Zhejiang University

project three

Beautiful Subsequence

Professor: Qinming He

A report submitted in partial fulfilment of the requirements
to summarize and present our work in project three

April 19, 2020

Declaration

We hereby declare that all the work done in this project titled "Beautiful Subsequence" is of our independent effort as a group.

Duty Assignments:

Contents

Chapter 1

Introduction

1.1 What is given

A sequence which have no less than one pair of adjacent elements with difference no larger than a specific number is defined as beautiful sequence.

1.2 Our mission

In this problem, we need to figure out the total number of beautiful subsequences of a given sequence. It is important to note that the relative position of elements in subsequence cannot be changed.

1.3 Input

First Line: n,m

$\boldsymbol{n}$ (from 2 to 10^5) is the length of the sequence;
$\boldsymbol{m}$ (from 1 to 10^3)is the biggest difference between two neighbours that meet the definition of beautiful sequence;
Second Line: The sequence of n numbers

1.4 Output

Just one line: the number of the subsequences;

1.5 Sample

Sample Input:
4 2
5 3 8 6

Sample Output:
8

Chapter 2

Algorithm Specification

2.1 The Data Structure

To solve this problem, we use three arrays to store the values.

```
int sequence[100001];
//This array is used to store the elements of the input sequence;
static long sub[100001];
//This array is used to store the number of beautiful sequence of the subsequence which is
    ended with i from the first element to n-th elememt;
static long nonsub[100001];
//This array is used to store the number of nonbeautiful sequence of the subsequence which is
     ended with i from the first element to n-th elememt;
```

A global variable to iterative record the number of all beautiful subsequence.

```
static long sum;
//Store the number of beautiful subsequences
```

2.2 The Algorithm

2.2.1 Overall:

To solve this problem, we use Dynamic Programming to analyze it.

First, we classify all the subsequences according to which element they end with.

Then, classify each category again into two categories. Take the k^{th} element i for example, one is ended with i and is beautiful sequence and the other category is ended with i but is not beautiful sequence. The number of the subsequences of those two categories is 2^{k-1}.

So, we need to calculate the number of nonbeautiful sequence of the subsequence, which is ended with i from the first element to n^{th} elememt. It needs Dynamic Programming.

Here is the iterative equation:

$$\textbf{sub[i]} = \textbf{sum} + \textbf{sum of nonsub[k]}$$

sum means the total number of beautiful subsequence ended with from 1^{th} to $i-1^{th}$, sum of nonsub[k] means the total number of nonbeautiful subsequence ended with k_1^{th}, k_2^{th}, k_3^{th} element...k_i^{th} element should have difference with i^{th} element no larger than m.

2.2.2 The get_sub Function

This function is to get the total number of beautiful subsquence by adding the number of subsquence from element 1 to element n.

Algorithm 1 get_sub()

Input: n,m
  $nonsub[1] \leftarrow 1$; //the array of length 1 has no beautiful subsequence.
  for $i = 2 \rightarrow n$ **do**
    SUB_CHECK(i, m)
  end for

2.2.3 The sub_check Function

This function is used to figure out the number of beautiful sequence of the subsequence which is ended with i

Algorithm 2 sub_check()

Input: i,m
  $sub[i] \leftarrow sum$ // if there is no matched number from element 1 to element i-1 with element i then the number of beautiful subsequence stay the same
  for $j = 1 \rightarrow i - 1$ **do**
    if $sequence[i] - sequence[j]is \leq m$ **then**
      $sub[i] \leftarrow sub[i] + nonsub[j]$ // update the subsequence
    end if
  end for
  $nonsub[i] \leftarrow$ AVOID_OVERFLOW$(i-1\ ) - sub[i]$//get the value of nonsubsequence
  $sum \leftarrow sum + sub[i]$//add the number of beautiful subsequence of sequence[i] to the total

2.2.4 The avoid_overflow Function

This function is used to reduce the power by Euler Power‘s law to avoid overflow.

