CSE 421: Introduction to Algorithms

BFS

Shayan Oveis Gharan

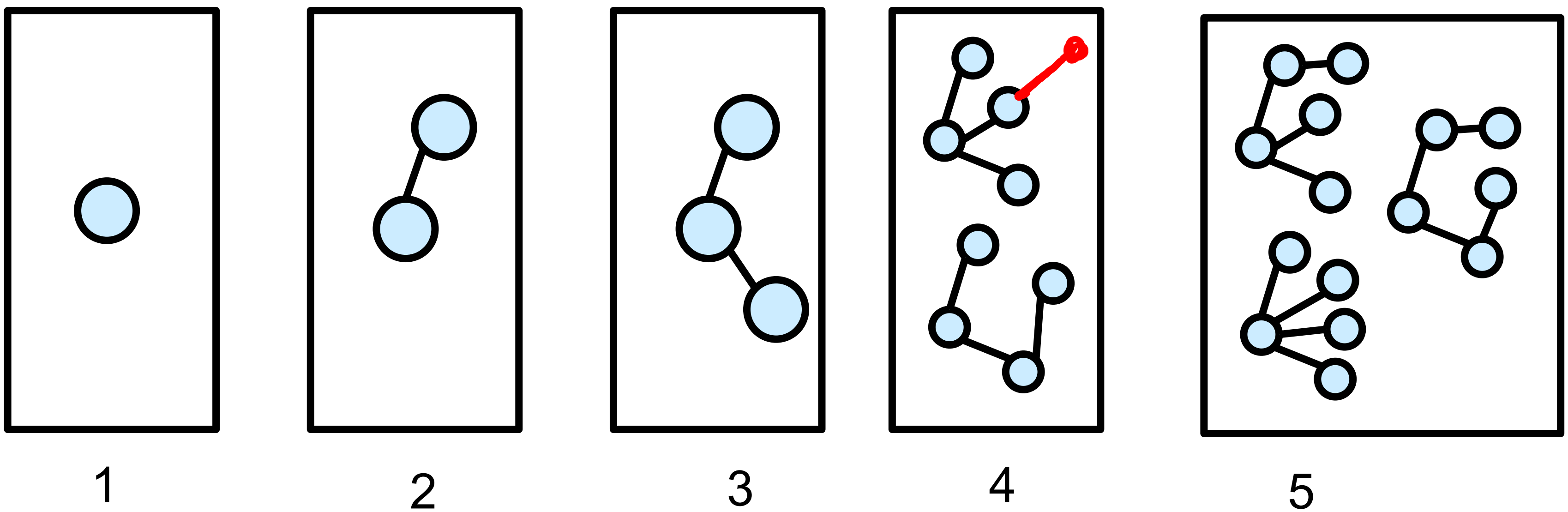
Induction

Induction in 311:

Prove $1 + 2 + \cdots + n = n(n + 1)/2$

Induction in 421:

Prove all trees with n vertices have $n - 1$ edges



#edges

Let $G = (V, E)$ be a graph with $n = |V|$ vertices and $m = |E|$ edges.

Claim: $0 \leq m \leq \binom{n}{2} = \frac{n(n-1)}{2} = O(n^2)$

Pf: Since every edge connects two distinct vertices (i.e., G has no loops)

and no two edges connect the same pair of vertices (i.e., G has no multi-edges)

It has at most $\binom{n}{2}$ edges.

Sparse Graphs

A graph is called sparse if $m \ll n^2$ and it is called dense otherwise.

Sparse graphs are very common in practice

- Friendships in social network
- Planar graphs
- Web braph

Q: Which is a better running time $O(n+m)$ vs $O(n^2)$?

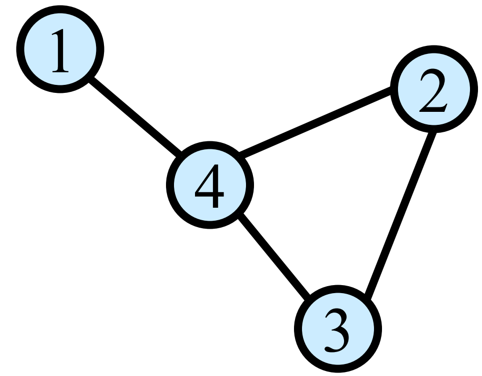
A: $O(n+m) = O(n^2)$, but $O(n+m)$ is usually much better.

Storing Graphs (Internally in ALG)

Vertex set $V = \{v_1, \ldots, v_n\}$.

Adjacency Matrix: A

- For all, $i, j, A[i,j] = 1$ iff $(v_i, v_j) \in E$
- Storage: n^2 bits



	1	2	3	4
1	0	0	0	1
2	0	0	1	1
3	0	1	0	1
4	1	1	1	0

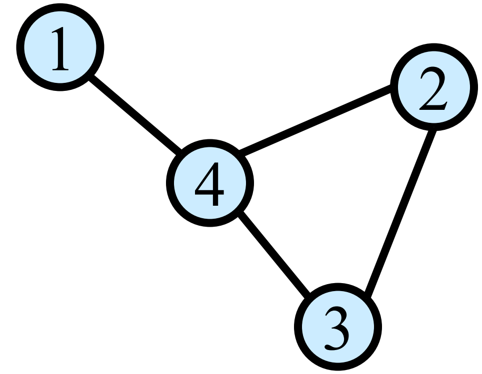
Advantage:

- $O(1)$ test for presence or absence of edges

Disadvantage:

- Inefficient for sparse graphs both in storage and edge-access

Storing Graphs (Internally in ALG)

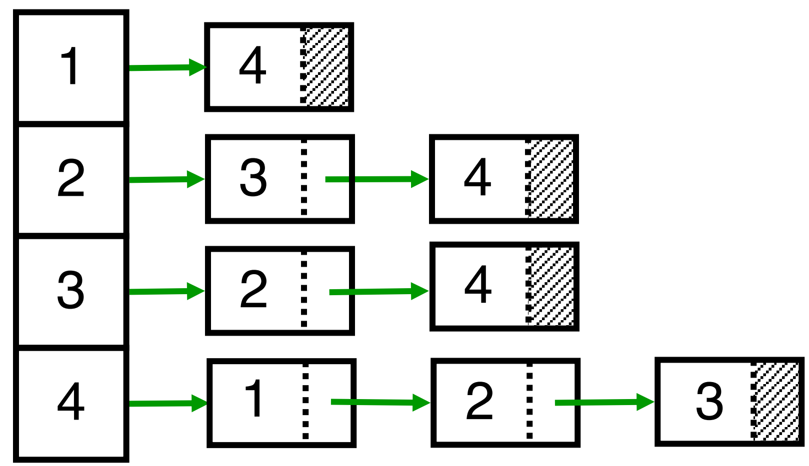


Adjacency List:

O(n+m) words

Advantage

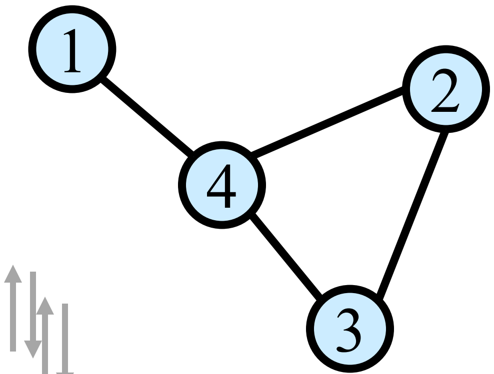
- Compact for sparse
- Easily see all edges



Disadvantage

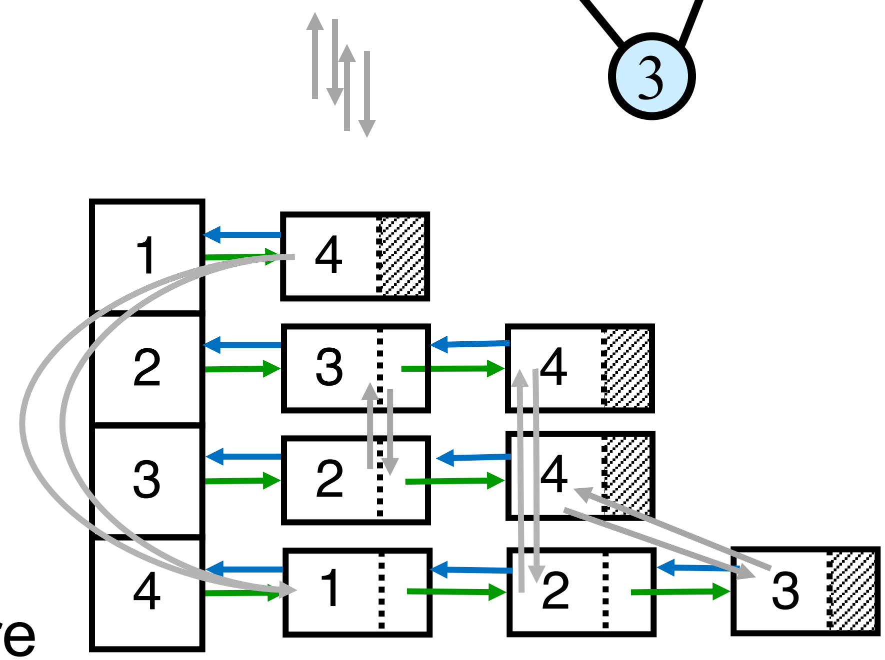
- No O(1) edge test
- More complex data structure

Storing Graphs (Internally in ALG)



Adjacency List:

O(n+m) words



Advantage

- Compact for sparse
- Easily see all edges

Disadvantage

- No O(1) edge test
- More complex data structure

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; many efficient algs build on it.

Applications:

- Finding Connected components of a graph
- Testing Bipartiteness
- Finding Aritculation points

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

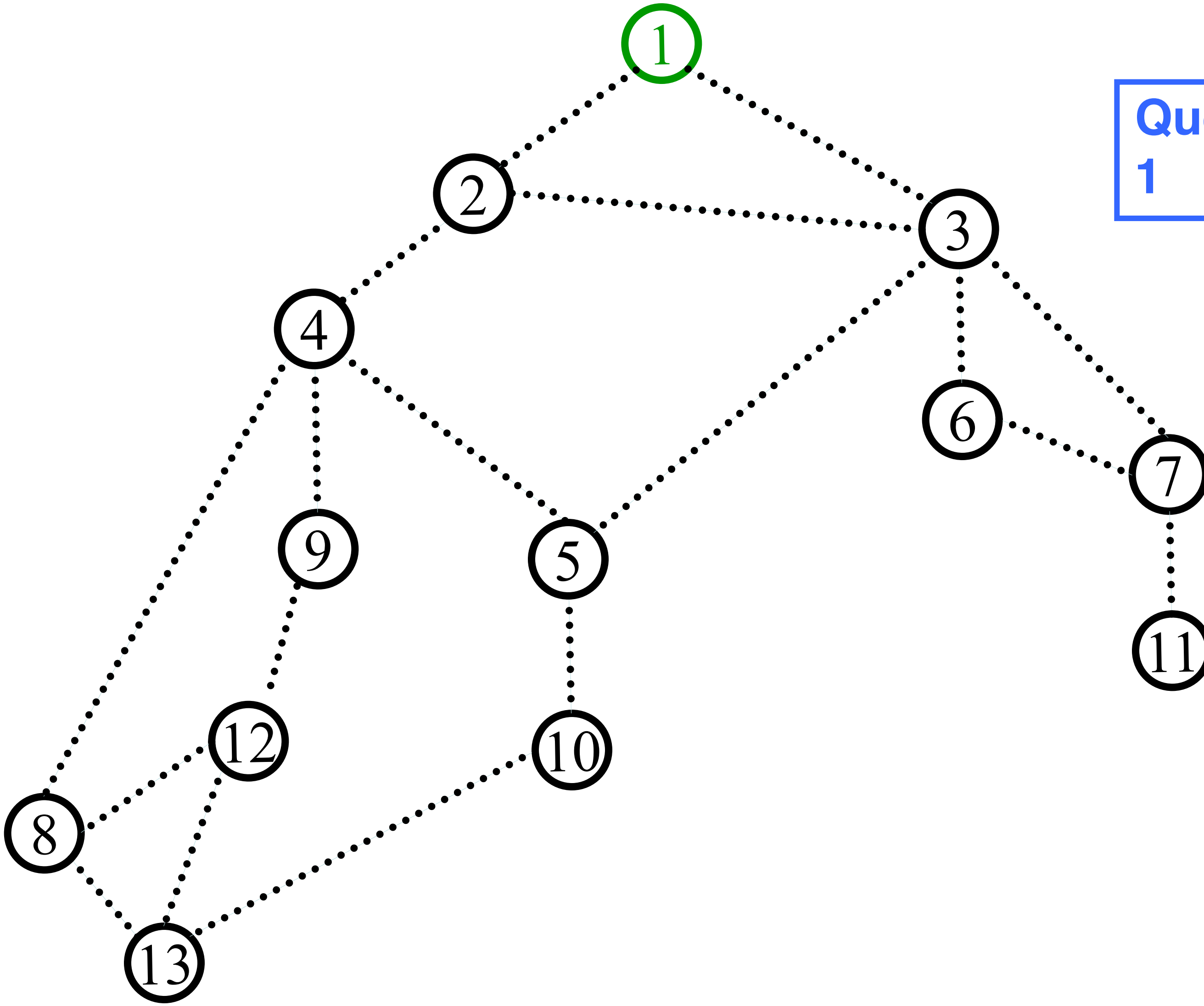
The queue will always have the list of Discovered vertices

