CS 401: Computer Algorithm I

BFS

Xiaorui Sun

Homework 1

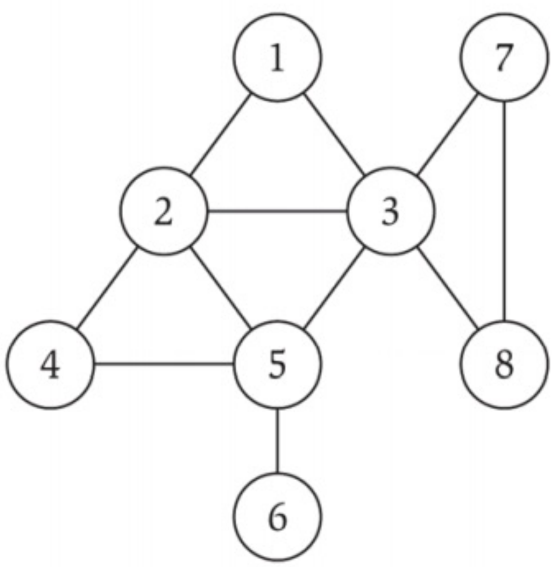
Homework 1 was out

- Deadline: February 9 11:59pm
- Submit your solution to gradescope
- More details can be found in the slides of the lecture on January 25

Undirected Graphs G=(V,E)

Notation. G = (V, E)

- V = nodes (or vertices)
- E = edges between pairs of nodes
- Captures pairwise relationship between objects
- Graph size parameters: n = |V|, m = |E|



V = {1, 2, 3, 4, 5 ,6, 7, 8}
E = {(1,2), (1,3), (2,3), (2,4), (2,5), (3,5), (3,7), (3,8), (4,5), (5,6), (7,8)}
m=11, n=8

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

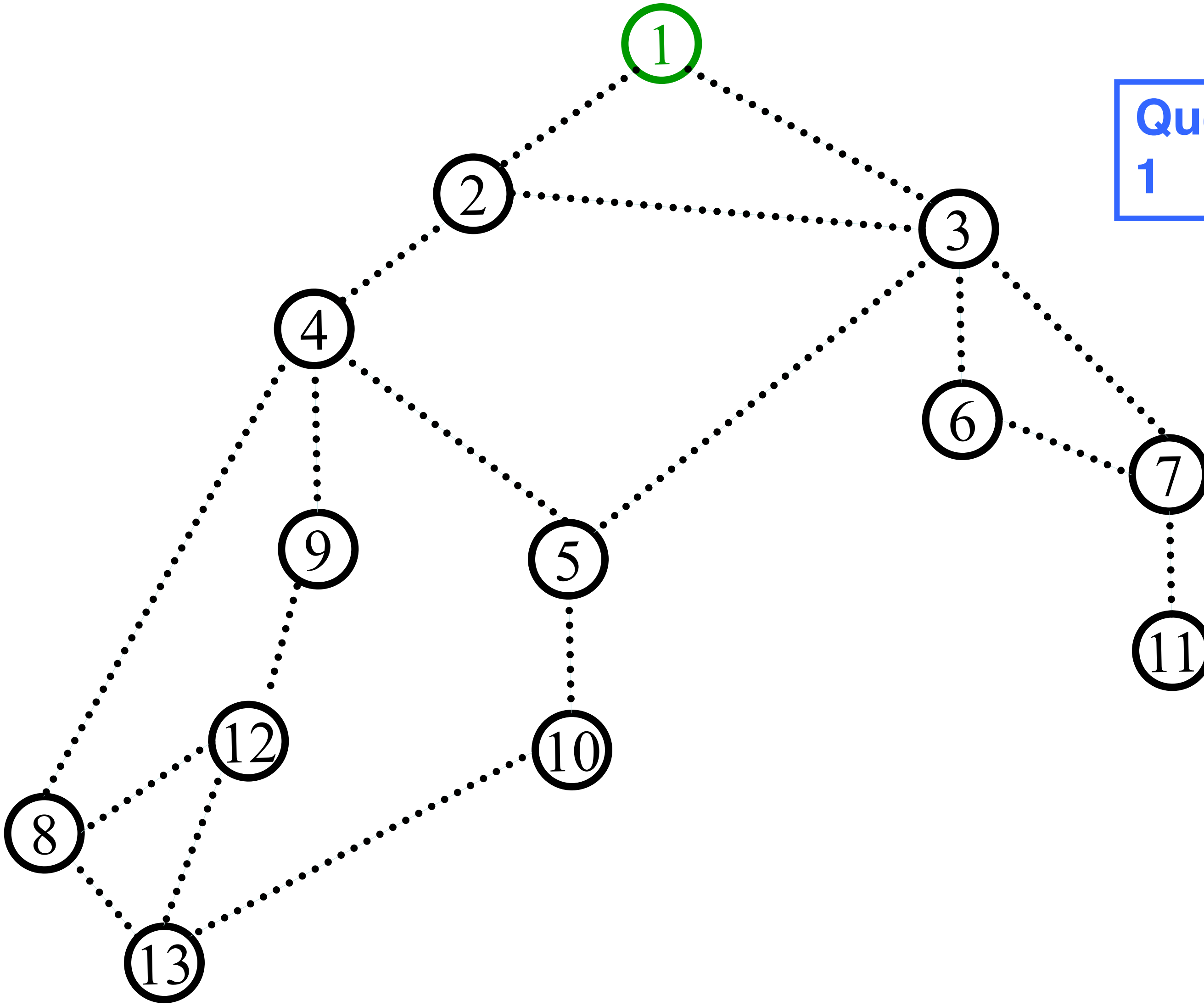
The queue will always have the list of Discovered vertices

BFS algorithm

Initialization: mark all vertices "undiscovered"

