ICS 311, Fall 2020, Problem Set 03, Topics 5 & 6 Solutions

#1. Expected Length of Coding Scheme

6 points

Suppose that several information sources generate symbols at random from a five-letter alphabet {A, B, C, D, E} with different probabilities. We will try different encoding schemes for encoding these symbols in binary. **Given the probabilities and an encoding scheme, you will compute the expected length of the encoding of n letters generated by the information source. You may use any rigorous method of analysis you like, but must show your work and justify your answer.**

(a) Suppose that the symbols occur with probabilities Pr[A] =0.4 , Pr[B] = 0.2, Pr[C] = 0.2, Pr[D] = 0.1, and Pr[E] = 0.1, and the coding scheme encodes these symbols into binary codes as follows:

A	**001**
B	**010**
C	**011**
D	**100**
E	**101**

What is the expected number of bits required to encode a message of *n* symbols?

Solution:
Every character is coded with 3 bits, so the probabilities of symbols does not matter: it will take 3n bits to encode n symbols.

(b) Now suppose that the symbols occur with the same probabilities Pr[A] =0.4 , Pr[B] = 0.2, Pr[C] = 0.2, Pr[D] = 0.1, and Pr[E] = 0.1, but we have a different encoding scheme:

A	**0**
B	**10**
C	**110**
D	**1110**
E	**1111**

What is the expected number of bits required to encode a message of *n* symbols? Which encoding scheme is better for these probabilities?

Solution:

Let X_i be a random variable that defines the size of the encoding of the i-th symbol.

Then $E[X_i] = (1 * Pr[X_i=1]) + (2 * Pr[X_i=2]) + (3 * Pr[X_i=3]) + (4 * Pr[X_i=4])$

$= 1 * 0.4 \quad + 2 * 0.2 \quad +3 * 0.2 \quad + 4 * (0.1 + 0.1)$

$= 0.4 + 0.4 + 0.6 + 0.8$

$= 2.2$

And let X be the random variable that defines the size of encoding of n symbols.

Then $X = \sum_{i=1}^{n} X_i$

$$E[X] = E\left[\sum_{i=1}^{n} X_i\right]$$

$$= \sum_{i=1}^{n} E[X_i] \qquad \text{// By Linearity of Expectations}$$

$$= \sum_{i=1}^{n} 2.2$$

$$= 2.2n$$

(Comment not required in student solutions: If we encoded each symbol using 3 bits the length would be 3n, so the encoding has saved space.)

(c) Now consider a different information system that generates symbols with probabilities Pr[A] =0.5 , Pr[B] = 0.3, Pr[C] = 0.1, Pr[D] = 0.05, and Pr[E] = 0.05. We will use the same encoding scheme:

A	**0**
B	**10**
C	**110**
D	**1110**
E	**1111**

What is the expected number of bits required to encode a message of *n* symbols?

Solution:

Let X_i be a random variable that defines the size of the encoding of the i-th symbol.

Then $E[X_i] = (1 * Pr[X_i=1]) + (2 * Pr[X_i=2]) + (3 * Pr[X_i=3]) + (4 * Pr[X_i=4])$

$= 1 * 0.5 \quad + 2 * 0.3 \quad +3 * 0.1 \quad + 4 * (0.05 + 0.05)$

$= 0.5 + 0.6 + 0.3 + 0.4$

$= 1.8$

And let X be the random variable that defines the size of encoding of n symbols.

Then $X = \sum_{i=1}^{n} X_i$

$$E[X] = E\left[\sum_{i=1}^{n} X_i\right]$$
$$= \sum_{i=1}^{n} E[X_i] \qquad \text{// By Linearity of Expectations}$$
$$= \sum_{i=1}^{n} 1.8$$
$$= 1.8n$$

#2. Random Gene Sequences

6 points

Suppose a strand of DNA is generated by appending random nucleotides with equal probability from the set {A,T,G,C} to the end of the sequence. What is the expected length of the sequence needed to get two As in a row? <u>You may use any rigorous method of analysis you like, but must show your work and justify your answer.</u>

Solution: Let the expected number of nucleotides be *x*. There are three possible cases for the beginning of the sequence:

1) If the first nucleotide is not A, then the expected number of nucleotides becomes x+1. The probability of this case is ¾.
2) If the first nucleotide is A, but the second is not A, then the expectation becomes x+2. The probability of this case is ¾*¼ = 3/16.
3) If the first two nucleotides are A, then we are done and the length of the sequence is 2. The probability of this case is ¼*¼=1/16.

The expectation is just the sum of the expectations computed in these three cases, weighted by their probabilities. Then we solve for x.

x = ¾ * (x+1) + 3/16 * (x+2) + 1/16 * 2
16x = 12x + 12 + 3x + 6 + 2
x = 20

#3. Hashing with Chaining

6 points

(a) (2 pts - 1 for correct solution, 1 for argument or proof) Consider a hash table with m slots that uses chaining for collision resolution. The table is initially empty. What is the

probability that, after k keys are inserted, there is a chain of size k? Include an argument for or proof of your solution.

