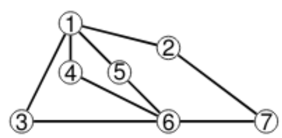
Breadth First Search

1 Graphs

- A graph $G = (V, E)$ is a pair of sets: vertices V and edges E
- To give an adjacency list representation of a graph, for each vertex v list all the vertices adjacent to v
- To given an adjacency matrix representation of a graph create a square matrix A and label the rows and columns with the vertices: the entry in row i column j is 1 if vertex j is adjacent to vertex i and 0 if it is not
- Can also represent a graph by an array of its edges

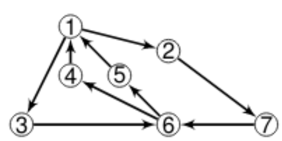
1.1 Representations

Undirected



$$\begin{array}{ll} 1: & 2,3,4,5 \\ 2: & 1,7 \\ 3: & 1,6 \\ 4: & 1,6 \\ 5: & 1,6 \\ 6: & 3,4,5,7 \\ 7: & 2,6 \end{array} \qquad \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Directed



$$\begin{array}{ll} 1: & 2,3 \\ 2: & 7 \\ 3: & 6 \\ 4: & 1 \\ 5: & 1 \\ 6: & 4,5 \\ 7: & 6 \end{array} \qquad \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

For each representation:

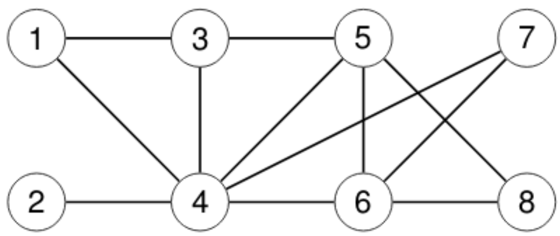
- How much space do we need to store it?
- How long does it take to initialize an empty graph?
- How long does it take to make a copy?
- How long does it take to insert an edge?
- How long does it take to list the vertices adjacent to a vertex u?
- How long does it take to find out if the edge (u,v) belongs to G?

2 Breadth-First Search

- Input: a graph $G = (V, E)$ and a source vertex s
- Aim: to find the distance from s to each of the other vertices in the graph
- Idea: send out a **wave** from s
  - The wave first hits vertices at distance 1
  - Then the wave hits vertices at distance 2
  - and so on

- BFS maintains a queue that contains vertices that have been discovered but are waiting to be processed
- BFS colours the vertices:
  - White indicates that a vertex is undiscovered
  - Grey indicates that a vertex is discovered but unprocessed
  - Black indicates that a vertex has been processed
- The algorithm maintains an **array** d (distance)
  - $d[s] = 0$ where s is the source vertex
  - if we discover a new vertex v while processing u, we set $d[v] = d[u] + 1$

2.1 Example



- Initialization: source vertex grey, others are while; distance to source is 0; add source to the queue
- While the queue is not empty
  - Remove first vertex v from the queue
  - add white neighbours of v to queue and colour them grey; distance is 1 greater than to v
  - colour v black

2.2 Pseudocode

```
BFS (G, s)
for each vertex u ∈ V[G] − {s}
    do colour[u] ← WHITE
        d[u] ← ∞
            π[u] ← NIL
colour[s] ← GREY
d[s] ← 0
π[s] ← NIL
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    do u ← DEQUEUE(Q)
        for each v ∈ Adj[u]
            do if colour[v] = WHITE
                then colour[v] = GREY
                    d[v] ← d[u] + 1
                    π[v] ← u
                    ENQUEUE(Q, v)
        colour[u] ← BLACK
```

2.3 Analysis of running time

- We want an upper bound on the worst case running time
- Assume that it takes constant time for each operation such as to teat and update colours, to make changes to distance (and predecessor) and to enqueue and dequeue
- Initialization takes time $O(V)$
- Each vertex enters (and leaves) the queue exactly one. So queueing operations take $O(V)$
- In the loop the adjacency lists of each vertex are scanned. Each list is read once, and the combined lengths of the lists is $O(E)$
- Thus the total running time is $O(V + E)$

2.4 More than distances

- What if as well as finding the distance to each vertex, we want to be able to find a shortest possible path from the source to each vertex?
  - Recursively ask predecessors of nodes until you get back to the start node
  - BFS used the predecessor of v and v for each vertex v. Note that the predecessor is denoted by Π
  - The path from the source S in the Breadth First Tree is a shortest path from S to V
- What should we add to the algorithm to achieve this?

3 Breadth-first search

- Note that the algorithm runs on both directed and undirected graphs
- Notice that the highlighted edges (the ones used to discover new vertices) form a tree: we call this a **Breadth-first tree**. A path from s to another vertex v through the tree is the shortest path between s and v
- The predecessor of a vertex is the one from which is was discovered (i.e. its parent in the Breadth-first tree). We can record predecessors in an array Π when we run the algorithm and then use this array to construct the breadth-first tree