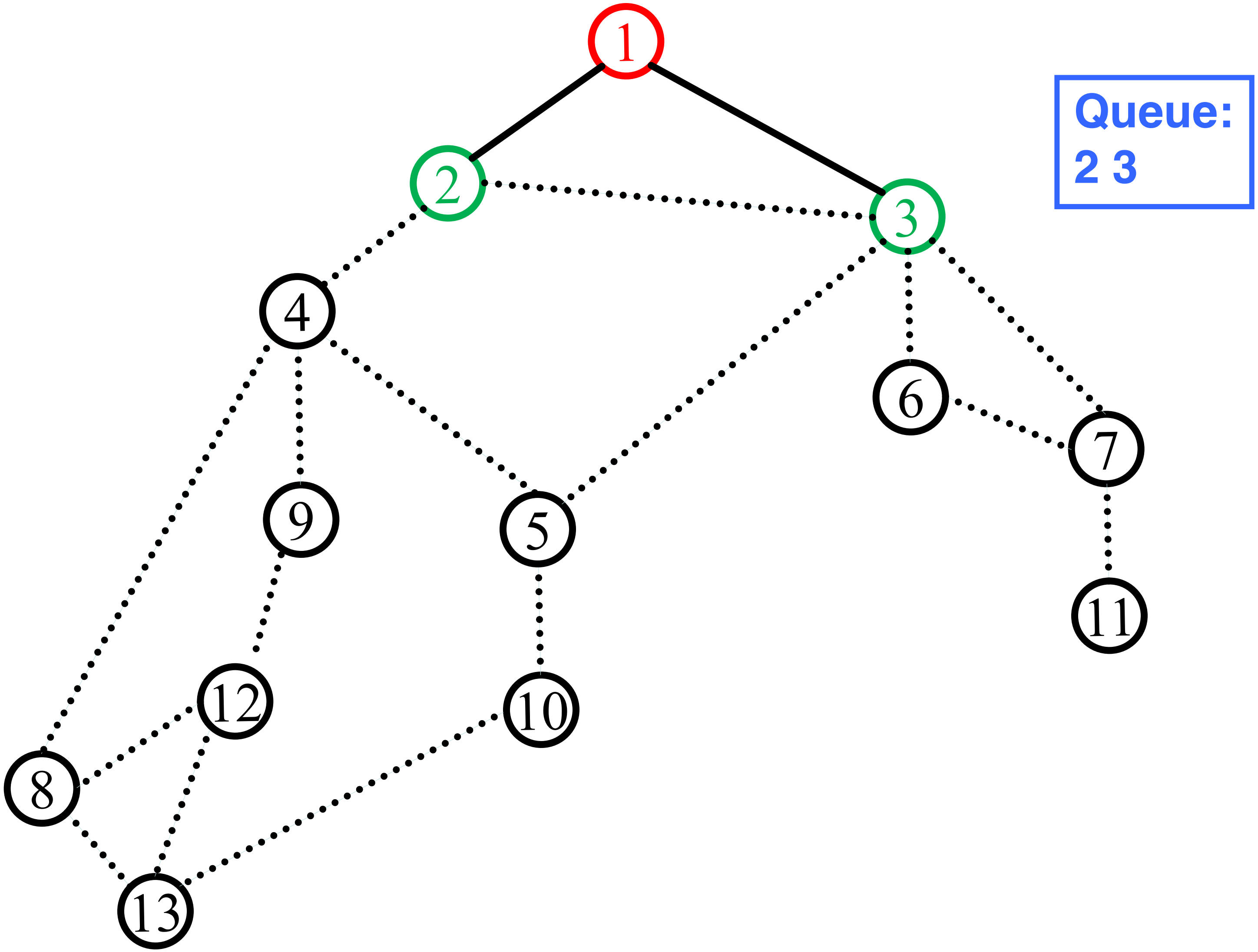
```
BFS(s)
    mark s discovered
    queue = { s }
    while queue not empty
        u = remove_first(queue)
        for each edge {u, x}
            if (x is undiscovered)
                mark x discovered
                append x on queue
        mark u fully-explored
```

BFS(1)

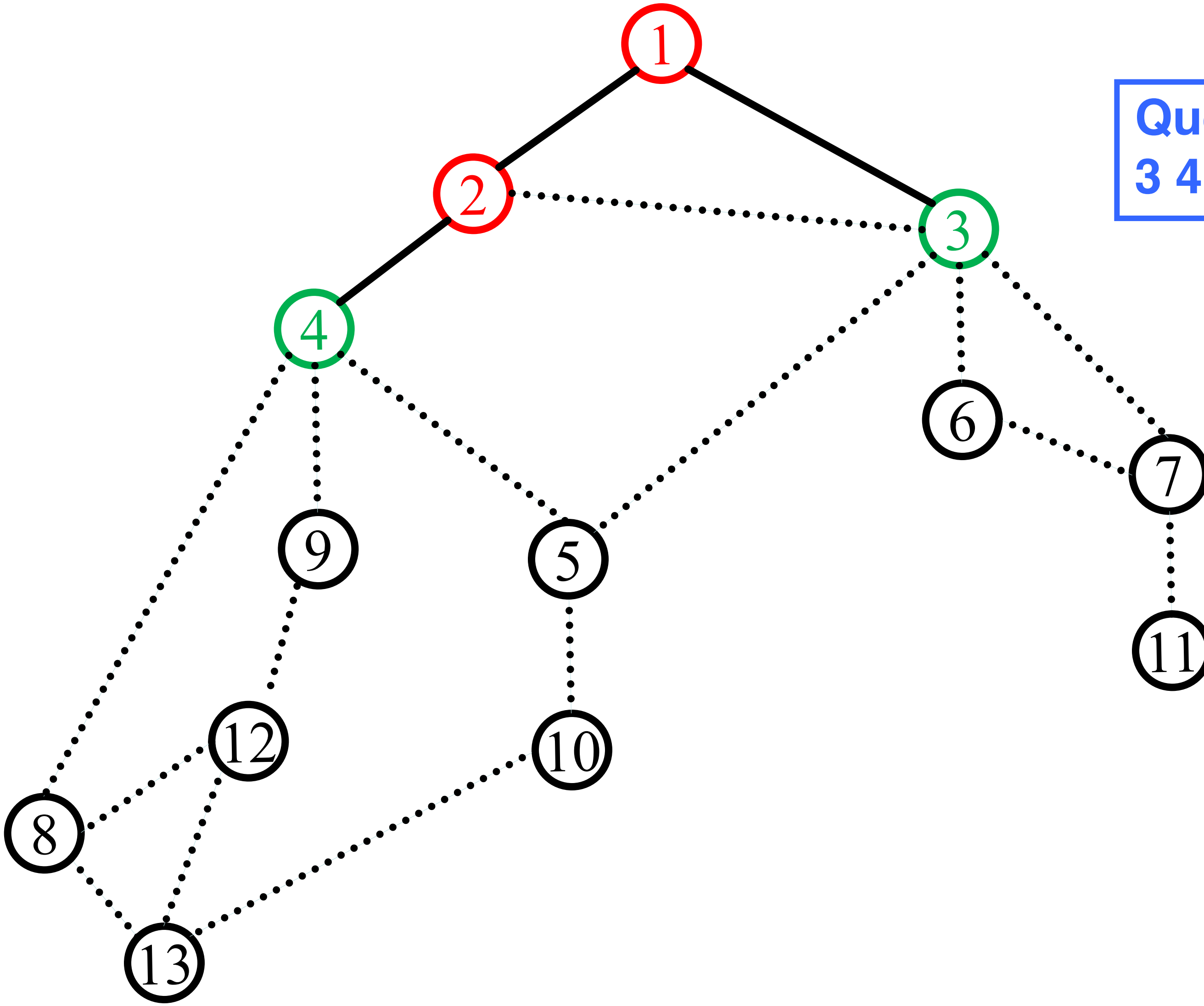


Queue:
1

BFS(1)