BFS implementation

Global initialization: mark all vertices "undiscovered"

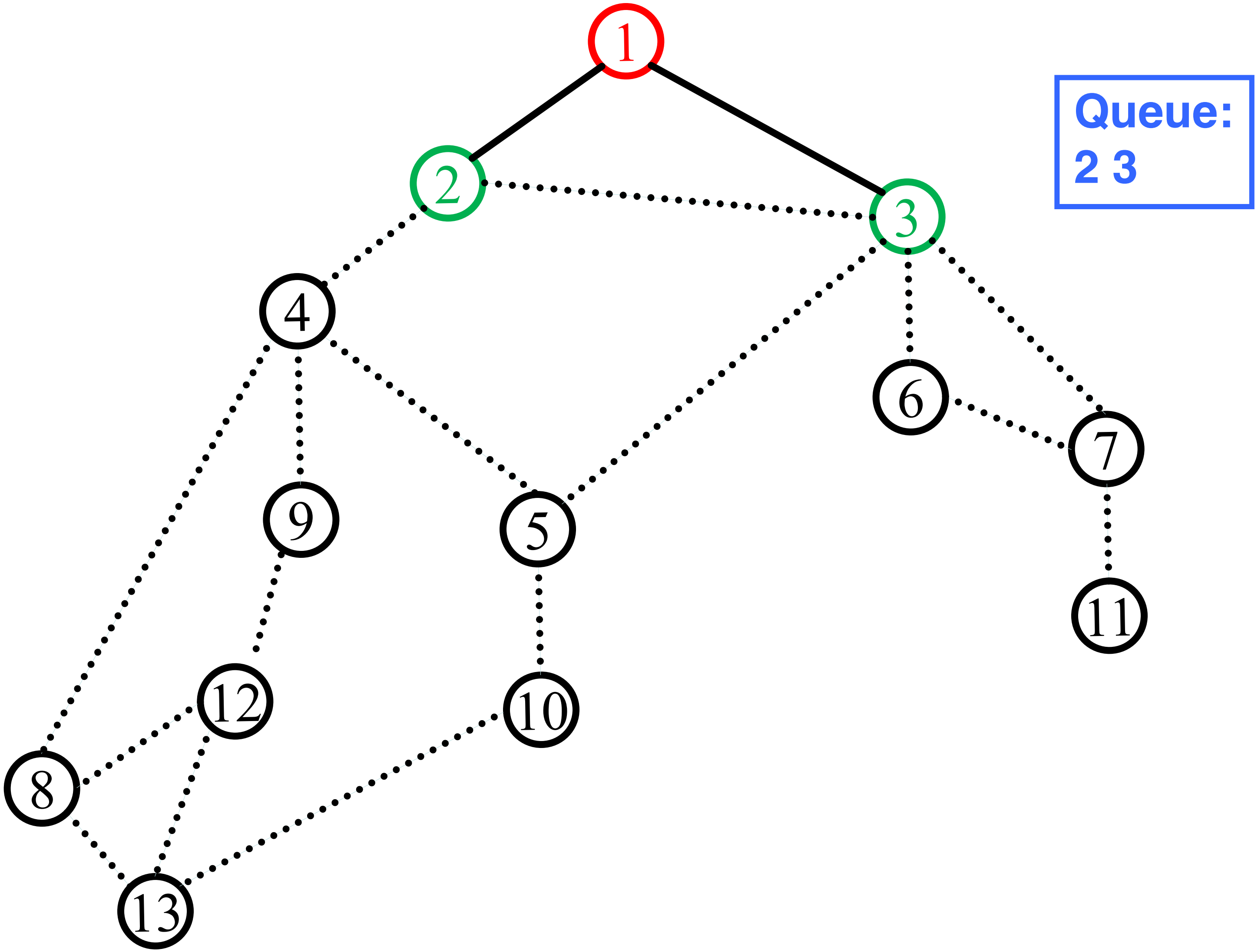
```
BFS(s)
    mark  s "discovered"
    queue = { s }
    while queue not empty
        u = remove_first(queue)
        for each edge {u,x}
            if (x is undiscovered)
                mark x discovered
                append x on queue
        mark u fully-explored
```

BFS(1)

