Analysis and Design of Algorithms
Homework 3

Arturo Fornés Arvayo A01227071

March 19, 2017

1. From Algorithms by Dasgupta:

   (a) Excercise 5.18
   Given the following alphabet and frequencies:

blank 18.3%	r 4.8%	y 1.6%
e 10.2%	d 3.5%	p 1.6%
t 7.7%	l 3.4%	b 1.3%
a 6.8%	c 2.6%	v 0.9%
o 5.9%	u 2.4%	k 0.6%
i 5.8%	m 2.1%	j 0.2%
n 5.5%	w 1.9%	x 0.2%
s 5.1%	f 1.8%	q 0.1%
h 4.9%	g 1.7%	z 0.1%

   i. What is the optimum Huffman encoding for this alphabet?
   Using my implementation of the Huffman Procedure included in the file *huffman.c* to encode the alphabet

blank - 111	r - 0000	y - 100101
e - 010	d - 10111	p - 100110
t - 1100	l - 10110	b - 100100
a - 1010	c - 00101	v - 1101110
o - 1000	u - 00100	k - 11011110
i - 0111	m - 110110	j - 1101111110
n - 0110	w - 110101	x - 1101111111
s - 0011	f - 110100	q - 1101111100
h - 0001	g - 100111	z - 1101111101

   ii. What is the expected number of bits per letter?

$$E(letter) = \sum_{letter=blank}^{z} \text{number of bits used for letter} * \text{probability that letter appears}$$

   This summation yields, as calculated using the code included in the file *expected.cpp*:

$$E(letter) = 4.201$$

iii. Entropy of the frequencies:

$$H = \sum_{i=0}^{26} p_i lg(\frac{1}{p_i})$$

Where $p_i$is the frequency of each letter i, it yields:

$$H = 4.1493$$

The entropy represents the randomness of the frequencies, and yields that there are 4.1493 bits of randomness in the alphabet. It would be smaller than that of the expected value as entropy models, in a way, the expected value.

iv. Is this the limit of how much the english language can be compressed? What features besides letters and their frequencies should be taken into account?
No, as we are still lacking punctuation, we could still compress by sentences and their uses or importance, instead of frequencies and other variations.

(b) The code for the implementation of Huffman is in *huffman.c*. Compressing a string of 21 characters which usually would ocupy 168 bits, resulted in a bit sequence of length 87. A compression rate of 1.93103448, reducing it's length almost in half.

2. Cormen 15.10

(a) Prove that there exist an optimal investment strategy such that you make a single investment every year putting all your money into a single investment.

Supposing that you were to invest into multiple companies, in year 1 say $d_1$dollars into a company i with rate r_{i1} and d_2 dollars into a company a with rate r_{a1}. Keeping this investment for j years will yield the following sequence:

$$r_{i1} + ... + r_{ij} > r_{a1} + ... + r_{aj}$$

Following this, it would have been better to invest $d_1 + d_2$ in company i, this sequence works for j years, then there exists an optimal investment strategy putting all your money into one company per year.

(b) Prove that your optimal investment strategy exhibits optimal substructure.

Each investment decision is independent to the amount of money made previously, so each decision is bound to the fees, whether we move the money or we don't. Moving the money transfers the problem to another company, while staying keeps the problem in the same company. This decision yields that we have an optimal substructure.

(c) Algorithm included in the function invest in file *investment.c*. Complexity is $O(j * n)$, where j is the years and n is the companies.

(d) Having a limit of the money made makes the problem a different one based on every single decision that leads to reaching 15,000 at any point, thus overcomplicating the problem, not exhibiting an optimal substructure anymore as every decision is made dependant on the money at that point.

(e) The file *investment.c* includes the solution to the given problem.

3. Cormen 17.2

(a) Since we don't know in which list a value may be, and lists hold no relationship to each other, we search linearly through the lists and do a binary search in each one; the worst case (every list is used and the binary search is a complete one). Each list has length 2^i so traversing it can be done in $O(i)$, since binary search takes $O(lgn)$, but because $0 \leq i \leq lgn$, the total search is $O((lgn)^2)$.

(b) In insertion, assume we change a lists from 1 to 0, then we change a 0 to a 1 in the binary representation of n. We then merge lists A_0 through A_{a-1} into list A_a, then insert the new element into list A_a . We would be merging two lists, sorted lists, this can be executed in linear time, where a is the total length of the resulting list: the time this takes is $O(2^a)$, since each list is of length 2^i, and 2^a represents the size of the resulting list. Worst case, it takes time $O(n)$, since we could be merging at the last list, whose length is $k = \lceil log(n+1) \rceil$. For the amortized cost, the total of elements in the list is 2^a, then it follows that the cost per element is $O(1)$, then the n operations performed in the worst time has amortized cost $O(n)$.

(c) For deletion, in the binary representation of n, get the smallest m that follows: $n_m \neq 0$. When element to be deleted is not in this list A_m , remove it from its list and replace it with an element from the found list A_m. Since we need to search the list to find the element to delete, this takes $O(lgn)$. Finally, inversely to the insertion, we split list A_m into lists A_o through A_{m-1}, these split takes $O(m)$ time. The worst case of delete would take $O(lgn)$.