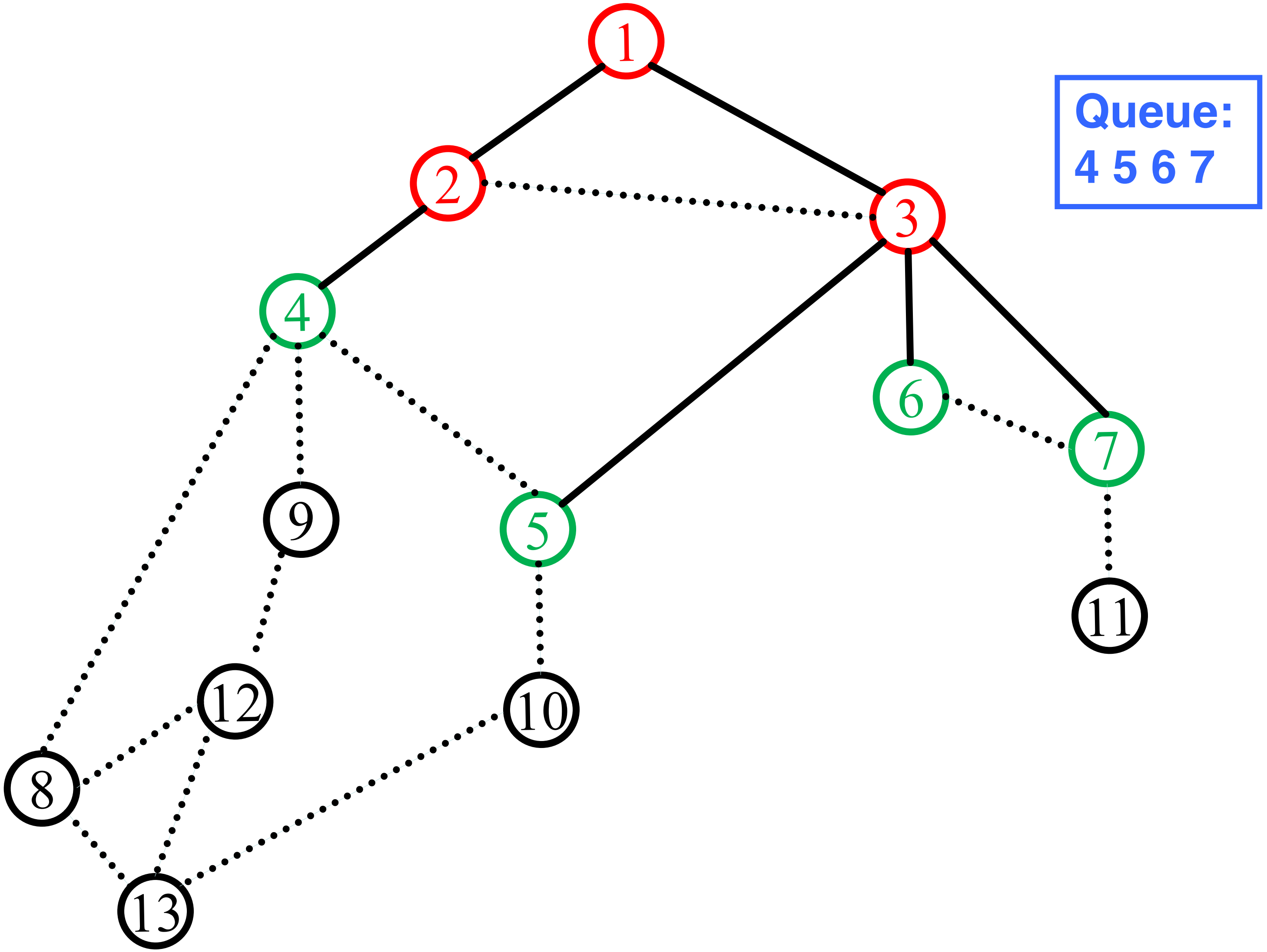


BFS(1)

