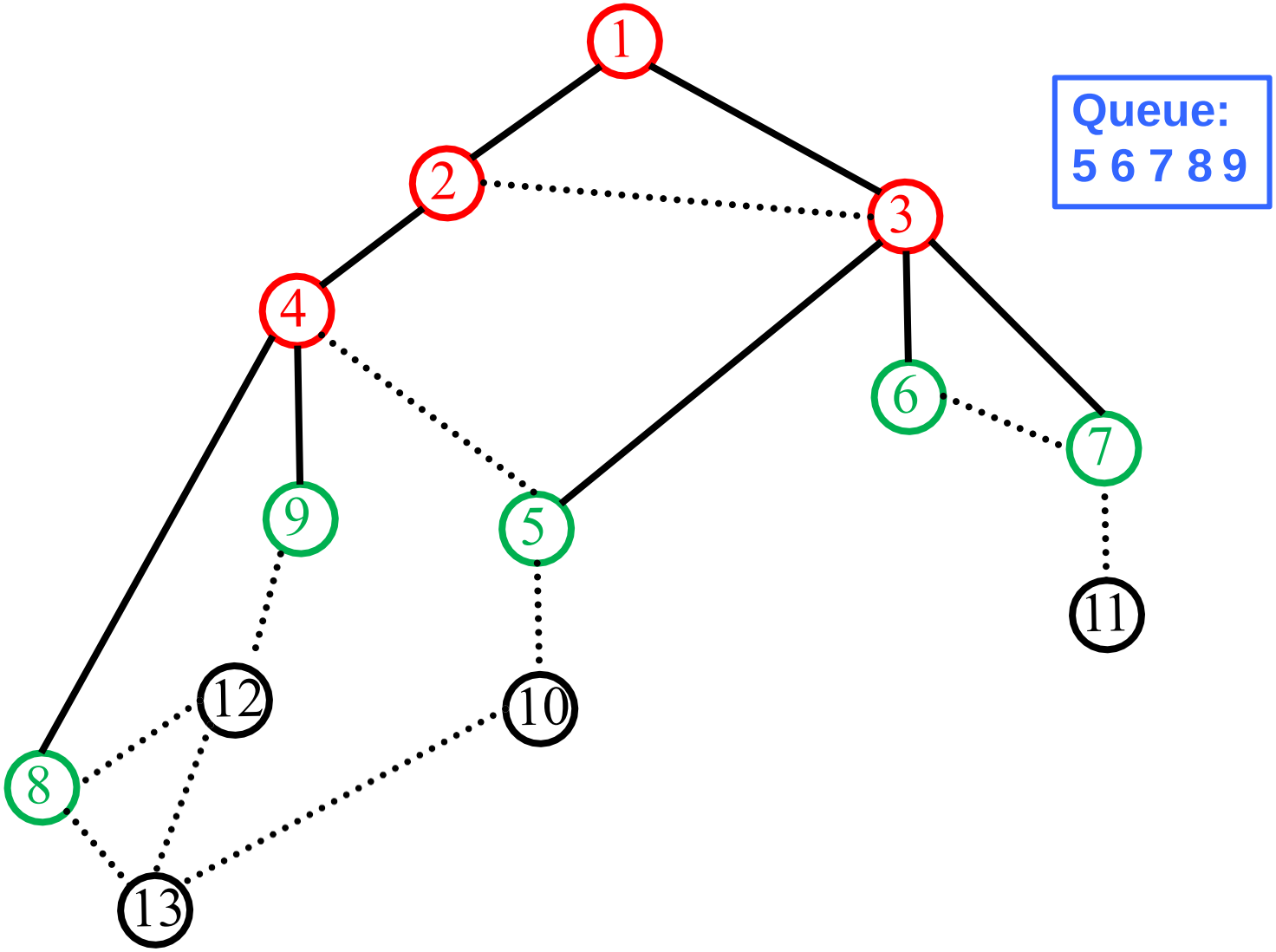



**Queue:**
**3 4**

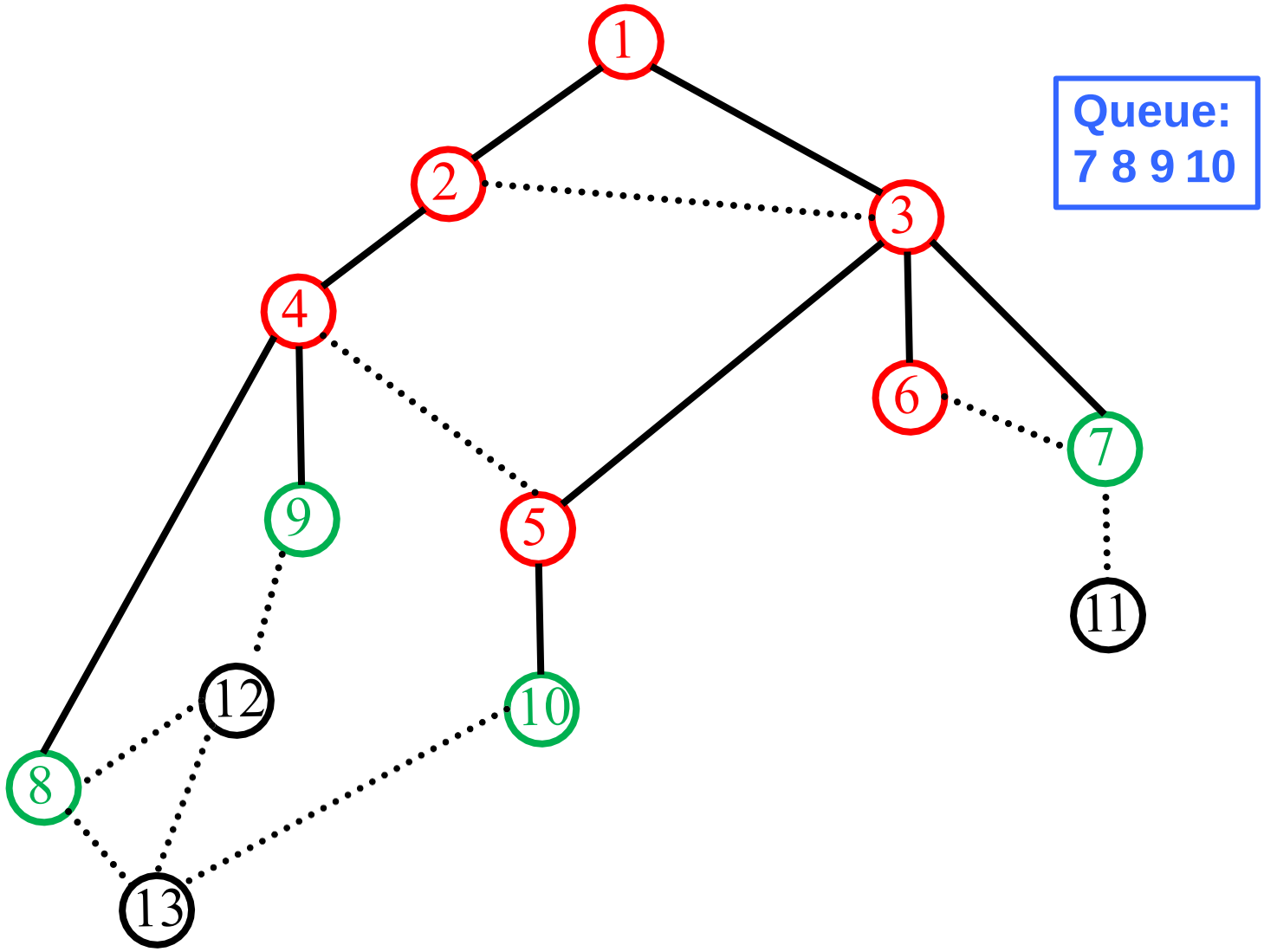
