Graphs, the Internet, and Everything

- Packets of bits/information routed on the internet
  - Message divided into packets client-side
  - Packets routed toward destination over internet
    - 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 not global info?
  - Criteria used in routing?

- Mapquest, Tomtom, Garmin, Googlemap
  - How do you get from here to there?
    - When you’re not anywhere at all

Odes and Poems

- I think that I shall never see, a structure lovely as a tree

- Yo that is whack, yo talk some smack, yo pop some stack

- She said it’s really not my habit to intrude, Furthermore I hope my meaning won’t be lost or misconstrued, but I’m afraid that element hasn’t yet been queued

- To everything there is a season. A time to weep, and a time to laugh; a time to model using a graph.

Graphs: Structures and Algorithms

- 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?)
  - What is a graph? Algorithms on graphs?
  - Graph representation?
Old and New: Fan Chung Graham

- Graphs are everywhere
- http://math.ucsd.edu/~fan/graphs/gallery/

Sir Tim Berners-Lee

I want you to realize that, if you can imagine a computer doing something, you can program a computer to do that.

Unbounded opportunity... limited only by your imagination. And a couple of laws of physics.

- TCP/IP, HTTP
  - How, Why, What, When?

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 $v_i$ and $v_{i+1}$ 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
  - connected components, groups of connected vertices
- Shortest path between two vertices (weighted graphs?)
  - Breadth first search is storage expensive
  - Dijkstra’s algorithm efficient, uses a priority queue!
- 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
  - Order in which vertices visited can be important
  - Storage/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 nodes one away from a node one away
  - Visit every node three away from start, ...

- Put vertex on queue to start (initially just one)
  - Repeat: dequeue vertex, enqueue adjacent vertices
  - Avoid enqueuing already visited/queued nodes
  - When are 1-away vertices enqueued? 2-away? N?
  - How many vertices on queue?

Code for breadth first

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

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);
        }
    }
}
```

- Clones are stacked up, problem? Can we make use of stack explicit?
**BFS compared to DFS**

```java
public Set<Graph.Vertex> bfs(Graph.Vertex start) {
    Set<Graph.Vertex> visited = new TreeSet<Graph.Vertex>();
    Queue<Graph.Vertex> qu = new LinkedList<Graph.Vertex>();
    visited.add(start);
    qu.add(start);
    while (qu.size() > 0) {
        Graph.Vertex v = qu.remove();
        for (Graph.Vertex adj : myGraph.getAdjacent(v)) {
            if (!visited.contains(adj)) {
                visited.add(adj);
                qu.add(adj);
            }
        }
    }
    return visited;
}
```

**BFS becomes DFS**

```java
public Set<Graph.Vertex> dfs(Graph.Vertex start) {
    Set<Graph.Vertex> visited = new TreeSet<Graph.Vertex>();
    Queue<Graph.Vertex> qu = new LinkedList<Graph.Vertex>();
    visited.add(start);
    qu.add(start);
    while (qu.size() > 0) {
        Graph.Vertex v = qu.remove();
        for (Graph.Vertex adj : myGraph.getAdjacent(v)) {
            if (!visited.contains(adj)) {
                visited.add(adj);
                qu.add(adj);
            }
        }
    }
    return visited;
}
```

**DFS arrives**

