ADS Project-4
The Best Peak Shape

Author Names：

Programmer:彭子帆

Tester:陈宇威

Reporter:王钟毓

Date: 2019-04-19

I. Introduction

1.1 Problem Description

The Best Peak is a very classic max-increasing-substring which can be easily solved by DP method.

After input a series of string, we find the longest sub-sequence that forms a peak shape. If there is a tie (a tie refers to the same length of sub-sequence), the most symmetric one will be chosen. The peak shape means the whole string can be divided into three parts --- the rising edge, peak and falling edge. In mathematical formula, there exists an index i $(1 < i < L)$ such that $D1 < \ldots < D_{i-1} < Di > Di+1 > \ldots > DL$.

The Best Peak Shape problem is very common in science, such as physics, data analysis, image processing and so on. Today, we will introduce a quite simple method towards this problem.

II. Algorithm Specification

The whole program is rather simple to be designed. First, we insert the size of input and a series of numbers. We can interpret the whole string as the string [a,i] and string[i, b]. Where the string[a, i] is the left increasing substring of the whole peak shape. And string[i, b] is the decreasing substring of the input. However, we can change its sequence and see it as the string[b, i], then it becomes the increase substring as well. Thus, the whole problem can be simplified into solving two longest increasing substring problems.

During longest-increasing-substring problems, a state transition equation can easily be obtained that $\mathrm{length}[\mathrm{i}] = \max(length[i], length[i-1]+1)$. And we can just use two loops to traverse through all the substring of the whole string to get the correct answer which can be vividly seen in the pseudo-code.

2.1 pseudo-code

The pseudo-code of solving max_increasing_substring function is as follows.

```
int main()
{
    input L and string;
    for(i from 0 to L) // Loop the end of substring
    {
        Initialize length_longest_substring[i]
        for(j from 0 to i) // Loop the start of substring
        {
            if(string(i) > string (j))
                // state transition equation
                length = MAX(length,length_longest_substring[j] + 1)

        }
    }
    return length;
}
```

2.2 Postscripts

There may be some differences between the pseudo-code given above and the actual program. But the whole algorithm is more or less the same.

III. Testing Results

3.1.1 comprehensive cases

```
20
1  3  0  8  5  -2  29  20  20  4  10  4  7  25  18  6  17  16  2  -1
Results:  10  14  25
```

```
20
527  -620  -9455  -1272  -2674  -8815  -5328  -9025  -5597  5177  -6431  -3044  1533
5908  -6572  -4645  3864  -351  -2693  3928  7118  -9671  -9221  -5904  52  8374  -9
853  1261  1638  -6103  -932  8156  -4477  -1517  5219  -6783  -330  1432  -7788  12
61  2229  1325  -2043  -6003  -2482  498  1916  -6169  -3010  1125  2237  -8318  989
3  2008  -6276  6309  -1259  3704  -36  -6912  1746  1305  2488  -434  9432  -2833  1
566  -2403  374  -4098  2141  8136  1154  -3953  -8994  -861  6512  290  -1706  1570
 -1990  -884  -3396  -718  -3675  6327  6201  -6905  -8569  -8915  -3427  4636  2402
 4540  -8454  -3596  -4235  -7983  -1973  -6332  7763  -55  436  -6441  7955  1674  -
1615  -7064  2280  -9526  -4824  3380  2387  -6278  -7871  1459  1049  9279  5655  -
9692  464  -4484  4238  -1463  -7170  -1414  9479  -7949  -6364  8352  -7630  7392
1077  5730  7715  -1663  5007  -2430  -9815  -8832  9506  1362  -4645  2915  5651
```

```
1017  -5678  -4597  -6554  1978  -5825  5408  -4520  -7569  1013  -112  -8529  -3684
 -6904  7669  -6709  226  -2150  7190  -3992  -9456  7537  -7247  -5251  -927  -5877
 -4521  386  8385  -6507  -8392  1694  273  -8581  770  -6187  -6579  -1510  8101  -
8280  4302  7678  -2056  -1253  23  -18  789  -7589  -1345  4587  432  1  3  15  96
Results:  34  65  9432
```

3.1.2 no heap shape

```
5
-1  3  8  10  20
Results:  No  peak  shape
```

```
15
1211  189  75  23  20  11  4  -15  -40  -45  -46  -77  –89  -91  -152
Results:  No  peak  shape
```

3.1.3 Minimum scale

```
3
-1  3  1
Results:  3  2  3
```

```
3
-1  3  4
Results:  No  peak  shape
```

3.1.4 Maximum scale

```
10000
The  test  data  is  too  long,  so  they  are  not  shown  here.  Please  refer  to  “case3.t
xt”

Results:  275  4185  9991
```

After verification, we can know that all the test results are correct.

