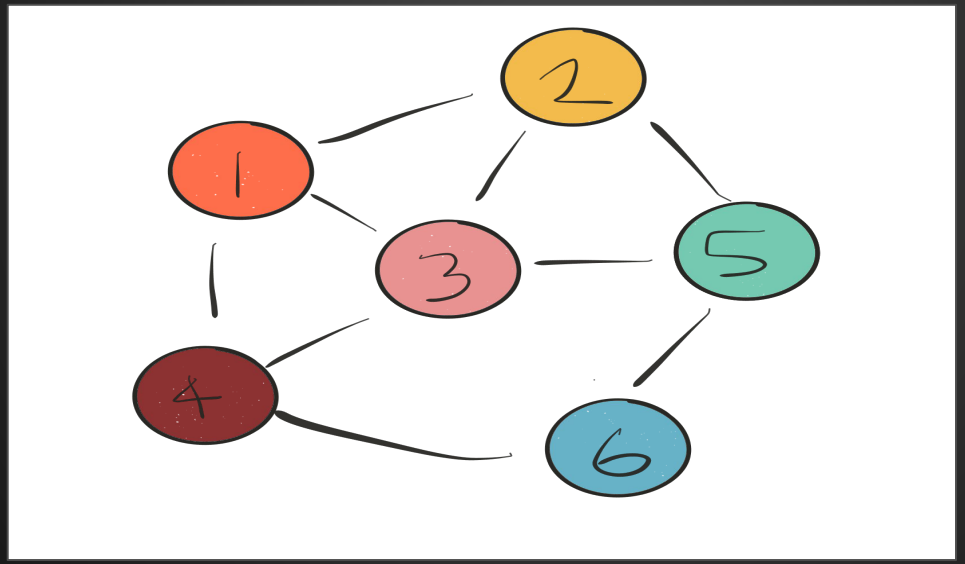
Data Structures IV

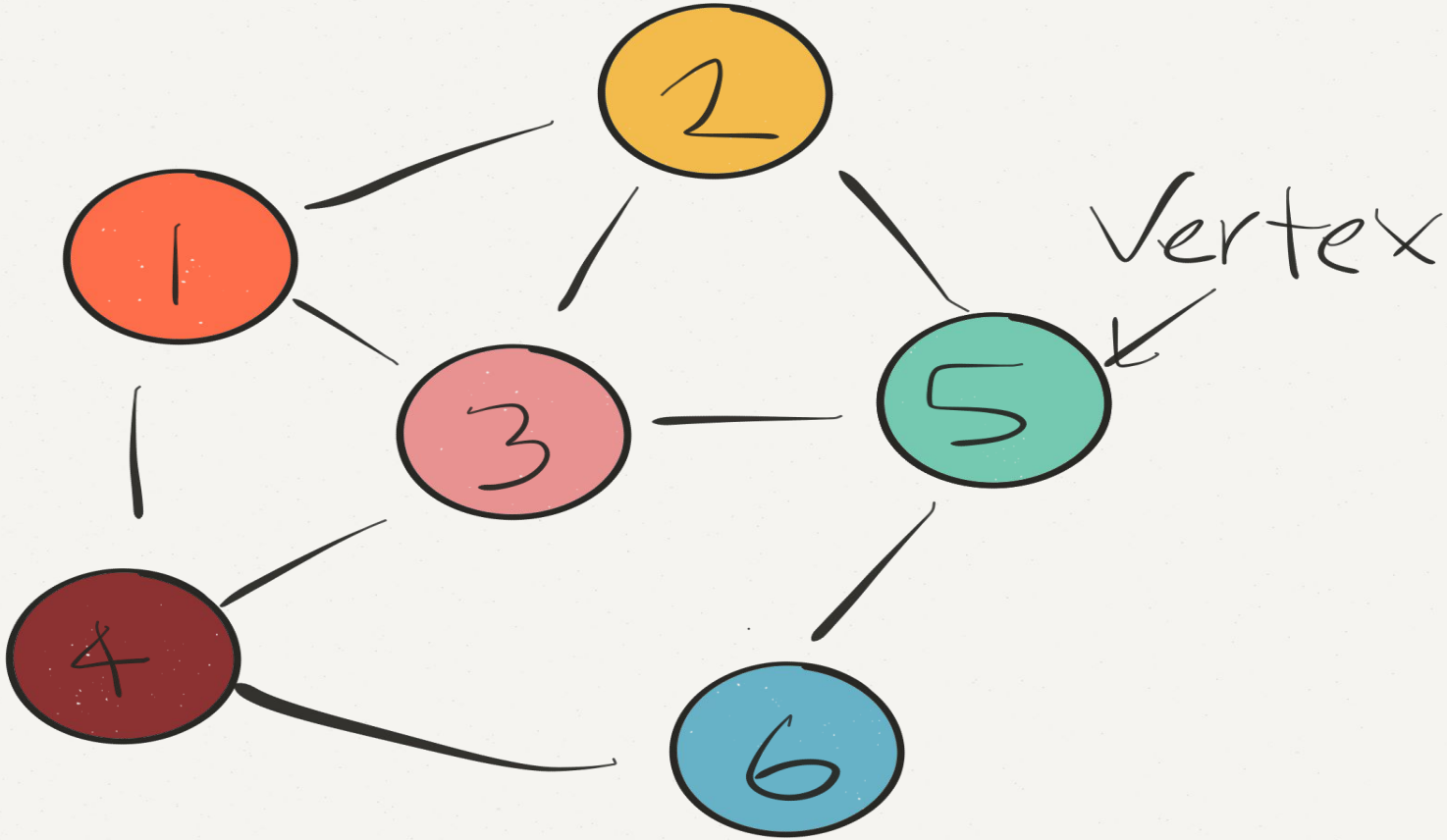
Graphs

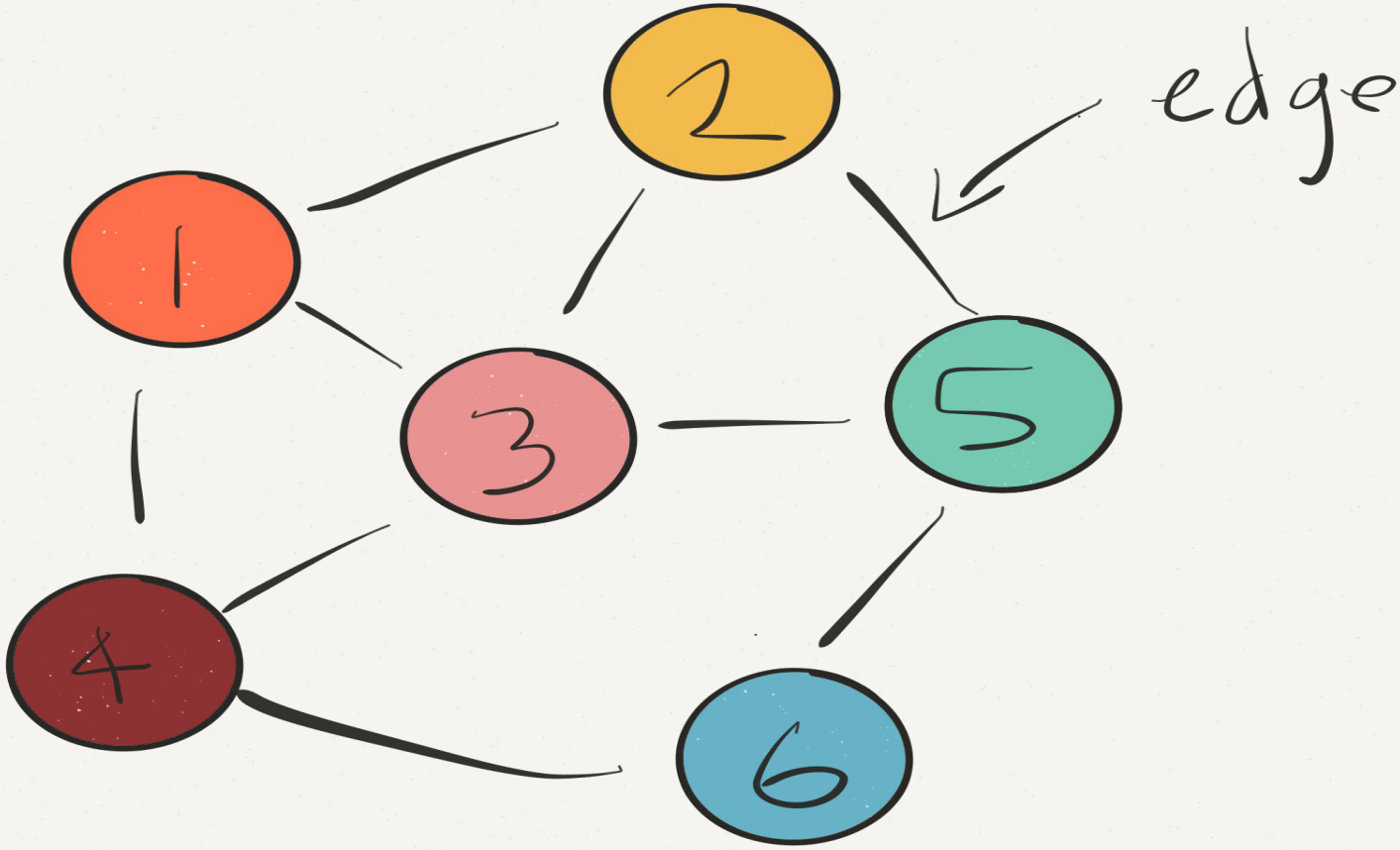
Whats a graph?

