ETH zürich

Introduction to Scientific Computation
Lecture 5
Fall 2020

Graphs, graph algorithms

Mikhail Usvyatsov October 28, 2020

Graph

Seven Bridges of Königsberg

Leonhard Euler, 1736

Graph

$$G :< V, E >$$

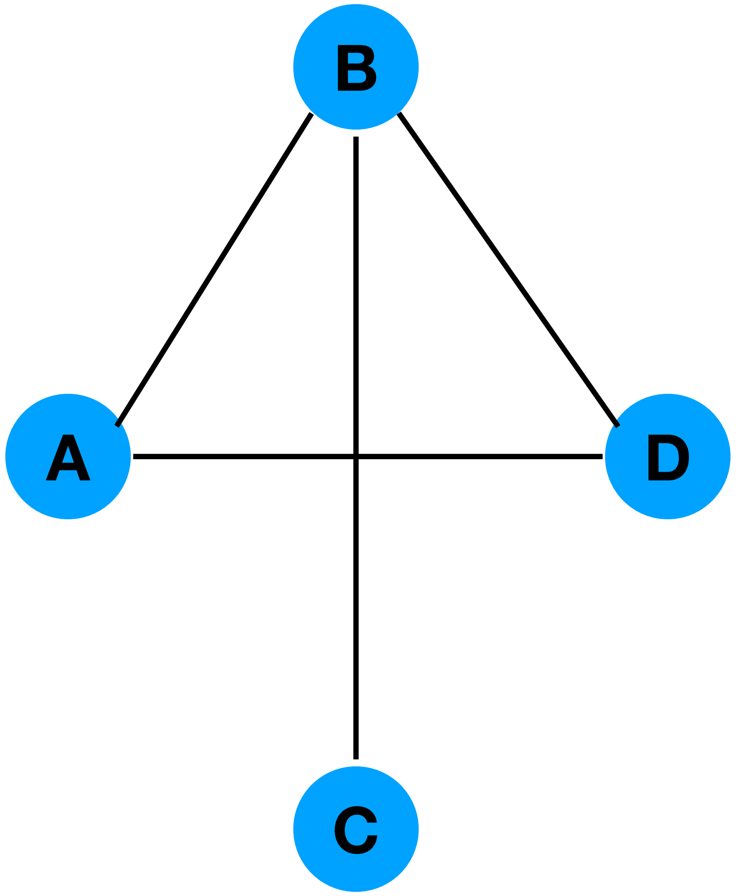
V node in the graph

E edge in the graph, the connection between two nodes

Applications

- Road networks
- Electronic circuits
- Telecommunication networks
- Social networks
- Any relationships …

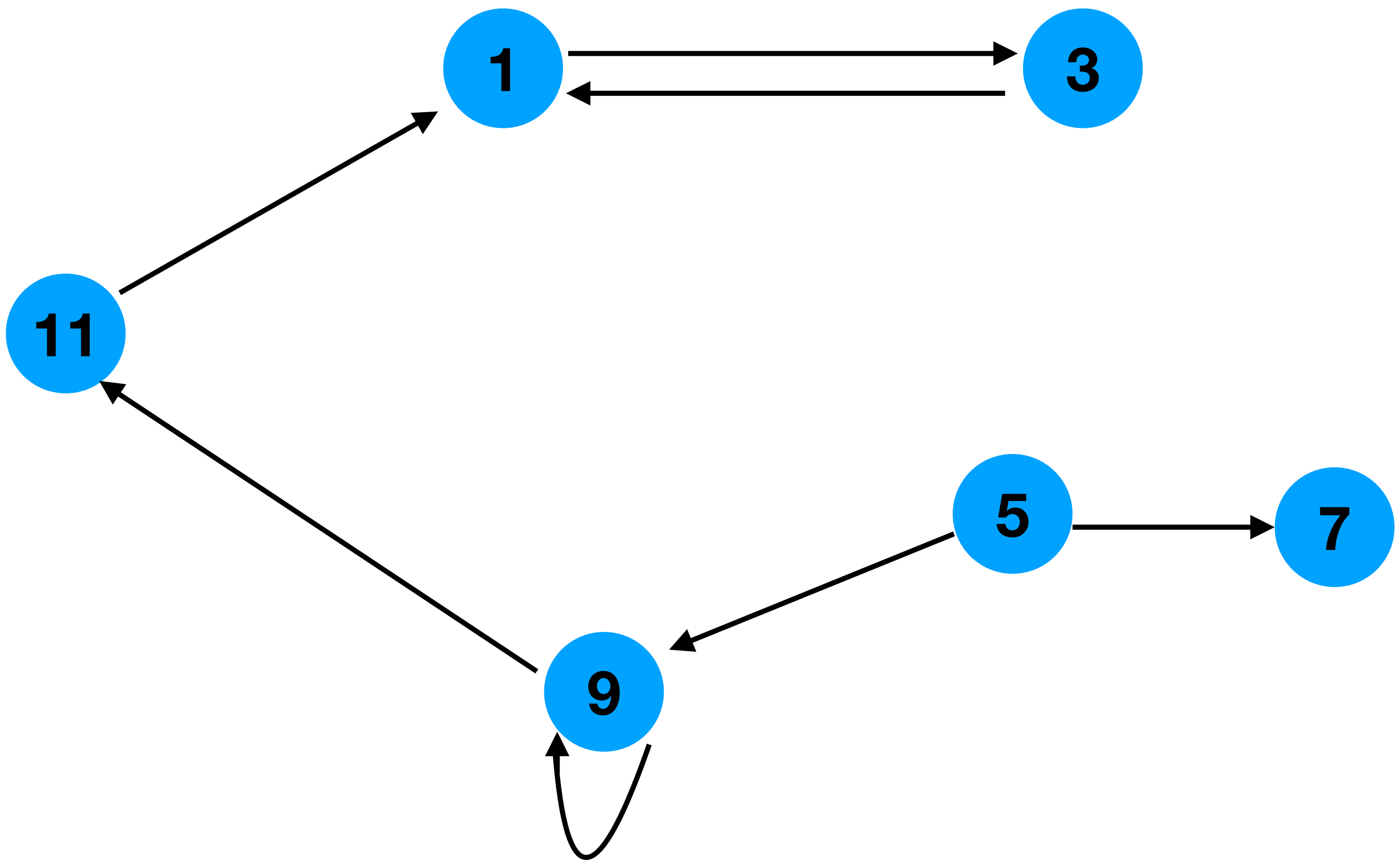
Types



Undirected G

$$V = \{A, B, C, D\}$$
$$E = \{AB, BD, AD, BC\}$$

Types



Directed G

$$V = \{1,3,5,7,9,11\}$$

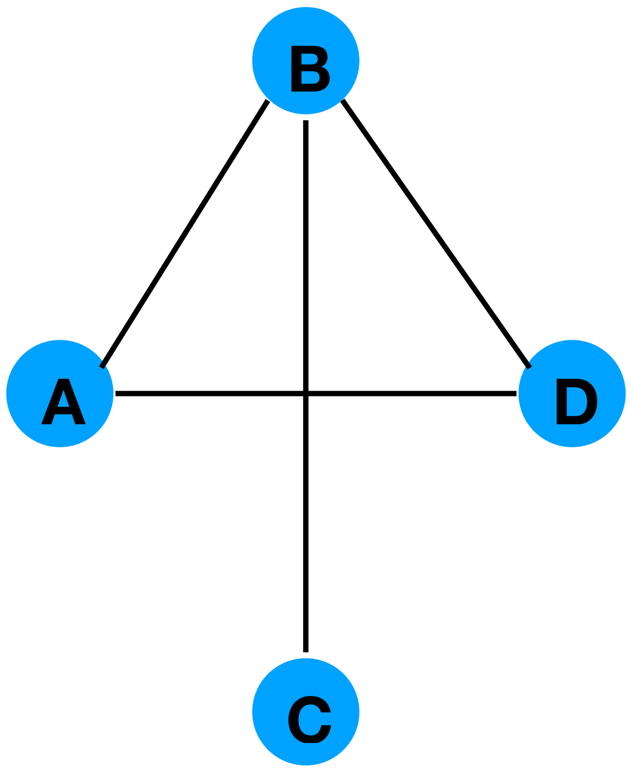
$$E = \{(1,3), (3,1), (5,7), (5,9), (5,7), (5,9), (9,9), (9,11), (11,1)\}$$

Definitions

Adjacency - two vertices are called **adjacent** if they are connected by edge

Path - the sequence of vertices which connects two nodes in a graph

Complete graph - every vertex is connected to every other vertex



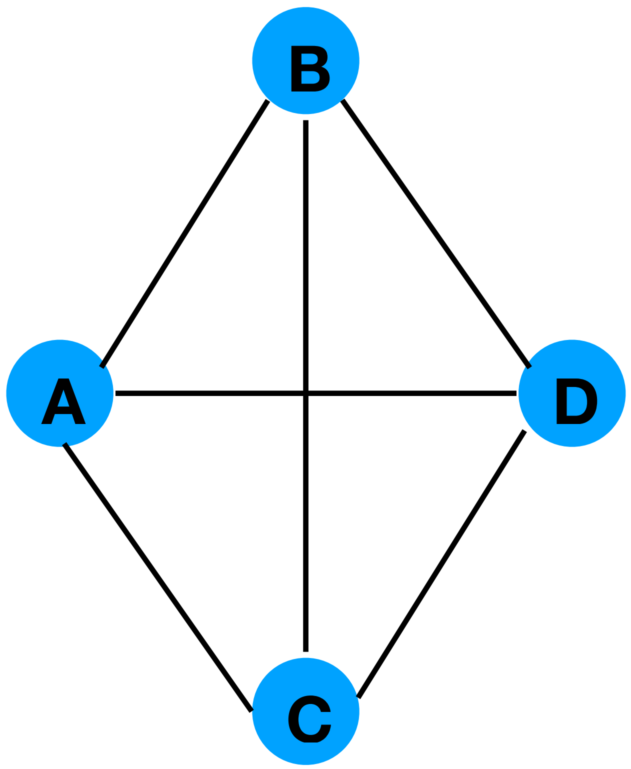
Wighted graph - graph with the values assigned to edges

Definitions

Adjacency - two vertices are called **adjacent** if they are connected by edge

Path - the sequence of vertices which connects two nodes in a graph

Complete graph - every vertex is connected to every other vertex



Wighted graph - graph with the values assigned to edges

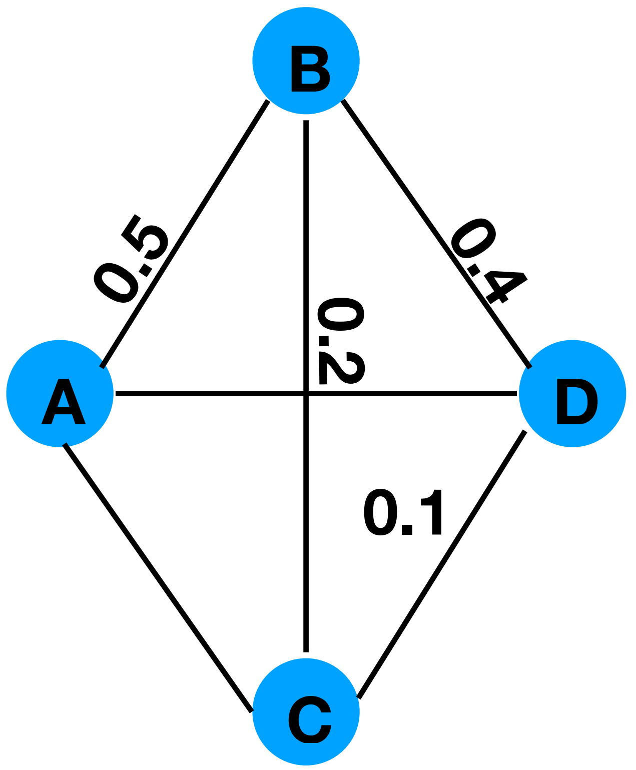
Definitions

Adjacency - two vertices are called **adjacent** if they are connected by edge

Path - the sequence of vertices which connects two nodes in a graph

Complete graph - every vertex is connected to every other vertex

Wighted graph - graph with the values assigned to edges



Definitions

Quiz: how many edges exist in a complete graph ?

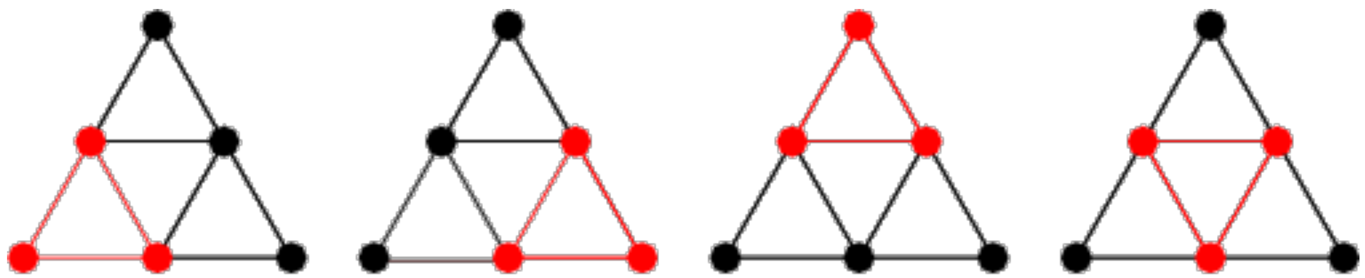
Definitions

Quiz: how many edges exist in a complete graph ?

