I. Core / D. Dynamic programming and greedy algorithms

- Dynamic programming review and overview (Wed, Sept 25)
- Dynamic programming practice (Fri, Sept 27–Wed, Oct 2)
- Greedy algorithms overview (Fri, Oct 4)
- Greedy algorithms practice (last week Monday)
- Review for Test 1 (last week Wednesday)
- Test 1, itself (last week Friday)
- Greedy algorithms finale: Huffman encoding (**today**)
- (Begin FFT Wednesday)

Today:

- Some test comments
- Huffman encoding overview
- Lemma 16.2 and proof
- Exercises from Section 16.3

If $f(n) = \omega(g(n))$ then $f(n) \neq O(g(n))$.

Proof. *Suppose* $f(n) = \omega(g(n))$. *Further suppose* $f(n) = O(g(n))$.

By definition of big-Oh, there exists c *and* n_0 *such that for all* $n \geq n_0$, $0 \leq f(n) \leq cg(n)$. *By definition of little-omega, there exists* n_1 *such that for all* $n \geq n_1$, $cg(n) < f(n)$.

Let $n_2 = \max(n_0, n_1)$. *Then* $cg(n_2) < f(n) \leq cg(n_2)$, *which is a contradiction.*

Therefore $f(n) \neq O(g(n))$. $\square$

No algorithm to transform an arbitrary binary tree with n comparable keys to a binary search tree with the same keys in expected time $o(n \lg n)$ exists.

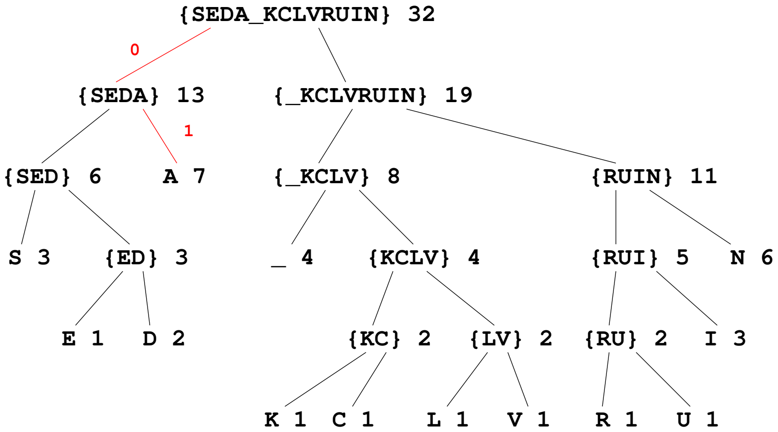
Proof. *Suppose such an algorithm for BST-building exists. Then, construct the following algorithm:*

1. *Given an array with of comparable keys, transform that array into a binary tree, such as by making it a long dangly list-like tree. This takes* $\Theta(n)$ *time.*
2. *Transform that binary tree into a BST using the supposed algorithm. This takes* $o(n \lg n)$ *time, according to our supposition.*
3. *Transform the BST into a sorted array by traversing the tree. This takes* $\Theta(n)$ *time.*

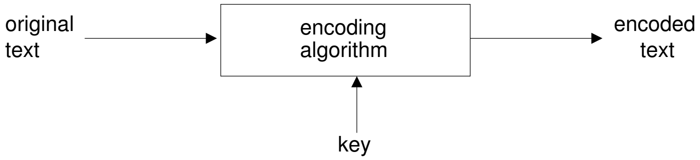
This algorithm sorts the given array, and takes $\Theta(n) + o(n \lg n) + \Theta(n) = o(n \lg n)$ *time. By Theorem 8.1, this is impossible. Therefore no such algorithm for BST-building exists.* $\square$

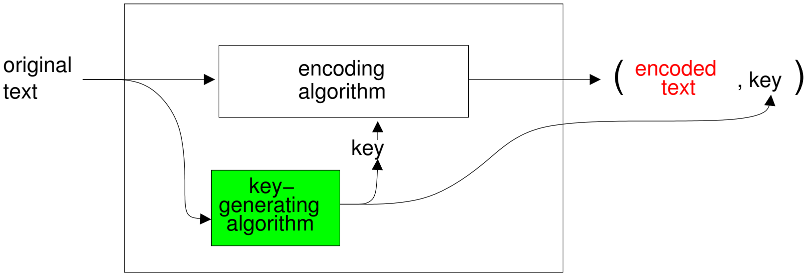
4.c. Consider the sequence of `delete()` operations from just after a defragmentation up through and including the next defragmenting. Let m be `size` at the beginning of this sequence. There will be $\frac{m}{2} - 1$ operations that are constant time, and the, last, defragmenting operation costs $O(m)$. All together this costs $O(m)$. Using the aggregate method, we spread the $O(m)$ cost across the $\frac{m}{2}$ operations to consider them $O(1)$ each. Using the accounting method, charge each of the $\frac{m}{2} - 1$ non-defragmenting operations 3 units, one for the nulling of the position itself, and two for its contribution to the next defragmenting (one for a nulling, one for a filling).

