Algorithms

Solve any problem to achieve a rank

View Leaderboard

Topics: State space reduction

State space reduction

TUTORIAL PROBLEMS

In problems for which dynamic programming solutions are considered, there is a concept of a state. A state is, in general, a point in a d-dimensional space, where d is called the number of dimensions in the solution. This may sound quite formal, but in fact, each person who solved at least one problem using dynamic programming approach used this concept.

For example, in **Longest Common Subsequence** problem, also known as **LCS**, the goal is for given two strings $S[1..n]$ and $T[1..m]$, to find the length of their longest common subsequence. This is a classical problem which can be solved efficiently using dynamic programming approach. The idea is to compute $f[i][j]$, which is the length of the longest common subsequence of $S[1..i]$ and $T[1..j]$. The value of $f[i][j]$ can be computed in this problem knowing the values of $f[i-1][j]$, $f[i][j-1]$ and $f[i-1][j-1]$. Then the final answer is the value of $f[n][m]$. In this approach, an entry in f table is called a state, and since the table is 2-dimensional, this dynamic programming solution is 2-dimensional.

The above problem is very classical, so the above approach is well known and it can be shown that it is optimal. In other words, in a general case the above $O(n \cdot m)$ solution with $O(n \cdot m)$ states is optimal. However, when a new problem has to be solved from the scratch, the common situation is that at first some correct dynamic programming approach is defined, just to check if the problem can be solved correctly with this method. After that, the next step is to try to reduce the time complexity of the solution if it is too slow.

The time complexity of a dynamic programming approach can be improved in many ways. The most common are to either use some kind of data structure like a segment tree to speed up the computation of a single state or trying to reduce the number of states needed to solve the problem. Describing the latter is the main goal of this article.

Reducing the number of states

The number of states in a dynamic programming solution can be reduced in a few ways. One possible way is to try to reduce the size of one of existing dimensions. When the size is reduced by a factor of 2, then the overall time complexity of the solution remains the same, but the constant factor is improved. The most significant improvement can be achieved by getting rid of a dimension completely or reducing its size to a small constant.

The actual method of reducing the state space is very problem-specific. There is no general method, which can be applied to all the problems. However, one general idea worth to mention is that the number of dimensions can be reduced if there are some dimensions capturing the same information, or if there is a dimension capturing irrelevant information. Studying more and more war stories is the best approach to learn how to improve existing dynamic programming solutions.

In the following section, an example problem will be given, for which the initial solution will be improved.

Problem:

For a given array of distinct n integers $A[1..n]$ and two integers X and Y, the goal is to find the number of subsequences of A with exactly X local minimums and Y local maximums. For a subsequence $A[i_1], A[i_2], \ldots, A[i_k]$ of A, element $A[i_j]$ for $1 < i_j < k$ is a local minimum if and only if $A[i_{j-1}] > A[i_j] < A[i_{j+1}]$. Similarly, $A[i_j]$ is a local maximum if and only if $A[i_{j-1}] < A[i_j] > A[i_{j+1}]$.

Since subsequences can be build incrementally, dynamic programming approach to this problem sounds promising.

Initial solution

Let $dp[i][x][y][0]$ be the number of subsequences of $A[1..i]$ ending in $A[i]$ with exactly x local minimums, exactly y local maximums and the last element smaller than the one before last element taken to a subsequence.

Similarly, let $dp[i][x][y][1]$ be the number of subsequences of $A[1..i]$ ending in $A[i]$ with exactly x local minimums, exactly y local maximums and the last element greater than the one before last element taken to a subsequence.

With these values defined, any entry of dp table with the i as the value of its first dimension can be computed using dp entries with values of first dimension smaller than i.

Details of computing a single state

Let i be the fixed value of the first dimension of a dp state. In order to update any value for such a state, a possible method is to iterate over all indices $1 \leq j < i$, and for a fixed j, consider two possibilities: either $A[j] < A[i]$ or $A[j] > A[i]$. Since all elements of A are distinct, equality is not possible here.

If $A[j] < A[i]$, a subsequence of length 2 can be created using just $A[j]$ and $A[i]$, so this subsequence can be added to $dp[i][0][0][1]$. Similarly, if $A[j] > A[i]$, such a subsequence can be added to $dp[i][0][0][0]$.

Next step is to extend any previously counted subsequences of length at least 2 to subsequences ending with $A[i]$. In order to do this, let $0 \leq x \leq X$ be the fixed number of local minimums and $0 \leq y \leq Y$ be a fixed number of local maximums. Again, there are two cases to consider:

If $A[j] < A[i]$, a value of $dp[j][x][y][1]$ can be added to $dp[i][x][y][1]$. Moreover, if $x < X$, a value of $dp[j][x][y][0]$ can be added to $dp[i][x+1][y][1]$ creating a new local minimum at index j.