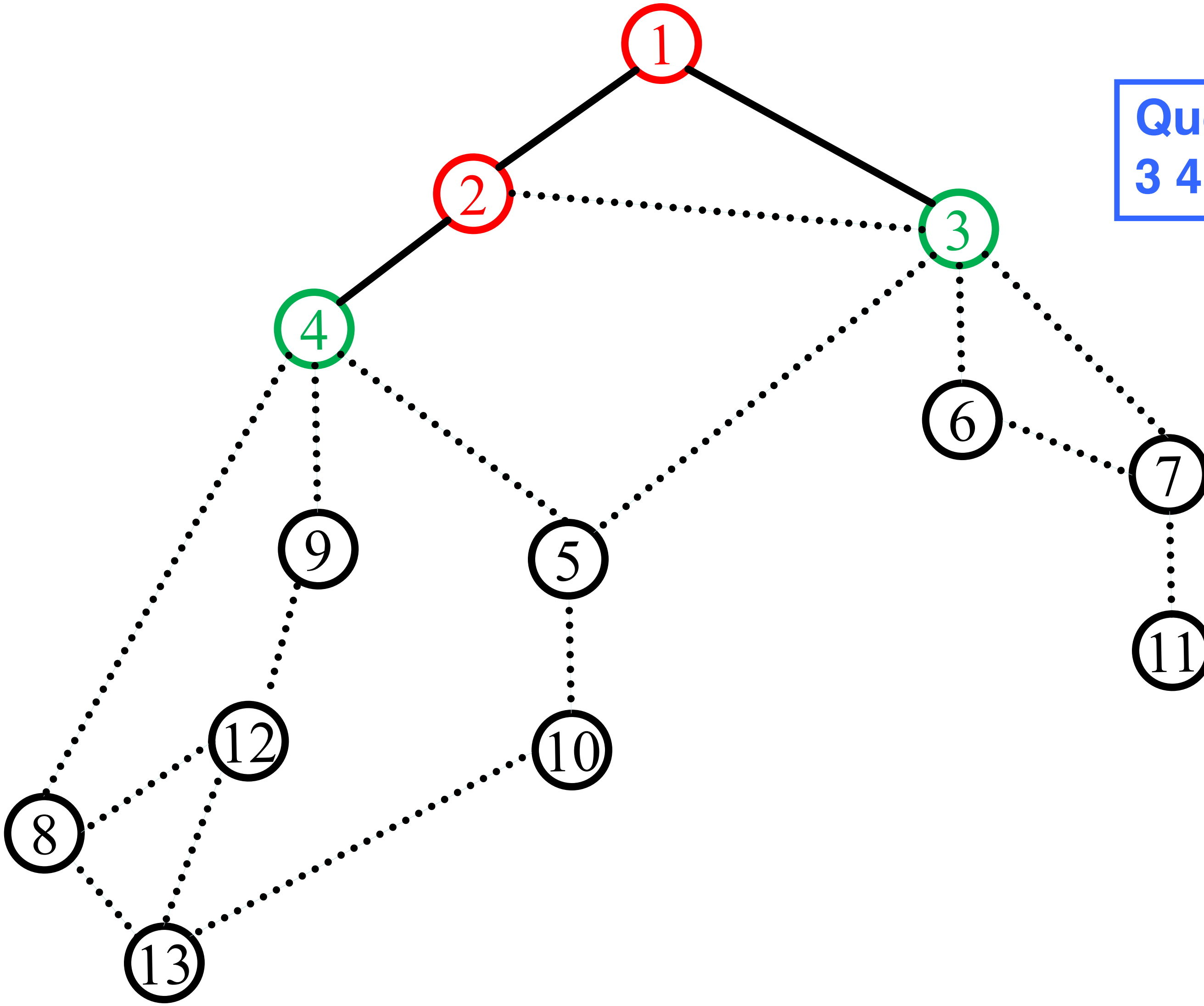


Queue:
1

BFS(1)

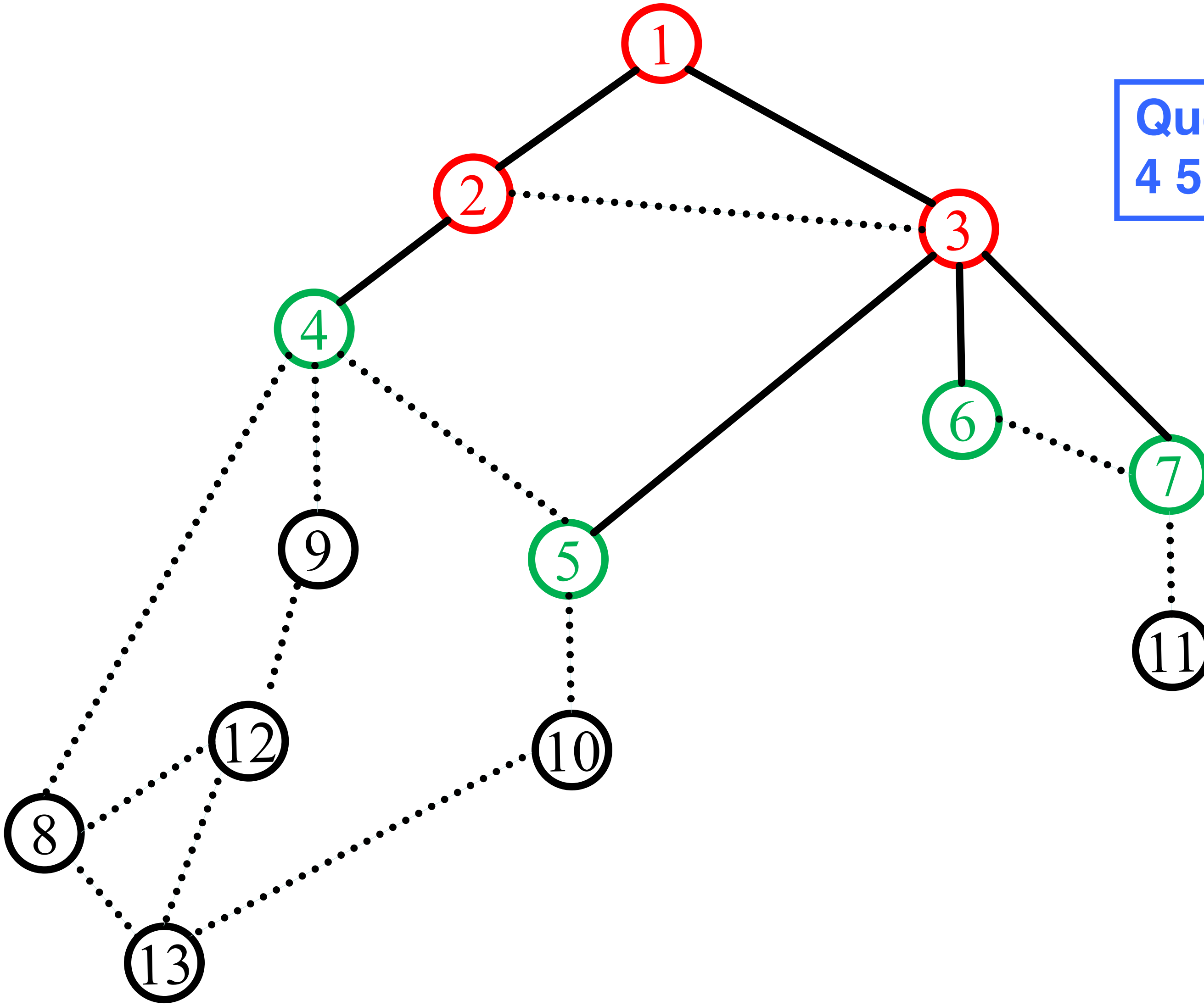


BFS(1)



