CSE 421: Introduction to Algorithms

Graph

Yin-Tat Lee

Exercise: #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.

Exercise: Degree 1 vertices

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

(Every tree has a leaf)

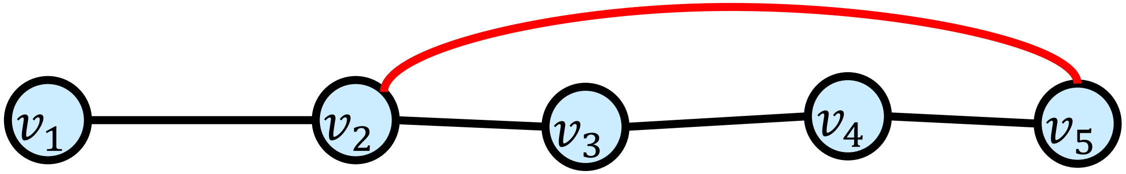
Proof: (By contradiction)

Suppose every vertex has degree ≥ 2.

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



Exercise: Trees and Induction

Claim: Every tree with n vertices has $n - 1$ edges.

Proof: (Induction on n.)

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

Induction: 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.

Sparse Graphs

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

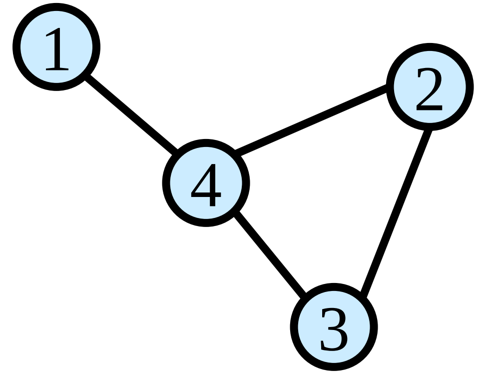
Sparse graphs are very common in practice

- Friendships in social network
- Planar graphs
- Web graph

$O(n + m)$ is usually much better runtime than $O(n^2)$.

Storing Graphs

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



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

Adjacency Matrix: A

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

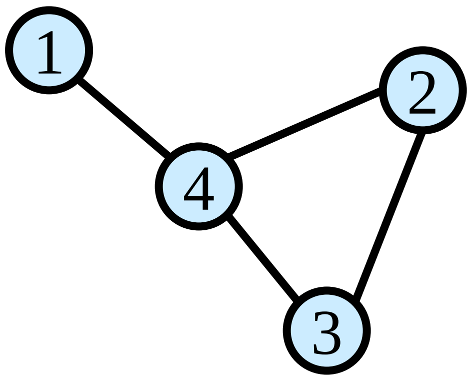
Advantage:

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

Disadvantage:

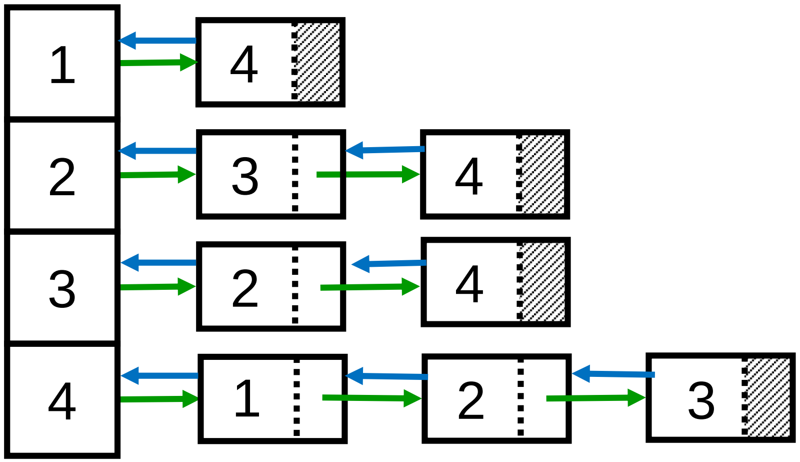
- Inefficient for sparse graphs both in storage and edge-access

Storing Graphs



Adjacency List:

$O(n+m)$ words



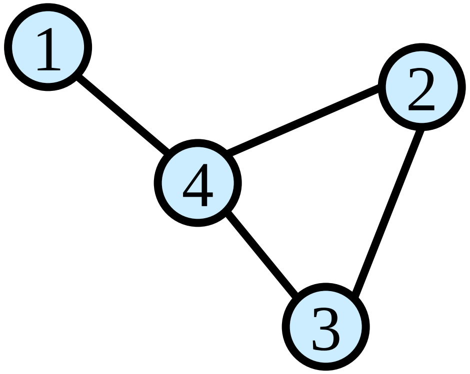
Advantage

- Compact for sparse
- Easily see all edges

Disadvantage

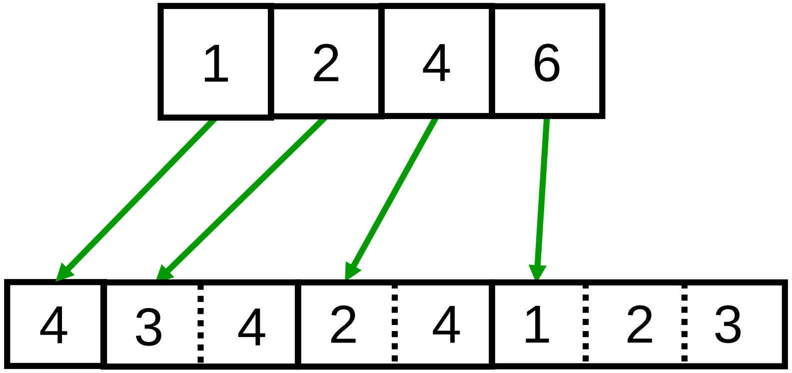
- Bad memory access
- Not good for parallel algorithms.

Storing Graphs



Adjacency Array:

O($n + m$) words



Advantage

- More compact for sparse
- Easily see all edges
- Better for memory access
- Better for parallel algorithms.

Disadvantage

- Difficult to update the graph

Storing Graphs

Implicit Representation:

$f(v)$ outputs an iterator of neighbor of v.

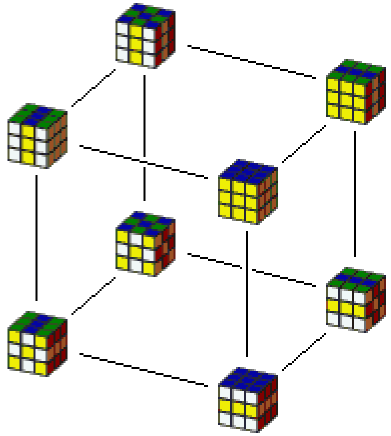
Aka, f(v)->next()->next()->next()->next()

Advantage

- No space is required

Disadvantage

- Mainly work for abstractly defined graph



2,125,922,464,947,725,402,112,000 states.

Storing Graphs

In practice, pick the representation according to the algorithm.

In this course, we focus on asymptotic runtime.

We can simply do this:

- For each vertex, use a hash table to store its neighbors
- This gives O(1) time for many operations

