CSE 421: Introduction to Algorithms

Breadth First Search

Yin Tat Lee

Degree 1 vertices

Claim: If G has no cycle, then it has a vertex of degree ≤ 1

(Every tree has a leaf)

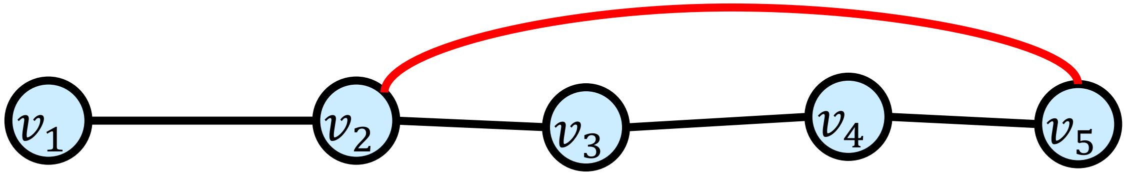
Proof: (By contradiction)

Suppose every vertex has degree ≥ 2.

Start from a vertex v_1 and follow a path, $v_1, \dots, v_i$ when we are at v_i we choose the next vertex to be different from v_{i-1}. We can do so because $\deg(v_i) \geq 2$.

The first time that we see a repeated vertex $(v_j = v_i)$ we get a cycle.

We always get a repeated vertex because G has finitely many vertices



Trees and Induction

Claim: Show that every tree with n vertices has $n - 1$ edges.

Proof: (Induction on n.)

Base Case: $n = 1$, the tree has no edge

Inductive Step: Let T be a tree with n vertices.

So, T has a vertex v of degree 1.

Remove v and the neighboring edge, and let T' be the new graph.

We claim T' is a tree: It has no cycle, and it must be connected.

So, T' has $n - 2$ edges and T has $n - 1$ edges.

Graph Traversal

Walk (via edges) from a fixed starting vertex s to all vertices reachable from s.

- Breadth First Search (BFS): Order nodes in successive layers based on distance from s
- Depth First Search (DFS): More natural approach for exploring a maze;

Applications of BFS:

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

Breadth First Search (BFS)

Completely explore the vertices in order of their distance from s.

Three states of vertices:

- Undiscovered
- Discovered
- Fully-explored

Naturally implemented using a queue

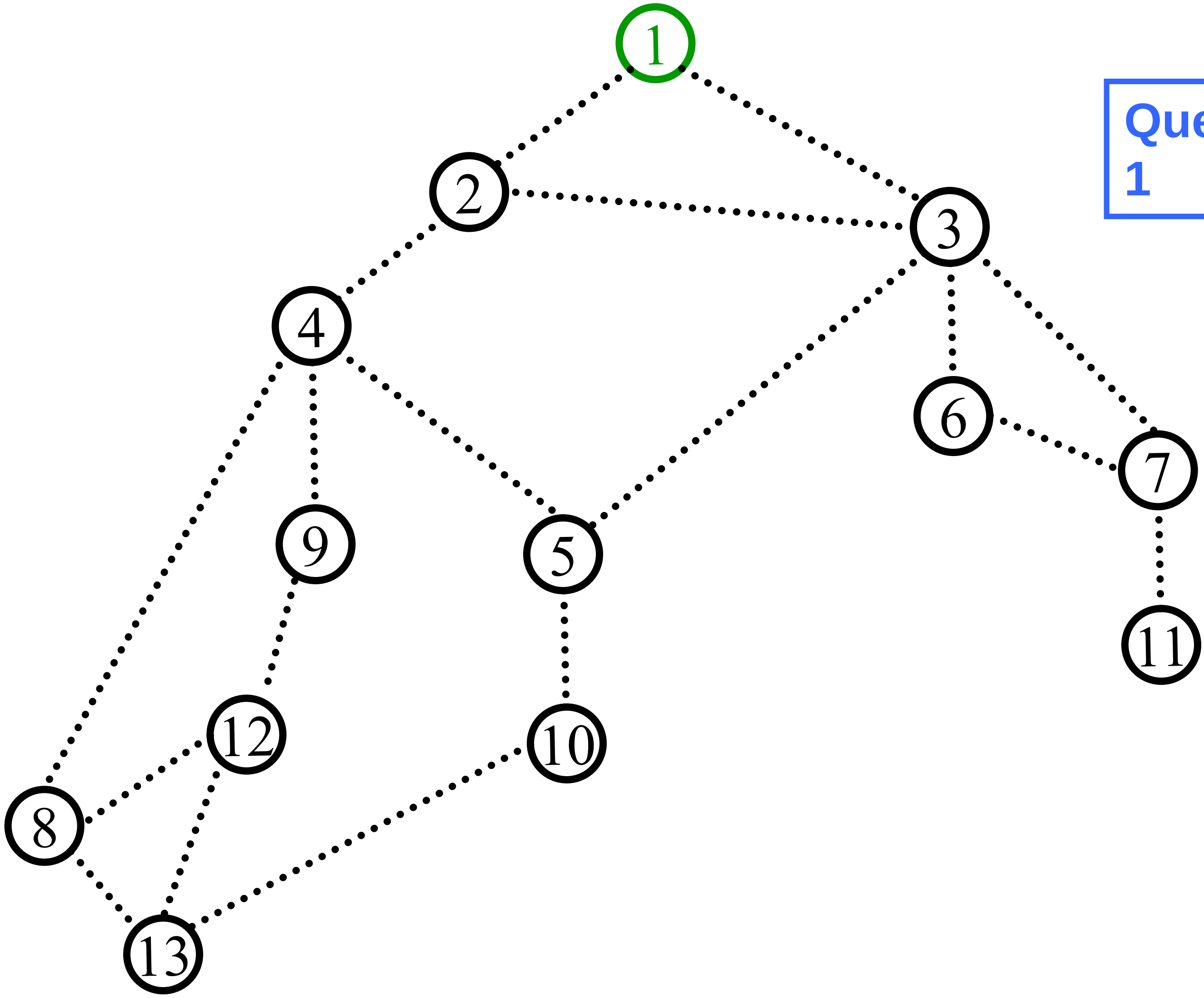
The queue will always have the list of Discovered vertices

BFS implementation

Initialization: mark all vertices "undiscovered"