# BFS(1)



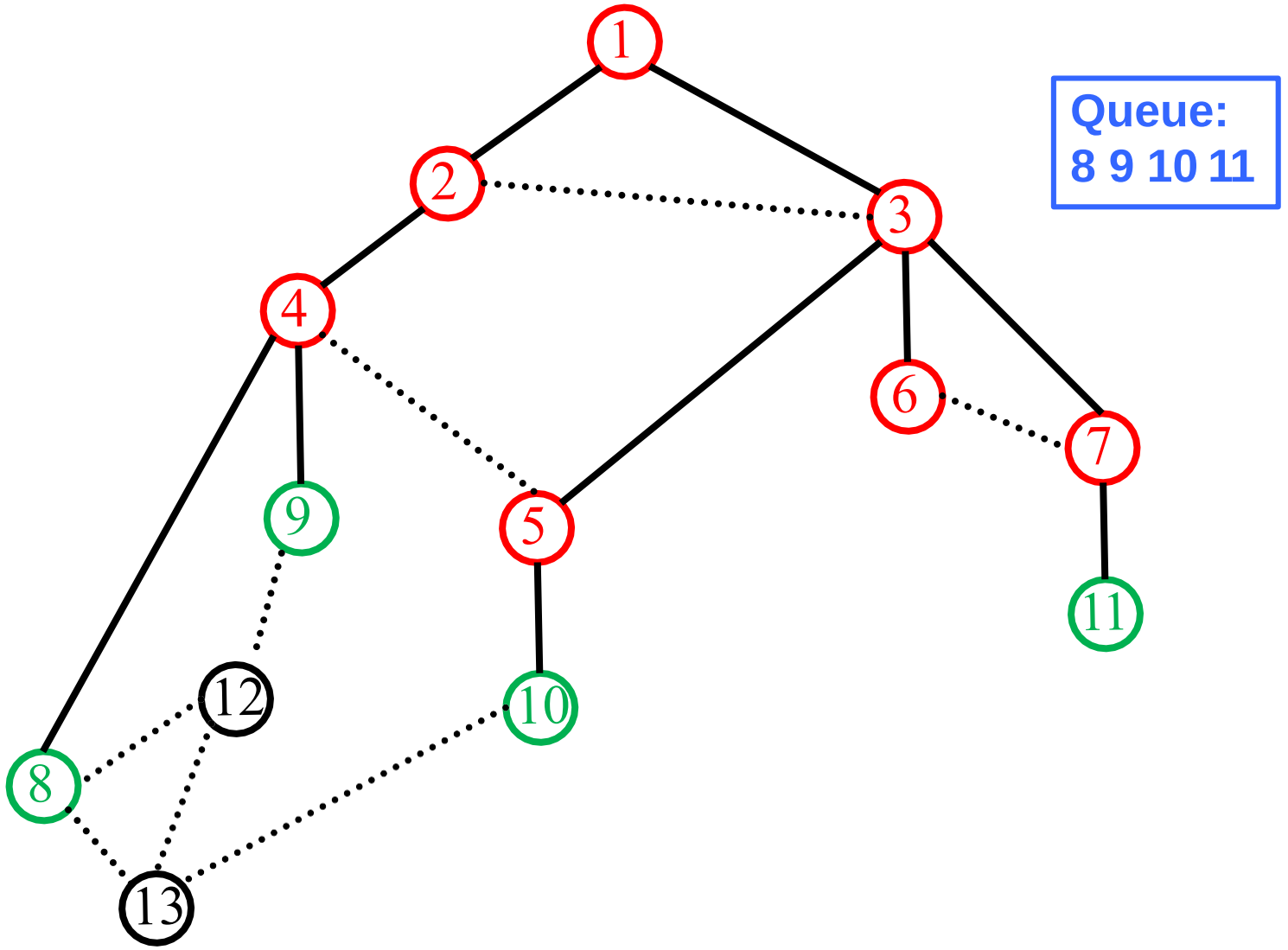
# BFS(1)



