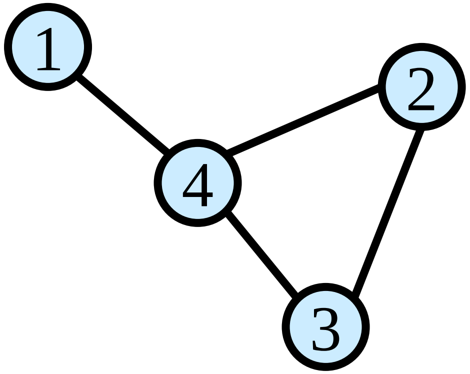
CSE 421: Introduction to Algorithms

Trees/Graph Traversal

Shayan Oveis Gharan

Storing Graphs (Internally in ALG)



Adjacency List:

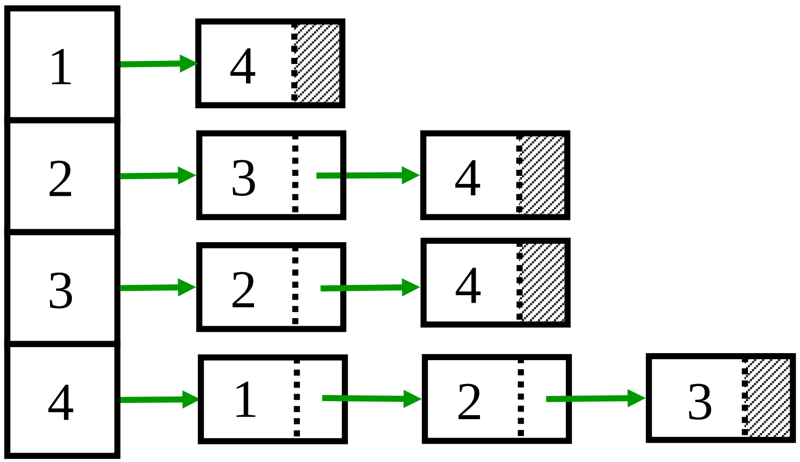
O(n+m) words

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:

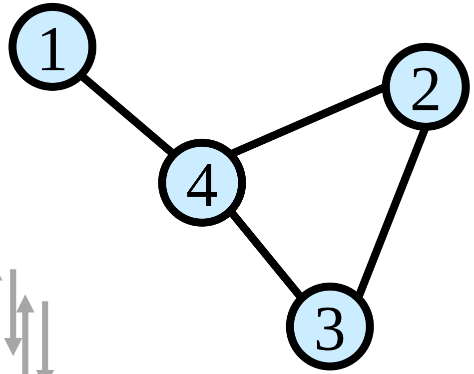
O(n+m) words

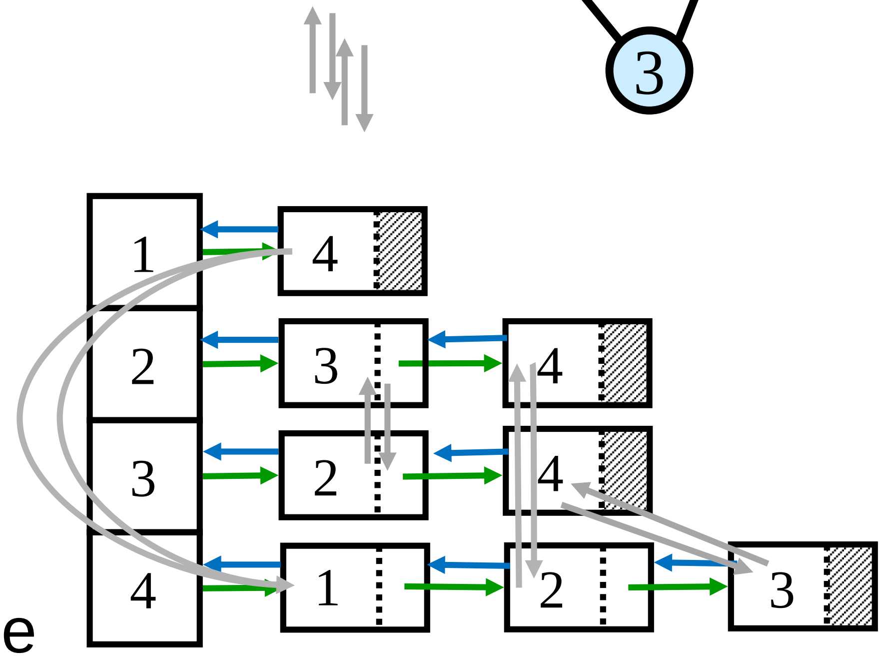
Advantage

- Compact for sparse
- Easily see all edges

Disadvantage

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





Degree 1 vertices

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

(So, every tree has a leaf)

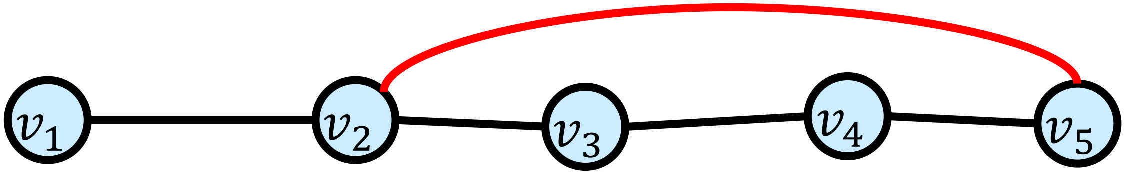
Pf: (By contradiction)

Suppose every vertex has degree ≥ 2.

Start from a vertex v_1 and follow a path, $v_1, \ldots, v_i$ 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.

Pf: By induction.

