Competitive Programming

Ivan Grujic

September 2018

Contents

1 Leaders in an array 4

2 Kadane's Algorithm 5

3 Missing Number in Array 6

4 Trapping Rain Water 7

5 Maximum of all subarrays of size k 8

6 Next Larger Element 9

7 Towers 10

8 Finding Team Member 11

9 Mega-city 12

10 Find Pair 13

11 Two Heaps 14

12 Inversion Count 15

13 Largest Even Number 16

14 Firing Employees 17

15 Check for BST 18

16 Preorder Traversal and BST 19

17 Maximum Path Sum 20

18 Ilya and Queries 21

19 Alice, Bob and Chocolate 22

20 Number of Ways **23**

21 Little Girl and Maximum Sum **24**

22 Update the Array **25**

23 Nested Segments **26**

24 Pashmak and Parmida's Problem **27**

25 Powerful Array **28**

26 Trees and Queries **29**

27 Nested Segments (Segment Tree) **30**

28 Pashmak and Parmida's Problem (Segment Tree) **31**

29 Circular RMQ **32**

30 X Total Shapes **33**

31 Bipartite Graph **34**

32 Fox Names **35**

33 Learning Languages **37**

34 Checkposts **38**

35 Minimum Spanning Tree **39**

36 Strange Food **40**

37 N Meetings in one Room **42**

38 Chat Room **43**

39 Magic Number **44**

40 Wilbur and Array **45**

41 Alternative Thinking **46**

42 Lexicographically Maximum Subsequence **48**

43 Woodcutters **49**

44 Queue **50**

45 IWGBS **52**

46 Longest Common Subsequence **53**

47 0-1 Knapsack **54**

48 Longest Increasing Subsequence **55**

49 Minimum Number of Jumps **56**

50 Edit Distance **57**

51 Longest Bitonic Sequence **58**

52 Subset Sum Problem **59**

53 Vertex Cover **60**

54 Longest Palindromic Subsequence **61**

55 Misha and Forest **62**

56 Longest Prefix Suffix **63**

57 Shift the String **64**

58 New Distinct Substrings **65**

Problem 1

Leaders in an array

Problem Statement

Write a program to print all the LEADERS in the array. An element is leader if it is greater than all the elements to its right side. The rightmost element is always a leader.

Solution

Scan the array from right to left and output the element at array position i iff it contains the element which is greater or equal to the maximum element encountered in the slice $[i + 1 : n - 1]$.

Time and Space Complexity

The time complexity is linear, the space is constant.

Problem 2

Kadane's Algorithm

Problem Statement

Given an array containing both negative and positive integers. Find the contiguous sub-array with maximum sum.

Solution

Scan the array from left to right and repeat the following:

1. Decide between starting a new subarray or continue the current one. This is done by taking the maximum between the current subarray sum plus the current element and the current element alone. In formula:

   ```
   current_sum = max(current_sum + A[i], A[i])
   ```

2. Update the best solution encountered until now. This is done by taking the maximum between the current best solution and the current solution updated in the previous point. In formula:

   ```
   best_sum = max(best_sum, current_sum)
   ```

Time and Space Complexity

The time complexity is linear, the space is constant.

Problem 3

Missing Number in Array

Problem Statement

Given an array of size $n-1$ and given that there are numbers from 1 to n with one missing, the missing number is to be found.

Solution

The solution is a very simple application of the Gauss formula. The Gauss formula computes the sum s of the first n natural numbers in the following manner:

$$s = \frac{n(n+1)}{2}$$

Now of course if we know that in the array there is a missing number in the range $[1 : n]$, all we need to do is to subtract the sum of the array elements from the above number s. The result is the missing number.

Time and Space Complexity

The time complexity is linear, the space is constant.

Problem 4

Trapping Rain Water

Problem Statement

Given n non-negative integers in array representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

Solution

Scan the array from left to right from position i and accumulate $A[j]$ in a until it is smaller than $A[i]$. When a number greater than $A[i]$ is found, add to the final counter the quantity $(j - i - 1) * A[i] - a$. Then i becomes $j + 1$ and j is incremented by 1. The procedure is repeated until both i and j are smaller than n. The special case verifies when j reaches n while searching for $A[j] > A[i]$. In that case we reverse the procedure from $n - 1$ to i.

Time and Space Complexity

The time complexity is linear (2n to be exact), the space is constant.

Problem 5

Maximum of all subarrays of size k

Problem Statement

Given an array and an integer k, find the maximum for each and every contiguous subarray of size k.

Solution

The idea is to employ a queue that allows the emplacement from the front and behind, i.e. a `deque`. This data structure will contain the maximal elements in decreasing order in the current k-window. As the sliding window slides from left to right the deque will update accordingly. If the max element gets out from the window, it is poped out and the next element is chosen to be the new max. Then the current element is processed. If it is greater than the first element in the queue, the queue is cleared and the current element is emplaced from the front. Otherwise the back is poped out until a greater than current element is found. The max then gets output.

Time and Space Complexity

The time complexity of this solution is linear, since each element is emplaced and removed once in the queue. Moreover the vector is traversed only once. The space is $O(k)$ since at most k elements can be in the queue.

Problem 6

Next Larger Element

Problem Statement

Given an array A having distinct elements, the task is to find the next greater element for each element of the array in order of their appearance in the array. If no such element exists, output -1.

Solution

We scan the array from right to left and employ a stack data structure for the processing. The stack is used to hold the element in increasing order. Each time the ith element of A is processed the following procedure is applied:

- pop the top of the stack until the current element is greater than the top of the stack
- output the top of the stack
- push on top of the stack the current element

Time and Space Complexity

The time complexity of this solution is linear, since each element is pushed and poped once in the stack. Moreover the vector is traversed only once. The space is $O(n)$ since at most n elements can be in the stack. **Note** that we reuse the input vector to store the next larger elements.

