Data Structure and Algorithm

Homework #3

Due: 1:20pm, Thursday, May 16, 2017

TA email: dsa1@csie.ntu.edu.tw

=== Homework submission instructions ===

- For Problem 1-3, please put all your solutions in a PDF file and name it *[StudentID]_hw3.pdf*, *e.g.*, *b05902987_hw3.pdf*. Please also **include your name and student ID on the first page** of the PDF file before submitting it to the online judge (http://140.112.91.212/judge/). You can either type them or write on papers and scan them. If you choose the latter, please make sure that your writing is recognizable and clear enough, otherwise you might receive some penalties. For Problem 4 and 5, they should be submitted individually and will be judged by the online judge (http://140.112.91.212/judge/).

- Discussions with others are encouraged. However, you should write down the solutions in your own words. In addition, for each problem you have to specify the references (the Internet URL you consulted and/or the people you discussed with) on the first page of your solution to that problem.

- For all programming problems, only C99 standard C language programs are accepted. (i.e., C++, or anything else, is not supported by the online judge system)

- For all the problems, up to one day of delay is allowed; however, you will get some penalties according to the following rule (the time will be in seconds):

$$\text{LATE SCORE} = \text{ORIGINAL SCORE} \times (1 - \frac{\text{Delay Time}}{86400})$$

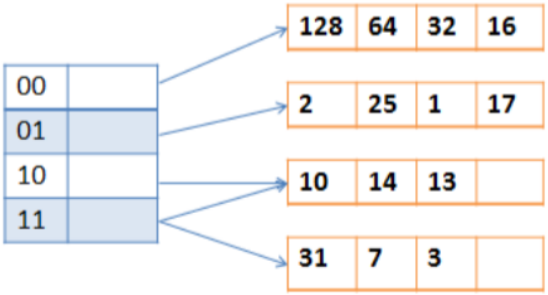
  Penalty will be calculated separately for each of the following three parts (1) problems 1-3 (non-programming problems); (2) problem 4; (3) problem 5.

Problem 1. Hash (25%)

1. (8%) In class, we have learned hash open addressing. Assume the primary hash function is $h_1(k) = k \bmod m$, and the secondary hash function is $h_2(k) = 1 + (k \bmod (m\text{-}1))$. Please insert the keys $\{18, 34, 9, 37, 40, 32, 89\}$ in the given order into a hash table of length $m = 11$. Draw the procedure to simulate how the keys are inserted into the hash table by using open addressing with double hashing and with linear probing in a step-by-step manner after every insertion.

2. (6%)

   (a) (3%) Now we consider dynamic hashing. The figure below is a hash table using dynamic hashing with directory. Each table entry can hold up to 4 keys, and the hash function $h(k, n)$ is defined as the last n bits of binary representation of key k. But TA made some mistakes in drawing the graph. Please help TA fix the problems.



   (b) (3%) After you correct the hash table, please draw a graph to show the content of the hash table after inserting the keys $\{30, 49, 4\}$ into the table.

3. (5%) TAs want to play a remote version (e.g. by FB messenger texting) of rock-paper-scissors, yet TAs are all cunning and not trustworthy. A main problem of the remote version of rock-paper-scissors is someone might throw late (due to message delay). Thus, it's hard to keep the fairness of the game. We suppose that the connection is secured, please design an algorithm such that at least two TAs can play the remote version of rock-paper-scissors impartially without the third party. The algorithm should guarantee the fairness for any two players, no matter which player throws late.
   Hint: Use hash function like SHA-256 causing few collisions. What nice properties does this kind of hash function should have?

4. (6%) Cuckoo hashing is a hashing technique which guarantees O(1) worst case lookup time. It use 2 tables T_1 and T_2 of same size and 2 hash functions h_1 and h_2. To insert an element x, start by inserting into T_1 at position $h_1(x)$. If a collision happens, the stored element y is displaced to T_2 at position $h_2(y)$. In case of another collision, the element currently in T_2 at position $h_2(y)$ is again evicted and inserted to T_1 using h_1. Repeat this process, until all elements stabilize. In some cases, the process would never stop. If that happens, perform a rehash by choosing a new h_1 and new h_2 and inserting all elements back into the tables.