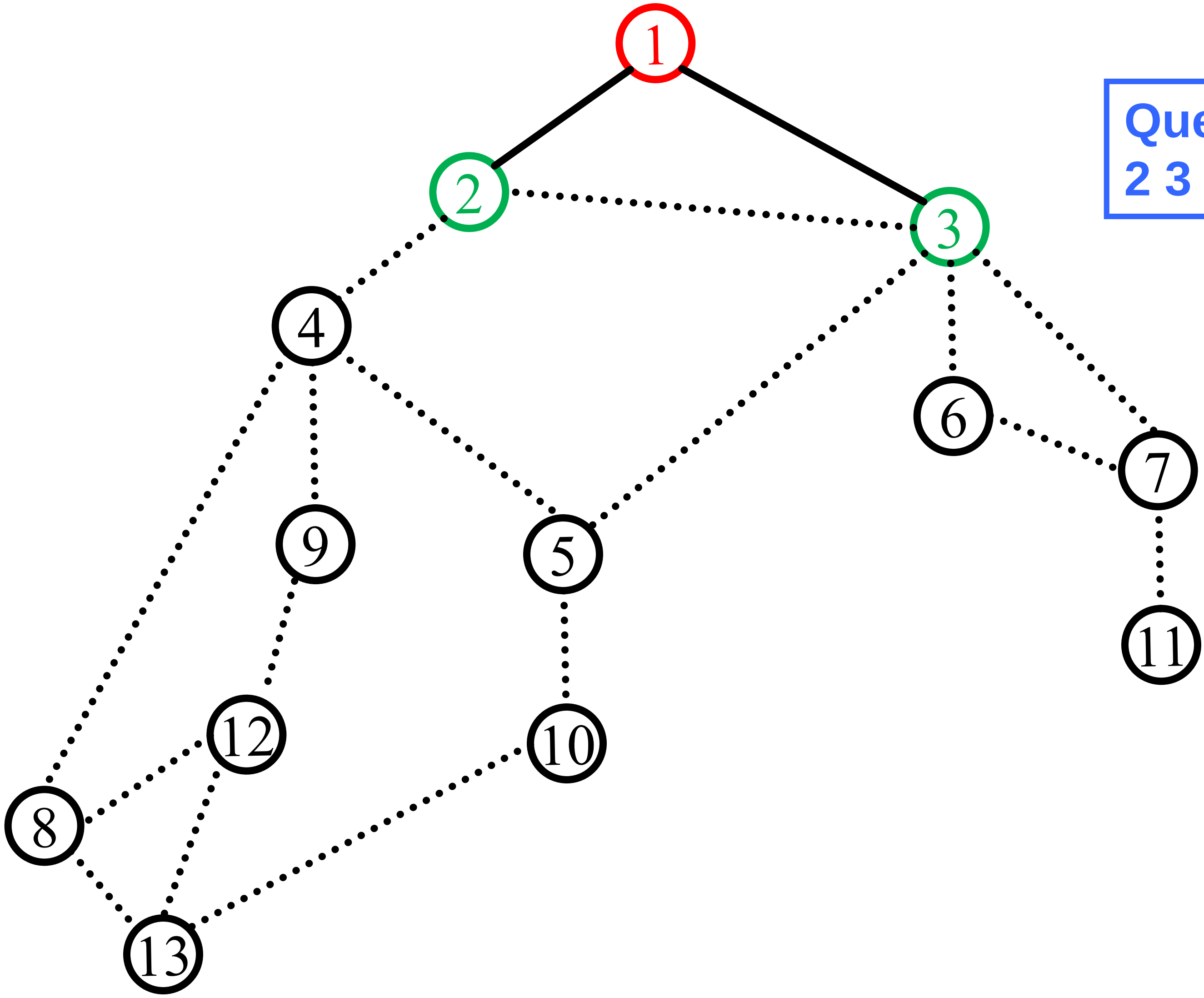
```
BFS($s$)
    mark $s$ discovered
    queue = { $s$ }
    while queue not empty
        $u$ = remove_first(queue)
        for each edge $\{u, x\}$
            if ($x$ is undiscovered)
                mark $x$ discovered
                append $x$ on queue
        mark $u$ fully-explored
```

BFS(1)



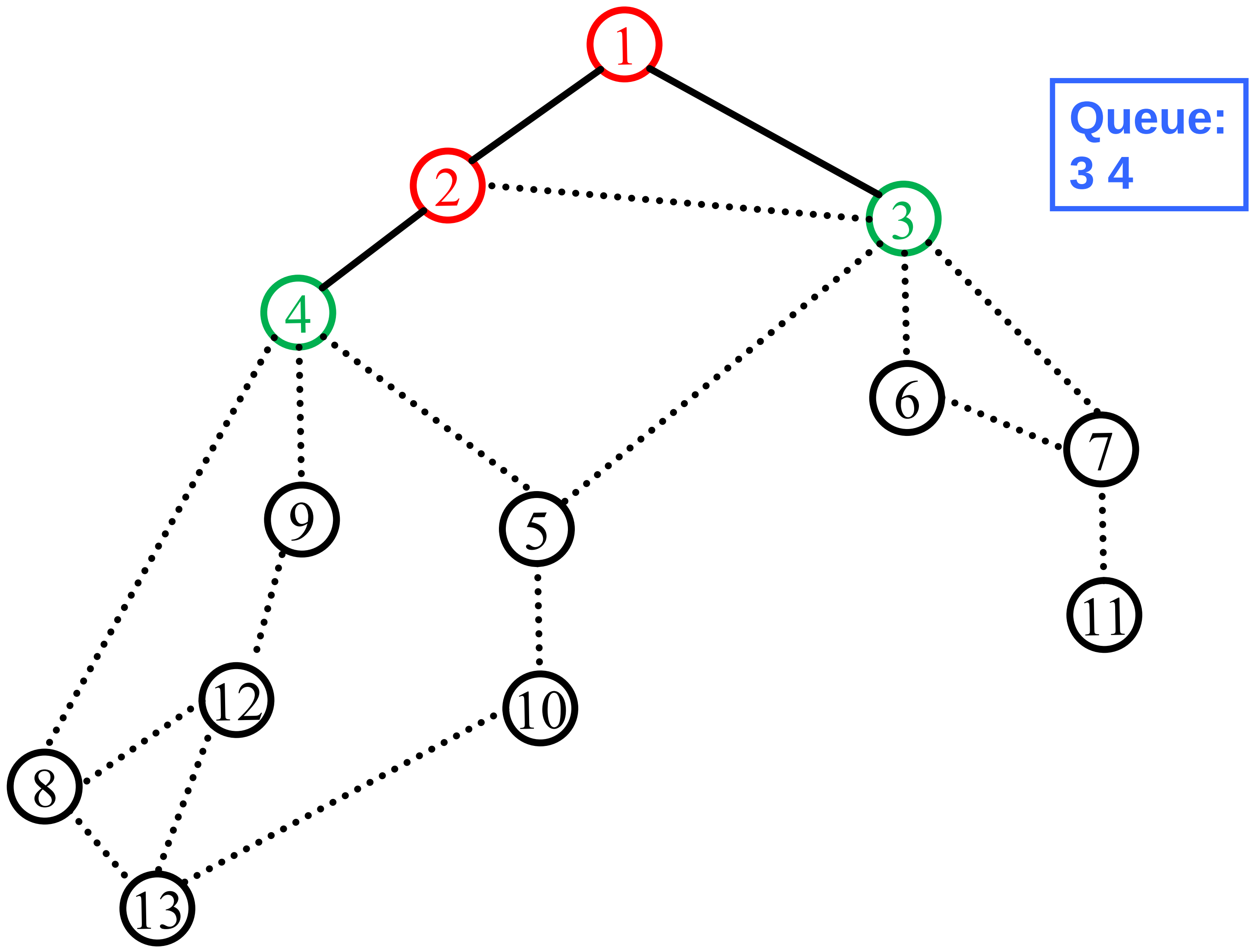
Queue:
1

BFS(1)