    Insert, Delete, Find, Next, …

CSE 421: Introduction to Algorithms

Breadth First Search

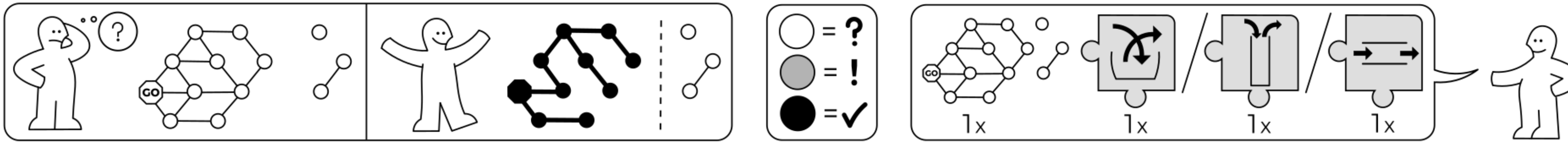
Yin Tat Lee

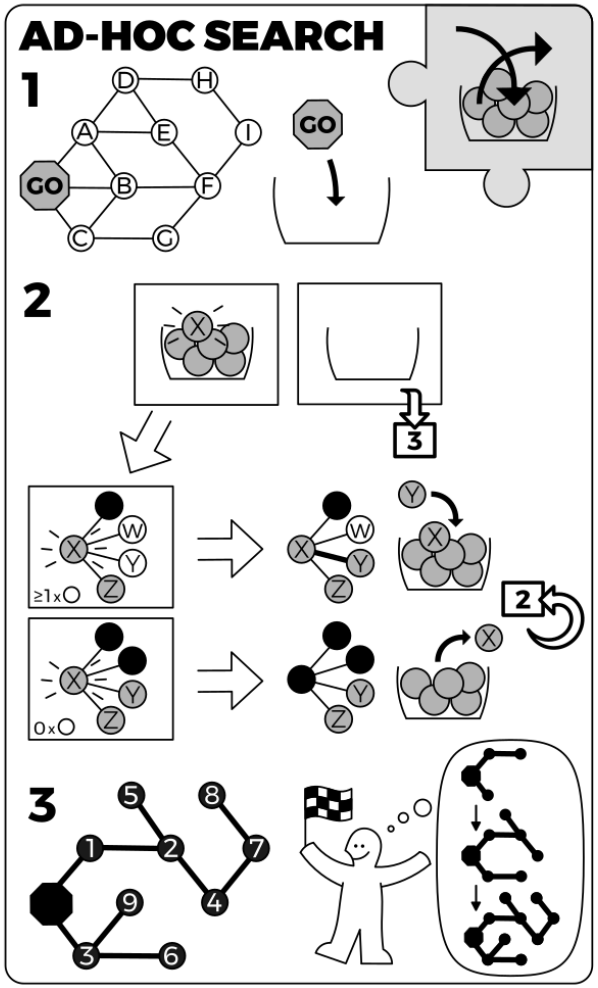
Graph Traversal

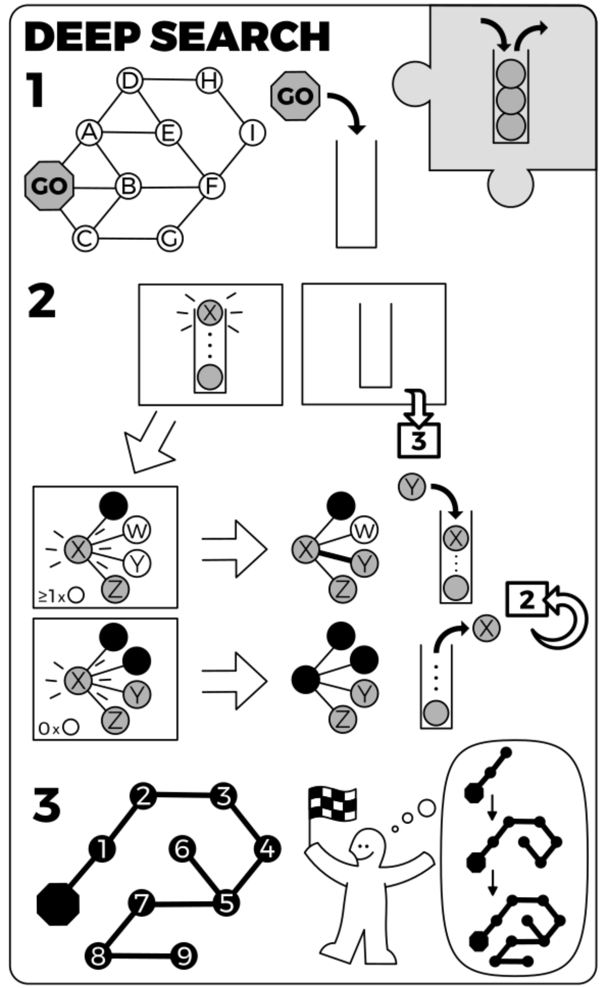
Walk (via edges) from a fixed starting vertex s to all vertices reachable from s.

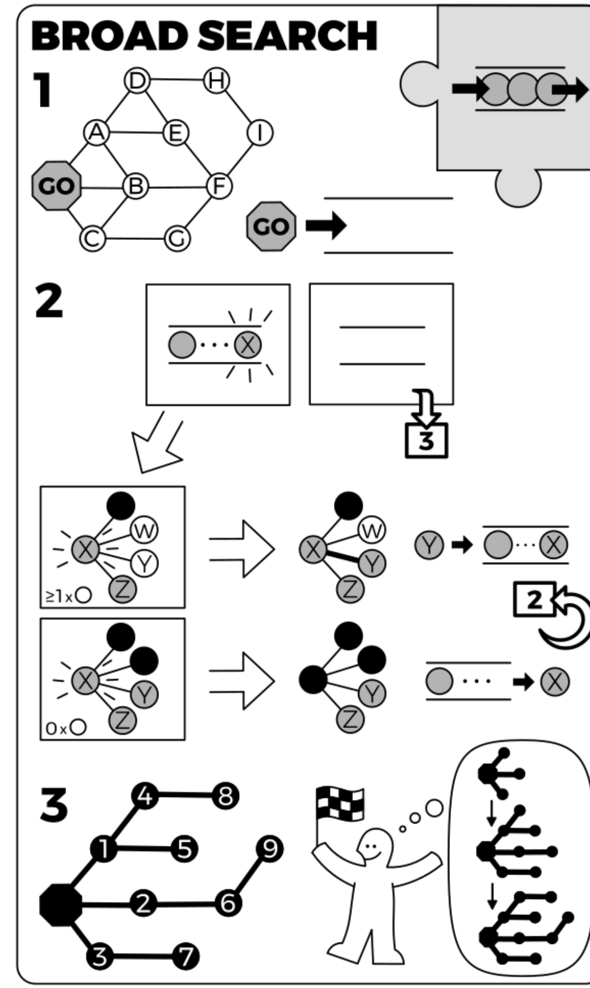
Applications:

- Web crawling
- Social networking
- Network Broadcasting
- Garbage Collection
- …









https://idea-instructions.com/

BFS implementation

Initialization: mark all vertices "undiscovered"

BFS(s)

mark s discovered

queue $Q = \{s\}$

Level[s] = 0

while Q is not empty

$u = Q.\mathrm{dequeue}()$

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

if (x is undiscovered)

