Data Structure and Algorithm, Spring 2017

Midterm Examination

Date: Tuesday, April 18, 2017

Time: 13:20-16:20pm (180 minutes)

140 points

Non-Programming problems

***Problem* 1.** Prove $\log_2 n! = \Theta(n \log n)$. (15%)

***Problem* 2.** Linked list with a loop at the end. (25%)

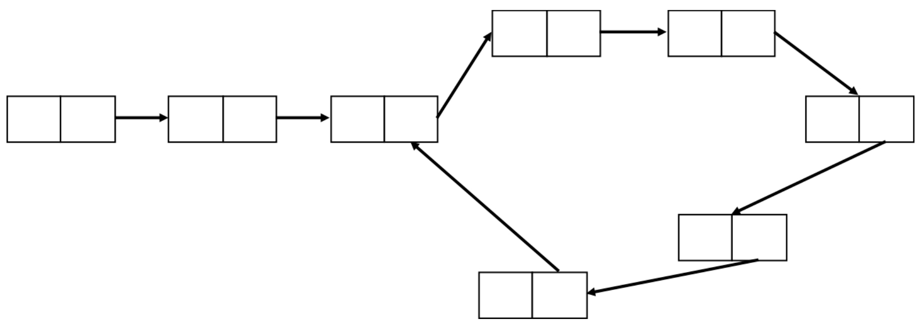
Figure 1 show the example of two different types of singly linked lists: one with a loop at the end and one with no loop (i.e., regular chain linked list). When you traverse the linked list with a loop at the end from the beginning, eventually you will encounter a list node where you have visited before. However if you traverse the linked list with no loop, during the traversal all the nodes you visited will be unique.

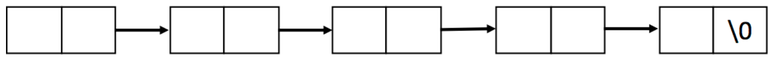
a. Please complete the C function below to determine whether the given linked list has a loop at the end. If so, return 1; otherwise, return 0. Your function should have an $O(n)$ worst-case running time. (Hint: use two pointers to traverse the linked lists at different speed.) (10%)

```c
struct list_node{
        void *data;
        struct list_node *next;
};

int is_linked_list_with_loop(struct list_node *head) {
        // head points to the first list node.
        // fill the blank here
}
```

b. Analyze and show that your implementation of is_linked_list_with_loop() runs in $O(n)$ worst-case time. (5%)


(a) With a loop



(b) With no loop

Figure 1: Linked lists with and with no loop at the end.

c. Implement the C function chain_length() below to find the number of list nodes *NOT* in the loop. If there is no loop in the given linked list, then this is simply the total number of nodes in the linked list. Your function should still run in $O(n)$-time. (Hint: think about how to determine the number of list nodes in the loop, if there is any, based on your implementation of is_linked_list_with_loop()) (10%)

```c
int chain_length(struct list_node *head) {
        // head points to the first list node.
        // fill the blank here
}
```

Problem 3. Space-efficient doubly linked list. (15%)

XOR, or exclusive OR, is logical operation that outputs true only when the two inputs differ. We can perform so called *bitwise XOR* (represented by the symbol $\wedge$) on two binary numbers of equal length, where on each pair of corresponding bits XOR is performed. For example, $0011 \wedge 1010 = 1001$.

Bitwise XOR has an interesting property. If $C = A \wedge B$, then $C \wedge A = B$ and $C \wedge B = A$. We can in fact utilize the property to use just one pointer per list node in a doubly linked list, as shown below. Note that a pointer is just an integer number indicating a memory address, and thus bitwise XOR is applicable.

```c
struct db_list_node{
        void *data;
```

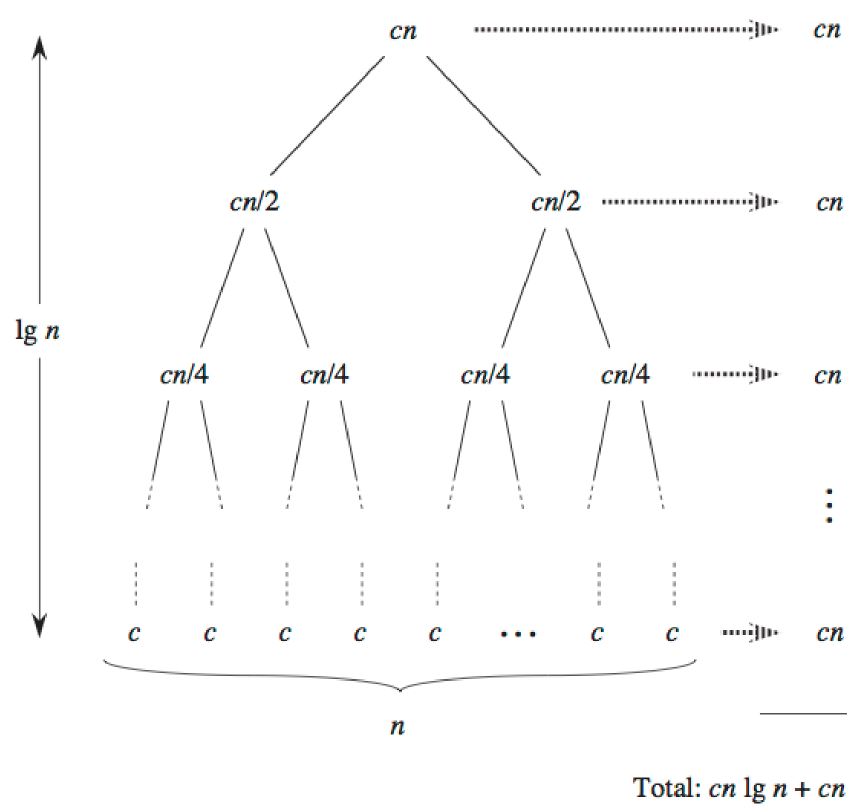


Figure 2: Using the recursion tree to analyze the standard merge sort algorithm, we can show that the worst-case running time is $cn \log n + cn = O(n \log n)$.

```c
        struct db_list_node * xor_pointer;
};
```

a. What values should be stored in the xor_pointer for the first list node, the last list node, and a list node in the middle, respectively? (6%)

b. Complete the C function below to count the number of list nodes in the given list. (9%)

```c
int number_of_node(struct db_list_node *head) {
        // head points to the first node.
        // Fill the blank here.
}
```

Problem 4. Combination of merge sort and insertion sort. (25%)

Although merge sort runs in $O(n \log n)$ worst-case time and insertion sort runs in $O(n^2)$ worst-case time, the small constant factors in insertion sort can make it faster in practice for small problem sizes on many machines. Therefore, it makes sense to develop a hybrid sorting algorithm by using insertion sort within merge sort when the subproblem become small enough.

In the following, consider the modified merge sort algorithm where n/k sublists of length k are sorted using insertion sort and then merged using the standard merging mechanism, where k is given based on the machine used.

a. Show that the insertion sort can sort the n/k sublists, each of length k, in $O(nk)$ worst-case time. (5%)

b. Figure 2 uses the recursion tree to calculate the total running time of the original merge sort algorithm, which can then be used to show that it has $O(n \log n)$ worst-case running time. Please use the same method to show that the hybrid algorithm can complete the sort in $O(nk + n \log(n/k))$ worst-case time. (10%)

c. Given a machine, in practice how do you determine the optimal k value? (10%)

Problem 5. Binary Search Tree. (20%)

a. Suppose that we have numbers between 1 and 1000 in a binary search tree, and we want to search for the number 363 in it. If each of the following sequences represents the order of nodes that we traverse, starting from the root, which one(s) is invalid? Please explain why. (5%)

   (1) 2, 252, 401, 398, 330, 344, 397, 363.

   (2) 924, 220, 911, 244, 898, 258, 362, 363.

   (3) 925, 202, 911, 240, 912, 245, 363.

   (4) 2, 399, 387, 219, 266, 382, 381, 278, 363.

   (5) 935, 278, 347, 621, 299, 392, 358, 363.

b. Given a node x in a binary search tree, the *successor* of x is the node with the smallest key greater than x's key. Similarly, the *predecessor* of x is the node with the largest key less than x's key. Prove that if a node in a binary search tree has two children, then its successor has no left child and its predecessor has no right child. (5%)

c. Assume that the binary search tree is implemented with the C structure definition below. Please complete the C function bst_successor() below to find the successor of x. (10%)

```c
struct bst_node{
        int key;
        void *data;
        struct bst_node *p, *left , *right;
        /*
          the above three pointers point to the node's
          parent , left child , and right child.
        */
};

struct bst_node *bst_successor(struct bst_node* x) {
        // fill the blank here
}
```

Programming problems

Submit your answer to `http://140.112.91.212/judge`

Problem 6. String compression. (20%)

Given two strings, A and B, we would like to compress them into one string S such that A is a prefix of S and B is a suffix of S.

