Beautiful Subsequence

Group 18

Date:2020-04-28

CONTENT

Chapter 1 Introduction

1.1 Background

Dynamic Programming is mainly an optimization over plain recursion. Wherever we see a recursive solution that has repeated calls for same inputs, we can optimize it using Dynamic Programming. The idea is to simply store the results of subproblems, so that we do not have to re-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial.

For example, if we write simple recursive solution for Fibonacci Numbers, we get exponential time complexity and if we optimize it by storing solutions of subproblems, time complexity reduces to linear.

1.2 Problem Description

In this project, we want to solve the problem for finding the Beautiful Subsequence in some general sequence. If a sequence with at least 2 elements contains 2 neighbors with diffference no larger than m, where m is the given integer, we call such sequence **Beautiful Sequence**. So the goal of the project is that we want to calculate the number of **Beautiful Subsequences** in the given sequences.

As an illustration, we would let $m = 2$, and give out a sequence which contains 4 elements, 5 3 8 6. In this way, we can calculate the number of beautiful subsequences. And for more specifically, we would hold 8 beautiful subsequences, they are:

5 3 5 6 8 6 5 8 6
5 3 8 5 3 8 6 5 3 6 3 8 6

In order to solve the problem of this project, we need to use some algorithms and make corresponding improvements, like dynamic programming algorithms. This report will introduce the implementation of the specific algorithm and the techniques for optimizing and using it in the Beautiful Subsequence problem.

Chapter 2 Algorithm Specification

2.1 Data Structure

The data structure of the program is:

```
//store the original sequence
int a[MAX];
//store the size of original sequence
int n;
//store the difference limided by the input data
int diff;
//store the dynamical results step by step
int k[MAX];
//bi[k] store the value of 2^k%(10e9+7)
int bi[MAX] = { 0 };

//dynamic programming;
void dyn();
//initialize
void init();
//init the bi sequence
void initbi(int n);
```

2.2 Initialize the Sequence

We use the algorithm of **quick power** which is taught in the course fundamentals of data structures to solve the value of 2^{j-1}.

The pseudocode of the initialization process is:

```
/*initialize*/
a[0] := 0;      /*As the reserved bit of the array*/
/*init the bi sequence*/
Function initbi(int n) {
    bi[0] := 1;
    for i := 1 to n-1
        bi[i] := (bi[i - 1] << 1) % Module;
    i:=i+1
    end for
    return;
}
```

2.2 Dynamic Programming

Set k[i] as the number of beautiful sequences in the sequence consisting of numbers from a[1] to the a[i]. There are three steps to calculate the number of beautiful sequences.

1. Suppose there are already $n-1$ elements in the sequence and $k[n-1]$.
2. After inserting $a[n]$, first of all, it is combined with the generated $k[n-1]$ beautiful subsequences, adding $k[n-1]$ beautiful subsequences.
3. Look for the element that will react with this new element from scratch. If the found element is $a[1]$ only one beautiful subsequence will be added. Else if the number of newly added subsequences is $2^{j-1}+k[j-1]-k[j]$.

The pseudocode of the dynamic programming algorithm is:

```
/*dynamic programming*/
Function dyn() {
    k[0] := 0;
    for i := 1 to n
        k[i] := (2 * (k[i - 1] % Module)) % Module;

        for j := 1 to i-1
            if abs(a[j] - a[i]) <= diff and j != 1 then
                k[i] = (k[i] % Module + bi[j-1] + k[j - 1] % Module - k[j] % Module) % Module;
                if (k[i] < 0) then k[i] := k[i] + Module;
                end if
```

```
            end if
            else if abs(a[j] - a[i]) <= diff and j == 1)
                k[i] = (k[i] % Module + 1) % Module;
            end else if
            j:=j+1;
        end for
        i:=i+1;
    end for
}
```