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

IH: Suppose every tree with n-1 vertices has n-2 edges

IS: 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; 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

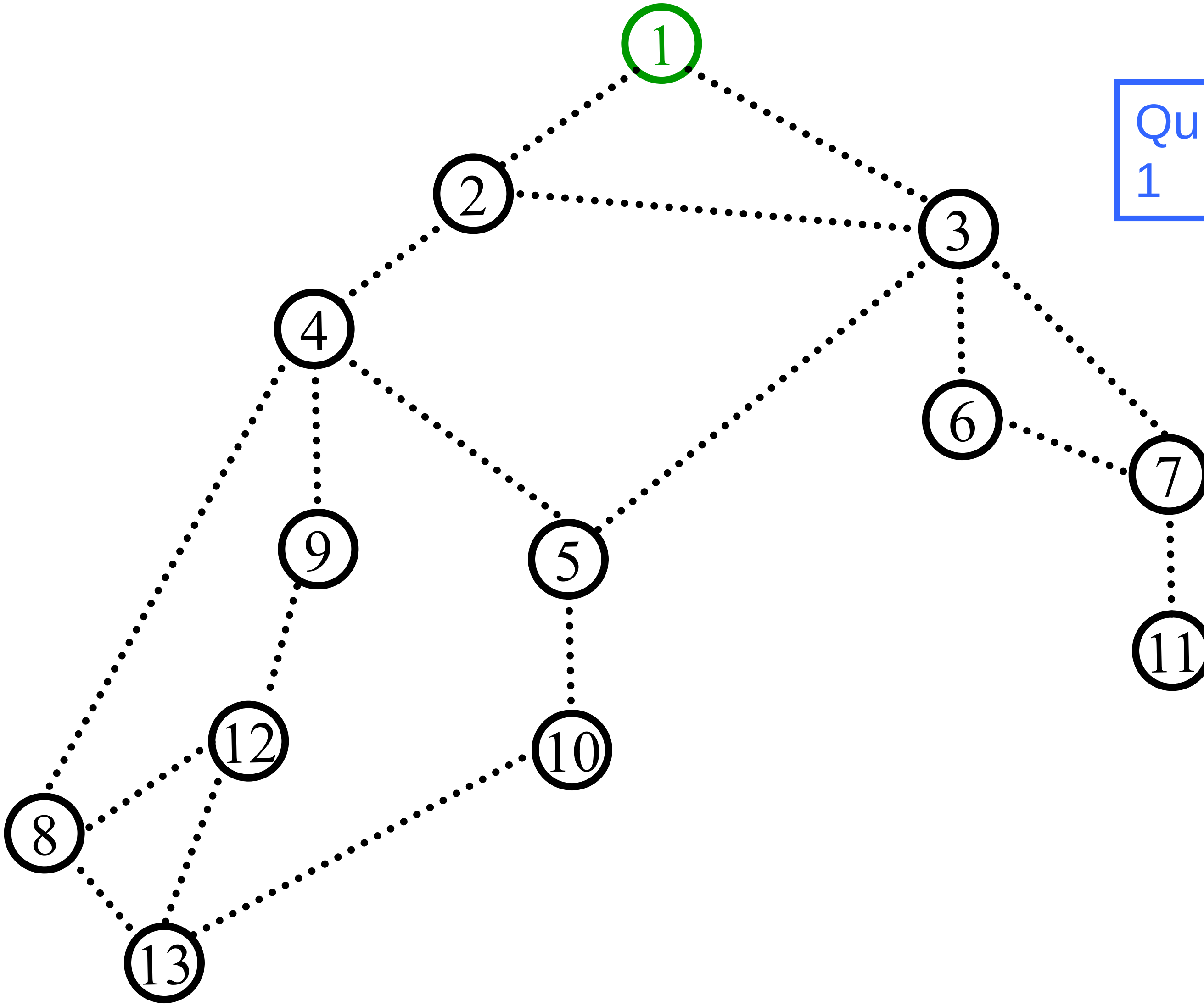
The queue will always have the list of Discovered vertices

BFS implementation

Global initialization: mark all vertices "undiscovered"

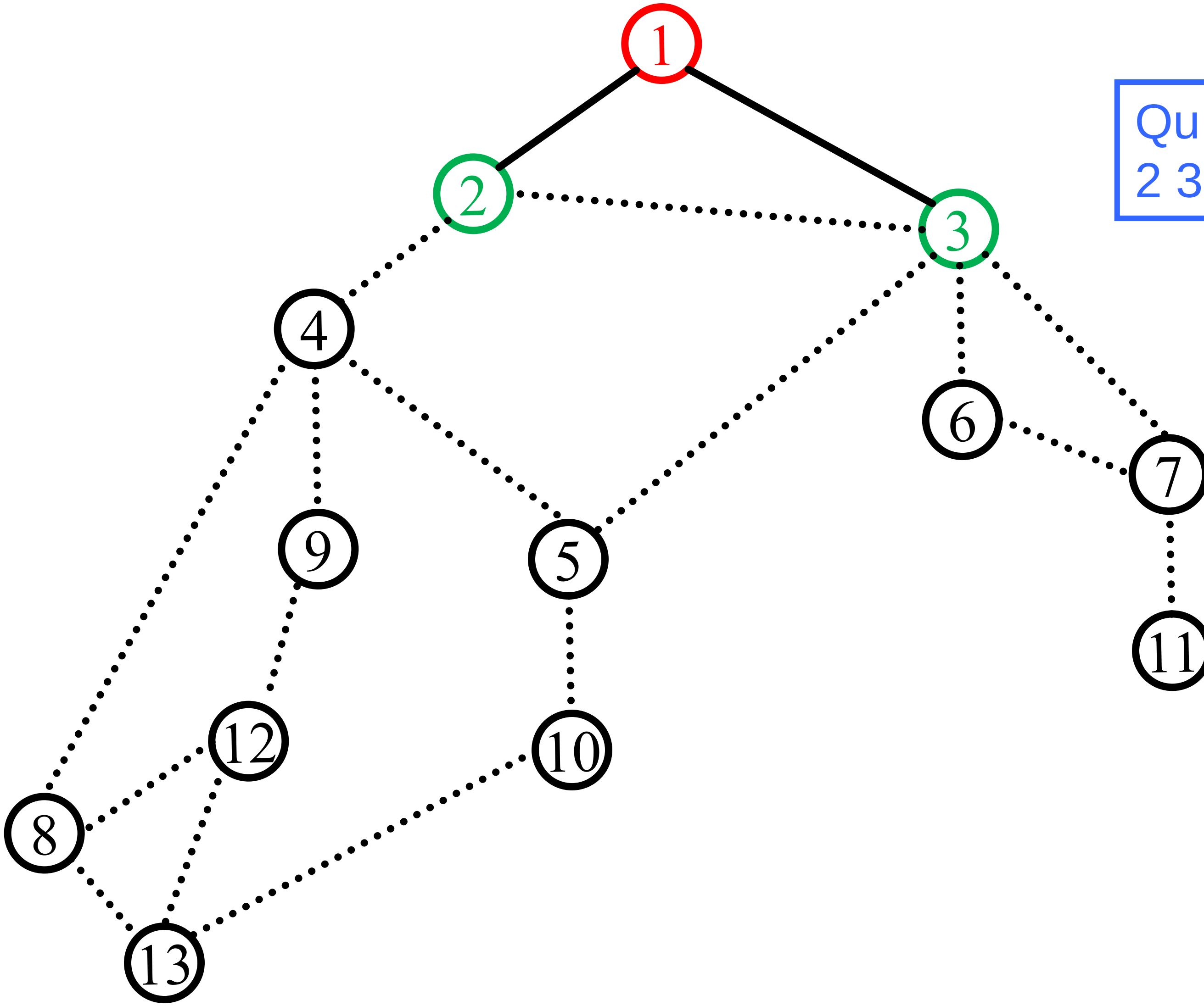
```
BFS(s)
    mark  s "discovered"
    queue = { s }
    while queue not empty
        u = remove_first(queue)
        for each edge {u,x}
            if (x is undiscovered)
                mark x discovered
                append x on queue
        mark u fully-explored
```