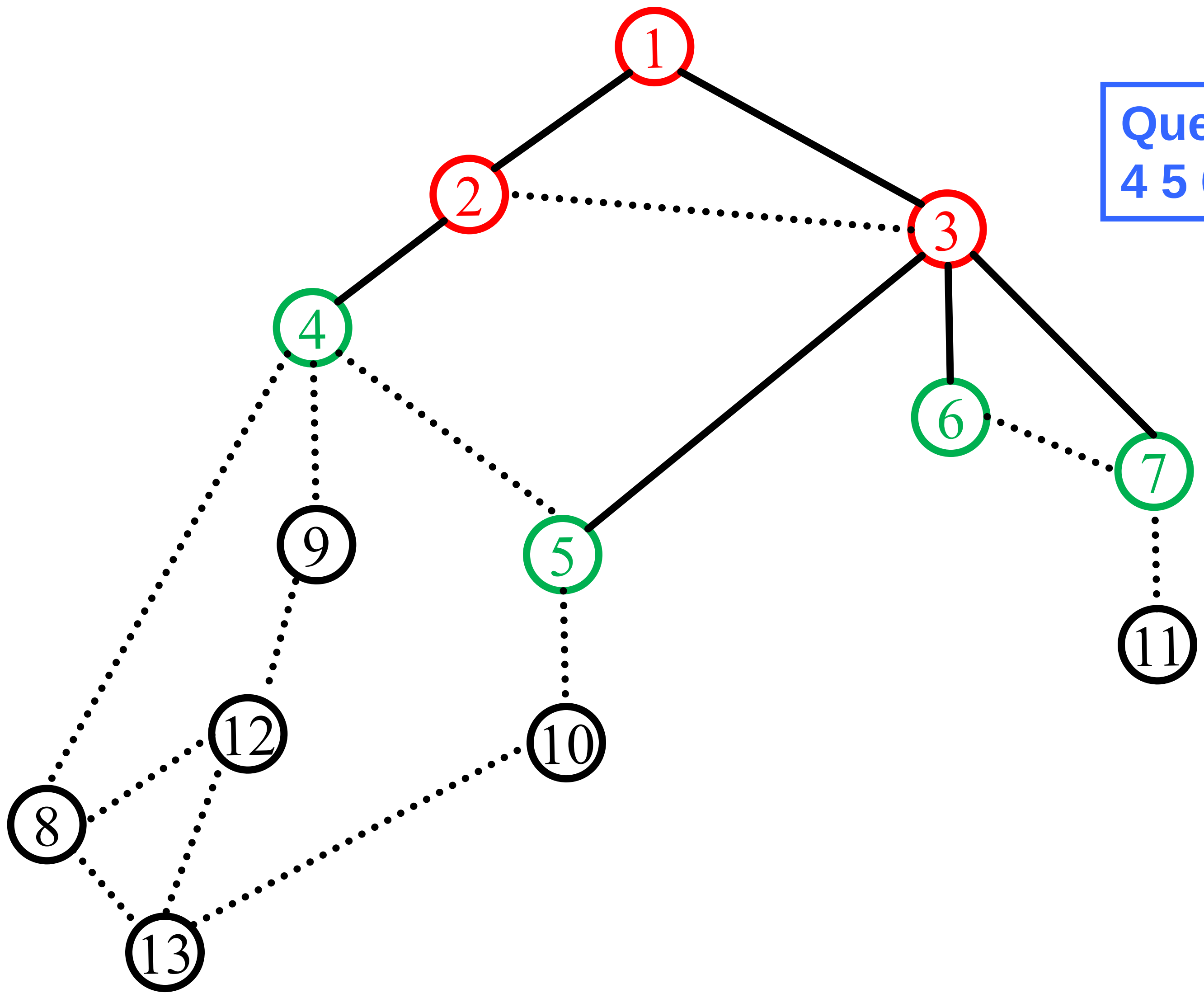


Queue:
2 3

BFS(1)

