浙江大学

Advanced Data Structures and Algorithm Analysis

Laboratory Projects

Beautiful Subsequence

Group 15

聂俊哲 张琦 张溢弛

Date: 2020-04-20

Content

Appendix: source code

References

Declaration

Signatures

Chapter 1: Introduction

- A sequence is called beautiful if it contains 2 neighbors with difference no larger than a given integer m, in this project, out target is to calculate the number of all the beautiful subsequences in a array of length n.
- Something we should take care of
    - the input form: n and m in a line and n integers in the next line, n is the length of array and m is the given difference
    - the data size: n is no larger than 100000 and m is no larger than 1000, each element in the array is no larger than 100000
    - the output form: the answer might be very large, so the result will be modulo by 1000000007.
- Related Algorithm: **Dynamic Programming**

Chapter 2: Data Structure / Algorithm Specification

- We use an important 2-dimension array `s[i][j]` to represent whether |num[i]-num[j]|<=m or not
- The **pseudo-code of our program** is as following

```
ALGORITHM:Beautiful Subsequence
INPUT: 2 integers n and m, and n integers
OUTPUT: The number of beautiful subsequences of the n integers

read in n and m
read in n integers
for(i=1;i<=n;i++){
    for(j=1;j<i;j++){
        set s[i][j]=1 if |num[i]-num[j]|<=m else set s[i][j]=0
    }
}
for(i=2;i<=n;i++){
    include_i=0 // caculate the number of the subsequences which is no longer a beautiful subsequence without the i-th element.
    for(j=1;j<i;j++){
        if(s[i][j]==1)
            set include_i+= pow(2, j - 1) + result[j - 1] - result[j];
    }
    set result[i]=2*result[i-1]+include_i;
    set result[i]%=1000000007
}
print the result: result[n]
```

Chapter 3: Testing Results

- We write a test program that can produce big-size test data
  - You can find the test program as test.cpp in the codes.
  - You can find the large size test data in the txt file.
- Test 1： the same as sample

```
Input:
4 2
5 3 8 6
Output:
8
```

- Test 2: an edge case that n=2

```
Input:
2 5
8 9
Output:
1
```

- Test3: an edge case that m=1

```
Input:
10 1
1 2 3 4 5 6 7 8 9 10
Output:
880
```

- Test4: an edge case that m=1000

```
Input:
5 1000
996 1896 2999 3120 4001
Output:
17
```

- Test5: a small size test case n=10 and m=100

```
Input:
10 100
318 982 162 046 602 042 827 616 885 918
Output:
516
```

- Test6: a middle size test case n=100 and m=100

```
Input:
100 100
654 956 164 199 427 511 730 018 643 027 997 557 866 337 797 181 725 134 712 251 293 301 550
555 161 782 550 884 841 488 272 897 089 018 005 399 410 553 099 041 359 373 741 017 550 292
138 304 245 531 453 111 752 474 078 910 312 311 367 877 366 830 873 897 081 296 591 402 404
386 243
051 947 320 048 217 559 032 785 143 415 739 712 401 434 354 193 296 907 046 103 022 308 931
031 570 414 253 348 590
Output:
836596602
```

- Test7: a large size test case n=100000 and m=1000

```
You can find the big-size test data in the testdata.txt
It is a waste of report space to put the input and output data here.
```

Chapter 4: Analysis and Comments

- A **naive traditional algorithm** is using backtracking to find all the subsequences of the n integers and check each of them whether is's a beautiful sequence or not. The space complexity is $O(2^n)$ and the time complexity is $O(n * 2^n)$.
- **Comments of our Algorithm**
  - we use the array `result[i]` calculate the number of beautiful subsequences of the first i integers.
  - we can divide the `result[i]` into 2 different parts
    - `2*result[i-1]`, it means that the number of beautiful subsequences will double when add a new integer in the end because this new integer could be inserted or not to the origin beautiful subsequences, which makes the origin beautiful subsequences still a beautiful subsequence.
    - `include_i`, which means the subsequence isn't beautiful before insert the i-th integer into the subsequence, and become a beautiful subsequence because of the addition of the i-th integer. For every element whose index is smaller than the i-th element and difference between which and the i-th element is no larger than m, the number of the beautiful subsqunences which is made a beautiful subsqunence by the i-th element and the j-th element equals to the number of the non-beautiful subsqunences minors `(result[j] - result[j - 1]).`
    - so we have such an equation
      $include\_i = \sum_{j<i}(s[i][j] * (2^{j-1} - (result[j] - result[j-1]))) = \sum_{j<i, s[i][j]=1}(2^{j-1} + result[j-1] - result[j])$
    - so the state transition equation will be
      $result[i] = 2 * result[i-1] + \sum_{j<i, |num[i]-num[j]| \le m}(2^{j-1} + result[j-1] - result[j])$
    - We can use a DP algorithm to calculate the number of beautiful subsequences
  - For appending, we found that when n is large enough (larger than 31), 2^n will overflow. So we need an approach to calculate the result of $2^n \bmod (10^9+7)$.
    - $$\begin{aligned} prove: & a * b \equiv (a\%m) * (b\%m) \bmod m \\ proof: & let\ a = m * x_a + y_a,\ b = m * x_b + y_b \\ & \therefore\ (a\%m) * (b\%m) = y_a * y_b \\ & \because\ a * b = m^2 x_a x_b + m(x_a y_b + x_b y_a) + y_a y_b \\ & \therefore\ a * b \equiv (a\%m) * (b\%m) \equiv y_a y_b \bmod m \end{aligned}$$
    - $$\begin{aligned} & so,\ a^b \equiv (a^2)^{\frac{b}{2}} \equiv (a^2\%m)^{\frac{b}{2}} \bmod m, when\ b\ is\ an\ even\ number; \\ & a^b \equiv (a^2)^{\lfloor \frac{b}{2} \rfloor} * a \equiv (a^2\%m)^{\frac{b}{2}} * a \bmod m, when\ b\ is\ an\ odd\ number. \\ & we\ can\ caculate\ this\ iteritively\ until\ b = 1. \end{aligned}$$
    - However, when a is large enough(larger than $\sqrt{10^9+7} \approx 31623$), a^2 and $(a^2\%m)^{\frac{b}{2}} * a$ may overflow, we need to caculate this more precisely. Lets begin with a^2.