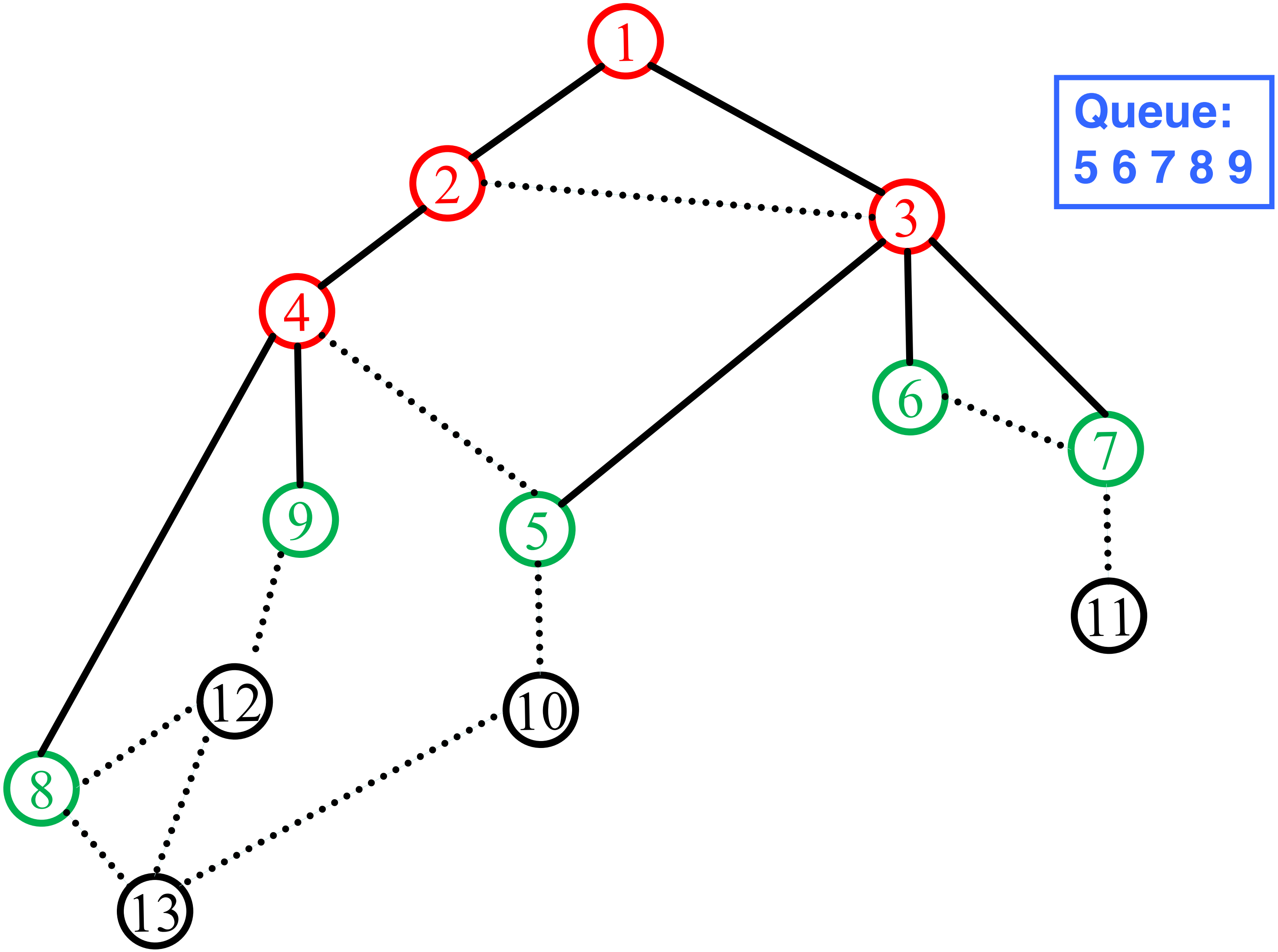


Queue:
3 4

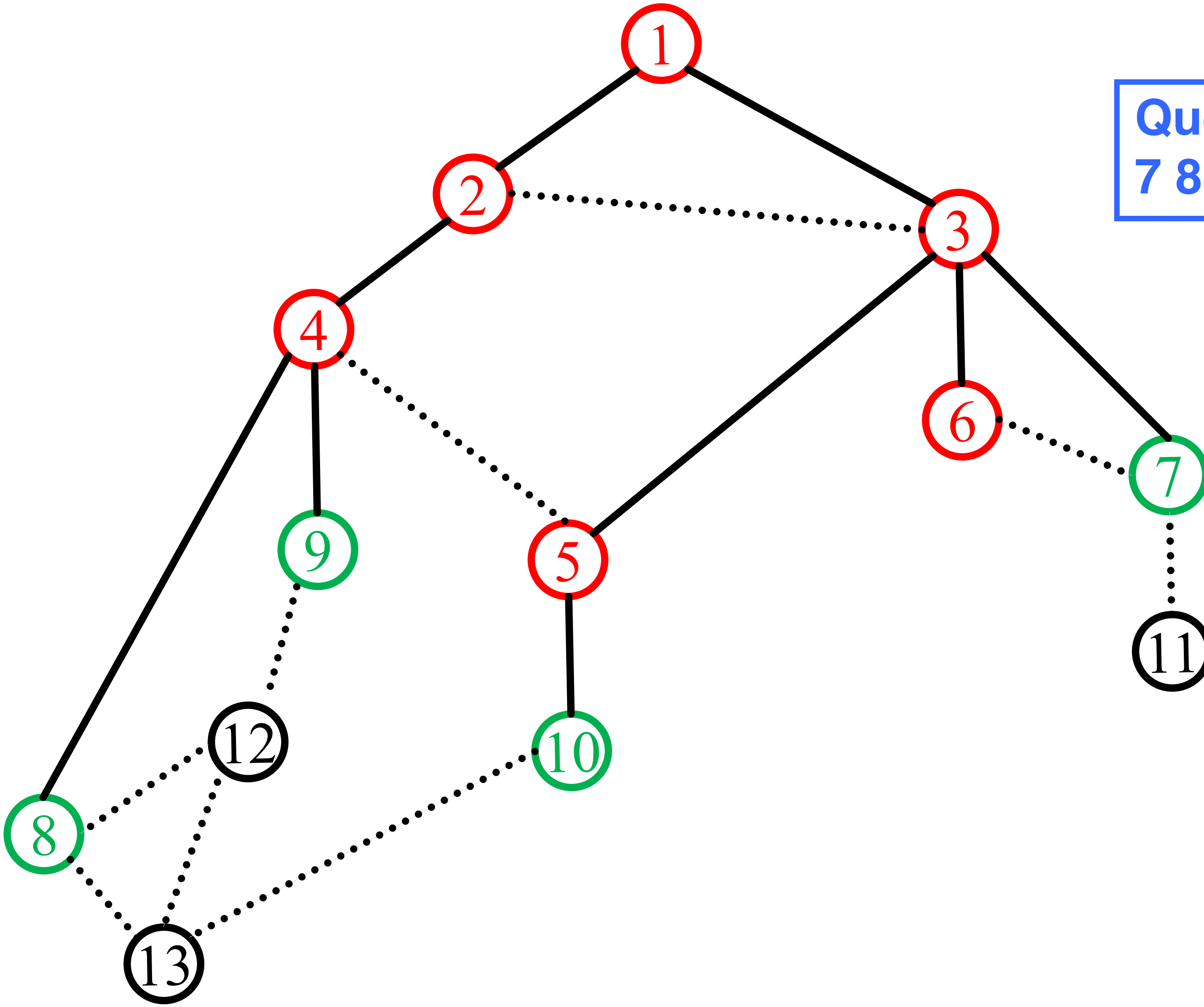
BFS(1)



BFS(1)