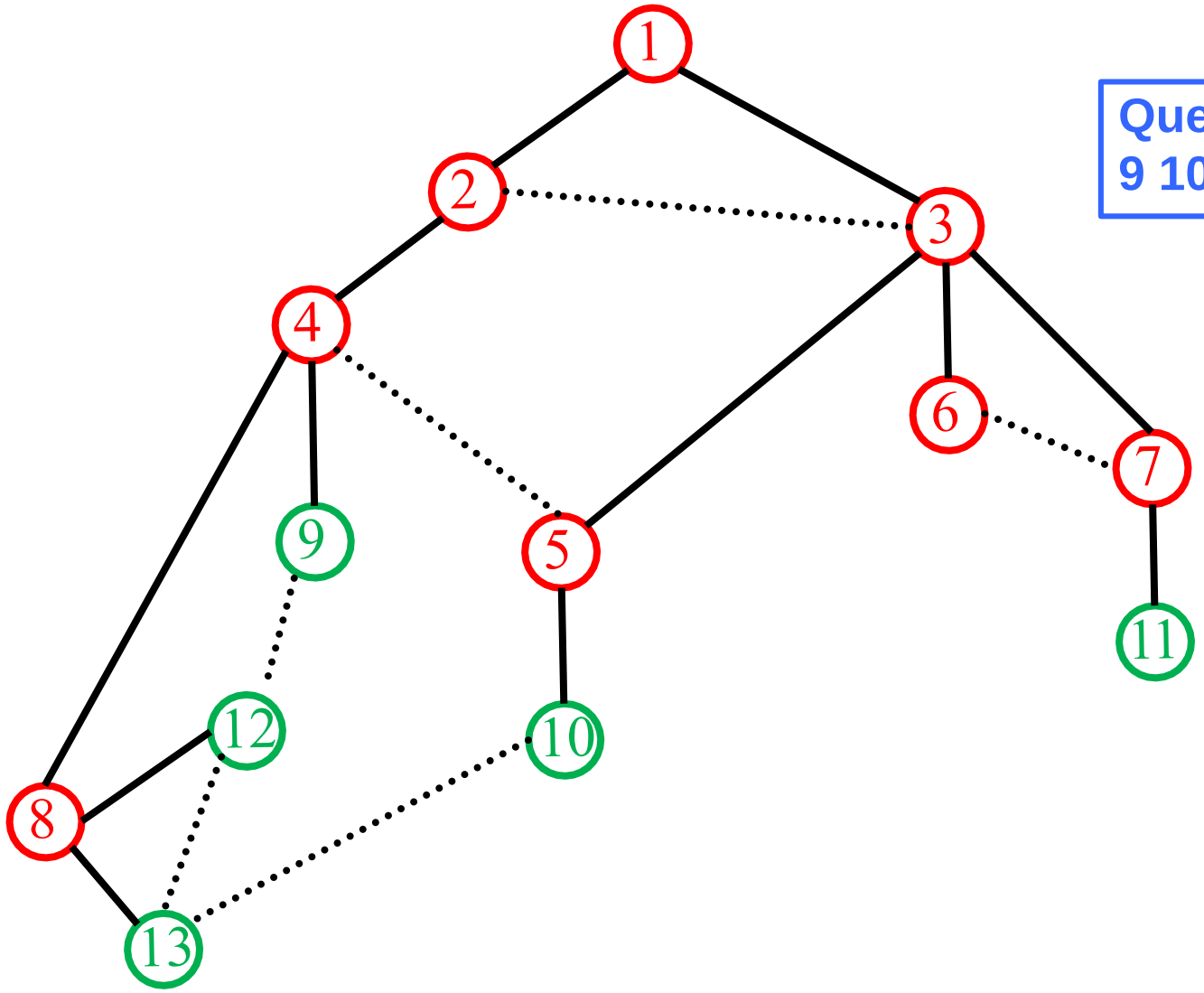
# BFS(1)