Algorithm 3 avoid_overflow()

Input: n;//the power
Output: result
  $result \leftarrow 1$
  $a \leftarrow 2$
  while $n > 0$ **do**
    if The power is odd **then** $result \leftarrow (result \times a) \bmod overflow$
    end if
    $n \leftarrow n \div 2$
    $a \leftarrow a^2 \bmod overflow$
  end while

Chapter 3

Test Results

3.1 The special test cases to check the program

To test the correctness of our program, we have made some special test cases where the program may make mistakes. The test cases, the purpose of the case, the expected results and the behavior of the program are as follow.

For n = 100000, the input is too big to be shown in the report, the input is shown in the file "MAX_input.txt" in the folder code.

The description of the case	**the purpose of the case**	**the input & output**	**the expected result**
The sample	Make sure that the program can work for the normal test case.	4 2 5 3 8 6 8	8
The least input	Make sure that the program can work for the least test case.	21 34 1	1
The maximum input (all the number are randomly generate)	Used to confirm that if the program is able to correctly calculate the maximum test case.	100000 857 Details omitted (Please refer to the "MAX_input.txt" for details) 420256882	420256882
A random case that has no duplicated numbers	Used to confirm that if the program is able to correctly calculate the case that has no duplicated numbers in the input sequence.	12 4 12 5 8 16 4 7 11 13 2 9 10 14 3785	3785

The description of the case	the purpose of the case	the input & output	the expected result
A random case that has duplicated numbers	Used to confirm that if the program is able to correctly calculate the case that has duplicated numbers in the input sequence. Here, the first number '2' are different from the second number '2'.	6 1 2 2 3 3 5 4	51
		51	
A case that all the numbers have differences larger than m.	Used to confirm that if the program can correctly calculate the case that all the numbers in the sequence has differences larger than m.	20 5 370 162 314 269 68 228 97 123 150 45 183 261 252 338 39 276 60 327 52 26	0
		0	
A case that all the numbers have differences less than m.	Used to confirm that if the program can correctly calculate the case that all the numbers in the sequence has differences less than m.	20 28 25 20 24 11 10 9 12 18 4 6 15 3 5 2 8 22 23 7 27 1	1048555
		1048555	

3.2 The time test

1. Use random function to generate large amounts of data for testing
2. The running time of the test cases:

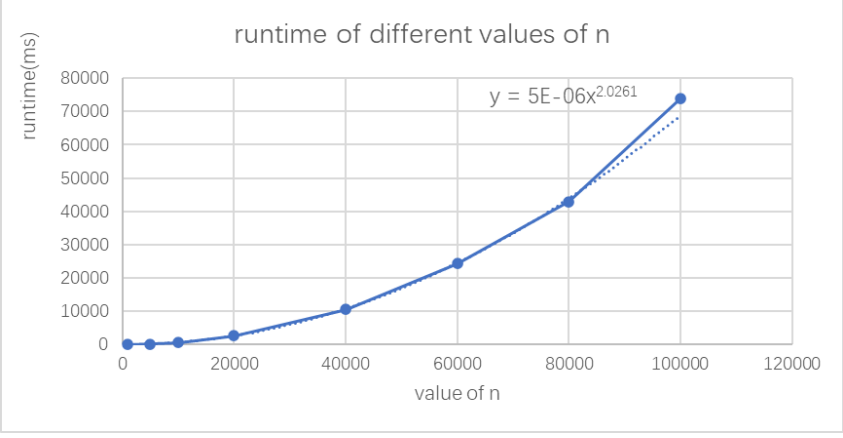

Figure 3.1: *The running time of the different values of n*

From the chart we can see that the time complexity of this program is probably O (N^2).