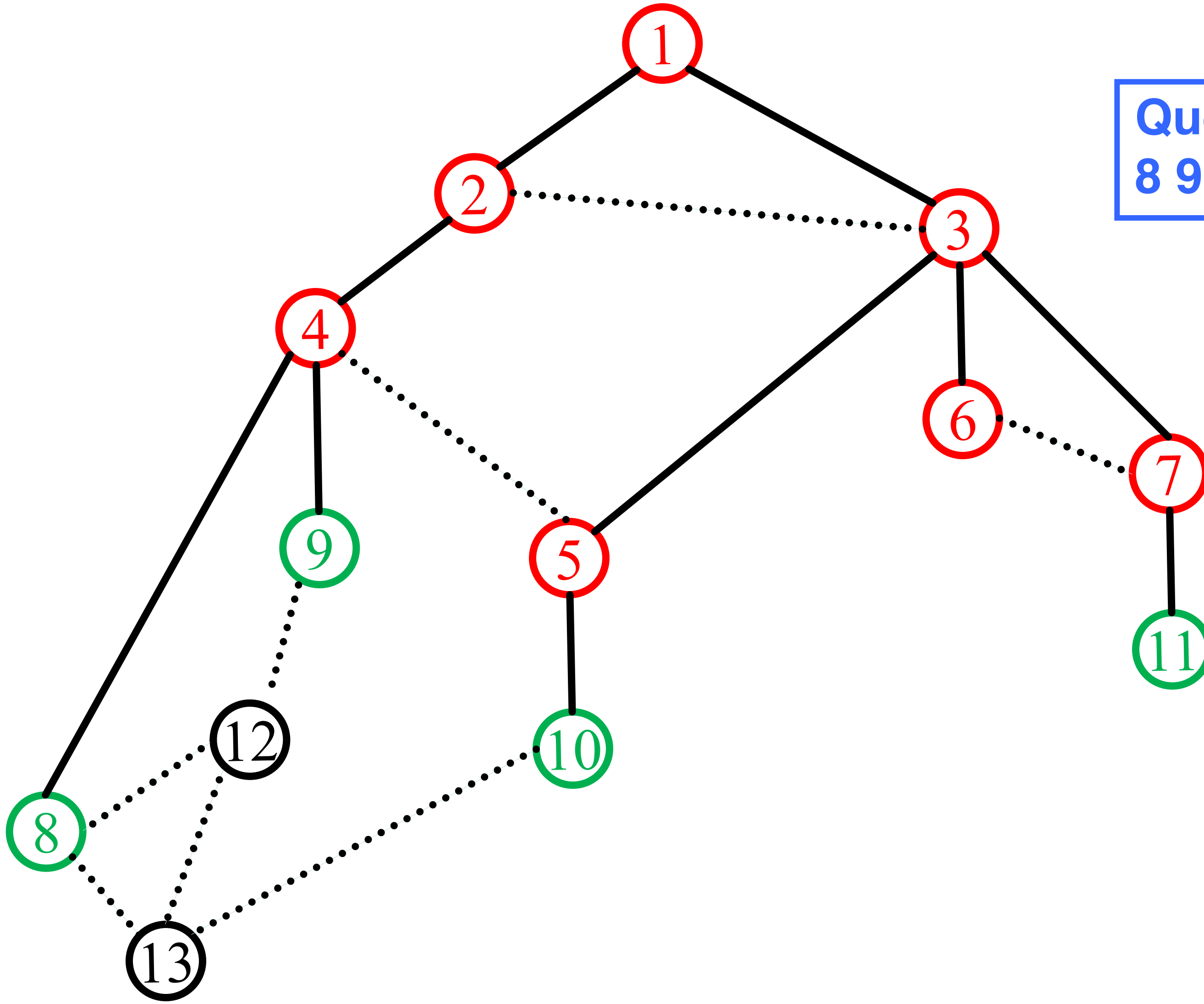


BFS(1)



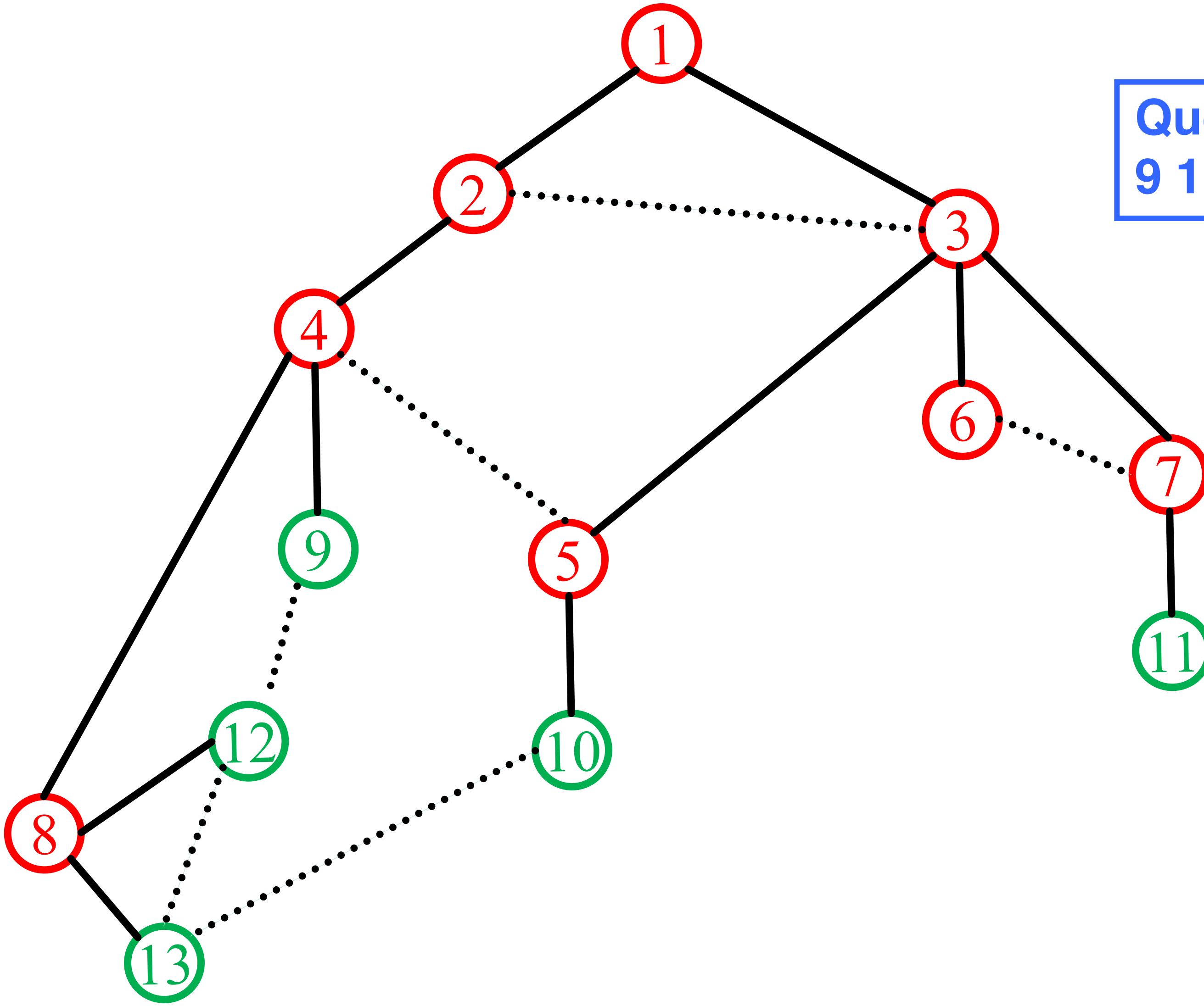
Queue:
7 8 9 10

BFS(1)