Analogously, if $A[j] > A[i]$, a value of $dp[j][x][y][0]$ can be added to $dp[i][x][y][0]$. Moreover, if $y < Y$, a value of $dp[j][x][y][1]$ can be added to $dp[i][x][y+1][0]$ creating a new local maximum at index j.

In order to compute the final result, all values $dp[i][X][Y][0]$ and $dp[i][X][Y][0]$ over all $1 \leq i \leq N$ have to be summed. At the end, if $X = 0$ and $Y = 0$, a value of N have to be added to the result in order to count all subsequences of length 1.

The total time complexity of this method is $O(N^2 \cdot X \cdot Y)$, because $O(N \cdot X \cdot Y)$ dp entries are filled, and a single one is filed in $O(N)$ time.

Improving the initial solution by reducing the state space

In the above solution, the state space can be reduced. The basic idea here is based on the fact that input values in array A are distinct.

Let P be any subsequence of A and let D be the difference between local minimums in P and local maximums in it. The crucial observation here is that since all elements of A are distinct, D can be only equal to -1, 0 or 1.

In order to prove that, let P be a subsequence of A with 1 local minimum and 0 local maximums. Let i be the index which is the local minimum. The important fact is that P is an increasing subsequence from index i, so any other subsequence W which have P as a prefix will be either increasing from index i till the end, or it will have two adjacent indexes i_j and i_{j+1} such that $A[i_j] > A[i_{j+1}]$, so the value D for W never grows over 1. The same argument can be used to show why the value of D is never less than -1.

This fact can be used to reduce the state space in the original solution, which leads to speed up the overall solution. After the improvement, dp non-constant size dimensions has been reduced from 3 to 2. In other words, for each state with $0 \leq x \leq X$ local minimums there are at most 3 possible values of y - the number of local maximums: the only possible ones are $x-1$, x and $x+1$. This observation leads to a solution with total time complexity $O(N^2 \cdot X)$.

This war story shows how the idea of removing one of two dimensions capturing the same information can be used by a deep analysis of the problem.

It is worth mentioning that even the above faster solution can be improved further using a data structure like a segment tree in order to speed up computing a single state of the solution.

Contributed by: pkacprzak

TEST YOUR UNDERSTANDING

Local Minimum-Maximum

For a given array of distinct n integers $A[1..n]$ and two integers X and Y, the goal is to find the number of subsequences of A with exactly X local minimums and Y local maximums. For a subsequence $A[i_1], A[i_2], \ldots, A[i_k]$ of A, element $A[i_j]$ for $1 < i_j < k$ is a local minimum if and only if $A[i_{j-1}] > A[i_j] < A[i_{j+1}]$. Similarly, $A[i_j]$ is a local maximum if and only if $A[i_{j-1}] < A[i_j] > A[i_{j+1}]$.

Since the result can be very big, the goal is to return it modulo $10^9 + 9$.

Input format:

In the first line there are 3 integers n, X, Y, denoting correspondently the size of the input array, the number of local minimums in desired subsequences, and the number of local maximums in desired subsequences.

In the second line, there are n space separated integers denoting the array A.

Output format:

In one line, output a single integer denoting the number of subsequences of array A with exactly X local minimums and exactly Y local maximums computed modulo $10^9 + 9$.

Constraints:

$3 \leq n \leq 1000$
$0 \leq X, Y \leq 50$

SAMPLE INPUT

```
4 0 1
1 3 5 4
```

SAMPLE OUTPUT

```
3
```

Enter your code or Upload your code as file. Save C (gcc 5.4.0)

```
/*
// Sample code to perform I/O:

scanf("%s", name);              // Reading input from STDIN
printf("Hi, %s.\n", name);      // Writing output to STDOUT

// Warning: Printing unwanted or ill-formatted data to output will cause the test case
```

```
 8  */
 9
10  // Write your code here
11
```

1:1

Provide custom input

Press Ctrl-space for autocomplete suggestions.

COMPILE & TEST SUBMIT

COMMENTS (3) SORT BY: Relevance

Login/Signup to Comment

Anshul Kothari a year ago

I don't understand, for a subsequence of two elements, local maxima and minima not possible because always one would be minima and other would be maxima. If we consider only subsequence of 3 elements and more, then only there are 2 valid ones: [1 3 5 4] and [3 5 4].. Then how is the answer = 3???

0 votes ● Reply ● Message ● Permalink

Sai Vikas D a year ago

[1 5 4] is also valid. You've mistaken subsequences for subarrays. A sub array is a contiguous part of the array. A subsequence contains elements of the array which need not be contiguous but they should be in the order they appear in the original array.

4 votes ● Reply ● Message ● Permalink

Anshul Kothari a year ago

got it, thanks !

0 votes ● Reply ● Message ● Permalink