

Breadth first search

Lecturer: Arran Stewart

Outline

- ▶ What is a hash table?
- ▶ Implementing hash tables

Breadth-first search

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))

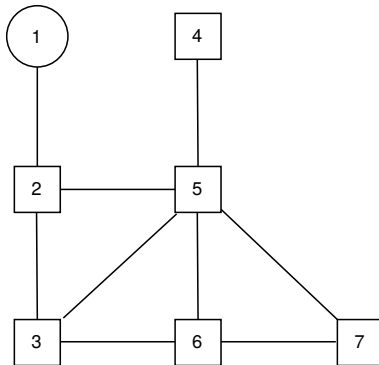
BFS Algorithm

- put v into the queue Q
- $\text{colour}[v] = \text{white}$ 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 $\text{colour}[x]$ is white:
 - $p(x) = w$
 - $\text{colour}[x] = \text{grey}$
 - enqueue x onto the tail of Q
- $\text{colour}[w] = \text{black}$

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.

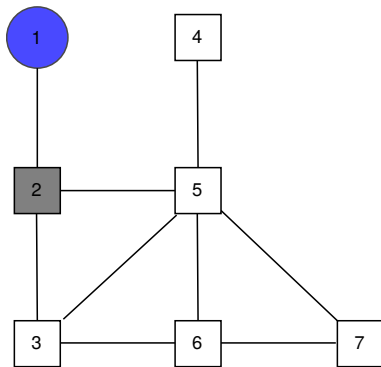
Let us examine how the BFS algorithm works on the following graph:



Initially, our queue Q contains 1.

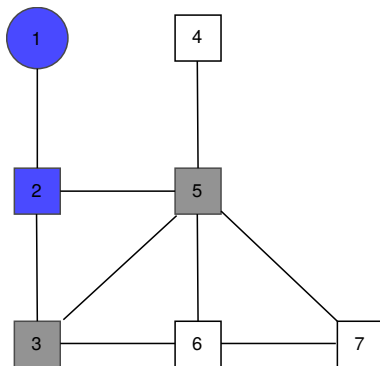
We will ignore the use of the p array for the moment – but basically, each time we add something to the queue, we use p to store where the search came *from* when moving to some node n .

BFS step 1



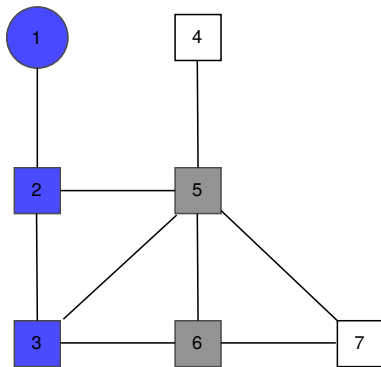
- ▶ $Q = [1]$
- ▶ Dequeue 1 from the queue Q
- ▶ Colour 2 grey and add it to the queue
- ▶ Colour 1 black

BFS step 2



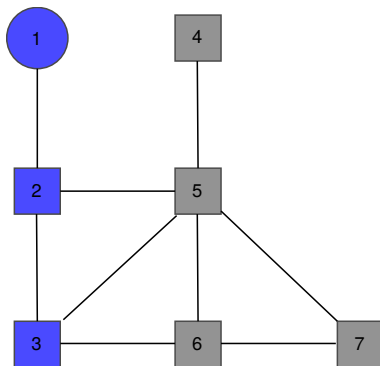
- ▶ $Q = [2]$
- ▶ Dequeue 2 from the queue Q
- ▶ Colour 3 grey and add it to the queue
- ▶ Colour 5 grey and add it to the queue
- ▶ Colour 2 black

BFS step 3



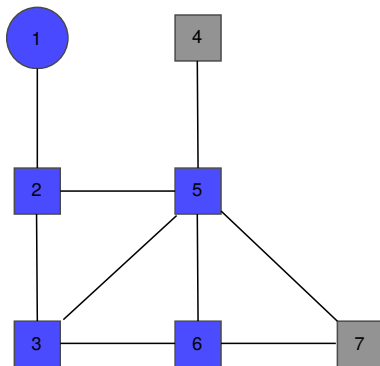
- ▶ $Q = [3, 5]$
- ▶ Dequeue 3 from the queue Q
- ▶ Colour 6 grey and add it to the queue
- ▶ Colour 3 black

BFS step 4



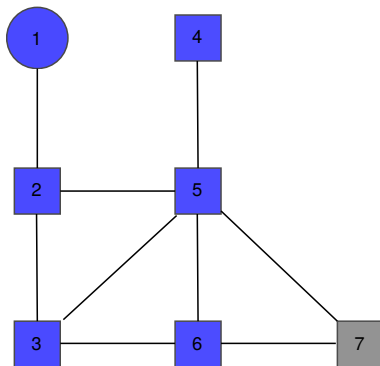
- ▶ $Q = [5, 6]$
- ▶ Dequeue 5 from the queue Q
- ▶ Colour 4 grey and add it to the queue
- ▶ Colour 7 grey and add it to the queue
- ▶ Colour 5 black

BFS step 5



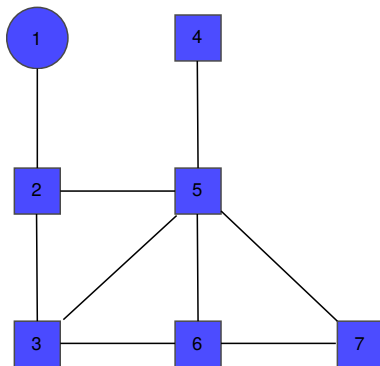
- ▶ $Q = [6, 4, 7]$
- ▶ Dequeue 6 from the queue Q
- ▶ Colour 6 black

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.