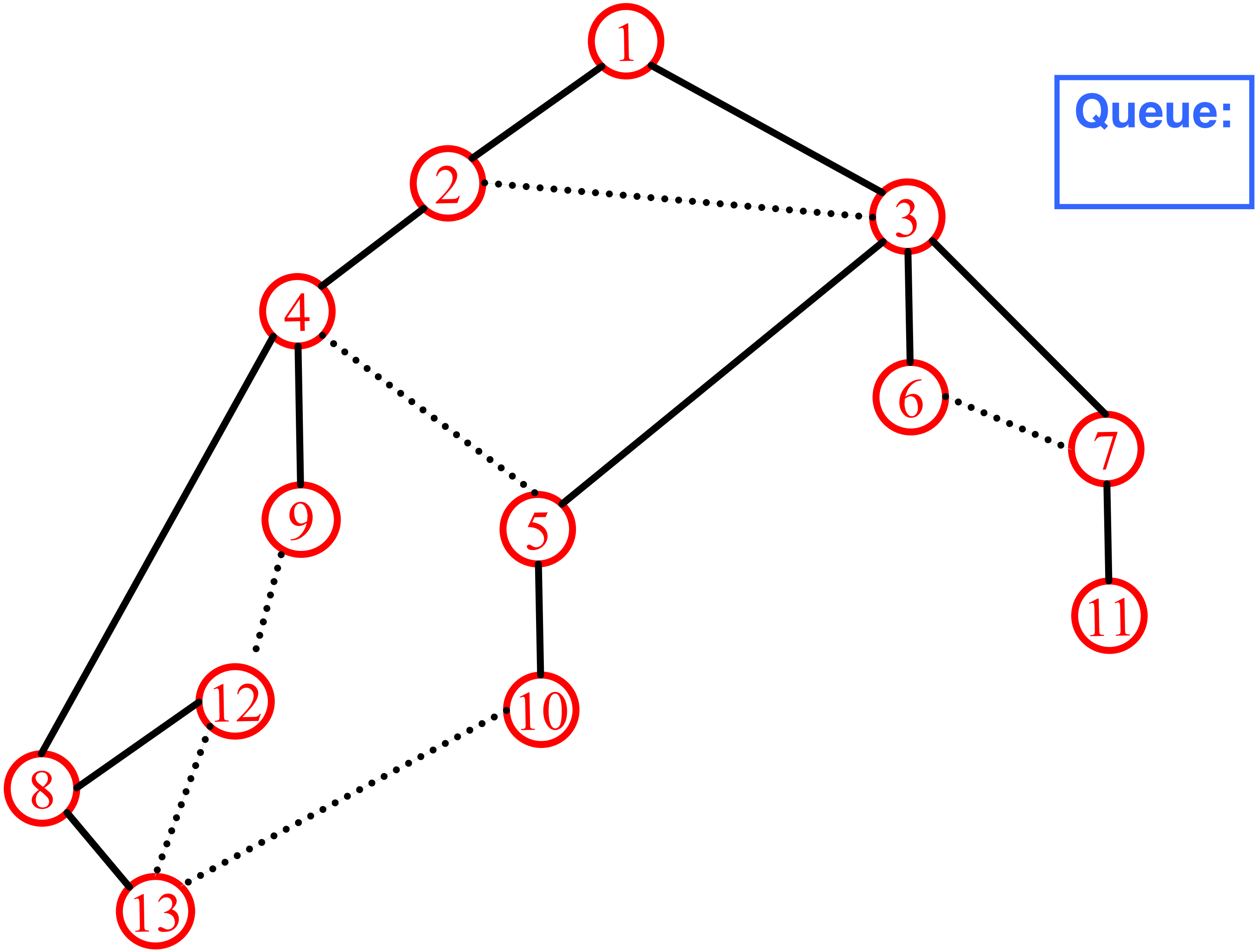
Queue:
8 9 10 11

BFS(1)



Queue:
9 10 11 12 13

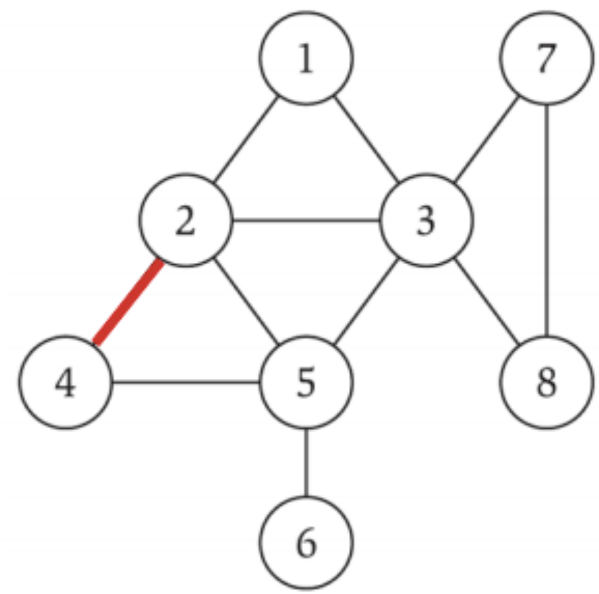
BFS(1)



Graph representation

Adjacency matrix. n-by-n matrix with $A_{uv} = 1$ if (u, v) is an edge.

- Space proportional to n^2.
- Checking if (u, v) is an edge takes $\Theta(1)$ time.
- Identifying all edges takes $\Theta(n^2)$ time.



	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

BFS Analysis

Initialization: mark all vertices "undiscovered"

