Chapter 21
Priority Queues

Introduction

- Want to delete from queue according to priority.
  - Max priority queue – delete the greatest.
  - Min priority queue – delete the least.
- Insert normally, but delete based on priority.
- We can implement priority queues using binary search trees, ordered or unordered lists, ordered or unordered arrays, etc.
- Let assume a linked list implementation.
  - Unordered
    - Insert – $O(1)$.
    - Delete – $O(n)$.
  - Ordered
    - Insert – $O(n)$.
    - Delete – $O(1)$.
- We may get some gains if we just partially sort the data. This is true especially if we only delete a small fraction of the information. Then we don’t incur the overhead of sorting the entire set of data.
- This partially ordered data is how a heap helps us.

Heaps and Heapsort

- Heap – complete binary tree in which each node is smaller than its parent.
  - Is this the same thing as a binary search tree (BST)?
- The binary tree for the heap is implemented as an array. This allows us easy access to children and parents as seen in the previous chapter. Notice that the array starts at 1, not 0. You just ignore the $0^{th}$ element of the array.
  - Left child – $2i$
  - Right child – $2i + 1$
  - Parent – $\lfloor i/2 \rfloor$
- Used as a priority queue – regular insertion, priority deletion.
- Insertion – put the new node at the first empty position, and sift the element up (if needed). See Figure 1.
  - $O(height) = O(log n)$
- **Deletion** – select root. Swap with last position (which will no longer be part of queue), and sift-down. See Figure 2.
  - $O(height) = O(\log n)$

- **Initialization** – $O(n)$
- Can use to sort:
  - Analysis: $O(n \log n)$ but about twice as slow as quicksort.
  - $O(n)$ to initialize, $O(\log n)$ to delete, and need to delete $n$ items. Thus $O(n \log n)$.
  - However, exhibits same complexity in worst case.
- This is an implicit data structure – no special structure need for heap except the binary tree. Thus no space overhead.

**Huffman Coding**

- Goal – Assign characters (or set of words of possible messages) a string of bits such that the average length is minimized. Variable length coding.


- **Goals:**
  - The resulting message should be decoded uniquely. For example:
    - a 101
    - b 10
    - c 1
    - The coding 1011 could be ac or bcc
  - The resulting message should be decoded instantaneously - no need for look ahead. For example:
    - a 101
    - b 10
    - c 11
    - The coding 1011 is bc, but you can't tell for sure until the end.
  - The average length of encoded messages should be minimized.

**Algorithm**

1. Given a set of messages with frequencies, place all in a priority queue.
2. Remove the messages with the two lowest frequencies, \( f_1 \) and \( f_2 \). Add to the set of messages a binary tree with frequency \( f_1 + f_2 \), having the two messages as children.
   - If the new subtrees are added to the list after the existing nodes of the same frequency, we produce a better, balanced tree.
3. Continue the previous step until the set of messages contains a single binary tree. This tree represents the encoding. From the root, going left represents a 0, going right a 1.

- Note: Huffman coding meets goals.
- For example: a 1 b 5 c 2 d 8 e 3 f 7 g 1 h 5
- See Figure 3 for the construction of the tree.
Figure 3 Building a Huffman Tree