Algorithms and Data Structures
MSIS 26:198:685
Homework 3

Instructor: Farid Alizadeh
Due Date: Sunday November 17, 2019, at 11:50PM

last updated on December 5, 2019

Please answer the following questions in **electronic** form and upload and submit your files to Sakai site, before the due date. Make sure to push the **submit** button.

You can typeset your answer using MS Word (and MS equation for mathematical formulas), or LaTeX, or you may simply handwrite your note and scan and turn it not a **single pdf format file** and upload to Sakai. This homework also has two sets of programs. For questions 2 and 5, submit two separate files (in Python, R , or any other language.) Make sure to provide ample comments so we know what you are doing.

26:198:685, Fall 2019
Homework 5 **Due date:** 11/17/19

1. Express the binary search algorithm iteratively and without recursion.

   See the Wikipedia page for binary search.

2. **You need to write a few simple scripts (in any language) to do this exercise.** Encode the letters of alphabet as follows, where the last character is the blank or white space character. We do not distinguish between capital and lower case letter. To avoid using 0 an 1 (since raising 0 or 1 to any power will generate 0 or 1 again) we start with 2:

a	b	c	d	e	f	g	h	i	j	k	l	m
02	03	04	05	06	07	08	09	10	11	12	13	14

n	o	p	q	r	s	t	u	v	w	x	y	z	" "
15	16	17	18	19	20	21	22	23	24	25	26	27	28

   Alice and Bob are participating in an online community that uses the RSA public key system for secure communications. For Alice define the prime numbers $p_A = 97$, and $q_A = 113$. For Bob define $p_B = 127$, $q_B = 73$, Answer the following questions:

   2a) Compute n_A and the Euler function $\phi(n_A)$, and n_B and $\phi(n_B)$.

   2b) Alice chooses $\ell_A = 73$ for her public key in an RSA system. Compute her private key k_A. Similarly, Bob chooses $\ell_B = 41$. Compute his private key k_B. (You have to use a script to compute k_A, k_B, include your script with this homework.) Indicate for Alice and Bob, which numbers they should keep secret, and which ones they share publicly.

   2c) Suppose Bob wants to send the message "buy google" to Alice. Compute the encrypted message letter by letter, including the blank character. (You need to write a script to achieve this, include your script with this homework.)

   2d) Bob signs his messages with the letters "bob". How would he sign a potential message to Alice? How would Alice ascertain that she has received Bob's signature? (Again, you may need a script for this, include it with this homework.)

   See the attached Python script.

3. Consider a three-pass protocol with an encryption function using XOR:

$$S_k(M) = k \oplus M.$$

3a) Show that the XOR encryption function has the commutative property required for the three pass protocol.

The XOR operation is both commutative and associative, that is $a \oplus b = b \oplus a$, and $a \oplus (b \oplus c) = (a \oplus b) \oplus c$. Also, the encryption and decryption functions with XOR and key k are the identical: $(a \oplus k) \oplus k = a \oplus (k \oplus k) = a \oplus 0 = a$. So, if $S_k(a) = a \oplus k$, we have $P_k(a) = a \oplus k$. Therefore, for example,

$$S_k\Big(P_\ell\big(P_k(M)\big)\Big) = M \oplus k \oplus \ell \oplus k = P_\ell\Big(S_k\big(P_k(M)\big)\Big)$$

Any other order works the same way since using XOR is commutative.

3b) Suppose Alice uses encryption key $k_A = 11001100$, and Bob uses the encryption key $k_B = 01110101$. Suppose Alice wishes to send the message M which in binary equals $M = 11001010$. In step by step detail show the messages that Alice and Bob send to each other using their private keys in the three pass protocol so that Bob recovers Alice's message.

In Python, recall that writing '`0b`' if front of a sequence of zeros and ones makes it into a number in base 2. Also, the '`^`' symbol is the XOR operation. So here is what we do in Python:

```
ka=0b11001100
kb=0b01110101
ka,kb
Out[113]: (204, 117)
M=0b11001010
M
Out[115]: 202
print("Alice to Bob: ", M ^ ka)
Alice to Bob:  6
print("Bob to Alice: ", 6 ^ kb)
Bob to Alice:  115
print("Alice to Bob: ", 115 ^ka)
Alice to Bob:  191
print("Bob decrypts Alice's message: ", 191 ^ kb)
Bob decrypts Alice's message:  202
bin(191^kb)
Out[120]: '0b11001010'
```

3c) Is there a mechanism that when Bob receives Alice's message, he knows for sure it is from her, and not somebody else pretending to be her? Discuss your answer.

In general, the answer is no, at least not within the protocol. Since participants have their own private key and have no information about others' keys, they cannot verify where the messages are coming from. David, for example, can easily send a message to Bob, pretending that he is Alice. Bob would not have any mechanism within the three-pass protocol to verify that the message came from Alice.

4. Use heapsort to sort the letters "SEARCHINGEXAMPLE". At each stage show the heap both in tree format and in array format.

5. **You need to write a few simple scripts (in any language) to do this exercise.** Suppose in a company we have a universe U of 4-digit ID numbers, from 0000 to 9999.

   5a) Suppose we allocate a hash table of size m for this company's set of ID numbers. Write a short script called `hashf1(k,A,m)` where the key `k` is hashed by the multiplicative method $h(k) = \lfloor m(kA \mod 1)\rfloor$.

   5b) Write another short script called `hashit(keys,A,m,hf)` which takes an array of `keys` and using the hash function `hf(k,A,m)` and hashes all keys in the `keys` array. This function should return three objects:
   - array or list `h`: where `h[i]` contains $h(i)$ for the hash function $h(\cdot)$, (that is the value i is hashed to),
   - a two-dimensional array or list `T`: the actual hash table where `T[i]` is a list of all keys that are hashed to `i`, (using lists in Python or R or similar languages should be enough; do not directly implement linked lists yourself),
   - array or list `c`: where `c[i]` has the number of keys that were hashed to `i`.

   5c) Generate a random sample of 100 integer-valued keys in the range 0-9999 and save them in the `keys` array. Repetition is allowed

   5d) Now for $m = 10$ and $A = \frac{\sqrt{5}-1}{2}$, run the `hashit` functions and print the resulting `h, T` and `c`. Also, print the number of empty cells in `T`, and the *variance* of lengths of lists (variance of `c`).

   5e) Repeat part 5d but this time use $A = \frac{\pi}{6}$.

   5f) Repeat part 5d but this time use the division method hash function $h(k) = (k \mod m)$ and for $m = 63$. Compare the number of empty cells, the variance in the length of chains for the three hash functions. Which one is preferable based on this experiment?

5g) **Optional extra credit (10 points):** Draw the histogram of length of chains for each of the three hash functions.

See the attached python script.
Looking at the results we see that

- for the multiplicative method with $A = \frac{\sqrt{5}-1}{2}$ only one cell is empty (with my randomly generated set of keys) the variance of lengths of chains is 3.26, and the longest chain contains ten keys,
- for $A = \pi/6$, three cells are empty, the variance of lengths of chains is 3.87, and the longest chain also contains ten keys,
- And for the division method with $m = 63$, there are zero empty cells, but the variance of lengths of chains is 9.46, and the longest chain contains 15 keys.

So based on this experiment, it seems that the multiplicative method with $A = \frac{\sqrt{5}-1}{2}$ gives the best results.