CSCE 411: Design and Analysis of Algorithms

Lecture 6: Greedy Algorithms

Date: February 5, 2025 *Nate Veldt, updated by Samson Zhou*

Course Logistics

- Greedy algorithms and amortized analysis this week: Chapter 16
- Homework 3 is posted, due this Friday

1 Binary Codes

- An ________________ is a set of characters, e.g., $C = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}\}$
- A *binary code* for C is

Consider three possible codes for $C = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}\}$.

Character:	a	b	c	d	e	f
example code 1:	000	001	010	011	100	101
example code 2:	0	101	100	111	1101	1100
example code 3:	0	1	10	01	11	00

(1)

Types of codes:

- **Fixed length code**:
- **Variable-length code**:
- **Prefix code**: the codeword for every $c \in C$ is

Question 1. Which of the codes in Table 1 is a prefix code?

A Example code 1

B Example code 2

C Example code 3

D Example codes 1 and 2

E Example codes 1, 2, and 3

1.1 Encoding and Decoding

Consider a string of characters from the alphabet $C = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}\}$ where the frequency of each character is given in the following table and we have two codes.

Character:	a	b	c	d	e	f
Frequency	0.45	0.13	0.12	0.16	0.09	0.05
FLC:	000	001	010	011	100	101
VLC:	0	101	100	111	1101	1100

(2)

1.2 The Optimal Prefix Code Problem

The optimal prefix code problem Given an alphabet C and a frequency $c.freq$ for each $c \in C$ in a given string s,

Representing a prefix code as a tree Every prefix code for a string s can be represented as a binary tree T where

- Left branches are associated with
- Right branches are associated with
- Each leaf corresponds to a character $c \in C$, whose binary code is given by
- Each node is associated with the frequency of characters in its subtree

Example

Character:	`a`	`b`	`c`	`d`	`e`	`f`
Frequency	0.45	0.13	0.12	0.16	0.09	0.05
VLC:	0	101	100	111	1101	1100

The cost of the tree is give by

$$B(T) = \sum_{c \in C}$$

where $d_T(c)$ is the depth of c in the tree T.

Question 2. True or false: every binary tree (with ℓ leaves) gives a valid prefix code (for an alphabet with ℓ characters).

A True

B False

Theorem 1.1.

Proof.

Optimal prefix code problem:

Nice properties about the optimal tree In the optimal tree representation,

-
-

1.3 Huffman Codes

Huffman Code: a prefix code obtained by greedily building a tree representation for C

Huffman Code Tree Process

- Associate each character in C with a node, labeled by its frequency
- Identify two nodes x and y in C with the ______________________________
- Create new node z with frequency ____
- Make $z.left = x$, and $z.right = y$.
- Repeat the above procedure on the alphabet obtained by _________________________

Observe: x and y will be siblings in T at _____________________________

Activity Create a Huffman code for:

Character:	`a`	`b`	`c`	`d`	`e`	`f`
Frequency	0.45	0.13	0.12	0.16	0.09	0.05

```
HuffmanCode(C)
  n = |C|
  T ← empty graph
  Q ← ∅
  for c in C do
      Insert c with value c.freq into T and Q
  end for
  for i = 1 to n − 1 do
      x =
      y =
      create node z









  end for
  Return T
```

1.4 Code and Illustration

1.5 Runtime Analysis

Given a set of objects C, with values $c.val$ for $c \in C$ and $n = |C|$, a binary min-heap for C is a binary tree such that

- All levels
- The value of a node is

It has the following operations:

- $\textsc{BuildBinMinHeap}(C)$:
- $\textsc{ExtractMin}(Q)$:
- $\textsc{Insert}(Q, z, z.freq)$:

Question 3. Assume we use a binary min heap to store and update Q in the pseudocode above. Then what is the overall runtime of creating a Huffman code, in terms of $n = |C|$?

A $O(\log n)$

B $O(n)$

C $O(n \log n)$

D $O(n^2)$

1.6 Optimality

It turns out that a Huffman code optimally solves the prefix code problem. The argument hinges on the following lemma.

Lemma 1.2. *Let C be an alphabet where $c.freq$ is the frequency of $c \in C$. Let x and y be the two characters in C that have the lowest frequencies.*

Then, there exists an optimal prefix code for C in which x and y have the same length code words and only differ in the last bit.

Equivalently:

Why does this imply optimality? Consider a slightly different (but equivalent) approach to defining a Huffman code:

1. Associate each character with a node, labeled by its frequency
2. Identify the two nodes x and y with the smallest frequencies
3. Create new node z with frequency $z.freq = x.freq + y.freq$.
4.
5. Create tree T from T' by making $z.left = x$, and $z.right = y$.

Claim This produces an optimal tree T.

Proof. Note that the cost of the tree T is

$$B(T) = B(T') + x.freq + y.freq \tag{3}$$

We prove the result by contradiction: suppose that T is not optimal. Then

We can assume that x and y are at maximum depth in R.

Let R' be the tree obtained by taking R and

Similar to (3) we have that:

We then get an impossible sequence of inequalities:

$$\begin{aligned} B(R') &= B(R) - x.freq - y.freq \\ &< B(T) - x.freq - y.freq \\ &= B(T') \end{aligned}$$