OH TODAY: 10pm - 11pm ET

Emails

Exam 2: 3/2

no Heaps on the exam

Heap
- Array object that can be viewed as an almost complete binary tree.

- Max-heap property
  - ∀ i, A[parent(i)] ≥ A[i]

- Height of a heap containing n elements is O(\log n)

- MaxHeapify(A, i) takes time proportional to \log n

  // fixes the MaxHeap property at node i

  // prerequisites: the left and right trees
// if i are max heaps.

- BuildHeap(A)
  // build a heap

Running time: $O(n \log n)$

$\Theta(n)$

Lemma: $\sum_{h=0}^{\left\lfloor \log \frac{n}{2} \right\rfloor} \left\lfloor \frac{n}{2^{h+1}} \right\rfloor \leq n \cdot h$

Cost: $O\left(\sum_{h=0}^{\left\lfloor \log \frac{n}{2} \right\rfloor} \left\lfloor \frac{n}{2^{h+1}} \right\rfloor \cdot h\right)$

= $O\left(n \sum_{h=0}^{\left\lfloor \log \frac{n}{2} \right\rfloor} \frac{h}{n} \right)$

= $\sum_{i=0}^{\left\lfloor \log \frac{n}{2} \right\rfloor} c \cdot i^2$, $c<1$

= $O(n)$.

Heap Sort (A)
Build Heap (A) (Max-heap)

for j ← |A| downto 2 do

    Swap A[i] with A[j]

    heapsize ← heapsize - 1

MaxHeapify (A, i)
Running time: $\Theta(n \log n)$

Priority Queues: (Max priority queue)

data that has keys (priorities)

Maximum (A)

Extract Max (A)

Increase Key (A, i, k)

Insert (A, k)

Maximum (A)

return A[1] $O(1)$

Extract Max (A)
\text{max} \leftarrow A[1]

\text{Swap} \ A[1] \text{ with } A[\text{heapsize}] \quad O(1)

\text{heapsize} --

\text{MaxHeapify} (A, i) \quad - \quad O(\log n)

\text{return} \quad \text{max}

\text{Increa } \text{ny} (A, i, k)

A[i] \leftarrow k

\text{while} \ (i > 1) \text{ and } (A[\text{parent}(i)] < A(i))

\text{swap} \ A[i] \text{ and } A[\text{parent}(i)]

i \leftarrow \text{parent}(i)

O(\log n)
Insert \((A, k)\)

\[
\text{heapsize} + 1
\]

\[
A[\text{heapsize}] \leftarrow -\infty
\]

Increment key \((A, \text{heapsize}, k)\).

**Huffman Coding.**

**Motivation:** Text \(\rightarrow\) binary encoding.

26 letters
6 other chars.

\# bits per letter: \(5\) \((2^5 = 32)\)

Clearly, 4 is not good. \((2^4 < 32)\).
**Fixed Length Encoders**

**Variable Length encoders**

higher freq letters get shorter encods lower " " long " ".

Hope is that avg. # bits per letter ↓

**Morse Code**

<table>
<thead>
<tr>
<th>e</th>
<th>t</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>01</td>
<td></td>
</tr>
</tbody>
</table>

```
0 1 01
0 1 01
```

**Prefix Code (Prefix free code)**
A code $\gamma: S \to \text{binary strings}$ is a prefix code if $\forall x, y \in S \ s.t. x \neq y,$ $\gamma(x)$ is not a prefix of $\gamma(y)$.

Example of a prefix code:

$\gamma(a) = 00$
$\gamma(b) = 01$
$\gamma(c) = 10$
$\gamma(d) = 11$

<table>
<thead>
<tr>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

Problem

Input: text containing $n$ characters
- alphabet $S$
for each $x \in S$, $f_x$ is the freq. of letter $x$ in the text. That is, $f_x$ : freq. of times $x$ appears in the text. i.e., $n.f_x$ is the # times $x$ appear in the text.

Obj: To obtain prefix codes for $x \in S$ s.t. total length of the encoding $= \sum_{x \in S} n.f_x \cdot |v(x)|$

is minimized.

Avg. # bits per letter (ABL) $= \frac{\sum_{x \in S} n.f_x \cdot |v(x)|}{n}$

= $\left[ \sum_{x} f_x \cdot |v(x)| \right]_{\text{minimized}}$
Our example:

\[
\begin{align*}
\gamma(a) &= 00, \quad f_a = 0.32 \\
\gamma(b) &= 01, \quad f_b = 0.25 \\
\gamma(c) &= 10, \quad f_c = 0.20 \\
\gamma(d) &= 11, \quad f_d = 0.18 \\
\gamma(e) &= 100, \quad f_e = 0.05 \\
\end{align*}
\]

Optimal?

Objective value

\[
(0.32 \times 2) + (0.25 \times 2) + (0.20 \times 3) + (0.18 \times 2) + (0.05 \times 3) = 2.25
\]

Binary trees give us prefix codes?

Suppose we have a binary tree in which the leaves are labeled with letters S.

- What is the encoding of a letter?

- Are these codes prefix-free?
For $r(x)$ to be a prefix of $r(y)$, $x$ must be on the path from root to $y$, which is not possible as $n$ is a leaf.

$$\text{Obj fn} \quad \min \sum_{x \in S} f_x \cdot |\text{depth}(x)|$$

Lemma: The binary tree corr. to an opt. prefix code is **full**.

**Divide & Conquer.**
- Shape of the full binary tree.

- Once we have the shape, how to assign letters in $S$ to leaves in the tree.
Someone gives us the shape: \[ a, b, c, d, e, f. \]

Greedily assign letters to leaves: assign higher freq. letters to leaves higher up in the tree.

Q: When will the two lowest freq. leaves be sitting?

Lowest freq.: at a leaf in the bottommost level

Second lowest freq.: at a leaf in the bottommost level.
Learning: There exists an opt prefix code with a corr tree \( T^* \) s.t. the two lowest freq letters are assigned to the sibling leaves in the bottommost level.

\[\text{A1. Huffman}(S)\]

\[
\begin{align*}
\text{if } |S| \leq 2 & \text{ then} \\
\quad \text{Done} \\
\text{else} \\
\quad y^*, z^*: \text{two lowest freq letters.} \\
\quad S' & \leftarrow S \setminus \{y^*, z^*\} \cup \{\omega\} \\
\quad f \omega & \leftarrow f y^* + f z^* \\
\quad T' & \leftarrow \text{Huffman}(S')
\end{align*}
\]
Traverse $T'$ and at the leaf labeled $w$, create two children $y^*$ and $z^*$. Call this new $T$. Return $T$. 

\[ T' \]