Answer: $$\frac{N^2 - N}{2}$$

Definitions

Clique - complete subgraph


cliques

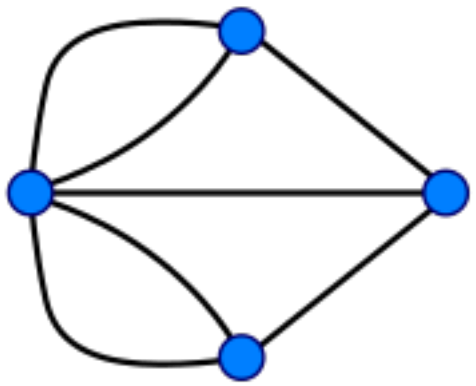
Euler trail (path) - the path in a finite graph which visits every edge exactly once

Degree $d(V)$ **-** number of edges incident to V

Definitions

Clique - complete subgraph

Euler trail (path) - the path in a finite graph which visits every edge exactly once



Degree $d(V)$ **-** number of edges incident to V

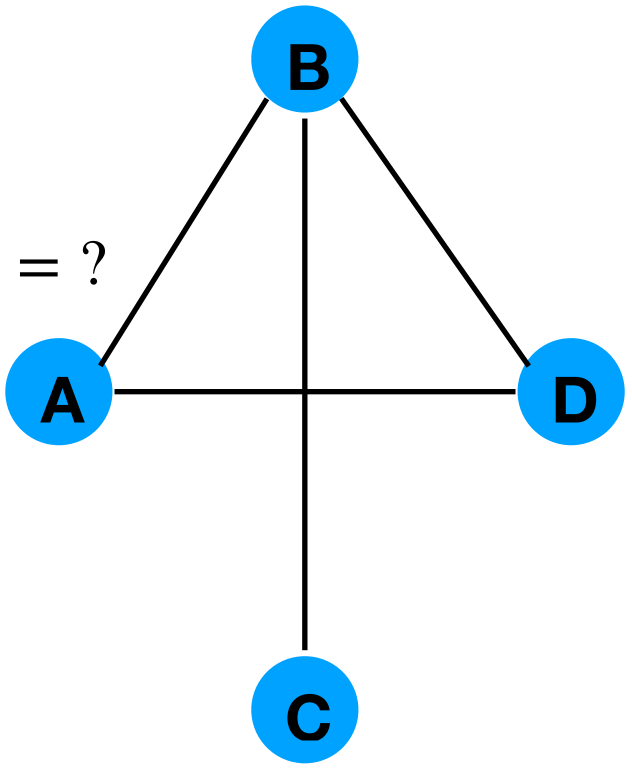
Definitions

Clique - complete subgraph

Euler trail (path) - the path in a finite graph which visits every edge exactly once

Degree $d(V)$ **-** number of edges incident to V

$d(A) = ?$

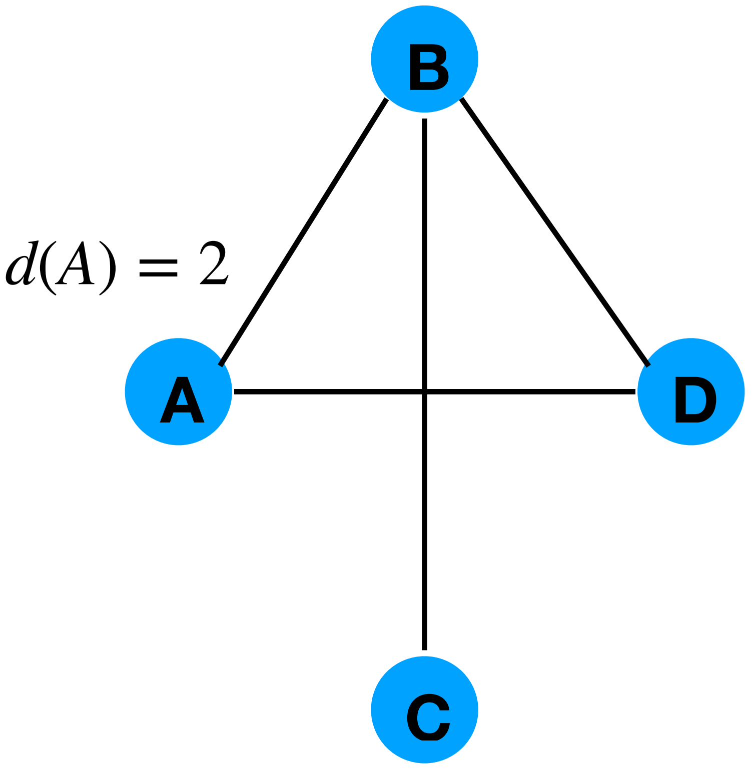


Definitions

Clique - complete subgraph

Euler trail (path) - the path in a finite graph which visits every edge exactly once

Degree $d(V)$ **-** number of edges incident to V



Definitions

Handshaking lemma

$$G = < V, E >$$

$$\sum_{u \in V} d(u) = 2|E|$$

in a party of people some of whom shake hands, an even number of people must have shaken an odd number of other people's hands.

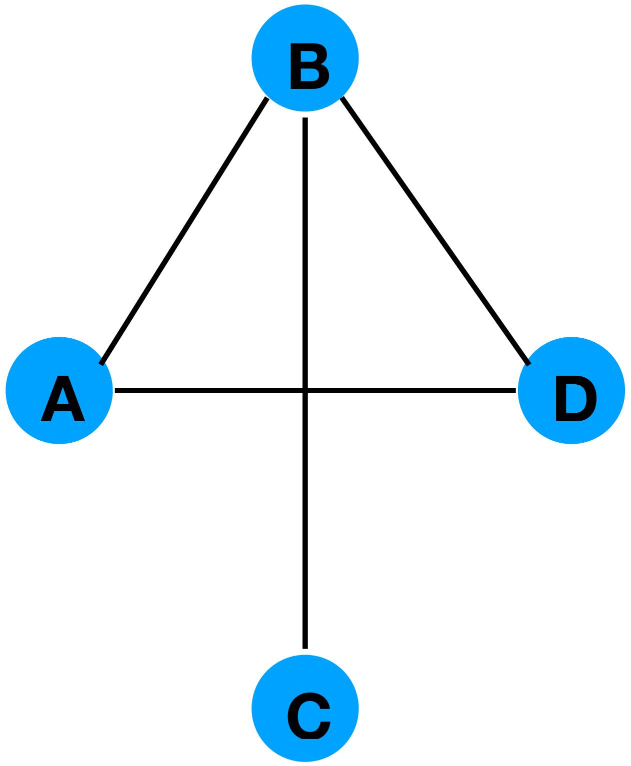
Graph

Back to Seven Bridges of Königsberg

An undirected graph has an Euler path if and only if exactly zero or two vertices have odd degree, and all of its vertices with nonzero degree belong to a single connected component.

