[Task 1 7](#_Toc151906442)

[01 TheWidestPath 7](#_Toc151906443)

[[Elementary / 30 mins] Given a forest containing N trees, find the width of the widest vertical path that can be built without cutting any tree. 7](#_Toc151906444)

[02 CardPayments 10](#_Toc151906445)

[[Easy / 40 mins] Given a list of transactions within one year, calculate the final balance of an account. Add a fee for each month that did not include at least three card payments for a total sum of at least 100. 10](#_Toc151906446)

[03 CastleBuilding 12](#_Toc151906447)

[[Easy / 60 mins] Find the number of castles that can be built. 12](#_Toc151906448)

[04 CommonLetter 14](#_Toc151906449)

[[Easy / 60 mins] Given an array of strings, find a pair of strings that share the same letter at the same position. 14](#_Toc151906450)

[05 CountBananas 16](#_Toc151906451)

[[Easy / 20 mins] Calculate how many times you can print the word "BANANA" using the letters given in string S. 16](#_Toc151906452)

[06 CreatePalindrome 17](#_Toc151906453)

[[Easy / 20 mins] Replace all question marks in a given string to obtain a palindrome. 17](#_Toc151906454)

[07 DiversityString 18](#_Toc151906455)

[[Easy / 40 mins] Find a string of a given length containing as many different lower-case letters as possible, in which each letter occurs an equal number of times. 18](#_Toc151906456)

[08 EraseOneLetter 19](#_Toc151906457)

[[Easy / 20 mins] Remove one letter from a word so that resulting string will be as small as possible (alphabetically). 19](#_Toc151906458)

[09 EvenPairsOnCycle 20](#_Toc151906459)

[[Easy / 40 mins] Given numbers on a circle, find the maximum number of neighbouring pairs with an even sum. 20](#_Toc151906460)

[10 ForbiddenTriosSwaps 22](#_Toc151906461)

[[Easy / 20 mins] Given a string S consisting of letters 'a' and 'b', return the minimum number of swaps needed to obtain a string with no instances of three identical consecutive letters. 22](#_Toc151906462)

[11 FormatArray 23](#_Toc151906463)

[[Easy / 75 mins] Given an array of integers, output a string that presents the integers in tabular form. 23](#_Toc151906464)

[12 MonitorsDelivery 26](#_Toc151906465)

[[Easy / 40 mins] Given a set of orders, calculate how many of them can be fulfilled. 26](#_Toc151906466)

[13 SameDigitMerge 28](#_Toc151906467)

[[Easy / 30 mins] Count the number of pairs in which the last digit of the first selected number is the same as the first digit of the second selected number. 28](#_Toc151906468)

[14 ShortestUniqueSubstring 29](#_Toc151906469)

[[Easy / 20 mins] Given a string, count the length of the shortest word which occurs in it exactly once. 29](#_Toc151906470)

[15 SmallestDigitSum 30](#_Toc151906471)

[[Easy / 20 mins] Given an integer N, find the smallest integer whose digits sum to N. 30](#_Toc151906472)

[16 ValueOccurrences 31](#_Toc151906473)

[[Easy / 40 mins] Given a sorted array, calculate how many insertion and removal operations are required to make every value X in the array occur exactly X times. 31](#_Toc151906474)

[17 AsphaltPatches 33](#_Toc151906475)

[[Easy / 40 mins] Given a description of a road with potholes, find the minimum number of patches needed to repair them all. One patch repairs three consecutive road segments. 33](#_Toc151906476)

[18 ABString 35](#_Toc151906477)

[[Easy / 5 mins] Check, whether in a given string all letters 'a' occur before all letters 'b'. 35](#_Toc151906478)

[19 EndsTheSame 36](#_Toc151906479)

[[Medium / 20 mins] Count how many times the first and last letters of a string are the same during a sequence of operations. 36](#_Toc151906480)

[Task 2 37](#_Toc151906481)

[20 HolidayTrip 37](#_Toc151906482)

[[Easy / 40 mins] Calculate the minimum number of cars needed to take all of the passengers. 37](#_Toc151906483)

[21 Missions 39](#_Toc151906484)

[[Easy / 40 mins] Calculate the minimum number of days required to complete all of the missions in a game. 39](#_Toc151906485)