a data structure consisting of nodes and edges



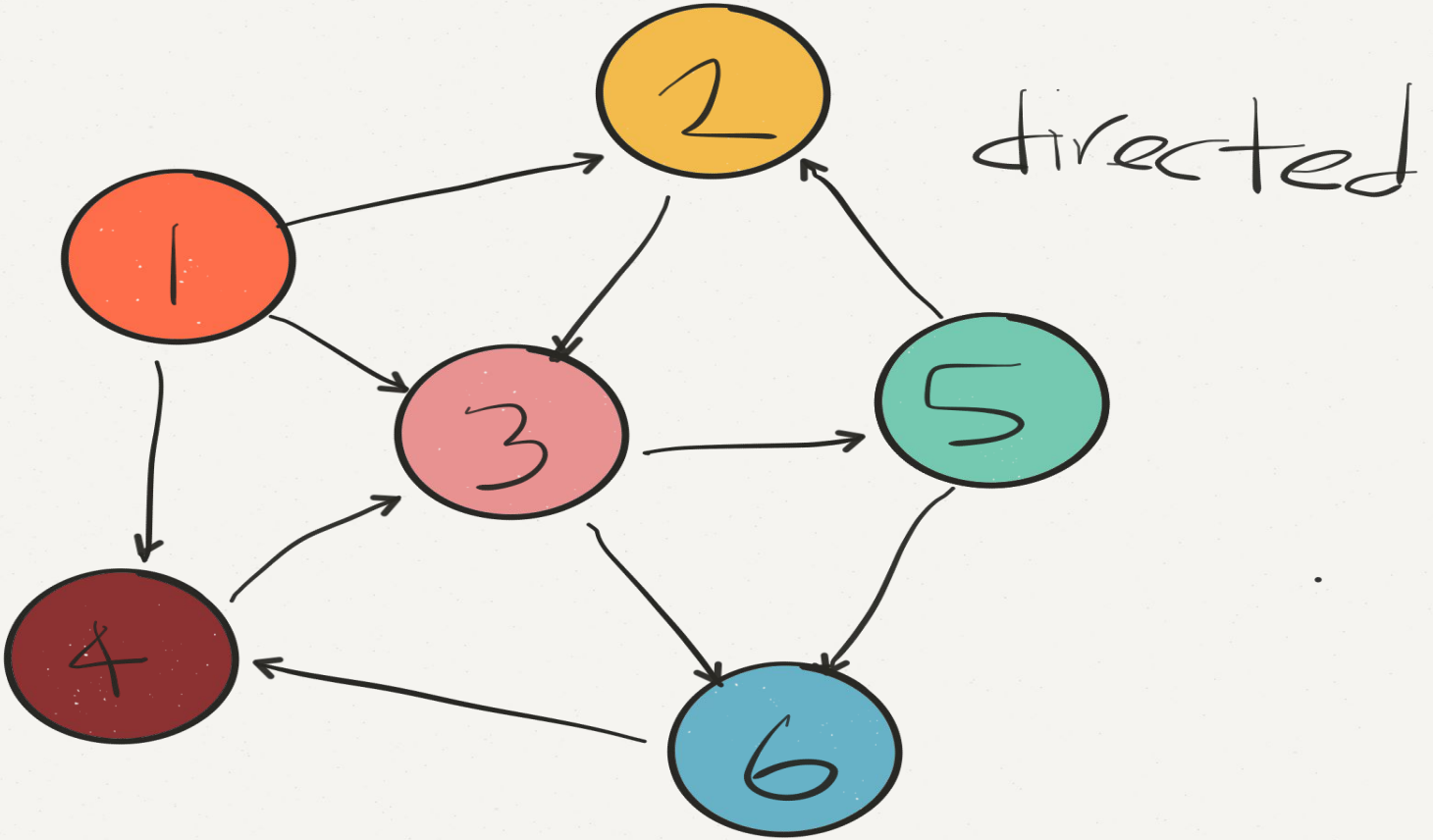
Terminology

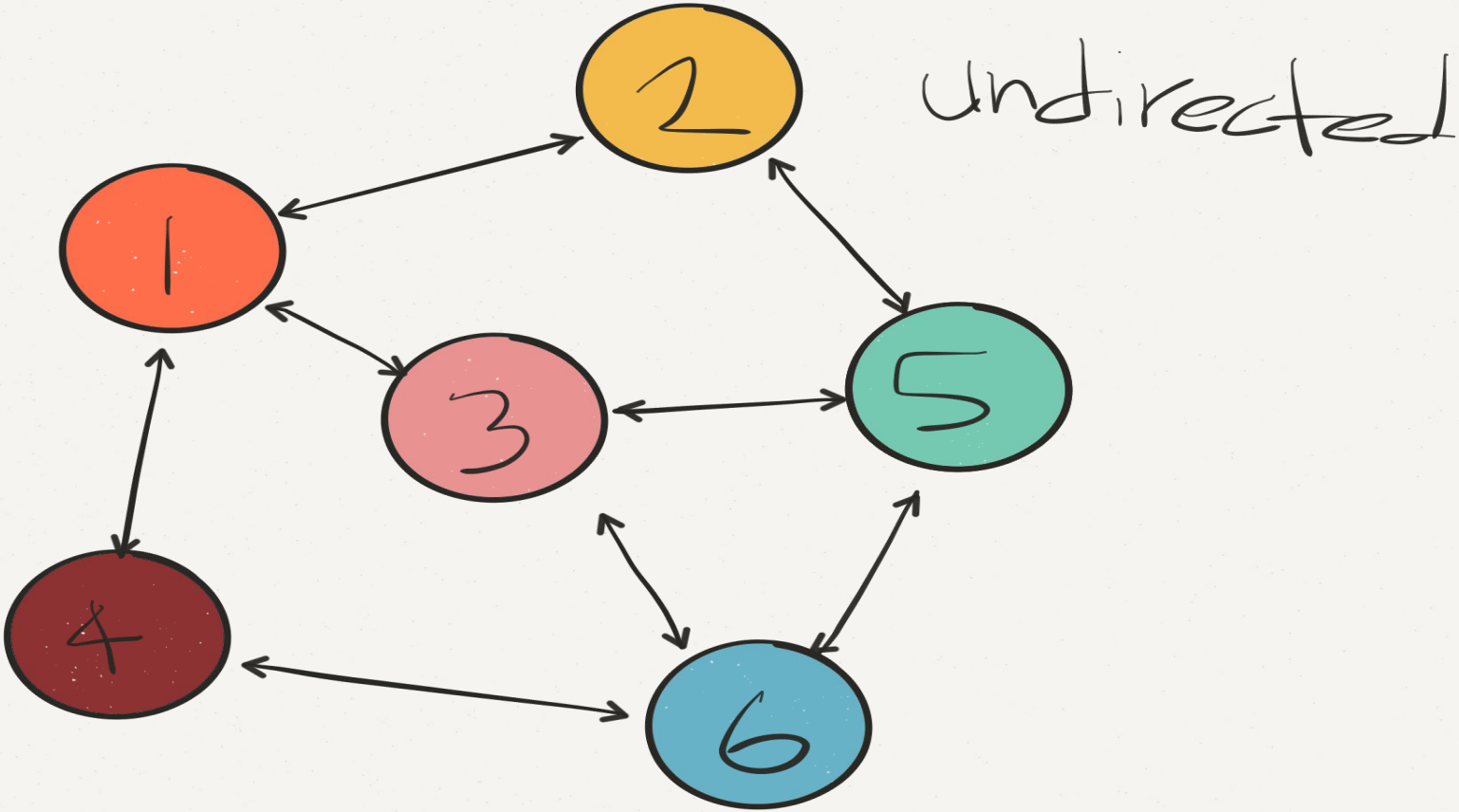

2
1
Vertex
3
5
4
6

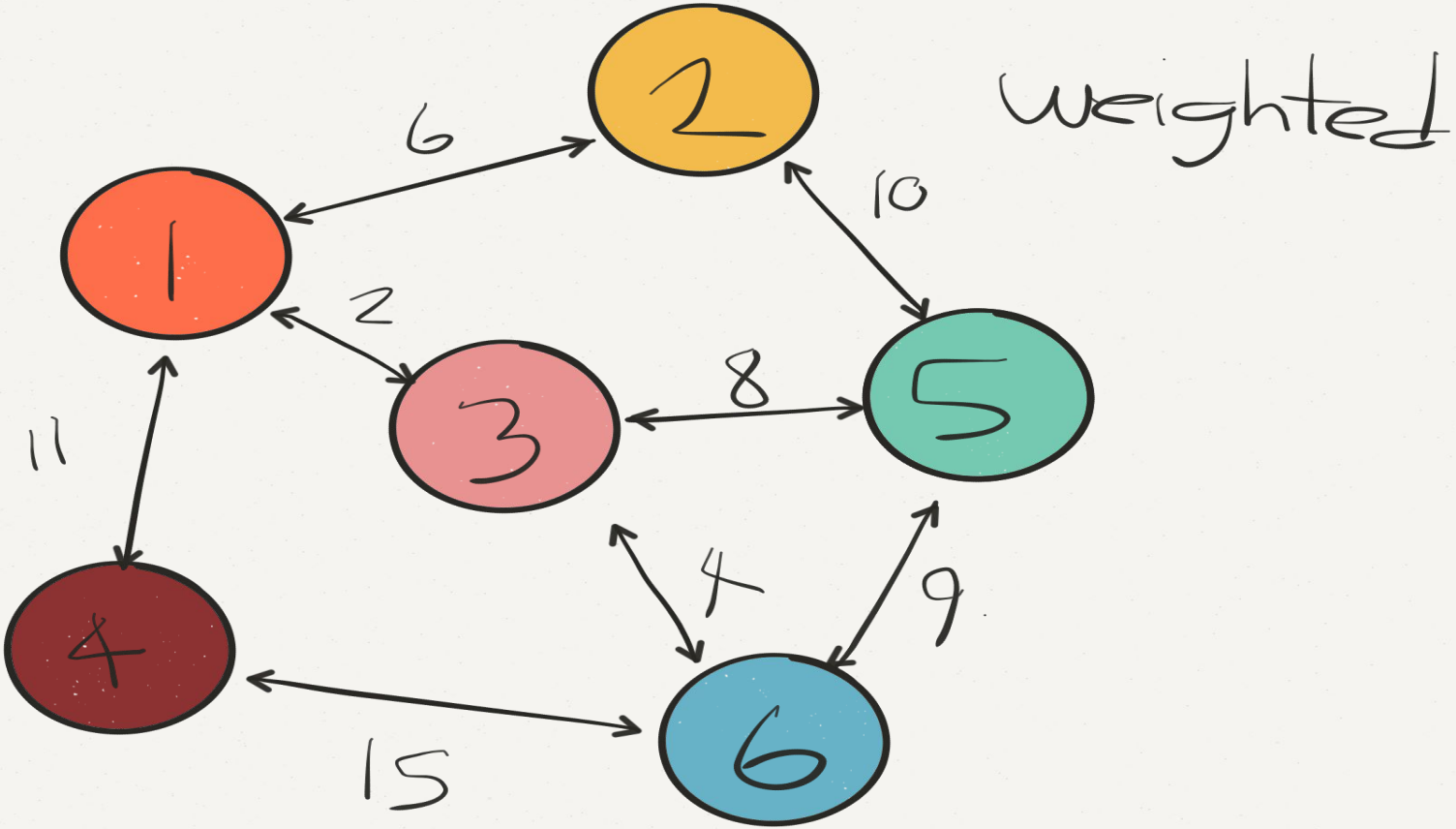

2
edge
1
3
5
4
6

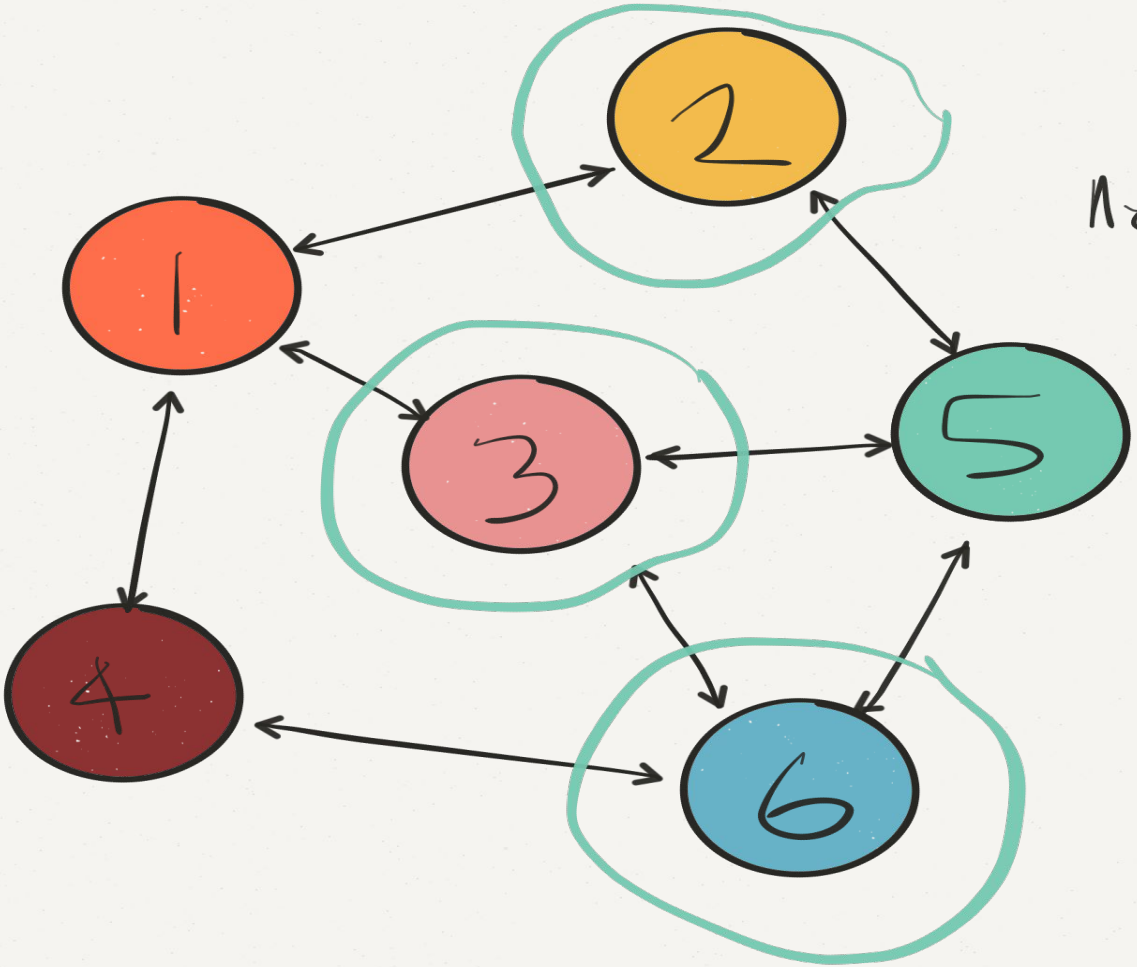
Directed vs Undirected Graphs

- Directed graphs have edges pointing from one node to the next
  - Eg. Twitter/Instagram follows
- Undirected graphs have edges without specific direction
  - Eg. Facebook friends, LinkedIn connection