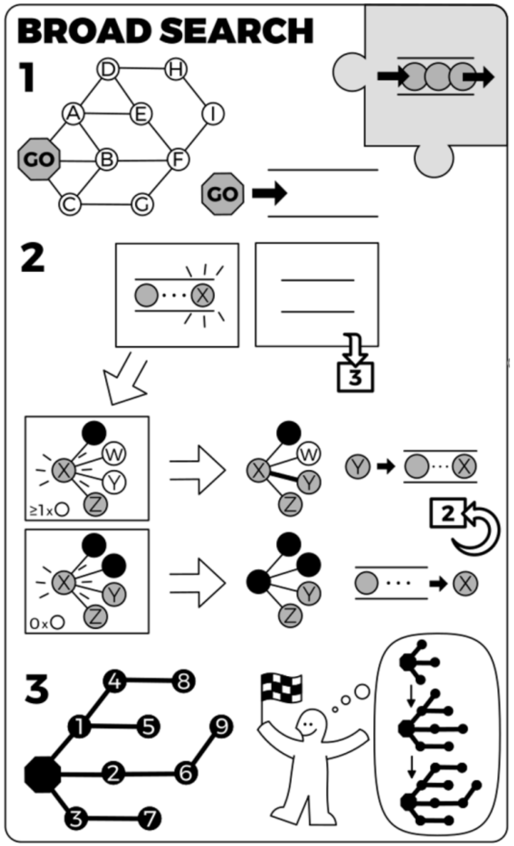
mark x discovered

append x on Q

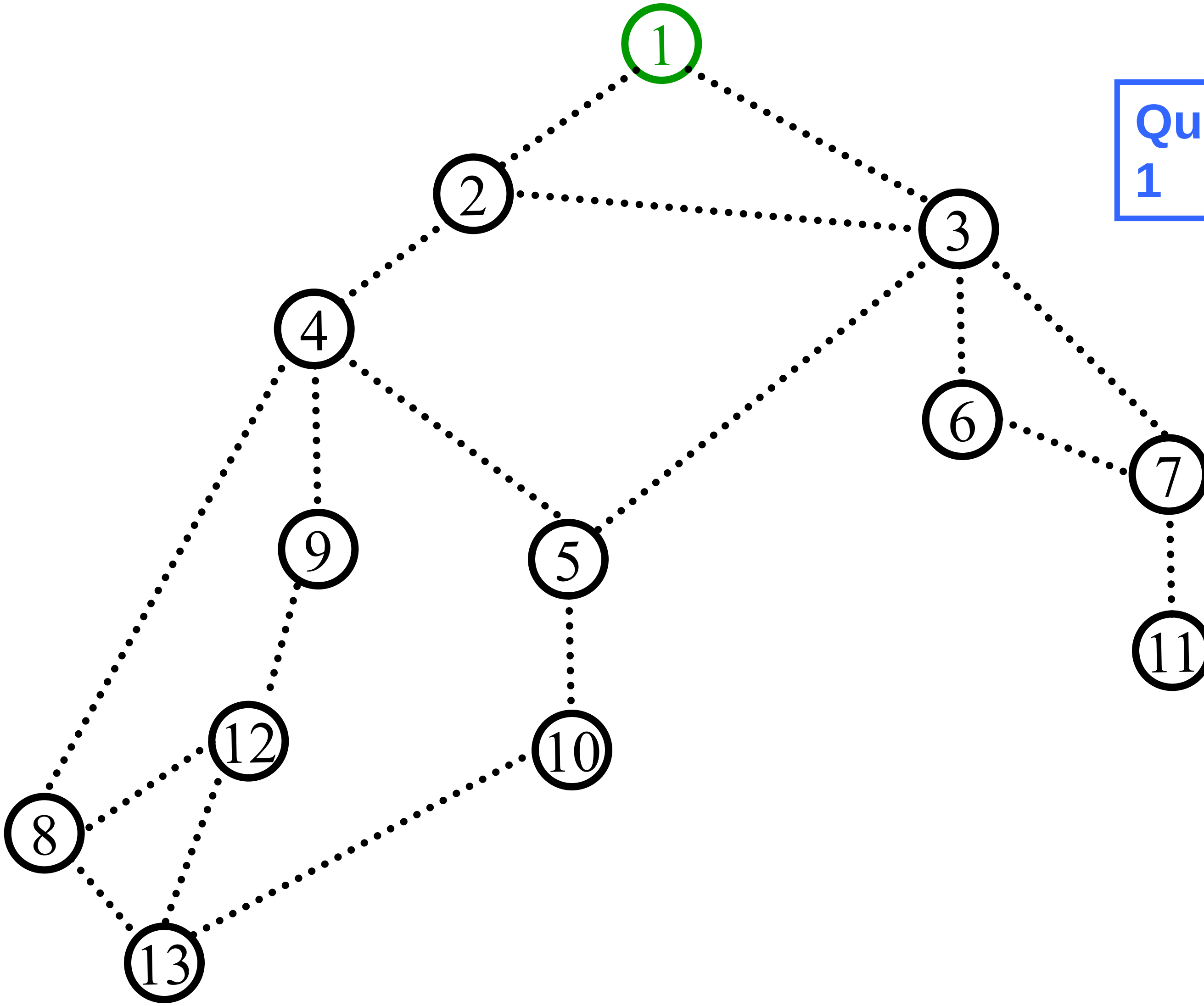
$x.\mathrm{parent} = u$

Level[x] = Level[u] + 1.

mark u fully-explored

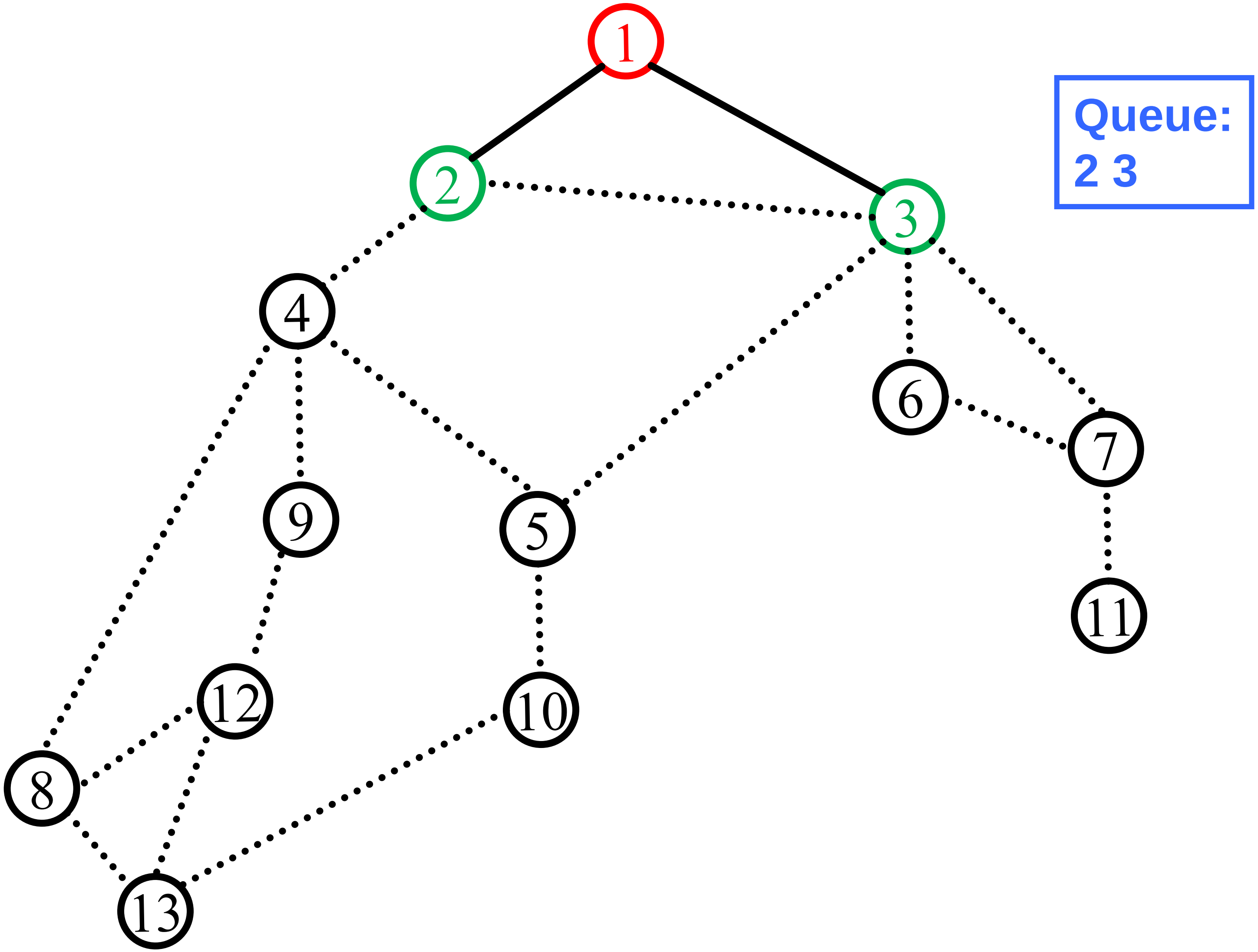


BFS(1)