(a) (3%) Given 2 tables of size 7 each and 2 hash functions $h_1(k) = k \mod 7$ and $h_2(k) = \lfloor \frac{k}{7} \rfloor \mod 7$. Insert elements $[6, 31, 2, 41, 30, 45, 44]$ in the given order. Draw the 2 hash tables after each insertion.

(b) (3%) Find another element which makes the insertion leads to infinite sequence of displacements. Note that the $(h_1(k), h_2(k))$ pair for each element should be unique. Write down the element you insert and all the elements included in this sequence of displacements.

Problem 2. Heap (15%)

Given a sequence of numbers:[3, 5, 2, 6, 7, 8, 1, 10, 4, 9]. Please answer question 1, 2, 3.

1. (3%) Please draw the corresponding Max Heap of numbers shown above and write down the in-order number sequence. (The insertions are performed with the exact order of the sequence.)

2. (3%) Please draw the corresponding Min Heap of numbers shown above and write down the in-order number sequence. (The insertions are performed with the exact order of the sequence.)

3. (3%) Given a query q, please design an algorithm for finding all the elements in the min heap that are smaller or equal to q with time-complexity O(k), where k is the amount of qualified elements. For example, given query q=4, the algorithm outputs [1, 4, 2, 3] and k = 4. Noted that the output could be unsorted. Write down your algorithm in pseudo code and prove that the time-complexity is indeed O(k).

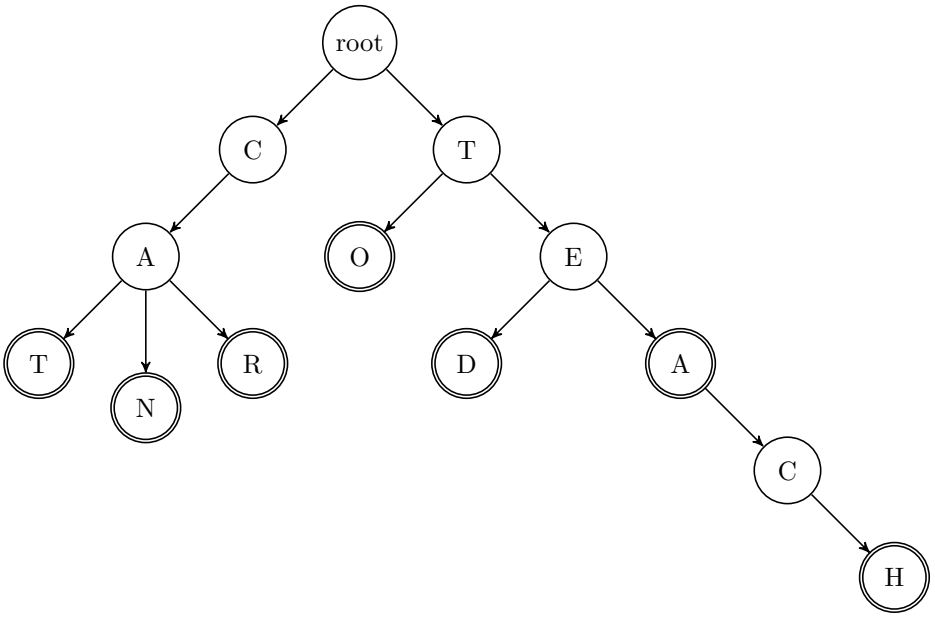
4. (6%) A binary heap is usually stored in an array for practical reasons. Please propose a method to "heapify" an array with time complexity O(N), where N is the amount of elements in the heap. Explain how the index conversion implicitly perceives the relation of a node and its children, and prove the time complexity of your method is O(N). (Hint: The position of index 1 stores the root of the heap.)

Problem 3. Trie Trie See (20%)

You have learned much about trees. Now it's time for tries! A *trie* is a special kind of tree that satisfies the following:

- Each non-root node contains a character (or symbol) from an alphabet Σ. Typically $\Sigma = \{a, b, \ldots, z\}$ is the set of lower-case English letters.
- The characters along the paths from the root to any leaf node (and possibly some internal nodes) form a *word*. These strings are said to be generated by the trie. We denote these *end-of-word* nodes with double circles.

