Data Structure and Algorithm

Homework #1

Due: 1:20pm, Tuesday, March 21, 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 submit 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 langauge 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) problem 1-3 (non-programming problems); (2) problem 4; (3) problem 5.

Problem 1. Asymptotic Notation (25%)

1.1. (9%) Rank the following five functions by the order of growth; that is, for functions, find a permutation $\langle g_1(n), g_2(n), g_3(n), g_4(n), g_5(n)\rangle$ of these functions such that $g_1(n) = \Omega(g_2(n))$, $g_2(n) = \Omega(g_3(n))$, $g_3 = \Omega(g_4(n))$, $g_4(n) = \Omega(g_5(n))$. Use any tool you prefer to plot all five functions to compare their order of growth.

- $n^{\frac{1}{\ln n}}$
- 2^n
- $n^3 - n$
- $n!$
- $e^{\ln n}$

1.2. (6%) Prove that $n! = \omega(2^n)$ and $n! = o(n^n)$.

1.3. (10%) Let $f(n)$ and $g(n)$ be asymptotically positive functions. Prove or disprove each of the following statements. Note that you need to prove both directions for "if and only if" statements.

(a) (2%) $f(n) = \mathrm{O}(g(n))$ if and only if $g(n) = \Omega(f(n))$

(b) (2%) $f(n) = \Theta(g(n))$ if and only if $f(n) = \mathrm{O}(g(n))$ and $f(n) = \Omega(g(n))$

(c) (3%) $f(n) = \mathrm{O}(g(n))$ if and only if $f(n) \cdot g(n) = \mathrm{O}(g(n)^2)$

(d) (3%) $f(n) = \mathrm{O}(g(n))$ if and only if $(f(n))^2 = \mathrm{O}(g(n)^2)$

Problem 2. Time and Space Complexities (15%)

Consider an **unsorted** integer array A with N distinct elements. All integers in A are between 0 and K. The following considers algorithms which check if there exist pairs of two elements $(A[i], A[j])$, such that $A[i] + A[j] = k$, $i < j$.

2.1. (4%) The two functions below, *Binary_Search* and *Count_Search*, check for the existence of $(A[i], A[j])$. Assume that the *sort*() function is able to sort the array in $O(n \log n)$ and the *malloc*() function takes O(1) time, what are the time complexities of the two functions, respectively? What is the space complexity of *Count_Search*? Which of these two methods is better in your opinion? Please explain.

```
1  Binary_Search(A, N, k) {
2      sort(A);
3      for (i = 0; i < N; i++) {
4          search = k - A[i];
5          left = 0, right = N-1;
6          while (left <= right) {
7              mid = (left + right) / 2;
8              if (A[mid] == search) return true;
9              else if (A[mid] < search) left = mid + 1;
10             else if (A[mid] > search) right = mid - 1;
11         }
12     }
13     return false;
14 }
15
16 Count_Search(A, N, K, k) {
17     B = malloc(max(K, k)+1);
18     for (i = 0; i < N; i++) {
19         if (k > A[i] and B[k - A[i]])
20             return true;
21         B[A[i]] = true;
22     }
23     return false;
24 }
```

2.2.

(a) (3%) Let M be the number of pairs of elements, $(A[i], A[j])$, such that $A[i] + A[j] = k, i < j$. Please design a brute-force algorithm and write down the pseudo code that calculates the value of M with time complexity $O(n^2)$.

(b) (3%) Now consider using the binary search algorithm. How can we improve the running time of the algorithm to $O(n \log n)$? Please write down your pseudo code.

2.3. (5%) Given an integer array A **sorted** in ascending order, please design an algorithm with time complexity $O(n^2)$ that checks if there exist tuples of three distinct integers $(A[i], A[j], A[k])$, $A[i], A[j], A[k] \in A$, such that $A[i] + A[j] + A[k] = m$, $i < j < k$. Please write down the pseudo code and show that its worst case time complexity is indeed $O(n^2)$.

Problem 3. Stack and Queue (20%)

A sequence of n numbers is *stack-valid* if it can be the output of the following procedure:

- There is an initially empty stack S.
- n push and n pop operations are intermixed and performed on S.
- The n push operations push the numbers 1 to n sequentially to S.
- A number is printed when popped out of S.
- The stack is empty after all push/pop operations.

For example, [5, 1, 4, 3, 2] is not stack-valid. On the other hand, [1, 3, 4, 5, 2] is stack-valid, since it can be obtained by performing the following stack operations:

```
PUSH 1
POP
PUSH 2
PUSH 3
POP
PUSH 4
POP
PUSH 5
POP
POP
```

The definition of *queue-valid sequence* is similar, with push/pop replaced with enqueue/dequeue, respectively. For instance, [5, 4, 1, 3, 2] is not queue-valid, while [1, 2, 3, 4, 5] is.

3.1. (4%) Determine whether [3, 2, 4, 1, 5] is stack-valid. If so, write down the exact order of the stack operations that output this sequence (similar to the example shown above). If not, briefly explain why.

For the subproblems 3.2 through 3.5, you can assume there will be only $1, 2, \ldots, n$ in the given sequence, and each number appears exactly once.

3.2. (2%) Given a sequence of n numbers, design an algorithm (pseudo code) to determine whether it is queue-valid. Analyze and give the time complexity of your algorithm with the Big-O notation.

3.3. (6%) Given a sequence of n numbers, design an algorithm (pseudo code) to determine whether it is stack-valid. Analyze and give the time complexity of your algorithm with the Big-O notation.

3.4. (2%) Given a sequence of n numbers, briefly explain how to determine whether it is queue-valid using only O(1) additional space. Moreover, analyze and give the time complexity of your algorithm with the Big-O notation.

3.5. (6%) Given a sequence of n numbers, briefly explain how to determine whether it is stack-valid using only O(1) additional space. Moreover, analyze and give the time complexity of your algorithm with the Big-O notation.

Hint: Let $A = [a_1, a_2, \ldots, a_n]$ *be the given sequence. It is stack-valid if and only if the following holds:* $\forall a_j, a_k \in \{a_m \in A | 1 \leq i < m \leq n \text{ and } a_m < a_i\}$, $a_j > a_k$ *if* $j < k$. *In other words, the subsequence of all smaller-than-*a_i *numbers to the right of* a_i *is in descending order. You can use this fact after you roughly explain why this is true.*

Problem 4. Calculator (Programming problem) (20%)

This problem requires you to implement a program to evaluate arithmetic expressions. However, the input may not be a valid arithmetic expression, in which case your program should report an error.
A valid arithmetic expression is defined as follows:

- Let the set of valid arithmetic expression be **E**.
- Let the set of non-negative integer without leading zeros be **N**.
- Then, we have **E** = **N** ∪ **E**+**E** ∪ **E**-**E** ∪ **E*****E** ∪ (**E**).

For example, these are valid arithmetic expressions:
`"0","1","2+3","4-5","6*7","(8)","(1+2)","(1-2)*3"`
These are not valid arithmetic expressions:
`"0+","()","3(","--","(-)","01"`

Input format

The first line contains one positive integer T indicating that there will be T arithmetic expressions to evaluate. The following T lines each contain an arithmetic expression E.

Input constraint

It is guaranteed that

- Each character of E is one of `"0123456789+-*()"`
- The sum of the length(s) of the given T arithmetic expression(s) is less than or equal to 10^5 characters

Output format

For each arithmetic expression, you should output one line. If it is a valid arithmetic expression, you should output the evaluated value, modulo 1000000007 ($10^9 + 7$). If it is not a valid arithmetic expression, you should output `"invalid"` without the quotation mark.

Sample Input	Sample Output
3 123 (1-2)+3 0+	123 2 invalid

Problem 5. Good String (Programming problem) (20%)

For a string T, T is a *good string* if it contains all 26 of the lowercase English alphabets. For example, "abcdefghijklmnopqrstuvwxyz", "ababbcdefghijklmnopqrstuvwxyz" and "zyxwvutsrqponmlkjihgfedcbaxx" are all *good strings*; however, "azsd" and "abcdefghijklmnopqrstuvwxy" are not.

The *good value* of a string S is defined as the number of distinct integer pairs (i, j) with $i, j \geq 0$ and $i + j < |S|$ such that if we remove the first i character(s) and the last j character(s) from S, the resulted string S' is still a *good string*.

Please write a program to evaluate the *good values* of the given strings.

Note: $|S|$ is the length of string S.

Input format

The first line contains one positive integer K indicating that there will be K given strings to evaluate. The following K lines each contains a string S.

Input constraint

It is guaranteed that

- Each string S is not empty and consists of lowercase letters only.
- The sum of the length(s) of the given K string(s) is less than or equal to 1000000.

Output format

For each string to evaluate, you should output its good value in one line.

Sample input

2
ababbcdefghijklmnopqrstuvwxyz
abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz

Sample output

3
378