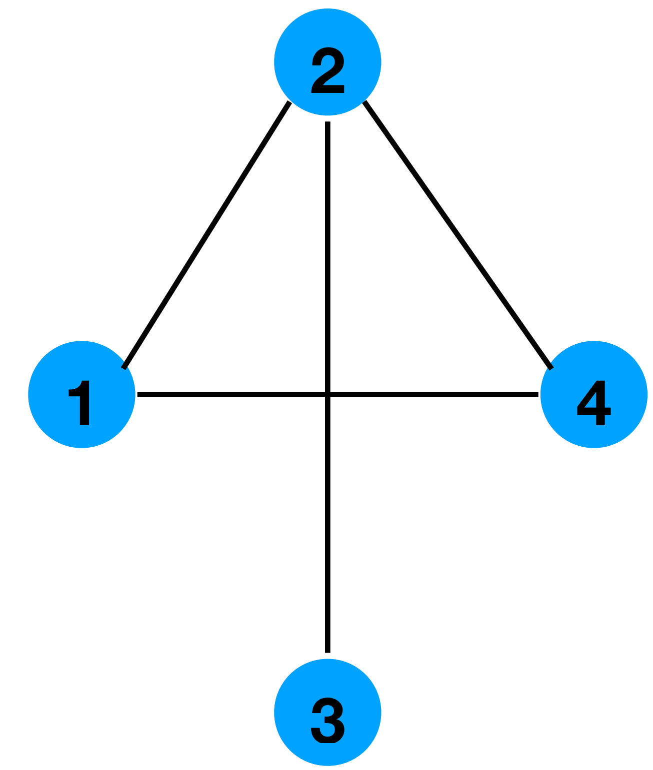
Leonhard Euler, 1736

CS representation



CS representation

$$G = < V, E >$$



Adjacency matrix

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

Adjacency list

1 → 2 → 4

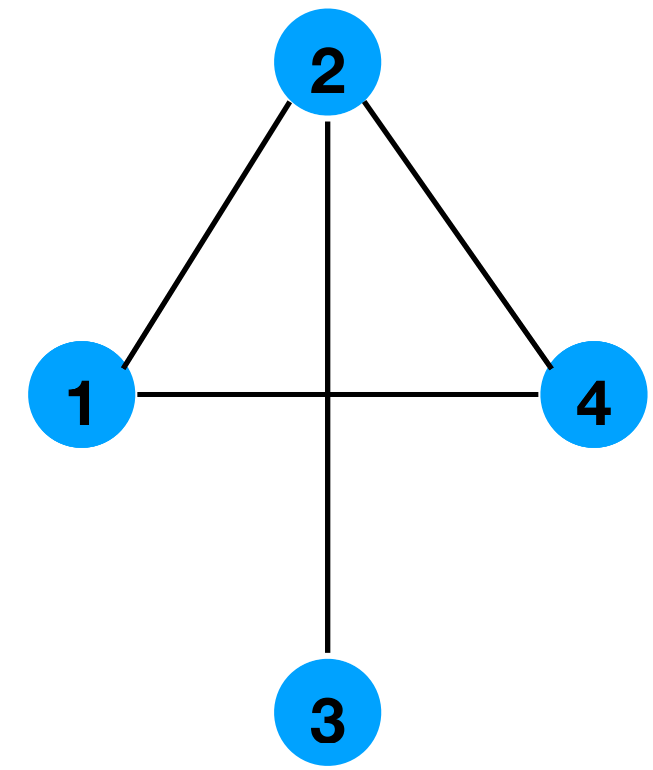
2 → 1 → 4

3 → 2

4 → 1 → 2

CS representation

$G = < V, E >$

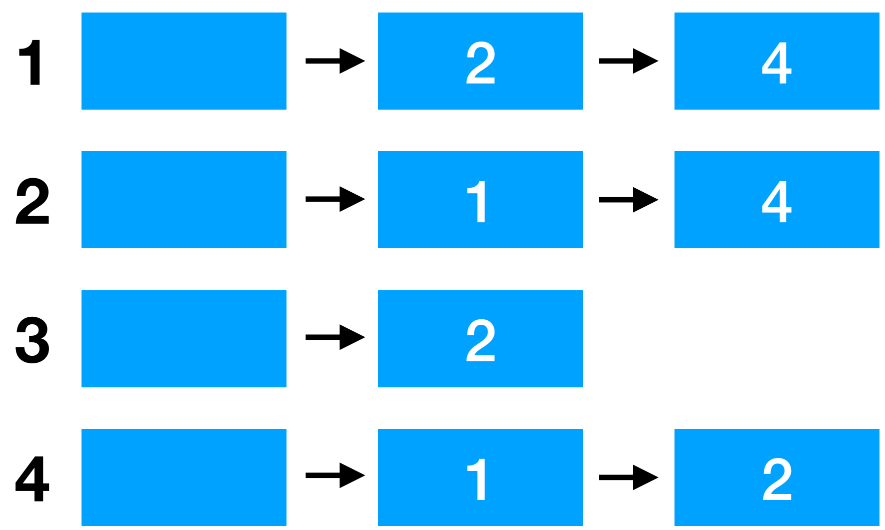


Adjacency matrix

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

Directed: N^2
Undirected: N^2

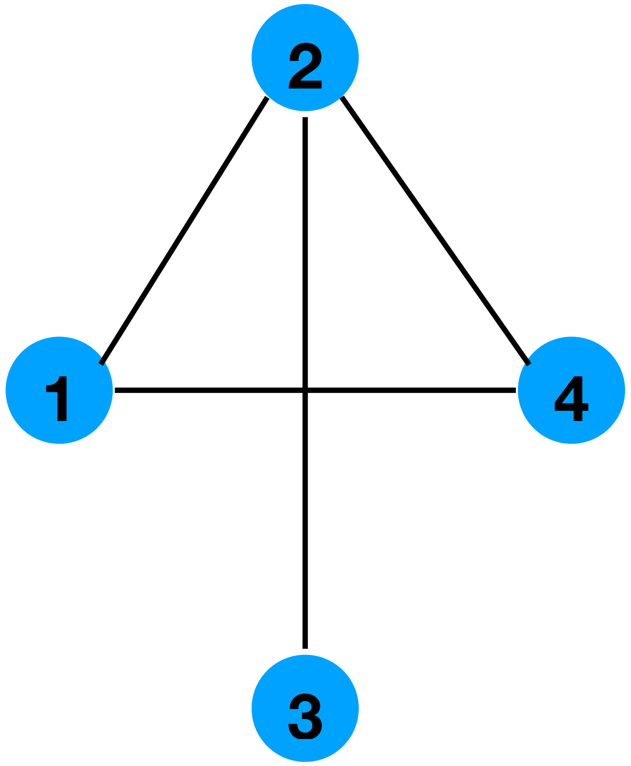
Adjacency list



Directed: $N + M$
Undirected: $N + 2M$

CS representation

$G = < V, E >$



Adjacency matrix

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

Adjacency list

1 → 2 → 4

2 → 1 → 4

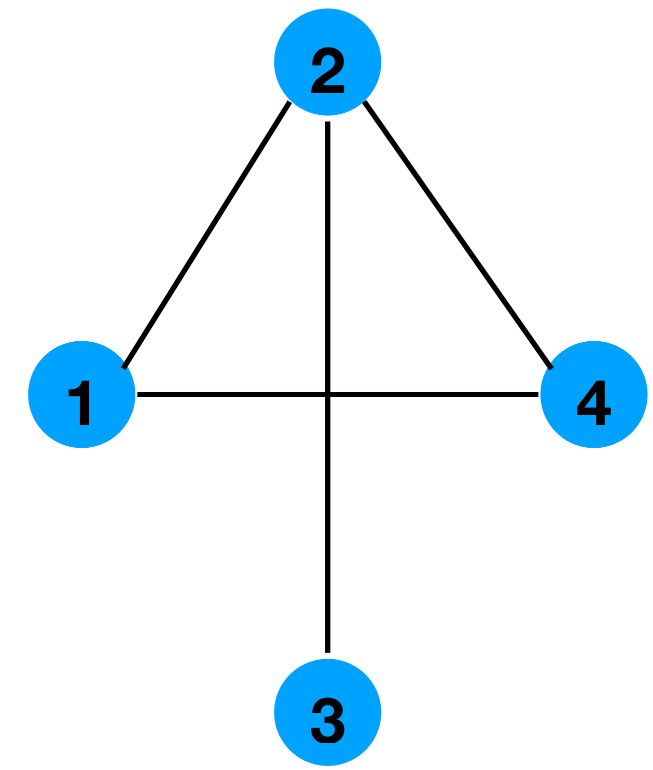
3 → 2

4 → 1 → 2

Quiz: What is better to test if an edge is in the graph ?

CS representation

$G = < V, E >$



Adjacency matrix

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

Adjacency list

1 → 2 → 4

2 → 1 → 4

3 → 2

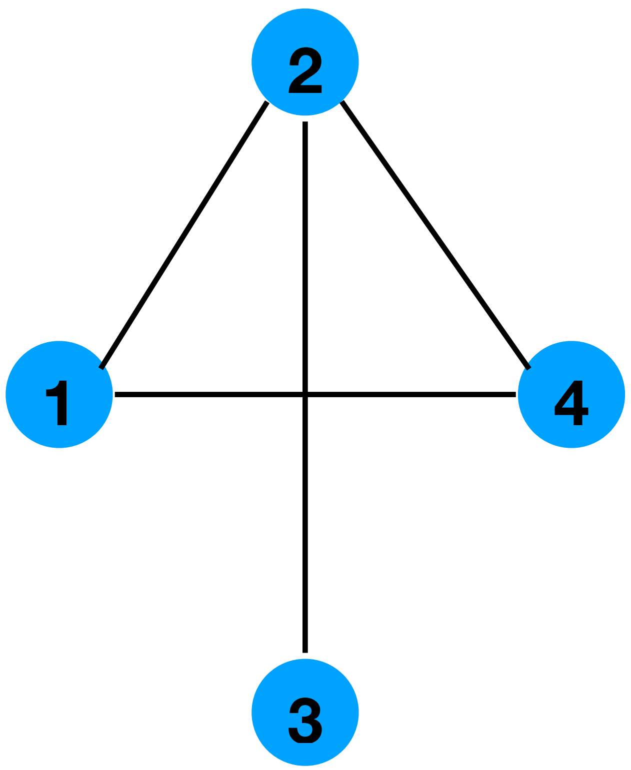
4 → 1 → 2

Quiz: What is better to test if an edge is in the graph ?

Answer: matrix

CS representation

$G = < V, E >$



Adjacency matrix

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

Adjacency list

1 [ ] → 2 → 4

2 [ ] → 1 → 4

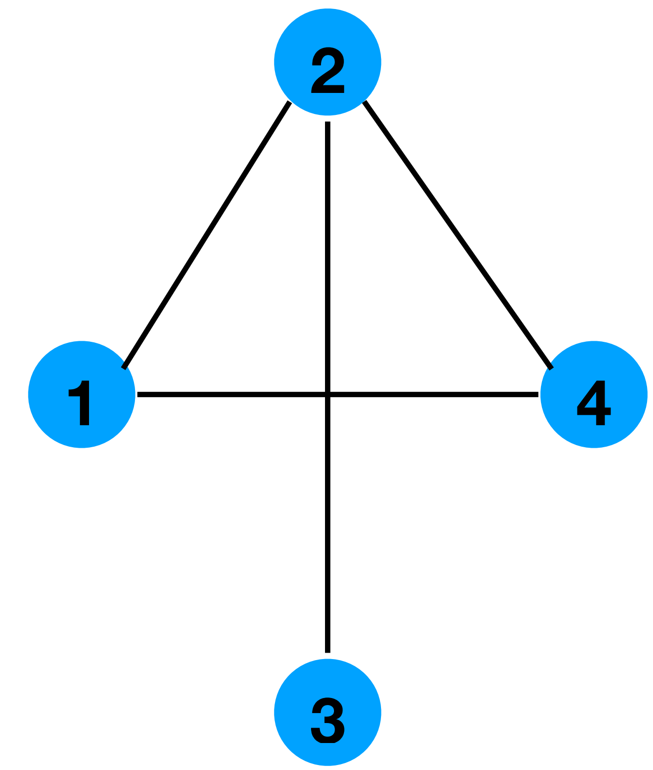
3 [ ] → 2

4 [ ] → 1 → 2

Quiz: What is faster to find the degree of vertex ?

CS representation

$G = < V, E >$



Adjacency matrix

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

Adjacency list

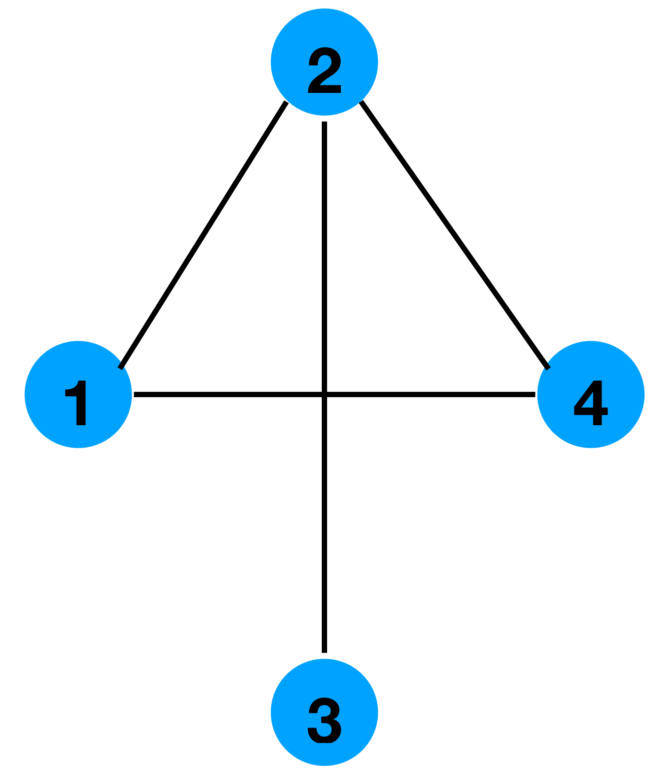
1 → 2 → 4
2 → 1 → 4
3 → 2
4 → 1 → 2

Quiz: What is better to test if an edge is in the graph ?

Answer: list

CS representation

$G = < V, E >$



Adjacency matrix

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

Adjacency list

1 → 2 → 4

2 → 1 → 4

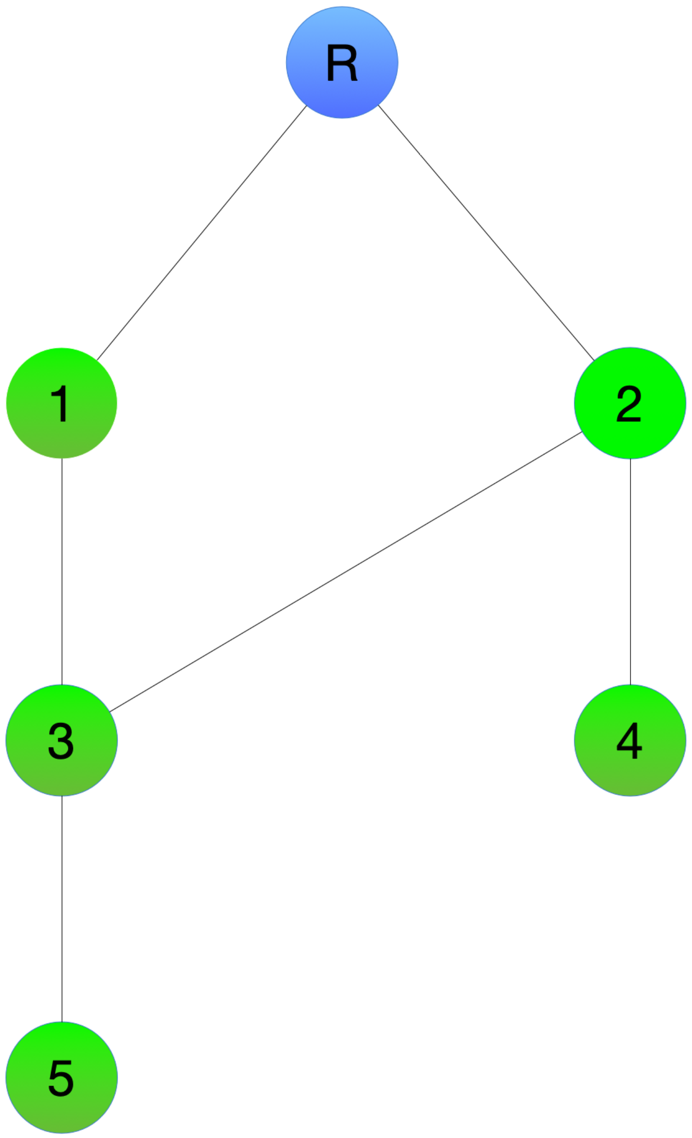
3 → 2

4 → 1 → 2

$M + N$ **VS** N^2

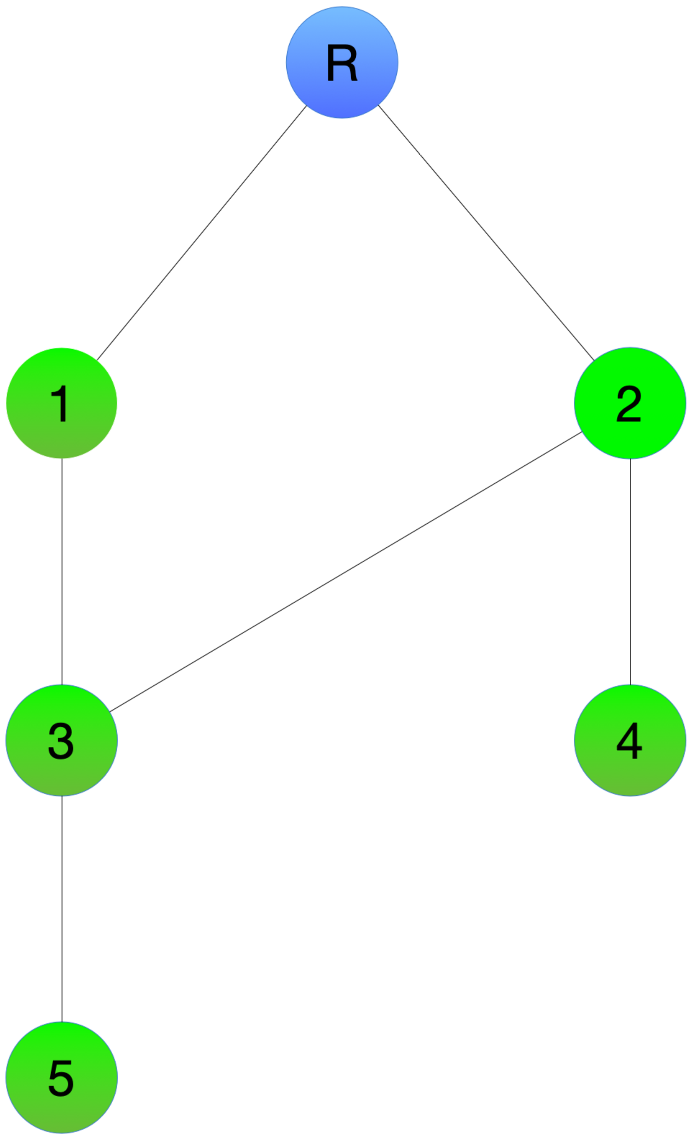
Graph traversals

- Breadth-First Search (BFS)
- Depth-First Search (DFS)



Breadth-First Search (BFS)

Main Idea: mark each vertex when we first visit it & monitor completely unexplored items.



