Graph algorithms and Breadth-first search

Lecturer: Arran Stewart

Outline

- What is graph search?
- What is breadth-first search (BFS)?

Graph algorithms

We said in previous lectures that what graph representation is best depends on what sort of questions we want to ask about a graph.

Questions we might ask are things like:

- What is the sum of all the vertices in a graph G, where each vertex holds an `int`?
- What is the shortest weighted path from one vertex (say, a vertex representing Perth airport) to another (say, a vertex representing Chongqing Jiangbei International Airport)?
- What is the least costly way of connecting vertices representing electric power stations and buildings?

Graph algorithms

Questions like

- What is the sum of all the vertices in a graph G, where each vertex holds an `int`?

are examples of *graph traversal* – they ask us to visit every vertex in the graph.

Two possible ways of performing graph traversal we could consider are:

- breadth-first search (BFS):
  - From an initial vertex, imagine starting *multiple* paths to neighbours, and advance each path one step at a time
- depth-first search (DFS):
  - Put the edges leading out of a starting vertex in some order, left to right, and continue exploring the left-most path out of each vertex until you reach an end to the path. Then start exploring one-after-the-left, then the second after the left, and so on ("deepest" first)

Depth-first search (DFS)

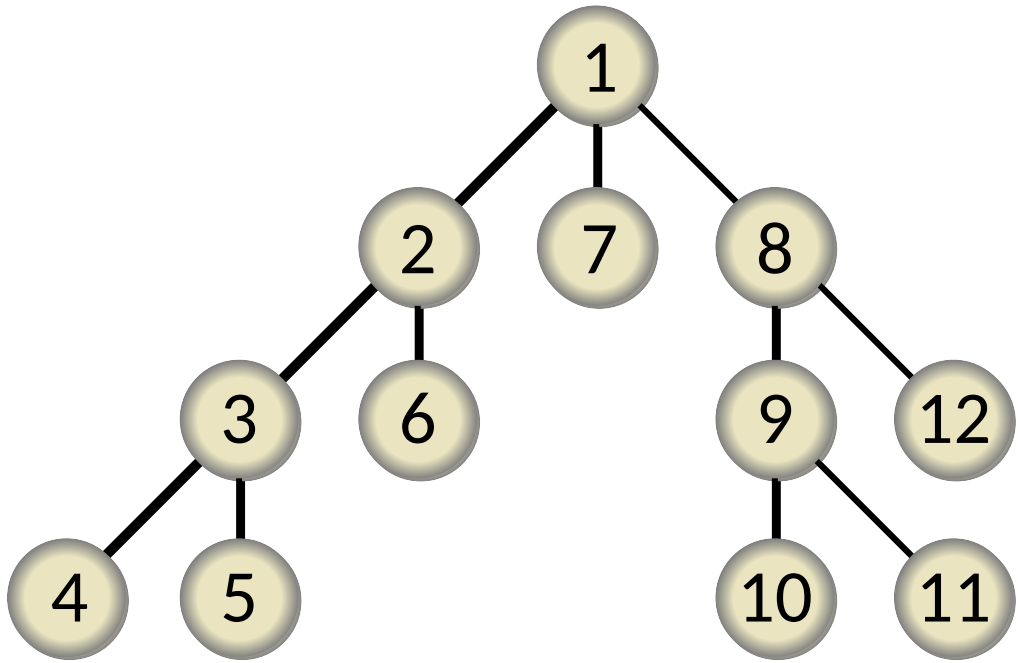
We have already encountered these terms (BFS and DFS) when looking at *trees*.

We said that for trees, depth-first search results from doing a "pre-order" traversal of a tree.

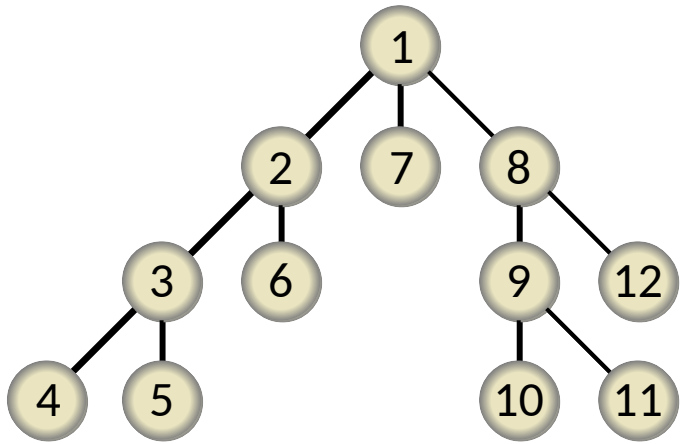
- Recall that the mnemonic for pre-order binary tree traversal is "NLR" (visit a *N*ode, then its *L*eft child, then the *R*ight, recursively).

