CMPSCI 311: Introduction to Algorithms

Dan Sheldon

University of Massachusetts

Last Compiled: February 7, 2017

Graph Traversal

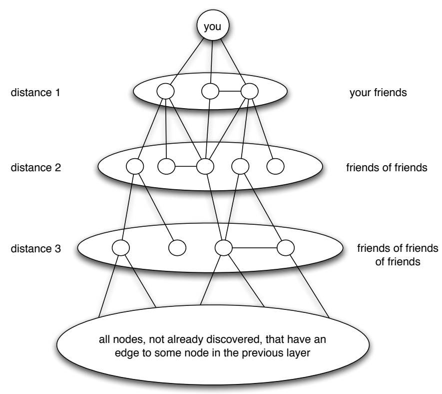
Thought experiment. World social graph.

- Is it connected?
- If not, how big is largest connected component?
- Is there a path between you and Barack Obama?

How can you tell algorithmically?
Answer: graph traversal! (BFS/DFS)

Breadth-First Search

Explore outward from starting node s by distance. "Expanding wave"



Breadth-First Search: Layers

Explore outward from starting node s.

Define layer L_i = all nodes at distance exactly i from s.

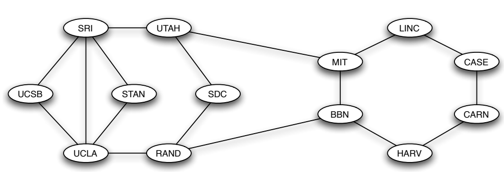
Layers

- $L_0 = \{s\}$
- L_1 = nodes with edge to L_0
- L_2 = nodes with an edge to L_1 that don't belong to L_0 or L_1
- ...
- L_{i+1} = nodes with an edge to L_i that don't belong to any earlier layer.

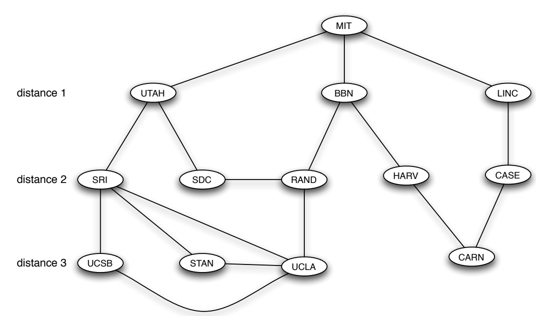
Observation: There is a path from s to t if and only if t appears in some layer.

BFS

Exercise: draw the BFS layers for a BFS starting from MIT

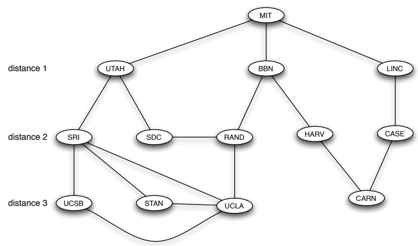


BFS Tree



We can use BFS to make a tree.

BFS Tree



Claim: let T be the tree discovered by BFS on graph $G = (V, E)$, and let (x, y) be any edge of G. Then the layer of x and y in T differ by at most 1.
Proof on board

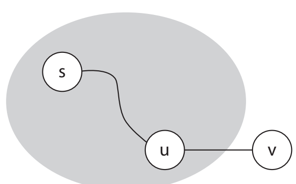
BFS and non-tree edges

Claim: let T be the tree discovered by BFS on graph $G = (V, E)$, and let (x, y) be any edge of G. Then the layer of x and y in T differ by at most 1.
Proof

- Let (x, y) be an edge
- Suppose $x \in L_i$, $y \in L_j$, and $j > i + 1$
- When BFS visits x, either y is already discovered or not.
  - If y is already discovered, then $j \leq i + 1$. Contradiction.
  - Otherwise since $(x, y) \in E$, y is added to L_{i+1}. Contradiction.

A More General Exploration Strategy

To explore the connected component containing s:

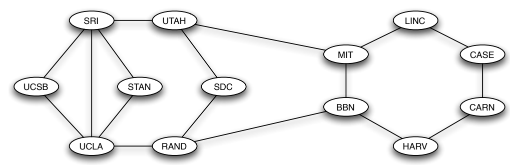


Add any node v for which

- (u, v) is an edge
- u is explored, but v is not

Depth-First Search

Depth-first search (DFS): keep exploring from the most recently added node until you have to backtrack.
Example.



Recursive DFS

DFS(u)
  Mark u as "Explored"
  for each edge (u, v) incident to u **do**
    if v is not marked "Explored" **then**
      Recursively invoke DFS(v)
    end if
  end for

Example on board

DFS Tree

Can also extract tree T from DFS.

- $(u, v) \in T$ if v explored from u—i.e., DFS(u) calls DFS(v)

Claim: let T be a depth-first search tree for graph $G = (V, E)$, and let (x, y) be an edge that is in G but not T (a "non-tree edge"). Then either x is an ancestor of y or y is an ancestor of x in T.

Proof on board

DFS and Non-tree edges

Claim: let T be a depth-first search tree for graph $G = (V, E)$, and let (x, y) be an edge that is in G but not T (a "non-tree edge"). Then either x is an ancestor of y or y is an ancestor of x in T.
Proof

- Suppose not and suppose that x is reached first by DFS.
- Before leaving x, we must examine (x, y).
- Since $(x, y) \notin T$, y must have been explored by this time.
- But y was not explored when we arrived at x by assumption.
- Thus y was explored during the execution of $\mathtt{DFS}(x)$.
- Implies x is ancestor of y.

Exploring *all* Connected Components

How to explore entire graph even if it is disconnected?

while there is some unexplored node s **do**
  BFS(s) ▷ Run BFS starting from s.
  Extract connected component containing s
end while

Usually OK to assume graph is connected. State if you are doing so and why it does not trivialize the problem.

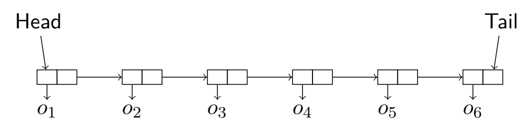
Running time? What's the running time of BFS?

Implementation

- How do we *implement* graph traversal? What is the running time?
- Preliminaries
  - Let $m = |E|$ be the number of edges
  - Let $n = |V|$ be the number of nodes
  - Data structure to represent graph? ...

Interlude (Data Structures)

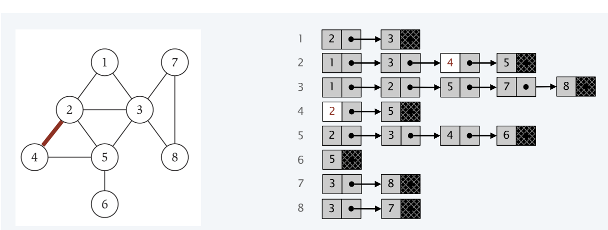
Linked List:



- Always remove items from front (Head)
- Queue: Insert at Tail (FIFO)
- Stack: Insert at Head (LIFO)
- Insert/Removal are $O(1)$ operations.

Graph representation: adjacency lists

Adjacency lists. Each node keeps list of neighbors



- Each edge stored twice
- Space? $\Theta(m + n)$
- Checking if (u, v) is an edge? $O(\text{degree}(u))$ time (degree = number of neighbors)

Traversal Implementations

Generic approach: maintain set of explored nodes and discovered nodes

- Explored = have seen this node and explored its outgoing edges
- Discovered = the "frontier". Have seen the node, but not explored its outgoing edges.

Generic Graph Traversal

Let A = data structure of discovered nodes
Traverse(s)
  Put s in A
  while A is not empty **do**
    Take a node v from A
    if v is not marked "explored" **then**
      Mark v as "explored"
      for each edge (v, w) incident to v **do**
        Put w in A ▷ w is discovered
      end for
    end if
  end while

Note: one part of this algorithm seems wasteful. Why?
Can put multiple copies of a single node in A.

Generic Graph Traversal

Let A = data structure of discovered nodes
Traverse(s)
  Put s in A
  while A is not empty **do**
    Take a node v from A
    if v is not marked "explored" **then**
      Mark v as "explored"
      for each edge (v, w) incident to v **do**
        Put w in A ▷ w is discovered
      end for
    end if
  end while

BFS: A is a queue (FIFO)
DFS: A is a stack (LIFO)

BFS Implementation

Let A = empty Queue structure of discovered nodes
Traverse(s)
  Put s in A
  while A is not empty **do**
    Take a node v from A
    if v is not marked "explored" **then**
      Mark v as "explored"
      for each edge (v, w) incident to v **do**
        Put w in A ▷ w is discovered
      end for
    end if
  end while

Is this actually BFS? Yes. Proof by example.

BFS Running Time

How many times does each line execute?

Traverse(s)
  Put s in A
  while A is not empty **do**
    Take a node v from A
    if v is not marked "explored" **then**
      Mark v as "explored"
      for each edge (v, w) incident to v **do**
        Put w in A
      end for
    end if
  end while

BFS Running Time

How many times does each line execute?

Traverse(s)
  Put s in A 1
  while A is not empty **do** $2m$
    Take a node v from A $2m$
    if v is not marked "explored" **then** $2m$
      Mark v as "explored" n
      for each edge (v, w) incident to v **do** $2m$
        Put w in A $2m$
      end for
    end if
  end while

Running time $O(m + n)$

DFS Implementation

Let A = empty Stack structure of discovered nodes
Traverse(s)
  Put s in A
  while A is not empty **do**
    Take a node v from A
    if v is not marked "explored" **then**
      Mark v as "explored"
      for each edge (v, w) incident to v **do**
        Put w in A ▷ w is discovered
      end for
    end if
  end while

Is this actually DFS? Yes
Running time? $O(m + n)$

Back to Connected Components

while There is some unexplored node s **do**
  BFS(s)
  Extract connected component containing s
end while

Running time?

Naive: $O(m+n)$ for each component $\Rightarrow O(c(m+n))$ if c components.

Better: BFS on component C only works on nodes/edges in C

- Time for component C: $O(\#\text{edges in } C + \#\text{nodes in } C)$
- Total time: $O(m+n)$

Summary

- Graph traversal by BFS/DFS
  - Different versions of general exploration strategy
  - Produce trees with different properties
  - $O(m+n)$ time
  - Basic algorithmic primitive — used in many other algorithms