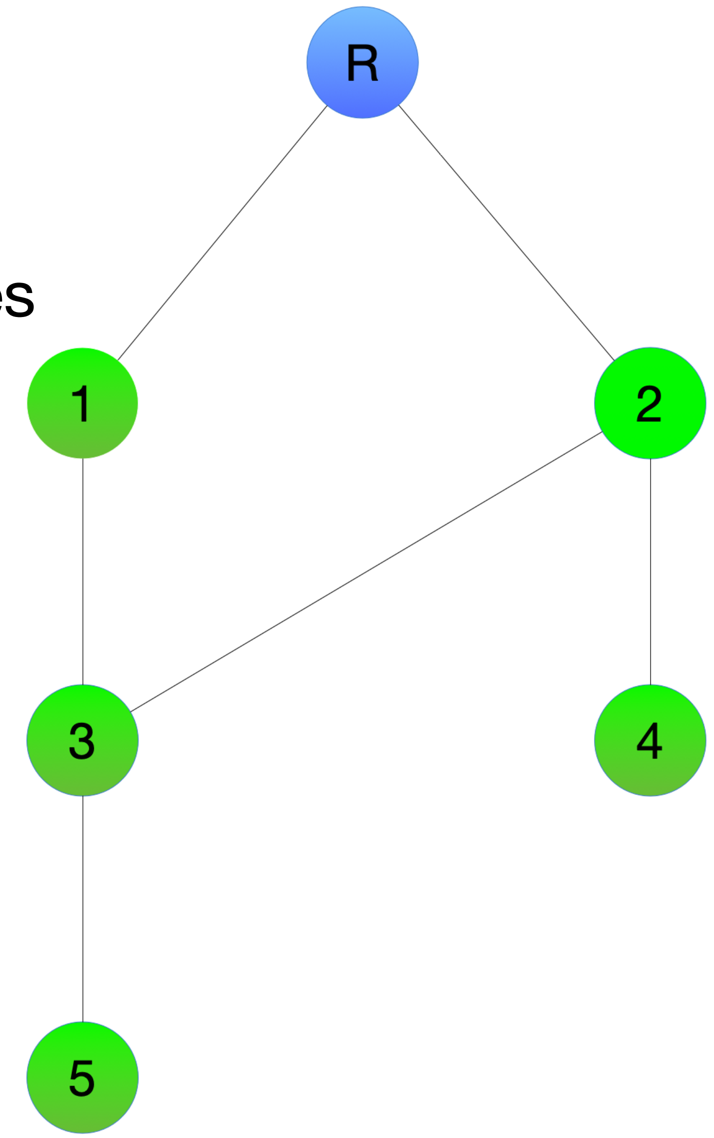
Breadth-First Search (BFS)

Main Idea: mark each vertex when we first visit it & monitor completely unexplored items.

Each vertex can exist in one of three states:

- Undiscovered - the node has never been visited
- Discovered - the node has been found, but not all its edges are visited
- Processed - all adjacent edges of the node are visited

Container to be used - queue.



Breadth-First Search (BFS)

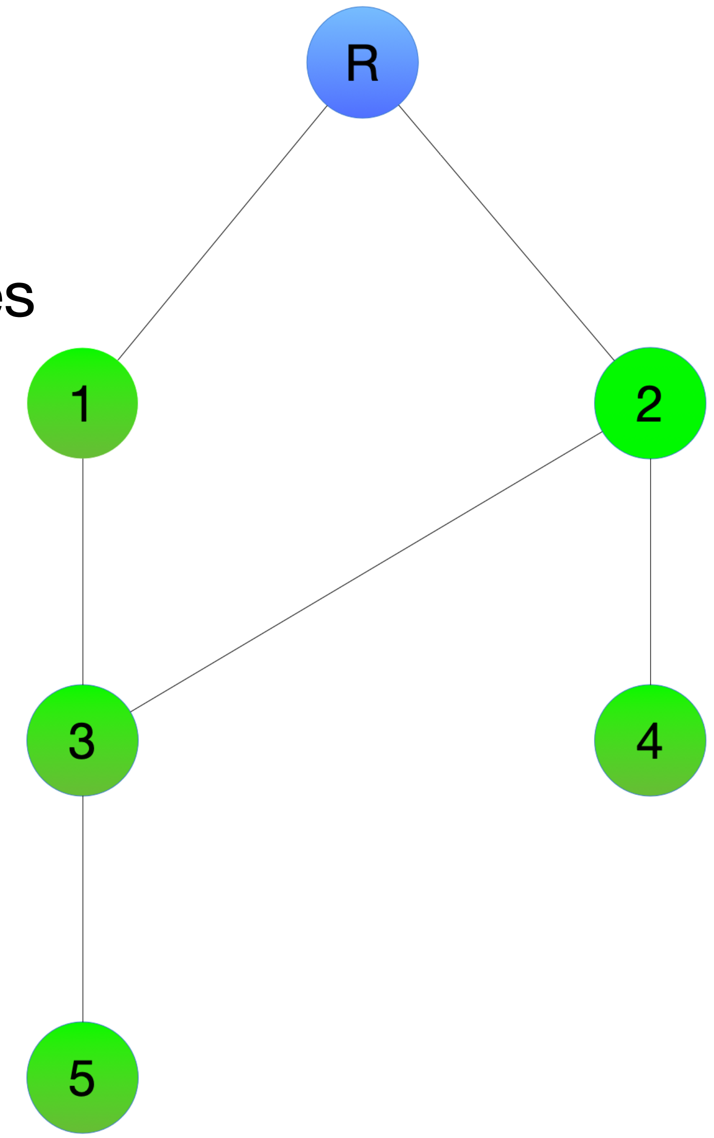
Main Idea: mark each vertex when we first visit it & monitor completely unexplored items.

Each vertex can exist in one of three states:

- Undiscovered - the node has never been visited
- Discovered - the node has been found, but not all its edges are visited
- Processed - all adjacent edges of the node are visited

Container to be used - queue.

Quiz: What is the complexity of BFS ?



Breadth-First Search (BFS)

Main Idea: mark each vertex when we first visit it & monitor completely unexplored items.

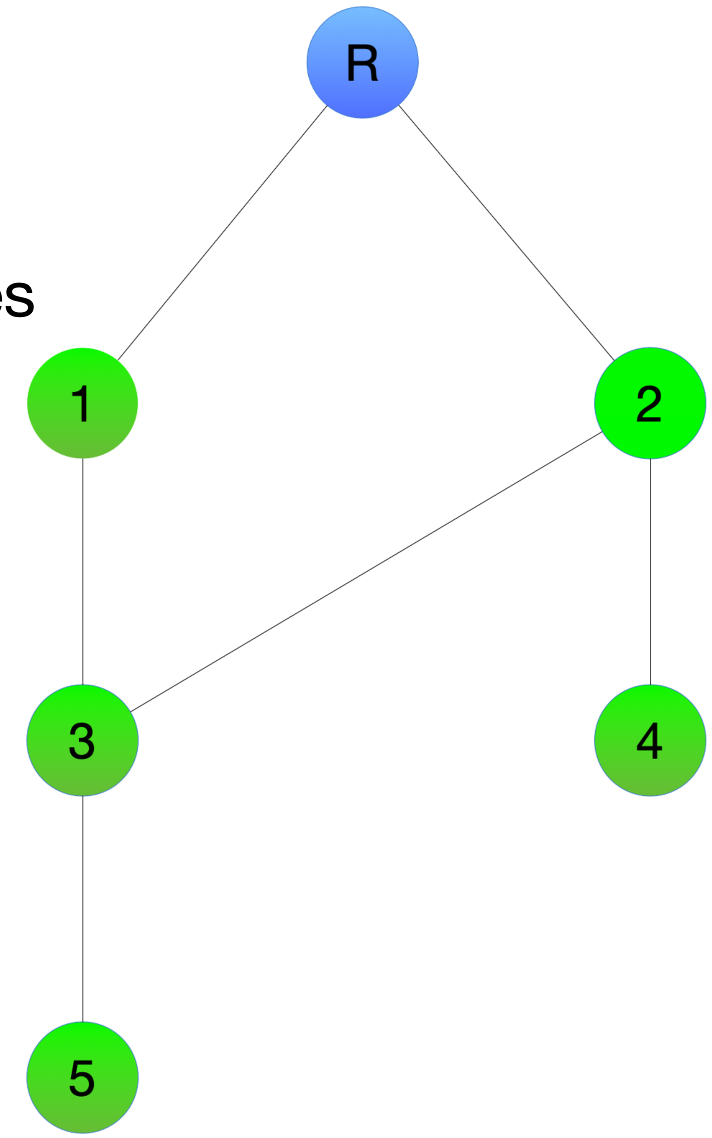
Each vertex can exist in one of three states:

- Undiscovered - the node has never been visited
- Discovered - the node has been found, but not all its edges are visited
- Processed - all adjacent edges of the node are visited

Container to be used - queue.

Quiz: What is the complexity of BFS ?

Answer: $O(V+E)$

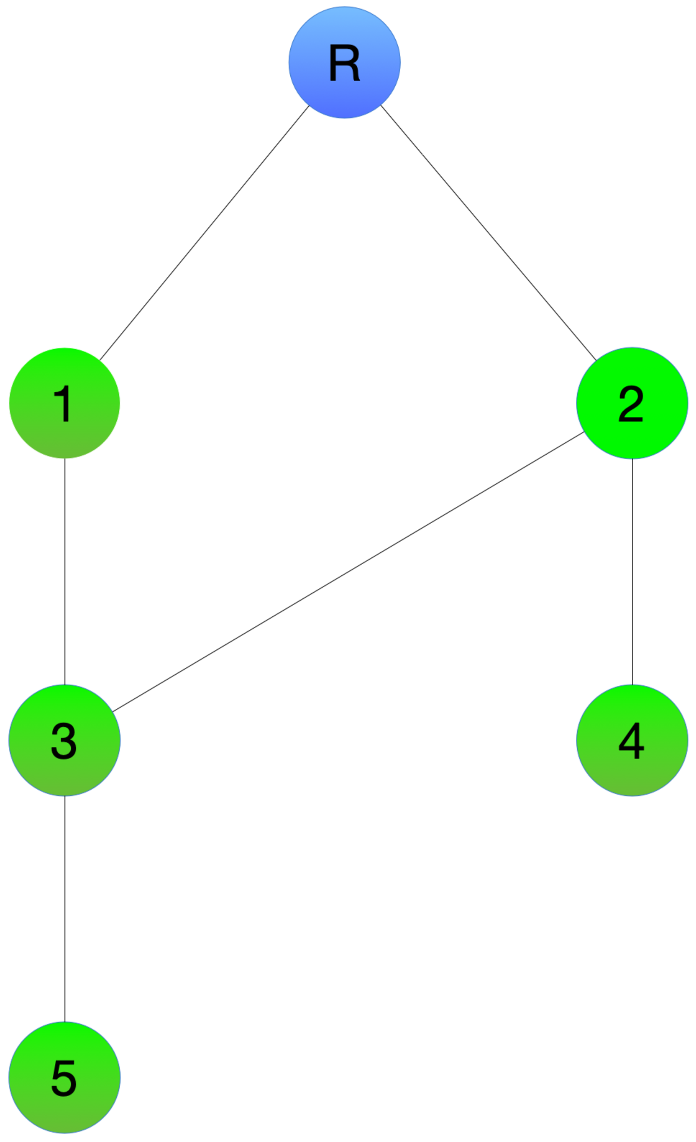


Depth-First Search (DFS)

Main Idea: start DFS from every next node until there are nodes which are not visited.

Each vertex can exist in one of two states:

- Undiscovered - the node has never been visited
- Discovered - the node has been visited



Container to be used - stack (recursion).