The following is an example of trie which generates cat, can, car, to, ted, tea, teach.



1. (5%) Please draw a trie that generates the following words. Use double circles to represent end-of-word nodes in the trie.

   apple, banana, app, base, basket

2. (5%) Assume all the alphabet are lower case letters(a-z). Prove that the following trie operations can be done with the time complexity $O(N)$, where `N` is the length of `word`:

```
1 void insert(char word[], int N); // insert word into the trie
2 void delete(char word[], int N); // delete word from the trie
3 int query(char word[], int N); // check if word is in the trie
```

   The structure of a trie node is defined below:

```
1 struct Node {
2     struct Node *children[26];
3     int is_word; // Change to 1 for insert(), 0 for delete()
4     int tag;     // may be useful for prob 3 & 4
5 };
```

3. (5%) Given a group of N non-empty words W_i where $1 \leq i \leq N$ and Q queries "S is a prefix of how many words" or "S is a suffix of how many words", where S is a (possibly different) string in each query.

   Please give an algorithm to preprocess the N strings in $O(\sum_{i=1}^{N} |W_i|)$ time and answer each query respectively in $O(|S|)$ time. Note that you can **not** store all the queries and answer them in $O(Q|S|)$ time.

   Hint: you can construct multiple tries.

4. (5%) We introduce the *prefix game*. First, a list of N non-empty words are given. During the game, two players build a word together, which is initially empty. The players move in turns. In each turn, the player add a single letter at the end of the word, and the resulting word must be the prefix of at least one word from the list. A player loses if he cannot move anymore in his/her turn.

   We want to find out whether any player has a winning strategy. (That means he/she can always win no matter how the other player moves). Please give an algorithm to determine whether the first player, the second player, or neither of them has a winning strategy.

In problem 2, you need to *analyze* the algorithm of the functions.

In problem 3 and 4, we highly recommend you to write down the pseudo code and explanation for your algorithm. If you only give one of them, please make sure it's written clear enough. Otherwise, these parts may get large penalty.

Problem 4. Machine Manager (Programming problem) (20%)

Eddy is a magager who is responsible for several machines. Every morning, all the machines are inactive initially. Then, several requests will be sent to Eddy. Eddy should follow the request and do some response. Exactly, there are two types of request:

- `Activate` s : Eddy should make machine named s active.
- `Check` s : Eddy should check whether machine named s is active now.

However, there are too many requests sent to Eddy. He can't handle them at all. Therefore, Eddy asks you to help him writing a program to do the business.

Actually, there are indeed **too many** requests. Therefore, the request will be given to you in a special way. **Please read the input section carefully.**

Input Format

There will be totally Q requests given to you in following way.

The first line contains seven integers $Q, L, T, a_0, b_0, c_0, d_0$. Following T lines each contains five integers q_j, a_j, b_j, c_j, d_j.

Then, you can generate exact requests as following code:

```
int a, b, c, d;
int c_seed = 0, s_seed = 0;
char cmd, *s;
void next_request(){
  c_seed = (c_seed * a + b) & 255;
  cmd = "AC"[c_seed & 1];
  // don't forget to allocate memory for s
  for( int i = 0 ; i < L ; i ++ ){
    s_seed = (s_seed * c + d) & 1023;
    s[i] = (s_seed >> 3) & 63;
  }
}
```

`cmd` being `'A'` indicate the request is in the type of `Activate`. `cmd` being `'C'` indicate the request is in the type of `Check`. Before first request, you should set $a = a_0, b = b_0, c = c_0, d = d_0, c\_seed = 0, s\_seed = 0$. Then, for each q_j, just before q_j-th request, you should reset $a = a_j, b = b_j, c = c_j, d = d_j, c\_seed = 0, s\_seed = 0$. It's guaranteed that q_j is given in the increasing order.

It's recommended that just copy the code piece above to make sure generating correct requests.

Input Constraint

It is guaranteed that

- $1 \leq Q \leq 10^8$
- $1 \leq L \leq 5$
- $0 \leq T \leq 10^4$
- $0 \leq a_i, b_i, c_i, d_i < 2^{16}$
- $1 < q_1 < q_2 < \ldots < q_T \leq Q$
- For 40% points, $L \leq 3$

Output format

Please also read this output section carefully.

