Administrative Details

• Super Lexicon lab today
  • May work with a partner
  • But must work *with* your partner
    • Attend same lab section
    • “Pair program” in the lab (or elsewhere)

• Posted hints to get you started

• Tools to help you test
  • Main.java
  • small.txt, small2.txt, ospd2.txt
Last Time

- Huffman Codes (AN ANTARCTIC PENGUIN)

- Briefly talked about how to represent a tree using an array (or vector/list)
Today’s Outline

• Finish binary-trees-as-arrays discussion
• Discuss priority queues
• (maybe) Introduce heaps
Using Arrays to Store Trees

• Implicitly encode tree structure using indexes:
  • Consider a **full** tree
  • Index nodes as in level-order traversal

• Instead of pointers, use math to walk the tree
  • Children of node $i$ are at $2i+1$ and $2i+2$
  • Parent of node $j$ is at $(j-1)/2$
Example

Orange

Green
- Blue
- Indigo

Violet
- Red
- Yellow

0 1 2 3 4 5 6
Same Contents, Different Tree

- Green
  - Blue
  - Orange
    - Indigo
  - Yellow
    - Red
Cost of Imbalance

• Possible nodes in level $i$ of a binary tree?
  • $2^i$

• For a tree with $n$ elements...

<table>
<thead>
<tr>
<th>Height</th>
<th>Total Array Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Tree:</td>
<td>$\log_2(n)$</td>
</tr>
<tr>
<td>“Degenerate” Tree:</td>
<td>$n$</td>
</tr>
</tbody>
</table>
ArrayTree Tradeoffs

• Why are ArrayTrees good?
  • Save space for links (no “slots” needed)
    • Relationships between values are implicitly stored (index + math)
  • Works well for complete trees
    • “A complete binary tree of height h is a full binary tree with 0 or more of the rightmost leaves of level h removed”

• Why bad?
  • Could waste a lot of space (sparse trees)
  • Height of n requires $2^{n+1}-1$ array slots even if only $O(n)$ elements
Open Question: What Does it Mean to be “Fair”?  

- How are people “served” in:  
  - Cafeterias  
    - A Queue  
  - Airplanes  
  - Emergency room
Priority Queues

- Name is misleading
- PQs are a bit like normal queues, except they are not FIFO
- Always dequeue object with highest priority regardless of when it was enqueued
- Data can be received/inserted in any order, but it is always returned/removed in same order (according to priority)
Priority Queues vs. Ordered Structures

• Like ordered structures (i.e., OrderedVectors and OrderedLists), PQs appear to keep data in order
  • What did we gain from ordered structures?
    • Search cost
  • What is the cost of maintaining order?
    • Insert cost

• Unlike ordered structures, PQs allow the user only to remove its “smallest/best” element
  • Can’t search, no random access
Priority Queues vs. Linear Structures

• PQs are also similar to Linear structures (i.e., stacks and queues):
  • values are added to the structure one at a time
  • may be inspected or removed one at a time

• Unlike Linear structures, not LIFO or FIFO
  • Always removed the minimum value (i.e., value with highest priority)
Priority Queue Uses

- Priority queues are used for:
  - Scheduling processes in an operating system
    - Priority is function of time lost + process priority
  - Order services on server
    - Low priority tasks shouldn’t interfere with high priority tasks
      - Backup, virus scanning, certain updates
  - Medical waiting room
  - Huffman codes - order by tree size/weight
  - To generally rank choices that are generated out of order
public interface PriorityQueue<E extends Comparable<E>> {
    public E getFirst();  // Non-destructive
    public E remove();
    public void add(E value);  // Do not specify location, priority
    public boolean isEmpty();
    public int size();
    public void clear();
}
Things to Note about PQ Interface

• Unlike previous structures, we do not extend any other interfaces
• PriorityQueue methods *consume* Comparable parameters and *return* Comparable values
• Possibilities besides using Comparables?
  • Comparators
Implementing PQs

• Queue?
  • Wouldn’t work so well because we can’t insert and remove in the “right” way (i.e., keeping things ordered)

• OrderedVector?
  • Keep ordered vector of objects
  • O(n) to add/remove from vector
  • Details in book…
  • Can we do better than O(n)?

• Heap?
  • Partially ordered binary tree
A heap is a **complete** binary tree where:
- Root holds smallest (highest priority) value
- Left and right subtrees are also heaps (this is important!)
- Any path from root to leaf is in descending order
- Invariant for nodes
  - node.value() <= node.left.value()
  - node.value() <= node.right.value()
- Several valid heaps for same data set (no unique representation)
A trie is a letter-tree that efficiently stores strings. A node in a trie represents a letter. A path in the trie traces out a sequence of letters that represent a prefix or word in the lexicon.

Managing the trie!

• Implementing a trie is an excellent exercise in algorithm design, as you must consider:
  - ease of writing and debugging the code, performance of add/remove, and so on.
  - space-efficiency
  - tradeoffs between choosing different node structures, such as a static-sized array which can grow and shrink as needed, a linked list of children pointers, or a largely NULL 26-member array.

For the trie:

- Each trie node represents a letter.
- A path from the root to a node represent a letter. The root is the node for a given letter, (such as from Z to X) the array entry would be NULL.

Array[0] is the child for A, Array[1] refers to B, ... and Array[25] refers to Z. When storing above:

- For a given letter, you simply access the child for that letter.
- The search for a given letter, 

Instead of just two children as in a binary tree, each trie node has potentially many children, one for each letter. The trie is a shared structure.

The root to this node represents a word. Here's a conceptual picture of a small trie:

Each trie has a pointer to the root (Start). From the root, you can navigate through the trie by following pointers to the child for any letter. A flag at each node indicates whether the path leads to a word.

Leaf node: isWord must be true

What are the words represented in this trie?

Nodes:
- letter
- isWord

Node A is the child for A, Node R refers to R, Node S refers to S, and Node E refers to E.

Node A has children R, S, E, W, O, T, and N.
Representing Tries

• Not a binary tree... how to store children?
  • Options: an array of characters, a Vector, an OrderedStructure
    • Maximum number of children for any node?
  • If you have to scan 26 elements to find a child, how does this affect the Big-O cost of walking from root to leaf?
  • Why might it still be important to keep the children sorted?
Regular Expressions (Sort of…)

- The ‘*’ wildcard character matches any sequence of zero or more characters.
- The ‘?’ wildcard character matches either zero or one character.
Regular Expressions (Sort of…)

What word(s) match *T*?  What word(s) match *E*?  What word(s) match ?S?
Sets

- Store unique elements (ignore duplicates)
- Useful for checking membership *quickly*
- Giving the data structures we have covered, what would be an appropriate choice?
  - In reality, probably use *hashing*