Depth-first search (DFS)

The following diagram shows the order nodes would be visited in a graph when doing a depth-first search.



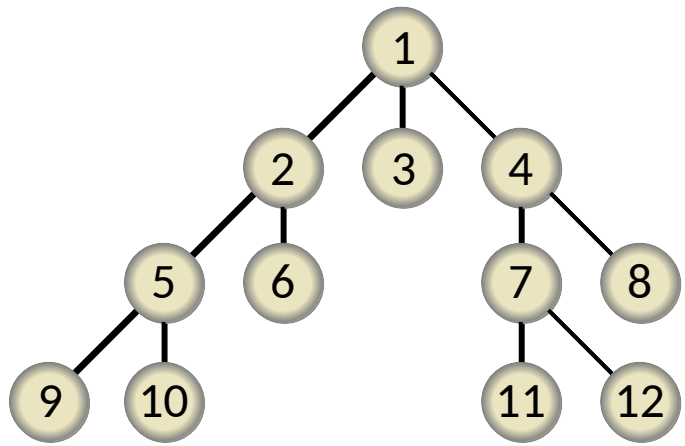
Depth-first search (DFS)



This graph happens to be a *tree*; trees are just a subset of graphs, where nodes in the tree are vertices in the graph, and an edge exists between any child node in the tree and its parent.

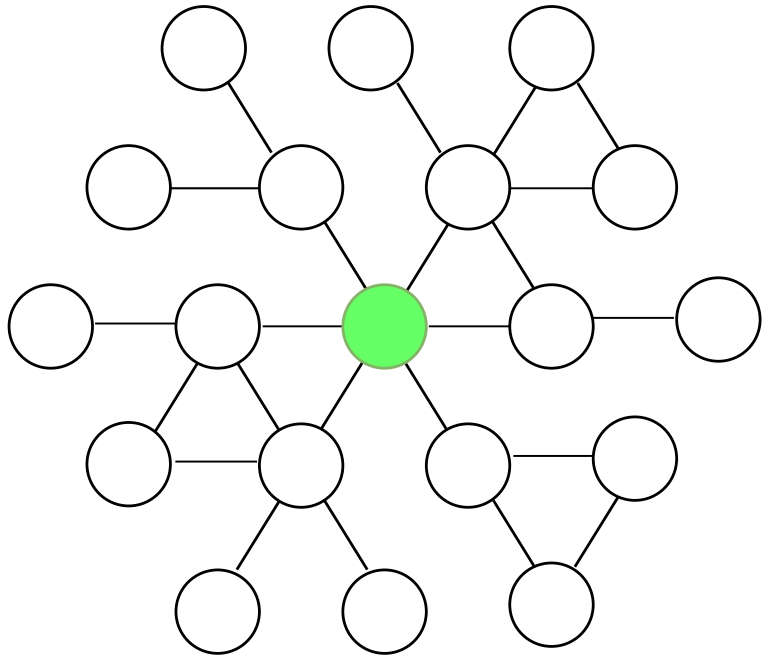
Breadth-first search (BFS)

Recall that for a tree, breadth-first search visits nodes "level by level"; we can do the same for graphs.