# BFS(1)

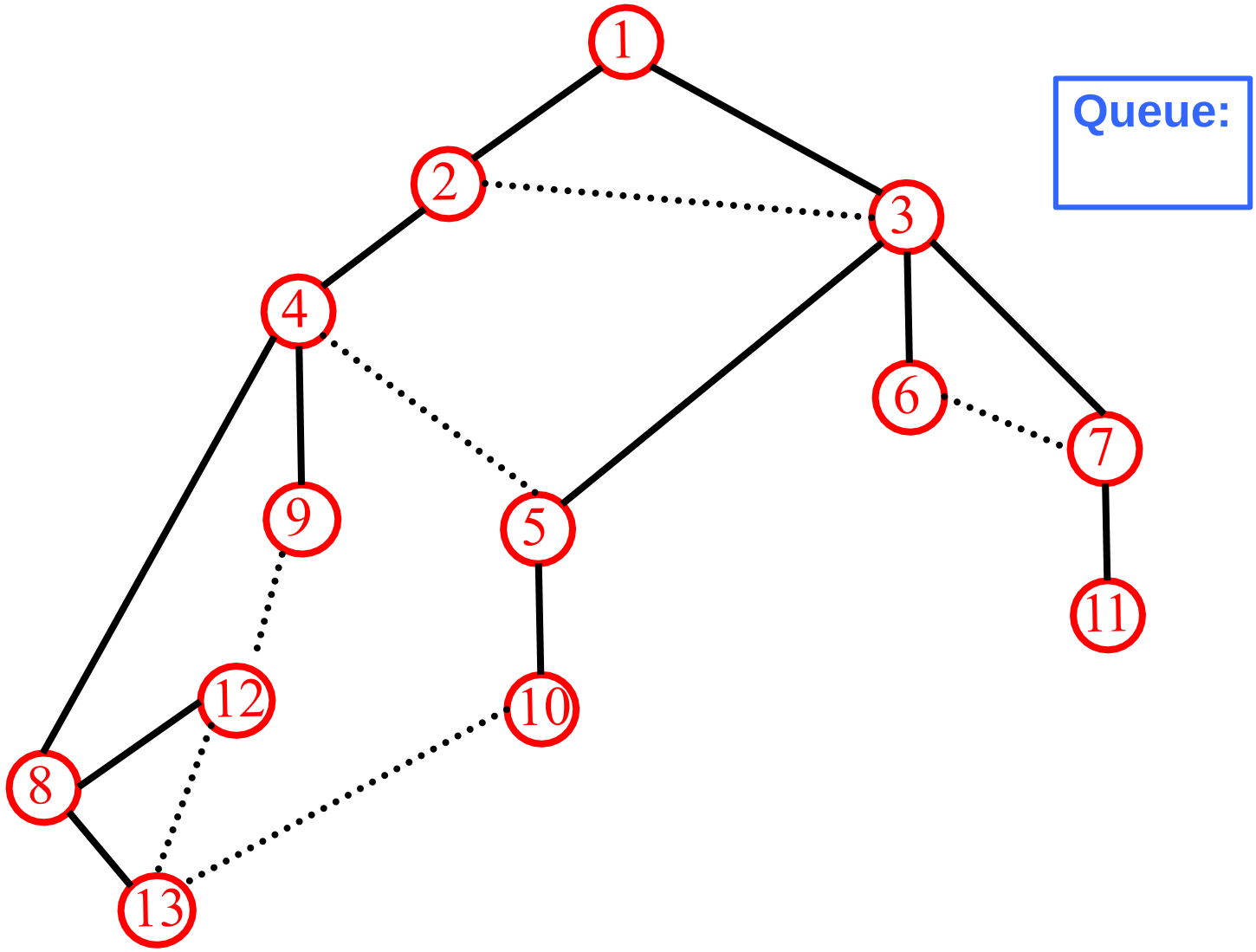


# BFS(1)




**Queue:**
**9 10 11 12 13**

# BFS(1)

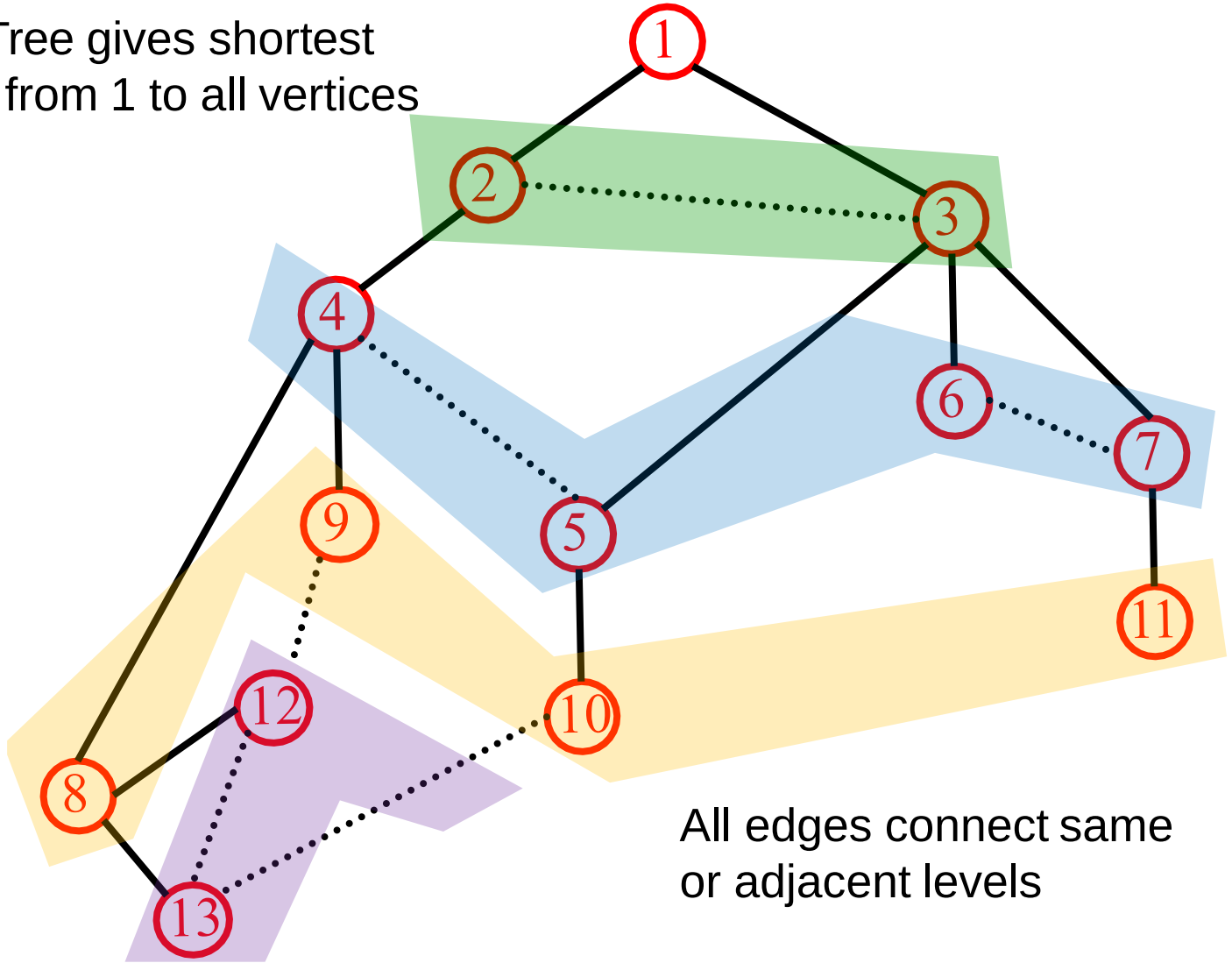


# Properties of BFS

- 