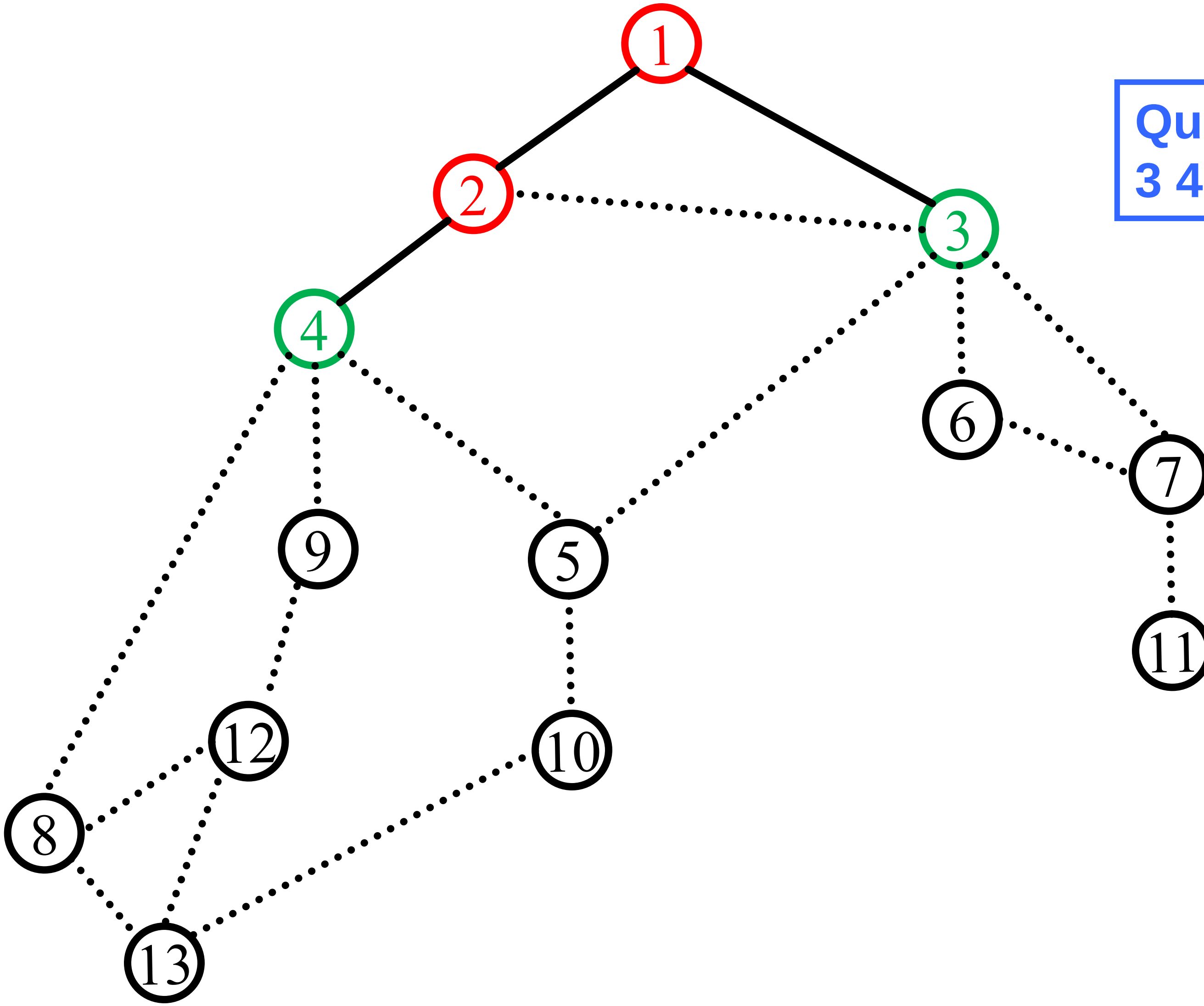


Queue:
1

BFS(1)

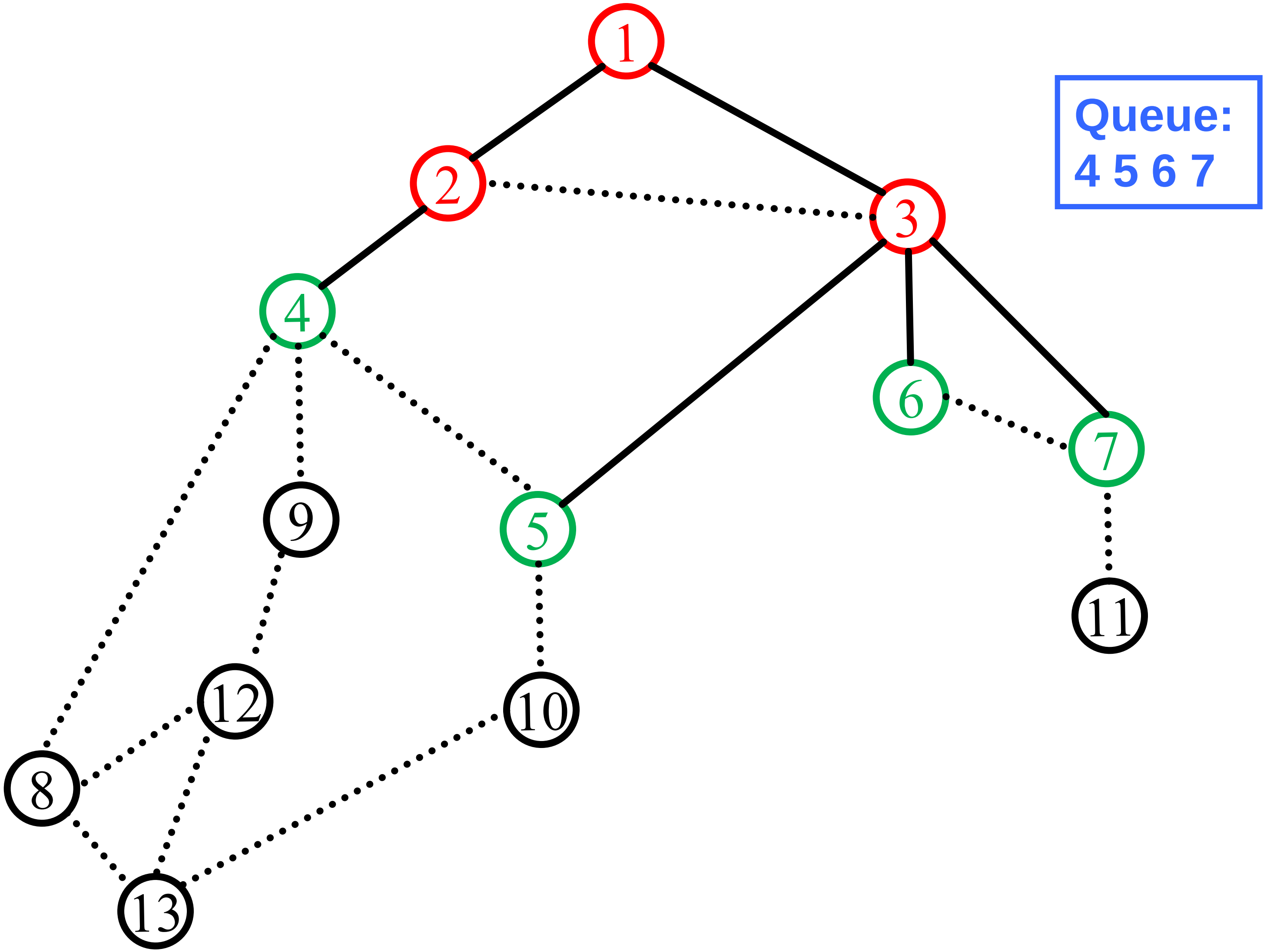


BFS(1)

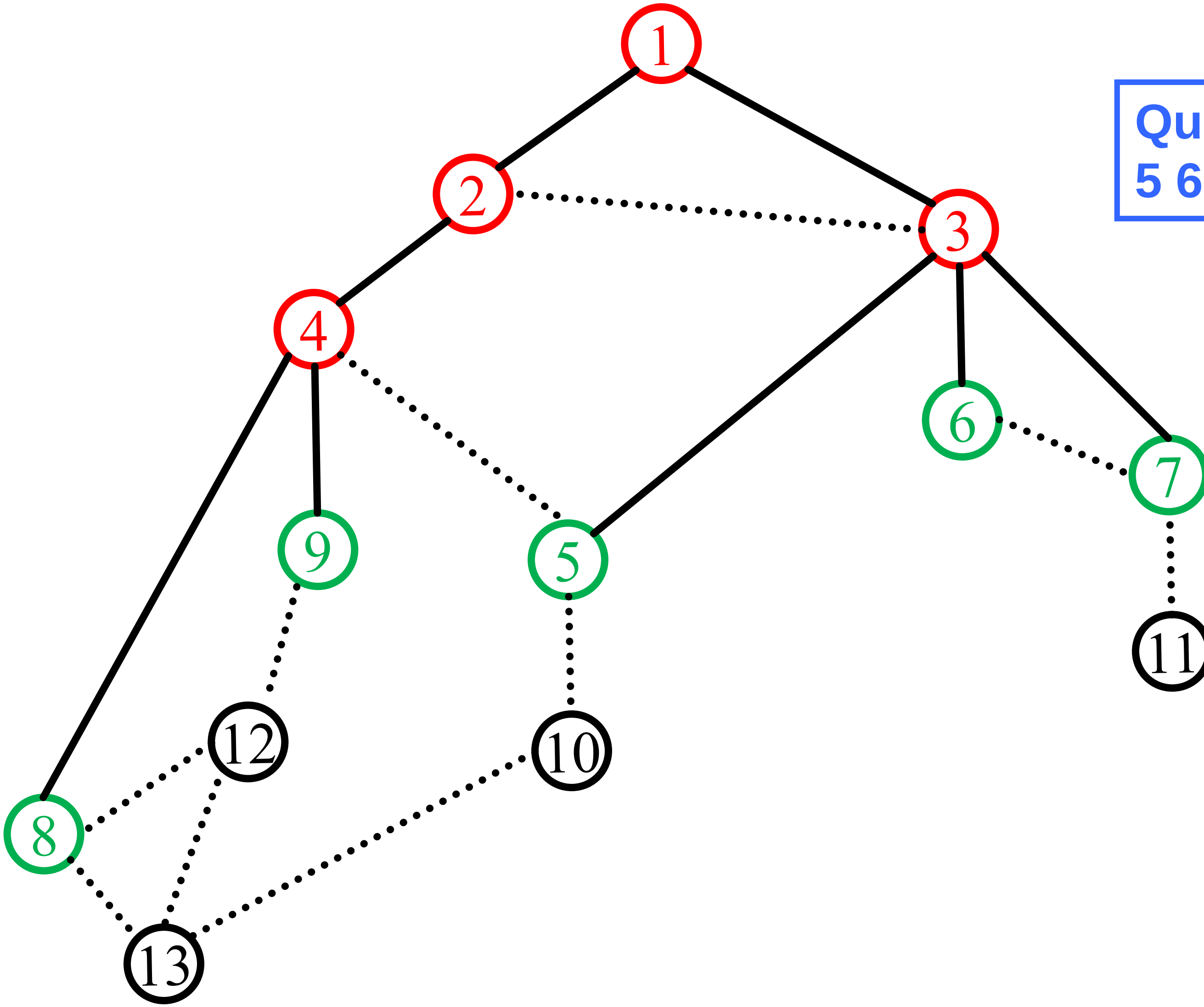


Queue:
3 4

BFS(1)

