CSEdWeek

- **Huffman questions and overview**
  - What should you expect when compressing?
  - How will you debug when things don't work?
  - What can you take away from assignment?

- **Toward understanding graphs**
  - Data Structures and Algorithms
  - One more Priority Queue application, basis for so many important algorithms

- **Google v Oracle: Another Personal Odyssey**
### Compressing kjv10.txt

<table>
<thead>
<tr>
<th>Encoding Length</th>
<th># values with this length</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1,159,124</td>
</tr>
<tr>
<td>4</td>
<td>1,487,471</td>
</tr>
<tr>
<td>5</td>
<td>712,325</td>
</tr>
<tr>
<td>6</td>
<td>485,333</td>
</tr>
<tr>
<td>7</td>
<td>261,611</td>
</tr>
<tr>
<td>8</td>
<td>84,107</td>
</tr>
<tr>
<td>9</td>
<td>81,467</td>
</tr>
<tr>
<td>10</td>
<td>48,019</td>
</tr>
<tr>
<td>11</td>
<td>21,065</td>
</tr>
<tr>
<td>12</td>
<td>1,863</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Encoding Length</th>
<th># values with this length</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>1,108</td>
</tr>
<tr>
<td>14</td>
<td>664</td>
</tr>
<tr>
<td>15</td>
<td>476</td>
</tr>
<tr>
<td>16</td>
<td>225</td>
</tr>
<tr>
<td>17</td>
<td>71</td>
</tr>
<tr>
<td>18</td>
<td>44</td>
</tr>
<tr>
<td>19</td>
<td>22</td>
</tr>
<tr>
<td>20</td>
<td>11</td>
</tr>
<tr>
<td>21</td>
<td>3</td>
</tr>
<tr>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>23</td>
<td>6</td>
</tr>
</tbody>
</table>
Reading pre-order traversal

- Uses same technique as in Tree APTs
  - Interior and Leaf nodes distinguished in Huff
- \[ 8 \ 4 \ x \ 6 \ x \ x \ 12 \ 10 \ x \ x \ 15 \ x \ x \]
  - If you read a number? Read two subtrees
  - If you read an x? return null, no recursion
- \[ 8 \ [4 \ x \ 6 \ x \ x] \ [12 \ 10 \ x \ x \ 15 \ x \ x] \]
  - 12 \ [10 \ x \ x] \ [15 \ x \ x]
How do you interpret bits?

● A tale of two files
  ➢ Foo.class and Bar.jpg
  ➢ What do you know about these files?

● How are these files "interpreted"?
  ➢ By humans, by programs/code

| 00000000: | cafe babe 0000 0034 001d 0a00 0600 0f09 | .......4........ |
| 0000010: | 0010 0011 0800 120a 0013 0014 0700 1507 | ............... |
| 0000020: | 0016 0100 063c 696e 6974 3e01 0003 2829 | .....<init>...() |

| 00000000: | ffd8 ffe0 0010 4a46 4946 0001 0200 0064 | .......JFIF.....d |
| 0000010: | 0064 0000 ffec 0011 4475 636b 7900 0100 | .d.......Ducky... |
| 0000020: | 0400 0000 5d00 00ff ee00 0e41 646f 6265 | ....]......Adobe |
Interpreting Bits

● Does the base matter? Is there a base with 0/1?
  ➢ Is there a good answer to this question?

● Does the JVM read bit-by-bit? By byte? By symbol?
  ➢ Why isn't this stored as text as in Foo.java?

```
0000000: cafe babe 0000 0034 001d 0a00 0600 0f09 ..........4........
0000010: 0010 0011 0800 120a 0013 0014 0700 1507 ..............
0000020: 0016 0100 063c 696e 6974 3e01 0003 2829 ......<init>...

0000000: 11001010 11111110 10111010 10111110 00000000 00000000 ......
0000006: 00000000 00110100 00000000 00011101 00001010 00000000 .4....)
```
Using a program to Interpret

"Hello World" – as seen by running with a Java Interpreter

Guernica jpg -- as seen from Wikimedia in Chrome
Mary Shaw

- **Software engineering and software architecture**
  - Tools for constructing large software systems
  - Development is a small piece of total cost, maintenance is larger, depends on well-designed and developed techniques

- Interested in computer science, programming, curricula, and canoeing, health-care costs

- ACM Fellow, Alan Perlis
  Professor of Compsci at CMU
Graphs, the Internet, and Everything

http://www.caida.org/
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?

● Googlemap, mapquest, Garmin, Applemap, ...
  ▶ 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?
Old and New: Fan Chung Graham

- Graphs are everywhere
- [http://math.ucsd.edu/~fan/graphs/gallery/](http://math.ucsd.edu/~fan/graphs/gallery/)

A subgraph of a BGP graph

A subgraph of the Hollywood graph.
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 $v_0, v_1, ..., v_{n-1}$ 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
We won't look in detail, but …

- Typically graph algorithm complexity is
  - $O(V+E)$ or $O(V^2)$ or $O(E \log V)$ or …
  - $V$ is number of vertices, $E$ is number of edges
  - Depends on how graph represented (to follow)

- Many problems are solved first by modeling the data/input as a graph and then
  - Finding properties of the graph
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! Traveling Salesperson
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, e.g., WordLadder

- 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 enqueueing 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<>();
    Queue<String> q = new LinkedList<>();
    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<String> bfs(String start) {
    Set<String> visited = new TreeSet<>();
    Queue<String> qu = new LinkedList<>();
    visited.add(start);
    qu.add(start);

    while (qu.size() > 0) {
        String v = qu.remove();
        // do something/process vertex v
        for (String adj : myGraph.getAdjacent(v)) {
            if (!visited.contains(adj)) {
                visited.add(adj);
                qu.add(adj);
            }
        }
    }
    return visited;
}
```
BFS becomes DFS

```java
public Set<String> dfs(String start) {
    Set<String> visited = new TreeSet<>();
    Queue<String> qu = new LinkedList<>();
    visited.add(start);
    qu.add(start);

    while (qu.size() > 0) {
        String v = qu.remove();
        // do something/process vertex v
        for (String adj : myGraph.getAdjacent(v)) {
            if (! visited.contains(adj)) {
                visited.add(adj);
                qu.add(adj);
            }
        }
    }
    return visited;
}
```
DFS arrives

public Set<String> dfs(String start){
    Set<String> visited = new TreeSet<>();
    Stack<String> qu = new Stack<>();
    visited.add(start);
    qu.push(start);

    while (qu.size() > 0){
        String v = qu.pop();
        // do something/process vertex v
        for(String adj : myGraph.getAdjacent(v)){
            if (! visited.contains(adj)) {
                visited.add(adj);

                qu.push(adj);
            }
        }
    }

    return visited;
}
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
Solving the Internet APT

- **Graph structure:** for each vertex: adjacent vertices
  - Map: vertex to Adjacent vertices (adjacency list)
- **Graph algorithm:** what is an articulation point?
  - Remove it, can't get to everywhere from ...
  - DFS/BFS from vertex V avoiding vertex W

- Create the graph, copy/paste DFS/BFS, start from V, avoid W, looping over all vertices as possible articulation points

What is the queen vertex?
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
  - Theory + Practice = Knowledge
- "...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 (continued)

- **Adjacency matrix**
  - Every possible edge represented, $O(V^2)$

- **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?**
  - Numbers instead of T/F bits
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 acquaintanceship?
  - 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  [EWD 709]
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 \( w \), 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 \( w \)