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

# BFS Application: Shortest Paths

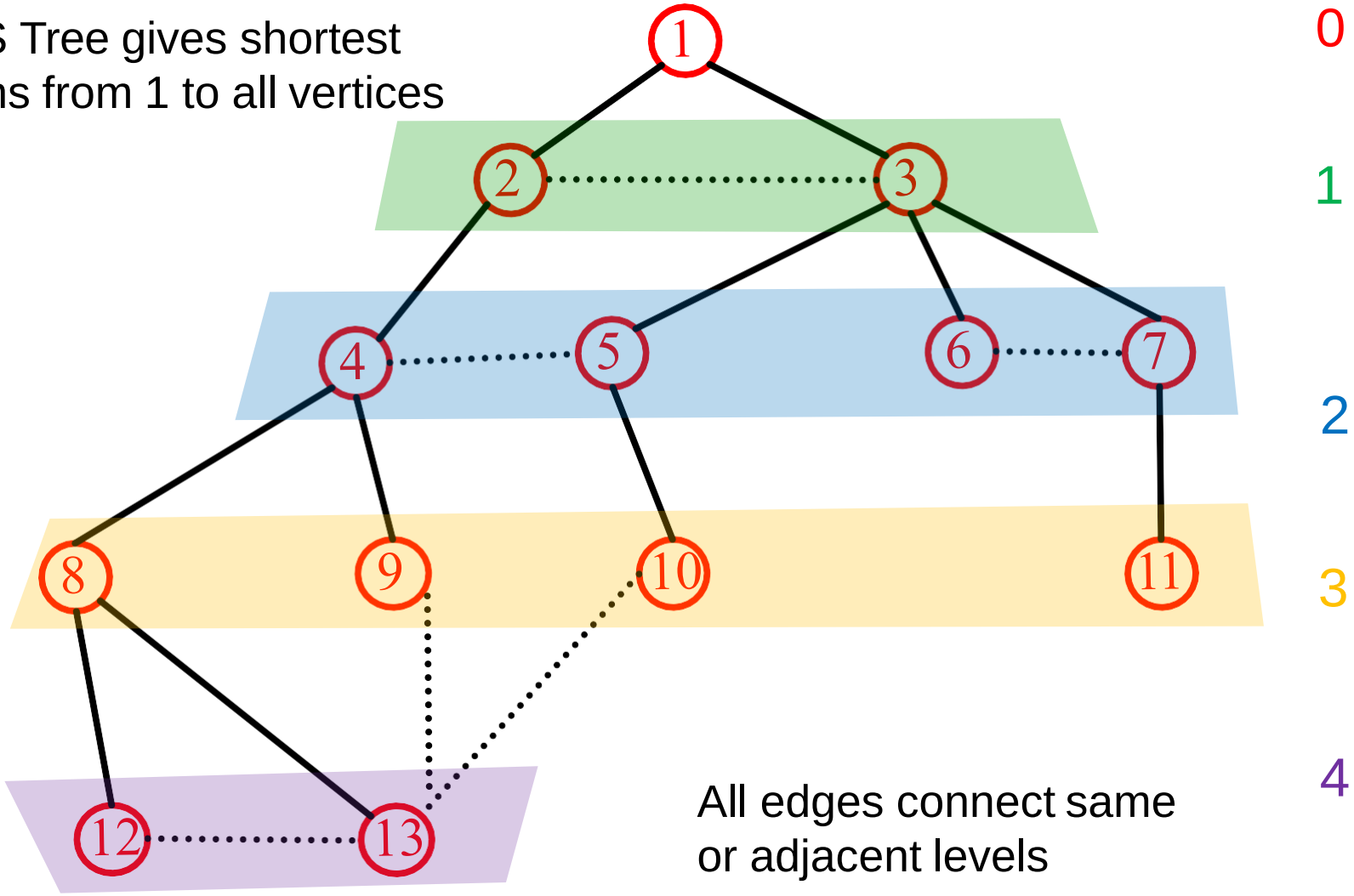
BFS Tree gives shortest paths from 1 to all vertices