Breadth-first search

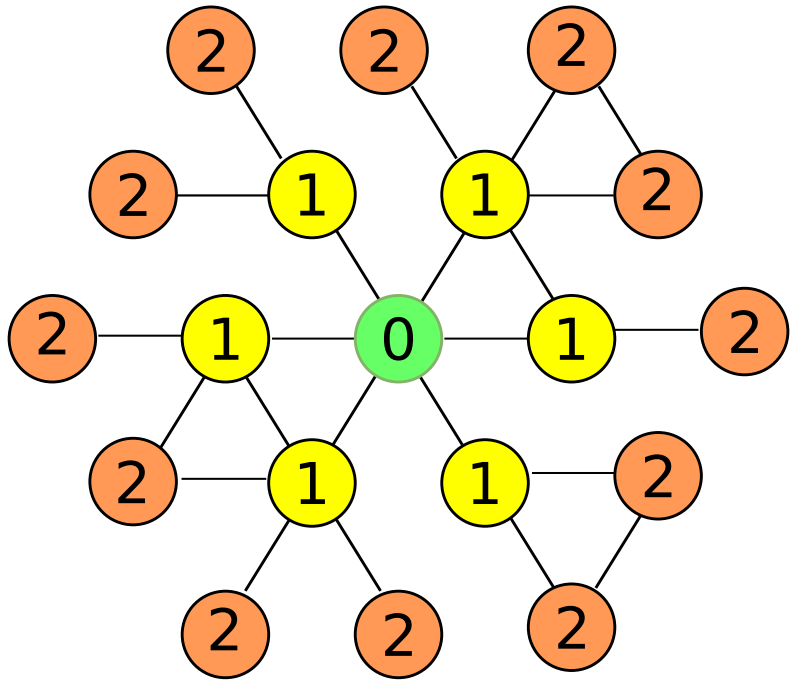
We can apply breadth-first search to graphs which are *not* trees, as well, like the following:



- We start with an initial node n, then we visit nodes which are one link distant from n, then 2, and so on.

Breadth-first search

The order we visit the nodes would be as follows



(where nodes with the same number are visited in some arbitrary order).

Previously, we only explained breadth-first search informally; we will now explain an algorithm for doing breadth-first search.

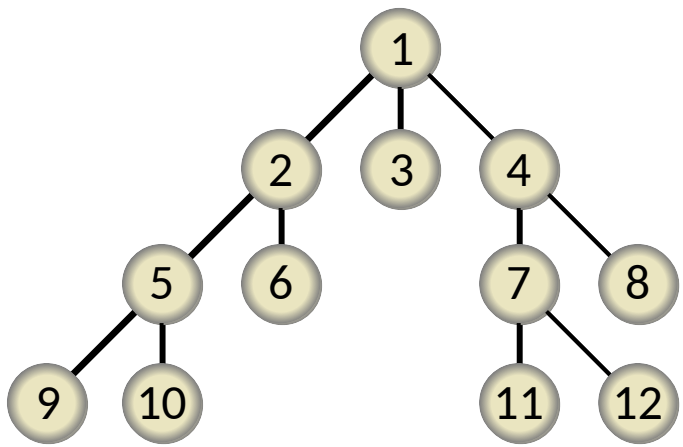
Breadth-first search (BFS)

This algorithm starts from a given vertex v and explores a graph G in a *breadth-first* manner.

We shall see also that as it does so, it constructs what we call a *spanning tree* for graph G, called the breadth-first tree.

Breadth-First Search Algorithm (BFS)

- The algorithm uses a **queue** as its main data structure.
- The basic idea is simple:
  - Starting with a queue containing just one node, consider the neighbours of each node.
  - Visit any new vertices from the neighbours, until there are no more vertices to be added.



Implementing BFS

In order to implement BFS we maintain three data structures:

1. **Q**, the queue of vertices to be processed
2. An array **colour,** with values white (0), grey (1) or black (2) for each vertex.
3. An array **p,** which where **p**(v) contains the immediate parent of v in the spanning tree. That is the edges of the spanning tree are given by (v, **p**(v))