$$
\begin{aligned}
&case1\ a < 31623: \\
&\qquad just\ caculate\ a^2\ directly. \\
&case1\ a \geq 31623: \\
&\qquad let\ a = 31623k + t,\ k \geq 1,\ 0 \leq t < 31623 \\
&\qquad so\ a^2 = 1000014129k^2 + 63246kt + t^2 \equiv 14122k^2 + 63246kt + t^2 \bmod (10^9 + 7)
\end{aligned}
$$

- **Analysis of algorithm complexity**
  - Time complexity: it's obvious in a DP problem that we have some double loops in the program, so the time complexity is $O(n^2)$
  - Space complexity: all the arrays are $O(n)$, so it is obvious that the time complexity is $O(n)$

Appendix: Source Code (if required)

```cpp
#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

const int constant = 1000000007;
const int radication = 31623;

int quick_power(int k); /*To caculate 2^k mod 1000000007 fastly. */
int quick_mul(int a, int b); /*To caculate a*b mod 1000000007 fastly. */
int quick_square(int a); /* To caculate a^2 mod 1000000007 fastly. */

int main()
{
    int n, m, i, j;
    cin >> n >> m;
    vector<int>num(n + 1); /* To store the numbers of the original sequence. */
    vector<int>result(n + 1, 0); /* To store the result of the subsequence from the 1-st element to the i-th element. */
    for(i = 1; i <= n; i++) cin >> num[i];
    for(i = 2; i <= n; i++) {
    unsigned int include_i = 0; /* Variable 'include_i' is to caculate the number of the subsequences which is no longer a beautiful subsequence without the i-th element. */
    for(j = 1; j < i; j++) {
        if(abs(num[i] - num[j]) <= m) include_i += quick_power(j - 1) + result[j - 1] - result[j]; /* For every element whose index is smaller than the i-th element and difference between which and the i-th element is no larger than m, the number of the beautiful subsquences which is made a beautiful subsquence by the i-th element and the j-th element equals to the number of the non-beautiful subsquences minors (result[j] - result[j - 1]). */
        include_i %= constant;
    }
    result[i] = 2 * result[i - 1] + include_i; /* 'result[i]' equals two times of result[i - 1] plus the the number of the subsequences which is no longer a beautiful subsequence without the i-th element. */
    if(result[i] >= constant) result[i] %= constant;
    }
    cout << result[n] << endl;
    return 0;
}

int quick_power(int k)
{
    int result = 1, n = 2;
    while(k > 0) {
    if(k % 2 == 1) result = quick_mul(result, n);
    k /= 2;
    if(n >= radication) n = quick_square(n);
    else n = n * n;
    }
    result %= constant;
    return result;
}
```

```
46
47  int quick_mul(int a, int b)
48  {
49      if(a < b) return quick_mul(b, a);
50      else if(a == b) return quick_square(a);
51      int x, y, result;
52      if((a + b) % 2 == 0) {
53      x = (a + b) / 2;
54      y = (a - b) / 2;
55      result = quick_square(x) - quick_square(y);
56      }
57      else {
58      x = (a + b) / 2;
59      y = (a - b) / 2;
60      result = quick_square(x) - quick_square(y) + b;
61      }
62      if(result < 0) result += constant;
63      return result;
64  }
65
66  int quick_square(int a)
67  {
68      int t = a / radication;
69      int p = a % radication;
70      int temp1_t = t * t / 70812;
71      int temp1_p = t * t % 70812;
72      int temp1 = (7057 * temp1_t + 14122 * temp1_p) % constant;
73      temp1 = (temp1 + p * p) % constant;
74      int temp2, temp2_t, temp2_p;
75      temp2_t = 2 * t * p / radication;
76      temp2_p = 2 * t * p % radication;
77      temp2 = (14122 * temp2_t + radication * temp2_p) % constant;
78      a = temp1 + temp2;
79      a %= constant;
80      return a;
81  }
82
```

References

List all the references here in the following format:

[1] 《算法导论》第三版

[2] PTA website https://pintia.cn/

Author List

- Zhang Yichi 3180103772
- Zhang Qi 3180103162
- Nie Junzhe 3180103501

Declaration

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

Signatures