CS 103 BFS Alorithm

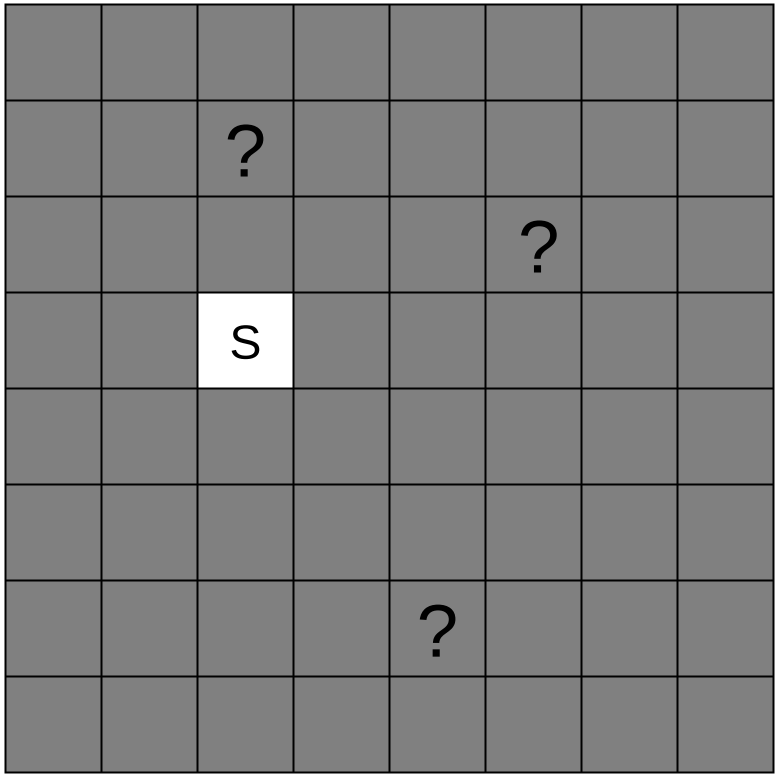
Mark Redekopp

Breadth-First Search (BFS)