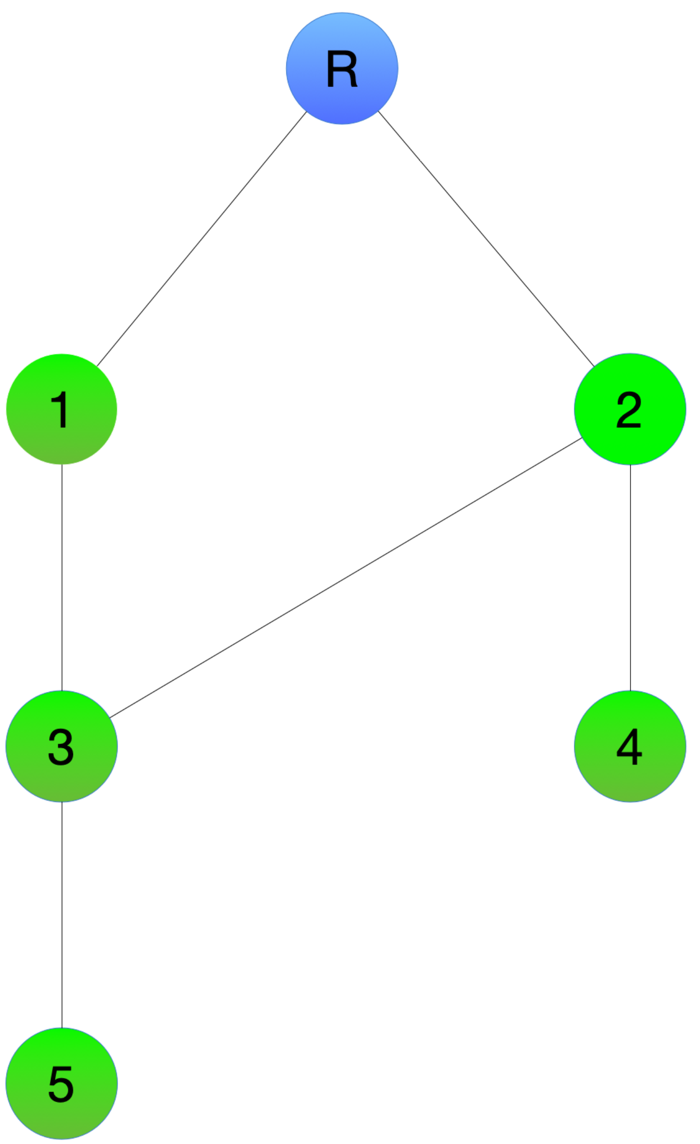
Quiz: What is the complexity of DFS ?

Depth-First Search (DFS)

Main Idea: start DFS from every next node until there are nodes which are not visited.

Each vertex can exist in one of two states:

- Undiscovered - the node has never been visited
- Discovered - the node has been visited



Container to be used - stack (recursion).

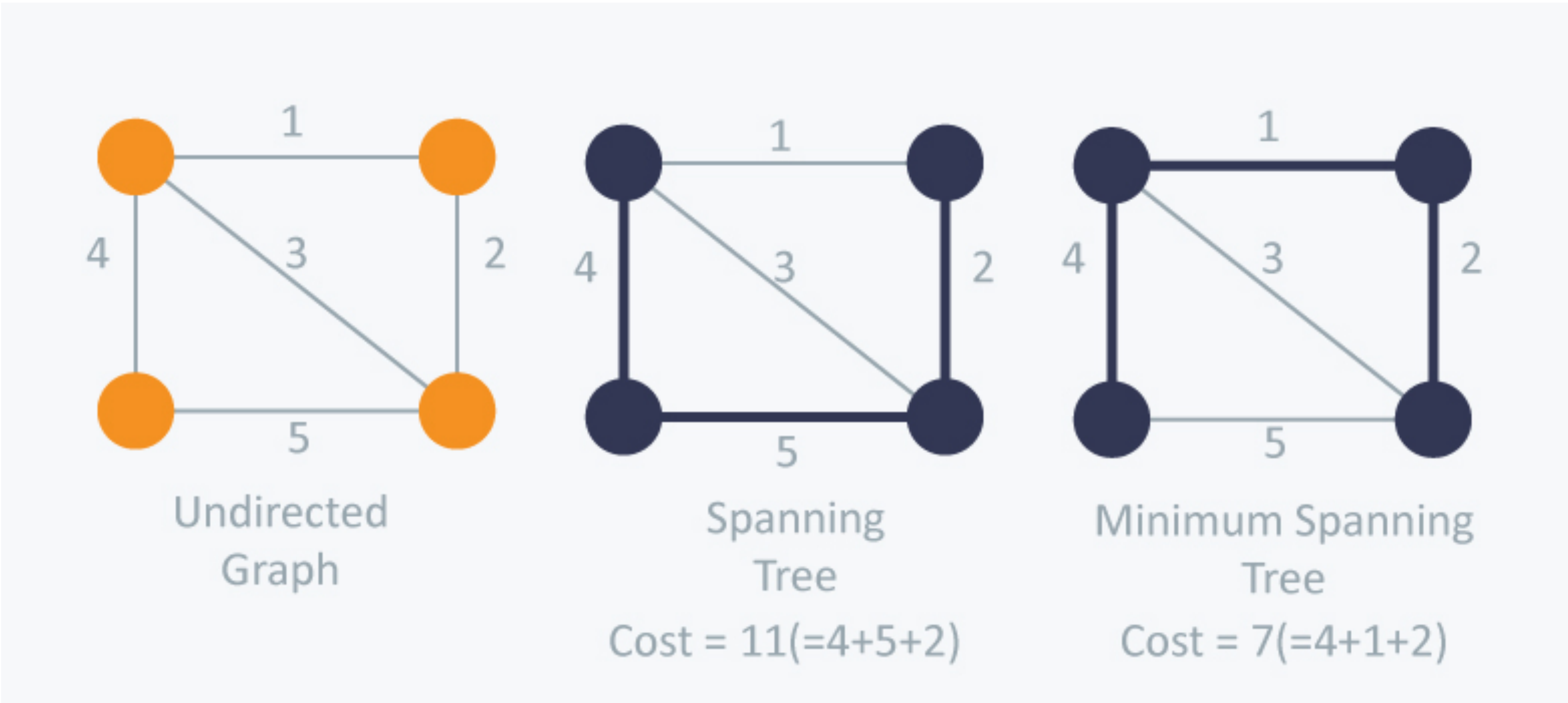
Quiz: What is the complexity of DFS ?

Answer: $O(V + E)$

Minimum Spanning Tree

Spanning tree: for graph **G = (V, E)** is a subset of edges from **E**, forming a tree connecting all vertices of **V**.

For edge-waited graph we are usually interested in minimum value spanning tree



Minimum Spanning Tree

Spanning tree: for graph **G = (V, E)** is a subset of edges from **E**, forming a tree connecting all vertices of **V**.

For edge-waited graph we are usually interested in minimum value spanning tree

Minimum Spanning Tree

Spanning tree: for graph **G = (V, E)** is a subset of edges from **E**, forming a tree connecting all vertices of **V**.

For edge-waited graph we are usually interested in minimum value spanning tree

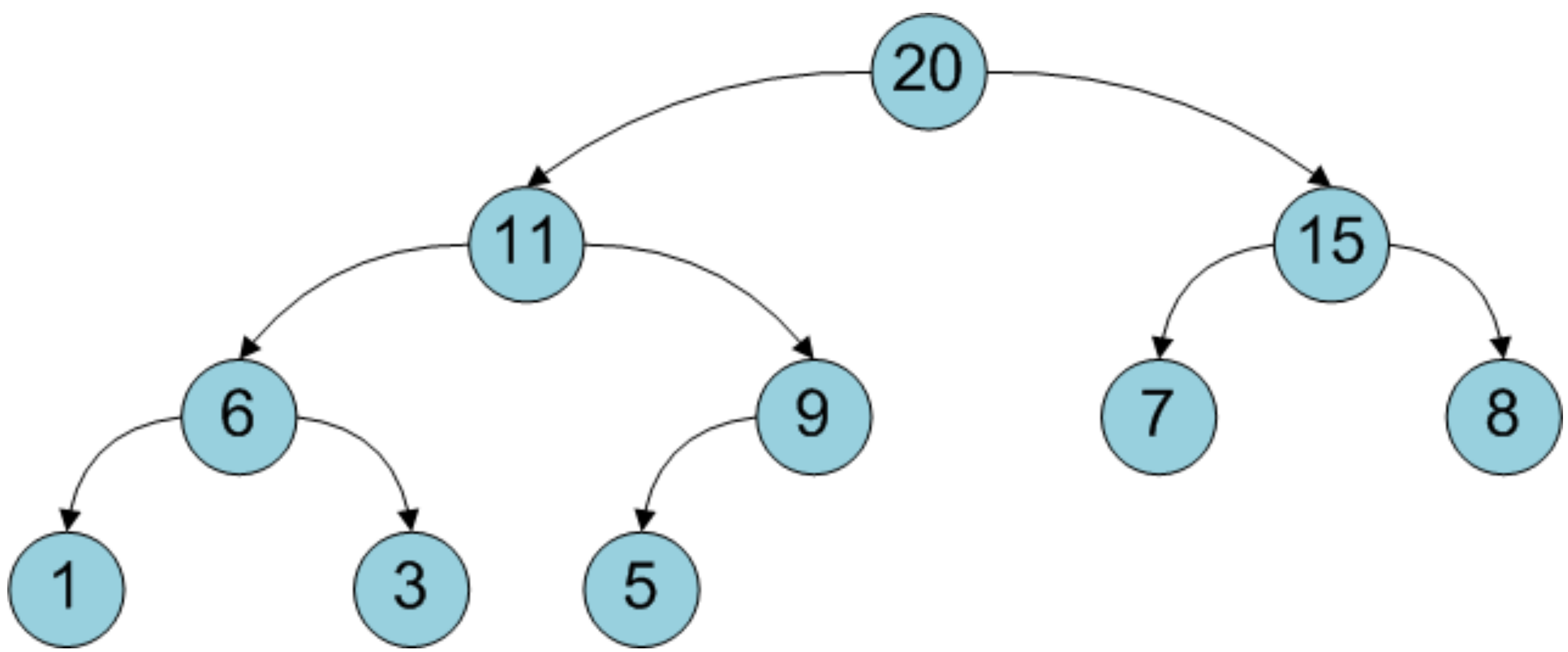
Two main algorithms:
- Prim
- Kruskal

- Minimum spanning tree minimises total length (weight) over all possible spanning trees.
- There could be more than one minimum spanning tree in the graph.

Algorithms are greedy

Heap

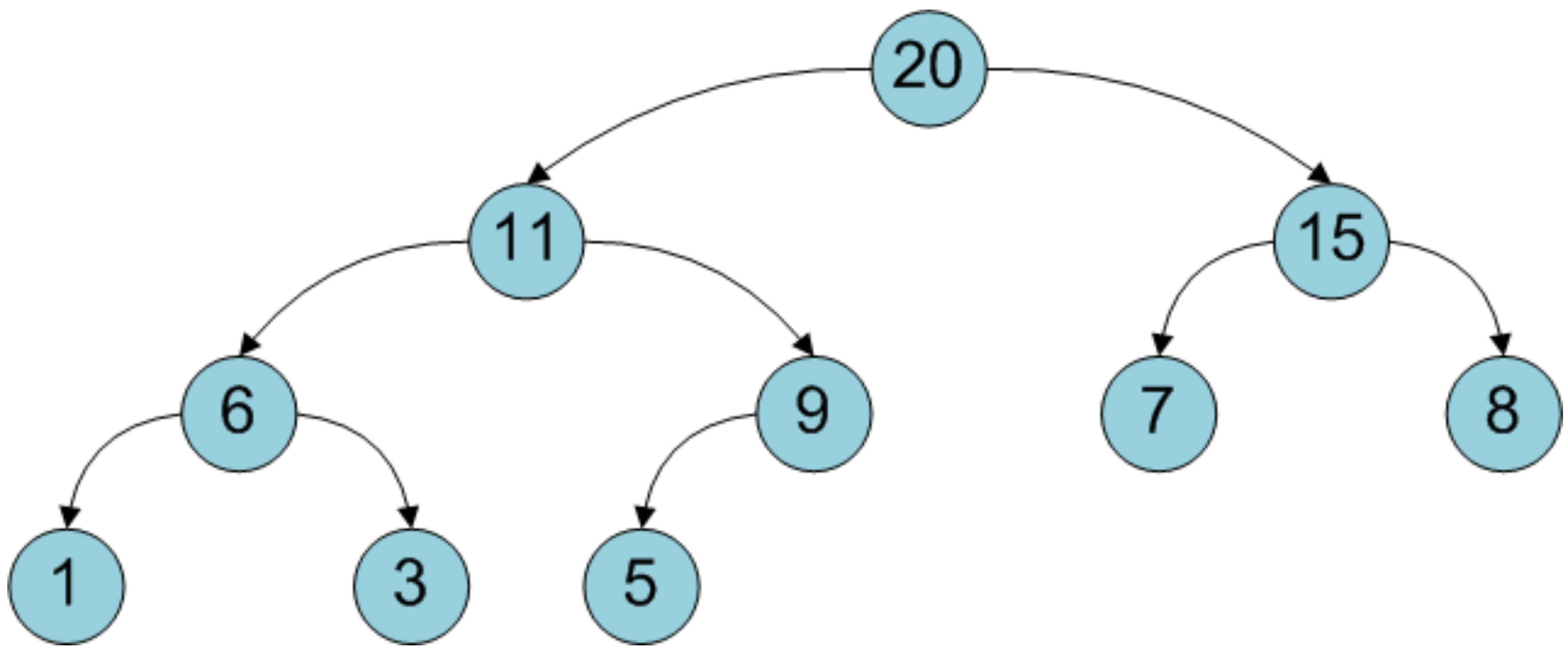
Def: is a binary tree with the property value of each vertex is less than the value of its ancestors