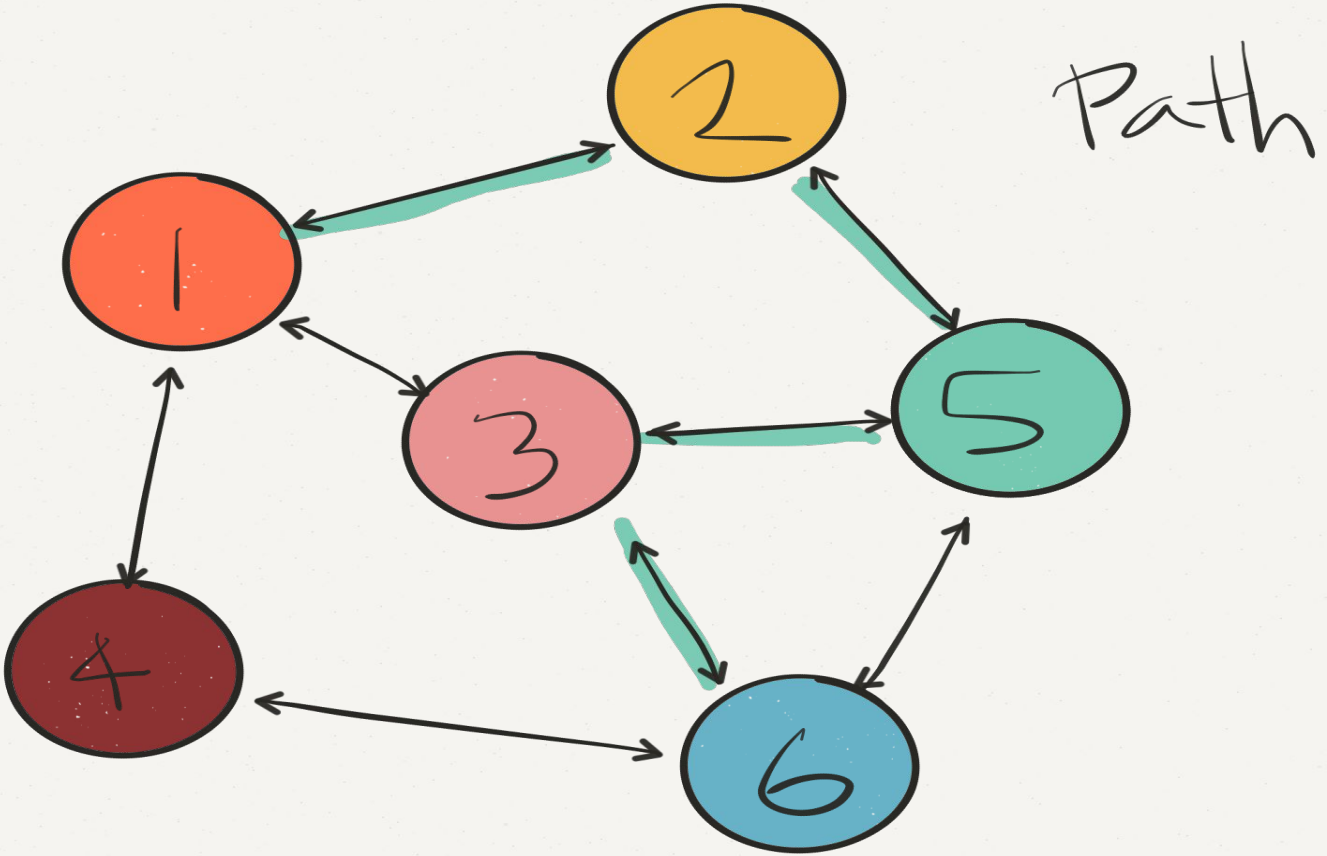

2
directed
1
3
5
4
6

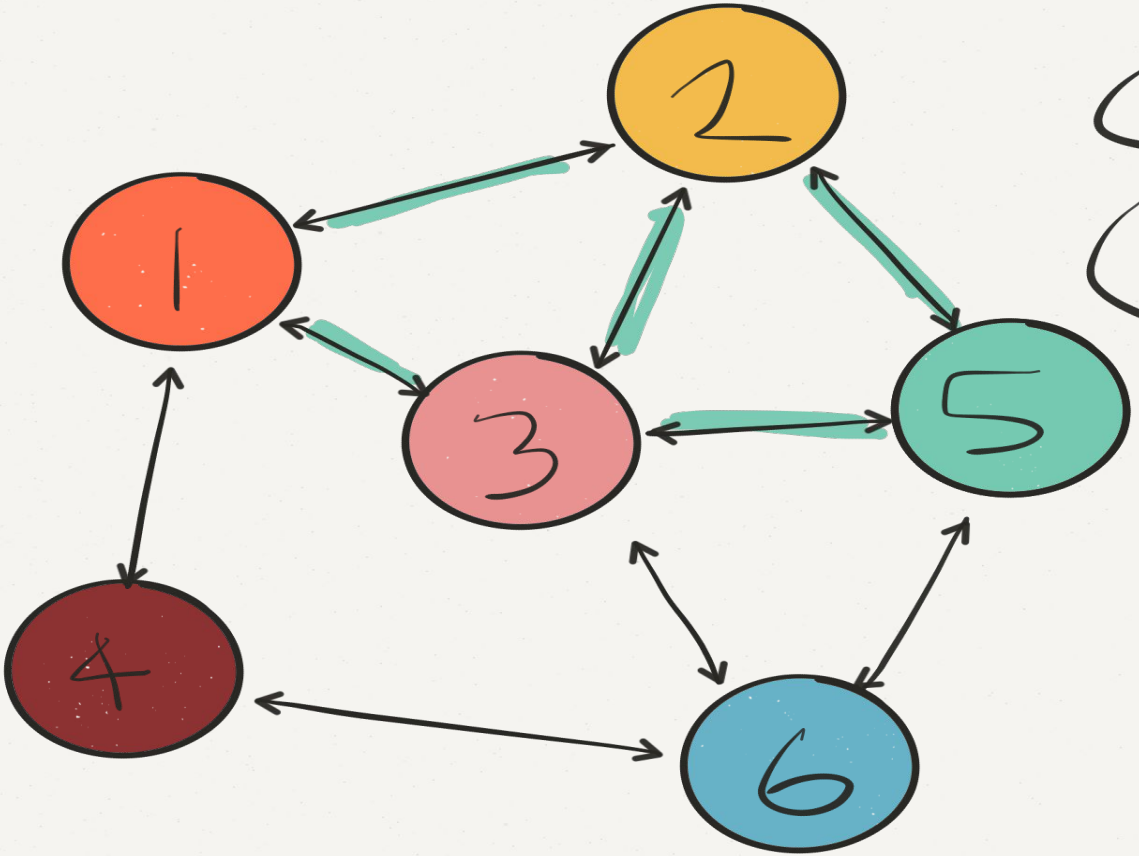

2
undirected
1
3
5
4
6


weighted
2
6
1
10
2
3
8
5
11
4
4
9
15
6



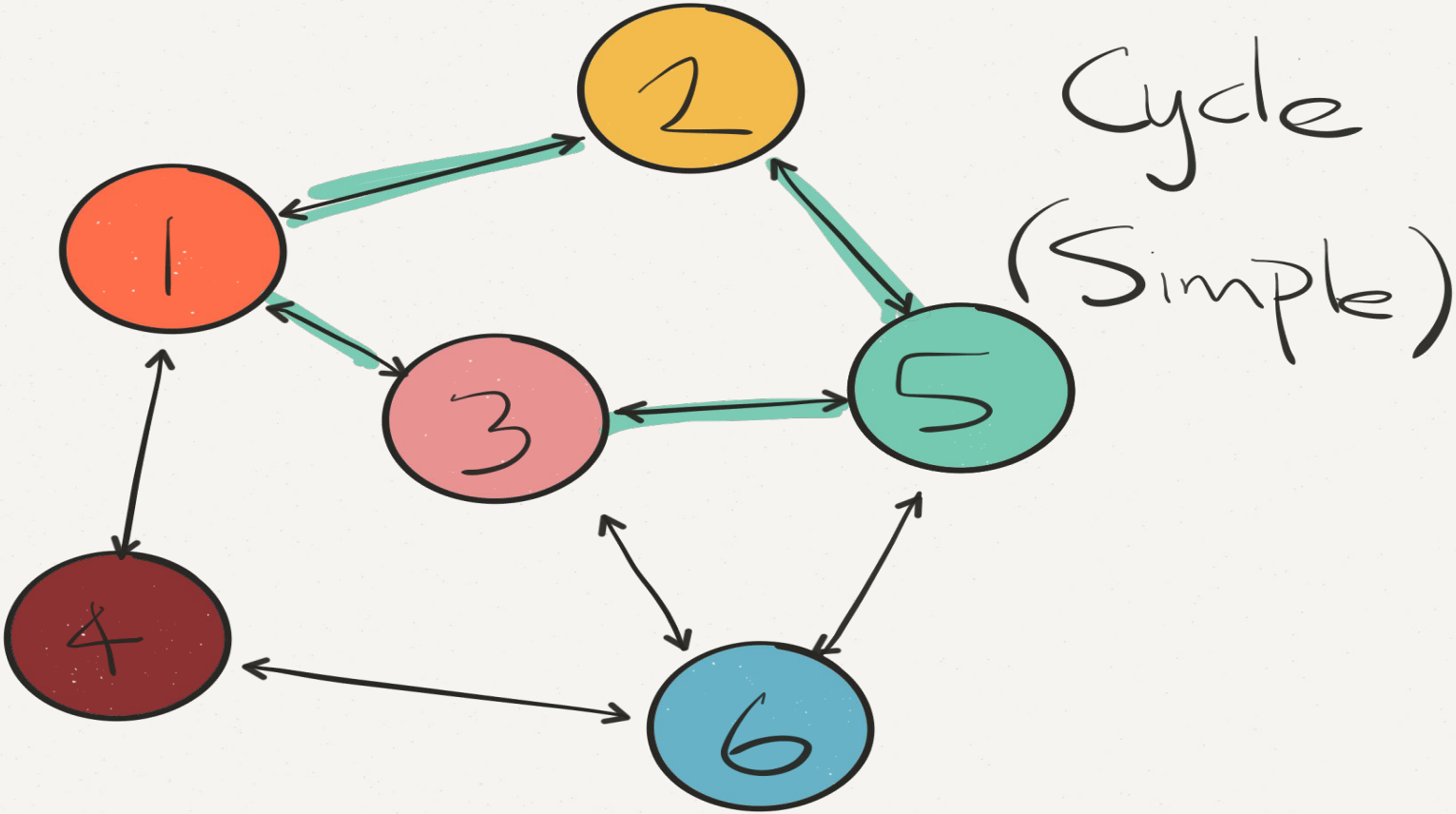
Neighbors of 5


Path
1
2
3
4
5
6

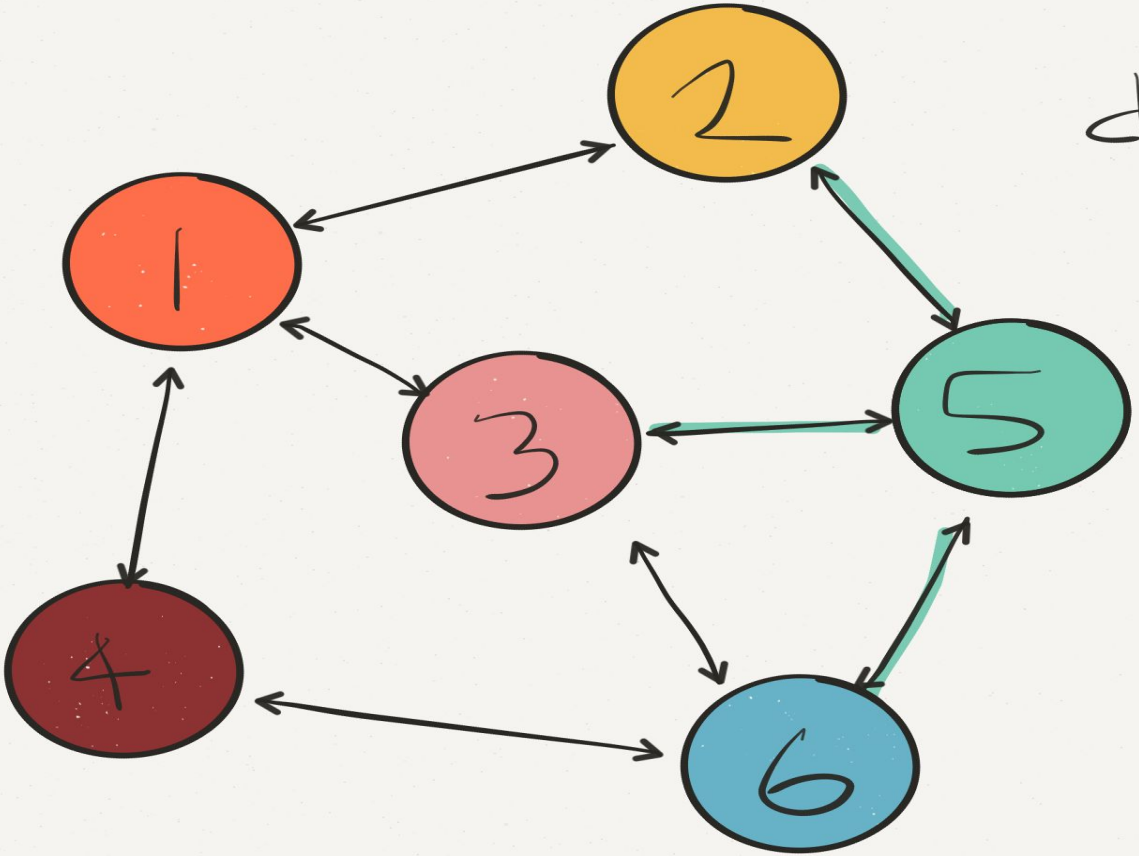


Cycle

(Closed-Walk)