Chapter 3 Testing Results

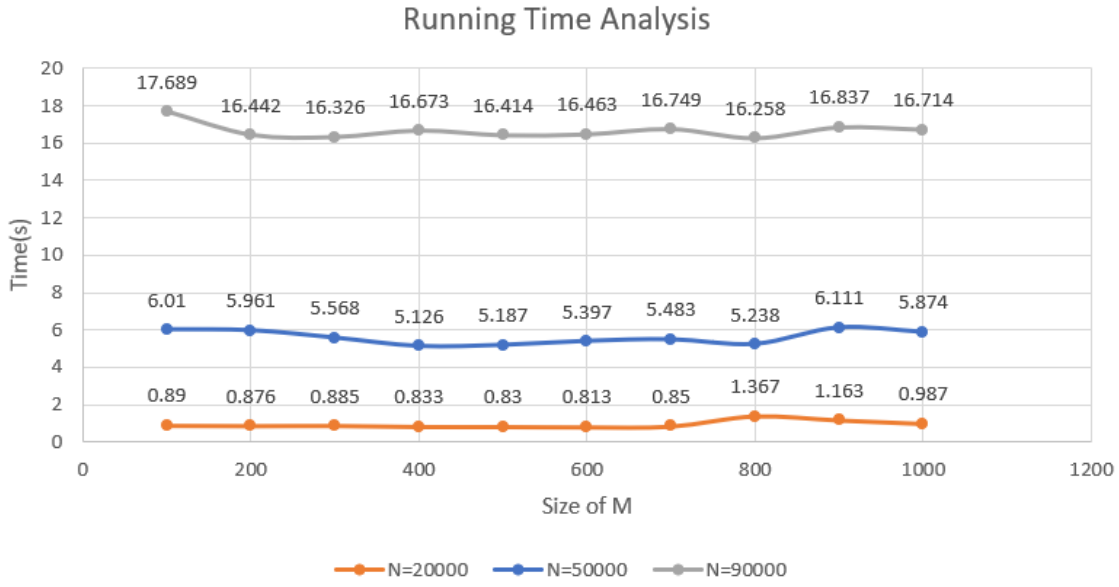
3.1 Testing Results in Table

No.	Testing File	N	M	Running Time (s)	Result
1	test1.txt	2000	100	0.012	271967835
2	test2.txt	2000	500	0.010	107640810
3	test3.txt	2000	1000	0.015	929561631
4	test4.txt	20000	100	0.991	856753706
5	test5.txt	20000	500	1.005	67435423
6	test6.txt	20000	1000	1.054	601896045
7	test7.txt	90000	100	17.857	354754462
8	test8.txt	90000	500	18.442	886043302
9	test9.txt	90000	1000	19.523	264387683

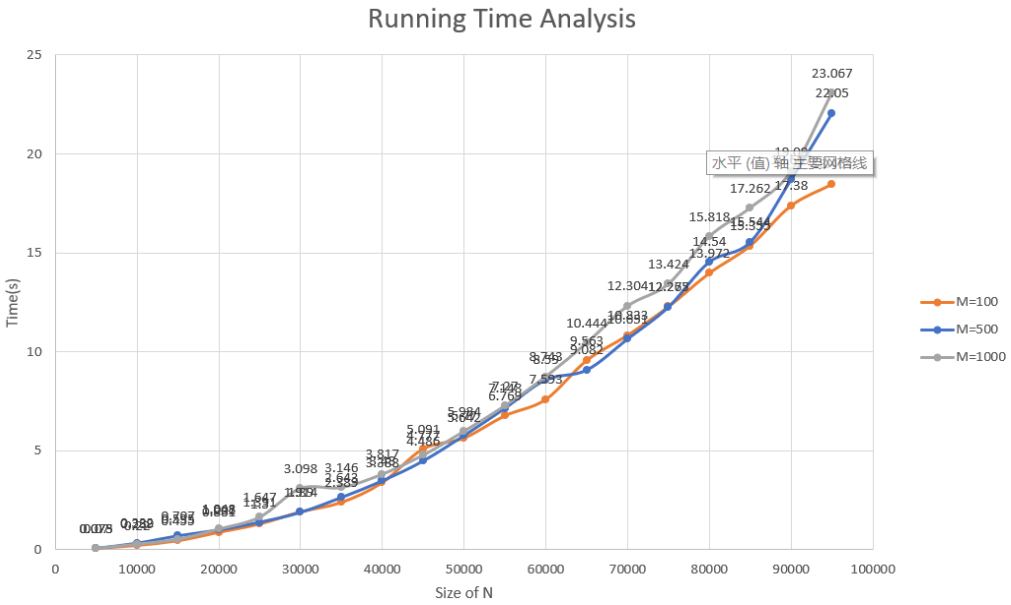
Since N is between 2 and 100000 and M is between 1 and 1000, we will discuss these two factors of their affects on running time.

From the testing results we can conclude that running time is strongly related to N and is partly related to M. When N grows and M fixed, the running time grows rapidly. When N is small, M affects almost nothing. When N is large enough, M begins to affect the running time slightly.

3.2 Testing Results in Graph



When N is set as 20000, 50000 and 90000 and M changes from 100 to 1000, we can see that running time almost doesn't change with M but N.