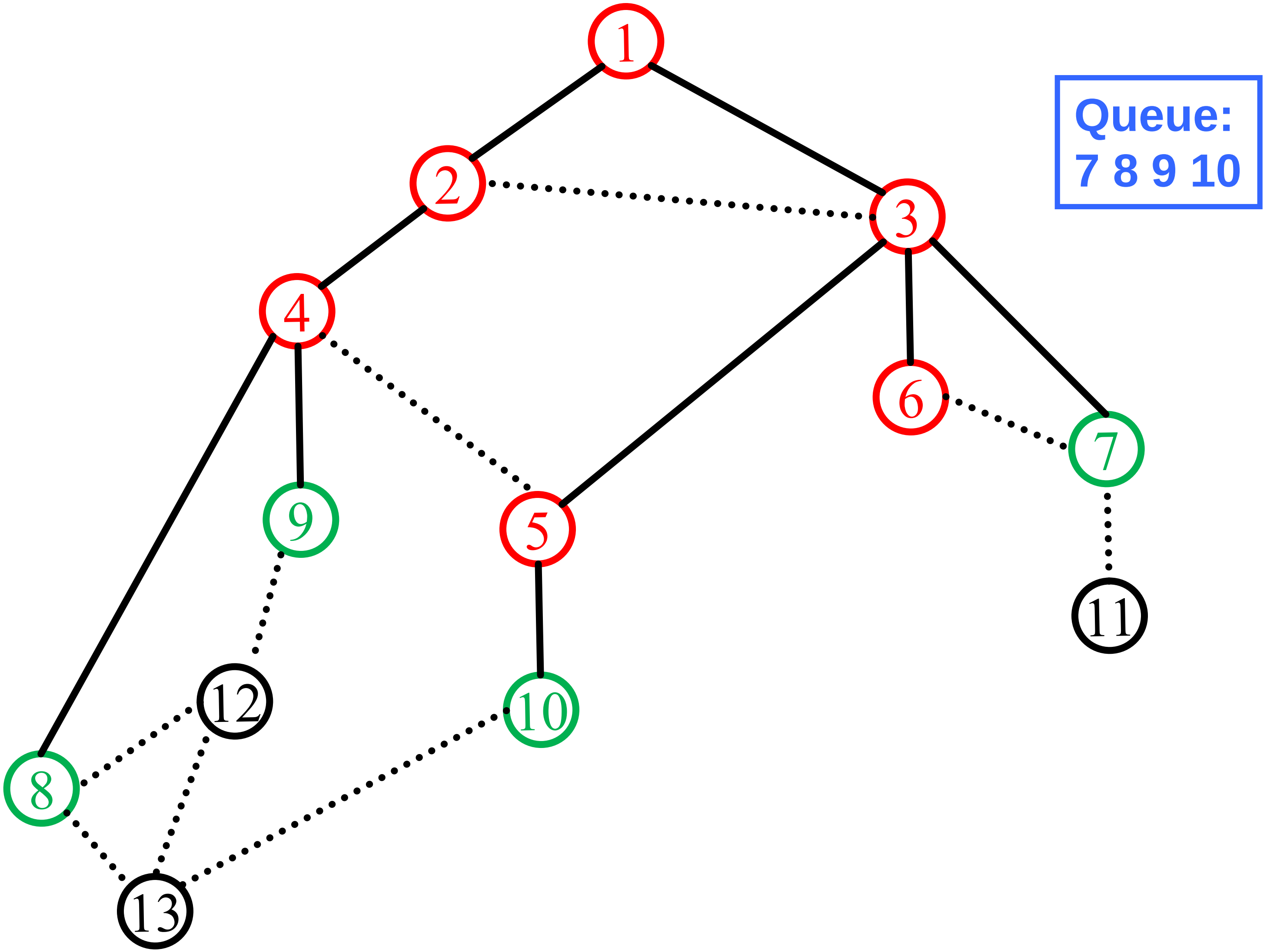


BFS(1)

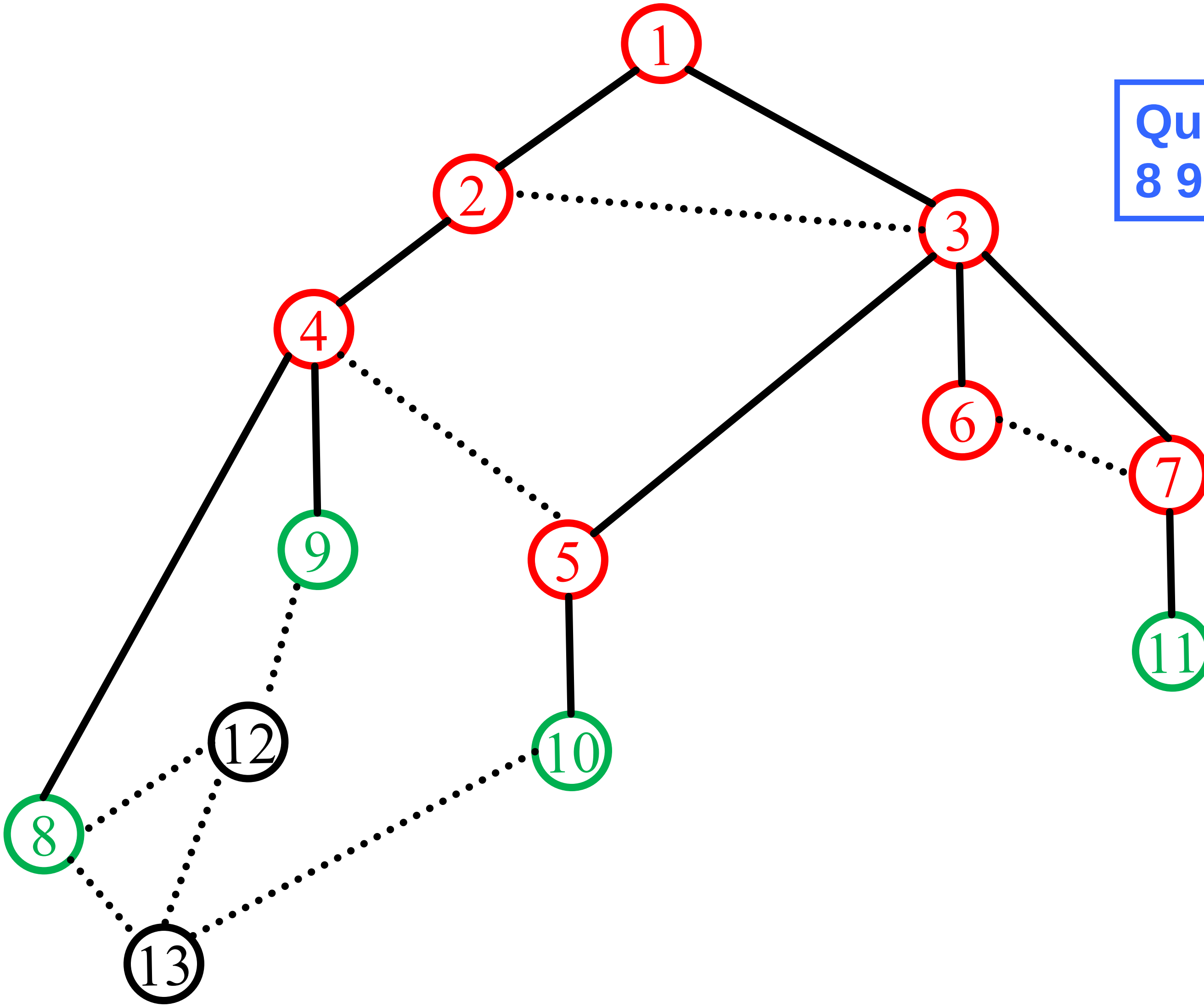


Queue:
5 6 7 8 9

BFS(1)

