An Example of Problem Solving

Lab 8
Finding Third Largest Number

Rung-Bi Lin

International Program in informatics for Bachelor
Yuan Ze University
Nov. 23, 2021

Yuan Ze University Study Regulations

Amended by the 1nd School Council Meeting of the 2015 School Year, on Novermber 4, 2015

Approved for future reference by Ministry of Education in Letter Tai-Kao-2-Tzu No. 1040177967, on December 24, 2015

Chapter 4 Absence, Truancy and Leave

26. Students who have been granted an approval for leave will be regarded as absence. Any absence other than those for which leave or approval for leave has been granted will be regarded as truancy. Rules for handling absence and truancy are as follows.
    - 26.1 Truancy for one hour is considered as absence for two hours. Students with sixty hours of truancy shall be ordered to withdraw from the university.
    - 26.2 If the number of hours of leave of absence in the course of a semester is one-third of the total class hours, the teacher may notify the Office of Academic Affairs to disqualify the student from taking the final examination and the grade of that particular course will be considered zero.
    - 26.3 If the number of hours of leave of absence of a semester is one-third of the total hours of taken courses, the student shall be ordered to apply for suspension.

Finding Second Largest Numbers

Problem Description

Given a sequence of *n* integers $a_1, a_2, a_3, \ldots, a_n$, find the second largest integers and the index of its last occurrence. Here, *j* is the index of a_j and $1 < n < 101$. Each integer is representable in 32 bits. If there is no second largest integer, set the second largest integer to the largest integer and set the index of its last occurrence to the index of the last occurrence of the largest integer.

Example 1: -1 2 2 4 11 -1 6 3 5 10 100 5 100 10 11 11 → 11 16

Example 2: 1 1 1 → 1 3

Input Format

The first line gives the number of test cases. It is then followed by the input data for each test case. The input for each test case has $n + 1$ integers which are in order of n, a_1, a_2, ..., a_n, where *n* gives the number of $a_i's$. These $n + 1$ integers are placed on a line. Integers are separated by whitespace(s). This is repeated for every test case.

Output Format

The output for a test case takes a line that contains two numbers. Each output line begins with a #, then a whitespace, and two numbers. The first number is the second largest integer and the second number is its index.

Problem Analysis

- From $1 < n < 101$, we know there are at least two integers. How about if $1 \leq n < 101$? Then, there is at least one integer. This is more general case than $1 < n < 101$. So if a program works for $1 \leq n < 101$, it will certainly work for $1 < n < 101$.
- Since the numbers can be same, there are two consequences:
  1. There could not have a second largest number. Namely, the second largest number should be set to the largest number according to the problem description.
  2. There could be more than one second largest number.
- These two consequences are key to solving the problem.

Reasoning

- Let's consider n ≥ 1. How about n==1? . Namely, we have only one number a_1. What is the solution? The solution is that the second largest number is a_1 and the index of its last occurrence is 1.
- How about n ==2? Namely, we have a_1 and a_2. The solution is that
  1. The second largest number is a_2 and the index of its last occurrence is 2 if $a_1 > a_2$.
  2. The second largest number is a_1 and the index of its last occurrence is 1 if $a_2 > a_1$
  3. The second largest number is a_2 and the index of its last occurrence is 2 if $a_1 == a_2$.
- How about n==3? It becomes more complex. Hence, we need a general algorithm to solve the problem. The key to the solution is making comparisons among numbers.

A Solution

- Initialization:

  Since there are at least two numbers, we can do numL = first number; inxL = 1; numSL = first number; inxSL = 1;

- Repeat reading the next number **numNxt** with index **inxNxt** one by one. Then, based on the first consequence, we have two situations, either **numL == numSL** or **numL > numSL**.

  If (numL == numSL)

  { 1. numNxt > numL → numL = numNxt; inxL = inxNxt;

  2. numNxt == numL → inxL = inxNxt; inxSL = inxNxt;

  3. numNxt < numL → numSL = numNxt; inxSL = inxNxt; }

  else { // numL > numSL

  1. numNxt > numL → numSL = numL; inxSL = inxSL;
     numL = numNxt; inxL = inxNxt;

  2. numNxt == numL → inxL = inxNxt; // Important

  3. numL > numNxt ≥ numSL → numSL = numNxt; inxSL = inxNxt;

  }

What Does A Solution Give?

- numL: largest number
- inxL: index of the last occurrence of the largest number.
- numSL: second largest number
- inxSL: index of the last occurrence of the second largest number.

Another Solution - Yes

- Store the numbers into an array. Sort the numbers in the descending (or ascending order). Then, find the second largest number and the index of its last occurrence.

Lab 8: Third Largest Number

Instead of finding the second largest number, the problem is now finding the third largest number and the index of its last occurrence from a given sequence of n integers $a_1, a_2, a_3, \ldots, a_n$, where $1 \leq n \leq 100$. The input and output format are the same as that of the problem of finding the second largest number.

Sample Input & Output

Sample Input	Sample Output
16	
3 1 1 1	# 1 3
3 1 1 2	# 1 2
3 1 2 2	# 1 1
5 1 1 2 2 2	# 1 2
1 1	# 1 1
2 1 1	# 1 2
2 1 2	# 1 1
2 2 1	# 1 2
3 1 2 3	# 1 1
3 3 2 1	# 1 3
16 -1 2 2 4 11 -1 6 3 5 10 100 5 100 10 11 11	# 10 14
5 73 11 5 -2100000000 2100000000	# 11 2
12 2 -4 7 9 3 9 21 4 2 21 9 7	# 7 12
20 -1 2 3 4 5 6 7 8 9 10 10 9 8 7 6 5 4 3 2 1	# 8 13
10 1 1 1 2 2 2 2 1 1 2	# 1 9
6 2 2 1 1 2 3	# 1 4