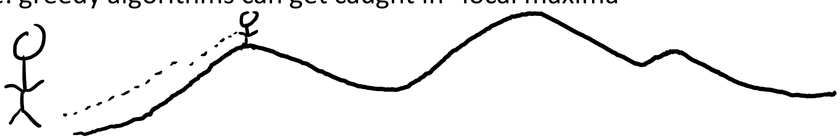
2019-02-04 Greedy Algorithms - Huffman Coding

Monday, February 4, 2019 3:41 PM

Greedy Algorithms

- General idea: For each decision point, do whatever makes the most sense at that point
  - "Should I take 1 or 2 pennies?" -> 2
  - "My goal is to get to the top of the mountain, should I take the steep slope or shallow slope" -> steep
  - "I'm trying to get to Eureka, should I take Old Arcata Road or HWY 101" -> HWY 101
- Implicit assumption: we are trying to minimize expensive work
- Many greedy algorithms use priority queues to manage decision making
- Downside: greedy algorithms can get caught in "local maxima"



  - Guy is trying to get to highest point, gets stuck because greedy says to not go down hill, only gain elevation
- Easy example: 1-0 knapsack problem
  - Given a bunch of stones with associated weights and values and a fixed-sized bag (knapsack); how much value can we store?

A 5 lb $10

B 7 lb $15

sack weight = 10 lbs

B greedy, not most efficient

  - Divisible Knapsack problem
    - You're allowed to chisel off portions of the rock

A 5 lb $10

B 7 lb $15

2.5B B = $21 ✓

Huffman Coding

- Huffman coding is a common compression algorithm using a binary trees and a greedy algorithm. Huffman coding is one possible compression algorithm that can be used for ZIP files.
- Huffman coding works by finding patterns in sequences.
  - This is very easily achievable in ASCII
- Consider the ASCII string "aaa"
  - In binary, 01100001 is "a" thus, "aaa" is 01100001 01100001 01100001
- Idea: run a substitution algorithm that replaces common sequences with shortcut characters
- Thus, if "a" becomes single bit 0, the sequence then becomes 000, which is a savings of 21 bits or 87.5% compression
- Goal: map long, wasteful bit sequences into shorter bit sequences

Example #1

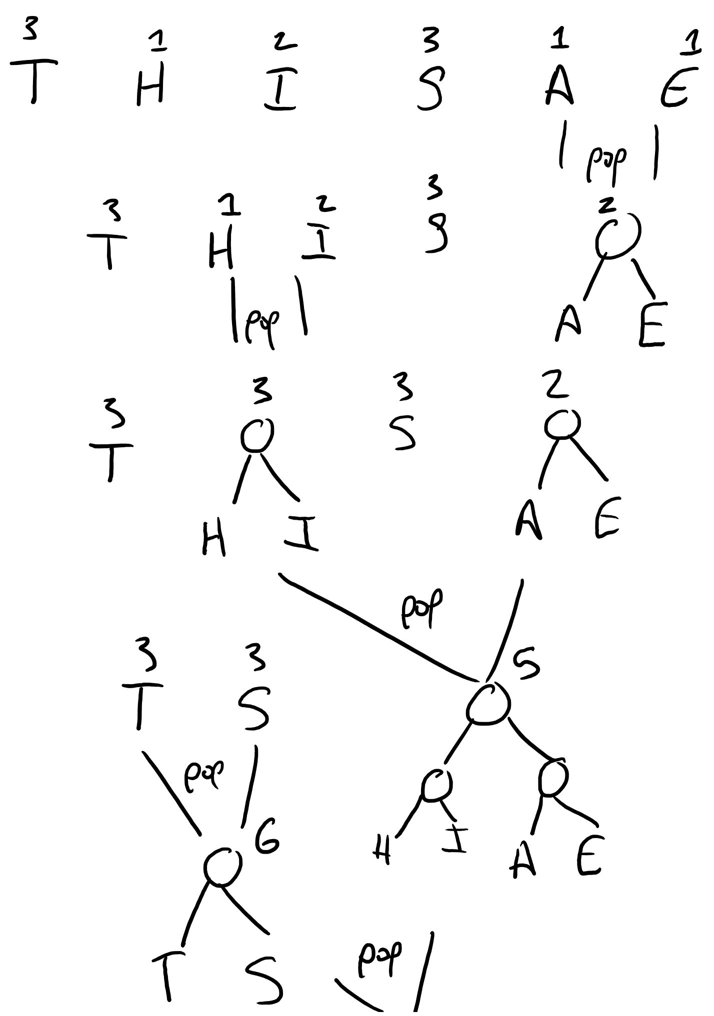
- Consider the string "aaabbc". What binary mapping would yield the shortest sequence?
- Option #1: c = 0, b = 1, a = 01 would result to: 010101110
- Option #2: a = 0, b = 1, c = 01 would result in: 0001101 <-- shorter
- It turns out that by giving the most common characters the shortest mappings will always yield the shortest sequence.

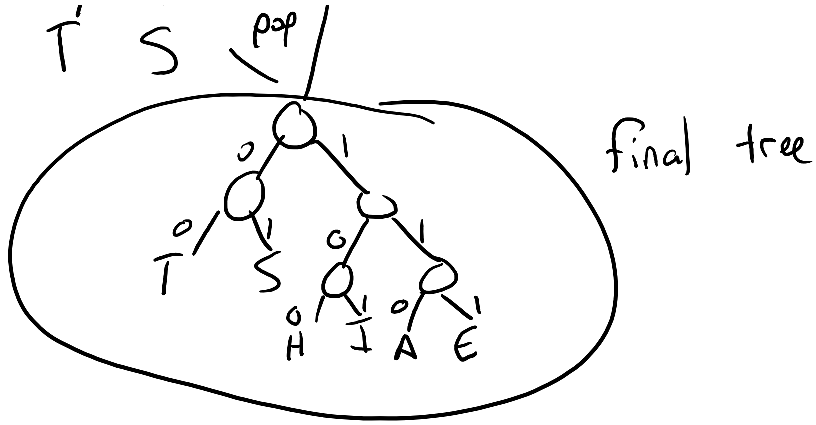
Huffman Coding Compression Algorithm

1. Build a frequency map of all unique chars in the file
- unordered_map<char,int>
- Example encode: "ThisIsATest" (assume not case sensitive)

T	H	I	S	A	E
III	I	II	III	I	I

2. Build a forest based on frequency distribution
   a. Each tree in forest has a value (char) and weight (frequency)
3. Until there exists a forest, merge forests based on weight
   a. We will throw all forests into a MIN PQ
   b. While the PQ is not empty:
      i. Pop off top two forests, Merge. Push resulting forest back onto PQ.





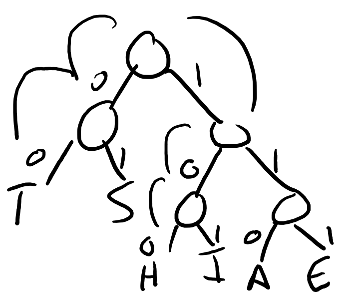
- Now that we're done, we conceptually label each edge in the tree. Left edges have a value of 0. Right edges have a value of 1. We walk the tree using a pre-order traversal in order to derive our final character mapping.

  T: 00
  S: 01
  H: 100
  I: 101
  A: 110
  E: 111

  ThisIsATest -> 00|100|101|01|101|01|110|00|111|01|00 (26 bits)
  Original was 11 bytes (88 bits). Compression is 26/88 = 0.2955; 1 - .2955 = 0.7045

Decompressing Huffman Codes

Given a huffman tree, to decompress, all you do is walk the tree; reset after leaf node.



0010010101101011100011110100

TH