BFS(1)



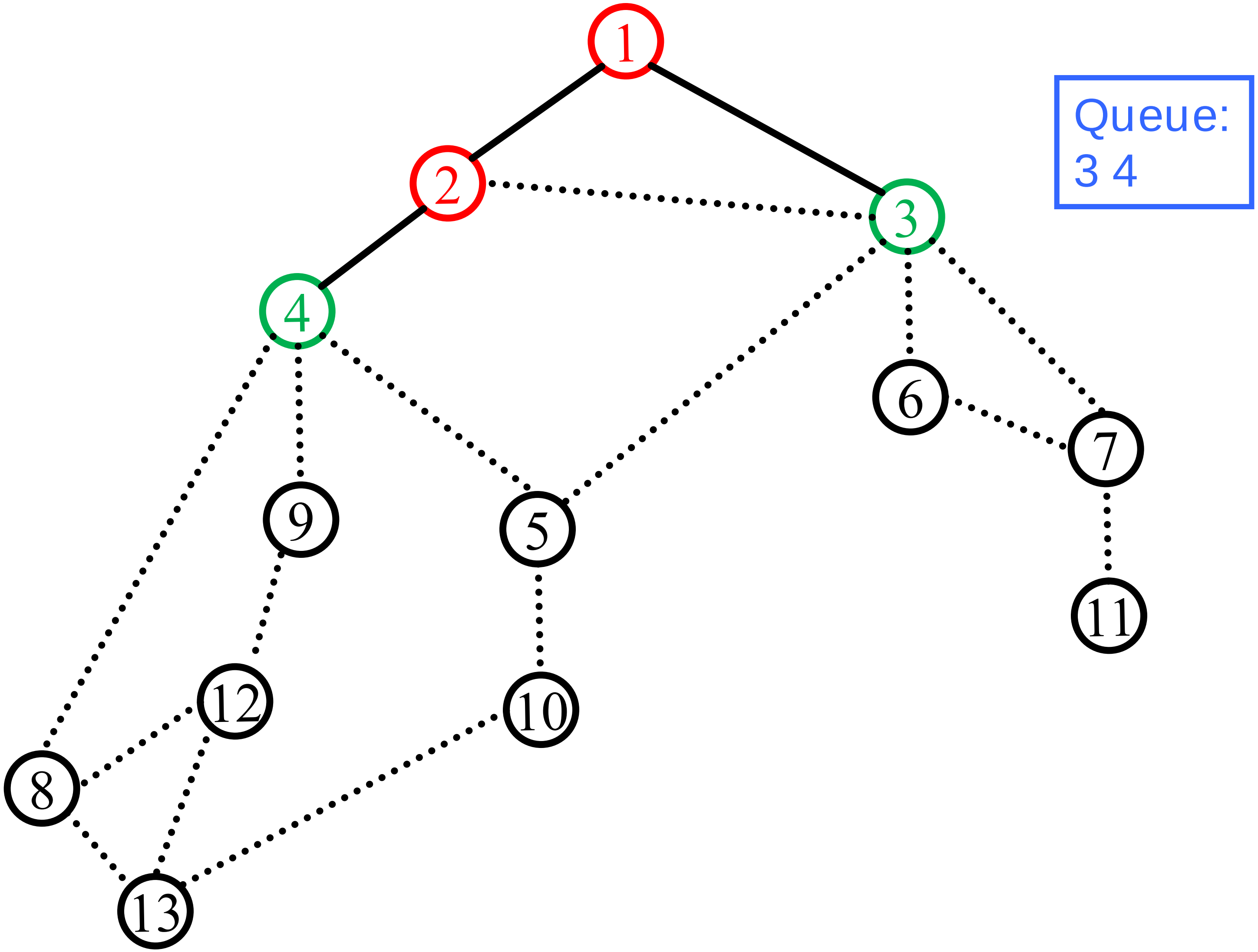
Queue:
1

BFS(1)