Heap

Def: is a binary tree with the property value of each vertex is greater than the value of its ancestors

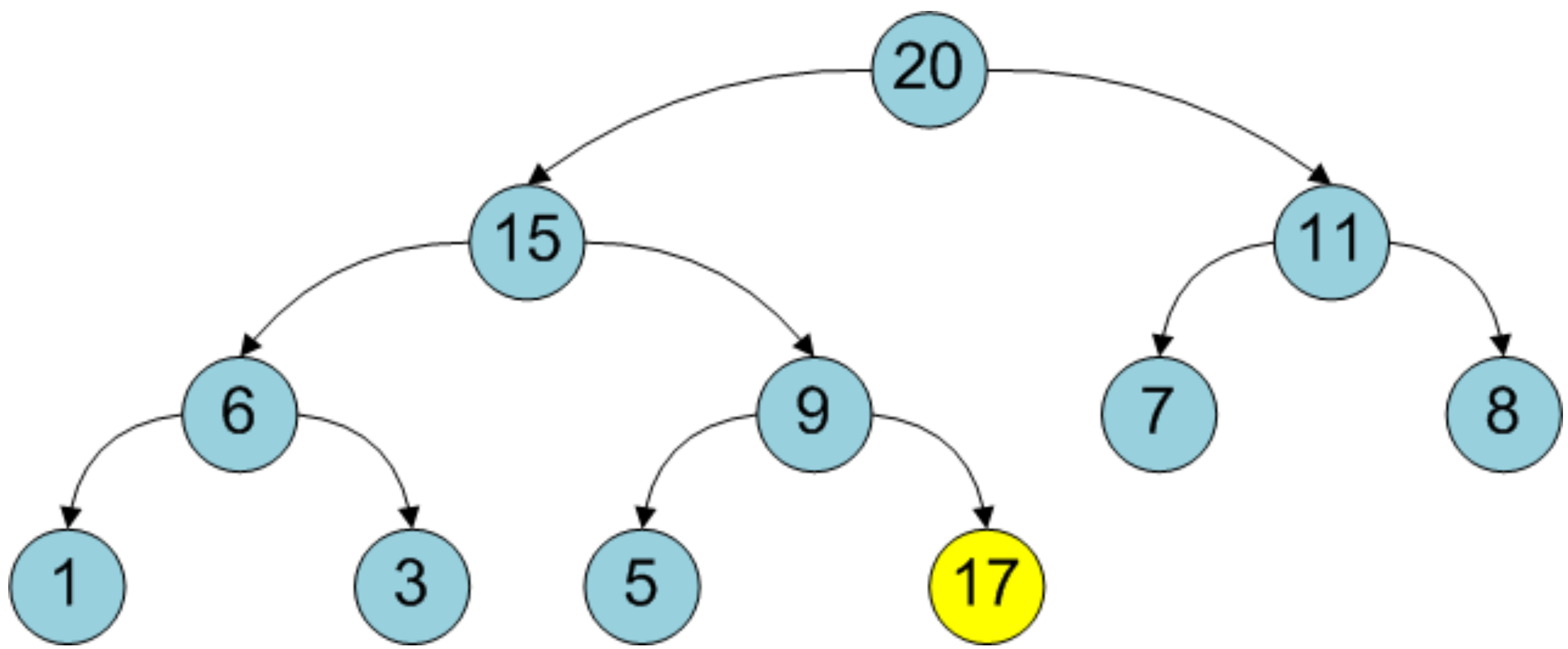
One can store entire data structure in an array. Root element is in index 0, left child of vertex I would have an index 2 * i + 1, while the right one would have an index 2 * i + 2



Heap

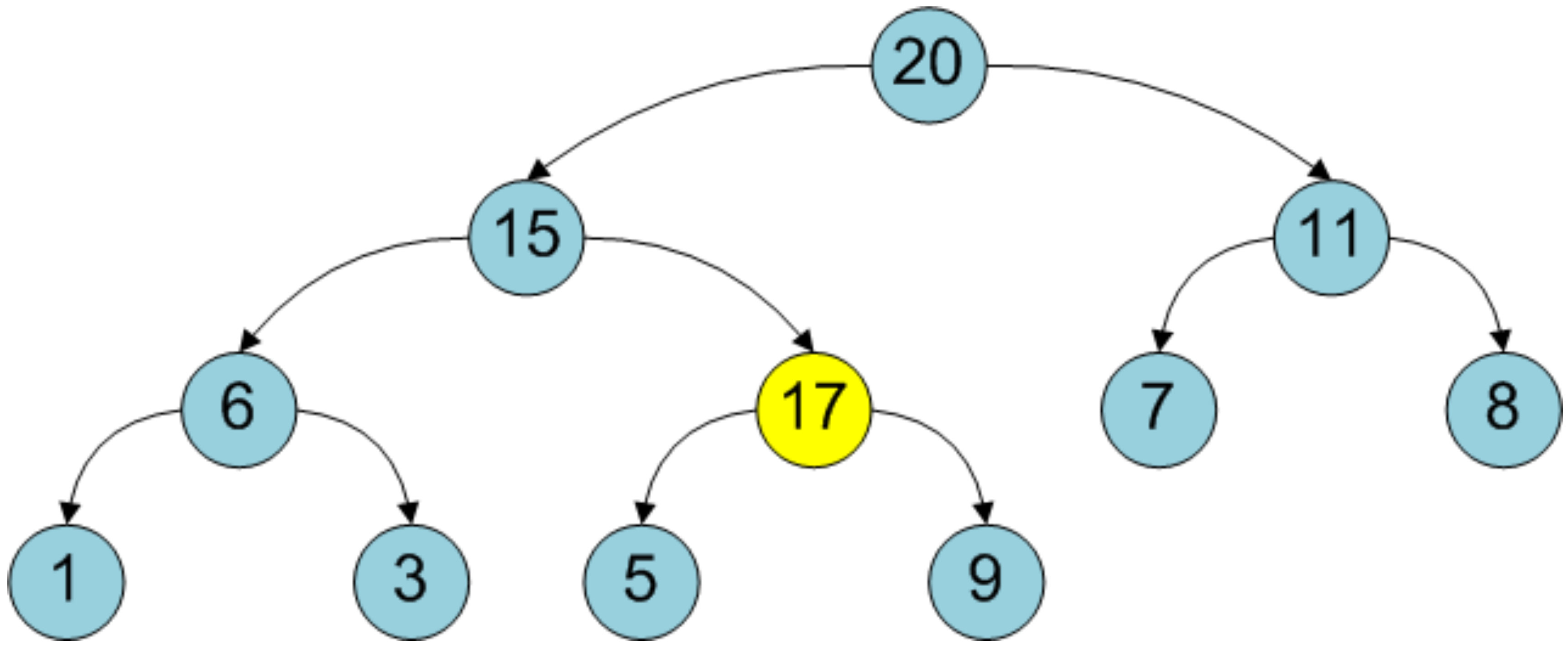
Push

Add element to the end of the array. Then fix the property of the heap.



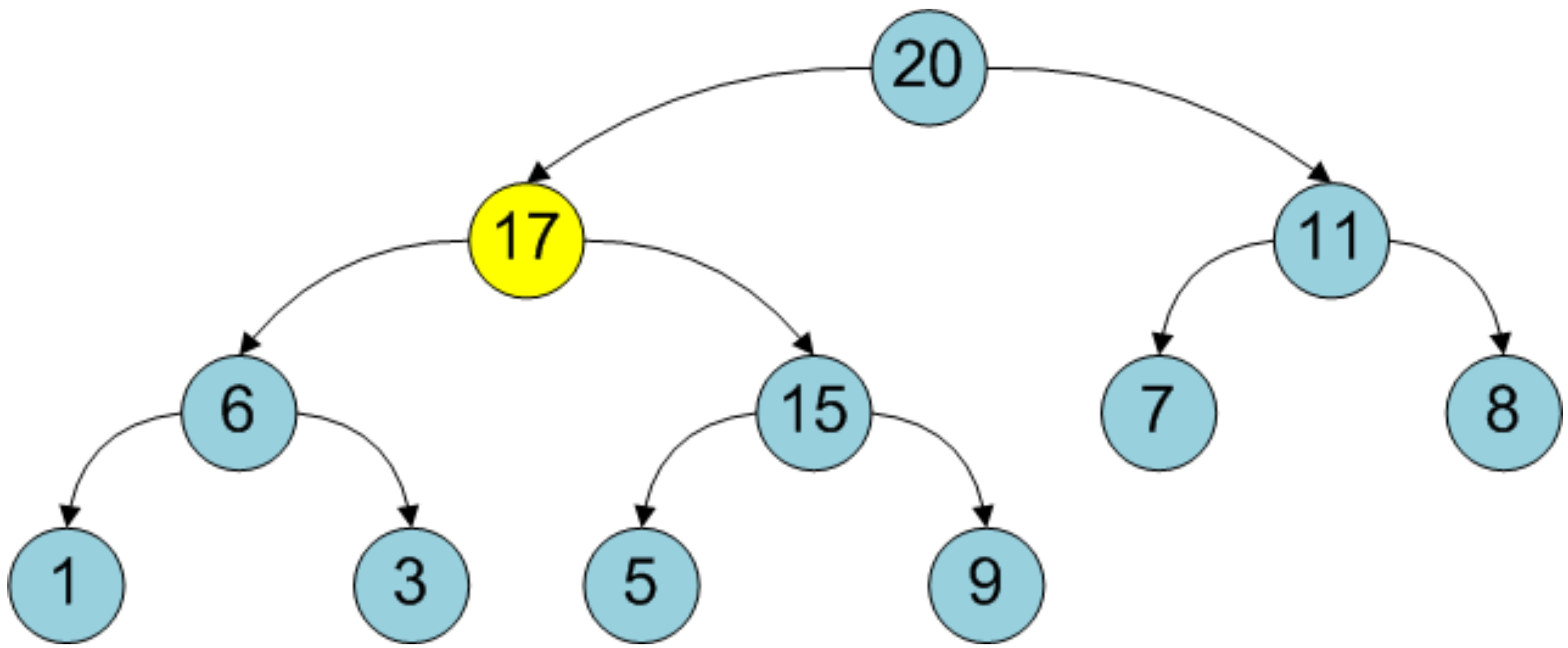
Heap

Push



Heap

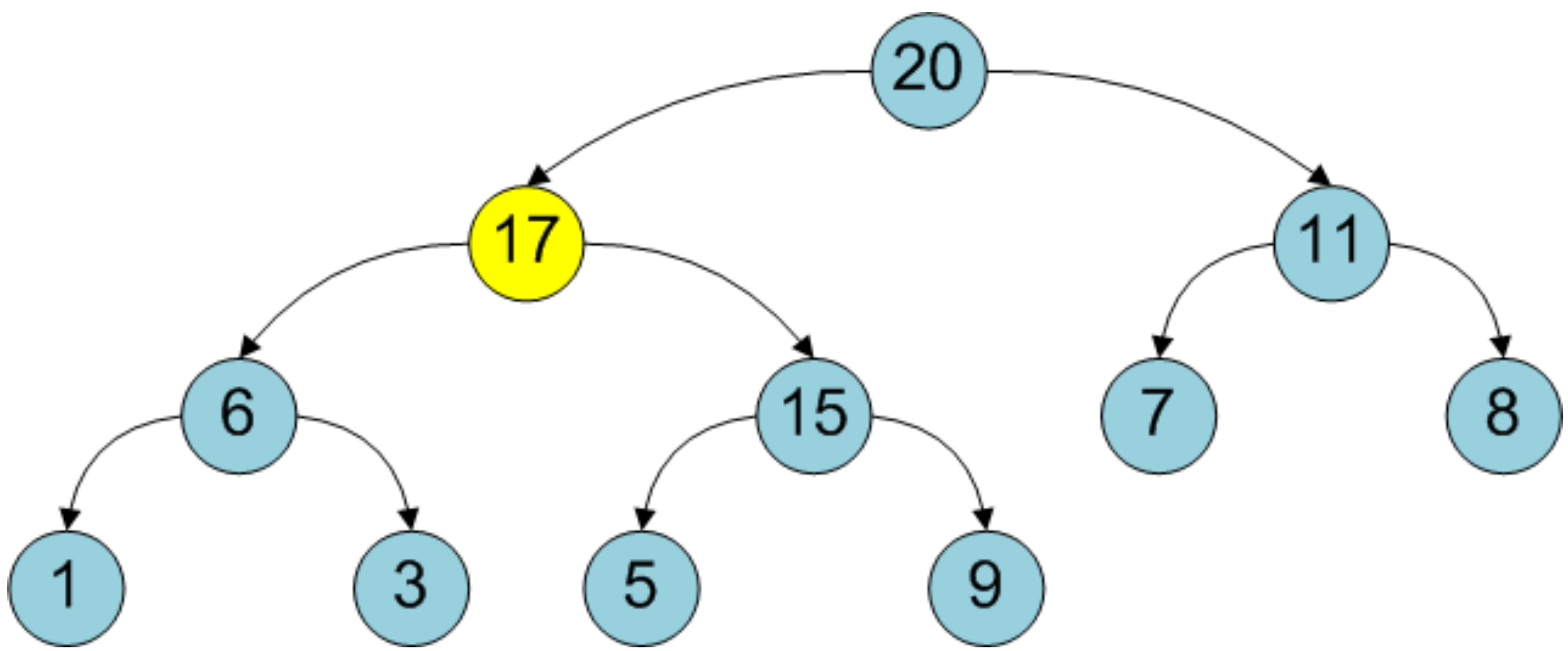
Push



Quiz: What is the complexity of adding an element in the heap ?

