The Best Peak Shape

Author Name:

Date: 2019-04-22

Chapter 1: Introduction

1.1 Problem description

In this project, it is required to find the longest subsequence in a sequence of numbers, such that there is a number in the subsequence, the number on the left side of which is smaller than it, and the number on the right side is larger than it. Such a subsequence is called a "peak shape".

Now give the N input numbers sorted by their index, some of them can be deleted to keep the remaining numbers in a peak shape. The best peak shape is the longest subsequence that forms the peak shape. If there is a tie, chose the one that the difference between the length of the left side and the right side much bigger.

input:

- `N` is the size of the input integers
- The next N integers are inputted in the required order as the sequence ordered by their index.

1.2 Background of the algorithms

Dynamic programming is both a mathematical optimization method and a computer programming method. Problems that can be done by dynamic programming refers to simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner.

Chapter 2: Algorithm Specification

2.1 Main Data Structure

Here, it uses two one dimension array `increase` `decrease` to record the information. For example `increase[i]` suggests how many numbers in the sequence that is a subsequence of the origin array in increasing order.

2.2 Main

In the main function, firstly, it calculate two arrays `increase` and `decrease`, which calculate the longest increasing or decreasing sequence ended or started by index. And then, traverse each number and let it act as peak number and calculate the length of the sequence by add the length of increasing sequence ended by it and the length of decreasing sequence started by it.

```
int main(void) {
    int n , m;
    int maximum = 0 , index = 0 , delta = INF;
    read the sequence;
    calculate the longest increasing sequence ended by index;
    record in increase[MAXSIZE];
    calculate the longest decreasing sequence started by index;
    record in decrease[MAXSIZE];
    for (every index i in the sequence) {
```

```
        if (one or more number on the left and right side of i
            && the sum of the two sides is no less than maximum) {
            if (the sum of the two sides equal to maximum
               && the difference between two sides is larger than delta)
                continue;
            update maximum with new sum;
            update the index of peak number;
            update the delta by new diffenrence;
        }
    }
    output the final maximum,index and delta;
}
```

2.3 Decrease/Increase array

These two functions are used to calculate the two arrays that suggest the length of longest increasing or decreasing sequence ended or started by index.

```
/*calculate the length of the longest incresing subsequence*/
void lis(vector<int>& nums , int *inc) {
    inc[0] = 1;  //init
    for (int i = 1; i < nums.size(); i++) {
        init the length of the sequence ended by i is 1;
        for (int j = 0; j < i; j++) {
            if (nums[j] < the end number of the sequence
                && add this number the sequence can be longer)
                update the length of the sequence ended by i;
        }
    }
}
/*calculate the length of the longest incresing subsequence*/
void lds(vector<int>& nums , int *dec) {
    dec[nums.size() - 1] = 1;  //init
    for (int i = nums.size() - 2; i >= 0; i--) {
        init the length of the sequence started by i is 1;
        for (int j = nums.size() - 1; j > i; j--) {
            if (nums[j] < the start number of the sequence
                && add this number the sequence can be longer)  //condition
                dupdate the length of the sequence started by i;
        }
    }
}
```

Chapter 3: Testing Results

3.1 How do we test

1. Go to PTA and find the same problem, use it to judge our correctness.

测试点	结果	耗时	内存
0	答案正确	3 ms	384 KB
1	答案正确	3 ms	512 KB
2	答案正确	93 ms	544 KB
3	答案正确	88 ms	512 KB
4	答案正确	3 ms	384 KB
5	答案正确	389 ms	512 KB
6	答案正确	3 ms	576 KB

1. Set some special cases and test its correctness.
2. Write a test program to deal with large N, the correct answer can't be calculated by ourself so we ask some friend to run their program and our input to get their answer. And we use their answer as expected answer to see whether our output is correct. Believe our correctness since we find several friends.

