Announcements

- Final Exam, December 19, 2-5pm
  - Covers topics up through today
- Open notes/Open book – what you can bring
  - Lecture notes and code handed out in lecture
  - Your notes from recitation
  - The book for the course – Sedgewick/Wayne
  - Up to 4 pages of additional notes

- Today – graphs and balanced trees

Interlude for trees

- Joyce Kilmer
- Balanced Trees
  - Splay
  - Red-Black
  - AVL
  - B-tree

Balanced trees we won't study

- B-trees are used when data is both in memory and on disk
  - File systems, really large data sets
  - Rebalancing guarantees good performance both asymptotically and in practice. Differences between cache, memory, disk are important

- Splay trees rebalance during insertion and during search, nodes accessed often more closer to root
  - Other nodes can move further from root, consequences?
    - Performance for some nodes gets better, for others ...
  - No guarantee running time for a single operation, but guaranteed good performance for a sequence of operations, this is good amortized cost (ArrayList.add)
Balanced trees, we’ll look at red-black

- Both kinds have worst-case $O(\log n)$ time for tree operations
- AVL (Adel’son-Velskii and Landis), 1962
  - Nodes are “height-balanced”, subtree heights differ by 1
  - Rebalancing requires per-node bookkeeping of height
- Red-black tree uses same rotations, but can rebalance in one pass, contrast to AVL tree
  - In AVL case, insert, calculate balance factors, rebalance
  - In Red-black tree can rebalance on the way down, code is more complex, but doable
  - Standard `java.util.TreeMap/TreeSet` use red-black

Red-Black Tree

- Invented by Bayr – (though called them something else)
- Robert Tarjan (Turing Award Winner) – noticed the rotations were $O(1)$
- Type of balanced tree – uses color scheme, recoloring and rotations to balance

Red-Black Tree

- Is a Binary Search Tree
- Properties:
  - Every node is red or black
  - The root is black
  - If a node is red, then its children are black
  - Every leaf is a null node and black (external node)
  - Every simple path from a node to a descendant leaf contains the same number of black nodes.

Example red-black tree

- In the figure, black nodes are shaded and red nodes are non-shaded
Example

- The five properties ensure that no path is more than twice as long as any other path.
- Def. The height \( h \) of a node is the length of the longest path from the node (downward) to a leaf (including external nodes).
- Def. The black height \( bh \) of a node \( x \) is the number of black nodes on any path from \( x \) (not including \( x \)) to a leaf.
- Examples: \( h(19) \) ? \( bh(19) \) ? \( h(8) \) ? \( bh(8) \) ?

Height of Red-Black Tree

- Lemma: A red-black tree with \( n \) internal nodes has height at most \( 2 \log(n+1) \).
- Operations:
  - Time for search for \( x \):
  - Time for min:
  - Time for list inorder:

Rotations

- We want to perform insertions and deletions in \( O(\log n) \) time. Adding or deleting a node may disrupt one of its properties, so in addition to some recolorings, we may also have to restructure the tree by performing a rotation (change some pointers).

```
left_rotate x
```

Right Rotate

- Note the rotations change the pointer structure while preserving the inorder property.

```
right_rotate x
```

- Note the inorder traversal in both is: abcde
Example of rotation

Insertion

• Insert node as RED using a binary search tree insert
  – Means insert as a Red leaf with two black NULL nodes
• Then fix-up so that properties still hold
  – Recoloring and/or 1-2 rotations
• Several cases to consider

Insertion – Case 1 – How to Fix

• Case 1 – sibling of parent of x (called y) is red
• To fix: recolor three nodes, then fix up new “x”
**Insertion – Case 2 How to Fix**
- Sibling of parent of x (call y) is black, x right child

![Diagram of case 2](image)

- To fix: set x to parent of x and left rotate x, then it becomes a case 3

**Insertion – Case 3 – How to Fix**
- Case 3 – sibling of parent of x (call y) is black, x left child

![Diagram of case 3](image)

- To fix: two recolorings and one right rotate of grandparent of x

**Example of Insert 4 w/ double rotation**

![Example of Insert 4](image)

**Analysis – Red Black Tree**
- Insert – $O(\log n)$
- Deletion – $O(\log n)$
Graphs, the Internet, and Everything

Odes and Poems

- I think that I shall never see, a structure lovely as a tree
- Yo that's whack, yo let's hack, yo pop a 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

- 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?
  - What's a route? How is this known?

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?
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 \( v_k \) 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
  - connected components, groups of connected vertices

- Shortest path between two vertices (weighted graphs?)
  - BFS works, possibly uses more storage than DFS
  - 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 if 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?

Pseudo-code for depth-first search

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?

Code for breadth first

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);
        }
    }
}

BFS compared to DFS

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

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...."

ACM Infosys Interview

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

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, ...

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?
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 Reviewed

- 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
        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 w, there can’t be a shorter path to w than distance[w] – it would go through processed u, we would have chosen u instead of w

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 know of no other way to end up doing creative work_