Heap

Push

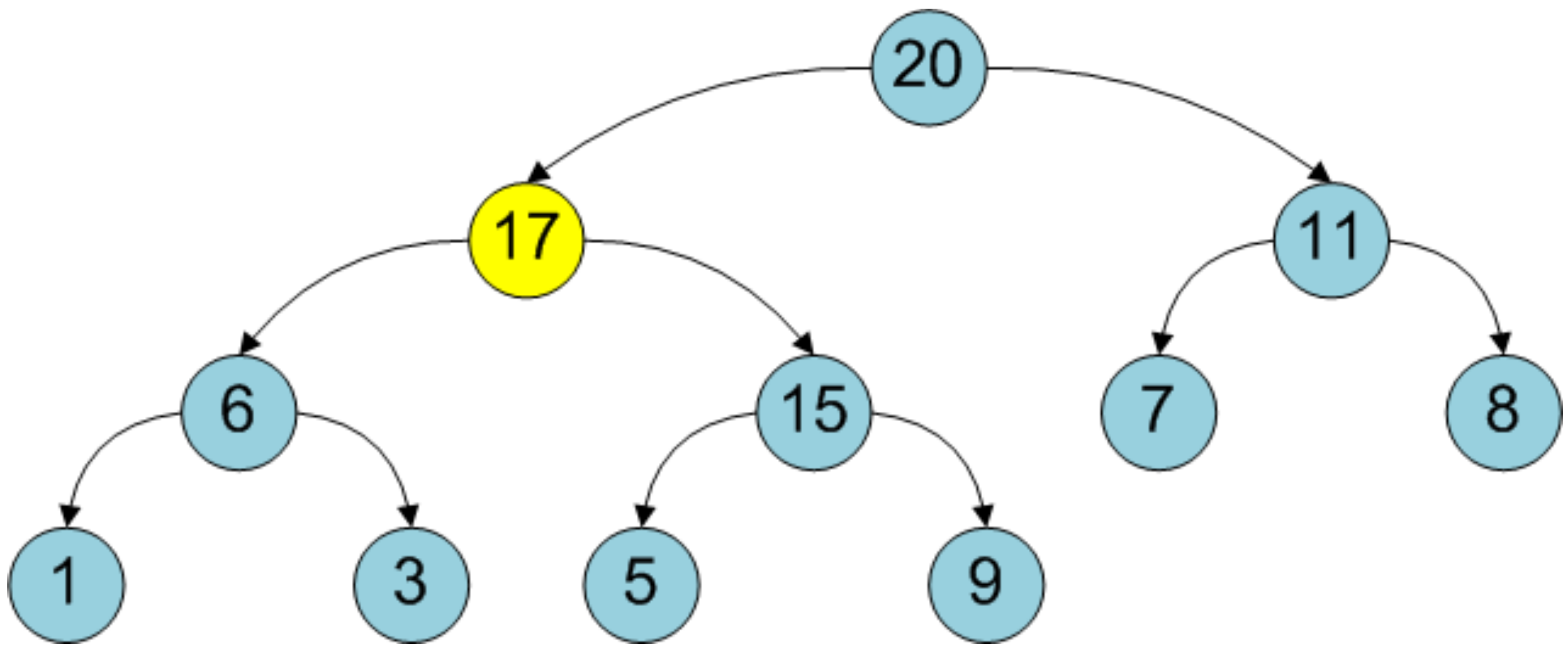


Quiz: What is the complexity of adding an element in the heap ?

Answer: $O(log(N))$

Heap

Push

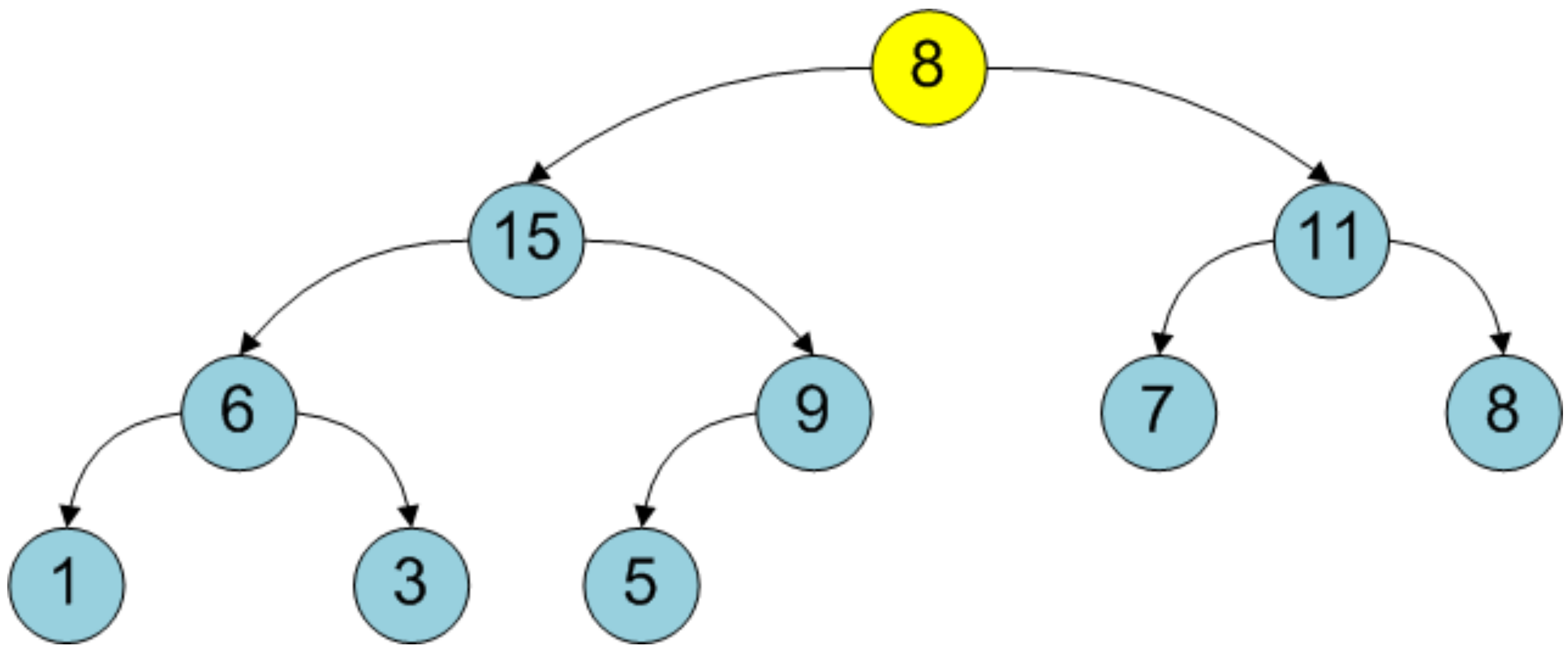


Quiz: What is the complexity of adding an element in the heap ?