HIGHLIGHTED ALGORITHM

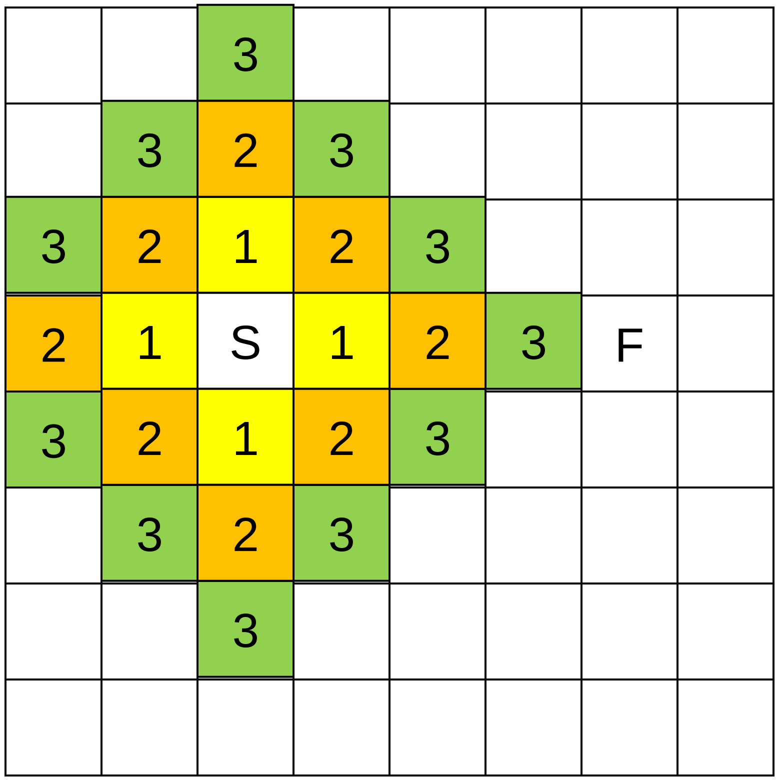
Path Planning

- We've seen BFS in the context of finding the shortest path through a maze



Path Planning

- We explore the 4 neighbors based on direction

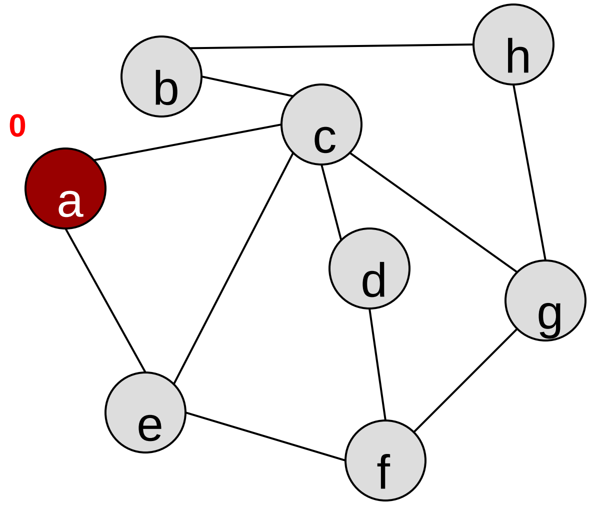


If you don't know where F is and want to find the shortest path, you have to do it this way

Uninformed search for shortest path:
Breadth-first

Breadth-First Search

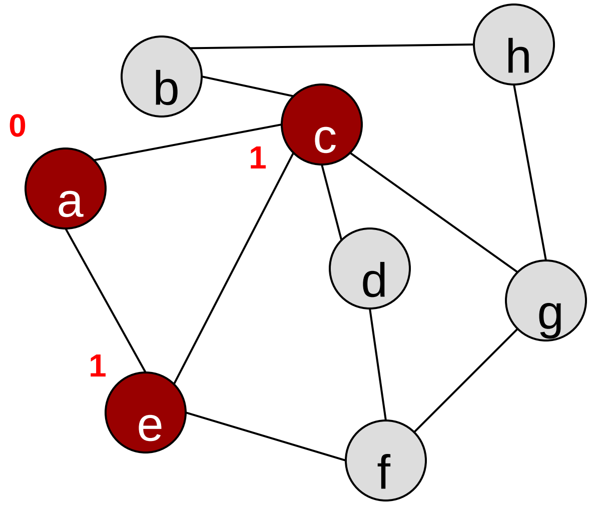
- Now let's generalize BFS to arbitrary set of connections/neighbors
- Given a graph with vertices, V, and edges, E, and a **starting vertex, u**
- BFS starts at **u** (‘a’ in the diagram to the left) and fans-out along the edges to nearest neighbors, then to their neighbors and so on
- Goal: Find the minimum number of hops (a.k.a. depth/distance) from the start vertex to every other vertex



Depth 0: a

Breadth-First Search

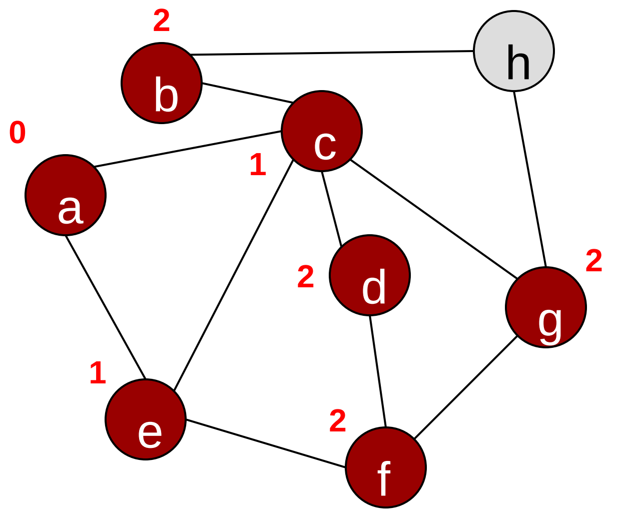
- Given a graph with vertices, V, and edges, E, and a starting vertex, u
- BFS starts at u ('a' in the diagram to the left) and fans-out along the edges to nearest neighbors, then to their neighbors and so on
- Goal: Find the minimum number of hops (a.k.a. depth) from the start vertex to every other vertex