Cycle
(Simple)



degree of 5 = 3

Terminology

- Order: total number of vertices
- Size: total number of edges

Graphs vs Trees

- All trees are graphs but not all graphs are trees
- Graphs can have cycles
- Graphs do not necessarily contain a root
- Can have more edges than vertices
- Can have directions and weights

Graph - Attributes

- Vertices - node class
- Edges - holds all connection/links between vertices
- Common methods
  - Add/Remove Edge
  - Add/Remove Vertex
  - Get/Set Vertex
  - Find Neighbors
  - Find Edge
  - Get/Set Edge (weighted graphs)

Representing Complexity

- Vertices and edges are distinct
  - O(V) is linear with respect to vertices
  - O(logE) is logarithmic with respect to edges
  - O(V + E) is linear with respect to the total number of vertices and edges
- Degree is average number of edges per vertex
  - O(d) is linear with respect to degree

Edge Representations

- List of Edges
- Adjacency Matrix
- Adjacency List

Edge Representation

List of Edges

- Array of tuples
- Can represent undirected or directed
- Add third element for weight

```
edges = [[0,2], [1,4], [1,3], [2,3], [3,4], [4,1]]
```

Edge Representation

List of Edges

- Time complexity of finding an edge?
  - O(E)
- Space complexity?
  - O(E)

```
edges = [[0,2], [1,4], [1,3], [2,3], [3,4], [4,1]]
```

Edge Representation

Adjacency Matrix

- V by V size
- Indices (i,j) correspond to vertices
- Represented by array of arrays
- Can include weights

```
edges = [ [0,0,1,0,0],
          [0,0,1,1,0],
          [0,0,0,1,0],
          [0,0,0,1,0],
          [0,1,0,0,0] ]
```

Edge Representation

Adjacency Matrix

- Time complexity of finding an edge?
  - O(1)
- Space complexity?
  - $O(V^2)$

```
edges = [ [0,0,1,0,0],
          [0,0,1,1,0],
          [0,0,0,1,0],
          [0,0,0,1,0],
          [0,1,0,0,0] ]
```

Edge Representation

Adjacency Lists

- Array of arrays
- Outer array index corresponds to vertex
- Nest array are each vertex's edges

```
edges = [ [1,2],
          [3,4],
          [0,3,4],
          [4],
          [1] ]
```

Edge Representation

Adjacency Lists

- Time complexity of finding an edge?
  - O(d)
- Space complexity?
  - O(V + E)

```
edges = [ [1,2],
          [3,4],
          [0,3,4],
          [4],
          [1] ]
```

http://visualgo.net/dfsbfs.html

depth first search & breadth first search

courtesy of visualgo.net

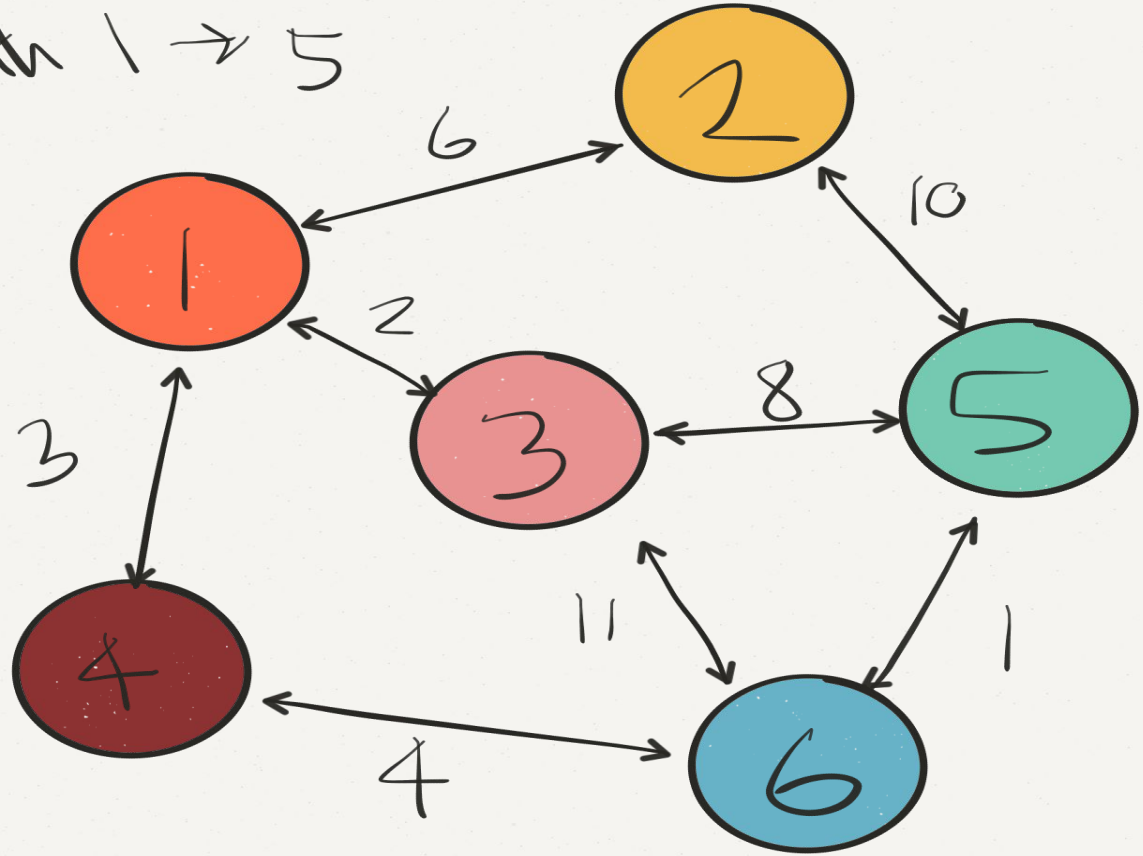
Graph Traversal

- Redundancy
  - Unlike trees, during traversal a node may be visited multiple times
  - Typically should track if a node has been visited
- Depth first search
  - Use recursion or a stack
- Breadth first search
  - Use a queue

Shortest Path

- Given a weighted graph
- Find the shortest path between two nodes
- Dijkstra’s Algorithm

