CS61B Summer 2006
Instructor: Erin Korber
Lecture 21: 1 Aug.

1 Tries

Tries are a more efficient way to store a dicionary as a tree.

- Have been considering all comparisons to be constant time.
- But conisder comparing strings - can take as long as the length of the string!
- Waste a lot of time looking at the same characters of some string over and over in order to compare it to a bunch of other strings.
- Idea: Make a multi-way search tree, with one comparison/decision for each *character* of the key.

Example: (see board)

- `find, insert, delete, closestBefore, closestAfter` all take time proportional to the length of the key being searched for now.

2 Huffman Encoding

2.1 Compression

- Want to represent a bunch of data in a way that takes up less space
- Idea: use codes of a particular fixed size, and choose one for each byte/character/whatever that we have.
- Note that for characters, 8 bits per character is the best we can do if we want to be able to represent all 256 ASCII characters.

2.2 Variable-length codes

- Note that in, e.g., usual written English, some characters appear a lot more often than others.
- Idea: use shorter codes to represent characters that we use a lot.

- But if we do this, how do we tell where one code begins and another ends?
- Answer: make it a *prefix-free* code (sometimes called a prefix code, annoyingly). No code is a prefix of another code.

2.3 Huffman's algorithm

Huffman coding is an optimal way to produce an encoding. It works by building an encoding tree from the bottom up based on the frequencies of the data.

The algorithm:

- Build a table containing the frequencies of each character in the input data.
- Make a node for each character with a "score" of its frequency
- Put all those nodes into a priority queue.
- While the priority queue has more than 1 item in it:
  - Take 2 nodes out of the priority queue, and create a new node whose "frequency" is their sum. Make it the parent of those nodes and put it in the priority queue.
- Now all the nodes are under 1 root node.
- Codes can be read by traversing the tree - use it to make a compressed file.

When we write out our compressed file, we write a "header" at the top that contains the frequency table that we used to build the tree. This way, when we decompress, we can use this table to reconstruct an identical tree to recover the codes used for compression. So to decompress we read in the header table, use it to build the encoding tree, and read the codes that we have in our compressed file using the tree to recover the original data.