Graph representation: adjacency matrix

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

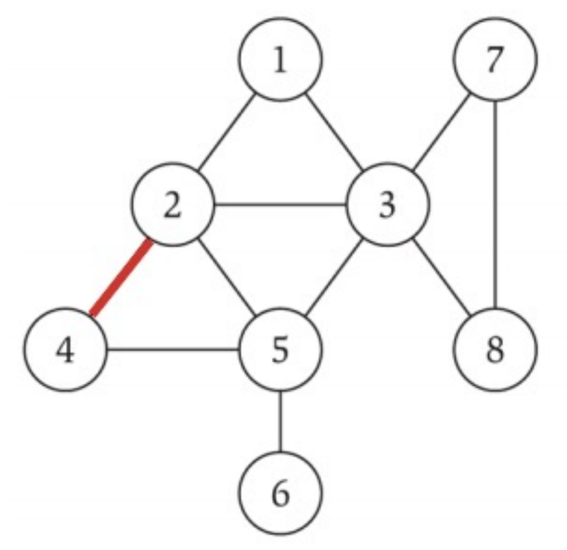
O(n) times: Check every vertex x

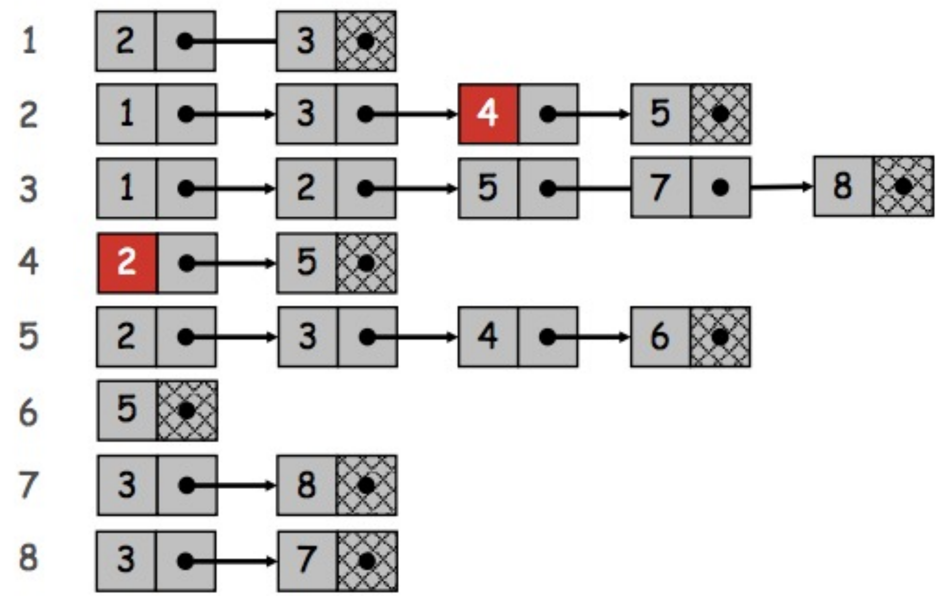
Overall: O(n^2) time

Graph representation

Adjacency list. Node indexed array of lists.

- Space proportional to $m+n$.
- Checking if (u, v) is an edge takes $O(\deg(u))$ time.
- Identifying all edges takes $\Theta(m+n)$ time.





BFS Analysis

Initialization: mark all vertices "undiscovered"

Graph representation: adjacency list

BFS(s)

mark s discovered

queue = { s }

O(n) times:
At most once per vertex

while queue not empty

u = remove_first(queue)

O(deg(u)) times:
At most twice per edge

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

if (x is undiscovered)