From DMFP:



01	111	111	1101	10100	01
A	N	N	I	K	A


original
text
encoding
algorithm
encoded
text
key


original
text
encoding
algorithm
key
key–
generating
algorithm
(encoded text , key)

Lemma 16.2, restated from CLRS pg 433:
Let x and y be characters in n alphabet with the lowest frequencies in the original text. Then there exists an optimal prefix code for the alphabet (that is, *optimal for the original text*) in which the encodings of x and y have the greatest length.
Proof sketch. Let T be an optimal tree. Let a and b be characters represented by sibling leaves of maximal depth. WOLOG, let $a.freq \leq b.freq$ and $x.freq \leq y.freq$. By how x, y, a, and b are chosen,

$$\begin{aligned} x.freq &\leq a.freq \\ y.freq &\leq b.freq \end{aligned}$$

Let T'' be the prefix code like T except with x and a switched, y and b switched. Then

$$\begin{aligned} B(T) - B(T'') &= \sum_{c \in C} c.freq \cdot d_T(c) - \sum_{c \in C} c.freq \cdot d_{T''}(c) \\ &= (a.freq - x.freq)(d_T(a) - d_T(x)) \\ &\quad +(b.freq - y.freq)(d_T(b) - d_T(y)) \\ &\geq 0 \end{aligned}$$

□

Lemma 16.3, summarized from CLRS pg 435:
Optimal trees have subtrees that are optimal for their corresponding subproblem.

Theorem 16.4, restated from CLRS pg 435:
Huffman trees are prefix codes that are optimal for the given original text.

Proof sketch. Let C be the alphabet of the text, augmented with character frequencies. By induction on the structure of the tree produce by the Huffman encoding.

Base case. Suppose C has only one character. Then there is only one possible tree for that alphabet, so it must be optimal.

Inductive case. Suppose C has more than one character, and let x and y be the the least frequent characters. Let C' be the alphabet like C but with x and y replaced with pseudo-character z. By structural induction, the Huffman encoding produces a tree that is optimal for C'. By Lemma 16.3, we can replace leaf z in the optimal tree for C' with a parent of siblings x and y to make a tree optimal for C. □

Invariant. (Worklist loop of Huffman tree-building algorithm)

(a) The `worklist` contains subtrees of an optimal key for `msg`.

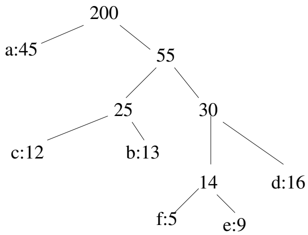
(b) Each character type in `msg` occurs in exactly one subtree of `worklist` exactly once.

Initialization. *Before the loop starts, the worklist contains only leaves, all of which must be in any prefix-code key for* `msg`*. Moreover, all of the character types in* `msg` *are represented by exactly one leaf.*

Maintenance. *Let k be an optimal prefix-code tree for* `msg` *that contains all the subtrees in* `worklist`*. . . .*

Solution to 16.3-2. Suppose T is a non-full prefix code binary tree. Let x be a node with one child. Replace that node with its child; that reduces the depth of all characters underneath by 1. Hence T was not optimal.

Solutiuon to 16.3-4. Claim: sum of the internal nodes' combined frequencies equals sum of the products of leaf frequencies and their depths. For example, consider this tree:



In this case, e's 9 occurrences each take 4 bits. The 9 is counted four times.
For an internal node x, the sum of the internal nodes' combined frequency of children is equal to the sum of leaf frequencies times their depth from x.

Solutiuon to 16.3-4, continued.

Proof. By structural induction.

Base case: Suppose x is an internal node both of whose children, a and b, are leaves. Then the combined frequency is

$$a.freq + b.freq = a.freq \cdot 1 + b.freq \cdot 1$$

...which is the leaf frequencies times their depths.

Inductive case 1: Suppose x is an internal node with one child being a leaf (a) and the other being itself an internal node (c); suppose that the claim we're making for the entire tree is true for the subtree rooted at c. Let d be the combined frequency of c and d' the sum of the combined frequencies of internal nodes under c. Let $c_1, \ldots c_m$ be the leaves under c with depths (from c), $c'_1, \ldots c'_m$. Then the sum of the combined frequencies under x is

$$\begin{aligned} a.freq + d + d' &= a.freq + d + c'_1 \cdot c_1.freq + \ldots c'_m \cdot c_m.freq && \text{by the ind hyp} \\ &= a.freq + c_1 + \cdot + c_m + \\ &\quad + c'_1 \cdot c_1.freq + \ldots + c'_m \cdot c_m.freq \\ &= 1 \cdot a.freq + (c'_1 + 1)c_1.freq + \ldots (c'_m + 1)c_m.freq \end{aligned}$$

The argument is similar in inductive case 2, where both children are themselves internal nodes. □

For next time:

Read Sec 30.1 (and the Chapter 30 introduction)

Do Ex 30.1-2

Read (and attempt) Ex 30.1-3. I don't have a solution to this one myself; I'm curious if any of you can make progress on it.