Solution: $1/m^{k-1}$
For a chain to have size k, all k keys must hash into the same location. We don't care where the first key hashes. Let's say it's in location *i*. Given this fact, we need to compute the probability that the remaining k-1 keys hash into the position *i* as well. Probability of a key hashing into a particular location is 1/m and is independent of any other value. Therefore, the probability of the remaining k-1 keys hashing into location *i* is $1/m^{k-1}$.

(b) (4 pts - 0.5 for correct placement of each) Show the table that results when 20, 51, 10, 19, 32, 1, 66, 40 are cumulatively inserted in that order into an initially empty hash table of size 11 with <u>chaining</u> and h(k) = k mod 11. Use the Google Doc table below, with positions indexed 0 to 10, and linked lists going off to the right. Do not enter anything in cells that are not used.

Solution:

h(k)	Linked list cells			
0	66			
1	1			
2				
3				
4				
5				
6				
7	40	51		
8	19			
9	20			
10	32	10		

Notice that the ordering is not 51, 40 or 10, 32: new entries are pushed on the linked list to save the cost of searching to the end.

#4. Open Addressing Strategies

12 points

(a) (4 pts - 0.5 for correct placement of each): Show the table that results when 20, 51, 10, 19, 32, 1, 66, 40 are cumulatively inserted in that order into an initially empty hash table of size 11 with linear probing and

h'(k) = k mod 11

h(k,i) = (h'(k) + i) mod 11, where the first probe is probe i=0.

Draw this and the next result as horizontal arrays indexed from 0 to 10 as shown below. (You can fill in the Google Doc table.) Show your work in part (b) to justify your answer!

Solution:

32	1	66	40				51	19	20	10
0	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***

(b) (1 pt - for correct count. Don't grade work but use it to debug problems): <u>How many re-hashes after collision are required for this set of keys?</u> *Show your work here so we can give partial credit or feedback if warranted.*

Solution: 10 rehashes/reprobes

I wrote an R script to generate the hashes. Here is a trace edited to make it shorter: count those with the 2nd argument:

```
> h0 <- function(k) k %% 11
> h <- function(k, i=0) (h0(k) + i) %% 11
> h(20)  ==>  9
> h(51)  ==>  7
> h(10)  ==>  10
> h(19)  ==>  8
> h(32)  ==>  10 # collides
> h(32, 1)  ==>  0  # rehashes have the 2nd argument
> h(1)  ==>  1
> h(66)  ==>  0 # collides
> h(66, 1)  ==>  1 # collides
> h(66, 2)  ==>  2
> h(40)  ==>  7# we get a string of collisions
> h(40, 1)  ==>  8
> h(40, 2)  ==>  9
> h(40, 3)  ==>  10
> h(40, 4)  ==>  0
> h(40, 5)  ==>  1
> h(40, 6)  ==>  2
> h(40, 7)  ==>  3 # Success!
```

(c) (4 pts - 0.5 for correct placements of each): Show the table that results when 20, 51, 10, 19, 32, 1, 66, 40 are cumulatively inserted in that order into an initially empty hash table of size m = 11 with double hashing and

h(k,i) = (h1(k) + ih2(k)) mod 11

h1(k) = k mod 11

h2(k) = 1 + (k mod 7)

Refer to the code in the book for how i is incremented. Show your work in part (d) to justify your answer!

Solution:

66	1	40		32			51	19	20	10
0	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***

(d) (1 pt - for correct count. Don't grade work but use it to debug problems)): <u>How many re-hashes after collision are required for this set of keys?</u> *Show your work here so we can give partial credit or feedback if warranted.*

Solution: 2 rehashes/reprobes

```
> h <- function (k, i=0) (h1(k) + i*h2(k)) %% 11
> h1 <- function(k) k %% 11
> h2 <- function(k) 1 + (k %% 7)
> h(20)  ==>  9
> h(51)  ==>  7
> h(10)  ==>  10
> h(19)  ==>  8
> h(32)  ==>  10 # collides
> h(32, 1)  ==>  4
> h(1)  ==>  1
> h(66)  ==>  0
> h(40)  ==>  7 # collides
> h(40, 1)  ==>  2
```

(f) (2 pts - 1 for correct answer, 1 for showing derivation from theorem): Open addressing insertion is like an unsuccessful search, as you need to find an empty cell, i.e., to not find the key you are looking for! If the open addressing hash functions above were uniform hashing, what is the expected number of probes at the time that the last key (40) was inserted? Use the theorem for unsuccessful search in open addressing and show your work. Answer with a specific number, not O or Theta.

> **Solution:** Theoretical is the same as unsuccessful search: $1 / (1 - \alpha) = 1 / (1 - n/m) = 1 / (1 - 7/11) = 1 / (4/11) = 11/4$ or 2.75. Thus our actual result of 2 is close to theoretical.

Skip Lists?

FYI, we decided to remove the skip list question, as it was merely tracing algorithms, not analyzing them.