For example, if A is *abcdbc* and B is *bcdbce*, then *abcdbcdbce* is a valid choice of S. However, *abcdbce* is also a valid choice of S, but has smaller length.

Given string A and B, you can find an arbitrary number of S that satisifies the above rules, and $\hat{S}$ is the one with minimum length among them. In this problem, we ask you to find the length of $\hat{S}$.

Input format

The first line contains one positive integer, T, the number of test cases, and T test case(s) follow. For each test case, there will be two lines. The first line contains the string A and the second line contains the string B.

Input constraint

It is guaranteed that

- String A and B are not empty and consist of lowercase English letters only.
- The summation of the lengths of all the $2 \times T$ strings is less than or equal to 1000000.
- 30% of the credit of this problem corresponds to test cases where the summation of the lengths of all the $2 \times T$ strings is less than or equal to 20000.

Output format

For each test case, you should output the length of the minimum-length string $\hat{S}$.

Sample input

```
3
abcdbc
bcdbce
aaa
bbbbb
ababab
ab
```

Sample output

```
7
8
6
```

Hint

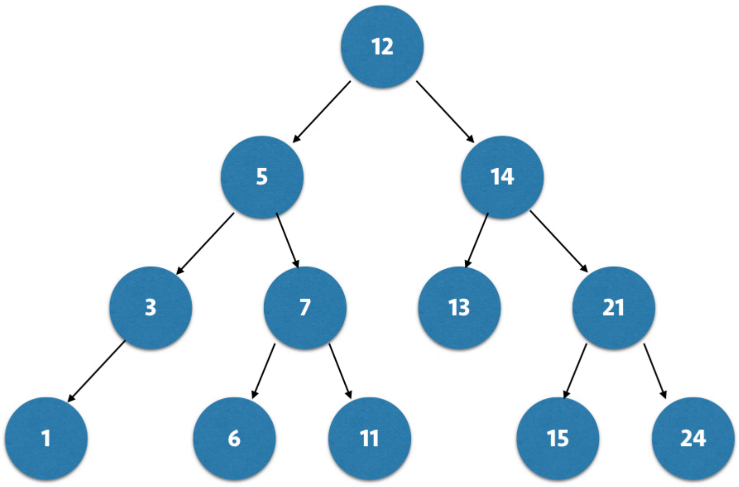
The minimum-length string $\hat{S}$ corresponding to the three sample test cases are *abcdbce*, *aaabbbbb* and *ababab*, respectively.

Problem 7. Post-order Sequence Verification. (20%)

As taught in class, there are 3 common ways of traversing a binary tree: (1) **Pre-order** (2) **In-order**, and (3) **Post-order**. Each of these traversing methods will produce an output sequence.

For example, the output sequences of the binary tree in the following figure are:

- Pre-order: `12 5 3 1 7 6 11 14 13 21 15 24`.
- In-order: `1 3 5 6 7 11 12 13 14 15 21 24`.
- Post-order: `1 3 6 11 7 5 13 15 24 21 14 12`.



In this problem, given multiple sequences, you are required to: (1) determine whether a sequence is a post-order one of a **binary search tree**, and (2) give the height of the tree if it is.

Input Format

The first line contains an integer, T, the number of test cases, and $2 \times T$ lines follow. For each of the test cases, there are 2 lines, the first containing an integer, L_i, which is the number of integers in the post-order sequence. The sequence, S_i, is given in the second line, with L_i integers separated by single spaces, as the example sequences in the above problem description.

Output Format

For each test cases, you need to output one line, *i.e.*, there will be `T` lines in total. Each line will be either a string, `NOT VALID`, if the given sequence is not a post-order one of any binary search tree, or an integer, `N`, the height of the corresponding binary search tree.

Input Constraints

It is guaranteed that:

- $1 \leq T \leq 100$
- $1 \leq L_i \leq 10000$ for `i = 0` to `T-1`.
- All integers in S_i are unique, for `i = 0` to `T-1`.

Sample input

```
3
12
1 3 6 11 7 5 13 15 24 21 14 12
12
1 11 6 3 7 5 13 15 24 21 14 12
6
1 3 6 11 7 5
```

Sample output

```
4
NOT VALID
3
```