When M is set as 100, 500 and 1000 and N changes from 10000 to 100000, we can see that running time almost doesn't change with M but N. We can also know that the running time grows as $O(N^2)$

Chapter 4 Analysis and Comments

4.1 Analysis on Time Complexity

There are 1 primary function and 3 secondary functions in the program.

int main() function simply calls *init*(), then print the result. Therefore it is $O(1)$ apart from *init*().

void init() function opens a file, reads data and start the other two functions. Since there are no loops or recursive calls, it is $O(1)$ apart from the other two functions.

void initbi(n) function calculate $2^1, 2^2, 2^3, \ldots, 2^n (mod\ (10e9 + 7))$, it contains a loop from 1 to n. Therefore it is $O(n)$.

void dyn() function using dynamic programming method and it contains a loop from 1 to n using i as outer iteration variable and a loop from 1 to i as inner iteration variable. There for it is $O(n^2)$.

In conclusion, the whole program is $O(n^2)$.

4.2 Analysis on Space Complexity

There are three global arrays with fixed size and no extra space is used in each function. Therefore, the space complexity is $O(1)$.

4.3 Comments

How to handle overflow is a problem in this program. At first we didn't think of it and something went wrong in the test. Then we use module almost everywhere, but the problem still existed. Finally we use an array to store the exponents, and there are no overflows in the end.

Appendix

Source Code in C

```c
#pragma warning(disable : 4996);
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define MAX 100000
#define Module 1000000007
#define FILE_NAME "./test1.txt"
//input:
//store the original sequence
int a[MAX];
//store the size of original sequence
int n;
//store the difference limided by the input data
int diff;
//store the dynamical results step by step
int k[MAX];
//bi[k] store the value of 2^k%(10e9+7)
int bi[MAX] = { 0 };
//dynamic programming;
void dyn();
//initialize
void init();
//init the bi sequence
void initbi(int n);
int main() {
    init();
    printf("%d", k[n]);
    return 0;
}
void init() {
    a[0] = 0; //As the reserved bit of the array
    FILE* fp;
    fp = fopen(FILE_NAME, "r");
    if (fp == NULL) {
        printf("fail to open ");
        printf(FILE_NAME);
        exit(0);
    }
    else {
        fscanf(fp, "%d %d", &n, &diff);
        for (int i = 0; i < n; i++) {
            fscanf(fp, "%d", &a[i + 1]);
        }
        initbi(n);
        dyn();
    }
    fclose(fp);
}
void initbi(int n) {
    bi[0] = 1;
    for (int i = 1; i < n; i++) {
        bi[i] = (bi[i - 1] << 1) % Module;
    }
    return;
}
/*dynamic programming*/
/*Set k[i] as the number of beautiful sequences in the sequence consisting of numbers from a[1] to
the a[i]*/
/*step1 Suppose there are already n-1 elements in the sequence and k[n-1]*/
/*step2 After inserting a [n], first of all, it is combined with the generated k [n-1] beautiful
subsequences, adding k [n-1] beautiful subsequences*/
/*step3 Look for the element that will react with this new element from scratch*/
/*If the found element is a[1]Â£Â¬ only one beautiful subsequence will be added*/
/*else if, the number of newly added subsequences is pow(2, j - 1) + k[j - 1] - k[j]*/
void dyn() {
    k[0] = 0;
    for (int i = 1; i <= n; i++) {
        k[i] = (2 * (k[i - 1] % Module)) % Module;
        for (int j = 1; j < i; j++) {
            if (abs(a[j] - a[i]) <= diff && j != 1) {
                k[i] = (k[i]%Module +bi[j-1] + k[j - 1] % Module - k[j] % Module) % Module;
                if (k[i] < 0)k[i] = k[i] + Module;
            }
            else if (abs(a[j] - a[i]) <= diff && j == 1) {
                k[i] = (k[i] % Module + 1) % Module;
            }
        }
```

```
    }
}
```

Testing Set Generating Script in PHP

```php
<?php
$N_start = 90000;
$M_start = 100;
for ($n = $N_start,$m = $M_start;$m <= 1000; $m = $m + 100){
    $file = fopen("test_m=".$m."n=".$n.".txt","w");
    $first_line = $n." ".$m."\n";
    fwrite($file, $first_line);
    for ($i = 0; $i < $n; $i++) {
    $next =  mt_rand(1,100000)." ";
    fwrite($file,$next);
    }
    fwrite($file, "\n");
    fclose($file);
}
?>
```

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

Programmer:
Tester:
Report Writer: