**Graphs, the Internet, and Everything**

- [Image of internet traffic](image)

  - [Link to CAIDA](http://www.caida.org/)

---

**Airline routes**

- [Map of airline routes](image)

---

**Word ladder**

- [Graph showing word ladder](image)

---

**Graphs: Structures and Algorithms**

- How do packets of bits/information get routed on the internet
  - Message divided into packets on client (your) machine
  - Packets sent out using routing tables toward destination
    - Packets may take different routes to destination
      - What happens if packets lost or arrive out-of-order?
      - Routing tables store local information, not global (why?)

- What about The Oracle of Bacon, Erdos Numbers, and Word Ladders?
  - All can be modeled using graphs
  - What kind of connectivity does each concept model?

- Graphs are everywhere in the world of algorithms (world?)
Vocabulary

- Graphs are collections of vertices and edges (vertex also called node)
  - Edge connects two vertices
    - Direction can be important, directed edge, directed graph
    - Edge may have associated weight/cost
- A vertex sequence $v_0, v_1, \ldots, v_{n-1}$ is a path where $v_k$ and $v_{k+1}$ are connected by an edge.
  - If some vertex is repeated, the path is a cycle
  - A graph is connected if there is a path between any pair of vertices

Graph questions/algorithms

- What vertices are reachable from a given vertex?
  - Two standard traversals: depth-first, breadth-first
  - Find connected components, groups of connected vertices
- Shortest path between any two vertices (weighted graphs?)
  - Breadth first search is storage expensive
  - Dijkstra’s algorithm is efficient, uses a priority queue too!
- Longest path in a graph
  - No known efficient algorithm
- Visit all vertices without repeating? Visit all edges?
  - With minimal cost? Hard!

Depth, Breadth, other traversals

- We want to visit every vertex that can be reached from a specific starting vertex (we might try all starting vertices)
  - Make sure we don’t visit a vertex more than once
    - Why isn’t this an issue in trees?
    - Mark vertex as visited, use set/array/map for this
      - Can keep useful information to help with visited status
  - Order in which vertices visited can be important
  - Storage and runtime efficiency of traversals important
- What other data structures do we have: stack, queue, …
  - What happens when we traverse using priority queue?

Breadth first search

- In an unweighted graph this finds the shortest path between a start vertex and every vertex
  - Visit every node one away from start
  - Visit every node two away from start
    - This is every node one away from a node one away
  - Visit every node three away from start, …
- Put vertex on queue to start (initially just one)
  - Repeat: take vertex off queue, put all adjacent vertices on
  - Don’t put a vertex on that’s already been visited (why?)
  - When are 1-away vertices enqueued? 2-away? 3-away?
  - How many vertices on queue?
- What is this equivalent to on a Binary Tree?
### Pseudo-code for Breadth First

```java
public void breadth(String vertex){
    Set visited = new TreeSet();
    Queue q = new LinkedList();
    q.addLast(vertex);
    visited.add(vertex);
    while (q.size() > 0) {
        String current = (String) q.removeFirst();
        // process current
        for EACH v ADJACENT to current{
            if (!visited.contains(v)){ // not visited
                visited.add(v);
                q.addLast(v);
            }
        }
    }
}
```

### Breadth First Search

- Un-mark all vertices
- Process and mark starting vertex and place in queue
- Repeat until queue is empty:
  1. Remove a vertex from front of queue
  2. For each unmarked adjacent vertex
     - process it
     - mark it
     - place it on the queue

### Pseudo-code for depth-first search

```java
void depthfirst(String vertex){
    if (!alreadySeen(vertex)) {
        markAsSeen(vertex);
        System.out.println(vertex);
        for EACH v ADJACENT to vertex {
            depthfirst(v);
        }
    }
}
```

- Depth First Search (recursive)
  - Un-mark all vertices (pre search initialization!!!)
  - Process and mark starting vertex
  - For each unmarked adjacent vertex do Depth First Search

- Clones are stacked up, problem? Can we make use of stack explicit?
Graph implementations

- Typical operations on graph:
  - Add vertex
  - Add edge (parameters?)
  - AdjacentVerts(vertex)
  - AllVerts(…)
  - String->int (vice versa)

- Different kinds of graphs
  - Lots of vertices, few edges, *sparse* graph
    - Use adjacency list
  - Lots of edges (max # ?) *dense* graph
    - Use adjacency matrix

Graph implementations (continued)

- Adjacency matrix
  - Every possible edge represented, how many?