IV. Analysis and Comments

As the test results have shown, our program has accomplished the goal to find the best peak shape. Using DP, our program divides the problem of best peak shape into the sub problems of the best peak shape of the sub sequences. Processing with the nodes one by one, before the node, the best peak and its uphill and downhill is preserved for comparison.

In our DP function, there are two n-times iterations is done to find and construct the peak’s uphill and downhill. The time complexity is O(N). In the main function, the data is input first, so the time complexity is O(N). Because each number must run the DP function and the scale of data is N, the time complexity is O(N^2). At last, compare each peak to find out which of them has best peak shape. This is done through an iteration by comparing each pair of the peak. As a result, the time complexity is O(N). To conclude, the overall time complexity of the program is O(N^2).

As for the space complexity, we must allocate some space for several fixed-sized arrays to store the data, such as the array named “seq” to store the data and “up” to store information about the uphill, to name just a few. The size of the space is related to the scale of the data. As a result, the space complexity of our program is O(N).

V. Appendix

```
#include<iostream>
#include<cstdlib>

using namespace std;

int seq[11000];//record sequence (sequence)
int up[11000];//record the longest sequence of increments with i as the last
 node
int down[11000];//records with i as the first node to decrement the longest
sequence


//Dynamic programming algorithm
void DP(int index, int N)
{
    int temp;
    temp = 0;

    //Calculating incremental sequences
```

```cpp
    for (int i = 0; i < index; i++)
    {
        if (seq[index] > seq[i] && temp < up[i] + 1)//if the sequence[index] > sequence[i] and temp < up[i]+1, means that this incremental sequence is longer
        {
            temp = up[i] + 1;
            up[index] = temp;                       //let down[index] be the longer incremental sequence
        }
    }

    temp = 0;

    //Calculating descending sequence(from the last number to the first number, contrary to the incremental sequence)
    for (int i = N - 1; i > N - 1 - index; i--)
    {
        if (seq[N - 1 - index] > seq[i] && temp < down[i] + 1)//if the sequence[index] > sequence[i] and temp < down[i]+1, means that this incremental sequence is longer
        {
            temp = down[i] + 1;
            down[N - 1 - index] = temp;                       //let down[N-1-index] be the longer incremental sequence
        }
    }

    return;
}


int main()
{

    //N is the number of the sequence，max records the maximum value subscript and determine if there is a peak
    int N, max = 0, maxindex, difference = 1000000;
    printf("please input the number of the sequence(3-10000):\n");
    //initialize the data
    cin >> N;
    printf("please input %d numbers(split with spaces or carriage returns):\n", N);
    for (int i = 0; i < N; i++)
```

```c
    {
        scanf("%d", &seq[i]);
        up[i] = 1;
        down[i] = 1;
    }

    //excute the dynamic programming algorithm
    for (int i = 0; i < N; i++)
        DP(i, N);


    //Find the longest "peak" and compare the symmetry of the longest "peak"

    for (int i = 0; i < N; i++)
    {
        if (up[i] > 1 && down[i] > 1)     //First, find the longest "peak"
        {
            if (max < up[i] + down[i] - 1 )
            {
                difference = abs(up[i] - down[i]);
                max = up[i] + down[i] - 1;
                maxindex = i;
            }
            //finally, if it is equal, find the most symmetrical "peak"(the absolute value of (up[i] - down[i]) is the smallest)
            else if (max == up[i] + down[i] - 1 && difference > abs(up[i] - down[i]))
            {
                difference = abs(up[i] - down[i]);
                max = up[i] + down[i] - 1;
                maxindex = i;
            }
        }
    }


    //Output the corresponding result if it exists
    if (max)
        printf("%d %d %d\n", max, maxindex + 1, seq[maxindex]);
    else printf("No peak shape\n");

    system("pause");
    return 0;
}
```