3.2 Detailed information of test cases

1. Sample1

Test case 1	
Purpose	Sample1 in the description of problem
Input	20 1 3 0 8 5 -2 29 20 20 4 10 4 7 25 18 6 17 16 2 -1
Expected output	10 14 25
Output	10 14 25
State	PASS

2. Sample2

Test case 2	
Purpose	Sample2 in the description of problem
Input	5 -1 3 8 10 20
Expected output	No peak shape
Output	No peak shape
State	PASS

3. If there is a tie, then the most symmetric one win

Test case 3	
Purpose	If there is a tie, then the most symmetric one win
Input	10 1 3 5 0 -1 2 4 6 0 9
Expected output	5 3 5
Output	5 3 5
State	PASS

4. Increasing sequence

Test case 3	
Purpose	Increasing sequence so peak doesn't exist.
Input	5 1 2 3 4 5
Expected output	No peak shape
Output	No peak shape
State	PASS

5. Decreasing sequence

Test case 3	
Purpose	Decreasing sequence so peak doesn't exist.
Input	8 9 8 7 2 -1 -4 -10 -12
Expected output	No peak shape
Output	No peak shape
State	PASS

6. Minimum N(N=1)

Test case 3	
Purpose	Minimum N (N=1)
Input	1 1
Expected output	No peak shape
Output	No peak shape
State	PASS

7. Minimum N which can find a successful case(N=3)

Test case 3	
Purpose	Minimum N which can find a successful case (N=3)
Input	3 1 3 0
Expected output	3 2 3
Output	3 2 3
State	PASS

8. Maximum N (N=10000)

Test case 3	
Purpose	Maximum N (N=10000)
Input	Too big, see test_10000.txt please.
Expected output	269 4758 4841
Output	269 4758 4841
State	PASS

9.Big input N and randomized sequence(N=3000)

Test case 3	
Purpose	Big input N and randomized sequence (N=3000)
Input	Too big, see test_3000.txt please.
Expected output	143 1126 4887
Output	143 1126 4887
State	PASS

10. Big input N and randomized sequence(N=5000)

Test case 3	
Purpose	Big input N and randomized sequence(N=5000)
Input	Too big, see test_5000.txt please.
Expected output	182 2480 4848
Output	182 2480 4848
State	PASS

3.3 Test table

No.	Purpose	State
1	Sample1	PASS
2	Sample2	PASS
3	Tie and the most symmetric one win	PASS
4	Increasing sequence	PASS
5	Decreasing sequence	PASS
6	Minimum N (N=1)	PASS
7	Minimum N which can find a peak shape	PASS
8	Maximum N (N=10000)	PASS
9	Big input N(N=3000)	PASS
10	Big input N(N=5000)	PASS

Chapter 4: Analysis and Comments

The whole procedure and cost of time of our program is as follows:

1. Read the input sequence

   `O(1)*N=O(N)`;

2. Calculate longest increasing subsequence

   In this function, we use a nested circle to implement. In the inner circle, it runs several operations and the cost of time equals `O(1)`.

   So `O(N) * O(N) * O(1) =O(N^2)`;

3. Calculate longest decreasing subsequence

   In this function, we also use a nested circle to implement.In the inner circle, it runs several operations and the cost of time equals `O(1)`.

   So `O(N) * O(N) * O(1) =O(N^2)`;

4. Find the maximum peak length with the least delta

   We traverse the sequence to find the maximum peak length. In every circle we only do some comparisons and assignments,so it costs `O(1)`.

   So `O(1)*N=O(N)`;

In total, the time complexity of our project is

$$O(N) + O(N^2) + O(N^2) + O(N) = O(N^2)$$

4.2 Space complexities

We use C++ to implement our program.

In our project, we use vector to allocate space dynamically according to the size of the input. Besides we also use two arrays whose size is defined as MAXSIZE(10000).Also, we have several variables used to help us complete the project.

So in conclusion, the space complexities is `O(1)+O(N)=O(N)`.

4.3 Comments

Our implementation of this project costs O(N^2) time complexities, so we may find that its performance confronted with big data seems a bit imperfect, but it still can output the correct answer.

After deep thoughts about this problem, we find a better solution whose time complexity is O(NlogN). The general algorithm doesn't change much, but we cut the time complexity down in the procedure of finding the longest increasing or decreasing sequence.

In effect, the problem can be seen as a LIS problem(Longest increasing sequence). And we can use binary search to better the time complexity.

Suppose that the sequence we want to find the longest increasing subsequence is a [n], and the increasing subsequence to be found is put into array B[].

step1: When the first element of array A is traversed, the element is put into B[]. The elements traversed later are compared with those already in B.

step2: If it is larger than every element in B, the element is inserted into the tail of B, and the length of B is increased by 1;

step3: If it is smaller than the last element in the B array, it is necessary to use binary search to find the first element larger than the current element, and then replace it.

In this way, we got a O(NlogN) algorithm.

Declaration

We hereby declare that all the work done in this project titled "The Best Peak Shape" is of our independent effort as a group.