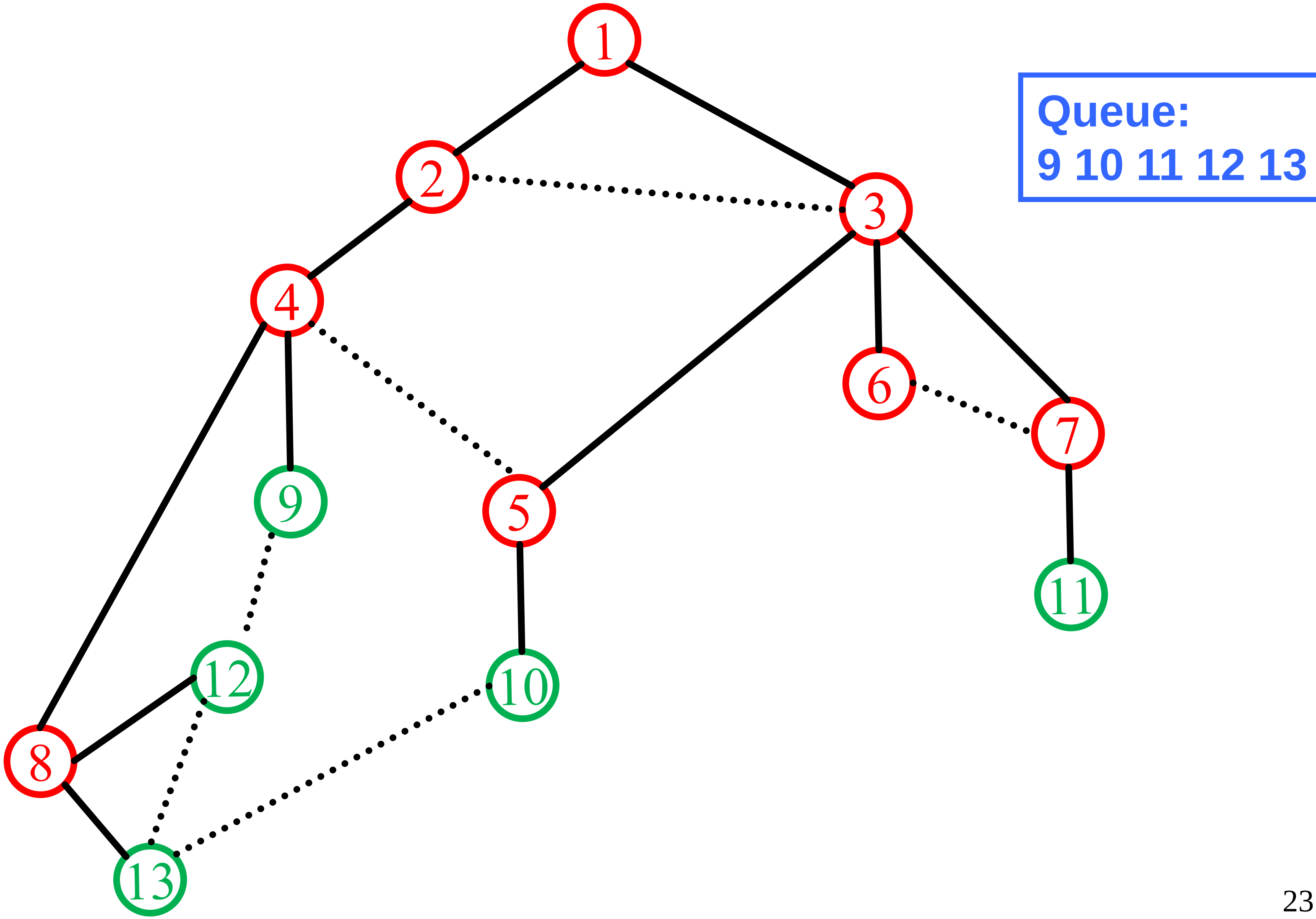


BFS(1)

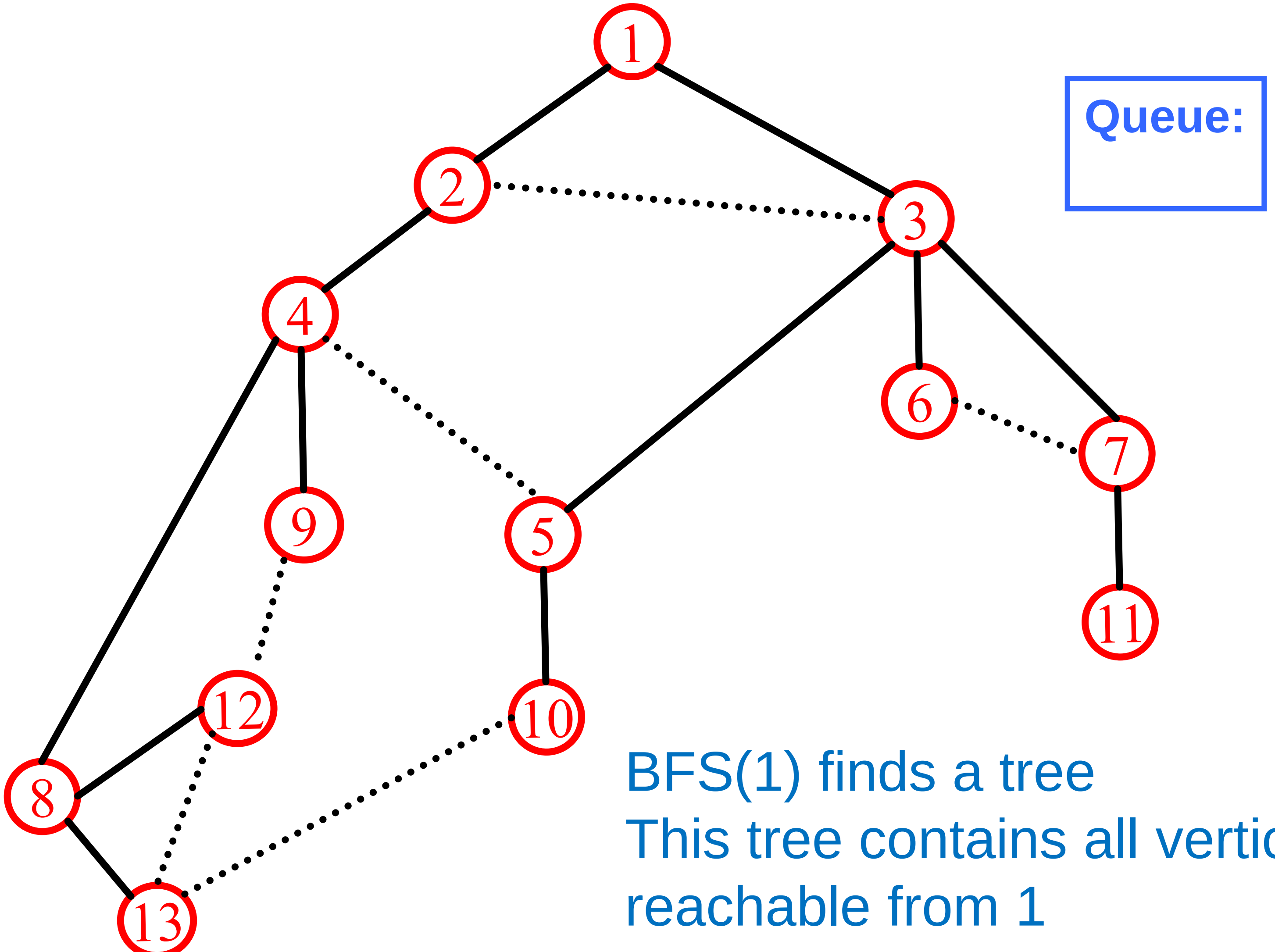


Queue:
8 9 10 11

BFS(1)



BFS(1)



BFS(1) finds a tree
This tree contains all vertices reachable from 1
We call it BFS tree

BFS Analysis

Initialization: mark all vertices "undiscovered"