Depth 0: a
Depth 1: c,e

Breadth-First Search

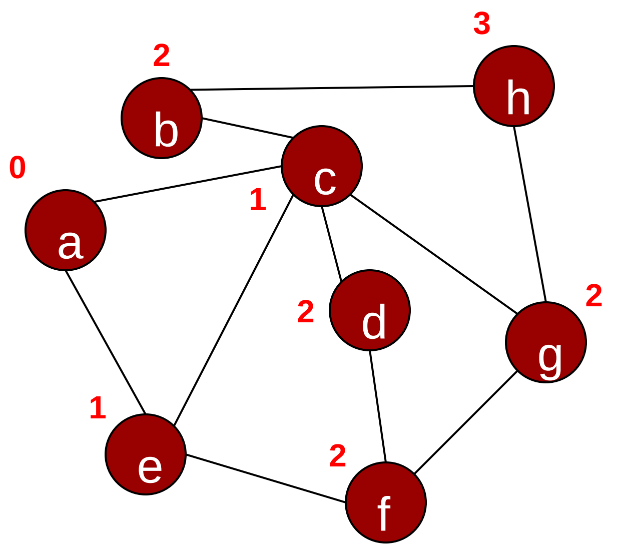
- Given a graph with vertices, V, and edges, E, and a starting vertex, u
- BFS starts at u (‘a’ in the diagram to the left) and fans-out along the edges to nearest neighbors, then to their neighbors and so on
- Goal: Find the minimum number of hops (a.k.a. depth) from the start vertex to every other vertex



Depth 0: a
Depth 1: c,e
Depth 2: b,d,f,g

Breadth-First Search

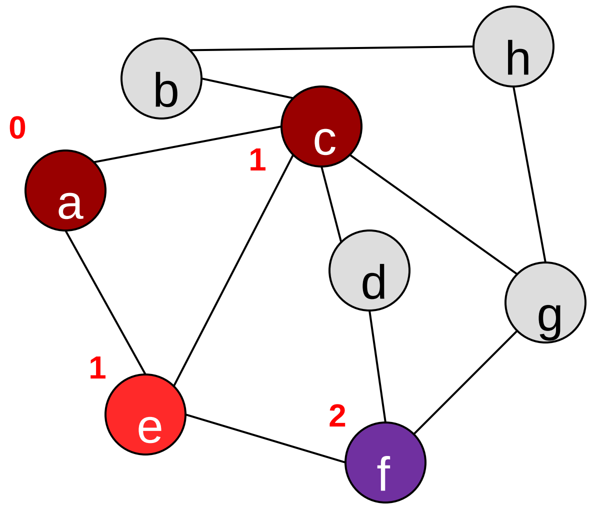
- Given a graph with vertices, V, and edges, E, and a starting vertex, u
- BFS starts at u ('a' in the diagram to the left) and fans-out along the edges to nearest neighbors, then to their neighbors and so on
- Goal: Find the minimum number of hops (a.k.a. depth) from the start vertex to every other vertex



Depth 0: a
Depth 1: c,e
Depth 2: b,d,f,g
Depth 3: h

Developing the Algorithm

- Key idea: Must explore all **nearer** neighbors before exploring further-away neighbors
- From 'a' we find 'e' and 'c'
  - Computer can only do one thing at a time so we have to pick either e or c to explore
  - Let's say we pick e…we will find f
  - Now what vertex should we explore (i.e. visit neighbors) next? Choices are c and f.
  - C!! (if we don't we won't find shortest paths…e.g. d)
  - Must explore all vetices at depth i before any vertices at depth i+1