3. In order to deduce this conclusion more directly, we made the following chart. Here we use the square root of runtime as Y-axis;

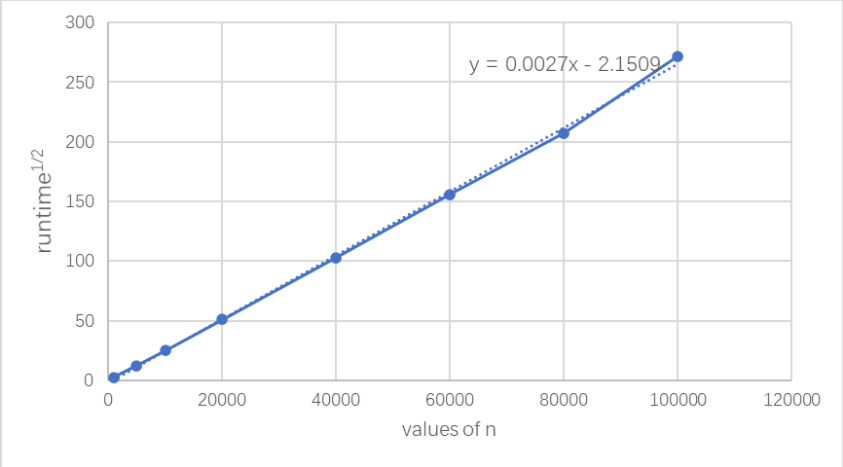

Figure 3.2: *Program run time*

Chapter 4

Analysis and Comments

4.1 Analysis of Time Complexity

The function **get_sub()** gets the total number of beautiful sequence by adding the number of subsquence from element 1 to element n. Thus the time complexity of function **get_sub()** is **O(N)**.

The function **sub_check()** is used to figure out the number of beautiful sequence of the subsequence which is started from 1 and ended with the input i. Thus the time complexity of function **sub_check()** is also **O(N)**.

The time complexity of the program is $\mathbf{O(N^2)}$ since function **get_sub()** calls **sub_check()** n times.

4.2 Analysis of Space Complexity

All the data is stored in arrays, whose size are fixed, so the space complexity of the program is **S (1)**.

4.3 The correctness of our program

As we mentioned in the algorithm-overall part, we use Dynamic Programming to analyze this problem through the iterative equation:

$$\textbf{sub[i] = sum + sum of nonsub[k]}$$

(sum means the total number of beautiful subsequence ended with from 1^{th} to $i-1^{th}$, sum of nonsub[k] means the total number of nonbeautiful subsequence ended with k_1^{th}, k_2^{th}, k_3^{th} element...k_i^{th} element should have difference with i^{th} element no larger than m.)

We classify all the subsequences according to which element they end with and then classify each category again into two categories, the beautiful ones and nonbeautiful ones.

This classification can include all the subsequences without repetition.

Calculate the number of subsequences ended with i^{th} element based on the iterative equation: **sub[i] = sum + sum of nonsub[k]**

For every element, the number of the beautiful subsequence is based on the elements in front of it. If the new subsequence ended with i^{th} element is beautiful, it has two possibilities:

1.The element in front of i^{th} element and adjacent to it in the new subsequence has the difference with i^{th} element no larger than m, then no matter the subsequence ended with this element of the new subsequence are beautiful or not, the new subsequence is beautiful;

2.The element in front of i^{th} element and adjacent to it in the new subsequence has the difference with i^{th} element larger than m, than only the subsequence ended with this element of the new subsequence are beautiful, the new subsequence can be beautiful.

So **sub[i] = the sum of (sub**$[k_i]$ **+ nonsub**$[k_i]$**) + the sum of sub**$[k'_i]$, $(k_i, k'_i \leq i)$ k_i means the k_i^{th} element which has difference no larger than m, k'_i means the element does not have difference no larger than m. Analyze the equation, we can find that no matter the k^{th} element has no larger than m difference with i^{th} element or not, if the subsequence ended with k^{th} element is beautiful, it must be beautiful adding i^{th} element, and if the subsequence ended with k^{th} element is not beautiful, it should has no larger than m difference with i^{th} element, so the equation can be: **sub[i] = sum + sum of nonsub[k]**, it is correct.