BFS(s)

mark s discovered

queue $Q = \{s\}$

while Q is not empty

$u = Q.\text{dequeue}()$

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

if (x is undiscovered)

mark x discovered

append x on Q

$x.\text{parent} = u$

mark u fully-explored

O(n) times:
At most once per vertex

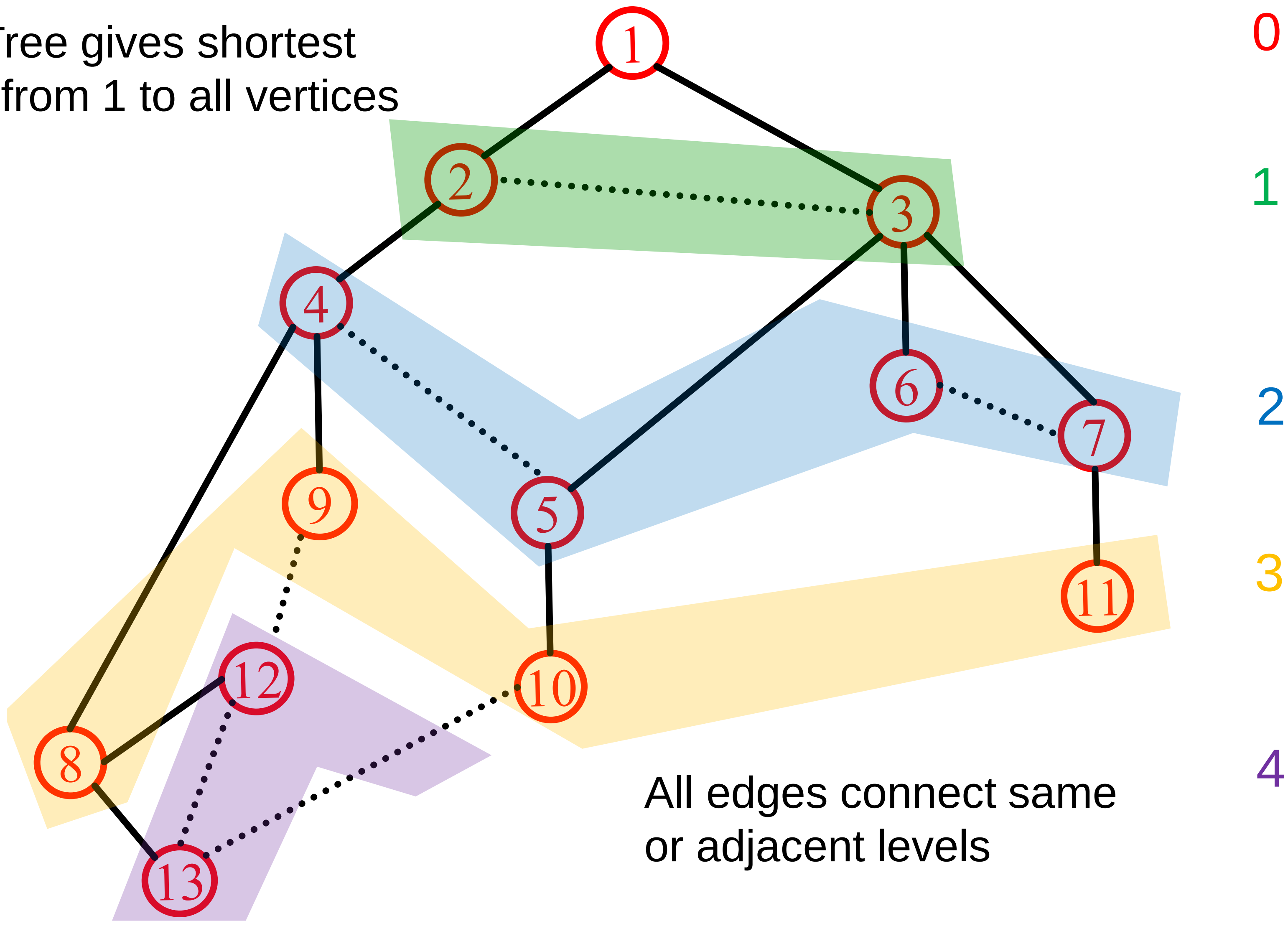
O(m) times:
At most twice per edge

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 "BFS tree" of G
- All nontree edges join vertices on the same or adjacent levels of the tree
- 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

BFS Application: Shortest Paths

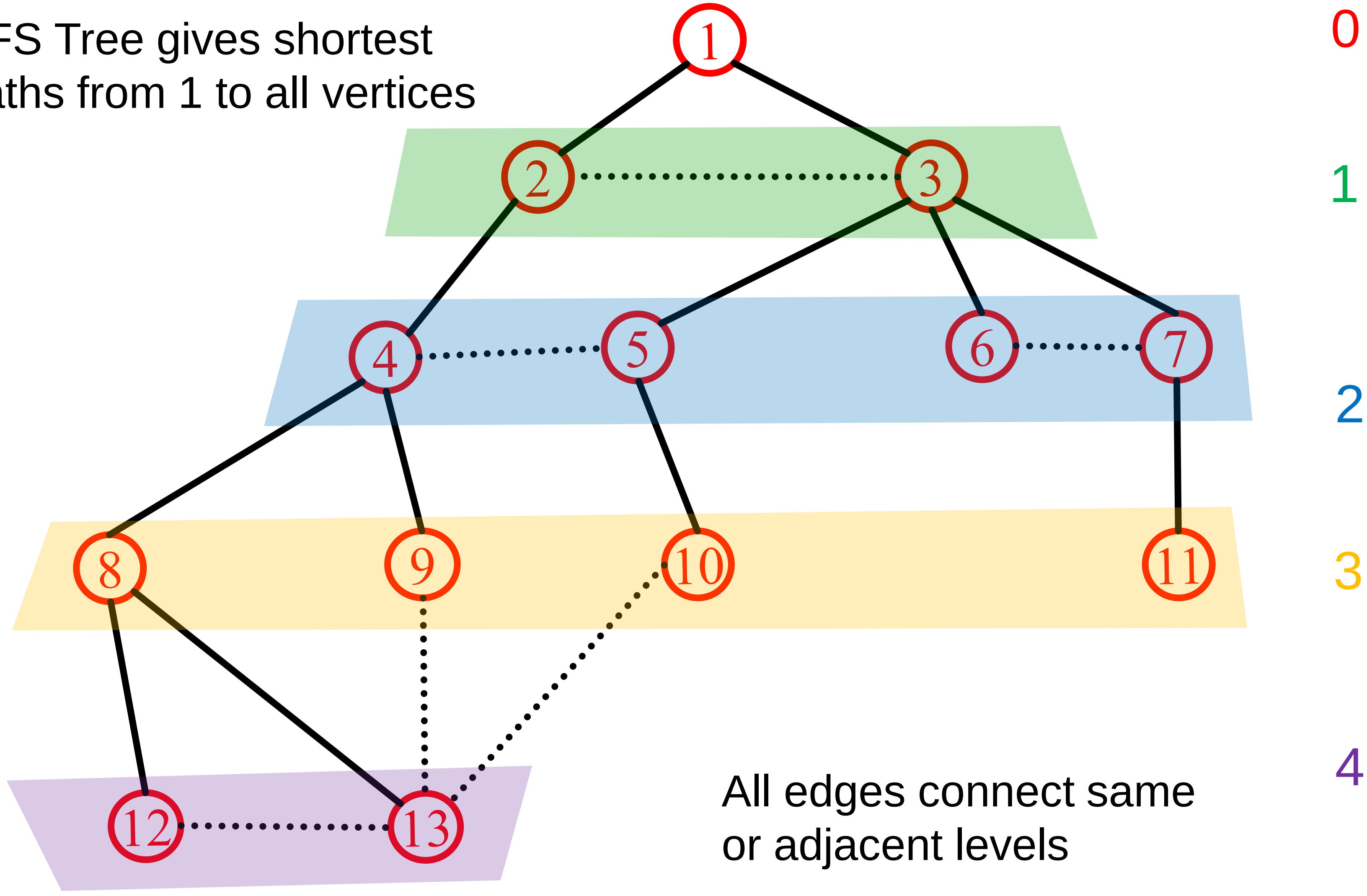
BFS Tree gives shortest paths from 1 to all vertices



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 $\leq i$
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 $s = v_0, v_1, \dots, v_i = v$ is the shortest path to v.
By previous claim,

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

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

CSE 421: Introduction to Algorithms

Application of BFS

Yin Tat Lee

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 $A[u] \leftarrow c$ 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 $O(m+n)$.

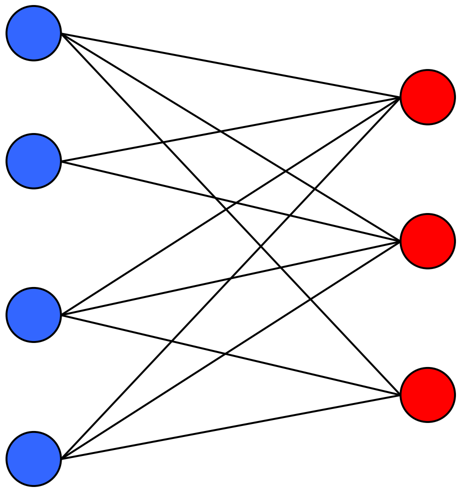
So, from now on, we can (almost) always assume the input graph is connected.