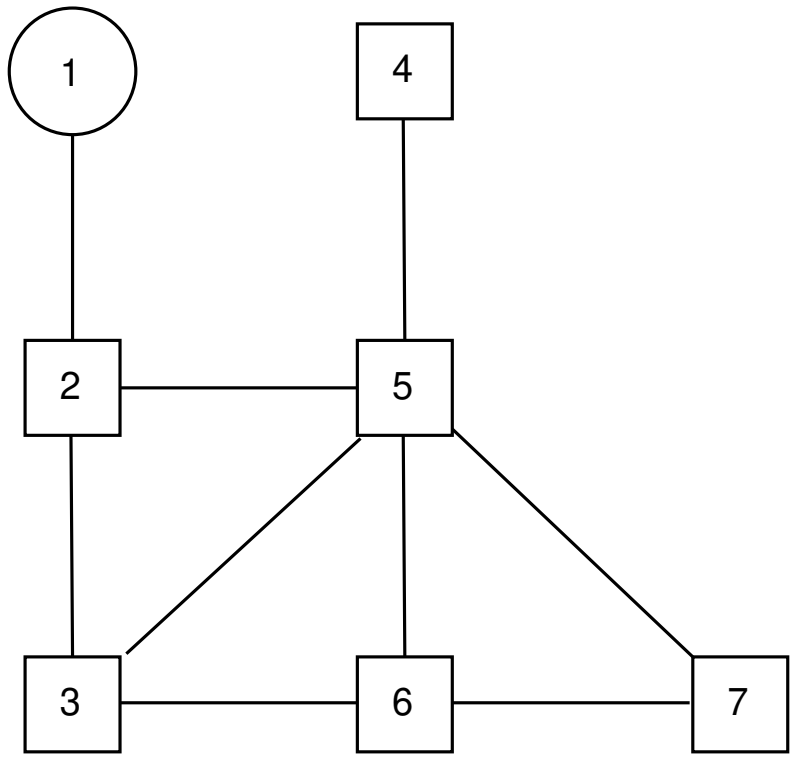
BFS Algorithm

```
- put v into the queue Q
- colour[v]=white for all vertices v,
- p(v) is undefined.
- while Q is not empty:
  - dequeue w from the head of Q
  - for each vertex x adjacent to w:
    - if colour[x] is white:
      - p(x) = w
      - colour[x] = grey
      - enqueue x onto the tail of Q
  - colour[w] = black
```

BFS Algorithm

At the end of the BFS search, every node will have the colour black and the parent array **p** will contain details of a BFS spanning tree.

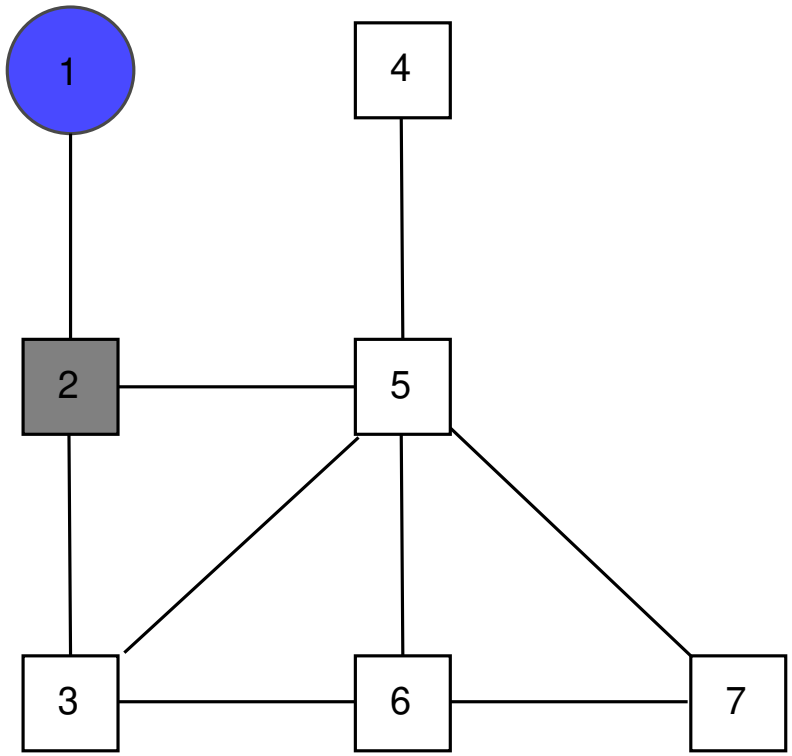
Let us examine how the BFS algorithm works on the following graph:



Initially, our queue Q contains 1.