- Adjacency list uses $O(V+E)$ space
  - What about matrix?
  - Which is better?

- What do we do to get adjacent vertices for given vertex?
  - What is complexity?
  - Compared to adjacency list?

- What about weighted edges?

Graph implementations (continued)

How far from A to B?

<table>
<thead>
<tr>
<th></th>
<th>Asheville</th>
<th>Durham</th>
<th>Greensboro</th>
<th>Manteo</th>
<th>Murphy</th>
<th>Raleigh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asheville</td>
<td>0</td>
<td>231</td>
<td>178</td>
<td>438</td>
<td>105</td>
<td>241</td>
</tr>
<tr>
<td>Durham</td>
<td>231</td>
<td>0</td>
<td>54</td>
<td>203</td>
<td>340</td>
<td>23</td>
</tr>
<tr>
<td>Greensboro</td>
<td>178</td>
<td>54</td>
<td>0</td>
<td>259</td>
<td>283</td>
<td>78</td>
</tr>
<tr>
<td>Manteo</td>
<td>438</td>
<td>203</td>
<td>259</td>
<td>0</td>
<td>543</td>
<td>197</td>
</tr>
<tr>
<td>Murphy</td>
<td>105</td>
<td>340</td>
<td>283</td>
<td>543</td>
<td>0</td>
<td>355</td>
</tr>
<tr>
<td>Raleigh</td>
<td>241</td>
<td>23</td>
<td>78</td>
<td>197</td>
<td>355</td>
<td>0</td>
</tr>
</tbody>
</table>

Graph Algorithms

- Topological Sort
  - Produce a valid ordering of all nodes, given pairwise constraints
  - Solution usually not unique
  - When is solution impossible?

- Topological Sort Example: Getting an AB in CPS
  - Express prerequisite structure (somewhat dated)
  - This example, CPS courses only: 6, 100, 104, 108, 110, 130
  - Ignore electives or outside requirements (can add later)
Topological Sort

- **Topological Sort Algorithm**
  1. Find vertex with no incoming edges
  2. Remove (updating incoming edge counts) and Output
  3. Repeat 1 and 2 while vertices remain

- **Refine Algorithm**
  - Use priority queue?
  - Complexity?

- **What is the minimum number of semesters required?**
  - Develop algorithm

---

Shortest Path (Unweighted)

- Mark all vertices with infinity (*)
- Mark starting vertex with 0
- Place starting vertex in queue
- Repeat until queue is empty:
  1. Remove a vertex from front of queue
  2. For each adjacent vertex marked with *,
     i. process it,
     ii. mark it with source distance + 1
     iii. place it on the queue.

  How do we get actual “Path”?

Shortest path in weighted graph

- We need to modify approach slightly for weighted graph
  - Edges have weights, breadth first by itself doesn’t work
  - What’s shortest path from A to F in graph below?

- Use same idea as breadth first search
  - Don’t add 1 to current distance, add ???
  - Might adjust distances more than once
  - What vertex do we visit next?

- What vertex is next is key
  - Use greedy algorithm: closest
  - Huffman is greedy, …
Greedy Algorithms

- A greedy algorithm makes a locally optimal decision that leads to a globally optimal solution
  - Huffman: choose two nodes with minimal weight, combine
    - Leads to optimal coding, optimal Huffman tree
  - Making change with American coins: choose largest coin possible as many times as possible
    - Change for $0.63, change for $0.32
    - What if we’re out of nickels, change for $0.32?

- Greedy doesn’t always work, but it does sometimes
- Weighted shortest path algorithm is Dijkstra’s algorithm, greedy and uses priority queue

Shortest Path (Weighted): Dijkstra

- Unmark all vertices and give infinite weight
- Set weight of starting vertex at 0 and place in priority queue
- Repeat until priority queue is empty:
  1. Remove a vertex from priority queue
     i. Process and mark (weight now permanent)
  2. For each adjacent unmarked vertex
     i. Set weight at lesser of current weight and (source weight + path weight).
        • May involve reducing previous weight setting
     ii. Place in priority queue (if not there already)

How do we get actual “Path”?
Other Graph Algorithms

- Traveling Salesman
- Spanning Trees
- Paths with negative weights