Quiz

Problem: Give an algorithm to detect whether a given undirected graph contains a cycle.

Bipartite Graphs

Definition: An undirected graph $G = (V, E)$ is bipartite
if you can partition the vertex set into 2 parts (say, blue/red or left/right) so that
all edges join vertices in different parts
i.e., no edge has both ends in the same part.



a bipartite graph

Application:

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

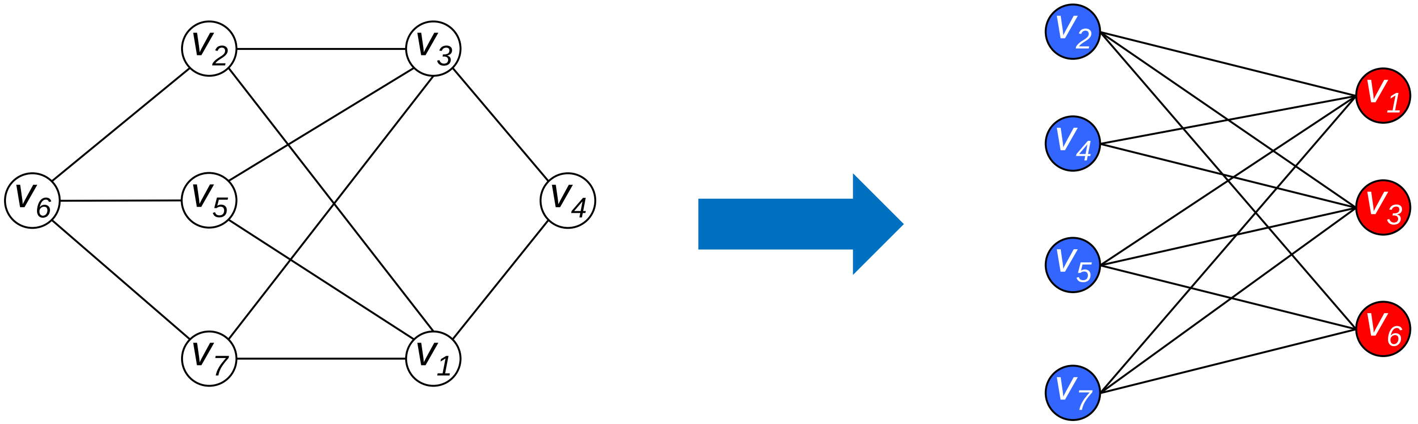
Testing Bipartiteness

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.



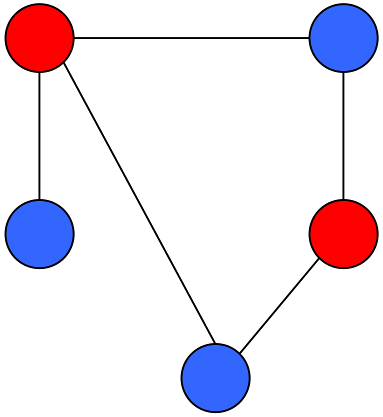
a bipartite graph G

another drawing of G

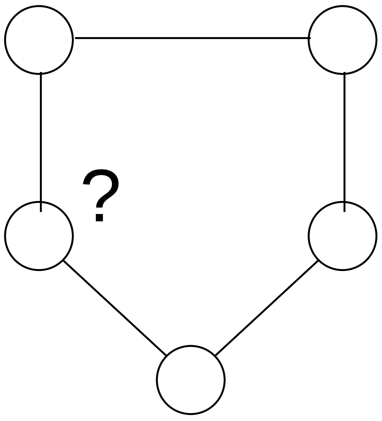
An Obstruction to Bipartiteness

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)



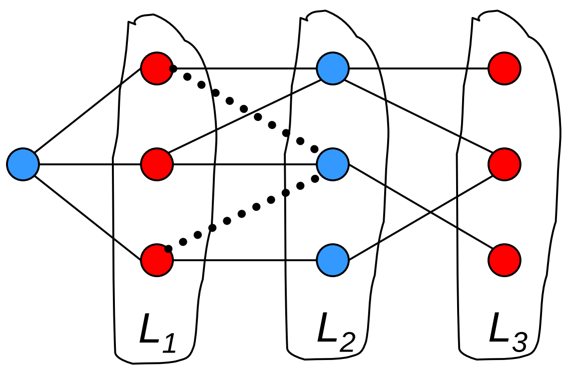
not bipartite
(not 2-colorable)

A Characterization of Bipartite Graphs

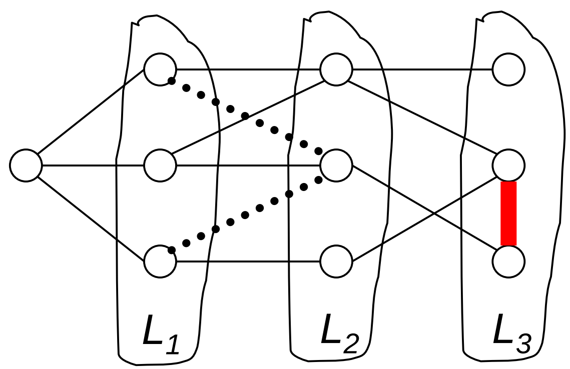
Lemma: Let G be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS(s). Exactly one of the following holds.

(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.

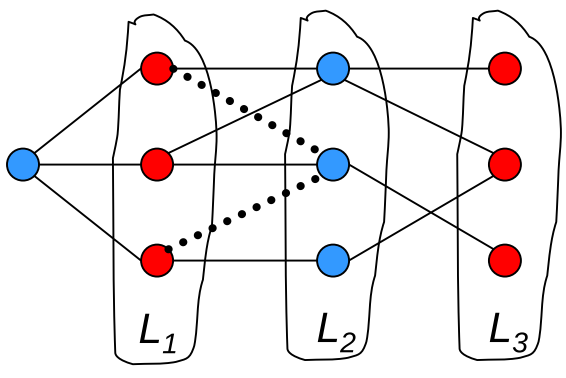
(i) No edge of G joins two nodes of the same layer, and G is bipartite.

(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)

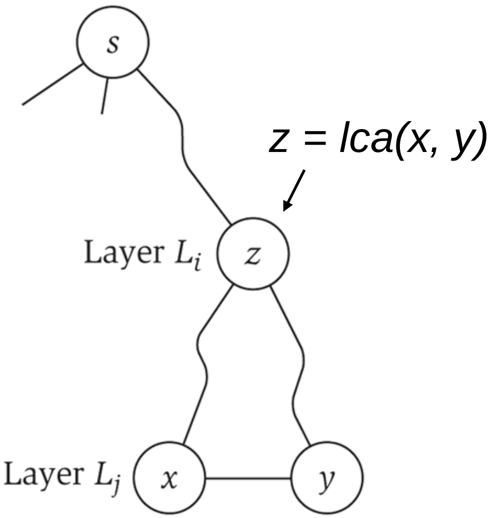
Suppose $\{x, y\}$ is an edge & x, y in same level L_j.

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.

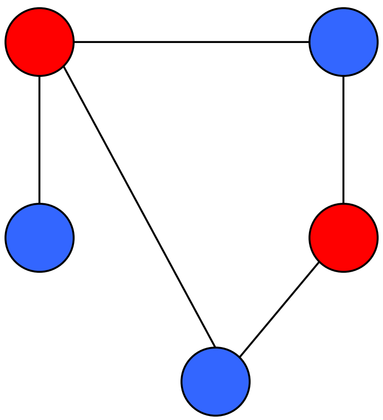
Its length is $1 + (j - i) + (j - i)$, which is odd.



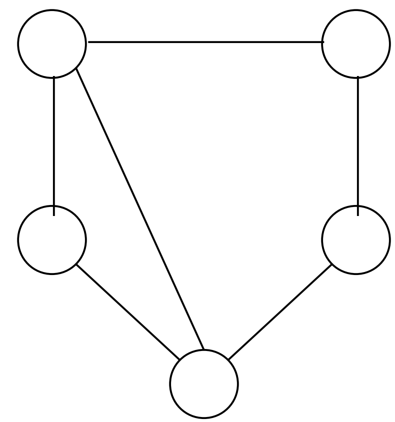
Obstruction to Bipartiteness

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)

Summary

- BFS(s) implemented using queue.
- 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 layers of the tree
- Applications:
  - Shortest Path
  - Connected component
  - Test bipartiteness / 2-coloring