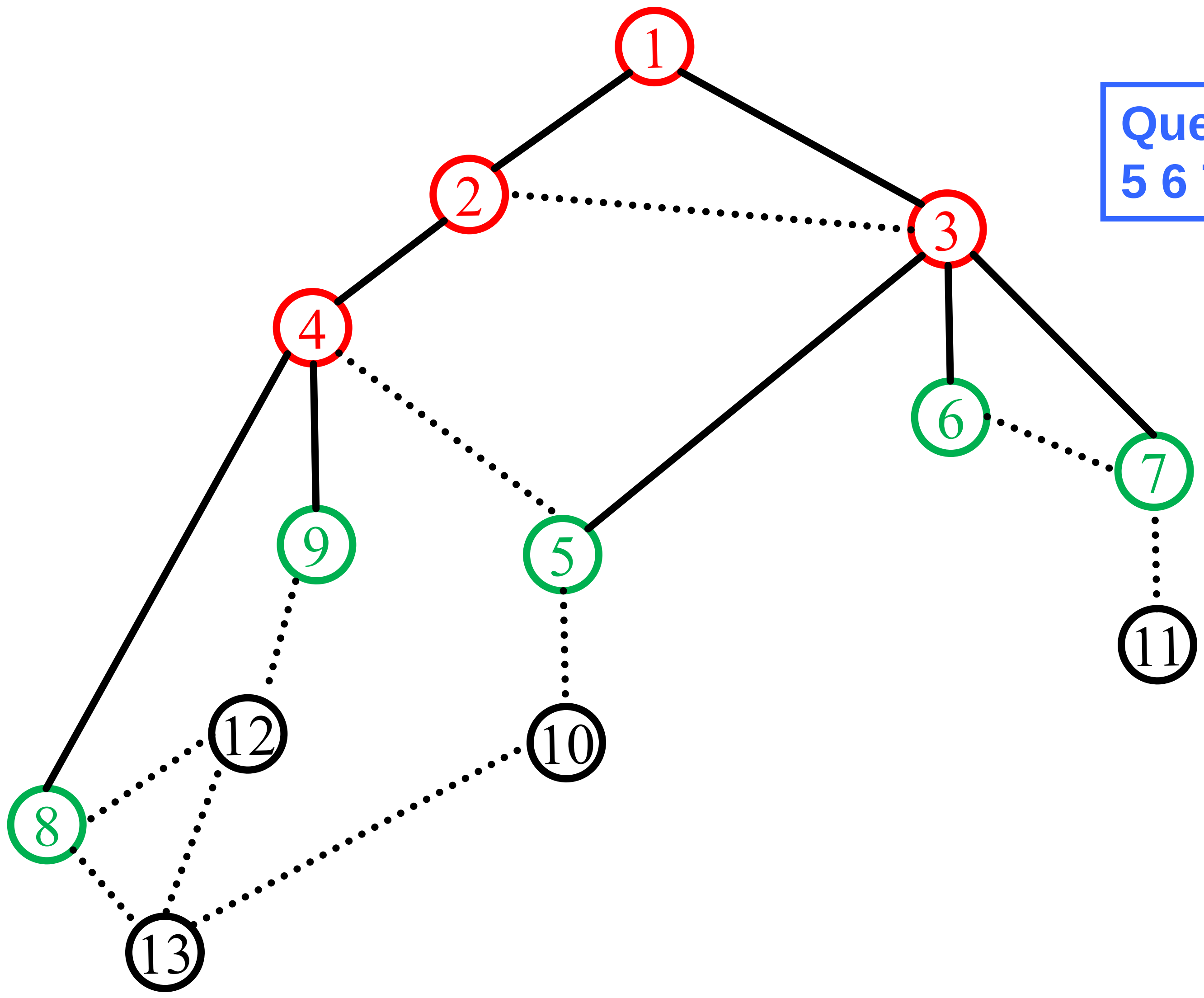


BFS(1)



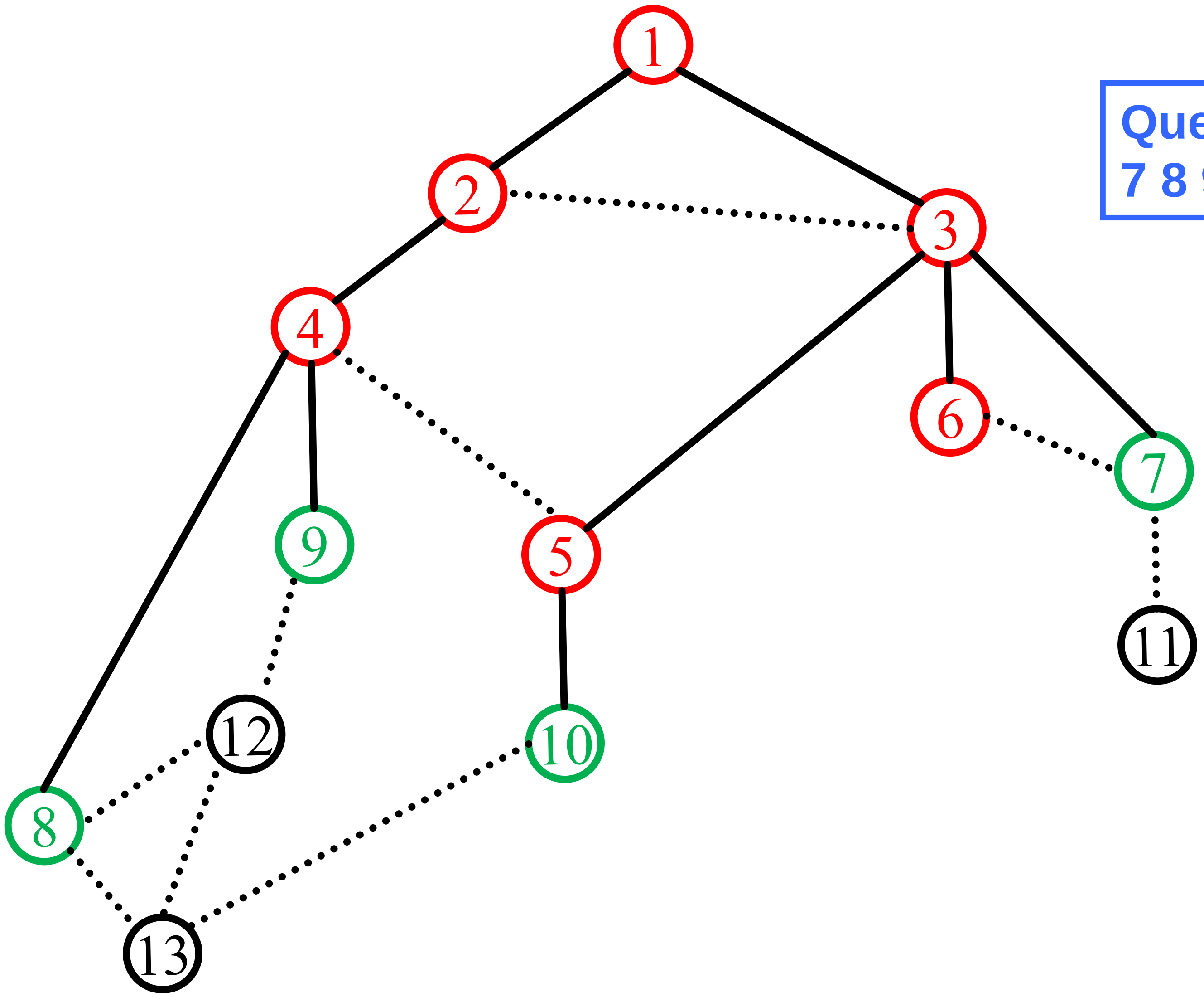
Queue:
4 5 6 7

BFS(1)