Depth 0: a
Depth 1: c,e
Depth 2: b,d,f,g
Depth 3: h

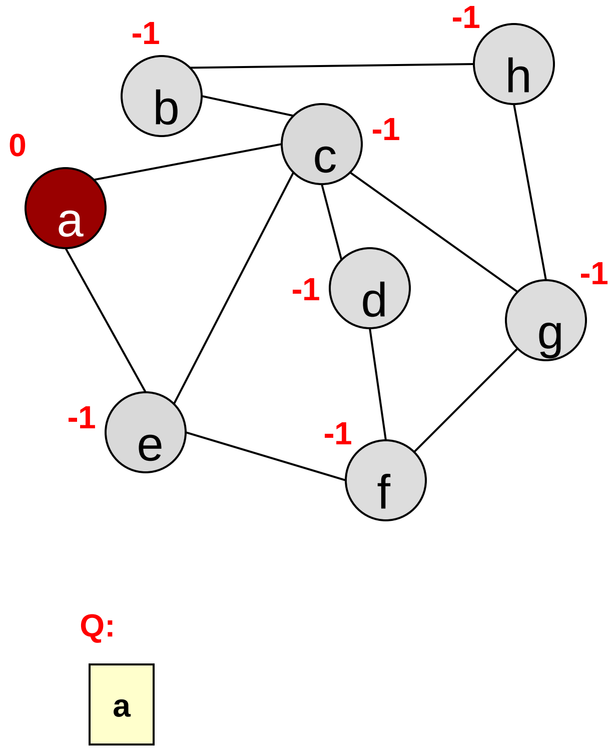
Developing the Algorithm

- Keep a first-in / first-out list (a.k.a. FIFO/first-come first-serve/queue/**deque**/etc.) of neighbors found
- Pull vertices out of the front of the list and explore their neighbors…when we find a new neighboring vertex we add it to the back of the list
- We don't want to put a vertex in the queue more than once…so we'll need to "mark" a vertex the first time we encounter it…we will only allow unmarked vertices to be put in the queue

Breadth-First Search

Algorithm:

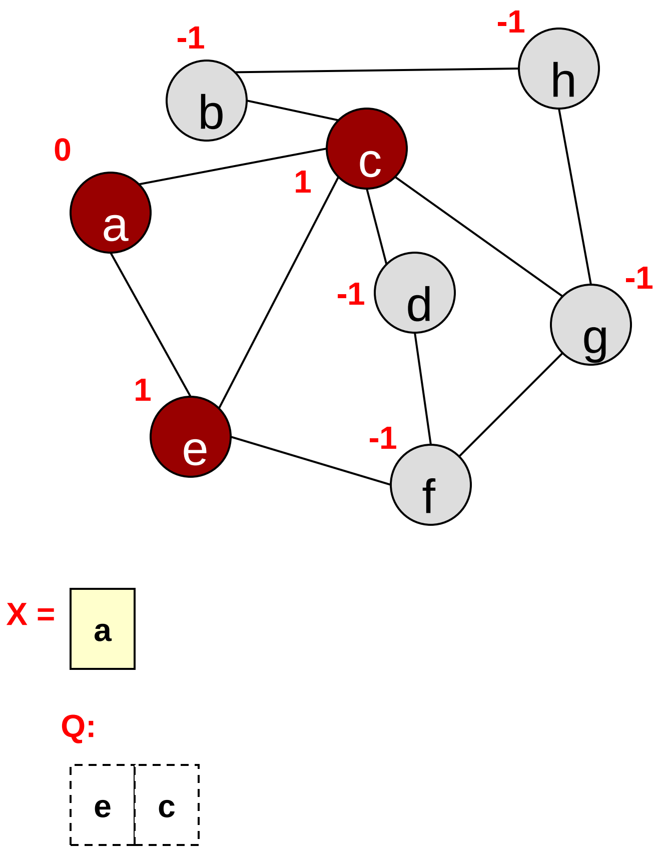
- Initialize all vertices as 'not found' by setting depth = -1
- Create a list, Q
- Add start vertex, u to Q
- Mark u as 'found' and depth = 0
- While(Q is not empty)
  - x = Remove front item
  - For each neighbor, y, of x
    - If vertex y is not found
      - Add y to back of the list, Q
      - Mark y as found by setting depth of y = depth of x + 1