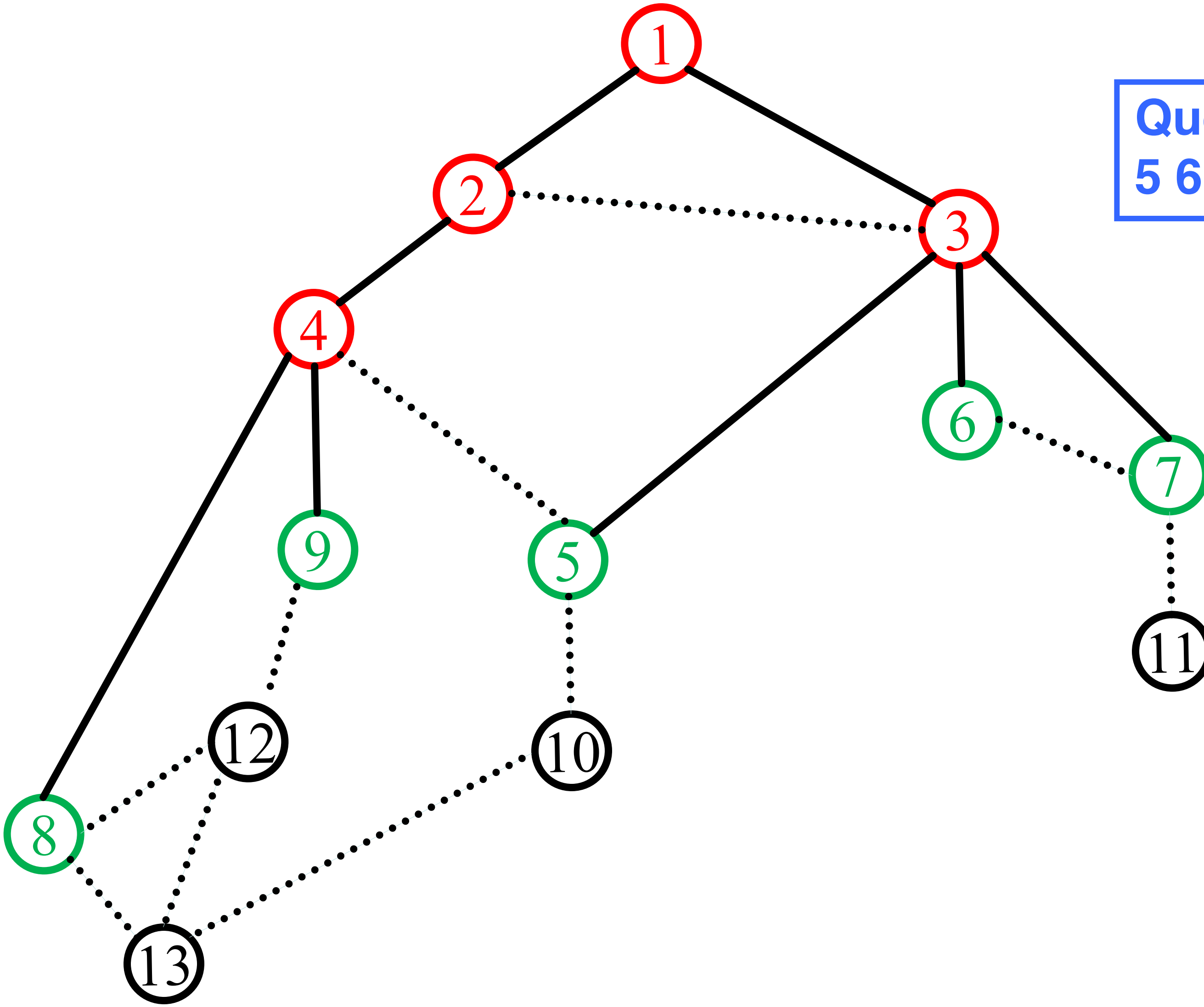
Queue:
3 4

BFS(1)



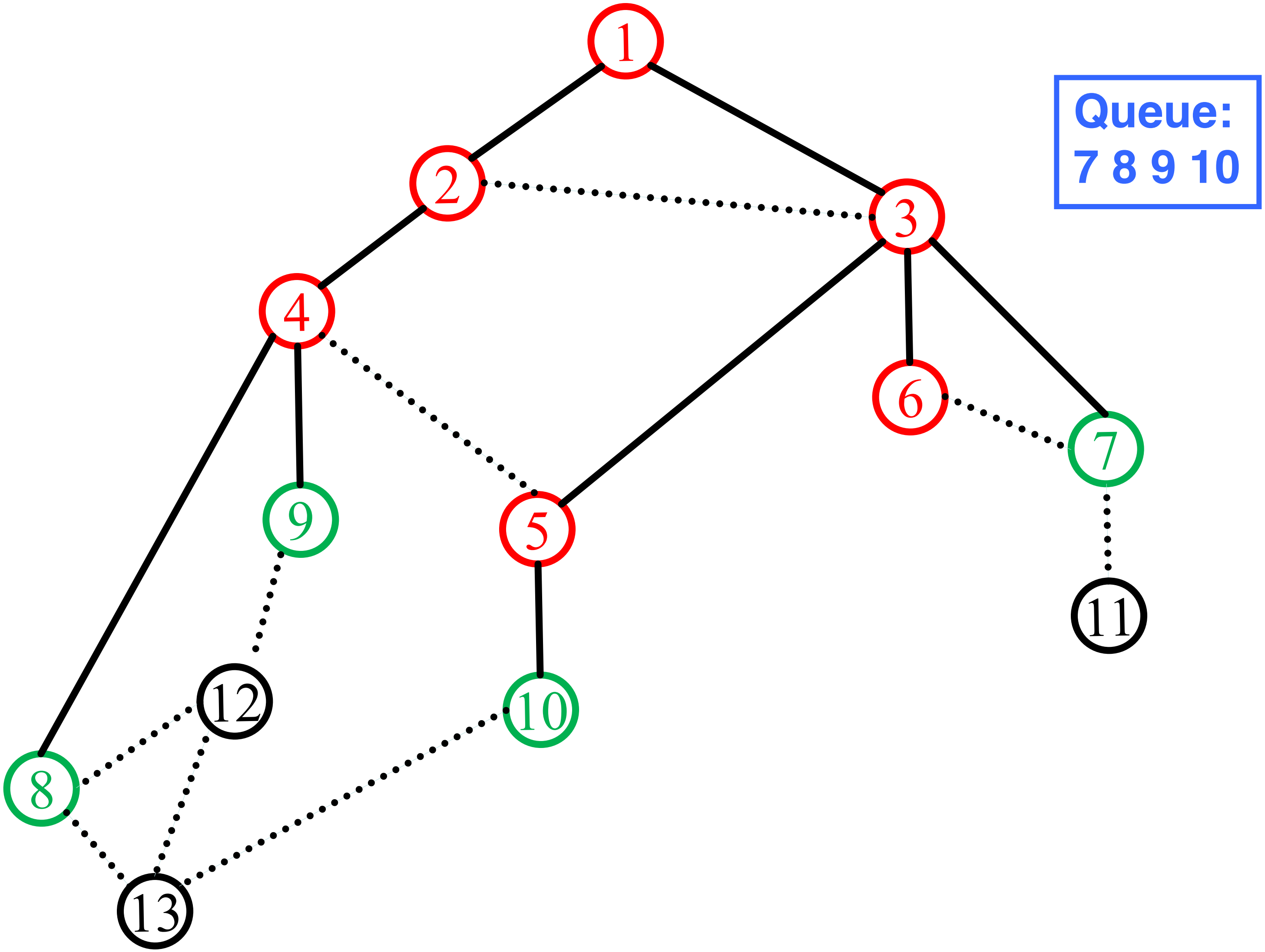
Queue:
4 5 6 7

BFS(1)