Problem 7

Towers

Problem Statement

Little Vasya has received a young builder's kit. The kit consists of several wooden bars, the lengths of all of them are known. The bars can be put one on the top of the other if their lengths are the same. Vasya wants to construct the minimal number of towers from the bars. Help Vasya to use the bars in the best way possible.

Solution

We employ a counter array to count the frequency of each length. Once the input array has been scanned, the counter array is scanned in order to compute the maximum frequency (max height) and the number of towers (number of non zero elements).

Time and Space Complexity

The time complexity in this case is always $\Theta(1000)$ since $n \leq 1000$. Generally, the time complexity would be $O(max(n, 1000))$. The space is $\Theta(1000)$.

Problem 8

Finding Team Member

Problem Statement

There is a programming contest named SnakeUp, $2n$ people want to compete for it. In order to attend this contest, people need to form teams of exactly two people. You are given the strength of each possible combination of two people. All the values of the strengths are distinct.

Every contestant hopes that he can find a teammate so that their team's strength is as high as possible. That is, a contestant will form a team with highest strength possible by choosing a teammate from ones who are willing to be a teammate with him/her. More formally, two people A and B may form a team if each of them is the best possible teammate (among the contestants that remain unpaired) for the other one.

Can you determine who will be each person's teammate?

Solution

The pairs in input are read and sorted in decreasing order. The list of ordered pairs is then scanned and each time both participant of the pair are free (that is they do not already have a team member), they get paired.

Time and Space Complexity

The time complexity is $O(n \log n)$ (being n the number of pairs), due to sorting of the pairs. The space is linear.

Problem 9

Mega-city

Problem Statement

The administration of the Tomsk Region firmly believes that it's time to become a mega-city (that is, get population of one million). Instead of improving the demographic situation, they decided to achieve its goal by expanding the boundaries of the city.

The city of Tomsk can be represented as point on the plane with coordinates (0; 0). The city is surrounded with n other locations, the ith one has coordinates (x_i, y_i) with the population of k_i people. You can widen the city boundaries to a circle of radius r. In such case all locations inside the circle and on its border are included into the city.

Your goal is to write a program that will determine the minimum radius r, to which is necessary to expand the boundaries of Tomsk, so that it becomes a mega-city.

Solution

We read the cities and then sort them in increasing order of their radius. The ordered vector is then scanned from left to right and while the population of the mega city is smaller than 1M we add the population of the currently processed city.

Time and Space Complexity

The time complexity is $O(n \log n)$ due to the sorting of the cities. The space complexity is constant.

Problem 10

Find Pair

Problem Statement

You've got another problem dealing with arrays. Let's consider an arbitrary sequence containing n (not necessarily different) integers $a_1, a_2, ..., a_n$. We are interested in all possible pairs of numbers $(a_i,\ a_j)$, $(1 \leq i, j \leq n)$. In other words, let's consider all n^2 pairs of numbers, picked from the given array. Let's sort all resulting pairs lexicographically by non-decreasing. Let us remind you that pair $(p_1,\, q_1)$ is lexicographically less than pair $(p_2,\, q_2)$ only if either $p_1 < p_2$, or $p_1 = p_2$ and $q_1 < q_2$. Let's number all the pair in the sorted list from 1 to n^2. Your task is formulated like this: you should find the kth pair in the ordered list of all possible pairs of the array you've been given.

Solution

Sort the initial array of n numbers and then do the following:

- The first number of the couple is on index $i = (k-1)/n$: $k-1$ because pair numeration starts from 1 and we divide by n since we are not generating all the pairs, thus we need to map n^2 to n
- The second couple is on index $j = \frac{(k-1)-ln}{r}$, where l is the number of numbers that precede a_i in the ordered array of input numbers and r is the number of occurrences of a_i. This formula is due to the repeated numbers and the idea is to change k from the input accordingly to this.

Time and Space Complexity

The time complexity is $O(n \log n)$ due to the sorting of the numbers. The space complexity is constant.

Problem 11

Two Heaps

Problem Statement

Valera has $2n$ cubes, each cube contains an integer from 10 to 99. He arbitrarily chooses n cubes and puts them in the first heap. The remaining cubes form the second heap.

Valera decided to play with cubes. During the game he takes a cube from the first heap and writes down the number it has. Then he takes a cube from the second heap and write out its two digits near two digits he had written (to the right of them). In the end he obtained a single four-digit integer - the first two digits of it is written on the cube from the first heap, and the second two digits of it is written on the second cube from the second heap.

Valera knows arithmetic very well. So, he can easily count the number of distinct four-digit numbers he can get in the game. The other question is: how to split cubes into two heaps so that this number (the number of distinct four-digit integers Valera can get) will be as large as possible?

Solution

Sort the numbers in input and then do the following: scan the array and take apart the occurrences of numbers repeated more than 2 times (in other words, leave at most two occurrences of each number). Assign then the heap id to these remaining numbers in alternative manner. The numbers left apart are then distributed equally among the two heaps.

Time and Space Complexity

The time complexity is $O(n \log n)$ due to the sorting of the numbers. The space complexity is linear.

Problem 12

Inversion Count

Problem Statement

Let $A[0, ..., n-1]$ be an array of n distinct positive integers. If $i < j$ and $A[i] > A[j]$ then the pair (i, j) is called an inversion of A. Given n and an array A your task is to find the number of inversions of A.

Solution

Merge Sort modified with the proper counting. Note that the computational complexity of sorting algorithms like merge sort can be studied also from the number of inversion needed in order to obtain the sorted list. Comparisons imply inversions.

Time and Space Complexity

The time complexity is $O(n \log n)$ due to the sorting of the array. The space complexity linear since this version of merge sort uses a temporary array during the recursion.