```java
public Set<Graph.Vertex> dfs(Graph.Vertex start) {
    Set<Graph.Vertex> visited = new TreeSet<Graph.Vertex>();
    Stack<Graph.Vertex> qu = new Stack<Graph.Vertex>();
    visited.add(start);
    qu.push(start);
    while (qu.size() > 0) {
        Graph.Vertex v = qu.pop();
        for (Graph.Vertex adj : myGraph.getAdjacent(v)) {
            if (!visited.contains(adj)) {
                visited.add(adj);
                qu.push(adj);
            }
        }
    }
    return visited;
}
```

**What is the Internet?**

- The Internet was originally designed as an "overlay" network running on top of existing phone and other networks. It is based on a small set of software protocols that direct routers inside the network to forward data from source to destination, while applications run on the Internet to rapidly scale into a critical global service. However, this success now makes it difficult to create and test new ways of protecting it from abuses, or from implementing innovative applications and services.

  [http://www.intel.com/labs/features/idf09041.htm]
How does the Internet work?

- Differences between the Internet and phone networks
  - Dedicated circuits/routes
  - Distributed, end-to-end

- Where is the intelligence?
  - Not in the network, per se, in the design and the ends
  - End-to-end Arguments in System Design

- Success of email, web, etc., relies on not building intelligence into the network
  - What about overlay networks?
  - What about PlanetLab?

Jon Kleinberg

- 2005 MacArthur Fellow, 2008 Infosys Award, 2008 Discover “20 Best Brains under 40”
- Networks course and book
  - Duke course 96 Spring 2010

"...Try to keep an open mind about topics and areas going on...It’s much easier to make progress on a problem when you are enjoying what you are doing. In addition to finding work that is important, find work that has some personal interest for you....I’ve benefited from a lot of mentoring throughout my career. I think it’s important to pass it on to the next generation and work in a mentoring capacity or a teaching capacity with people entering the field...."

Graph implementations

- Typical operations on graph:
  - Add vertex
  - Add edge (parameters?)
  - getAdjacent(vertex)
  - getVertices(…)
  - String->Vertex (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?
Shortest path in weighted graph

- We need to modify approach slightly for weighted graph
  - Edges have weights, breadth first 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, ...

What about connected components?

- What computers are reachable from this one? What people are reachable from me via acquaintance?
  - Start at some vertex, depth-first search (breadth?)
    - Mark nodes visited
  - Repeat from unvisited vertex until all visited

- What is minimal size of a component? Maximal size?
  - What is complexity of algorithm in terms of V and E?

- What algorithms does this lead to in graphs?

Greedy Algorithms

- A greedy algorithm makes a locally optimal decision that leads to a globally optimal solution
  - Huffman: choose minimal weight nodes, 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

Edsger Dijkstra

- Turing Award, 1972
- Algol-60 programming language
- Goto considered harmful
- Shortest path algorithm
- Structured programming
  “Program testing can show the presence of bugs, but never their absence”

For me, the first challenge for computing science is to discover how to maintain order in a finite, but very large, discrete universe that is intricately intertwined. And a second, but not less important challenge is how to mould what you have achieved in solving the first problem, into a teachable discipline: it does not suffice to hone your own intellect (that will join you in your grave), you must teach others how to hone theirs. The more you concentrate on these two challenges, the clearer you will see that they are only two sides of the same coin: teaching yourself is discovering what is teachable.
**Dijkstra's Shortest Path Algorithm**

- Similar to breadth first search, but uses a priority queue instead of a queue. Code below is for breadth first search (distance[] replaces set)

```java
Vertex cur = q.remove();
for(Vertex v : adjacent(cur)){
    if (!visited.contains(v)){ // if distance[v] == INFINITY
        visited.add(v);        // distance[v] = distance[cur]+1
        q.add(v);
    }
}
```

- Dijkstra: Find minimal unvisited node, recalculate costs through node

```java
Vertex cur = pq.remove();
for(Vertex v : adjacent(cur))
    if (distance[cur] + graph.weight(cur,v) < distance[v]) {
        distance[v] = distance[cur] + graph.weight(cur,v);
        pq.add(v);
    }
```

**Shortest paths, more details**

- Single-source shortest path
  - Start at some vertex S
  - Find shortest path to every reachable vertex from S
- A set of vertices is processed
  - Initially just S is processed
  - Each pass processes a vertex
  - After each pass, shortest path from S to any vertex using just vertices from processed set (except for last vertex) is always known
- Next processed vertex is closest to S still needing processing

---

**Dijkstra’s algorithm works (greedily)**

- Choosing minimal unseen vertex to process leads to shortest paths

```java
Vertex cur = pq.remove();
for(Vertex v : adjacent(cur))
    if (distance[cur] + graph.weight(cur,v) < distance[v]) {
        distance[v] = distance[cur] + graph.weight(cur,v);
        pq.add(v);
    }
```

- We always know shortest path through processed vertices
  - When we choose \( v \), there can’t be a shorter path to \( v \) than distance[\( v \)] – it would go through processed \( u \), we would have chosen \( u \) instead of \( v \)

---

**Shafi Goldwasser**

- RCS professor of computer science at MIT
  - Twice Godel Prize winner
  - Grace Murray Hopper Award
  - National Academy
  - Co-inventor of zero-knowledge proof protocols

*How do you convince someone that you know [a secret] without revealing the knowledge?*

- **Honesty and Privacy**

*Work on what you like, what feels right, I now of no other way to end up doing creative work*