All edges connect same or adjacent levels

# BFS Application: Shortest Paths

BFS Tree gives shortest paths from 1 to all vertices



# Properties of BFS

Claim: All nontree edges join vertices on the same or adjacent levels of the tree

Pf: Consider an edge {x,y}

Say x is first discovered and it is added to level $i$.

We show y will be at level $i$ or $i+1$

This is because when vertices incident to x are considered in the loop, if y is still undiscovered, it will be discovered and added to level $i+1$.

# Properties of BFS

Lemma: All vertices at level $i$ of BFS(s) have shortest path distance $i$ to s.

Claim: If $L(v) = i$ then shortest path $\leq i$

Pf: Because there is a path of length $i$ from $s$ to $v$ in the BFS tree

Claim: If shortest path = $i$ then $L(v) \leq i$

Pf: If shortest path = $i$, then say $s = v_0, v_1, \ldots, v_i = v$ is the shortest path to v.

By previous claim,

$$L(v_1) \leq L(v_0) + 1$$
$$L(v_2) \leq L(v_1) + 1$$
$$\ldots$$
$$L(v_i) \leq L(v_{i-1}) + 1$$

So, $L(v_i) \leq i$.

This proves the lemma.

# 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 simplified structure
- Solve the problem on the tree
- Use the solution on the tree to find a “good” solution on the graph

# Graph Search App: Connected Comp

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, A[u] is the label of the connected component that contains u

Therefore, question reduces to

If A[u] = A[v]?

# Connected Components Implementation

Initial State: All vertices undiscovered, c ←0

for v = 1 to n do
    If state(v) != fully-explored then
        BFS(v): setting A[u] ← c for each u found
        (and marking u discovered/fully-explored)

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 $O(m_i + n_i)$.

# 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

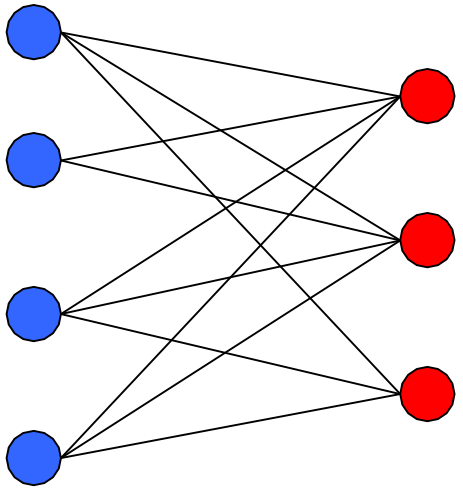
if you can partition the node set into 2 parts (say, blue/red or left/right) so that

all edges join nodes in different parts

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

Application:

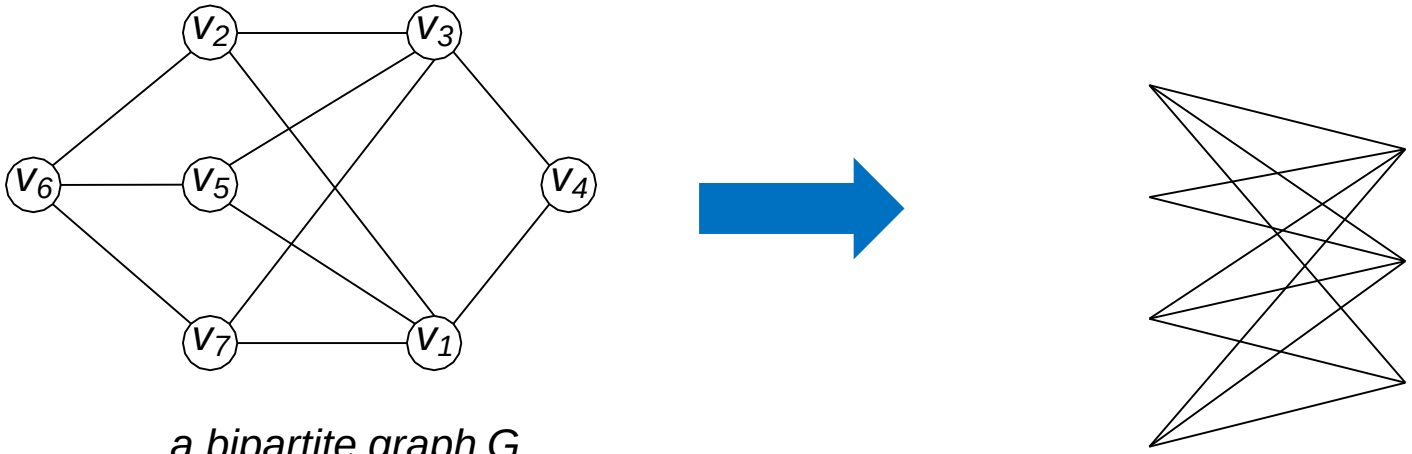
- Scheduling: machine=red, jobs=blue
- Stable Matching: men=blue, wom=red



*a bipartite graph*

# Testing Bipartiteness

Problem: Given a graph G, is it bipartite?




*a bipartite graph G*

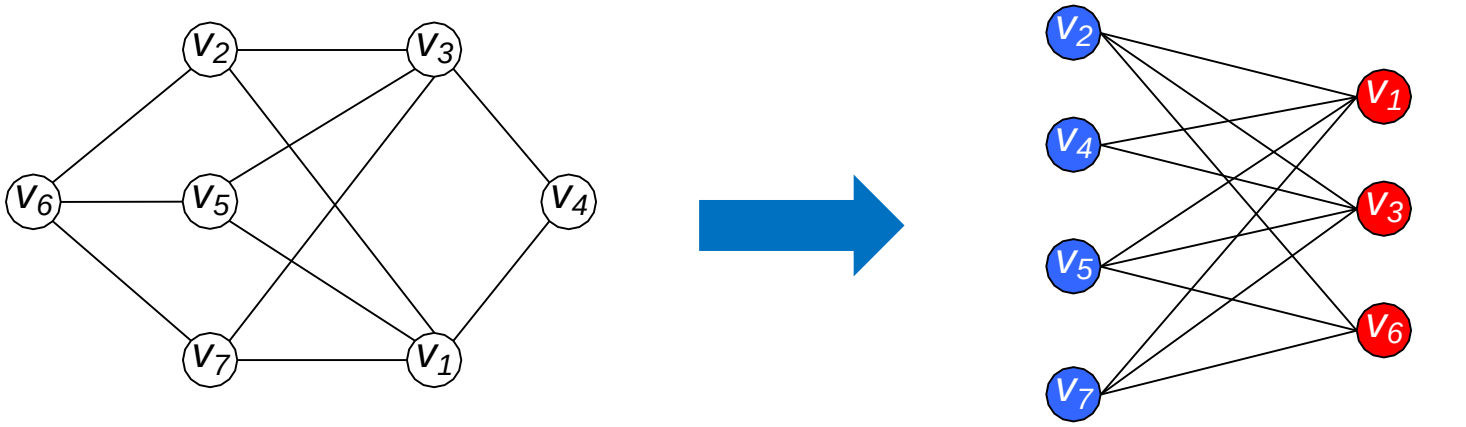
# Testing Bipartiteness

Problem: Given a graph G, is it bipartite?

Many graph problems become:

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

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



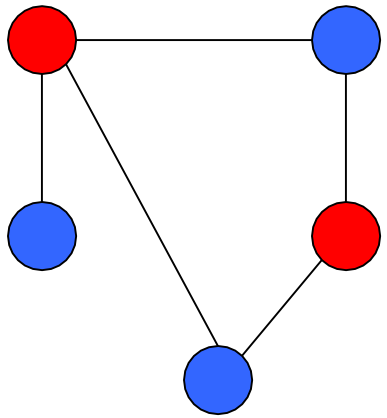

*a bipartite graph G*

*another drawing of G*

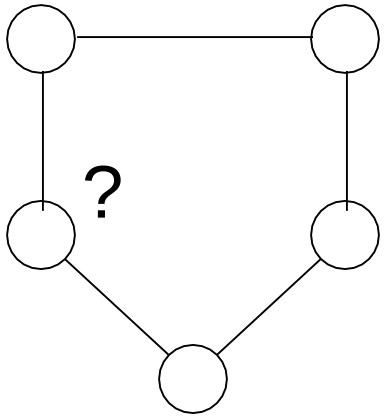
# An Obstruction to Bipartiteness

Lemma: If G is bipartite, then it does not contain an odd length cycle.

