Competitive programming

wu

September 27, 2022

Contents

1 Dynamic Programming

1.1 General

Problem 1.1.1 (LeetCode: Find All Good Indices). You are given a *0-indexed* integer array nums of size n and a positive integer k.

We call an index i in the range $k \leq i < n - k$ good if the following conditions are satisfied:

- The k elements that are just before the index i are in non-increasing order.
- The k elements that are just after the index i are in non-decreasing order.

Return an array of all good indices sorted in increasing order.

Solution. For j, suppose the non-increasing elements before j (including j) is $left_j$, the non-decreasing elements after j (including j) is $right_j$, then i is good iff $left_{i-1} \geq k$ and $right_{i+1} \geq k$ □

1.2 Digit DP

Problem 1.2.1 (LeetCode 788: Rotated Digits). An integer x is a **good** if after rotating each digit individually by 180 degrees, we get a valid number that is different from x. Each digit must be rotated - we cannot choose to leave it alone.

A number is valid if each digit remains a digit after rotation. For example:

- 0, 1, and 8 rotate to themselves,
- 2 and 5 rotate to each other (in this case they are rotated in a different direction, in other words, 2 or 5 gets mirrored)
- 6 and 9 rotate to each other, and
- the rest of the numbers do not rotate to any other number and become invalid.

Given an integer n, return the number of good integers in the range [1, n].

Solution. Given n. Let $f(pos, bound, diff)$ be the number of good numbers satisfying

1. Only consider posth digit and pos starts from left, which means 0th digit is the highest digit. And we assume the first $pos - 1$ digits are fixed
2. If digits in $[0, pos - 1]$ are first pos digits of n, then $bound$ is `true`
3. If digits in $[0, pos - 1]$ has at least one 2/5/6/9, then $diff$ is `true`

Therefore the answer is $f(0, true, false)$, and the transition formula is

$$f(pos, bound, diff) = \sum f(pos + 1, bound', diff')$$

- $bound'$ is true iff $bound$ is true and the digit we choose is the posth digit of n
- $diff'$ is true iff $diff$ is true or we chose 2/5/6/9

□

2 Graph

2.1 Union find

Problem 2.1.1 (LeetCode: Number of Good Paths). There is a tree (i.e. a connected, undirected graph with no cycles) consisting of n nodes numbered from 0 to $n-1$ and exactly $n-1$ edges.

You are given a 0-indexed integer array `vals` of length n where `vals[i]` denotes the value of the ith node. You are also given a 2D integer array edges where `edges[i] = [ai,bi]` denotes that there exists an undirected edge connecting nodes a_i and b_i.

A good path is a simple path that satisfies the following conditions:

1. The starting node and the ending node have the same value.
2. All nodes between the starting node and the ending node have values less than or equal to the starting node (i.e. the starting node's value should be the maximum value along the path).

Return the number of distinct good paths.

Note that a path and its reverse are counted as the same path. For example, `0 -> 1` is considered to be the same as `1 -> 0`. A single node is also considered as a valid path.

Solution. First, to solve the problem, we can enumerate the paths from the nodes with largest `vals`, and then delete these nodes and continue; this requires $O(n^2)$ time

If we reverse the direction, we are merging nodes with values from low to high, so what comes to our mind? Union find.

For each node s and its neighbor t:

1. if `vals[s]<vals[t]`, then pass
2. if `vals[s]=vals[find[t]]`, then add `size[find[s]]*size[find[t]]`
3. merge `s` and ~~t

□

3 General

3.1 Intervals

Problem 3.1.1 (LeetCode: Count Days Spent Together). Alice and Bob are traveling to Rome for separate business meetings.

You are given 4 strings `arriveAlice`, `leaveAlice`, `arriveBob`, and `leaveBob`. Alice will be in the city from the dates `arriveAlice` to `leaveAlice` (inclusive), while Bob will be in the city from the dates `arriveBob` to `leaveBob` (inclusive). Each will be a 5-character string in the format "MM-DD", corresponding to the month and day of the date.

Return the total number of days that Alice and Bob are in Rome together.

You can assume that all dates occur in the same calendar year, which is not a leap year. Note that the number of days per month can be represented as: [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31].