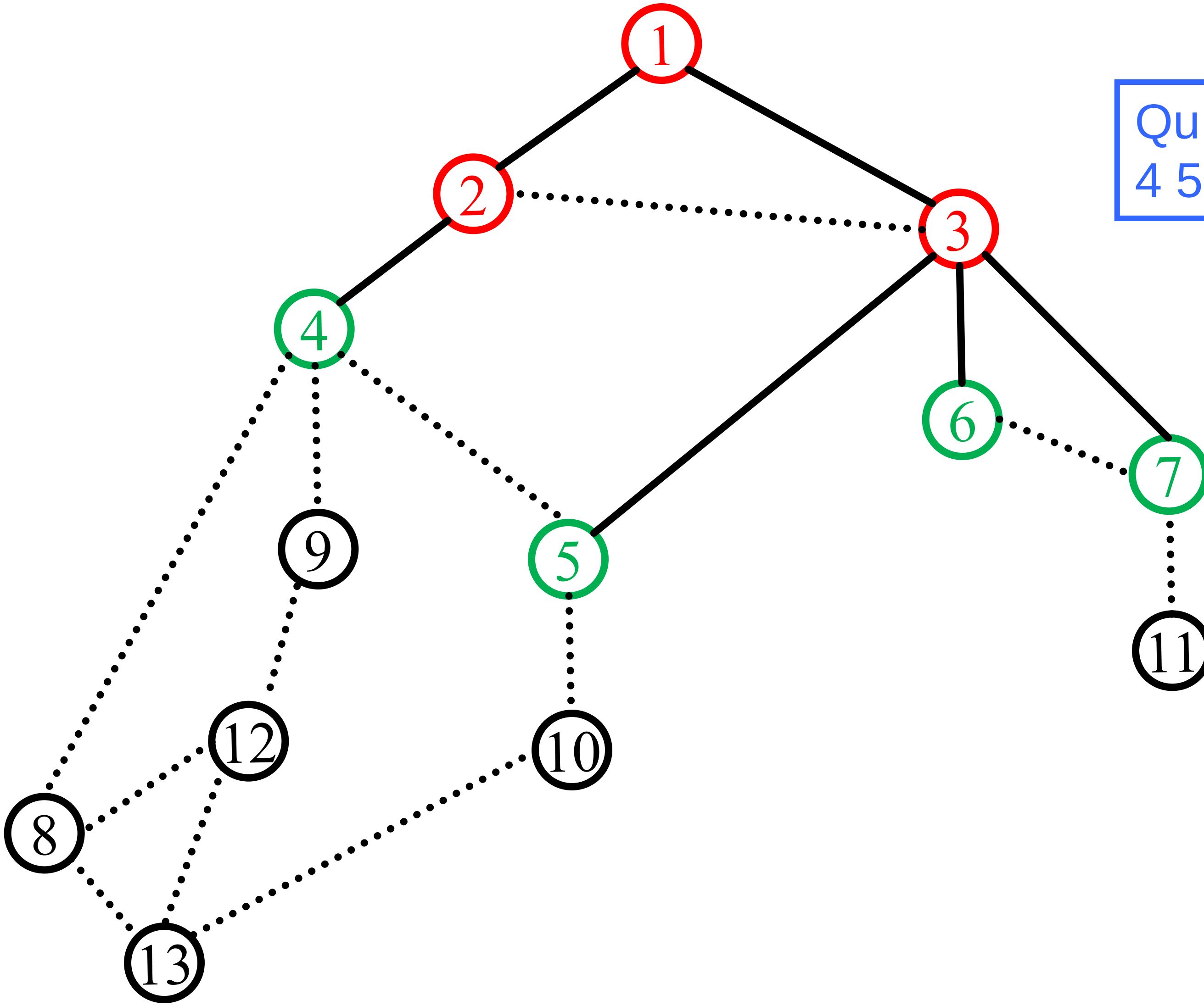


Queue:
2 3

BFS(1)

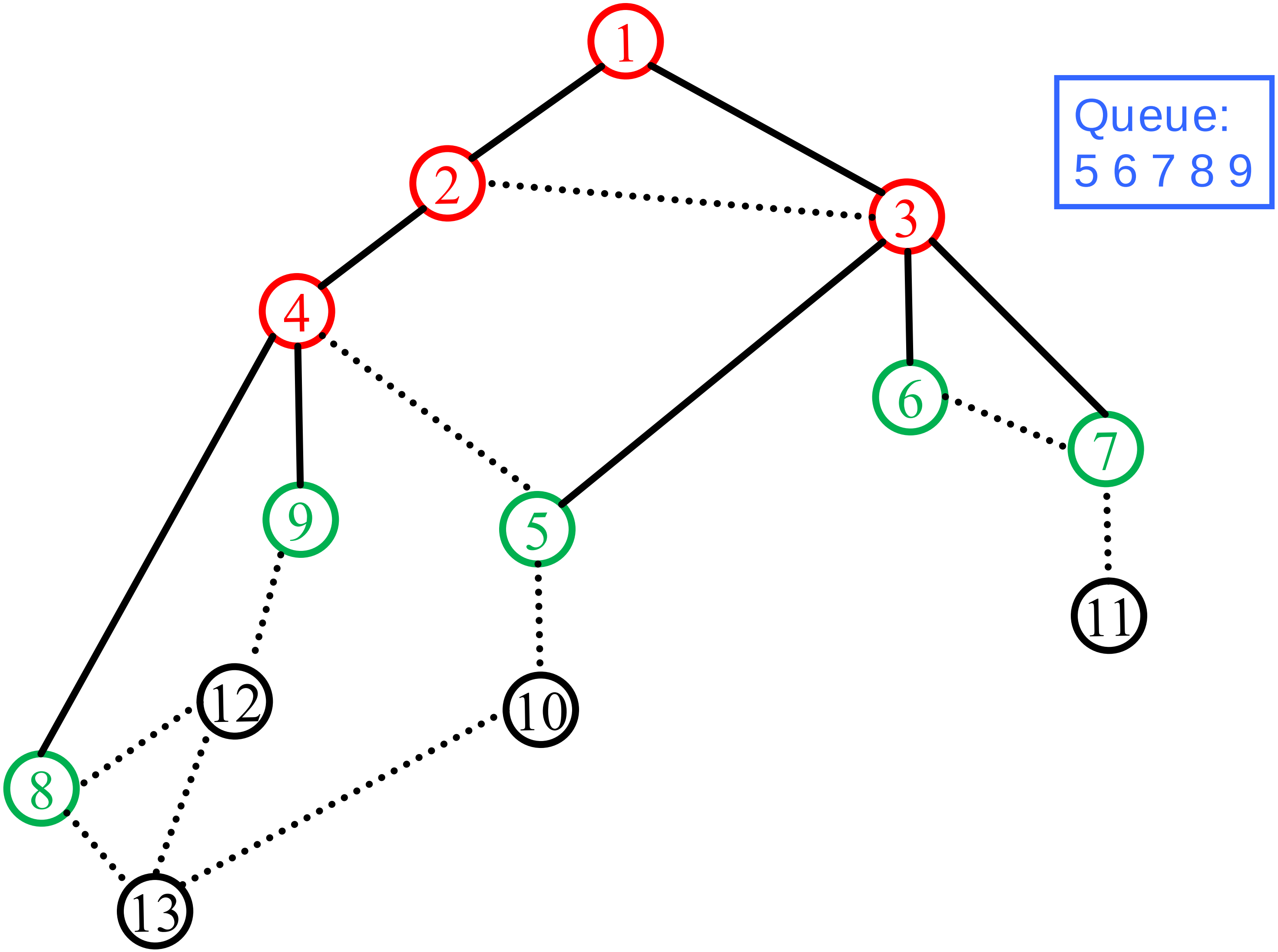


BFS(1)

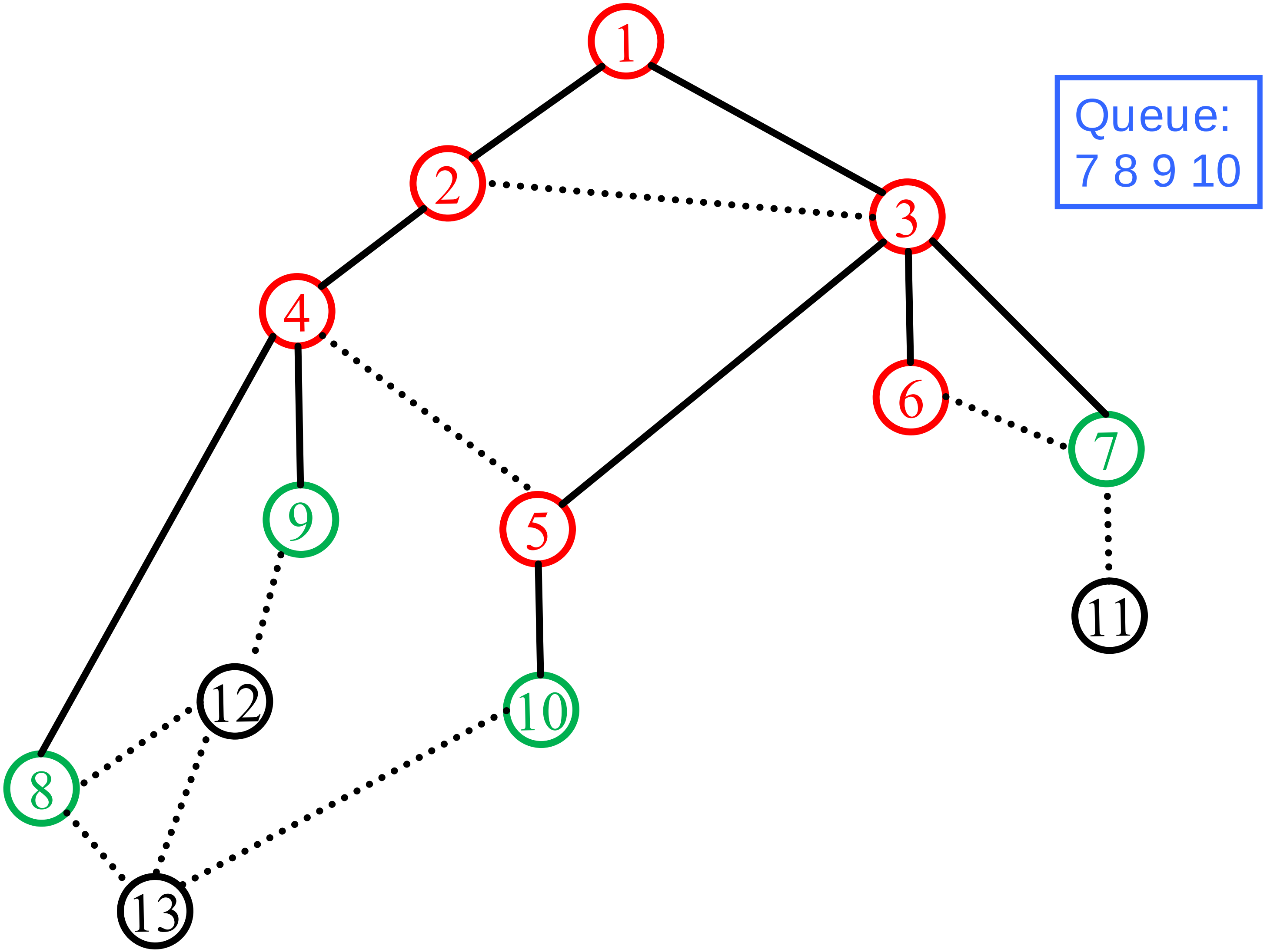


Queue:
4 5 6 7

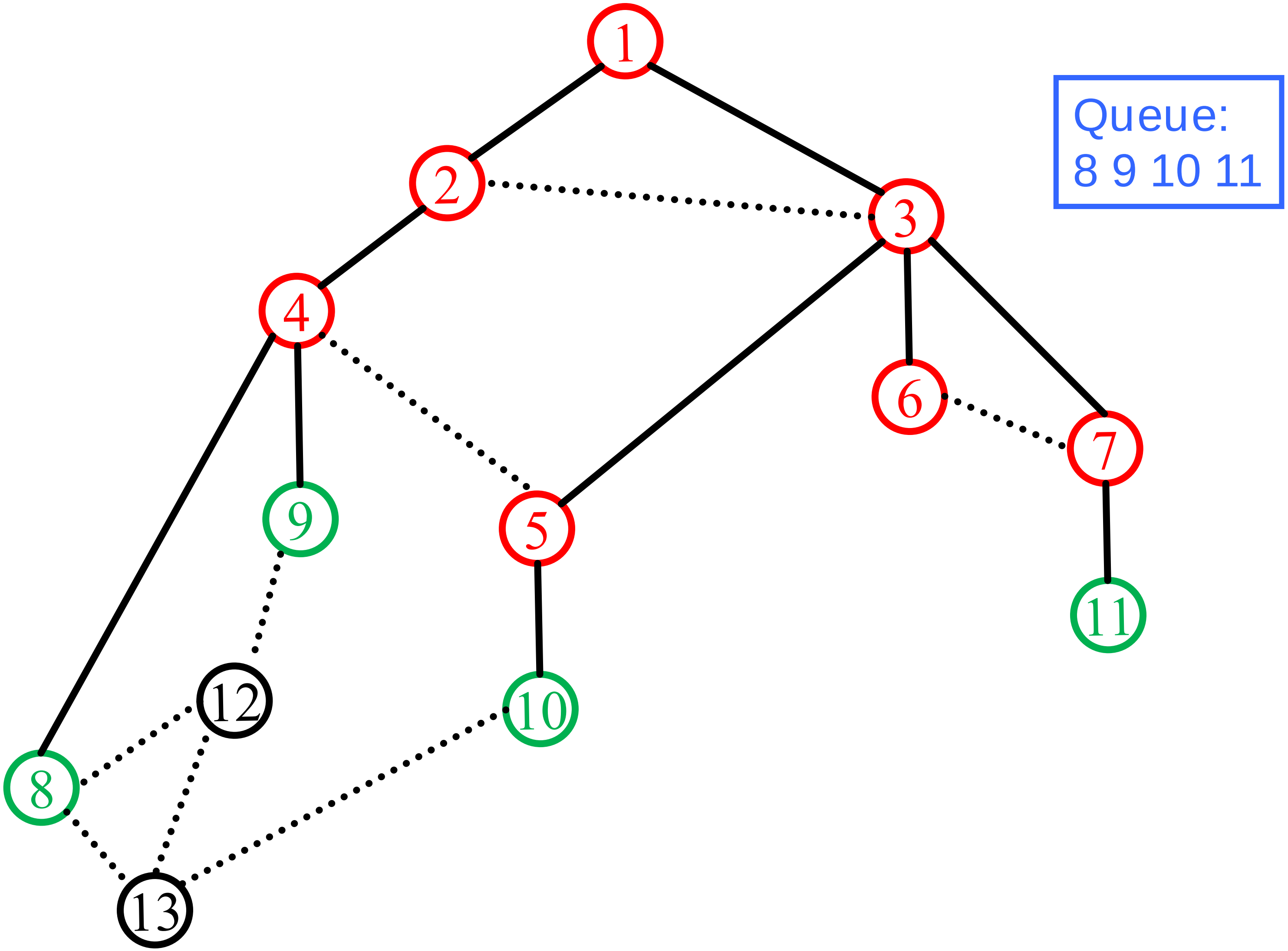
BFS(1)



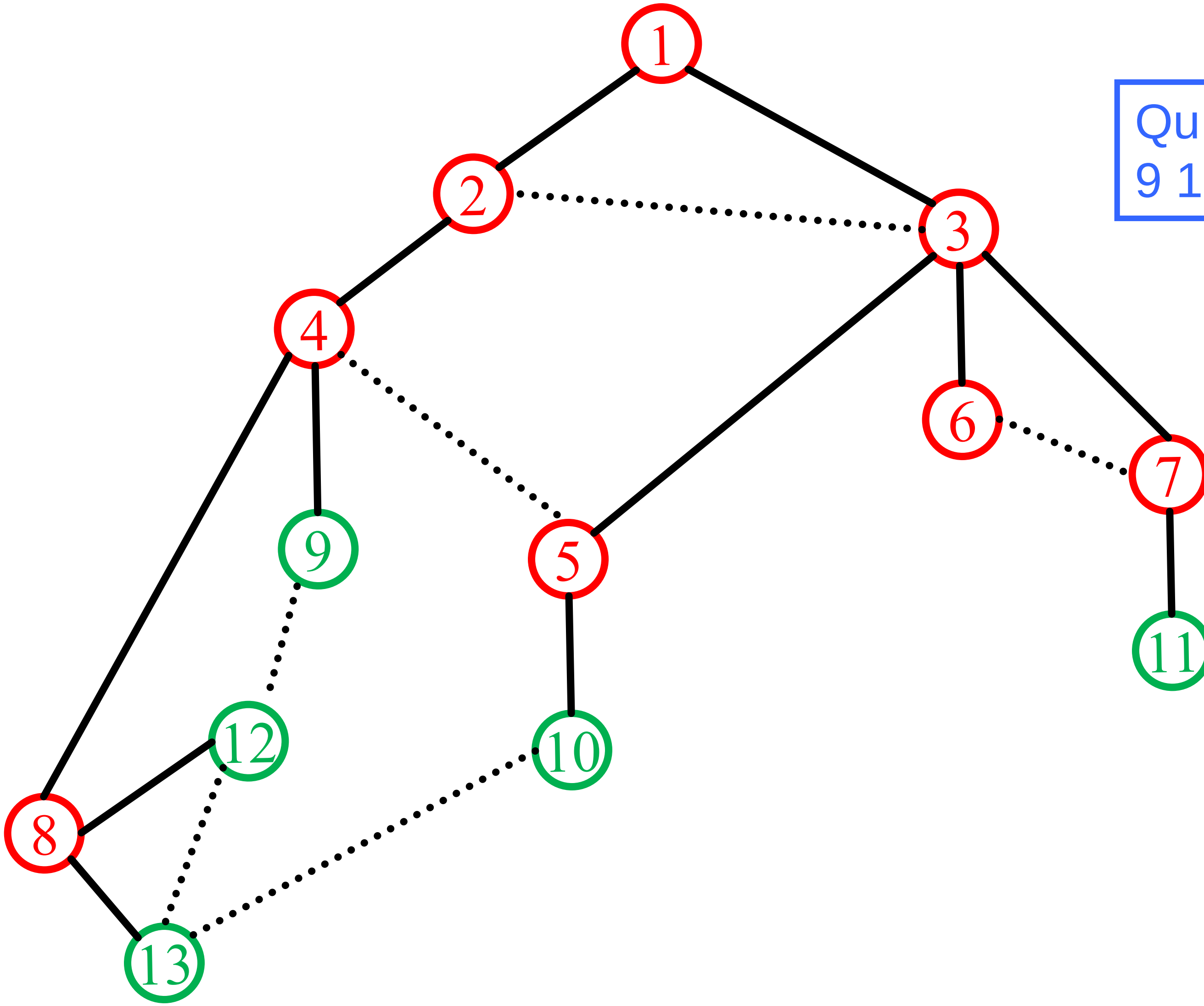
BFS(1)



BFS(1)

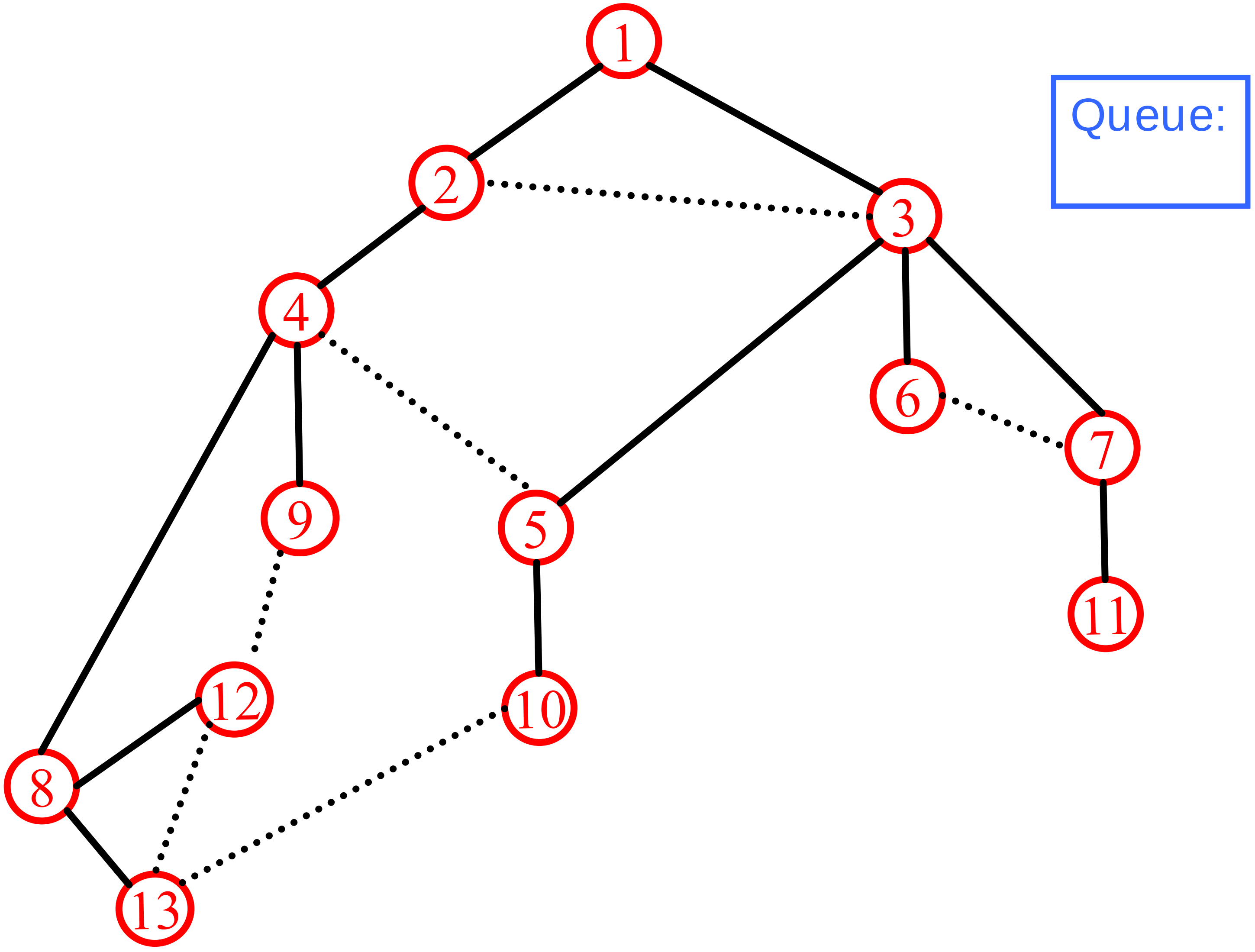


BFS(1)



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**

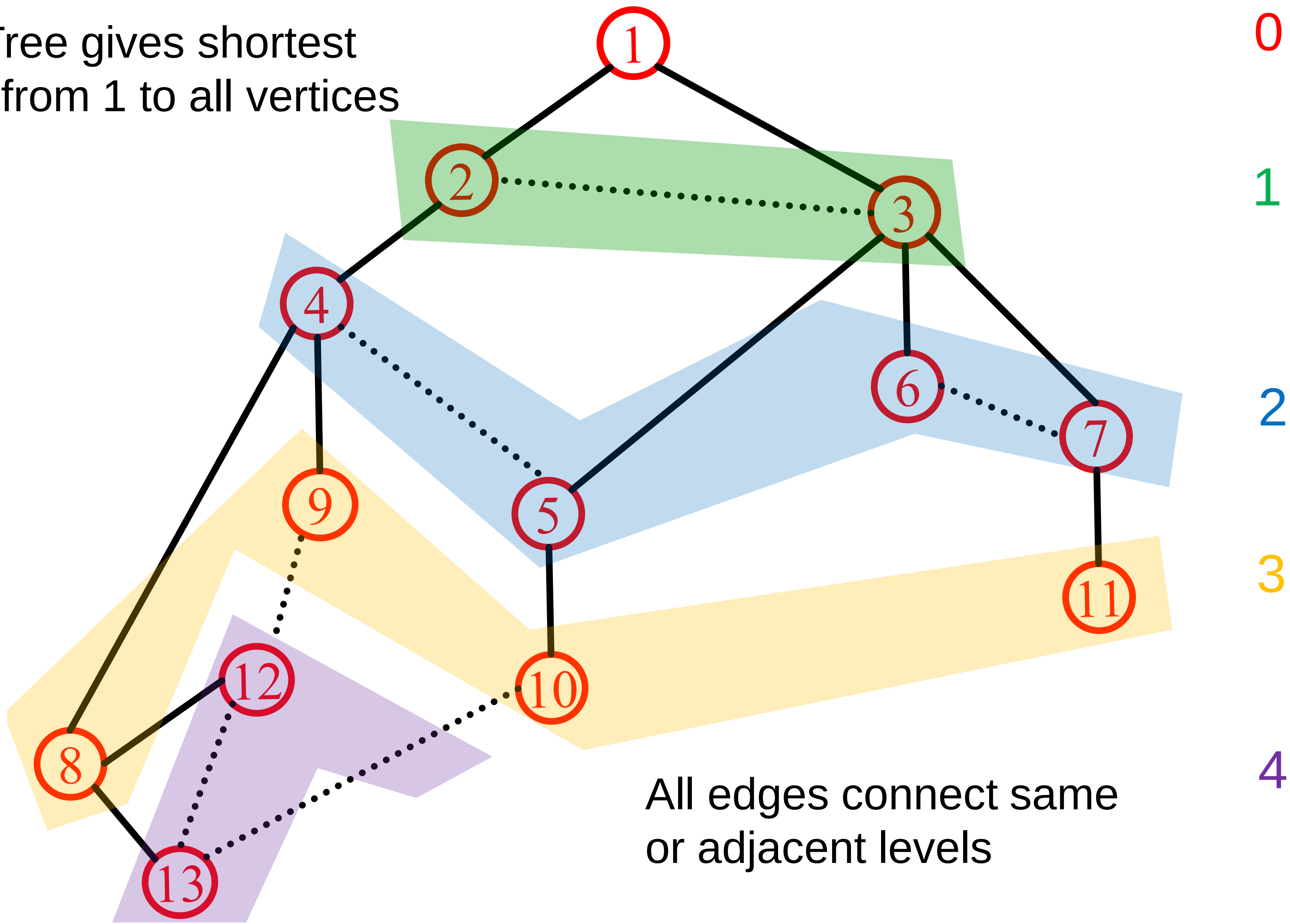
If we use adjacency list: $O(n) + O(\sum_v \deg(v)) = O(m+n)$

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

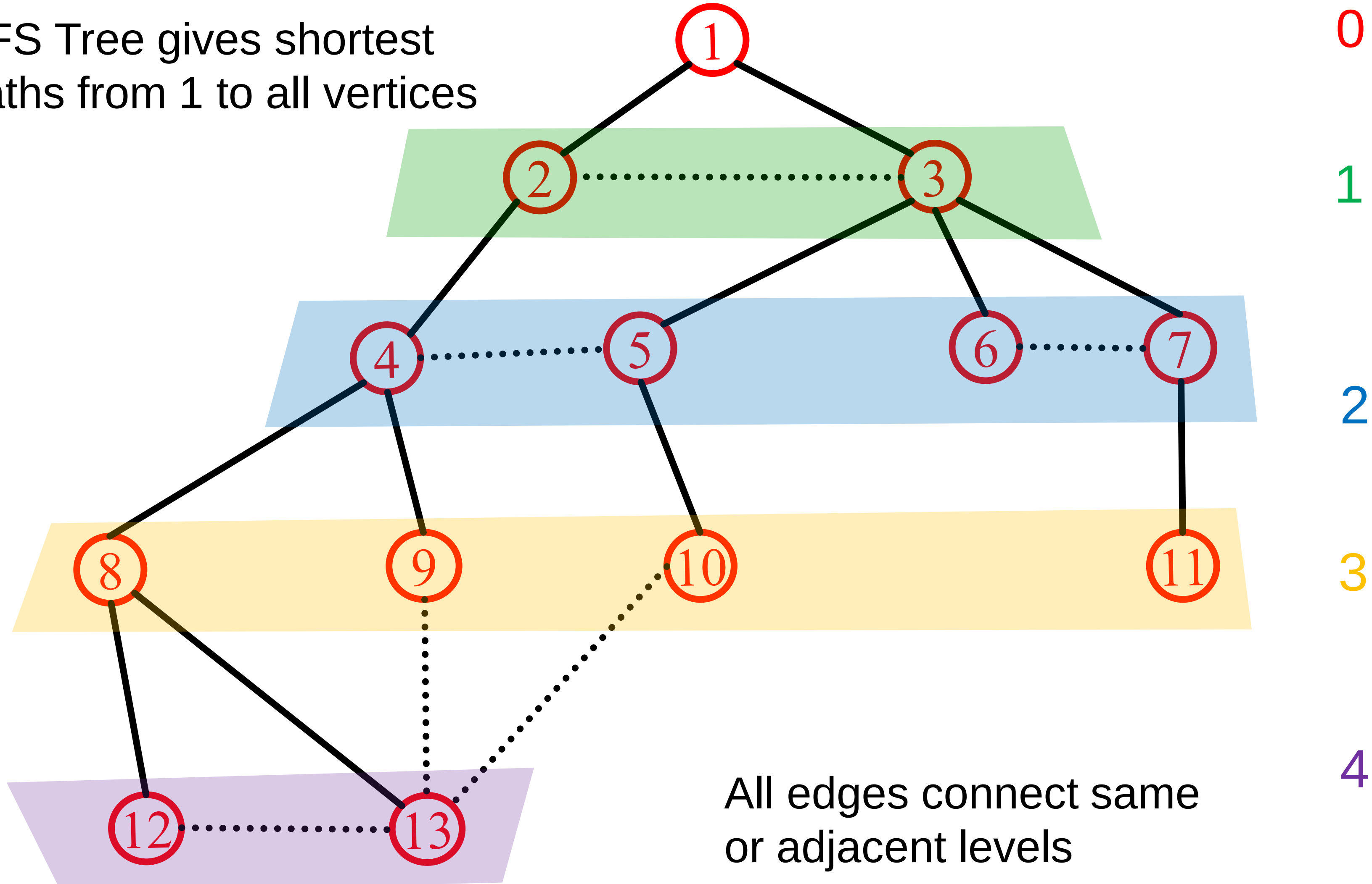
BFS Tree gives shortest paths from 1 to all vertices



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 addedto 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 $i + 1$.