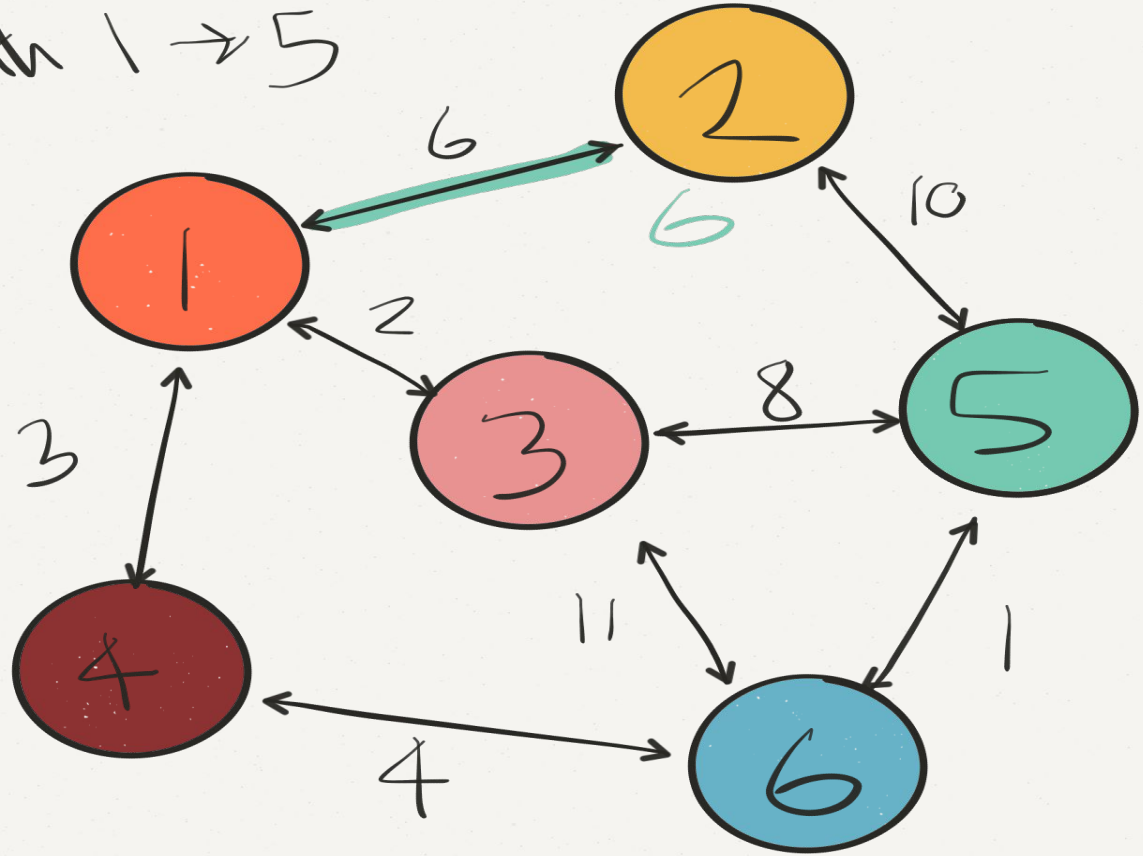
Path 1 → 5



Queue

Visited

Path 1 → 5

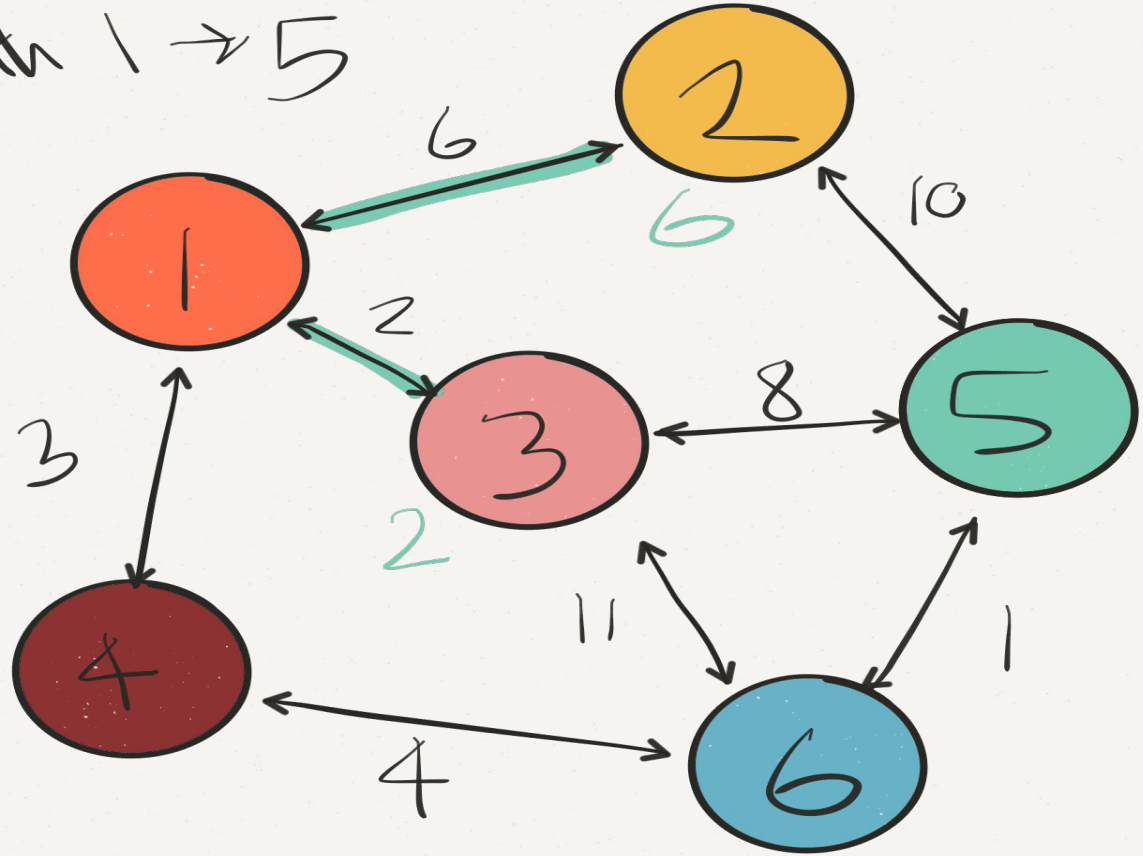


Queue

2

Visited

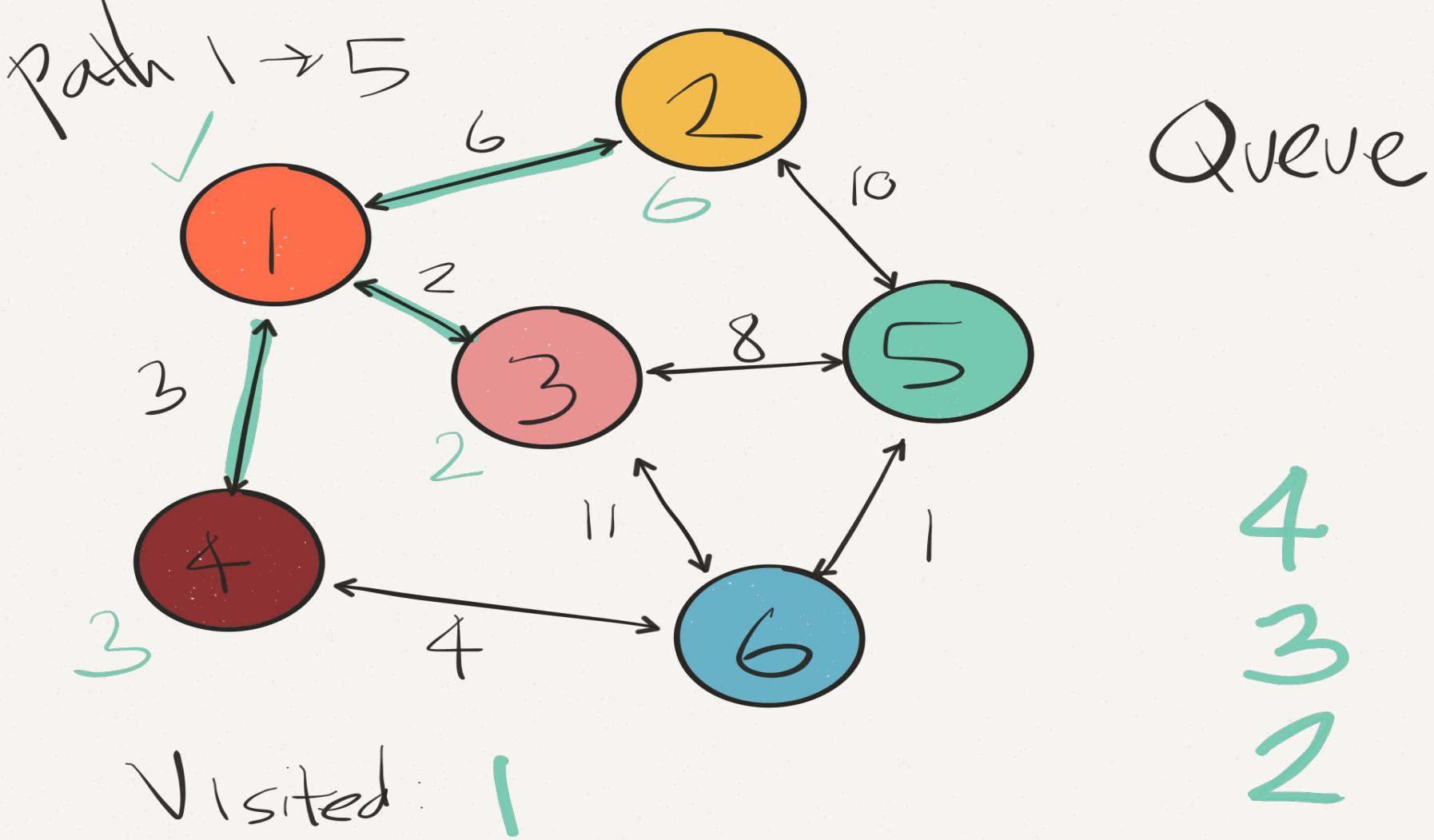
Path 1 → 5



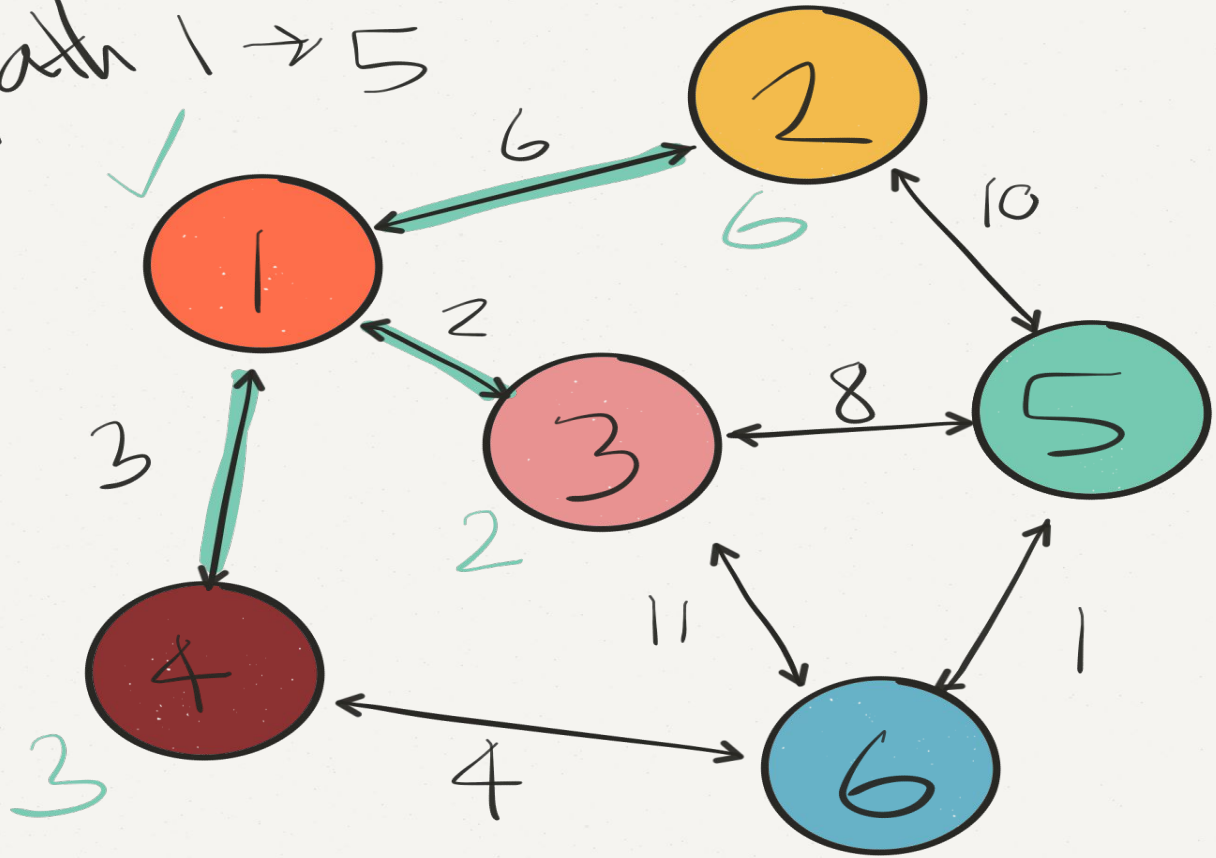
Queue

3
2

Visited


Path 1 → 5
1
2
3
4
5
6
6
10
2
3
8
3
2
11
1
3
4
Visited: 1
Queue
4
3
2

Path 1 → 5