Although you might need to check the specified machine active or not when being requested, you don't need to response it everytime. But you need to tell Eddy following two special value to persuade Eddy that you are actually helping him in exactly correct way.

- S: summation of all the check request number if the machine being checked is active then.
- X: xor sum of all the check request number if the machine being checked is active then.

You should output one line with S and X separated by a space.

For example, if for 2-nd request, the machine is active, 4-th request, the machine is inactive, 7-th request, the machine is active. Then, $S = 2 + 7 = 9, X = 2 \oplus 7 = 5$, where $\oplus$ is xor which is '^' in C.

Sample Input

```
3 1 1 0 0 0 384
2 1 1 0 384
```

Sample Output

```
2 2
```

Hint

For sample input :

	c_seed	s_seed	a	b	c	d	cmd	s	request
before 1-st request	0	0	0	0	0	384			
after 1-st request	0	384	0	0	0	384	'A'	"0"	`Activate` machine named "0"
before 2-nd request	0	0	1	1	0	384			
after 2-nd request	1	384	1	1	0	384	'C'	"0"	`Check` machine named "0"
before 3-rd request	1	384	1	1	0	384			
after 3-rd request	2	384	1	1	0	384	'A'	"0"	`Activate` machine named "0"

Problem 5. Yea, I'm a router. (Programming problem) (20%)

HsinHsin is responsible for managing network connections for the department. Now he wants to design a firewall, trying to prevent malicious access by allowing verified IPs only. However, he doesn't feel like coding himself, so he turned to you, a reliable friend, for assistance.

You would receive from HsinHsin a series of network masks, which indicate the sets of addresses verified and allowed. Then, for each of the IPs requesting connection, you would have to tell HsinHsin whether he should accept and establish a connection or not.

A network mask, a.k.a., subnet mask, will be given in the form as:

$$A.B.C.D/N$$

, where A, B, C and D are integers within the range of $[0, 255]$, while N is an integer within the range of $[1, 32]$. The part preceding the forward slash, $A.B.C.D$, is an IPv4 address, and N is the length of the mask. (You may consult **this wiki page** for more detailed explanation of the format.) For example, if HsinHsin gives you this mask: 123.213.132.231/20, which is a valid network mask, it means every IP whose binary form starts with 01111011110101011000 is supposed to be accepted, since it shares the same first 20 bits with the given IP address, 123.213.132.231.

Input Format

The first line contains an integer, T, the number of cases.
For each of the test cases, the first line contains two integers, M and N, the number of network masks and the numebr of IP addresses, respectively. M lines follow, each representing a network mask, in the format described above. Afterwards, N IPv4 addresses are given with each of them in one line, *i.e.*, separated by new-line symbols.

Input Constraint

It is guaranteed that

- $1 \leq T \leq 100$
- $1 \leq M_i \leq 2000000$ and $1 \leq N_i \leq 3000000$ for all test cases.
- All network masks and IPv4 addresses would be valid.

Output format

For the i-th of the T test cases, please output N_i lines.
The j-th line of the total N_i ones will be either `TRUE` if the j-th of the given IPv4 address should be accepted, or `FALSE` otherwise.

Sample Input

```
2
1 3
140.112.8.8/19
140.112.30.42
140.112.30.43
140.118.31.215
1 1
192.168.0.0/16
192.168.123.100
```

Sample Output

```
TRUE
TRUE
FALSE
TRUE
```

Sample I/O Explanation

The first test case consists of 1 network mask and 3 IPv4 addresses.
The network mask, 140.112.8.8/19, would allow all IPs of $140.112.[0-31].*$.
Therefore, the first two given IPv4 addresses are allowed, while the last is not.
In the second test case, the only network mask, 192.168.0.0/16 actually allows every IPv4 address starting with 192.168.

Hint

1. You might want to convert the decimal representation into binary one for further checking or processing.
2. If you couldn't get full scores on the judge system due to TLE problem, you might want to check out the **Problem 3** in this homework set.