mark x discovered

append x on queue

mark u fully-explored

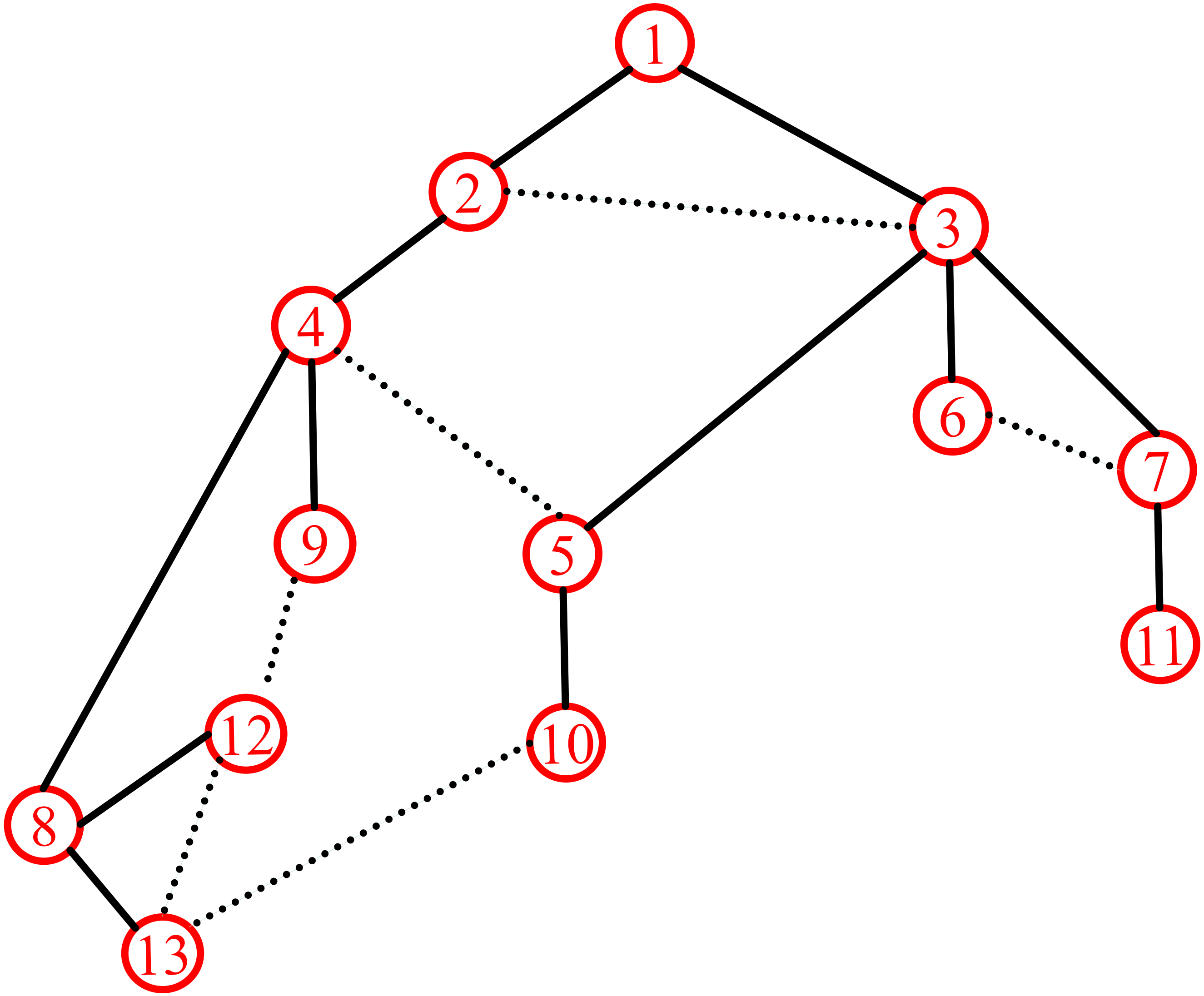
Overall: O(n+m) time

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

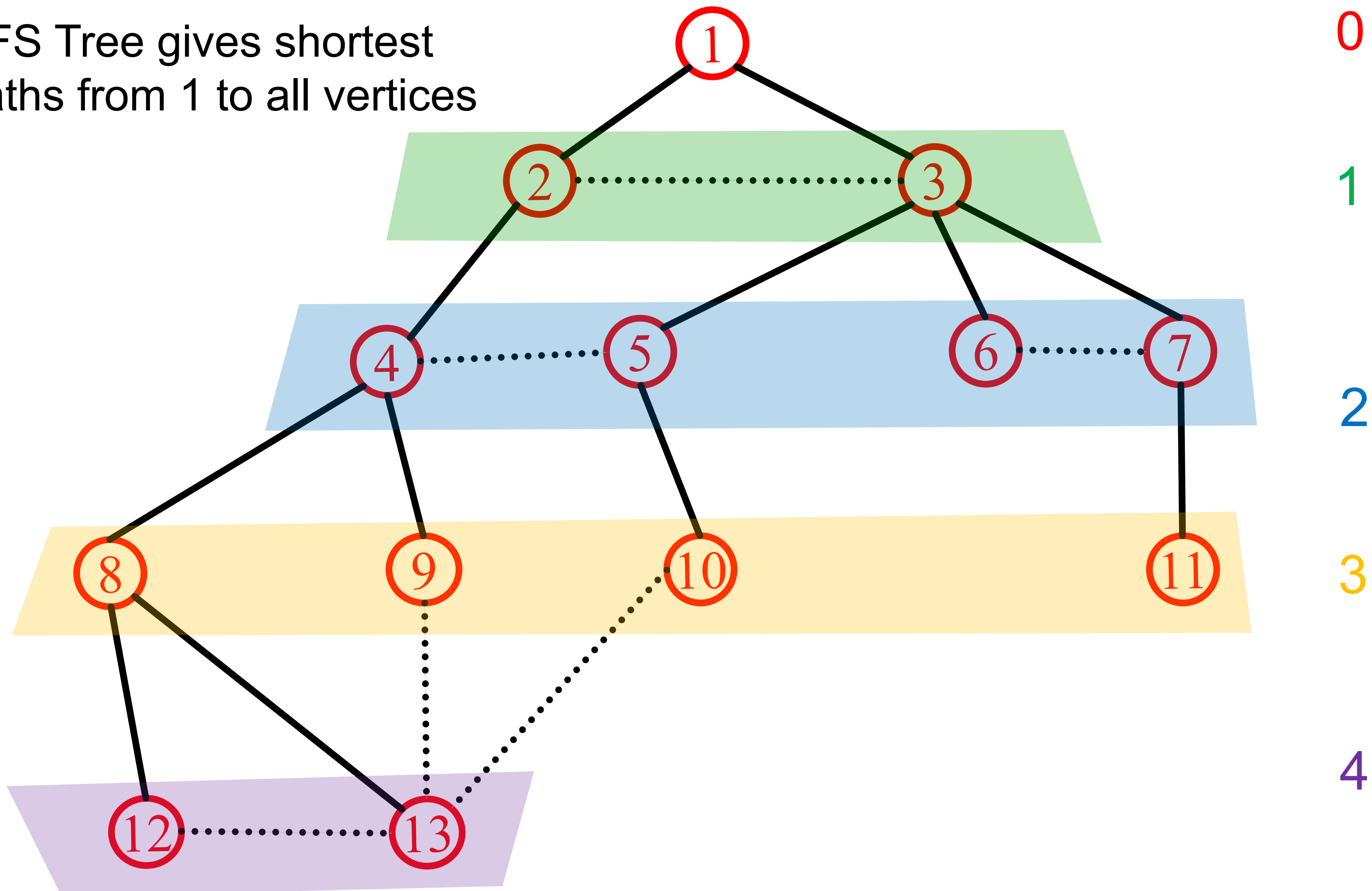
- BFS(s) visits a vertex v if and only if there is a path from s to v

- 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

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.
We have

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

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

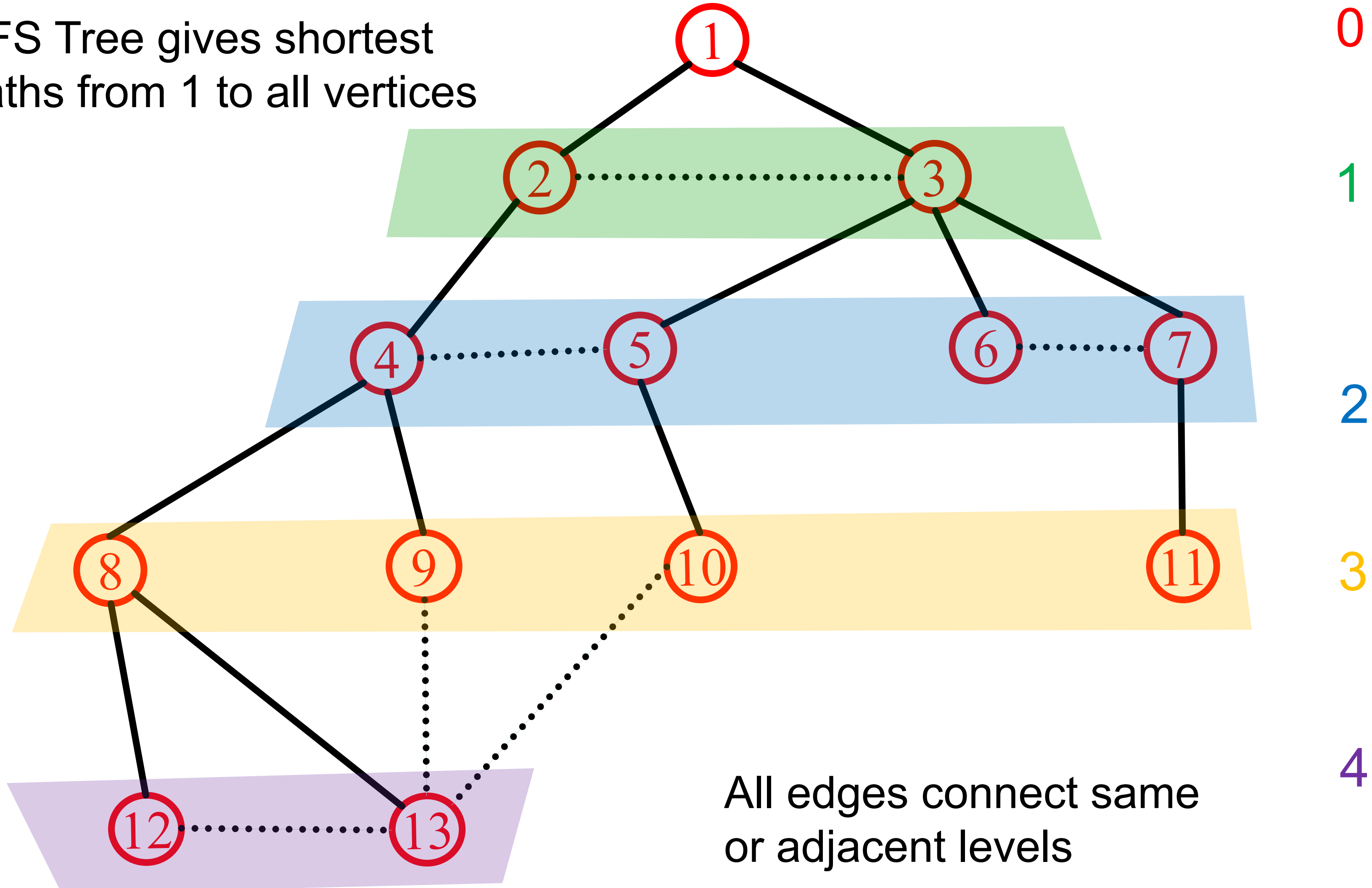
This proves the lemma.

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

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 $i + 1$.

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.