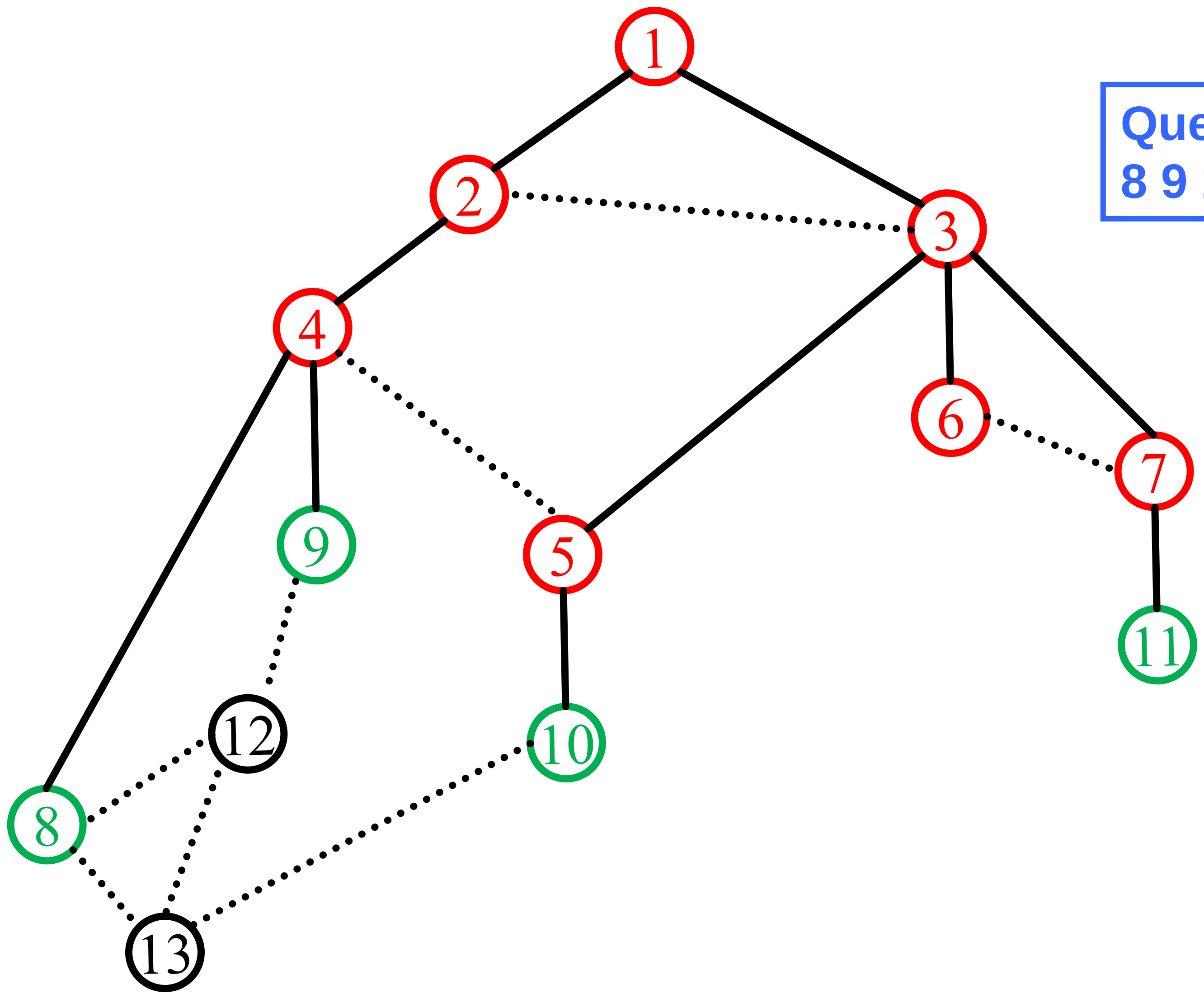
Queue:
5 6 7 8 9

BFS(1)



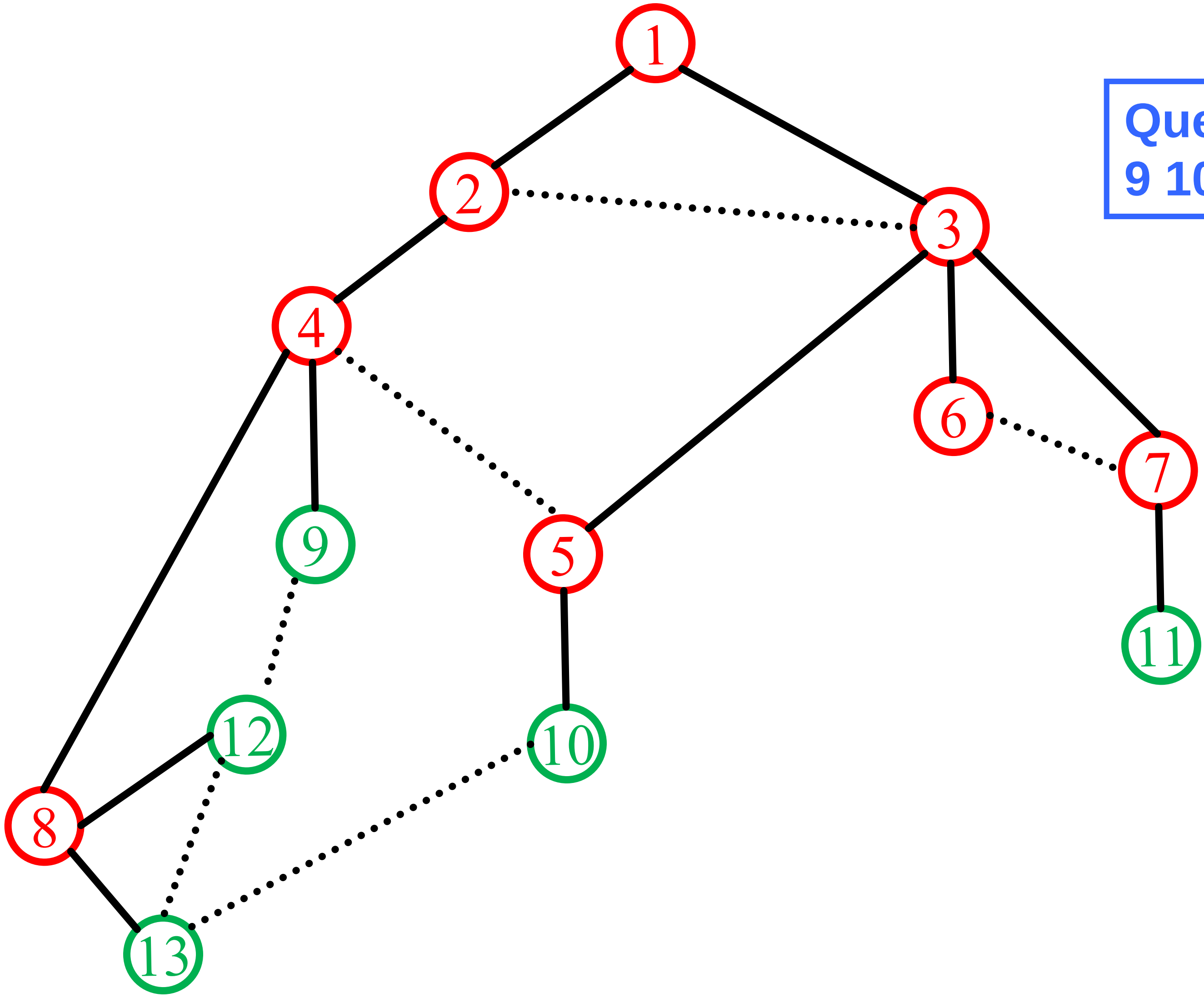
Queue:
7 8 9 10

BFS(1)