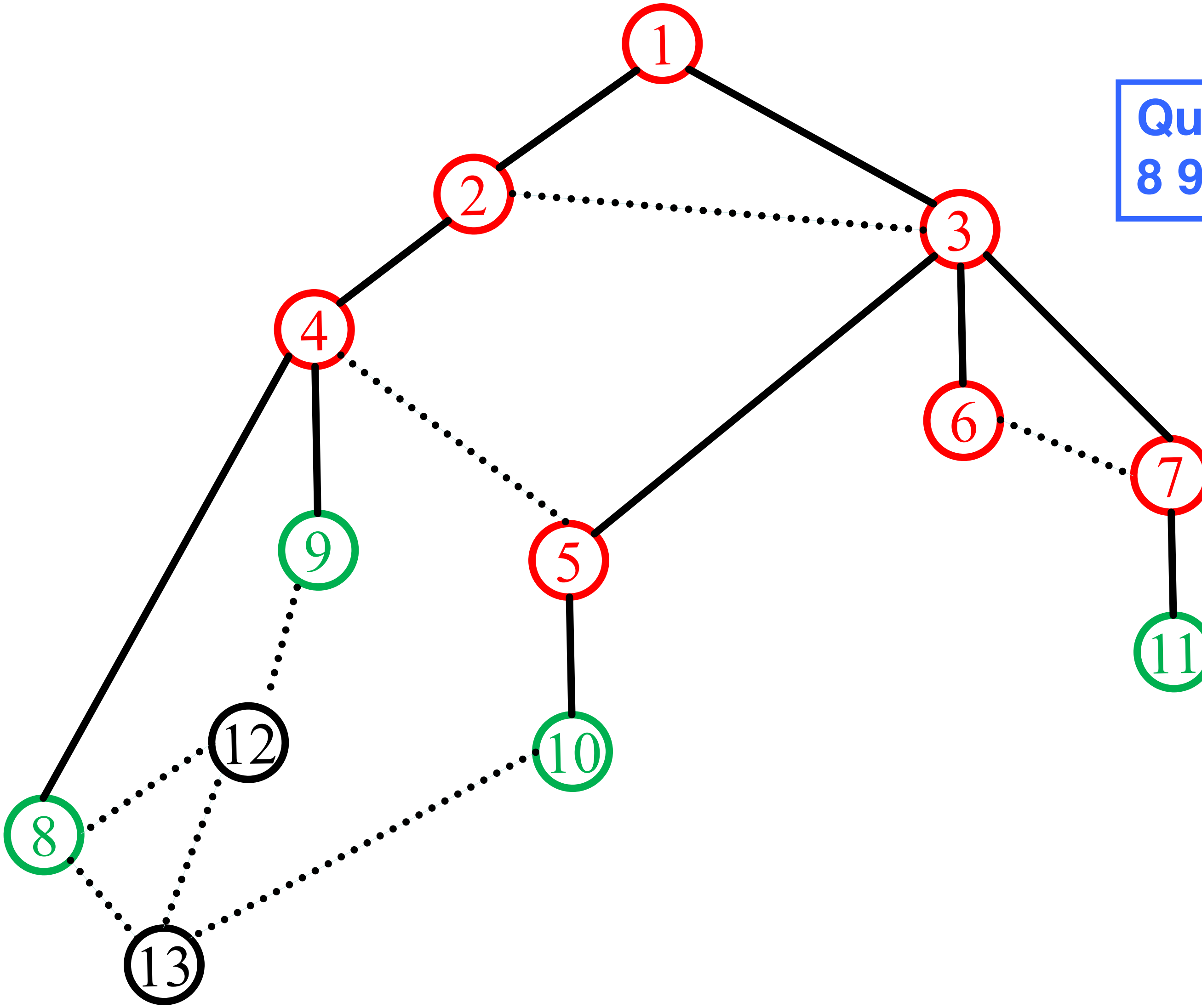


Queue:
5 6 7 8 9

BFS(1)

