浙江大学

The Best Peak Shape

Advanced Data Structures and Algorithm Analysis

Research Project 4

Wang Zhongwei 王中伟

Lin Jiafeng 林家丰

Jin Yuruo 金雨若

April 29, 2019

Contents

Chapter 1

Introduction

1.1 Problem Description

A ”peak shape” of length L is an ordered sequence of L numbers $\{D_1, ..., D_L\}$ satisfying that there exists an index $i(1 < i < L)$ such that $D_1 < ... < D_{i-1} < D_i > D_{i+1} > ... > D_L$.
The target of this problem is to find the longest ”peak shape” subsquence of a raw squence of length N.

1.2 Purpose of this report

In this report, we provide a method to transfer the ”Best Peak Shape” problem into the Longest Increasing Subscequence problem. Since the LIS problem has a well-known solution of $O(NlogN)$ time complexity, so this problem can be solved in $O(NlogN)$ time.

Chapter 2

Algorithm Specification

2.1 Transfer the problem into LIS problem

First, the ”peak shape” sequence can be considered to be a combination of an increasing sequence $\{A_1, ..., A_i\}$ and a decreasing sequence $\{A_i, ..., A_n\}$. Define I_i = the longest increasing subsequence of $\{A_1, ..., A_i\}$, D_i = the longest decreasing subsequence of $\{A_i, ..., A_n\}$. Thus, the longest ”peak shape” subsequence is

$$\{A_1, ..., A_i, ..., A_n\} \mid max\{\text{len}(I_i)+\text{len}(D_i) - 1\}.$$

If the $I_i, D_i (1 \leq i \leq n)$ is found in $O(T(n))$, then the time complexity of this problem is $O(n + T(n))$.

2.2 Longest Increasing Subsequence

Finding I_i, D_i (D_i is the reverse of the LIS of $\{A_n, A_{n-1}, ..., A_i\}$) is LIS problem. Note that I_i is only related to $I_j (1 \leq j \leq i - 1)$, so $I_i = I_j \bigcup \{A_i\} \mid \max\{\text{len}(I_j)\}$. By dynamic programming, this algorithm can be done in $O(n^2)$.

Algorithm 1 Longest Increasing Subsequence $O(n^2)$

function LONGEST-INCREASING-SUBSEQUENCE(A_i)
  for $i = 1$ to n **do**
    $I_i := I_j \bigcup \{A_i\} \mid \max\{\text{len}(I_j)\}, 1 \leq j \leq i - 1$
  end for
  return I_n
end function

2.3 LIS faster algorithm

When selecting I_j for $I_i (I_i = I_j \bigcup \{A_i\})$, if $\text{len}(I_j) = \text{len}(I_k)$ and $A_j < A_k$, I_j is always a better option than I_k because A_j must smaller than A_i. By this idea, we can just record the smallest last element E_k of LIS of length k. Every time we find a max E_k for I_i, and then update E_{k+1}. Obviously, E_k is a increasing sequence (otherwise, if $E_{k+1} < E_k$, then E_k can be updated by E_{k-1} and A_k), so E_k can be found by binary serach, this take $O(logn)$. The time complexity of whole algorithm is $O(nlogn)$.

Algorithm 2 Longest Increasing Subsequence $O(nlogn)$

function LONGEST-INCREASING-SUBSEQUENCE(A_i)
    for $i = 1$ to n **do**
        $k := j \mid \max\{A_{E_j} \leq A_i\}, 1 \leq j < i$
        $I_i := I_{E_k} \bigcup \{A_i\}$
        $E_{k+1} := \min\{A_{E_k}, A_i\}$
    end for
    return I_n
end function

Chapter 3

Testing Results

3.1 Correctness Test

3.1.1 Test data 1

Purpose

There are two ”best peak shape” sequence, the program must select the more symmetric one.

Input

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

Output

```
10 14 25
```

3.1.2 Test data 2

Purpose

The solution does not exist.

Input

```
5
1 2 3 4 5
```

Output

```
No peak shape
```

3.1.3 Test data 3

Purpose

There exists two same number adjacent.

