HW3

Miguel A. Montoya Z. (A01226045)

September 26, 2017

Answer if the theoretical complexities are correct

1. From Dasgupta

   (a) Do exercise 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%

   a) What is the optimum Huffman encoding for this alphabet?

   Using my implementation of the Huffman Procedure in **Huffman.ccp**:

blank	111	r	0000	y	100101
e	010	d	10111	p	100110
t	1100	l	10110	b	100100
a	1010	c	00101	v	1101100
o	1000	u	00100	k	11011010
i	0111	m	110111	j	1101101110
n	0110	w	110101	x	1101101111
s	0011	f	110100	q	1101101100
h	0001	g	100111	z	1101101101

b) What is the expected number of bits per letter?

$E(letter) = \sum_{c \in C} c.freq * usedBits(c)$
Using the code included in **Huffman.ccp** the summation yields

$$E(letter) = 4.201$$

c) Calculate the entropy of frequencies

$H = \sum_{c \in C} c.freq * lg\left(\frac{1}{c.freq}\right)$
Using the code included in **Huffman.ccp** the summation yields

$$H = 4.1493$$

Would you expect it to be larger or smaller than your answer above? Explain.

The entropy represents the randomness of the frequencies. In this case, how many random bits are we going to get in the alphabet. I was expecting both values to be close, as every expected bit should be random. In a way, both values represent the same thing.

Do you think that this is the limit of how much English text can be compressed? What features of the English language, besides letters and their frequencies, should a better compression scheme take into account?

No, a language is more complex than simple characters. Punctuation, complete words and full phrases even, could be taken into account to compress the language even more.

i. Then implement the Hoffman's code solution in C++.

   Code was implemented in **Huffman.cpp.** This includes the Hoffman's solution, level traversal, and the expected number of bits and entropy calculations.

ii. Then, selecting a large enough corpus compare tell me what was your average compression.

   Considering a character takes 8 bits, the number of bits necessary to encode 1000 characters would be 8000. Using this implementation, only 4150 bits would be used. This gives a compression rate of 1.92771084, reducing the lenght almost in half.

(b) Do exercise 5.19 (Written solution).

(c) Do exercise 6.7 (Written solution).

(d) So exercise 6.14.

i. Implement a solution in C++.

2. From Cormen

   (a) Do exercise 15.10 (Back of the chapter 15).

   a) Prove that there exists an optimal investment strategy that, in each year, puts all the money into a single investment.

   Supposing that an investment of d_1 dollars is made to the investment a and d_2 into investment b in year 1. Also supposing the money isn't moved from the investment in j years. The sum of the revenues would be:

$$R_a = r_{a1} + r_{a2} + \ldots + r_{aj}$$

$$R_b = r_{b1} + r_{b2} + \ldots + r_{bj}$$

   This would result in the solution S:

$$R_a \geq R_b$$

   So instead, we can invest $d_1 + d_2$ into a and generate at least as S, but in just one investment.

   b) Prove that the problem of planning your optimal investment strategy exhibits optimal substructure.

   Because each yearly decision is completely independent to the quantity of money made from previous years, the only decision to take (Apart of choosing the investment with a bigger revenue) is to decide whether or not to change the investment and pay a bigger fee. Thus, each year, the problem exhibits an optimal structure.

   c) Design an algorithm that plans your optimal investment strategy. What is the running time of your algorithm?

   $\mathcal{O}(jn)$, where j is the number of years, and n is the number of companies.

Algorithm 1 Investment algorithm

Investment(d, n)

1. Initialize I, where I[i] stores the investment number to be made in year i
2. Initialize R, where R[i] stores the accumulated return in year i
3. **for** k **in** range(9) //Number of years

3.1 $q = 0$

3.2.1 **for** i **in** range(n) //Number of investments

3.2.2 **if** $r_{ik} > r_{qk}$

3.2.3 $q = 1$

3.3 **if** $R[k-1] + d * r_{I[k-1]k} - f_1 > R[k-1] + d * r_{qk} - f_2$

3.3.1 $R[k] = R[k-1] + d * r_{I[k-1]k} - f_1$

3.3.2 $I[k] = I[k-1]$

3.4 **else**

3.4.1 $R[k] = R[k-1] + d * r_{qk} - f_2$

3.4.2 $I[k] = q$

4. **return** I as investment strategy, and R[9] as resultant return

d) Suppose that Amalgamated Investments imposed the additional restriction that, at any point, you can have no more than $15,000 in any one investment. Show that the problem of maximizing your income at the end of 10 years no longer exhibits optimal substructure.

Having a limited quantity of money I can invest in a company changes the problem. With this restriction, we have to keep in mind the quantity of money available to invest the next years. We have to analyze the coming years for each different quantity of initial money invested into a company.

i. Implement the solution in C++. Please, run it against the scenario described by http://statmath.wu-wien.ac.at/~zeileis/grunfeld/

The file **Investment.cpp** contains the implementation and data of the case.

(b) Do exercise 17.2 (Back of the chapter 17)

a) Describe how to perform the SEARCH operation for this data structure. Analyze its worst-case running time.

Because we don't know close to no information regarding the location of the value, and the relationships of the lists, we would have (In the worst case) search linearly the lists, and do a binary search in each one. Because in the worst case, every list is used ($\mathcal{O}(lg(n))$) and a full binary search is done ($\mathcal{O}(lg(2^i)) = \mathcal{O}(i)$), the complete complexity is $\mathcal{O}(lg^2(n))$

b) Describe how to perform the INSERT operation. Analyze its worst-case and amortized running times.

First we change the first m 1's in a row to 0's, and a 0 from the binary representation of n, to 1. Then we combine m lists (Trough A_0 to A_{m-1}), and insert the new item to this new list. Since merging sorted lists take the total length of the lists, the worst case is $\mathcal{O}(n)$. And because of merging 1 element is $\mathcal{O}(1)$, after n elements, the amortized cost is $\mathcal{O}(n)$.

c) Discuss how to implement DELETE.

For the deletion in the binary representation of n, get the smallest m that follows $n_m \neq 0$. Search the element to be deleted in the list A_m, if it isn't there, search for it in the other lists and swap it with an element of A_m. Afterwards, delete the element. This can be done in $\mathcal{O}(lg(n))$, because the search of the element in the list.