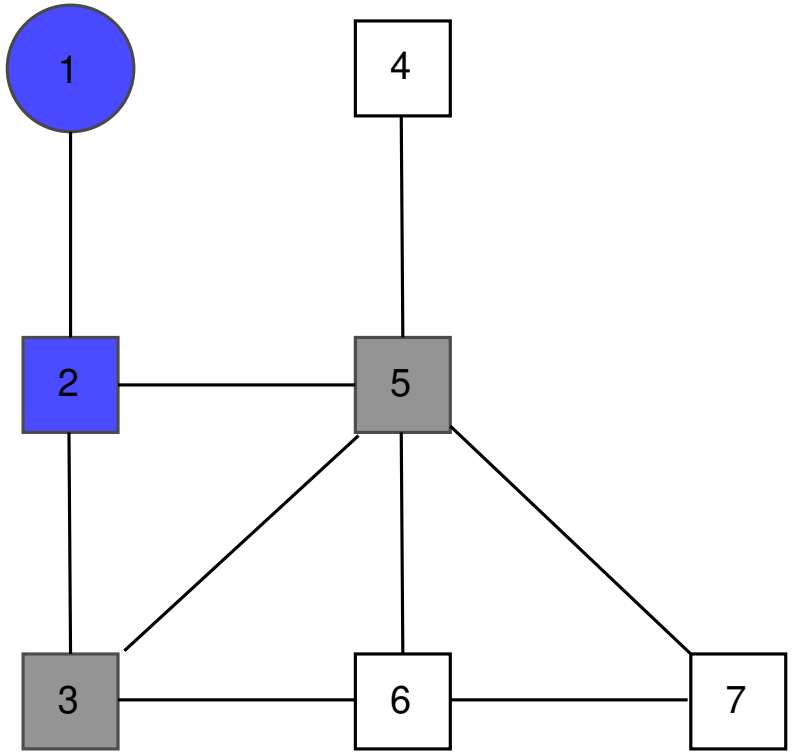
We will ignore the use of the p array for the moment – but basically, each time we add something to the queue, we use p to store where the search came *from* when moving to some node n.

BFS step 1



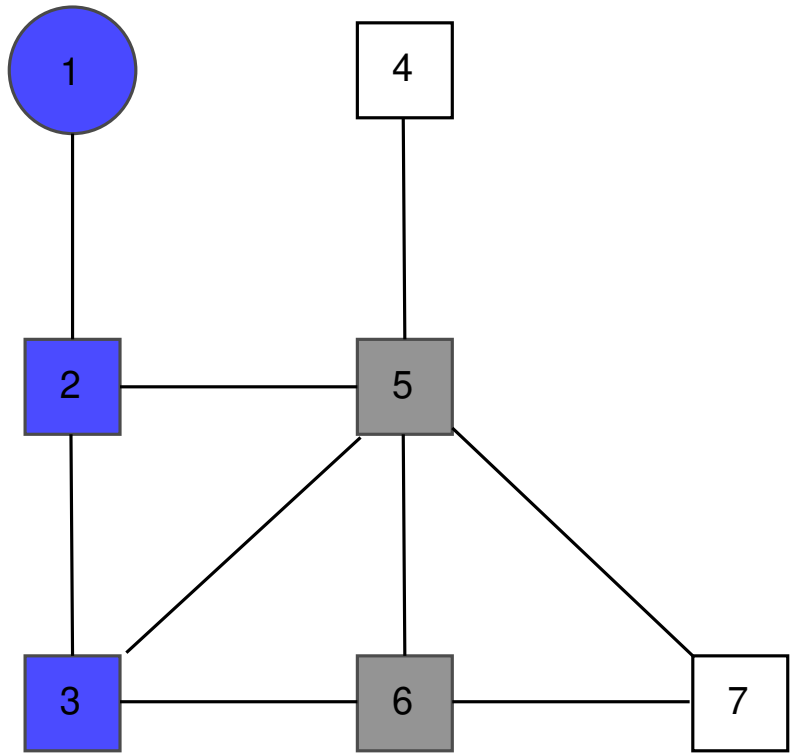
- Q = [1]
- Dequeue 1 from the queue Q
- Colour 2 grey and add it to the queue
- Colour 1 black