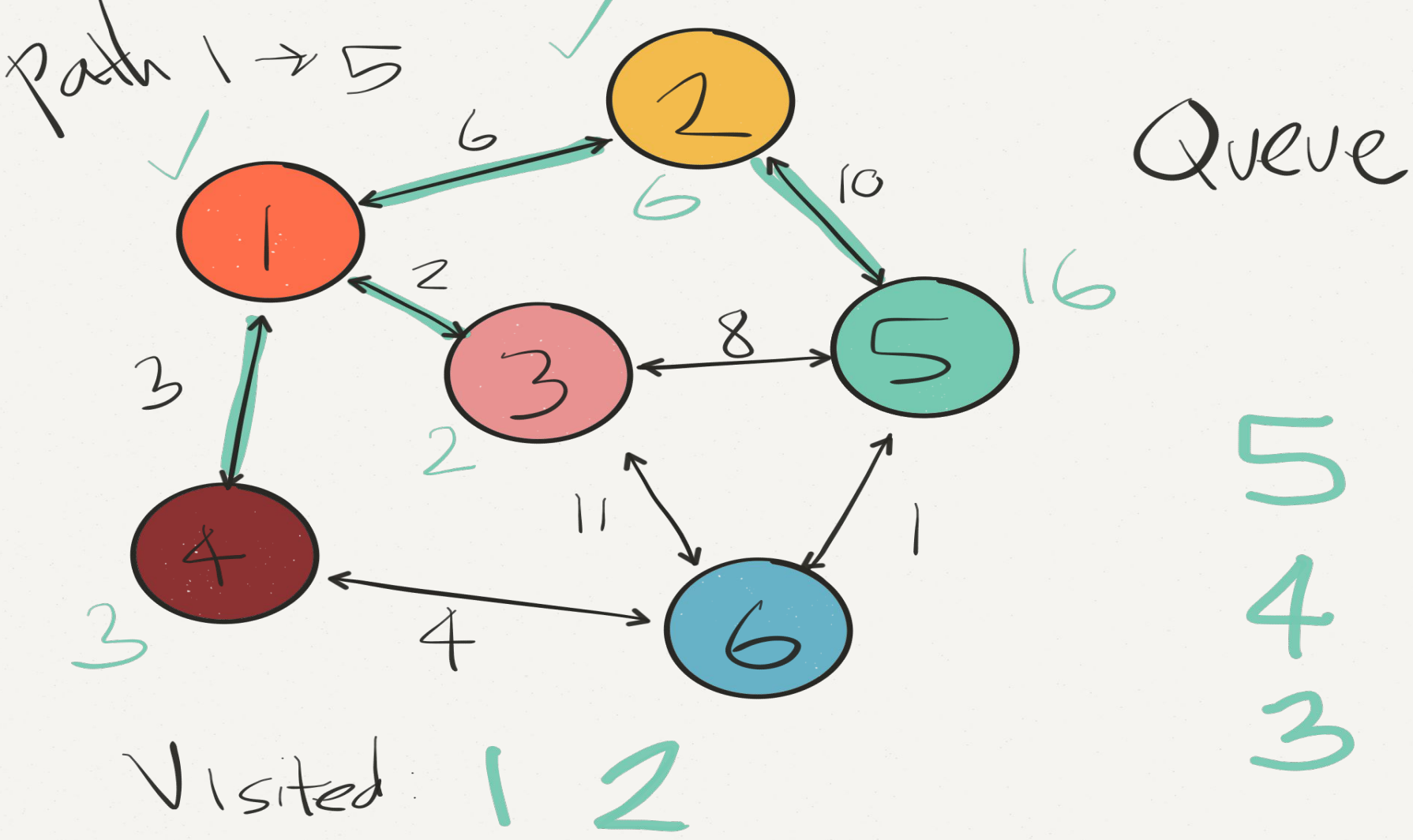


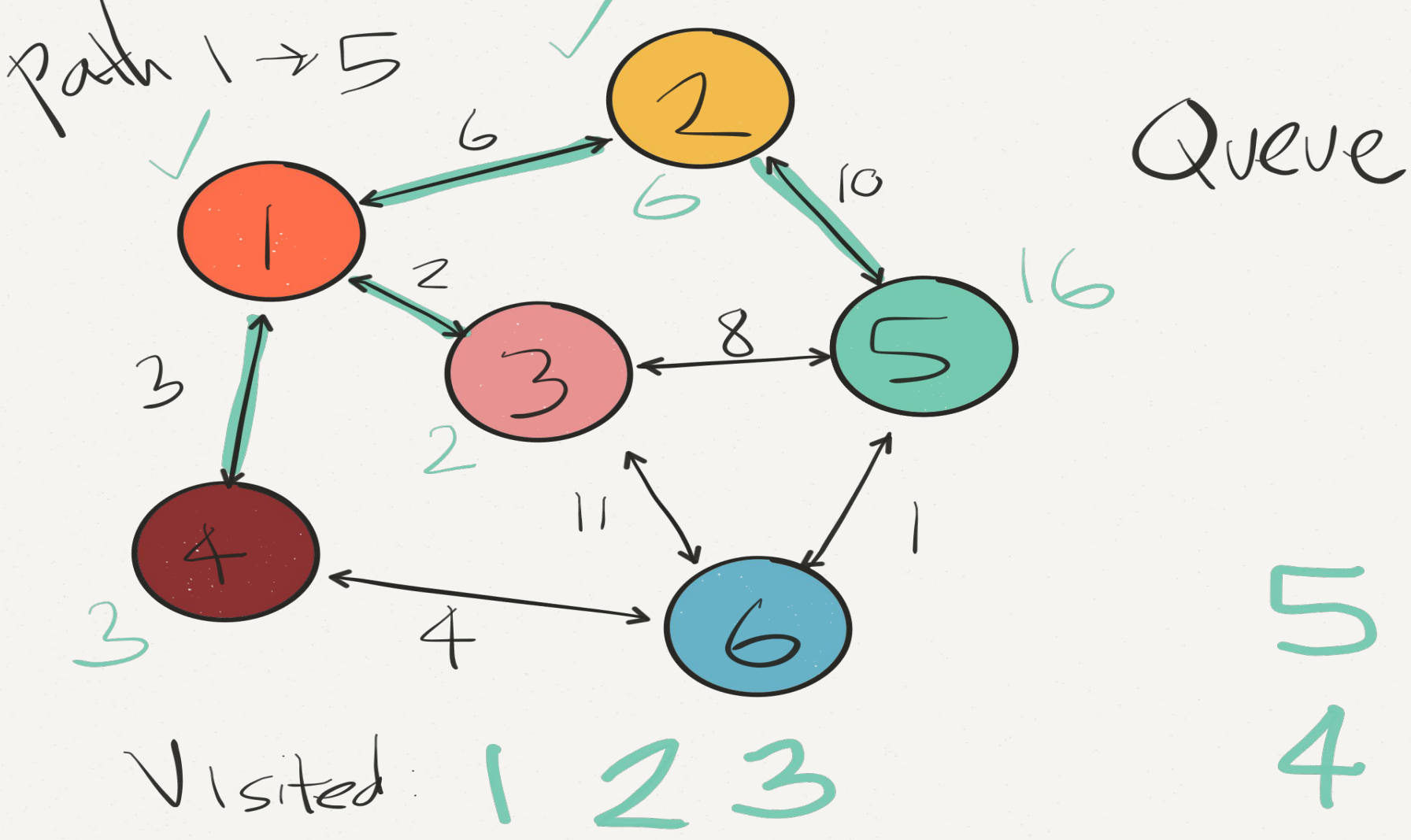
Queue

4

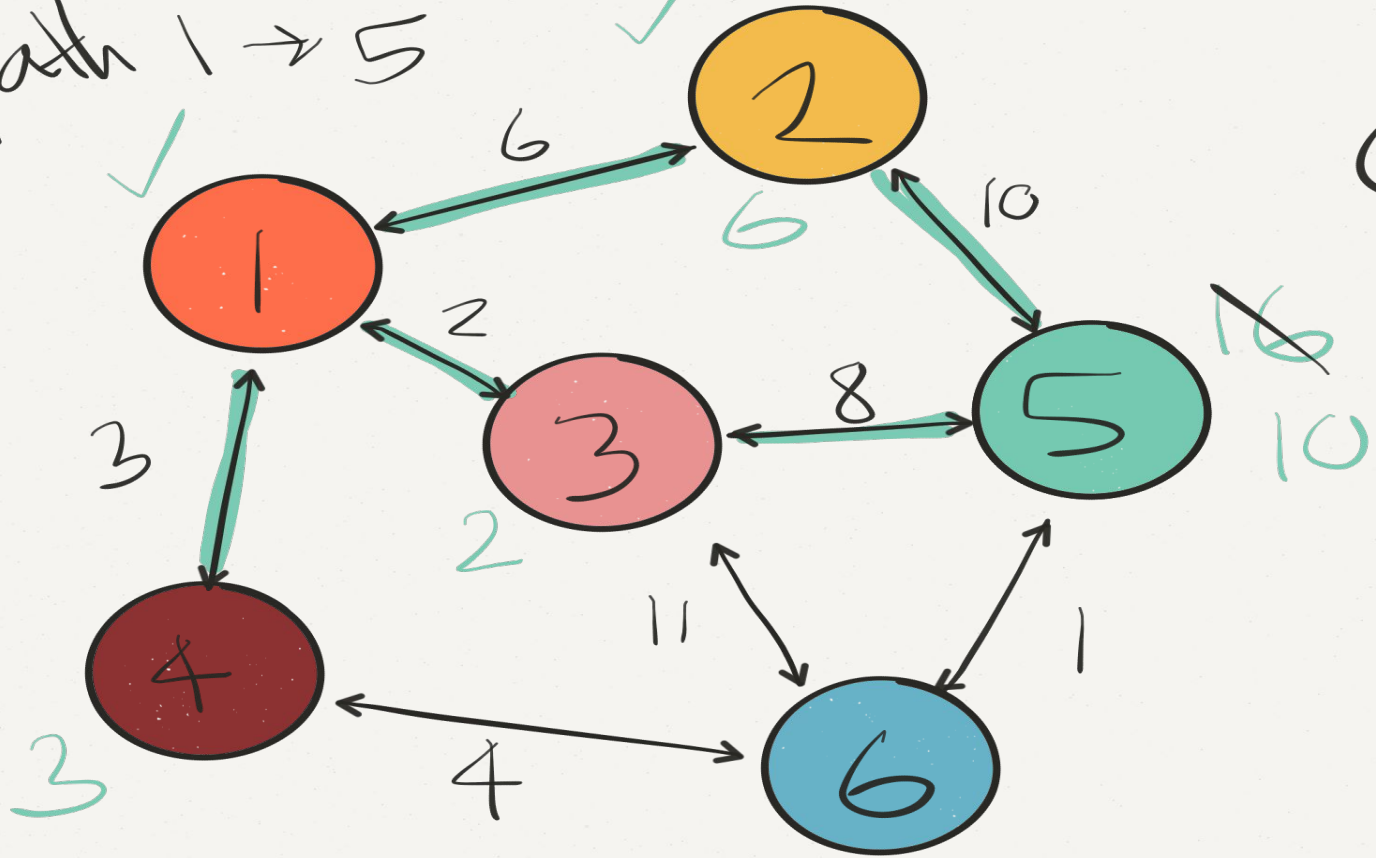
3

Visited: 1 2


Path 1 → 5
1
2
3
4
5
6
6
10
2
3
8
16
2
11
1
3
4
Queue
5
4
3
Visited
1
2


Path 1 → 5
1
2
3
4
5
6
6
10
2
3
8
16
2
11
1
3
4
Queue
5
4
Visited
1 2 3

Path 1 → 5

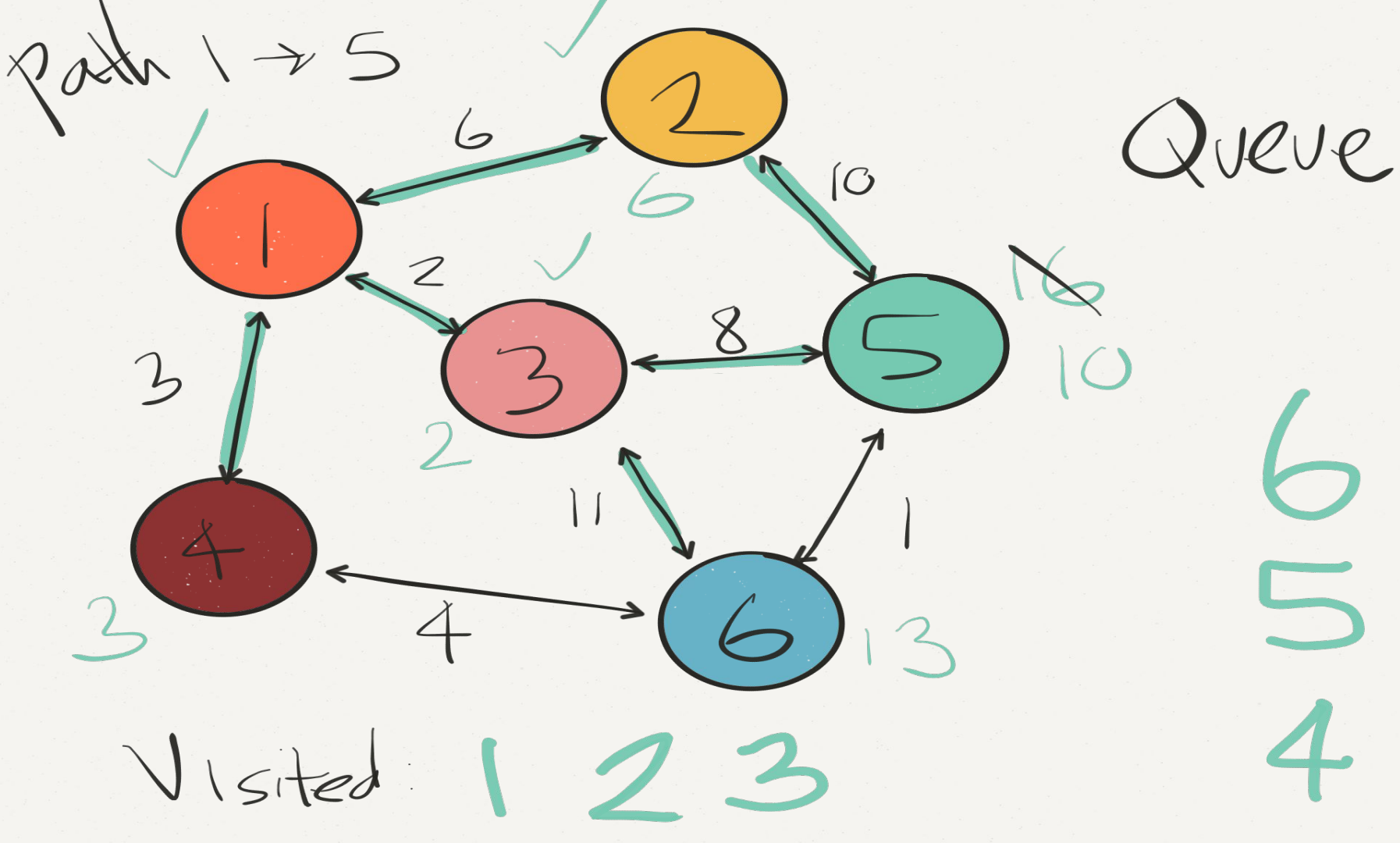


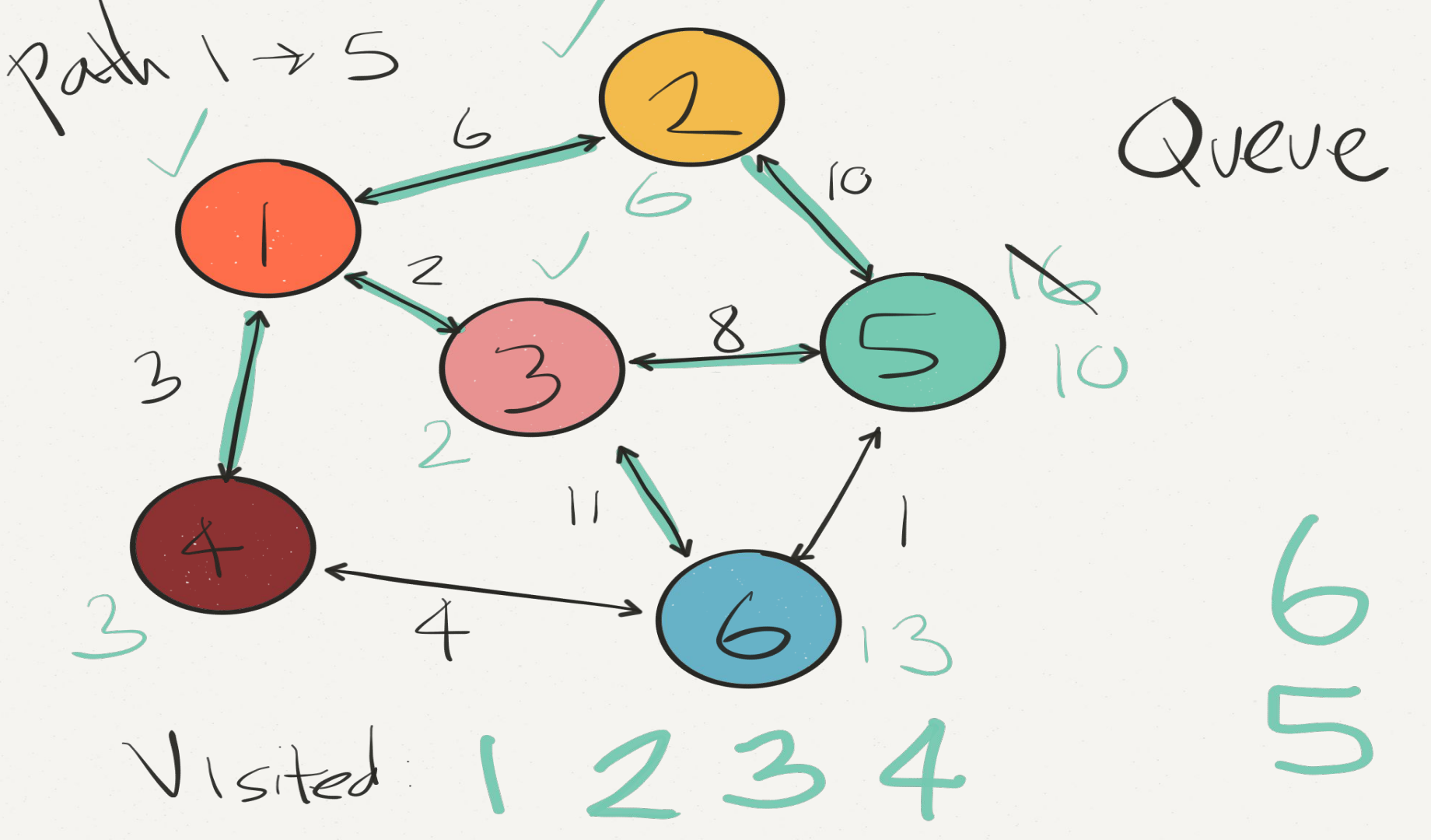
Queue

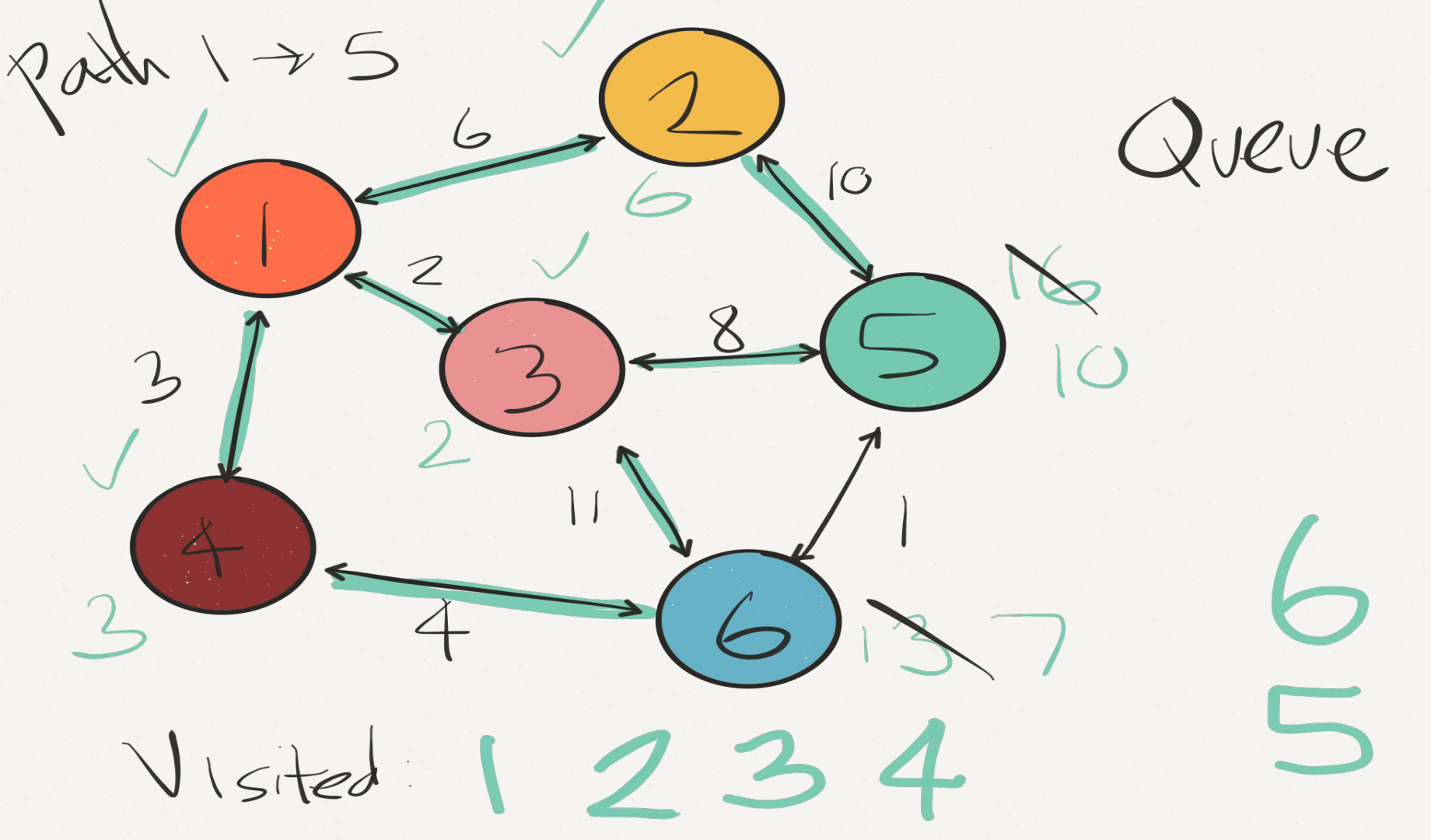
5

4

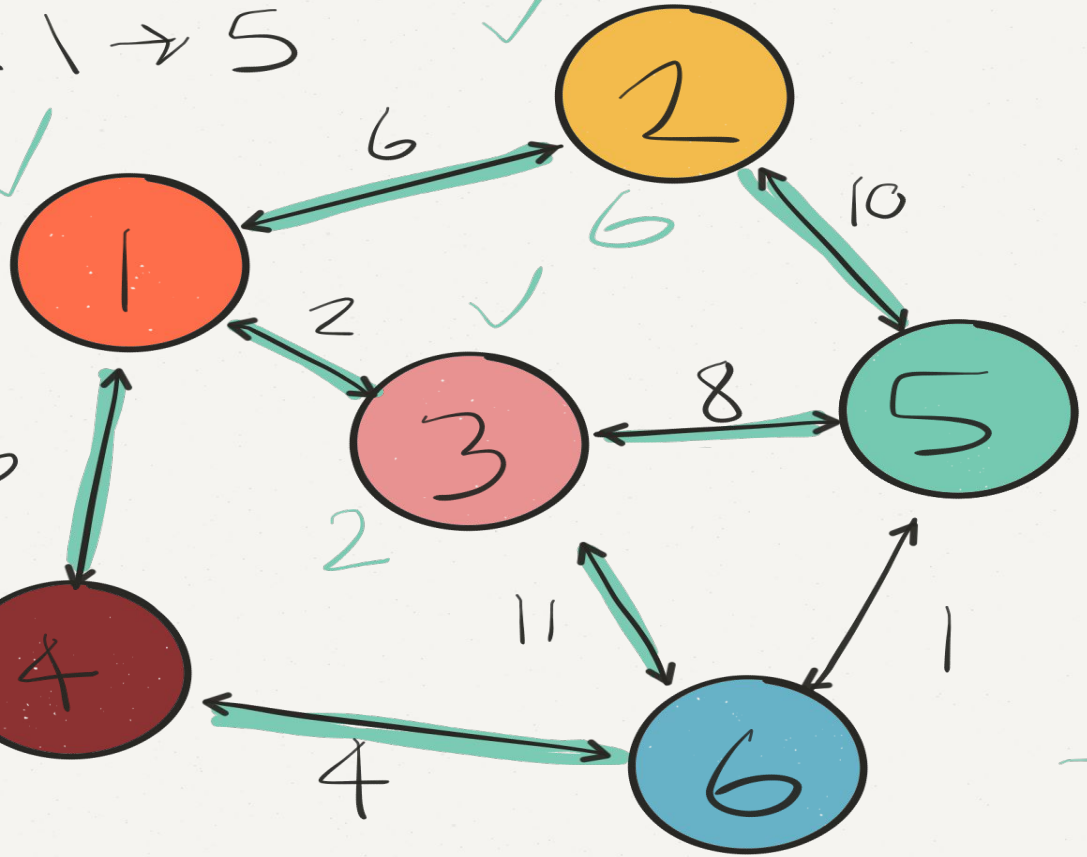
Visited: 1 2 3


Path 1 → 5
1
2
3
4
5
6
6
10
2
3
8
3
2
11
1
4
3
4
13
Queue
6
5
4
Visited
1 2 3


Path 1 → 5
Queue
1
2
3
4
5
6
6
10
2
3
8
11
1
4
6
10
2
3
13
Visited: 1 2 3 4
6
5


Path 1 → 5
Queue
1
2
3
4
5
6
6
2
10
2
3
8
3
10
2
11
1
3
4
13
7
Visited: 1 2 3 4
6
5

Path 1 → 5



Queue

10

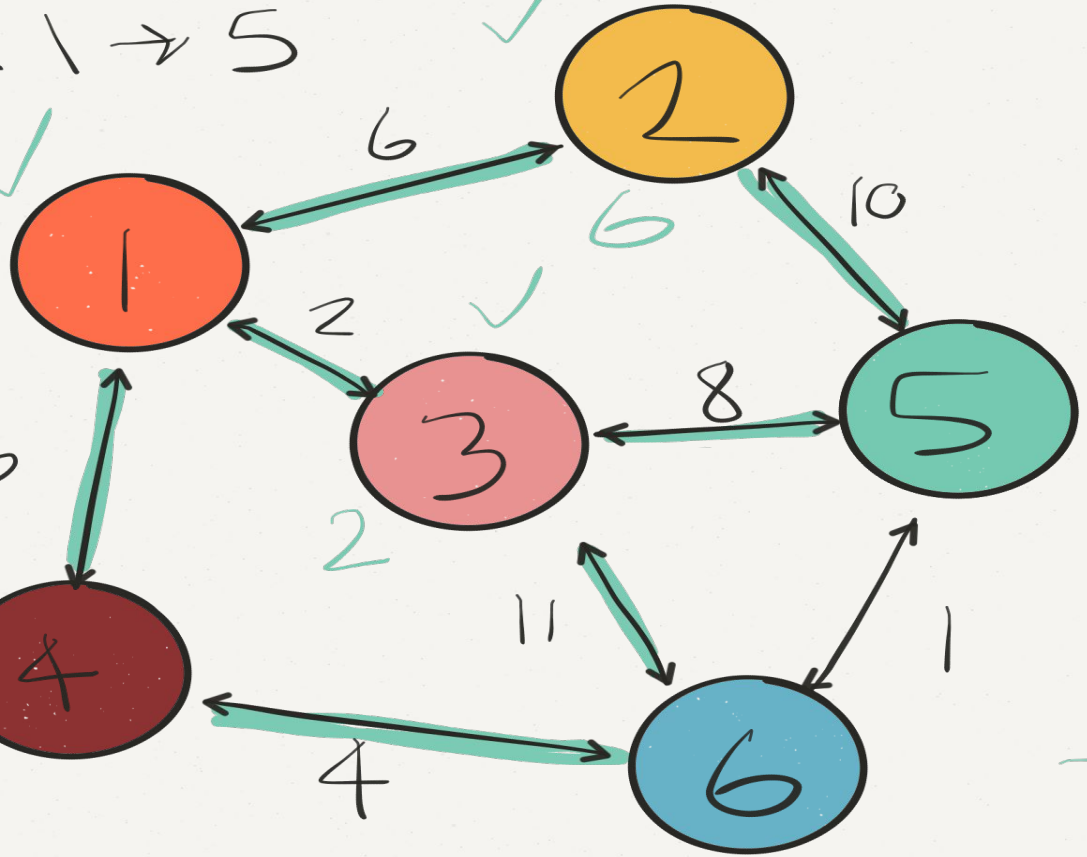
6

7

3

Visited: 1 2 3 4

Path 1 → 5



Queue

10

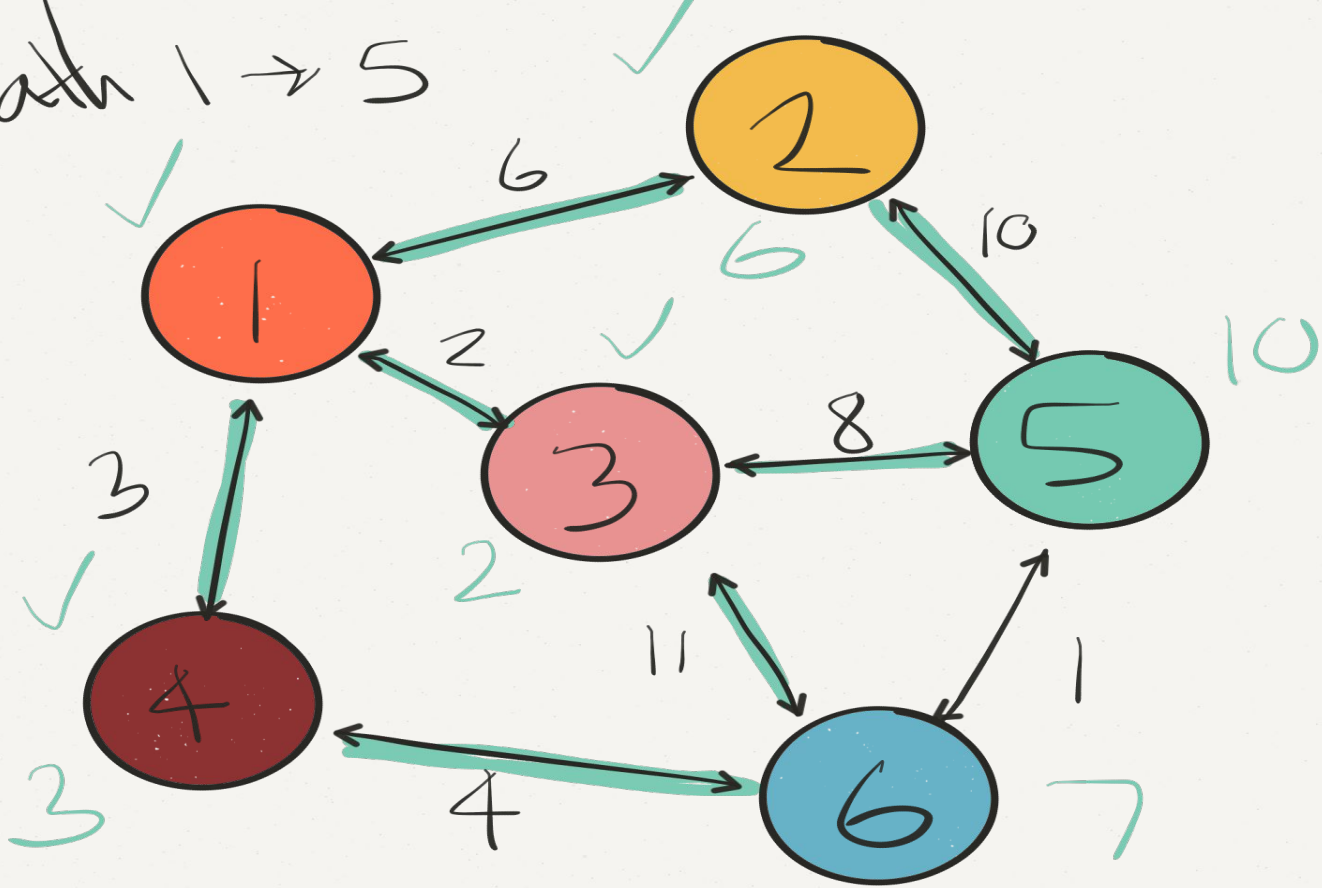
6

7

3

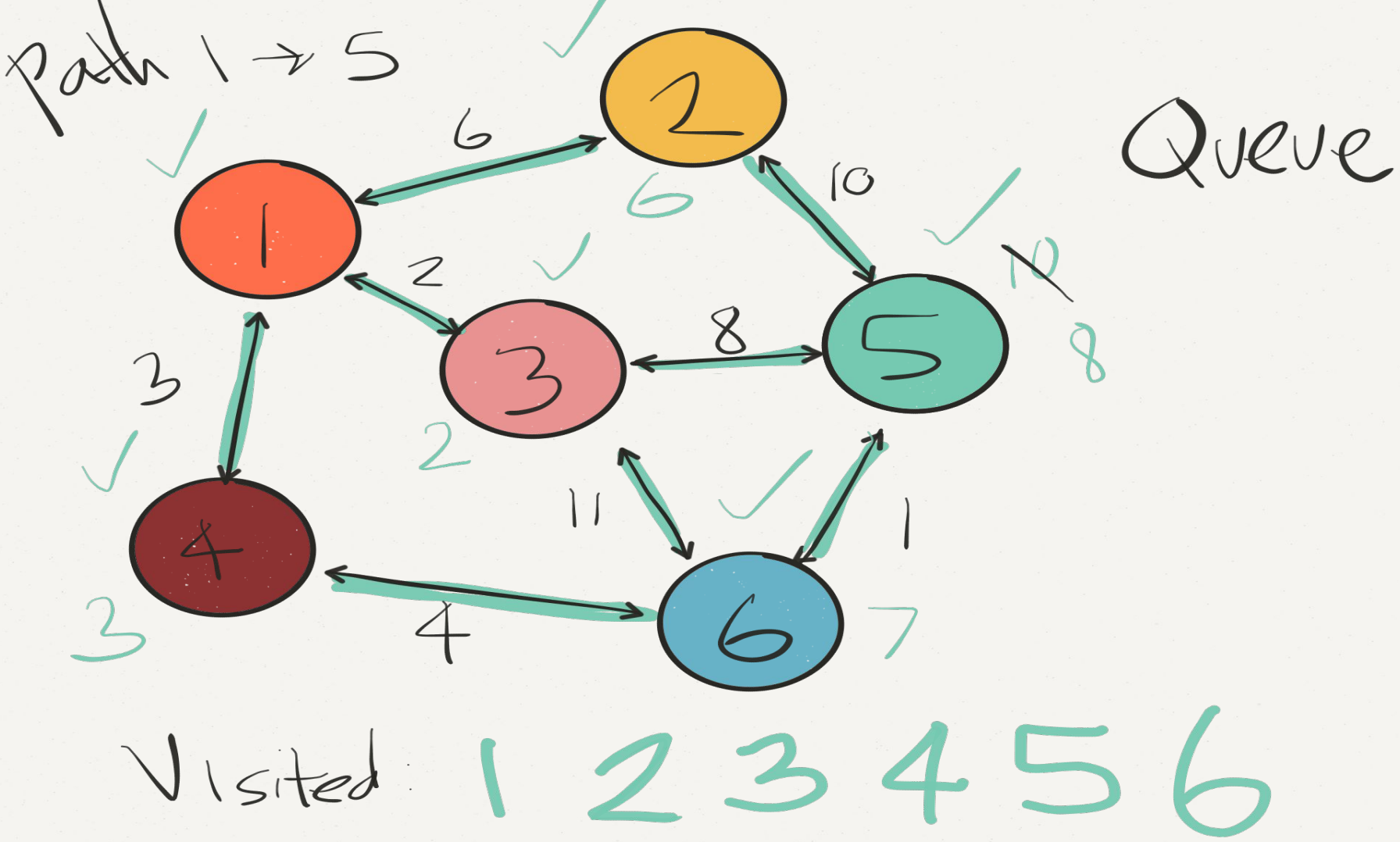
Visited: 1 2 3 4

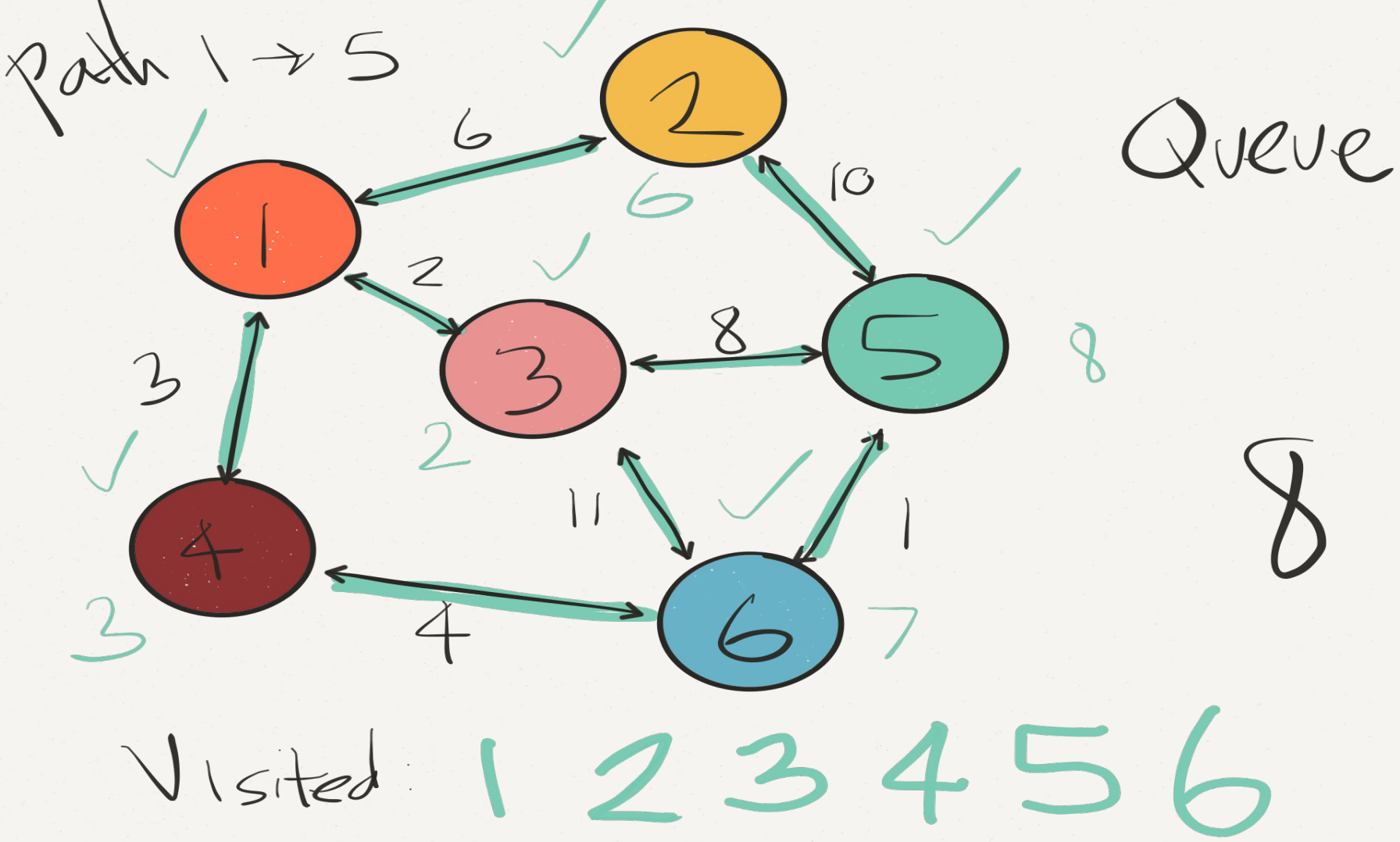
Path 1 → 5



Queue

Visited: 1 2 3 4 6


Path 1 → 5
Queue
1
2
3
4
5
6
6
10
2
8
3
11
1
4
Visited: 1 2 3 4 5 6


Path 1 → 5
1
2
3
4
5
6
6
10
2
8
3
11
1
4
Queue
8
Visited: 1 2 3 4 5 6

When to use graphs?

- Recognize the problem
  - network
  - grid
  - intertwined relationships
- Key words
  - vertices, nodes, edges, connections, connectivity, paths, cycles, and direction

Real world applications

- Many applications and problems involving graphs
  - Mapping systems
    - Shortest path between two points
    - Shortest path through multiple vertices
  - Ranking systems for search engines
  - Social networks
  - Pipe flow problems

Related Algorithms

- Traveling salesman problem
  - How to minimize path to reach every vertex in a graph
- Hamiltonian Cycle
  - Is there a path that visits each vertex once
- Euler Cycle
  - Is there a path that visits each edge once