Solution. First we can convert the string to ith day of the year, then Alice's interval is $[a, b]$, Bob's interval is $[c, d]$, then we need to calculate the intersection of these intervals.

$[a, b] \cap [c, d] \neq \emptyset$ iff $b \geq c \wedge d \geq a$.

$[a, b] \cap [c, d] = \min(b, d) - \max(a, c) + 1$ □

3.2 Bit operation

Problem 3.2.1 (Leetcode: Missing Two LCCI). You are given an array with all the numbers from 1 to N appearing exactly once, except for two number that is missing. How can you find the missing number in $O(N)$ time and $O(1)$ space?

You can return the missing numbers in any order.

Input	Output
`[1]`	`[2,3]`
`[2,3]`	`[1,4]`

`nums.length <= 30000`

Solution. Suppose the missing two numbers are x_1 and x_2, and if we add $1, \ldots, N$ to the end of the array A, then $x = \bigoplus A = x_1 \oplus x_2$.

By `x&-x` we can get the lowest bit of x, assume it's in lth bit. Then we can assume x_1's lth bit is 0, and x_2's lth bit is 1, and we can partition A into A_1 and A_2 by whether the elements' lth bit is 1, then $\bigoplus A_1 = x_1$ and $\bigoplus A_2 = x_2$ □

Problem 3.2.2 (LeetCode: Find a Value of a Mysterious Function Closest to Target).

```
func(arr, l, r) {
  if (r < l) {
    return -10000000000;
  }
  ans = arr[l];
  for (i = l + 1; i <= r; i++) {
    ans = ans & arr[i];
  }
  return ans;
}
```

Winston was given the above mysterious function `func`. He has an integer array `arr` and an integer `target` and he wants to find the values `l` and `r` that make the value `|func(arr, l, r) - target|` minimum possible.

Return the minimum possible value of `|func(arr, l, r) - target|`.

Notice that func should be called with the values l and r where `0 <= l, r < arr.length`.

Constraints:

- `1 <= arr.length <= 10^5`
- `1 <= arr[i] <= 10^6`
- `0 <= target <= 10^7`

Solution. If we fix r

- f is a non-decreasing function
- there is at most 20 different values for $f(arr, l, r)$ as $arr[r] \leq 10^6 < 2^{20}$, since from right to left, 0 won't be transformed into 1

□

Problem 3.2.3 (LeetCode: Smallest Subarrays With Maximum Bitwise OR). You are given a *0-indexed* array nums of length n, consisting of non-negative integers. For each index i from 0 to $n-1$, you must determine the size of the minimum sized non-empty subarray of nums starting at i (inclusive) that has the maximum possible bitwise OR.

Return an integer array answer of size n where answer[i] is the length of the minimum sized subarray starting at i with maximum bitwise OR.

A subarray is a contiguous non-empty sequence of elements within an array.

Solution. Induction and we build a new array $A = \{a_i : a_i = nums[i]\}$. In the ith round, for each $j < i$, check whether $a_j|a_i > a_j$. If so, $a_j|a_i$ is the new possible maximum for a_j and the possible $answer[j] \geq i - j + 1$.

If $a_j|a_i = a_j$, then $a_i \subseteq a_j$ in the sense of bits and for each $k < j$, $a_k|a_i = a_k|a_j$. So we don't need to consider $k < j$ □

3.3 Trick

Problem 3.3.1 (LeetCode: Minimum Money Required Before Transactions). You are given a 0-indexed 2D integer array transactions, where `transactions[i] = [costi, cashbacki]`.

The array describes transactions, where each transaction must be completed exactly once in some order. At any given moment, you have a certain amount of money. In order to complete transaction i, `money >= cost_i` must hold true. After performing a transaction, money becomes `money-cost_i+cashback_i`.

Return the minimum amount of money required before any transaction so that all of the transactions can be completed regardless of the order of the transactions.

Solution. The worst case is, we put money-losing transaction first and then put the transaction with highest cost after it (erase the transaction before if necessary, and assume its index is i)

Suppose $total$ is the total lose, then if the transaction is money-losing, then the money we need is

$$total - (cost[i] - cashback[i]) + cost[i] = total + cashback[i]$$

Otherwise

$$total + cost[i]$$

□