BFS step 2



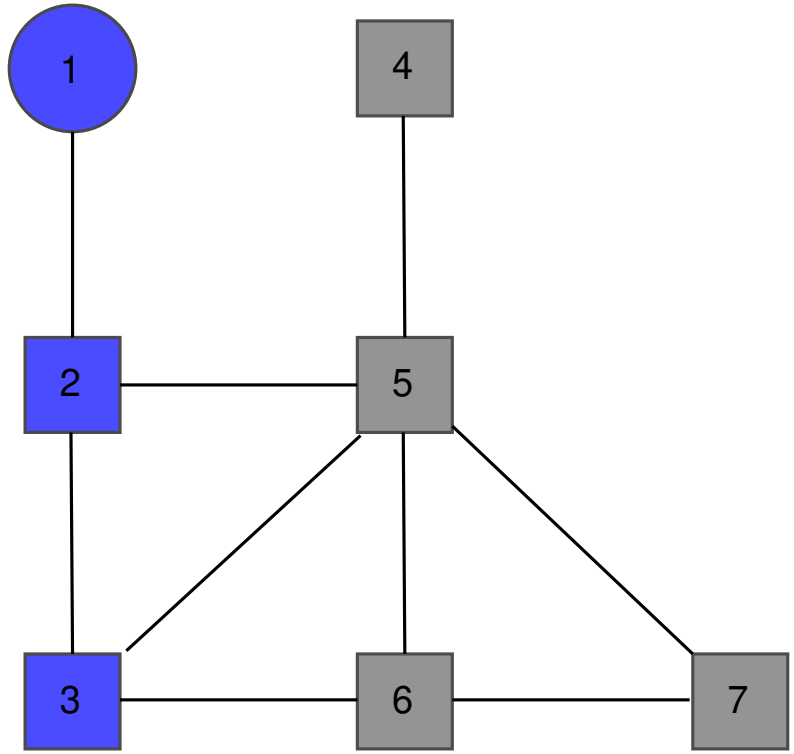
- Q = [2]
- Dequeue 2 from the queue Q
- Colour 3 grey and add it to the queue
- Colour 5 grey and add it to the queue
- Colour 2 black

BFS step 3



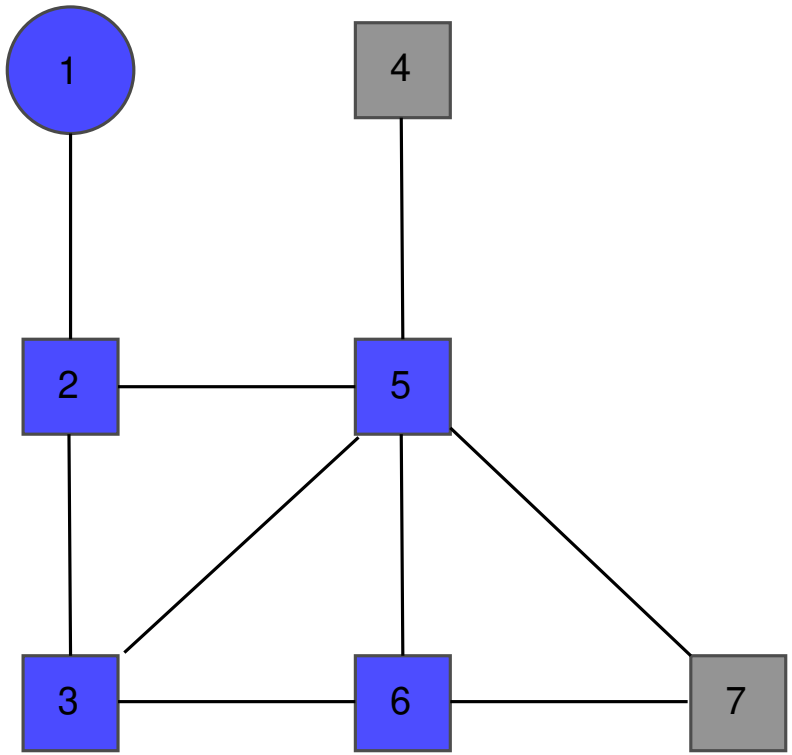
- Q = [3, 5]
- Dequeue 3 from the queue Q
- Colour 6 grey and add it to the queue
- Colour 3 black