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



*bipartite*
*(2-colorable)*



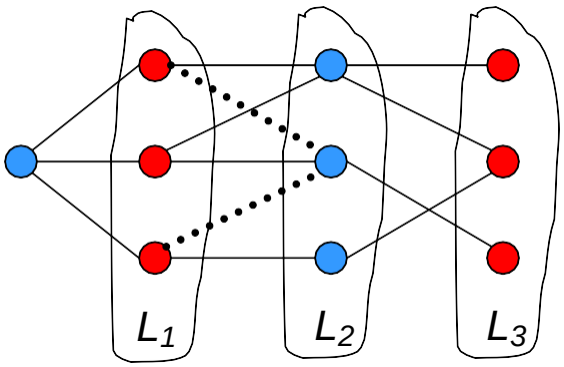

*not bipartite*
*(not 2-colorable)*

# A Characterization of Bipartite Graphs

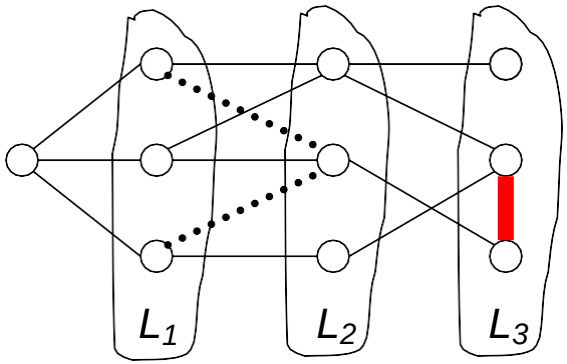
Lemma: Let G be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS(s). Exactly one of the following holds.

(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

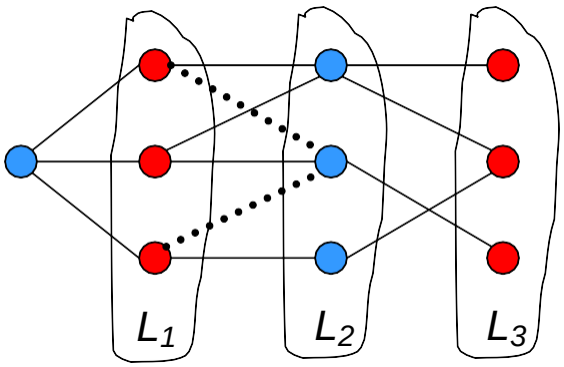
Lemma: Let G be a connected graph, and let $L_0$, … ,$L_k$ be the layers produced by BFS(s). Exactly one of the following holds.

(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).

Pf. (i) Suppose no edge joins two nodes in the same layer.

By previous lemma, 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).

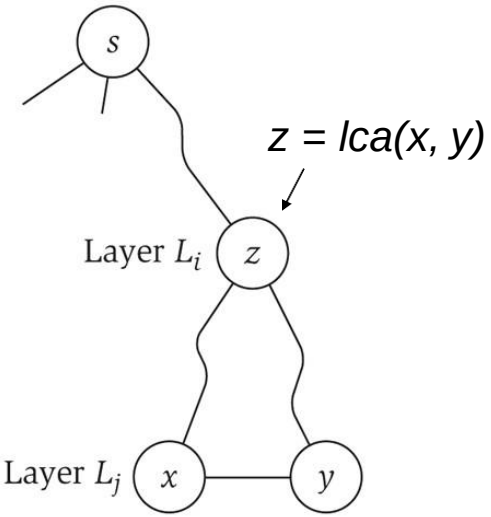
Pf. (ii) Suppose (x, y) is an edge & x, y in same level $L_j$.

Let z = their lowest common ancestor in BFS tree.

Let $L_i$ be level containing z.

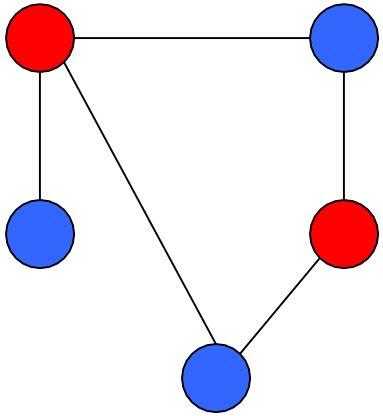
Consider cycle that takes edge from x to y, then tree from y to z, then tree from z to x.

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

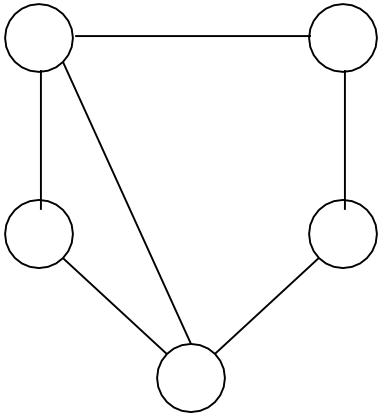


# Obstruction to Bipartiteness

Cor: A graph G is bipartite iff it contains no odd length cycles.



*bipartite*
*(2-colorable)*



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