Input

```
10
1 2 3 4 5 5 4 3 2 1
```

Output

```
9 5 5
```

3.1.4 Test data 4

Purpose

The input is zero

Input

```
0
```

Output

```
No peak shape
```

3.1.5 Test data 5

Purpose

The input is exactly the best peak shape.

Input

```
10
1 2 3 4 5 6 5 4 3 2 1
```

Output

```
10 5 6
```

Chapter 4

Analysis and Comments

4.1 Time complexity

The transition from the original problem to LIS problem takes $O(n)$ time, and the algorithm to find longest increasing subsequence takes $O(nlogn)$ time. Therefore, the time complexity of the whole program is $O(nlogn)$.

4.2 Space complexity

In the algorithms where time complexity are $O(n^2)$ and $O(nlogn)$, we use a table of length n to record information, so the space complexity is $O(n)$.
If it is necessary to list the result sequence, we can use another table to record the last element of every LIS from length 1 to n. The size of this table is also $O(n)$.

Appendices

Appendix A

Source Code (in C/C++)

```
#include <cstdio>
#include <algorithm>

#define SIZE 10001
#define INF (1 << 14)

int arr[SIZE];
int left_dp1[SIZE], left_dp2[SIZE];
int right_dp1[SIZE], right_dp2[SIZE];

void initDP();
void cal(int N, int *a, int *dp1, int *dp2);

int main(void)
{
    int N;
    /* get input data */
    scanf("%d", &N);
    for (int i = 0; i < N; i ++) {
        scanf("%d", &arr[i]);
    }
    /* calculate the LIS of 'arr' */
    /*'LIS' means the longest  increasing subsequence*/
    cal(N, arr, left_dp1, left_dp2);
    /*
    * D_i of 'arr' = I_i of reverse of 'arr'
    * calculate the LIS of reverse of 'arr'
    */
    std::reverse(arr, arr + N);
    cal(N, arr, right_dp1, right_dp2);

    /* enumerate the index of solution */
    int idx = 0, ans = 0, diff = INF;
    /*ans is the length*/
    for (int i = 0; i < N; i ++) {
```

```
        /* check if the solution is legal */
        if (left_dp2[i] <= 1 || \
            right_dp2[N - 1 - i] <= 1) {
            continue;
        }

        int cur_ans = left_dp2[i] + \
                      right_dp2[N - 1 - i] - 1;
        int cur_diff =std::abs(\
                      left_dp2[i] - right_dp2[N - 1 - i]);

        if (cur_ans > ans || \
            (cur_ans == ans && cur_diff < diff)) {
            ans = cur_ans;
            diff = cur_diff;
            idx = i;
        }
    }
    if (ans > 0) {
        /* 1-index */
        printf("%d %d %d\n", ans, \
               idx + 1, arr[N - 1 - idx]);
    }
    else {
        printf("No peak shape");
    }
    return 0;
}

void initDP(int *dp)  /*function to initial dp*/
{
    dp[0] = -INF;
    for (int i = 1; i < SIZE; i ++) {
        dp[i] = INF;
    }
}
/*
* N       : size of the array
* a[i]    : element
* dp1[i] : the smallest value of last element of LIS of
*          length i
* dp2[i] : the length of LIS whose last element is a[i]
*/
void cal(int N, int *a, int *dp1, int *dp2)
{
    initDP(dp1);

    for (int i = 0; i < N; i ++) {
```

```
    /*
    * lower_bound :
    * Find the first number greater than or equal to a[i]
    * from the dp1 to the dp1+N-1 , if found than returns
    * the number's adress , or return the dp1+N if not.
    */
    int *dp ;
    dp = std::lower_bound(dp1, dp1 + N, a[i]);
    /*
    * update smallest element of LIS of length (l + 1)
    * (*dp = dp[l + 1])
    */
    *dp = std::min(*dp, a[i]);

    /* length of LIS = (dp - dp1) */
    dp2[i] = (int)(dp - dp1);
    }
}
```

Appendix B

Declaration and Signatures

Declarațion

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

Signatures

毛仲伟 林家丰 金雨君