Announcements

- Homework 4 due next Monday
- I have old hw’s and midterms in my office
Review

- Finished Huffman optimality proof sketch
- Hash Tables ADT
  - O(1) insert and search
  - Use hash function to determine table index
- Collision resolution strategies:
  - Separate Chaining, Probing
Double Hashing Example

N = 7

h1(x) = x mod 7,  h2(x) = 5-x mod 5

insert 9

insert 16

insert 2
Today's Plan

- Loose ends re: hashing
- Rehashing
- String hash function example
- Graphs
- Definitions, implementation
Rehashing

- Like ArrayLists, we have to guess the number of elements we need to insert into a hash table.
- Whatever our collision policy is, the hash table becomes inefficient when load factor is too high.
- To alleviate load, rehash:
  - create larger table, scan current table, insert items into new table using new hash function.
When to Rehash

* For quadratic probing, insert may fail if load > 1/2
  * We can rehash as soon as load > 1/2
  * Or, we can rehash only when insert fails
* Heuristically choose a load factor threshold, rehash when threshold breached
**Rehash Example**

- **Current Table:**

<table>
<thead>
<tr>
<th>0</th>
<th>8</th>
<th>7</th>
<th>17</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

- **quad. probing with** $h(x) = (x \mod 7)$
  - 8, 0, 25, 17, 7

- **New table**

<table>
<thead>
<tr>
<th>0</th>
<th>17</th>
<th>7</th>
<th>8</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

- **h(x) = (x \mod 17)**
Rehash Cost

- No profound algorithm: re-insert each item
- Linear time
- If you rehash, inserting $N$ items costs $O(1) \times N + O(N) = O(N)$
- Insert still costs $O(1)$ amortized
Hashing a String

- Simple but bad $h(x)$
  - add up all the character codes (ASCII/Unicode)
  - ASCII 'a' is 97
- If keys are lowercase 5 character words, $h(x) > 485$
Hashing a String II

- Weiss: Treat first 3 characters of a string as a 3 digit, base 27 number
- Once again, ‘a’ is 97, ‘A’ is 65
String.hashCode()

- Java's built in String hashCode() method
  - $s[0] \times 31^{(n-1)} + s[1] \times 31^{(n-2)} + \ldots + s[n-1]$
- nth degree polynomial of base 31
- String characters are coefficients
Graphs

Trees

Linked Lists
Graphs
Graph Terminology

- A **graph** is a set of **nodes** and **edges**
  - nodes aka vertices
  - edges aka arcs, links
- Edges exist between pairs of nodes
  - if nodes x and y share an edge, they are **adjacent**
Graph Terminology

- Edges may have **weights** associated with them
- Edges may be **directed** or **undirected**
- A **path** is a series of adjacent vertices
  - the **length** of a path is the sum of the edge weights along the path (1 if unweighted)
- A **cycle** is a path that starts and ends on a node
Graph Properties

- An undirected graph with no cycles is a tree
- A directed graph with no cycles is a special class called a **directed acyclic graph (DAG)**
- In a **connected** graph, a path exists between every pair of vertices
- A **complete** graph has an edge between every pair of vertices
Graph Applications: A few examples

- Computer networks
- The World Wide Web
- Social networks
- Public transportation
- Probabilistic Inference
- Flow Charts
Implementation

- **Option 1:**
  - Store all nodes in an indexed list
  - Represent edges with *adjacency matrix*

- **Option 2:**
  - Explicitly store *adjacency lists*
Adjacency Matrices

- 2d-array $A$ of boolean variables
- $A[i][j]$ is true when node $i$ is adjacent to node $j$
- If graph is undirected, $A$ is symmetric

\[
\begin{array}{c|ccccc}
& 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 1 & 1 & 0 & 0 \\
2 & 1 & 0 & 0 & 1 & 0 \\
3 & 1 & 0 & 0 & 1 & 0 \\
4 & 0 & 1 & 1 & 0 & 1 \\
5 & 0 & 0 & 0 & 1 & 0 \\
\end{array}
\]
Adjacency Lists

* Each node stores references to its neighbors

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

![Diagram of a graph with nodes 1 to 5 and connections between them]
Reading

- Homework 4
- Weiss Section 5 (Hashing)
- Weiss Section 9.1-9.2