4.4 Comments

In our program, we used the DP(Dynamic Programming) for optimization. Decompose the original problem, find the number of subsequnces in the first i data that meets the requirements, and update the sum of beautiful subsequences. The state transition equation **sub[i] = sum + sum of nonsub[k]** of dynamic programming is determined, so that the overall time complexity is controlled at $\mathbf{O(N^2)}$.

4.5 Further possible improvements.

Firstly, in our program algorithm, there is still the problem of repeated calculation. If we can reduce these repeated calculations and find out a better state transition equation, we can greatly improve the running time of the program.

Secondly, using a segment tree may reduce repeated calculations to reduce the time complexity of the program.

Appendices

Appendix A

Source Code (in C)

```
// main.c
// Beautiful_Subsequence

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define overflow 1000000007 //used to deal with the overflow problem

void get_sub(int n,int m); //used to figure out the number of beautiful sequence of the
    subsequence
void sub_check(int i,int m); //used to figure out the number of beautiful sequence of the
    subsequence which is ended with i
long long avoid_overflow(int n); //used to solve the overflow problem by Euler's power law

int sequence[100001]; //used to store the elements
static long sub[100001]; //used to store the number of beautiful sequence of the subsequence
    which is ended with i from the first element to n-th elememt
static long nonsub[100001]; //used to store the number of nonbeautiful sequence of the
    subsequence which is ended with i from the first element to n-th elememt
static long sum; //store the number of beautiful subsequences

int main(int argc, const char * argv[])
{
    int i;
    int n,m;

    scanf("%d%d",&n,&m); //read the keynum and the sequence
    for(i=1;i<=n;i++)
    {
        scanf("%d",&sequence[i]); //get the elements
    }
    get_sub(n,m); //caculate the number of beautiful subsequences

    printf("%ld\n",sum);
    system("pause");
}

void get_sub(int n,int m) //get the total number of beautiful subsquence
{ //by adding the number of subsquence from element 1 to element n
```

```
    int i;
    nonsub[1]=1;
    for(i=2;i<=n;i++)
    {
        sub_check(i,m); //figure out the number of beautiful subsequences ended with i from
            the first to the last
    }
}

void sub_check(int i,int m)
{
    int j;

    sub[i]=sum; //if there is no matched number from element 1 to element i-1 with element i
                  //then the number of beautiful subsequence stay the same
    for(j=1;j<i;j++)
    {
        if(abs(sequence[j]-sequence[i])<=m) //if there is such a matched number j
        {
            sub[i]=sub[i]%overflow+nonsub[j]%overflow; //then the nonbeautiful subsequence
                from element 1 to j
            sub[i]=sub[i]%overflow; //become the beautiful subsequence

        }
    }
    nonsub[i]=avoid_overflow(i-1)%overflow-sub[i]%overflow; // get number nonbeautiful
        subsquence by subtracting nunber of
    while(nonsub[i]<0) // beautiful subsquence from the total number of subsequence
    {
        nonsub[i]=nonsub[i]+overflow; //dealing with the overflow problem in case the number
            is less than 0 (or underflow)
    }
    nonsub[i]=nonsub[i]%overflow;
    sum=sum%overflow+sub[i]%overflow;
    sum=sum%overflow; //dealing with the overflow problem in case the number is too large
}
long long avoid_overflow(int n) //this function is to deal with the overflow problem by using
     Euler's power law
{
    long long result; //result=(2^n)%overflow
    long long a;
    result=1;
    a=2;
    while(n>0)
    {
        if(n%2==1) //n&1==1
            result=(result*a)%overflow;
        n=n/2; //n=n>>1
        a=(a*a)%overflow;
    }
    return result;
}
```