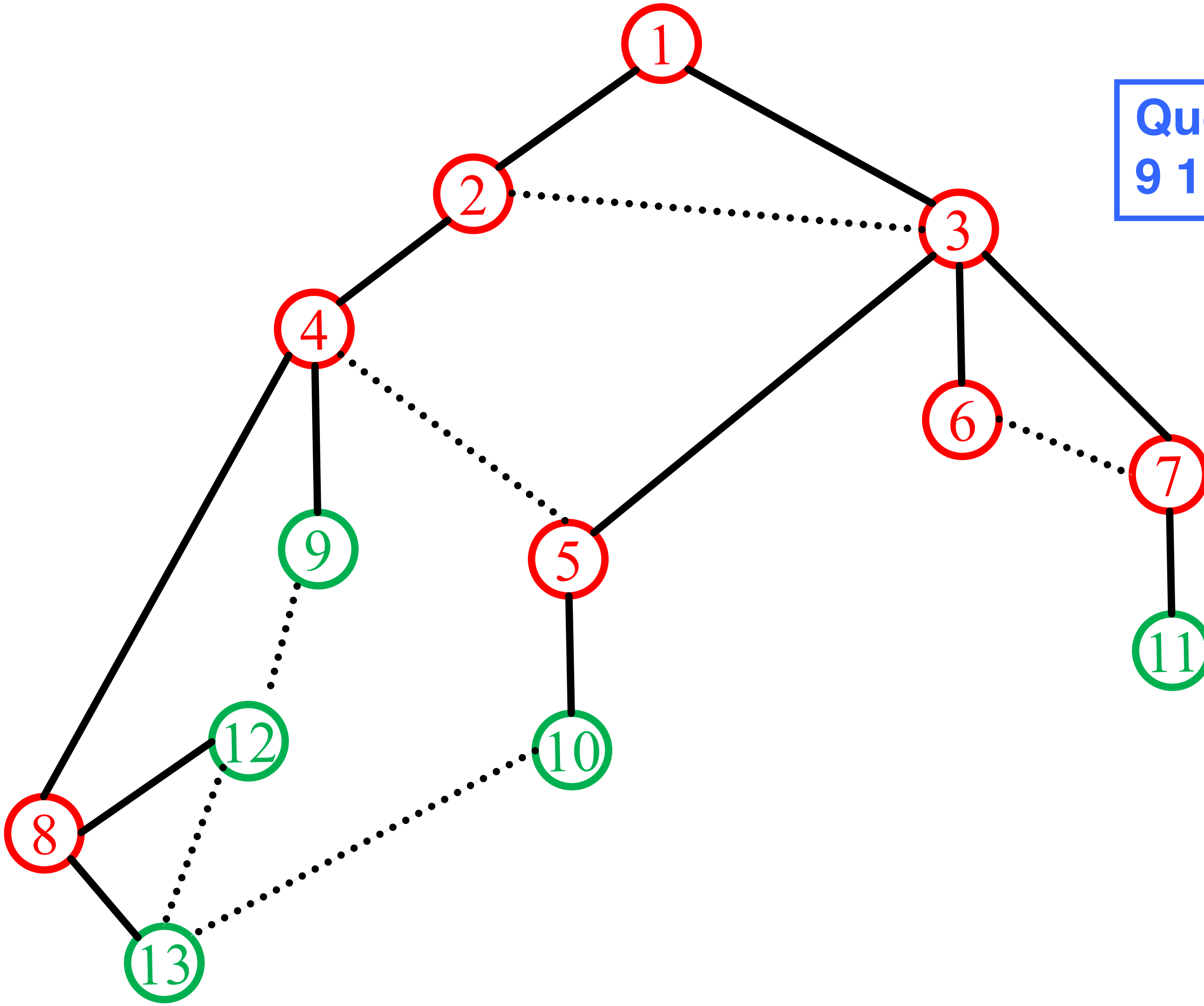


BFS(1)



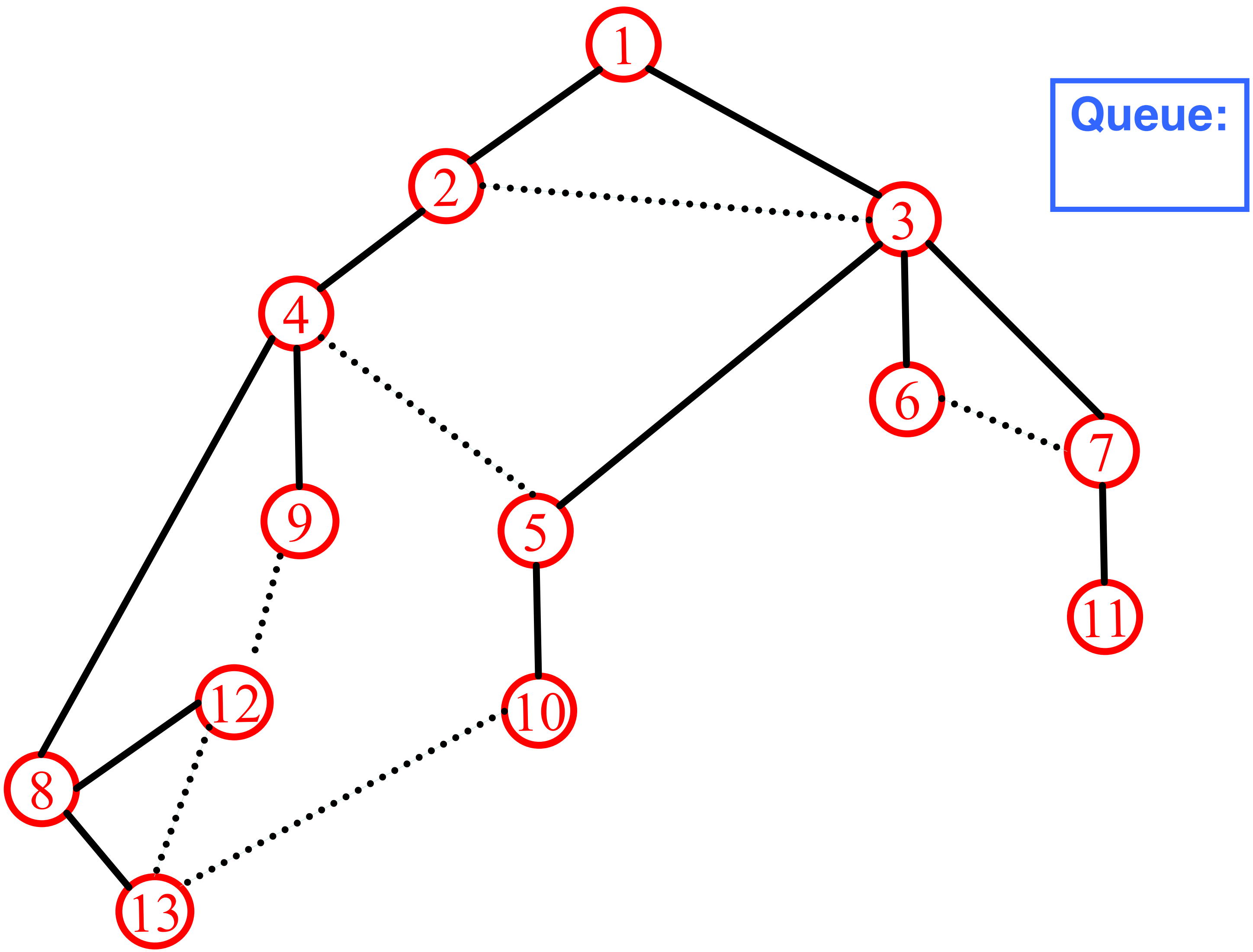
Queue:
8 9 10 11

BFS(1)



Queue:
9 10 11 12 13

BFS(1)