Breadth-First Search

Algorithm:

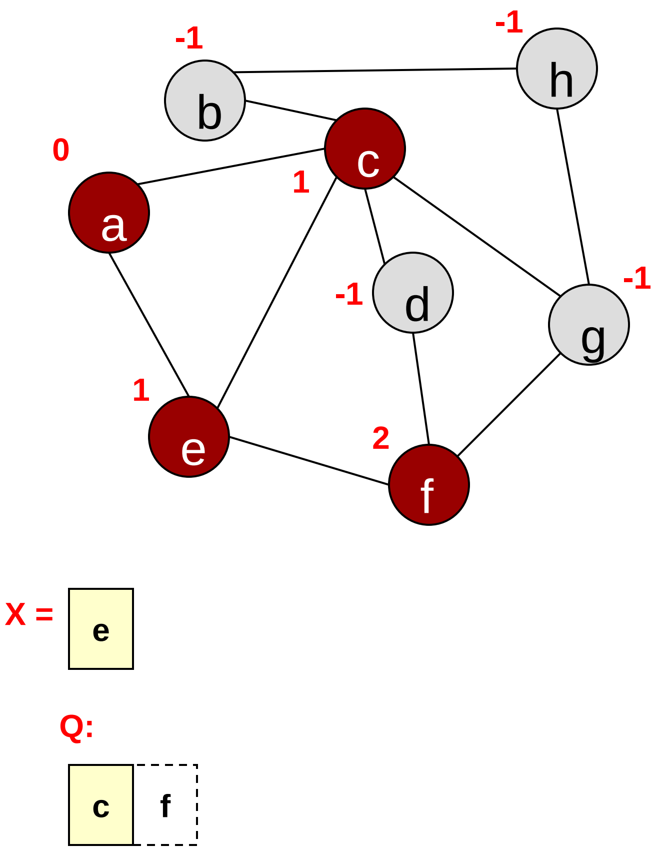
- Initialize all vertices as 'not found' by setting depth = -1
- Create a list, Q
- Add start vertex, u to Q
- Mark u as 'found' and depth = 0
- While(Q is not empty)
  - x = Remove front item
  - For each neighbor, y, of x
    - If vertex y is not found
      - Add y to back of the list, Q
      - Mark y as found by setting depth of y = depth of x + 1



Breadth-First Search

Algorithm:

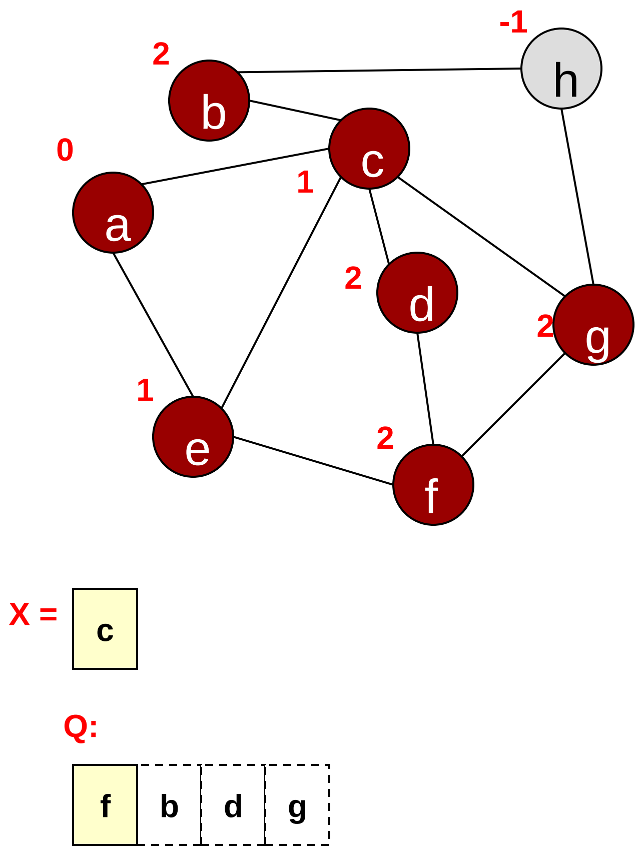
- Initialize all vertices as 'not found' by setting depth = -1
- Create a list, Q
- Add start vertex, u to Q
- Mark u as 'found' and depth = 0
- While(Q is not empty)
  - x = Remove front item
  - For each neighbor, y, of x
    - If vertex y is not found
      - Add y to back of the list, Q
      - Mark y as found by setting depth of y = depth of x + 1