Answer: $O(log(N))$

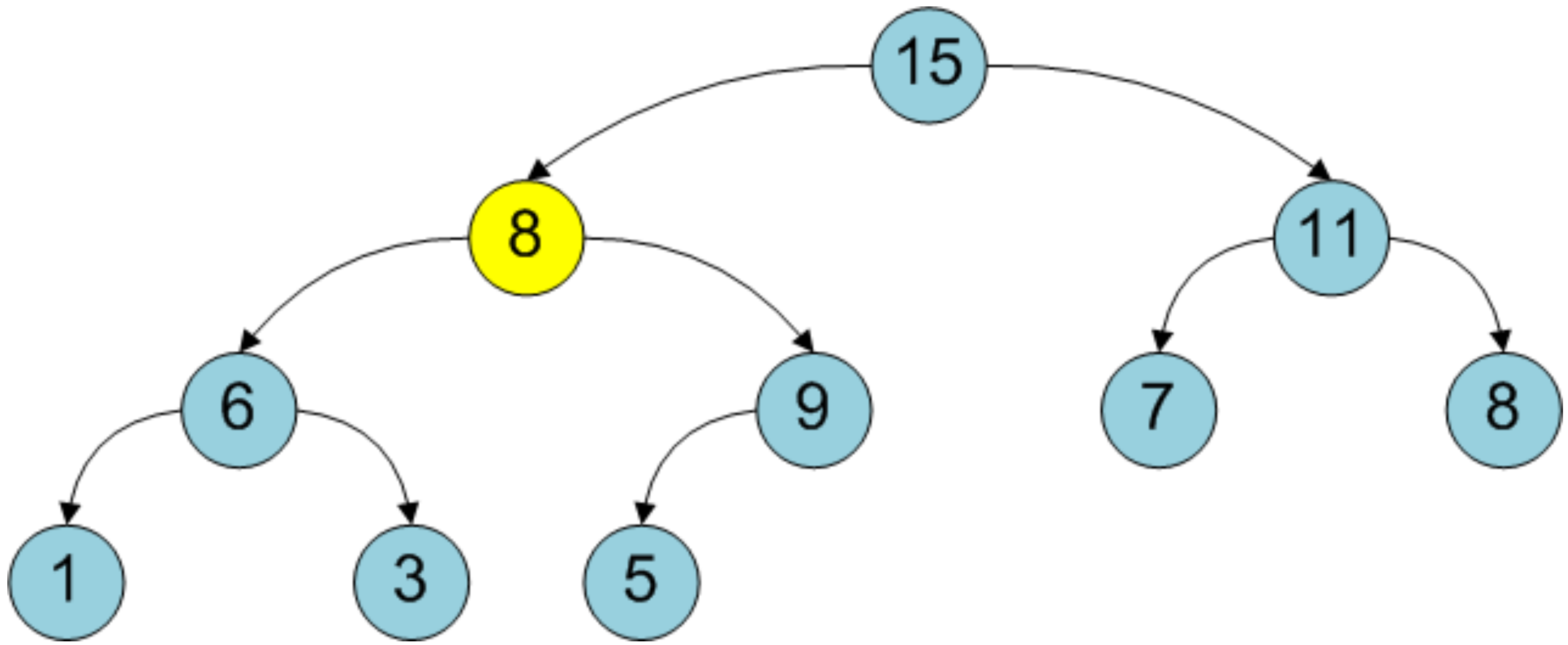
Heap

Heapify



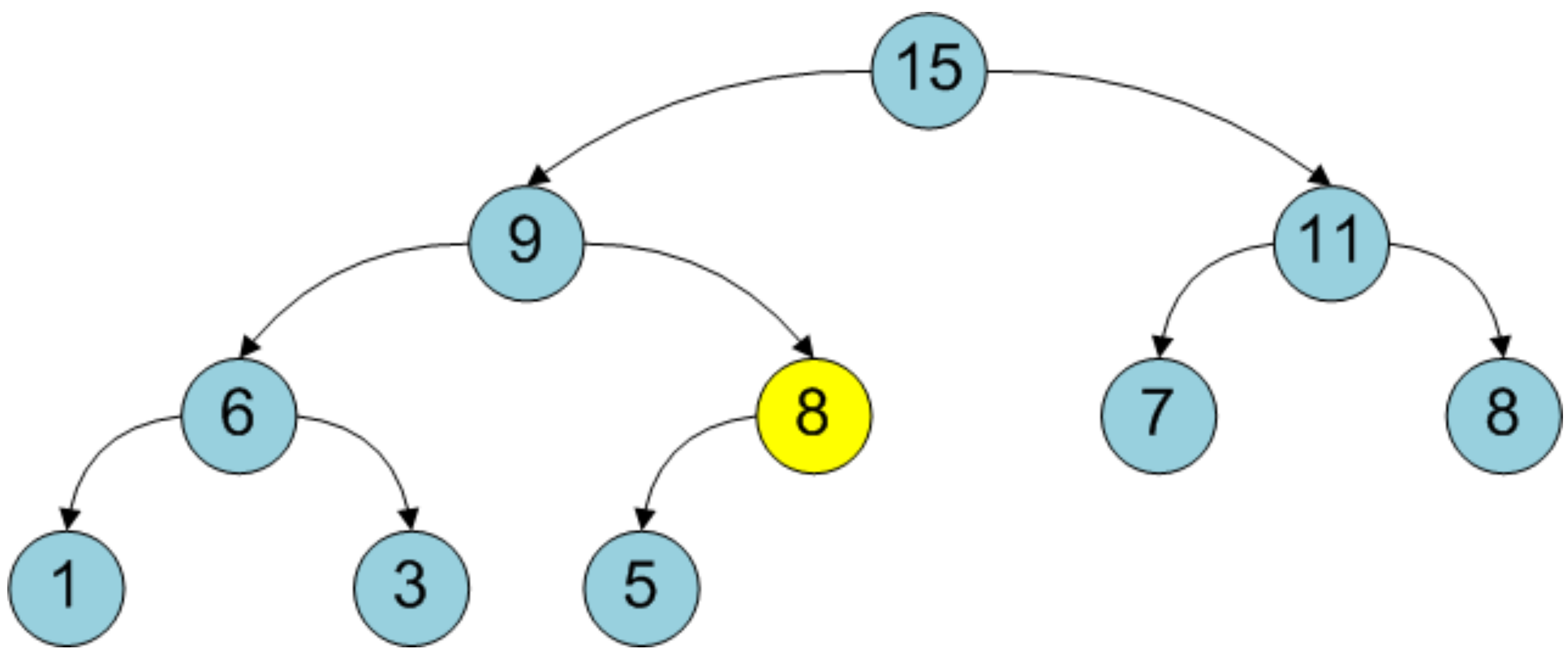
Heap

Heapify



Heap

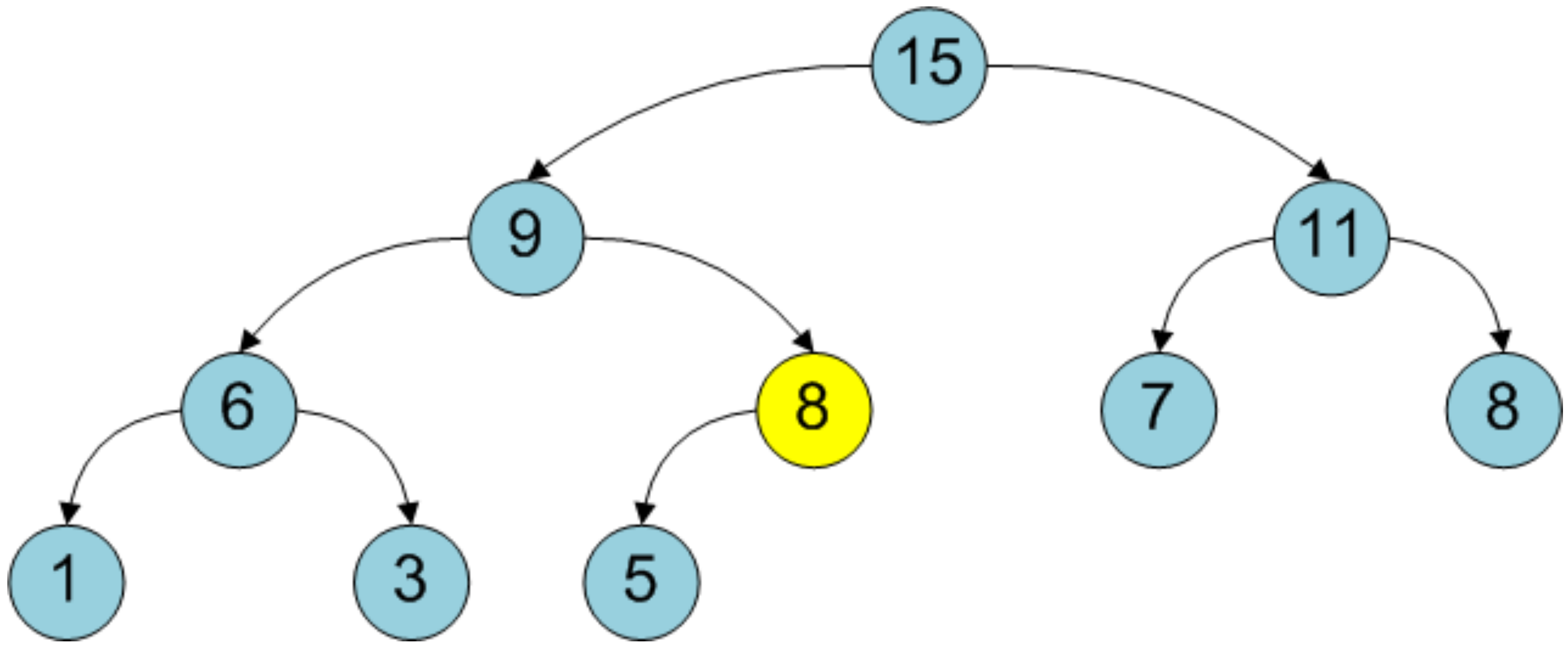
Heapify



Quiz: What is the complexity of heaping operation ?

Heap

Heapify



Quiz: What is the complexity of heaping operation ?

Answer: $O(log(N))$

Heap

Pop

1. Take root out
2. Put the last heap element on the place of the root
3. Heapify

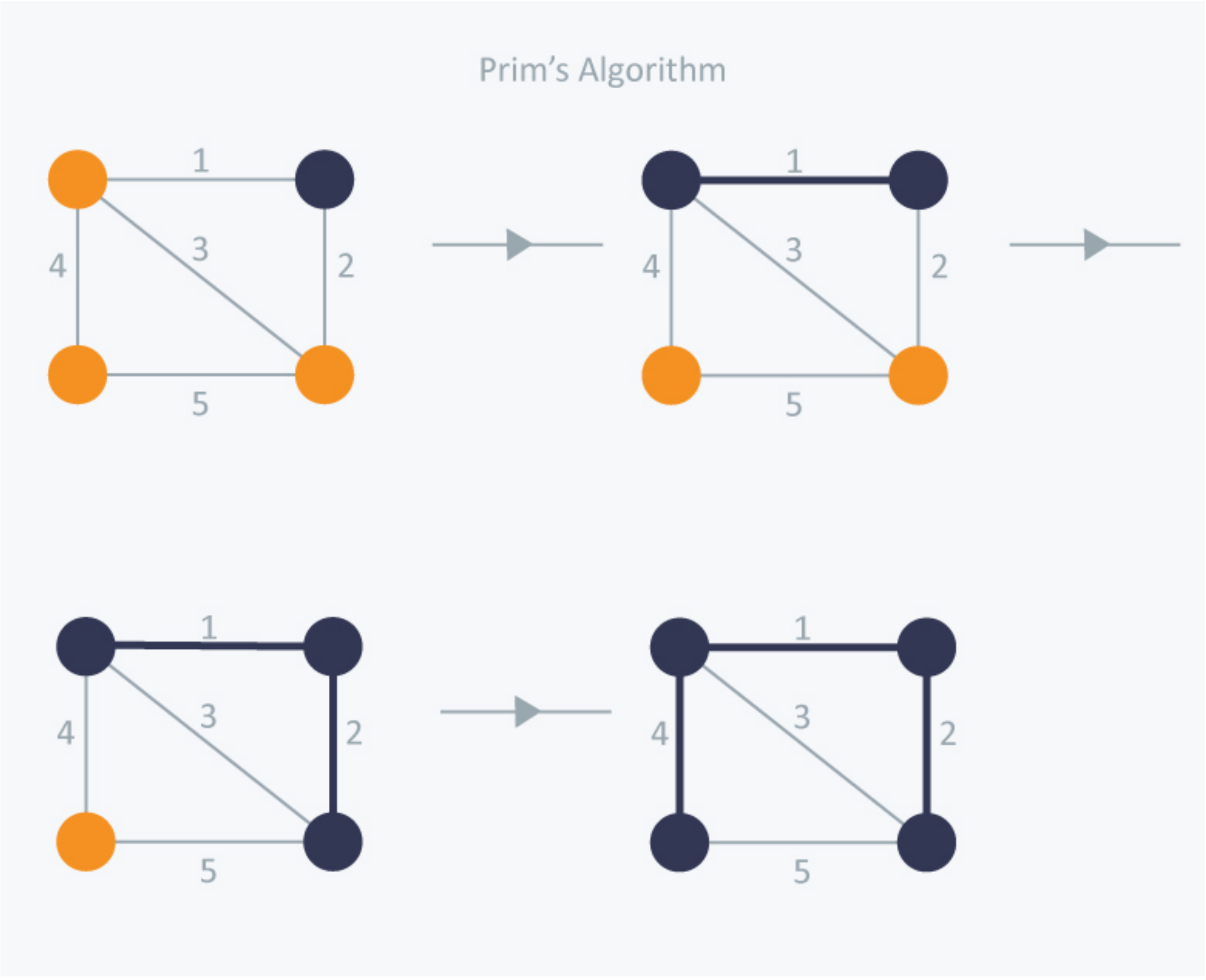
Heap

Applications

- Heap sort
- Priority queue

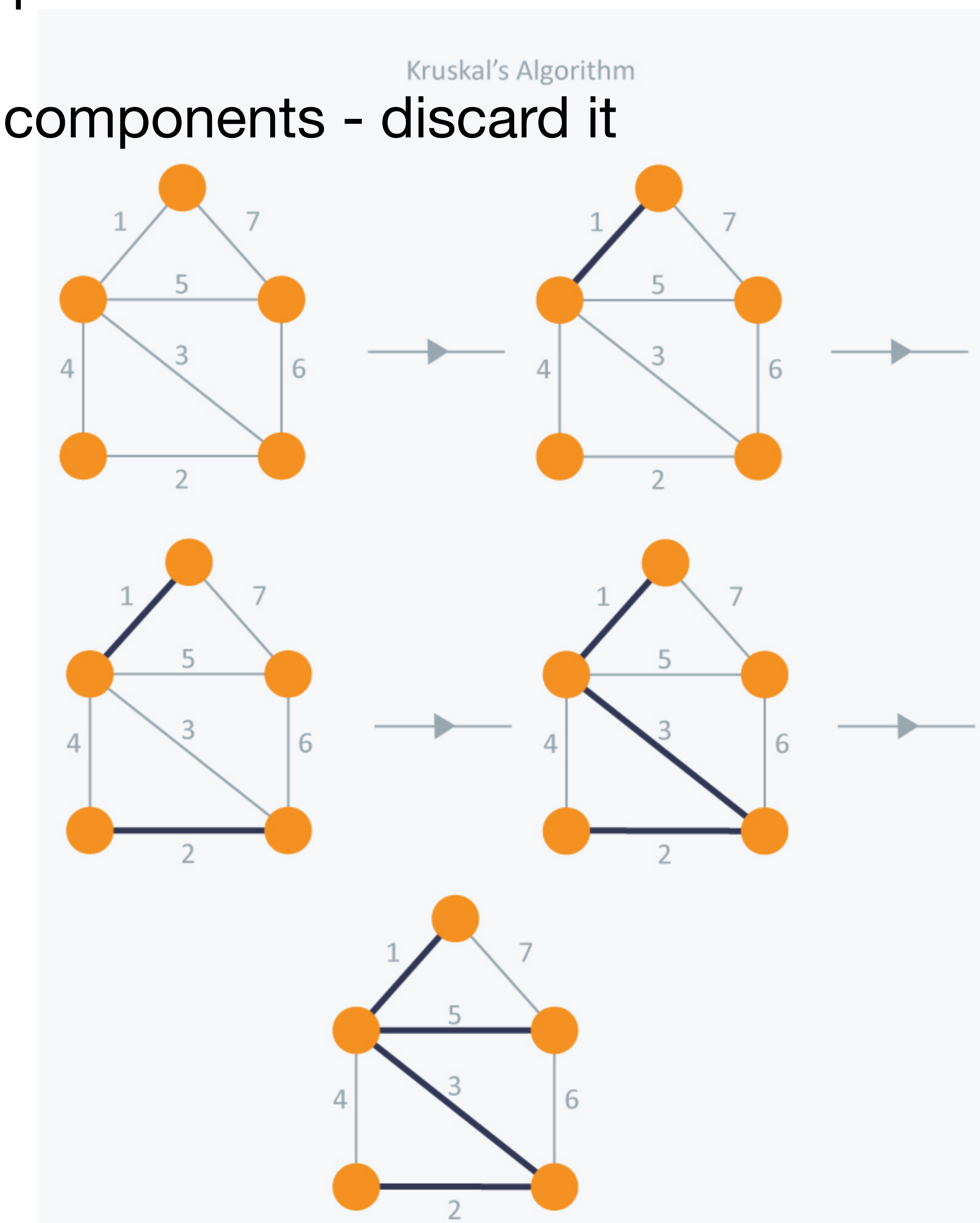
Prim algorithm

1. Select random node to start from
2. While non-tree nodes remaining:
   1. Select an edge of minimum weight between a tree and non-tree vertex
   2. Add selected edge and vertex to the tree



Kruskal algorithm

1. Initialise every node to be a single connected component
2. Consider the lightest edge
    1. If adjacent nodes are in the same connected components - discard it
    2. Otherwise, add edge, merge components



Prim vs Kruskal

- Prim **O(VE)**, if we don’t keep track of cheapest edge
- Prim **O(VV)**, if we do, but use simple data structure
- Prim **O(E + V log V)**, if use priority queue
- Kurskal **O(E log E)**

- Prim **is better on dense graphs**
- Kruskal **is better on sparse graphs**

Shortest paths

Path is a sequence of edges connecting two vertices

- For unweighted graphs can be found with BFS
- Same for the graphs with equal weights

Dijkstra algorithm find shortest path between start and end vertices.

- O(NN)
- Greedy
- On each step selects the cheapest to add edge
- Almost Prim

Dijkstra algorithm

What is different from Prim:
Instead of considering only the weight of edge consider the length of the path from the start node