BFS Analysis

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
```

O(n) times: Once from every vertex if G is connected

$\mathbf{deg}(\boldsymbol{u}) \leq \boldsymbol{O}(\boldsymbol{n})$ **times**

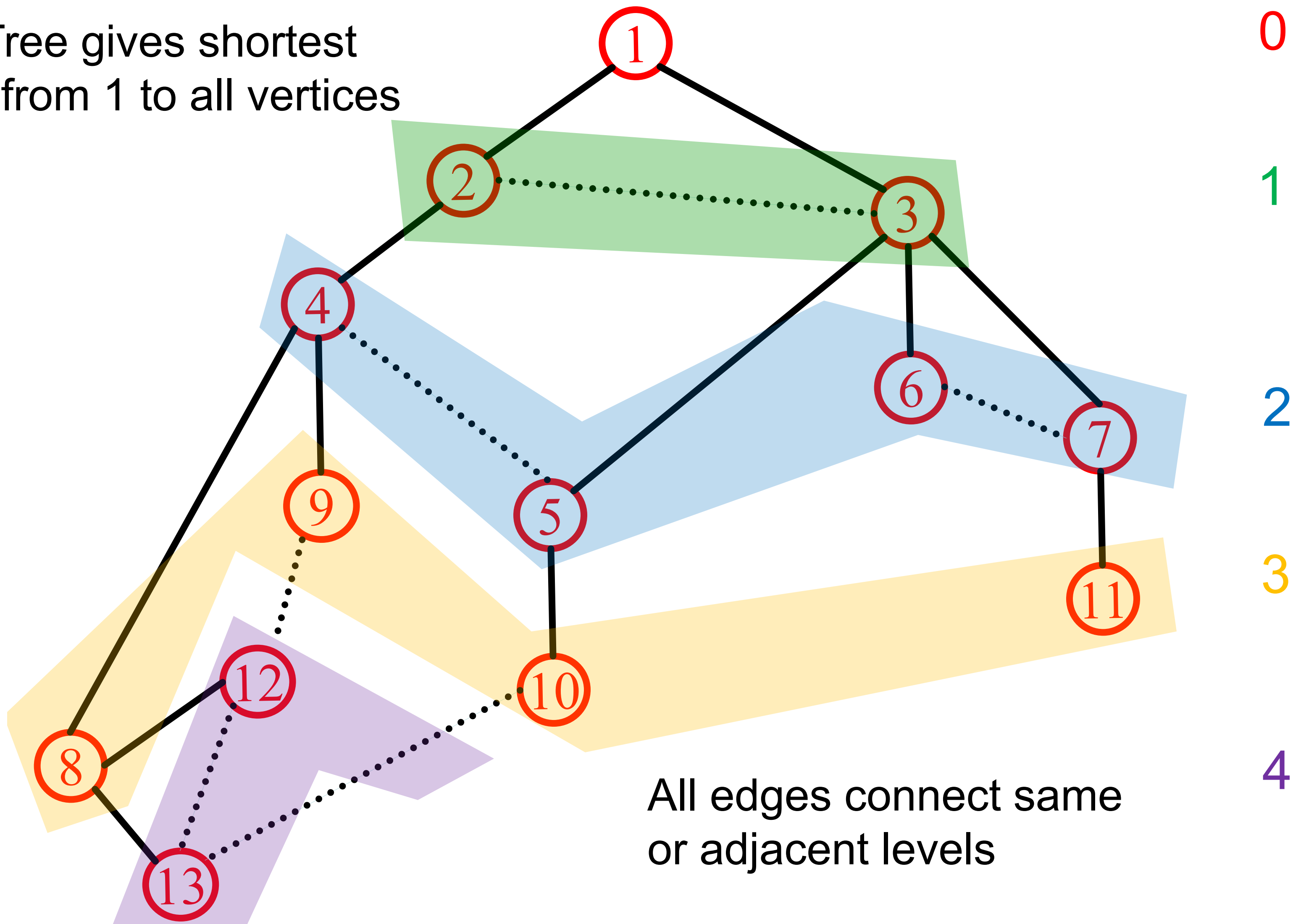
If we use adjacency list: $O(n) + O(\sum_v \deg(v)) = O(m+n)$

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

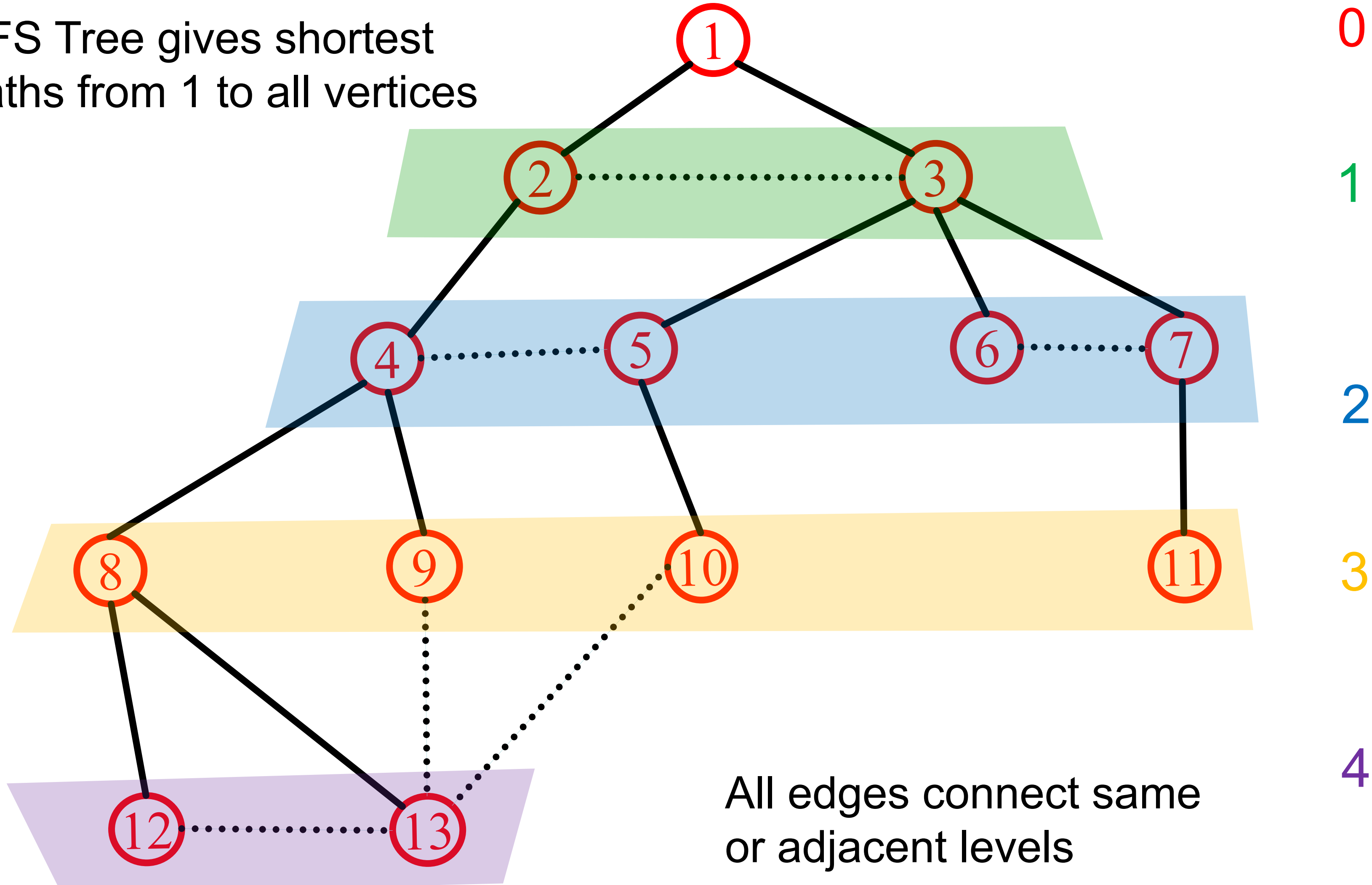
BFS Tree gives shortest paths from 1 to all vertices



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

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