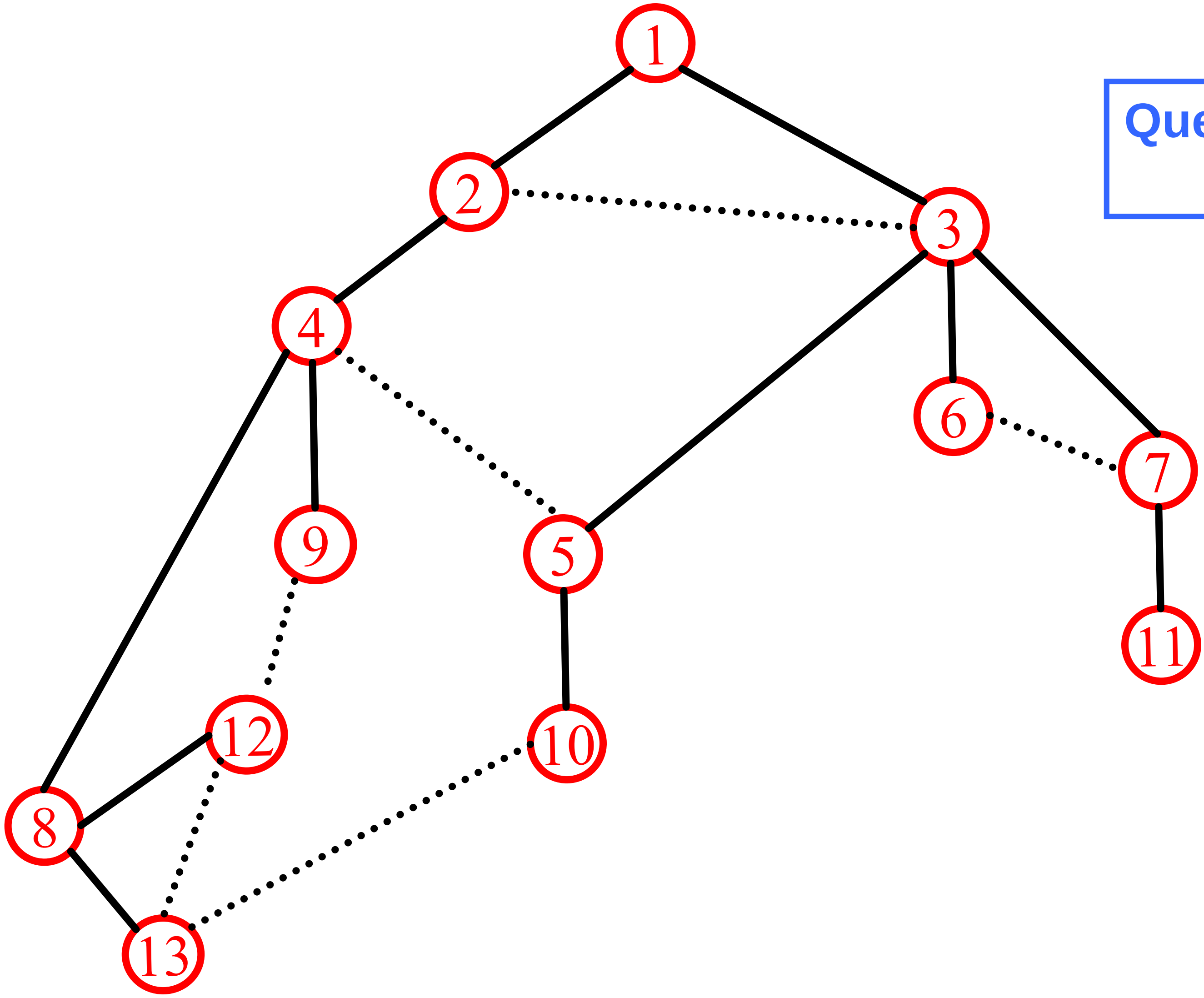
Queue:
8 9 10 11

BFS(1)



Queue:
9 10 11 12 13

BFS(1)



Queue:

BFS Analysis

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
```

O(n) times:
At most once per vertex

O(m) times:
At most twice per edge

Properties of BFS

- BFS(s) visits a vertex v if and only if there is a path from s to v

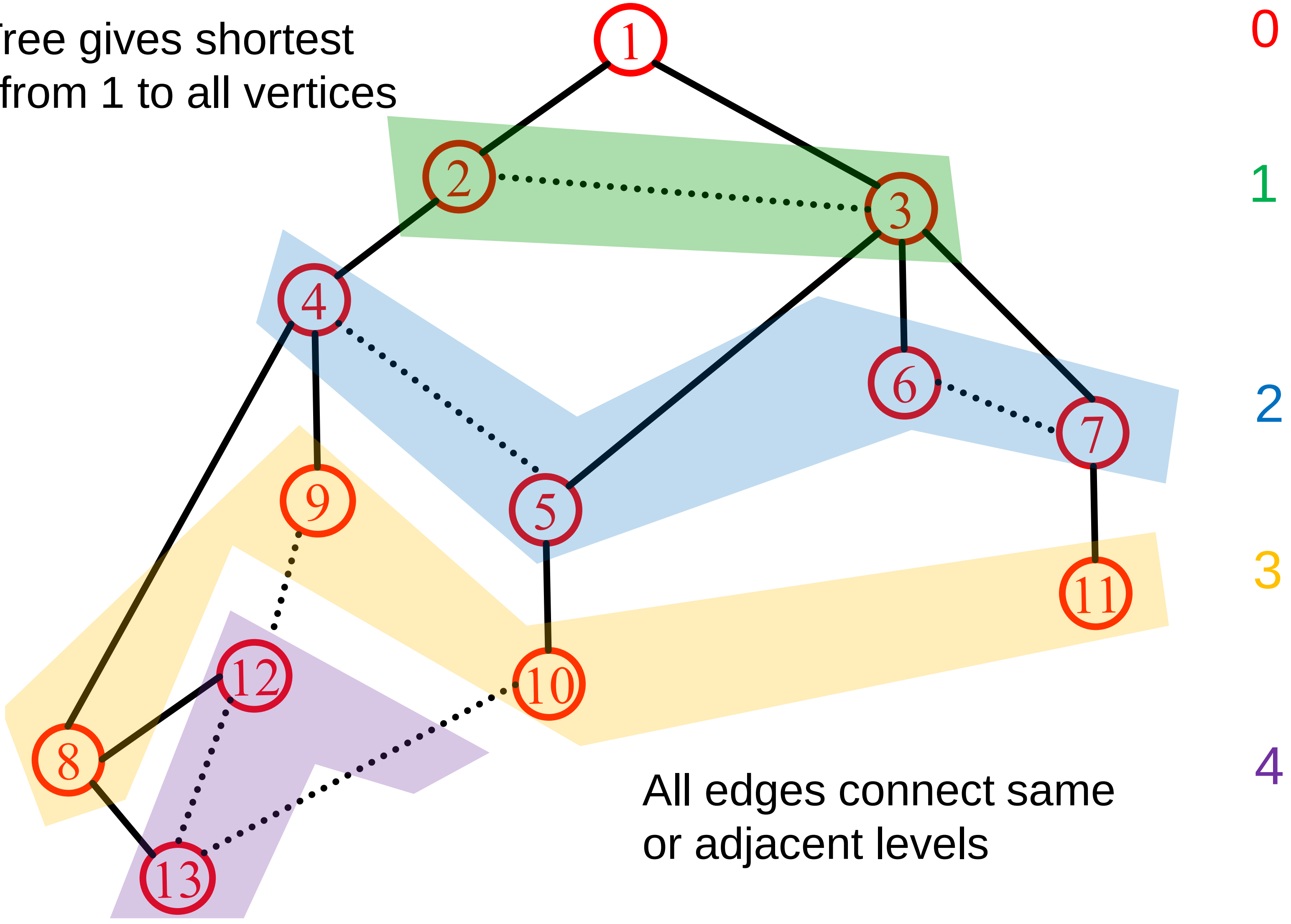
- Edges into then-undiscovered vertices define a tree – the “Breadth First spanning tree” of G

- 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

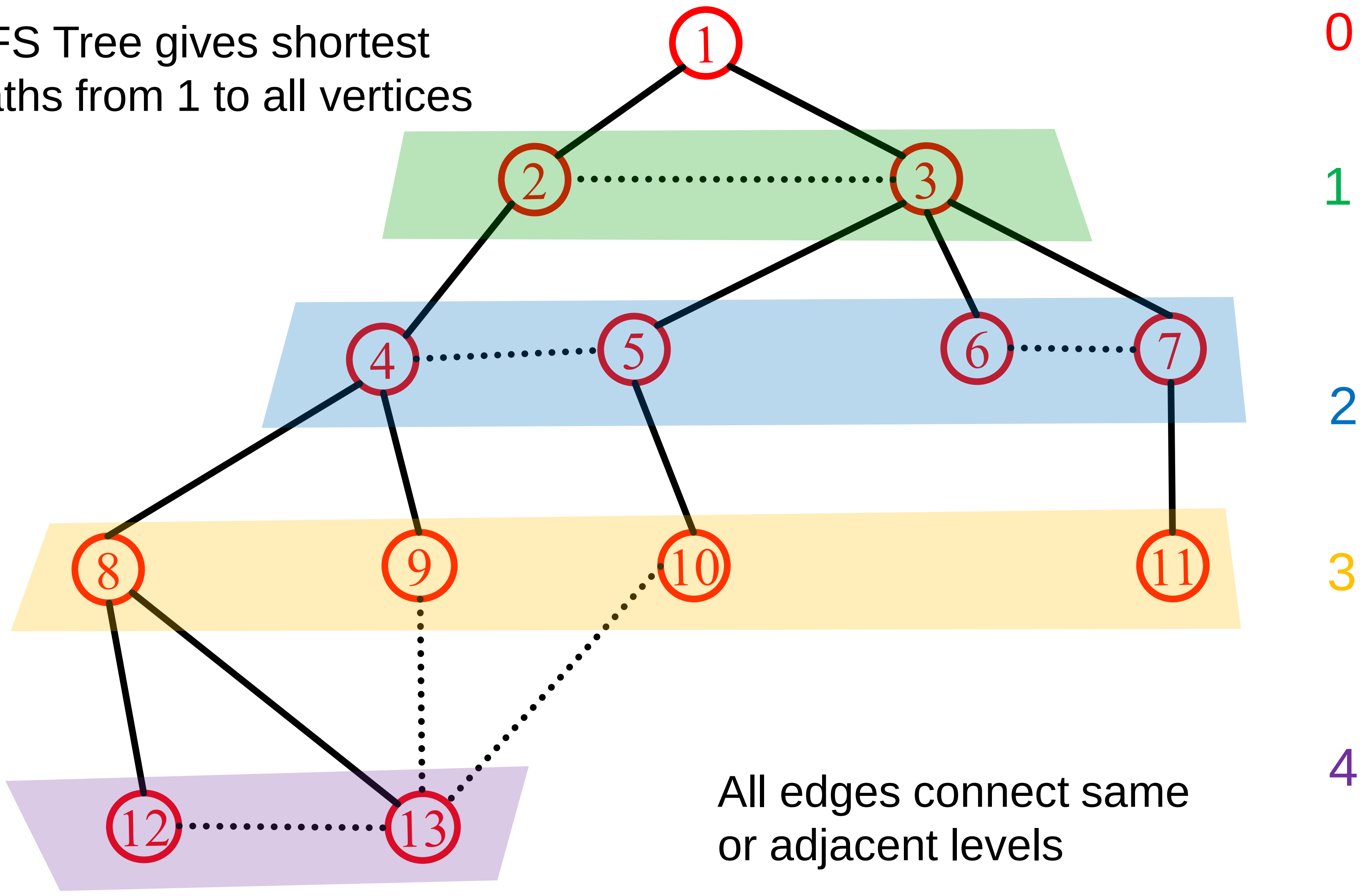
BFS Tree gives shortest paths from 1 to all vertices



All edges connect same or adjacent levels

BFS Application: Shortest Paths

BFS Tree gives shortest paths from 1 to all vertices



All edges connect same or adjacent levels

Properties of BFS

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

Proof: 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 $\le i$

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

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

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

By previous claim,

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

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

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

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.

Cycles in Graphs

Claim: If an n vertices graph G has at least n edges, then it has a cycle.

Proof: If G is connected, then it cannot be a tree. Because every tree has $n-1$ edges. So, it has a cycle.

Suppose G is disconnected. Say connected components of G have $n_1, \ldots, n_k$vertices where $n_1 + \cdots + n_k = n$

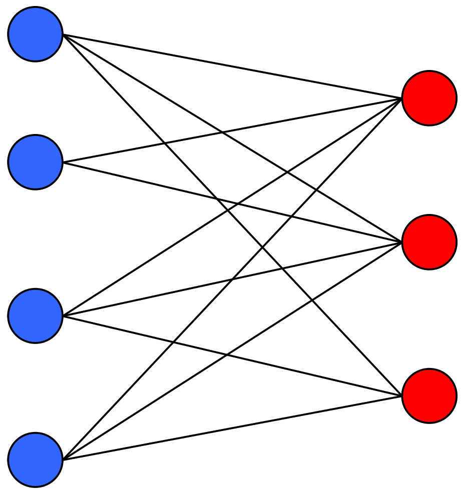