BFS step 4



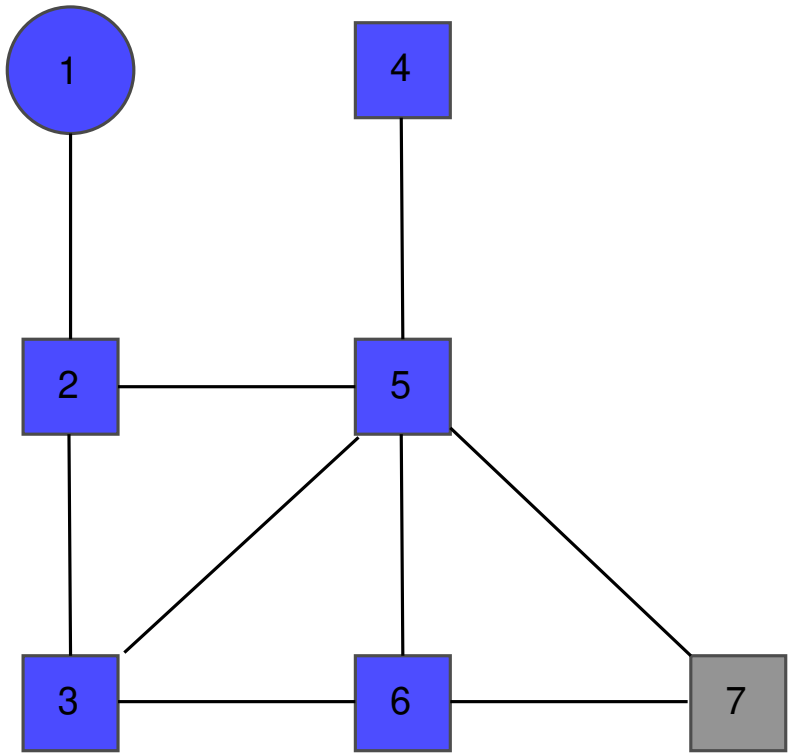
- Q = [5, 6]
- Dequeue 5 from the queue Q
- Colour 4 grey and add it to the queue
- Colour 7 grey and add it to the queue
- Colour 5 black

BFS step 5



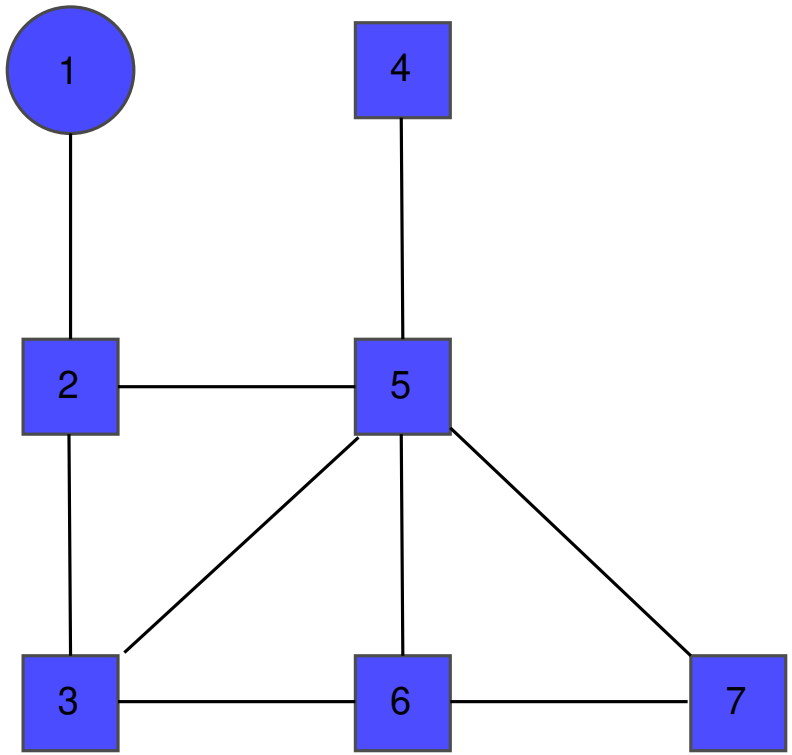
- Q = [6, 4, 7]
- Dequeue 6 from the queue Q
- Colour 6 black

BFS step 6



- Q = [4, 7]
- Dequeue 4 from the queue Q
- Colour 4 black

BFS step 7



- Q = [7]
- Dequeue 7 from the queue Q
- Colour 7 black

BFS Java code

The code for the BFS algorithm is provided in BFS.java in your Java code bundle.

BFS and "shortest path" problems

Breadth-first search also provides us with a way of answering questions like

- Given two vertices A and B in a graph G, what is the shortest path between them?

BFS and "shortest path" problems

- Given two vertices A and B in a graph G, what is the shortest path between them?

To answer this question, we just do breadth-first search *starting* from vertex A.

And each time we visit a vertex, we check whether we've found vertex B; if we have, we stop. And the p array, which stores the "path back" to node A, is the shortest route from A to B.