Breadth-First Search

Algorithm:

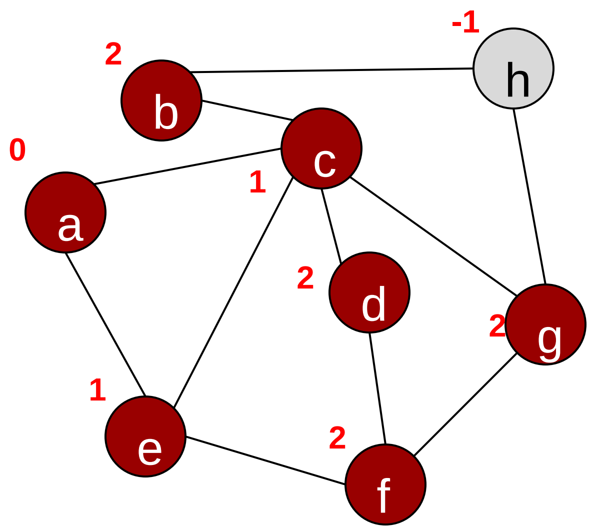
- Initialize all vertices as 'not found' by setting depth = -1
- Create a list, Q
- Add start vertex, u to Q
- Mark u as 'found' and depth = 0
- While(Q is not empty)
  - x = Remove front item
  - For each neighbor, y, of x
    - If vertex y is not found
      - Add y to back of the list, Q
      - Mark y as found by setting depth of y = depth of x + 1



Breadth-First Search

Algorithm:

- Initialize all vertices as 'not found' by setting depth = -1
- Create a list, Q
- Add start vertex, u to Q
- Mark u as 'found' and depth = 0
- While(Q is not empty)
  - x = Remove front item
  - For each neighbor, y, of x
    - If vertex y is not found
      - Add y to back of the list, Q
      - Mark y as found by setting depth of y = depth of x + 1



X =

f

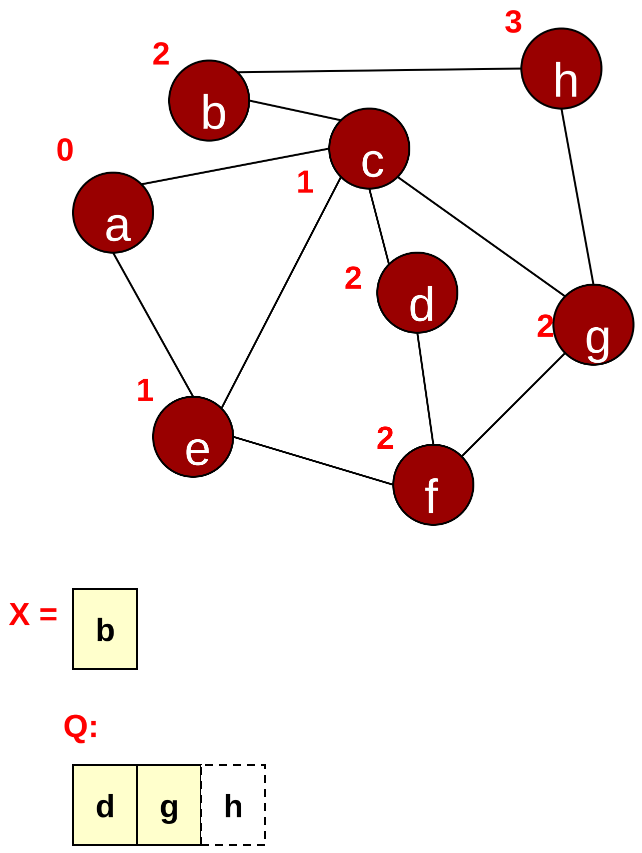
Q:

b	d	g

Breadth-First Search

Algorithm:

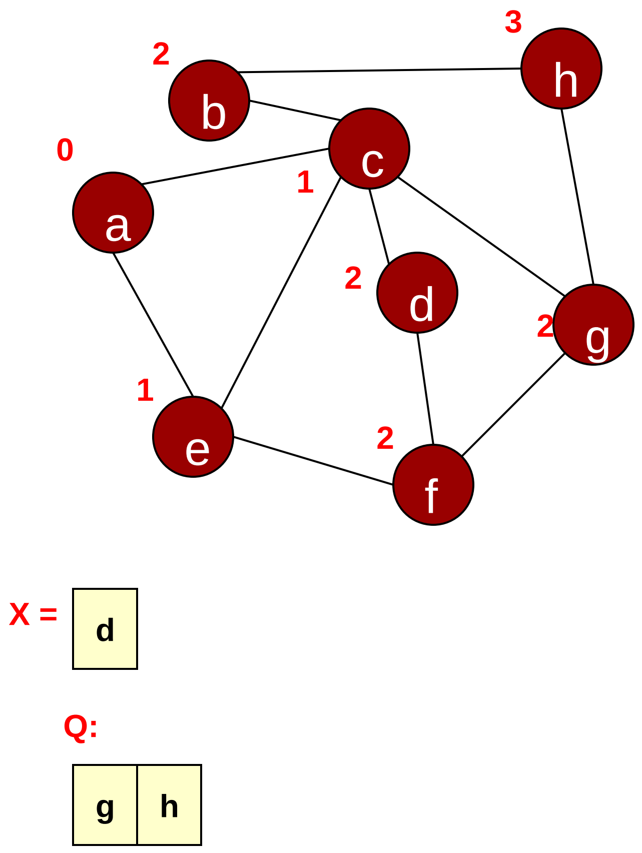
- Initialize all vertices as 'not found' by setting depth = -1
- Create a list, Q
- Add start vertex, u to Q
- Mark u as 'found' and depth = 0
- While(Q is not empty)
  - x = Remove front item
  - For each neighbor, y, of x
    - If vertex y is not found
      - Add y to back of the list, Q
      - Mark y as found by setting depth of y = depth of x + 1



Breadth-First Search

Algorithm:

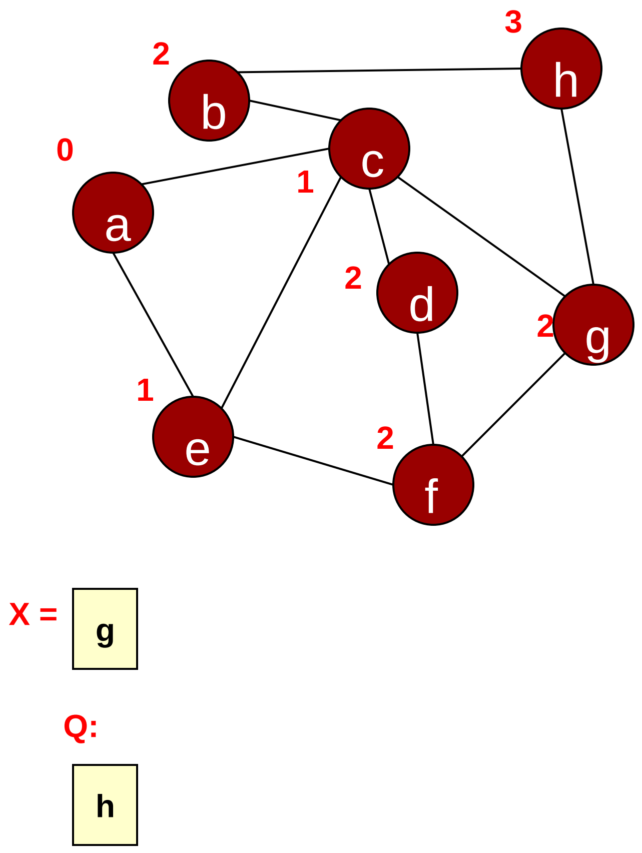
- Initialize all vertices as 'not found' by setting depth = -1
- Create a list, Q
- Add start vertex, u to Q
- Mark u as 'found' and depth = 0
- While(Q is not empty)
  - x = Remove front item
  - For each neighbor, y, of x
    - If vertex y is not found
      - Add y to back of the list, Q
      - Mark y as found by setting depth of y = depth of x + 1



Breadth-First Search

Algorithm:

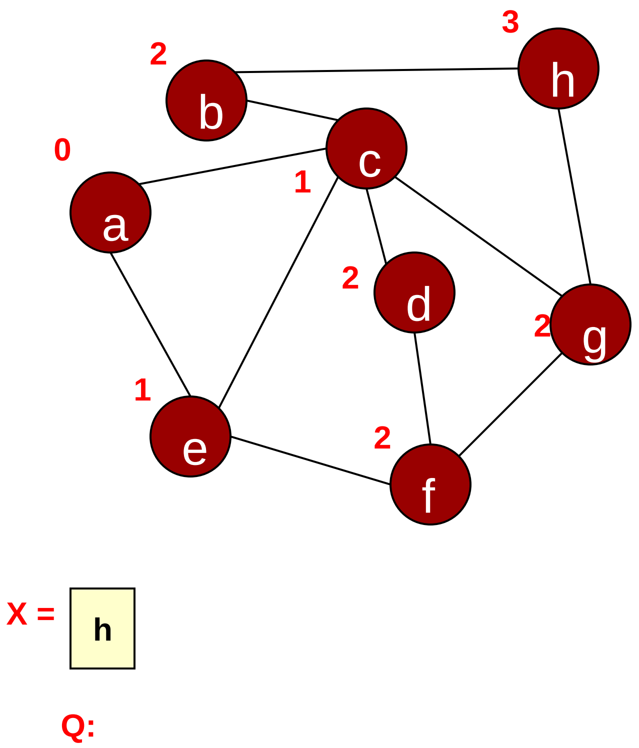
- Initialize all vertices as 'not found' by setting depth = -1
- Create a list, Q
- Add start vertex, u to Q
- Mark u as 'found' and depth = 0
- While(Q is not empty)
  - x = Remove front item
  - For each neighbor, y, of x
    - If vertex y is not found
      - Add y to back of the list, Q
      - Mark y as found by setting depth of y = depth of x + 1



Breadth-First Search

Algorithm:

- Initialize all vertices as ‘not found’ by setting depth = -1
- Create a list, Q
- Add start vertex, u to Q
- Mark u as ‘found’ and depth = 0
- While(Q is not empty)
  - x = Remove front item
  - For each neighbor, y, of x
    - If vertex y is not found
      - Add y to back of the list, Q
      - Mark y as found by setting depth of y = depth of x + 1



Tips for Implementing BFS in PA5

- Augment Users with a 'depth' and 'predecessor' field
  - Depth = -1 means not found yet
  - Predecessor is ID of User who found you
- 'friends' vector represents edges
- For the BFS queue we should use…
  - Deque
  - Place start vertex ID in it
- Continue processing vertices **while** the deque is not empty
  - Pull out vertices from front
  - Push newly found friends/users to the back
- After while loop, can traverse the predecessor trail or look at the depth of a specific user