Problem 13

Largest Even Number

Problem Statement

As usual Babul is back with his problem but this time with numbers. In his problem there is a number P (always a whole number) with N digits. Now he started finding out the largest possible even number formed by rearranging this N digit number. For example consider number 1324, after rearranging the digits the largest even number possible is 4312.

Note: In case the number does not contain any even digit then output the largest odd number possible.

Solution

Scan the whole number in input and count the single digits. Scan then the array of counters backwards and build the final n digit number. If during the first scan an even number has been encountered (the minimum one), spare it for the last position.

Time and Space Complexity

The time complexity is $O(n)$ where n is the number of digits of the input number. The space complexity is constant: just counters.

Problem 14

Firing Employees

Problem Statement

Mr. Alfred is the founder of FooLand Constructions. He always maintains a "Black list" of potential employees which can be fired at any moment without any prior notice.

This company has N employees (including Mr. Alfred), and each employee is assigned a rank ($1 \leq \text{rank} \leq N$) at the time of joining the company. Any person can have any rank from 1 to N. No two persons have the same rank. The company has a hierarchically organised management. Each employee has one immediate senior but can have any number of seniors. If a person A is the senior of B and B is senior of C, this implies that A is also a senior of C. Note that Mr. Alfred has no senior to him.

Mr. Alfred has a strange and unfair way of evaluating an employee's performance. Those employees the sum of whose rank and the number of his/her seniors is a prime number is put up on the "Black list".

You are required to find out the number of "Black listed" employees.

Note: The list won't contain Mr. Alfred's name as he is the founder of the company and the list is managed by him!

Solution

BFS on the tree plus primality check.

Time and Space Complexity

The time complexity is $O(n)$ but the time is heavily penalised by primality check. Space is also linear.

Problem 15

Check for BST

Problem Statement

Given a binary tree, return true if it is BST, else false

Solution

Collect the numbers with an in-order traversal and check if they are sorted.

Time and Space Complexity

Both time and space complexity are $O(n)$.

Problem 16

Preorder Traversal and BST

Problem Statement

Given an array, write a program that prints 1 if given array can represent preorder traversal of a BST, else prints 0.

Solution

The idea is to use a stack to hold the nodes during an imaginary dfs. A root variable will represent the node whose left part has already been processed and whose right part is currently being processed. Thus if we reach a point where the current number is less then the value in the current root, it will mean that the list of numbers does not represent the preorder traversal. At each iteration we pop from the stack the nodes whose values are smaller then the currently examined node. The last poped element becomes the new root.

Time and Space Complexity

The time complexity is linear and so is the space.

Problem 17

Maximum Path Sum

Problem Statement

Given a binary tree in which each node element contains a number. Find the maximum possible sum from one leaf node to another.

Solution

The idea is the following: at each node we take the maximum between the maximum path in the left child, the maximum path in the right child and the maximum path from one leaf in the left child to one leaf in the second child. This is done by traversing the tree and returning from each node two values: one that represent the maximum path from a that node to a leaf and one that is the maximum path from one leaf to another in the tree rooted in that node.

Time and Space Complexity

The time complexity is linear in the number of nodes and the space is constant due to the recursion.

Problem 18

Ilya and Queries

Problem Statement

Ilya the Lion wants to help all his friends with passing exams. They need to solve the following problem to pass the IT exam.

You've got string $s = s_1, s_2, ...s_n$ (n is the length of the string), consisting only of characters "." and "#" and m queries. Each query is described by a pair of integers (l_i, r_i) $(1 \leq l_i < r_i \leq n)$. The answer to the query (l_i, r_i) is the number of such integers i $(l_i \leq i < r_i)$, that $s_i = s_{i+1}$.

Ilya the Lion wants to help his friends but is there anyone to help him? Help Ilya, solve the problem.

Solution

We use a bit vector of the same dimension of the input string whose ith position is clear if and only if $s_i \neq s_{i+1}$, otherwise is full. Accordingly, we compute the exclusive prefix sum of this bit array and answer the queries in constant time by making the difference of the two array accesses per query.

Time and Space Complexity

The time complexity is linear in the length of the string and so is the space.

Problem 19

Alice, Bob and Chocolate

Problem Statement

Alice and Bob like games. And now they are ready to start a new game. They have placed n chocolate bars in a line. Alice starts to eat chocolate bars one by one from left to right, and Bob — from right to left. For each chocololate bar the time, needed for the player to consume it, is known (Alice and Bob eat them with equal speed). When the player consumes a chocolate bar, he immediately starts with another. It is not allowed to eat two chocolate bars at the same time, to leave the bar unfinished and to make pauses. If both players start to eat the same bar simultaneously, Bob leaves it to Alice as a true gentleman. How many bars each of the players will consume?

Solution

Keep two pointers, one from left and the other from right end of the array (respectively i and j). Move the two pointers in the following manner:

- Move i by 1 if $a[i] < a[j]$ and increment the counter for Alice. Meanwhile subtract $a[i]$ from $a[j]$.
- Move j by -1 if $a[i] > a[j]$ and increment the counter for Bob. Meanwhile subtract $a[j]$ from $a[i]$.
- If $a[i] = a[j]$ then move both pointers.

Repeat this process until $i < j$. If $i = j$ then Bob leaves the peace of chocolate to Alice, as he is a gentleman.

Time and Space Complexity

The time complexity is linear and the space is constant.

Problem 20

Number of Ways

Problem Statement

You've got array $a[1], a[2], ..., a[n]$, consisting of n integers. Count the number of ways to split all the elements of the array into three contiguous parts so that the sum of elements in each part is the same.

More formally, you need to find the number of such pairs of indices (i, j) $(2 \leq i \leq j \leq n - 1)$, that $\sum_{k=1}^{i-1} a_k = \sum_{k=i}^{j} a_k = \sum_{k=j+1}^{n} a_k$

Solution

The idea is to compute the number of suffixes of the input array that sum to the target sum (input sum / 3) and then use this array to efficiently count the number of ways to split the array in three sum-equal pieces. After the number of suffixes that sum to the target sum has been computed, the input array is re-scanned, this time from left to right, and each time the prefix sum is equal to the target sum, we add to the counter the $[i + 2]$th value of the suffix array previously computed.

Time and Space Complexity

Both time and space complexity are linear.

Problem 21

Little Girl and Maximum Sum

Problem Statement

The little girl loves the problems on array queries very much. One day she came across a rather well-known problem: you've got an array of n elements (the elements of the array are indexed starting from 1); also, there are q queries, each one is defined by a pair of integers (l_i, r_i) $(1 \leq l_i \leq r_i \leq n)$. You need to find for each query the sum of elements of the array with indexes from l_i to r_i, inclusive.

The little girl found the problem rather boring. She decided to reorder the array elements before replying to the queries in a way that makes the sum of query replies maximum possible. Your task is to find the value of this maximum sum.

Solution

The idea is to put large numbers in frequent positions. To this aim, we process the queries and count the position frequencies in an array f. In order do it efficiently we will increment the left end position and decrement the right end plus 1 position (if $< n$). Subsequently, we just need to compute the inclusive prefix sum on f and sum the products $f[i] * a[i]$. The values in f and a are previously sorted.

Time and Space Complexity

Time complexity is $O(n \log n)$ due to sorting. Space is linear since we need to store frequencies.

Problem 22

Update the Array

Problem Statement

You have an array containing n elements initially all 0. You need to do a number of update operations on it. In each update you specify l, r and val which are the starting index, ending index and value to be added. After each update, you add the 'val' to all elements from index l to r. After 'u' updates are over, there will be q queries each containing an index for which you have to print the element at that index.

Solution

The idea is to avoid updating the whole ranges of the input array. To this end we just update the left and right end position of the queries. After that an inclusive prefix sum is performed. In the end, the queries are answered in constant time.

Time and Space Complexity

Time complexity is $O(n + q)$ while the space complexity is $O(n)$.

Problem 23

Nested Segments

Problem Statement

You are given n segments on a line. There are no ends of some segments that coincide. For each segment find the number of segments it contains.

Solution

Order the segments in non non increasing order of left end and then process them one by one. Each ordered segment's right end r is inserted into a BIT. Prior the inserting, BIT is queried for the sum until r.

Time and Space Complexity

Time complexity is $O(n \log n)$ since ordering is $O(n \log n)$ and BIT processing is overall $O(n \log n)$. Space is $O(n)$.

Problem 24

Pashmak and Parmida's Problem

Problem Statement

Parmida is a clever girl and she wants to participate in Olympiads this year. Of course she wants her partner to be clever too (although he's not)! Parmida has prepared the following test problem for Pashmak.

There is a sequence a that consists of n integers $a_1, a_2, ..., a_n$. Let's denote $f(l, r, x)$ the number of indices k such that: $l \leq k \leq r$ and $a_k = x$. His task is to calculate the number of pairs of indices (i, j) $(1 \leq i < j \leq n)$ such that $f(1, i, a_i) > f(j, n, a_j)$.

Help Pashmak with the test.

Solution

We compute two arrays: array *right* that stores at ith position the number of a_is from i to n and array *left* that stores the number of a_is from 0 to i. This is achieved in linear time by using an unordered map. After that we scan the two arrays from right to left and at each iteration we insert into BIT *right*$[i]$ and we add to the counter the sum until position *left*$[i-1]-1$.

Time and Space Complexity

Time complexity is $O(n \log n)$ due to BIT. Space is linear.

Problem 25

Powerful Array

Problem Statement

An array of positive integers $a_1, a_2, ..., a_n$ is given. Let us consider its arbitrary subarray $a_l, a_{l+1} ..., a_r$, where $1 \leq l \leq r \leq n$. For every positive integer s denote by K_s the number of occurrences of s into the subarray. We call the power of the subarray the sum of products $K_s K_s s$ for every positive integer s. The sum contains only finite number of nonzero summands as the number of different values in the array is indeed finite.
You should calculate the power of t given subarrays.

Solution

Mo's algorithm. Add and remove functions use a table that stores frequencies in a current range. This table is updated accordingly as the boundaries of queries move.

Time and Space Complexity

Time complexity $O((n+m)\sqrt{n})$ where n is the array length and m is the number of queries.

Problem 26

Trees and Queries

Problem Statement

You have a rooted tree consisting of n vertices. Each vertex of the tree has some color. We will assume that the tree vertices are numbered by integers from 1 to n. Then we represent the color of vertex v as c_v. The tree root is a vertex with number 1.
In this problem you need to answer to m queries. Each query is described by two integers v_j, k_j. The answer to query v_j, k_j is the number of such colors of vertices x, that the subtree of vertex v_j contains at least k_j vertices of color x.

Solution

Mo's algorithm. We first build the tree and then dfs it in order to flat it to an array by means of a preorder traversal. The queries are then read and ordered according to usual Mo's order. We employ then two arrays, one for counting the frequencies of each colour and one for counting how many colours has particular number of vertices in the current query range. This gives us the possibility to answer the queries in constant time once the range has been set according to the current query.

Time and Space Complexity

Time complexity $O((n+m)\sqrt{n})$ where n is the array length and m is the number of queries.

Problem 27

Nested Segments (Segment Tree)

Problem Statement

You are given n segments on a line. There are no ends of some segments that coincide. For each segment find the number of segments it contains.

Solution

See problem 23 (pg 23). Instead of BIT use Segment Tree.

Time and Space Complexity

See problem 23 (pg 23).

Problem 28

Pashmak and Parmida's Problem (Segment Tree)

Problem Statement

Parmida is a clever girl and she wants to participate in Olympiads this year. Of course she wants her partner to be clever too (although he's not)! Parmida has prepared the following test problem for Pashmak.

There is a sequence a that consists of n integers $a_1, a_2, ..., a_n$. Let's denote $f(l, r, x)$ the number of indices k such that: $l \leq k \leq r$ and $ak = x$. His task is to calculate the number of pairs of indices (i, j) $(1 \leq i < j \leq n)$ such that $f(1, i, a_i) > f(j, n, a_j)$.

Help Pashmak with the test.

Solution

See problem 24 (pg 24). Instead of BIT use Segment Tree.

Time and Space Complexity

See problem 24 (pg 24).

Problem 29

Circular RMQ

Problem Statement

You are given circular array $a_0, a_1, ..., a_{n-1}$. There are two types of operations with it:

- inc(lf,rg,v) - this operation increases each element on the segment [lf,rg] (inclusively) by v
- rmq(lf,rg) - this operation returns minimal value on the segment [lf,rg] (inclusively)

Assume segments to be circular. Write program to process given sequence of operations.

Solution

Segment Tree with lazy propagation. On circular queries execute two different queries.

Time and Space Complexity

Time complexity is $O(n \log n)$ while space complexity is linear.

Problem 30

X Total Shapes

Problem Statement

Given $N \times M$ string array of O's and X's Return the number of 'X' total shapes. 'X' shape consists of one or more adjacent X's (diagonals not included).

Solution

Store the $N \times M$ matrix in an array. Depth first search on this array paying attention on what indices to use during the traversal. Each dfs finds out 1 connected component of X's and thus the result is the number of connected components.

Time and Space Complexity

Both time and space complexity is linear: $O(NM)$ (note, linear in the total length of the string in input).

Problem 31

Bipartite Graph

Problem Statement

Given an adjacency matrix representation of a graph g having 0 based index your task is to complete the function isBipartite which returns true if the graph is a bipartite graph else returns false.

Solution

Depth First Search and alternative colouring mechanism.

Time and Space Complexity

Both time and space complexity is linear in the number of nodes: $O(V)$.

Problem 32

Fox Names

Problem Statement

Fox Ciel is going to publish a paper on FOCS (Foxes Operated Computer Systems, pronounce: ”Fox”). She heard a rumour: the authors list on the paper is always sorted in the lexicographical order.

After checking some examples, she found out that sometimes it wasn’t true. On some papers authors’ names weren’t sorted in lexicographical order in normal sense. But it was always true that after some modification of the order of letters in alphabet, the order of authors becomes lexicographical!

She wants to know, if there exists an order of letters in Latin alphabet such that the names on the paper she is submitting are following in the lexicographical order. If so, you should find out any such order.

Lexicographical order is defined in following way. When we compare s and t, first we find the leftmost position with differing characters: $s_i \neq t_i$. If there is no such position (i.e. s is a prefix of t or vice versa) the shortest string is less. Otherwise, we compare characters s_i and t_i according to their order in alphabet.

Solution

The idea is of topological ordering of the letters of the alphabet. The algorithm scans the list of strings in input and builds a matrix G of size 26×26 to store the ordering of the letters. $G[i][j] = 1$ iff i should precede j in the ordering. Depth first search with cycle detection (three colours) is applied over this matrix in order to find out whether a reordering exists or not.

Time and Space Complexity

The space is constant while the time complexity is linear in the sum of the lengths of the input strings. In other words it is $O(\sum_{i=1}^{n} l_i)$ where l_i is the length of the ith string.

Problem 33

Learning Languages

Problem Statement

The "BerCorp" company has got n employees. These employees can use m approved official languages for the formal correspondence. The languages are numbered with integers from 1 to m. For each employee we have the list of languages, which he knows. This list could be empty, i.e. an employee may know no official languages. But the employees are willing to learn any number of official languages, as long as the company pays their lessons. A study course in one language for one employee costs 1 berdollar.

Find the minimum sum of money the company needs to spend so as any employee could correspond to any other one (their correspondence can be indirect, i.e. other employees can help out translating).

Solution

A group of people that can communicate with each other, even transitively, can be modelled thought connected components. The minimum number of languages to teach equals the number of connected components. Thus an adjacency matrix is built and then classical dfs is run on it. A special case when no one speaks any language is also considered.

Time and Space Complexity

Time is $O(n + m)$ while space is $O(n)$.

Problem 34

Checkposts

Problem Statement

Your city has n junctions. There are m one-way roads between the junctions. As a mayor of the city, you have to ensure the security of all the junctions. To ensure the security, you have to build some police checkposts. Checkposts can only be built in a junction. A checkpost at junction i can protect junction j if either $i = j$ or the police patrol car can go to j from i and then come back to i.

Building checkposts costs some money. As some areas of the city are more expensive than others, building checkpost at some junctions might cost more money than other junctions.

You have to determine the minimum possible money needed to ensure the security of all the junctions. Also you have to find the number of ways to ensure the security in minimum price and in addition in minimum number of checkposts. Two ways are different if any of the junctions contains a checkpost in one of them and do not contain in the other.

Solution

Use Tarjan algorithm to compute strongly connected components. Then for each component take the city with minimum cost. Sum the costs to get the result. As for the number of ways, for each strongly connected components take the number of cities that have the minimum cost. Multiply the single numbers so obtained and module it with 1000000007.

Time and Space Complexity

Time is linear in the number of nodes and edges, due to Tarjan's algorithm: $O(V + E)$. The space is linear in the number of nodes: $O(V)$

Problem 35

Minimum Spanning Tree

Problem Statement

Find the minimum spanning tree of the graph.

Solution

Kruskal algorithm with disjoint set data structure with union by rank with path compression heuristic.

Time and Space Complexity

Time $O(E \log V)$ while space is linear $O(V + E)$.

Problem 36

Strange Food

Problem Statement

There are 3 kinds of animals A,B and C. A can eat B, B can eat C, C can eat A. It's interesting, isn't it? Now we have n animals, numbered from 1 to n. Each of them is one of the 3 kinds of animals: A, B, C.
Today Mary tells us k pieces of information about these n animals. Each piece has one of the two forms below:

- $1xy$: It tells us the kind of x and y are the same
- $2xy$: It tells us x can eat y

Some of these k pieces are true, some are false. The piece is false if it satisfies one of the 3 conditions below, otherwise it's true:

- x or y in this piece is larger than n
- This piece tells us x can eat x
- This piece conflicts to some true piece before

Solution

We use disjoint sets data structure to maintain the information about which couples of animals has been processed for the first time and which not. Each time the two animals in whatever query of the two types in input belong to the same set, we simply check whether the query is valid or not. This is done thanks to a particular information that we store for each animal: relation with it's parent (array s). If the two animals do not belong to the same set, we connect their parents (union) and we update s accordingly.

Time and Space Complexity

The time complexity is basically $O(n + k)$, due to the path compression in find operation. Space is linear.

Problem 37

N Meetings in one Room

Problem Statement

There is one meeting room in Flipkart. There are n meetings in the form of $(S[i], F[i])$ where $S[i]$ is start time of meeting i and $F[i]$ is finish time of meeting i.

What is the maximum number of meetings that can be accommodated in the meeting room?

Solution

Sort the segments in non decreasing order according to their end time. Greedy approach from first one to last one taking those meetings whose start time is past the end time of the previously taken meeting.

Time and Space Complexity

Time is $O(n \log n)$ due to sorting. Space is constant.

Problem 38

Chat Room

Problem Statement

Vasya has recently learned to type and log on to the Internet. He immediately entered a chat room and decided to say hello to everybody. Vasya typed the word s. It is considered that Vasya managed to say hello if several letters can be deleted from the typed word so that it resulted in the word "hello". For example, if Vasya types the word "ahhellllloou", it will be considered that he said hello, and if he types "hlelo", it will be considered that Vasya got misunderstood and he didn't manage to say hello. Determine whether Vasya managed to say hello by the given word s.

Solution

Just scan the input string.

Time and Space Complexity

Time is $O(n)$, space $O(1)$.

Problem 39

Magic Number

Problem Statement

A magic number is a number formed by concatenation of numbers 1, 14 and 144. We can use each of these numbers any number of times. Therefore 14144, 141414 and 1411 are magic numbers but 1444, 514 and 414 are not. You're given a number. Determine if it is a magic number or not.

Solution

Just scan the number from right to left and for each position i do the following:

- if $N[i] = 1$ then check if $N[1 : i-1]$ is a magic number
- if $(N[i] = 4 \wedge N[i-1] = 1$ then check if $N[1 : i-2]$ is a magic number
- if $(N[i] = 4 \wedge N[i-1] = 4 \wedge N[i-2] = 4$ then check if $N[1 : i-3]$ is a magic number

Time and Space Complexity

Time is linear in the input number length and space is constant.

Problem 40

Wilbur and Array

Problem Statement

Wilbur the pig is tinkering with arrays again. He has the array $a_1, a_2, ..., a_n$ initially consisting of n zeros. At one step, he can choose any index i and either add 1 to all elements $a_i, a_{i+1}, ..., a_n$ or subtract 1 from all elements $a_i, a_{i+1}, ..., a_n$. His goal is to end up with the array $b_1, b_2, ..., b_n$. Of course, Wilbur wants to achieve this goal in the minimum number of steps and asks you to compute this value.

Solution

Start from the left and go to the right end of the array computing the at each iteration:

- `a` - an accumulator that stores the accumulative differences $N[i] - a$
- `c` - a counter which adds to itself the absolute value of the difference $N[i] - a$;

The result is in the counter c at the end of the iterations.

Time and Space Complexity

Time linear in the length of the array, and space is constant.

Problem 41

Alternative Thinking

Problem Statement

Kevin has just recevied his disappointing results on the USA Identification of Cows Olympiad (USAICO) in the form of a binary string of length n. Each character of Kevin's string represents Kevin's score on one of the n questions of the olympiad—'1' for a correctly identified cow and '0' otherwise.
However, all is not lost. Kevin is a big proponent of alternative thinking and believes that his score, instead of being the sum of his points, should be the length of the longest alternating subsequence of his string. Here, we define an alternating subsequence of a string as a not-necessarily contiguous subsequence where no two consecutive elements are equal. For example, {0,1,0,1}, {1,0,1}, and {1,0,1,0} are alternating sequences, while {1,0,0} and {0,1,0,1,1} are not.
Kevin, being the sneaky little puffball that he is, is willing to hack into the USAICO databases to improve his score. In order to be subtle, he decides that he will flip exactly one substring—that is, take a contiguous non-empty substring of his score and change all '0's in that substring to '1's and viceversa. After such an operation, Kevin wants to know the length of the longest possible alternating subsequence that his string could have.

Solution

First of all let's say that the alternative subsequence is at most n, the length of the string. Moreover, note that whatever is the substring that we flip, we can earn at most 2 more units in the length w.r.t. the greedy alternative subsequence without any flipping at all. Thus, we compute the length of the greedy alternative subsequence on the input string as it is, let's call this quantity c. We then output the minimum between n and c.

Time and Space Complexity

The time complexity is linear while the space is constant.

Problem 42

Lexicographically Maximum Subsequence

Problem Statement

You've got string s, consisting of only lowercase English letters. Find its lexicographically maximum subsequence.
We'll call a non-empty string $s[p_1p_2...p_k] = s_{p_1}s_{p_2}...s_{p_k}(p_1 \leq p_1 < p_2 < ... < p_k \leq |s|)$ a subsequence of string $s = s_1s_2...s_{|s|}$.
String $x = x_1x_2...x_{|x|}$ is lexicographically larger than string $y = y_1y_2...y_{|y|}$, if either $|x| > |y|$ and $x_1 = y_1, x_2 = y_2, ..., x_{|y|} = y_{|y|}$, or exists such number r ($r < |x|, r < |y|$), that $x_1 = y_1, x_2 = y_2, ..., x_r = y_r$ and $x_r + 1 > y_r + 1$. Characters in lines are compared like their ASCII codes.

Solution

We employ a deque to store the maximal lexicographic sequence. We process the input string from left to right. For each element in string we pop from the front of the queue all the letters that are lexicographically smaller. At the end of this process we emplace in front the current letter. We repeat this process for each element of the input string. The reverse order of the elements in the queue is the answer.

Time and Space Complexity

Both time and space complexity are linear, since each input element is emplaced and poped out from the queue at most once.

Problem 43

Woodcutters

Problem Statement

Little Susie listens to fairy tales before bed every day. Today's fairy tale was about wood cutters and the little girl immediately started imagining the choppers cutting wood. She imagined the situation that is described below.
There are n trees located along the road at points with coordinates $x_1, x_2, ..., x_n$. Each tree has its height h_i. Woodcutters can cut down a tree and fell it to the left or to the right. After that it occupies one of the segments $[x_i - h_i, x_i]$ or $[x_i, x_i + h_i]$. The tree that is not cut down occupies a single point with coordinate x_i. Woodcutters can fell a tree if the segment to be occupied by the fallen tree doesn't contain any occupied point. The woodcutters want to process as many trees as possible, so Susie wonders, what is the maximum number of trees to fell.

Solution

Greedy approach: the first cut always on left and the last always on right. In the middle prefer to cut on left when possible, otherwise cut on right if possible. Process items in order from left to right.

Time and Space Complexity

Both time and space complexity are linear.

Problem 44

Queue

Problem Statement

In the Main Berland Bank n people stand in a queue at the cashier, everyone knows his/her height h_i, and the heights of the other people in the queue. Each of them keeps in mind number a_i — how many people who are taller than him/her and stand in queue in front of him.
After a while the cashier has a lunch break and the people in the queue seat on the chairs in the waiting room in a random order.
When the lunch break was over, it turned out that nobody can remember the exact order of the people in the queue, but everyone remembers his number a_i.
Your task is to restore the order in which the people stood in the queue if it is possible. There may be several acceptable orders, but you need to find any of them. Also, you need to print a possible set of numbers h_i — the heights of people in the queue, so that the numbers a_i are correct.

Solution

First of all we sort the people in non decreasing order of a_i. After that we check in linear time if the there is at least one solution by checking that each sorted number is at most equal to it's position in the sorted list. If this check passes, we proceed further. We use a vector h to store the initial order of the people. For each person i we insert it somewhere from position $i - P[i]$.infront on (where P is the array of people sorted according to a_i values). In worst case this can cost $O(n^2)$. The heights are then assigned in order of people in the queue h in linear time.

Time and Space Complexity

The worst case time complexity is $O(n^2)$ due to the shifting. The space is linear.

Problem 45

IWGBS

Problem Statement

Dor is IWGolymp student so he has to count in how many ways he can make N digit numbers that is formed by ones and zeroes. But zeroes can not be next to each other. Help to him in how many different numbers can he make.

Solution

It is the Fibonacci sequence. The only fact is to use BigInteger class.

Time and Space Complexity

The time is linear in the input number, and so is space.

Problem 46

Longest Common Subsequence

Problem Statement

Given two sequences, find the length of longest subsequence present in both of them. Both the strings are of uppercase.

Solution

Dynamic Programming approach: we use an $n \times m$ matrix where each position (i, j) stores the longest common subsequence for $a[1:i]$ and $b[1:j]$. Very similar to the Edit Distance solution.

Time and Space Complexity

The time complexity is quadratic $O(nm)$ and so is the space.

Problem 47

0-1 Knapsack

Problem Statement

You are given weights and values of N items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack. Note that we have only one quantity of each item. In other words, given two integer arrays $v[0..N-1]$ and $w[0..N-1]$ which represent values and weights associated with N items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of v such that sum of the weights of this subset is smaller than or equal to W. You cannot break an item, either pick the complete item, or don't pick it (0-1 property).

Solution

Dynamic Programming approach: we employ a matrix of size $n \times c$, n is the number of objects and c is the capacity. Each position (i, j) stores the optimal value for objects $[1 : i]$ with capacity j.

Time and Space Complexity

The time complexity is quadratic $O(nc)$ and so is the space.

Problem 48

Longest Increasing Subsequence

Problem Statement

Given a sequence, find the length of the longest increasing subsequence from a given sequence. The longest increasing subsequence means to find a subsequence of a given sequence in which the subsequence's elements are in sorted order, lowest to highest, and in which the subsequence is as long as possible. This subsequence is not necessarily contiguous, or unique.

Solution

Dynamic Programming approach: we employ a vector of size n, where n is the length of the input sequence s. For each position i of the input string s, we process the previous positions one by one ($j \in [0 : i-1]$) and we insert in position i of this vector the maximum between the current value at position i and the jth value plus 1. The maximum value at the end of this double loop is the answer. Basically we could use a matrix $n \times n$, the use of vector is just for the space complexity purposes.

Time and Space Complexity

The time is quadratic $O(n^2)$ while the space is linear.

Problem 49

Minimum Number of Jumps

Problem Statement

Given an array of integers where each element represents the max number of steps that can be made forward from that element. Write a function to return the minimum number of jumps to reach the end of the array (starting from the first element). If an element is 0, then cannot move through that element.

Solution

Dynamic Programming approach: we use a vector A to store in position i the minimum number of jumps from that point to the end of the array. For each position we take the minimum $\min_{0 \leq j \leq N[i]} 1 + A[i+j]$, where N is the input sequence (obviously paying attention on array boundaries). The answer is found at position 0 after the algorithm has terminated.

Time and Space Complexity

The worst case time complexity is quadratic since for each position we could potentially check each successive position until the end of the array. Thus, due to Gauss formula we get $O(n^2)$ complexity; the space is linear.

Problem 50

Edit Distance

Problem Statement

Given two strings str1 and str2 and below operations that can performed on str1. Find minimum number of edits (operations) required to convert str1 into str2.

- Insert
- Remove
- Replace

All of the above operations are of cost=1. Both the strings are of lowercase.

Solution

The well known quadratic and dynamic programming solution for edit distance. Matrix of size $n \times m$.

Time and Space Complexity

Quadratic $O(nm)$ both time and space.

Problem 51

Longest Bitonic Sequence

Problem Statement

Given an array of positive integers. The task is to print the maximum length of Bitonic subsequence. A subsequence of array is called Bitonic if it is first increasing, then decreasing.

Solution

Compute the longest increasing subsequence I of the input sequence; then compute the longest increasing subsequence D of the inverse input sequence. The result is given by the $\max_i I[i] + D[n-i-1] - 1$ (minus 1 is for not counting the same element twice).

Time and Space Complexity

The time and the space are $O(n^2)$ in the input length n.

Problem 52

Subset Sum Problem

Problem Statement

Given a set of numbers, check whether it can be partitioned into two subsets such that the sum of elements in both subsets is same or not.

Solution

If the sum of the input array is not divisible by 2 then output NO, otherwise take the half of the sum and try to reach that target sum with some elements in input by applying the 01 knapsack solution. If it is possible, since the original sum was divisible by 2, automatically we have the answer.

Time and Space Complexity

Time complexity is $O(nc)$ where c is the capacity of the knapsack (original sum divided by 2). The space is the same.

Problem 53

Vertex Cover

Problem Statement

You are given an unweighted, undirected tree. Write a program to find a vertex set of minimum size in this tree such that each edge has as least one of its end-points in that set.

Solution

Depth first search with augmented tree with two values: for each node we store the min vertex cover when taken and when not taken. The result is the minimum between those two values in the root of the input tree after the execution of the depth first search on it.

Time and Space Complexity

Time is the one of the depth first search: $O(V)$. The space is the same.

Problem 54

Longest Palindromic Subsequence

Problem Statement

Given a string, find the longest palindromic subsequence.

Solution

Dynamic Programming approach: we use a square matrix M initialised in a particular way (see code). We start from the bottom row and we go up, each row processed left to right in the following manner: if $S[i-1] = S[j-1]$ then in position $M[i][j]$ we initially assign $2 + M[i+1][j-1]$ (plus 2 is due to the same letter at the extremes of the already known palindromic subsequence). Then in $M[i][j]$ we assign the max between three values: $M[i][j]$, $M[i+1][j]$ and $M[i][j-1]$. The result is in $M[1][n]$.

This solution is cache friendly.

Time and Space Complexity

The time complexity is quadratic $O(n^2)$ and so is the space.

Problem 55

Misha and Forest

Problem Statement

Let's define a forest as a non-directed acyclic graph (also without loops and parallel edges). One day Misha played with the forest consisting of n vertices. For each vertex v from 0 to $n-1$ he wrote down two integers, degreev and sv, were the first integer is the number of vertices adjacent to vertex v, and the second integer is the XOR sum of the numbers of vertices adjacent to v (if there were no adjacent vertices, he wrote down 0).
Next day Misha couldn't remember what graph he initially had. Misha has values degreev and sv left, though. Help him find the number of edges and the edges of the initial graph. It is guaranteed that there exists a forest that corresponds to the numbers written by Misha.

Solution

Since the graph is acyclic it is topologically sortable. Accordingly, we read the input and employ a queue for nodes with degree 1. We then do the following process while the queue of leaves is not empty: we pop the next node from the queue, we xor it's id from the node to which it is connected (the only one), and we decrement the degree of that connected node by one. If the degree of this node becomes 1 it is put in the queue.

Time and Space Complexity

Both time and space complexity are linear in the number of nodes in the worst case.

Problem 56

Longest Prefix Suffix

Problem Statement

Given a string of character, find the length of longest proper prefix which is also a proper suffix.

Solution

Dynamic Programming approach: use a vector to store the longest prefix suffix until position i. The idea is to verify if we can extend the current longest prefix suffix by checking whether the new char is the same as the one after the longest prefix. If yes just go ahead by incrementing the value of the current longest prefix suffix. If not, then we take the previously longest prefix and repeat the check. We unroll until we either find the match or we reach the empty prefix.

Time and Space Complexity

Both time and space complexity are linear.

Problem 57

Shift the String

Problem Statement

You are given two strings A and B of the same length. Each string contains N Lower case Latin character (from 'a' to 'z'). A shift operation will remove the first character of a string and add the same character at the end of that string. For example after you perform a shift operation on a string 'abcd', the new string will be 'bcda'. If you perform this operation two times, the new string will be 'cdab'. You need to use some (maybe none) shift operations on the string B to maximise the length of the longest common prefix of A and B. If more than one result can be found pick the one that use smallest number of shift operations.

Solution

First of all we concatenate the string B with B and then execute a version of Knuth-Morris-Prat on that string searching for the longest prefix of A in it.

Time and Space Complexity

Time and space is linear.

Problem 58

New Distinct Substrings

Problem Statement

Given a string, we need to find the total number of its distinct substrings.

Solution

Compute suffix array and LCP array and then apply the following formula to get the answer:

$$\frac{n^2+n}{2} - \sum_{i=0}^{n-1} lcp[i]$$

Time and Space Complexity

Building the suffix array takes $O(n \log n \log n)$ time while building LCP array takes $O(n)$. The remaining part of the algorithm is linear, thus the time complexity of this solution is $O(n \log n \log n)$. The space is linear.