Since G has $\geq n$ edges, there must be some i such that a component has n_i vertices with at least n_i edges.

Therefore, in that component we do not have a tree, so there is a cycle.

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

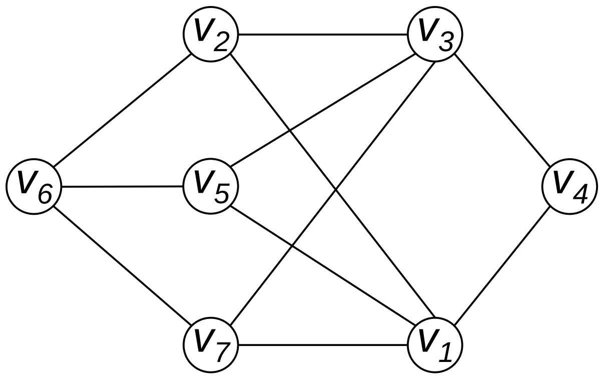
Testing Bipartiteness

Problem: Given a graph G, is it bipartite?

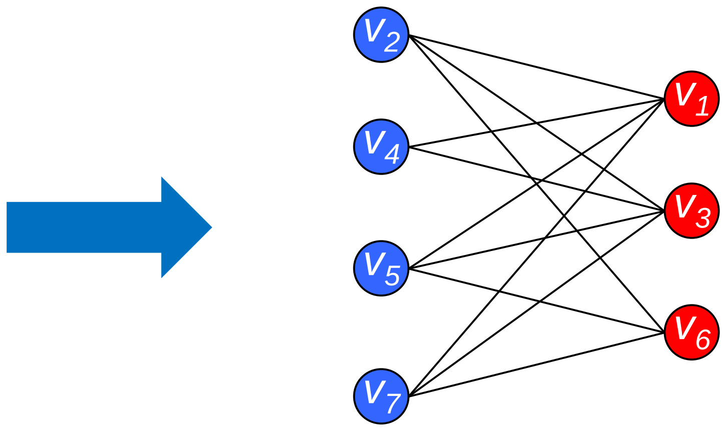
Many graph problems become:

- 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

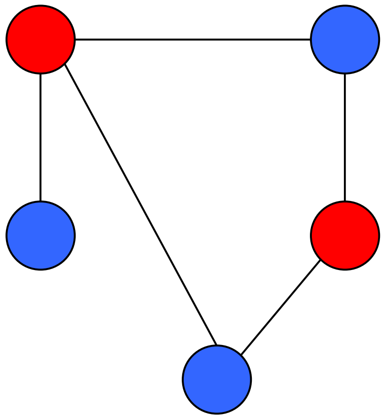


another drawing of G

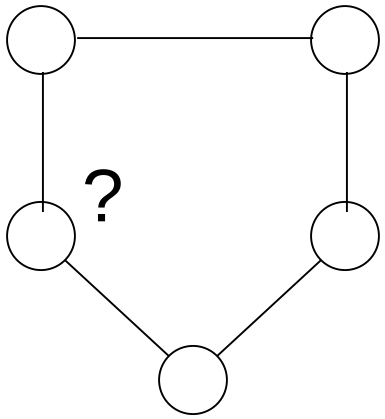
An Obstruction to Bipartiteness

Lemma: If 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)



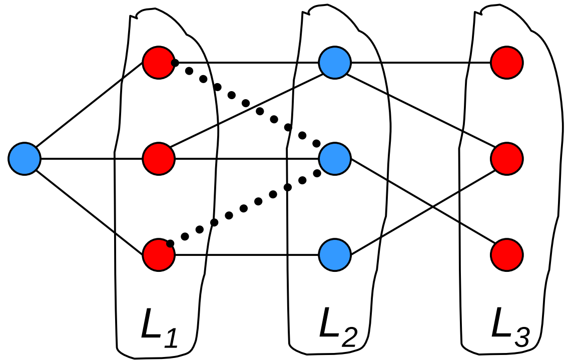
not bipartite
(not 2-colorable)

A Characterization of Bipartite Graphs

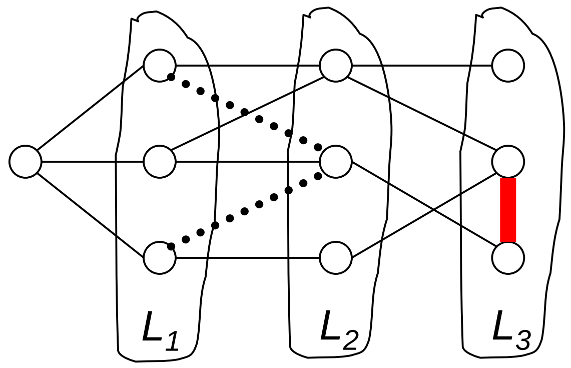
Lemma: Let G be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS(s). Exactly one of the following holds.

(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.

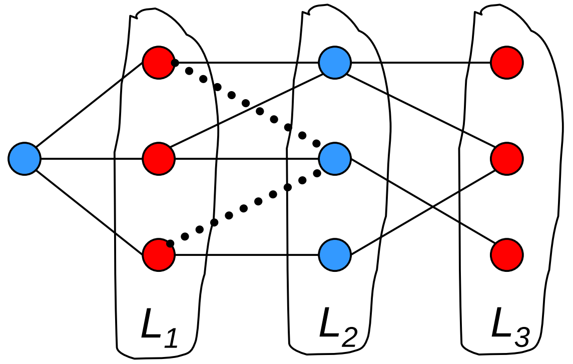
(i) No edge of G joins two nodes of the same layer, and G is bipartite.

(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.

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).

Proof. (ii)

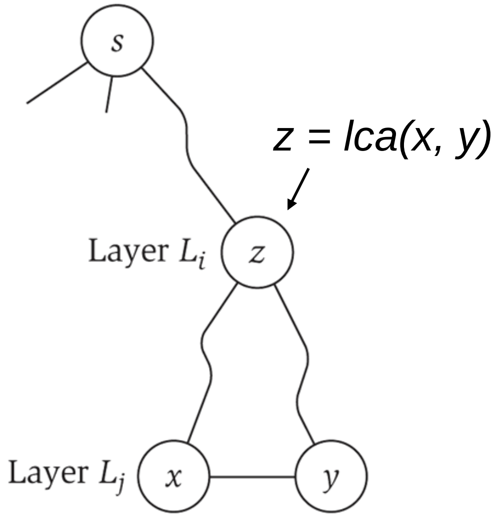
Suppose $\{x, y\}$ is an edge & x, y in same level L_j.

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.

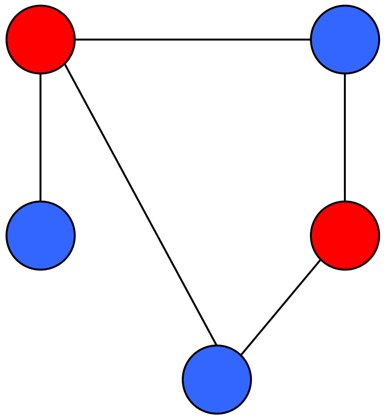
Its length is $1 + (j - i) + (j - i)$, which is odd.

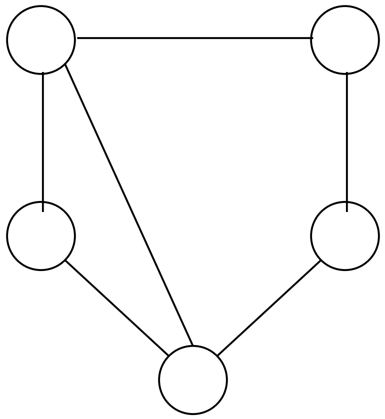


Obstruction to Bipartiteness

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)