

**PROBLEM SET 8**  
**Due: Friday, December 6**

**Reading:** CLR Chapter 36; Section 17.3 (Huffman coding)

**Suggested Problems:** 36.1-2, 36.1-7; 36.2-2, 36.2-4;

**Problem 1 [10]:** CLR 36.1-1 (p. 923). Note that the graph is an *unweighted* graph, so the length of a path is just the number of edges in it. To prove the “if and only if” condition, you must show two things:

- 1) If there is an algorithm that computes LONGEST-PATH-LENGTH in polynomial time, then LONGEST-PATH  $\in$  P.
- 2) If LONGEST-PATH  $\in$  P, then there is an algorithm that computes LONGEST-PATH-LENGTH in polynomial time.

**Problem 2 [15]:** CLR 36.2-8 (p. 929).

**Problem 3 [15]:** CLR 36.3-2 (p. 938).

**Problem 4 [10]** Snake Oil Software's CEO Kim Priest has just announced an amazing compression program that is guaranteed to compress *every* binary file by a factor of two, effectively doubling the size of your computer's memory. (A length- $n$  binary file is just a sequence of  $n$  0s and 1s.) You have been asked to review this program.

- a. Based on a simple counting argument, explain why the program cannot work as advertised.
- b. Extend the reasoning in part a to explain why any compression technique that shrinks some files must increase the size others.
- c. Given the problems raised by **a** and **b**, explain why compression techniques are still valuable in practice.

**Problem 5 [50]** (Adapted from Abelson and Sussman, *Structure and Interpretation of Computer Programs*, p. 125).

Here we consider various approaches to compressing the following 1950s style song lyrics:

```

get a job
sha na na na na na na na na
get a job
sha na na na na na na na na
wah yip yip yip yip yip yip yip yip
sha boom
    
```

For simplicity, the only characters are lower-case letters, spaces, and line-ending carriage returns (written `\n`). Here are some useful facts about the lyrics:

Total number of characters: 121

Total number of words (counting `\n` as a word): 41

Word frequency table:

a	boom	get	job	na	sha	wah	yip	
2	1	2	2	16	3	1	8	6

Character frequency table:

a	b	e	g	h	i	j	m	n	o	p	s	t	w	y		
22	3	2	2	4	8	2	1	16	4	8	3	2	1	8	29	6

- a. There are 17 distinct characters in the song (counting space and carriage return).
  - i. What is the smallest number of bits that it would take to represent one character in a fixed-length binary code?
  - ii. How many bits would it take to encode the entire song using this fixed-length code?
- b. i. Using the character frequency table, derive a Huffman code tree for this example.
  - ii. Make a table showing the codes for the 17 individual characters.
  - iii. Ignoring the space for encoding the tree/table, how many bits would it take to encode the song?
  - iv. Some encoding of the tree/table must be sent along with the encoded message to allow it to be decoded. There are many ways to encode the table. Let's assume it is encoded as a sequence of bindings followed by a distinguished 8-bit byte that separates the table from the message. Each binding consists of an 8-bit ASCII representation of the character being encoded, followed by an 8-bit binary number indicating the length  $n$  of the code for the character, followed by the  $n$  bits that are the code for the character. For example, if the character 'a' had the five-bit code 10011, then the binding would consist of the following bits:

ASCII 'A'	Binary 5	5-bit code
-----------	----------	------------

01100001	00000101	10011
----------	----------	-------

Based on this encoding of the table, how many bits would be required to encode the song, including the encoding of the table and the byte separating the table from the message?

**c.** Rather than using a Huffman code to encode the individual characters, it is also possible to use a Huffman code to encode the song as a sequence of words. (We count carriage returns as words, and need not encode spaces as they can always be inferred from the placement of words.)

**i.** Using the word frequency table, derive a Huffman code tree for the song when viewed as a sequence of words.

**ii.** Make a table showing the Huffman codes for the 9 words.

**iii.** Assuming words are no longer than 64 characters, each table binding can be encoded as the concatenation of:

- a 6-bit number encoding the number of characters  $k$  in the word.
- the 8-bit ASCII codes for the  $k$  characters of the word
- an 16-bit number encoding the length  $n$  of the Huffman code for the word
- the  $n$  bits of the Huffman code for the word.

Using this encoding, how many bits would be required to encode the song, including all the table bindings and a 6-bit separator between the table and message?

**d.** Another approach to compressing the lyrics is to use a completely ASCII-based representation in which the compressed message is preceded by a set of macro substitutions that are allowed in the text of the compressed message. A macro binds a name to a text string. Anywhere the macro name appears subsequently, it can be replaced by the text string.

Assuming that all strings in the compressed message contain no upper-case characters, we will use single upper-case characters as macro names. The syntax for a macro binding will be `<name>=<string>`. A sequence of macro bindings will be separated from the main text of the message by the two-character sequence `==`.

For example, the text "he said she said" could be encoded as

```
A=said==
he A she A
```

or

```
A=he B=said==
AB sAB
```

or even

```
C=he said she said==
C
```

In this case, all of the sample encodings actually use as many or more characters than the original text, but judicious substitutions can often result in a significant reduction in the size of the message.

A substitution defined in one binding is allowed too be used in subsequent bindings. For example, 'he ate slate late ' could be encoded as:

```
A=ateB=lA==  
he A sB B
```

Note that a use of a macro name can be distinguished from a definition because a definition is immediately followed by a single = while a use is not.

- i. Develop the shortest substitution-based encoding you can think of for the song.
- ii. Representing each character by an 8-bit ASCII code, how many bits would be required to represent your encoding?

### Extra Credit Problems:

**Problem EC1 [15]:** CLR 36.2-3 (p. 928). The problem could be rephrased as follows: Suppose you are given a black box predicate HAS-HAM-CYCLE?(G) that can determine in polynomial time whether G has a hamiltonian cycle. How could you use such a predicate to compute in polynomial time an ordered list of vertices for a hamiltonian path in a graph H for which HAS-HAM-CYCLE?(H) returns true? You do *not* need to give pseudocode, but need to give enough details so that it is clear how your algorithm works.

**Problem EC2 [10]:** CLR 36.2-7 (p. 929). *Hint:* Topologically sort the vertices. What must be true of the result?

### Problem EC3 [15]: Get Rich Quick with Huffman Coding.

The following game was invented by Mads Tofte at DIKU university in Denmark. In his own words:

[There are] two players, A and B. A rolls two dice, thus producing a sum between 2 and 12. B has to guess the sum. B can ask A any question he likes, but A must answer only yes or no. B pays one unit for each question he poses, until he knows the correct answer. Then the roles are switched. This cycle is repeated 10 times (to give statistics a chance) and the points are added up. The one who has asked the fewest questions altogether wins. The optimal strategy turns out to require roughly 33 questions for 10 rounds. A reasonable opponent uses around 36 questions, in my experience, which makes the opponent want to try again. Excellent pub game (that is what I invented it for), as long as one can remember the Huffman tree (and re-arrangements of it) by heart.

Construct the Huffman tree implied by the problem, and show why it offers the competitive advantage mentioned in Mads's description.

*Problem Set Header Page*  
*Please make this the first page of your hardcopy submission.*

**CS231 Problem Set 8**  
**Due Friday, December 6,1996**

Name:

Date & Time Submitted (*only if late*):

Collaborators (*anyone you collaborated with in the process of doing the problem set*):

*In the **Time** column, please estimate the time you spent on the parts of this problem set. Please try to be as accurate as possible; this information will help me to design future problem sets. I will fill out the **Score** column when grading your problem set.*

<b>Part</b>	<b>Time</b>	<b>Score</b>
General Reading		
Problem 1 [10]		
Problem 2 [15]		
Problem 3 [15]		
Problem 4 [10]		
Problem 5 [50]		
EC1 [15]		
EC2 [10]		
EC3 [15]		
<b>Total</b>		