[22 PlayersMovements 41](#_Toc151906486)

[[Easy / 40 mins] Calculate the number of players that will perform a move. 41](#_Toc151906487)

[23 SortedTwoLettersWord 43](#_Toc151906488)

[[Easy / 20 mins] Given a string, count the minimum number of letters that need to be deleted in order to obtain a string in the format "A...AB...B". 43](#_Toc151906489)

[Task 3 44](#_Toc151906490)

[24 DiceRolls 44](#_Toc151906491)

[[Easy / 40 mins] Given a subset of a series of dice roll results, and the arithmetic mean of all results, find the missing results. 44](#_Toc151906492)

[25 DistinctNumbersCount 46](#_Toc151906493)

[[Easy / 40 mins] Count the minimum number of integers that must be deleted from an array so that no two integers occur the same number of times. 46](#_Toc151906494)

[26 BankTransfers 47](#_Toc151906495)

[[Easy / 40 mins] Given a list of transfers between two banks, determine the minimum initial account balance of each bank required for the given transfers to be carried out. 47](#_Toc151906496)

[27 MaxSwitchingSlice 49](#_Toc151906497)

[[Easy / 40 mins] Given an array, find the length of the longest switching slice. 49](#_Toc151906498)

[28 BattleshipRecognition 50](#_Toc151906499)

[[Easy / 40 mins] Given a rectangular board containing ships of three different sizes, find the number of ships of each size. 50](#_Toc151906500)

[29 SanatoriumAccommodation 53](#_Toc151906501)

[[Easy / 40 mins] Find the minimum number of rooms needed to accommodate all guests in a sanatorium according to their preferences. 53](#_Toc151906502)

[30 FixTheTable 54](#_Toc151906503)

[[Easy / 40 mins] Given the distinct positions of holes, find the shortest length of two identical boards that can cover all the holes. 54](#_Toc151906504)

[31 XYSplit 55](#_Toc151906505)

[[Easy / 40 mins] Given a string, return the number of ways to split it into two parts, such that at least one of the parts has an equal number of 'x's and 'y's. 55](#_Toc151906506)

[32 EqualSegments 56](#_Toc151906507)

[[Medium / 50 mins] Find the maximum number of non-intersecting segments of length 2 with equal sums in a given array of integers. 56](#_Toc151906508)

[33 FreeingStorageSpace 57](#_Toc151906509)

[[Medium / 50 mins] Compute how many types of items can stay in a storeroom after removing R consecutive items. 57](#_Toc151906510)

[34 GardenArrangement 59](#_Toc151906511)

[[Medium / 50 mins] Calculate the minimum number of trees that must be planted or replanted in order to make the number of them equal in every section of a garden. 59](#_Toc151906512)

[35 ArraySlicing 61](#_Toc151906513)

[[Medium / 50 mins] Given a sequence of distinct integers, split it into the maximum possible number of slices such that sorting the separate slices sorts the full sequence. 61](#_Toc151906514)

[36 LongestEvenCount 63](#_Toc151906515)

[[Medium / 50 mins] Given a string, return the longest substring in which every letter occurs an even number of times. 63](#_Toc151906516)

[37 LongestTwoDigitFragment 64](#_Toc151906517)

[[Medium / 50 mins] Find the length of the longest fragment in a given array which can be written down using at most two different digits. 64](#_Toc151906518)

[38 MinDistinct 65](#_Toc151906519)

[[Medium / 50 mins] Given an array of N numbers within the range [1..N], find the minimum number of increments and decrements of a single element required to make the array a permutation. 65](#_Toc151906520)

[39 NoEqualDigits 66](#_Toc151906521)

[[Medium / 50 mins] Find the next integer with no two identical consecutive digits. 66](#_Toc151906522)

[40 PathDetection 67](#_Toc151906523)

[[Medium / 50 mins] Check whether the given graph contains a path going through all vertices, one by one, in increasing order of their numbers. 67](#_Toc151906524)

[41 PriceFluctuation 70](#_Toc151906525)

[[Medium / 40 mins] For a historical record of assets values, calculate the maximum income you could have made. 70](#_Toc151906526)

[42 Sticks 71](#_Toc151906527)

[[Medium / 10 mins] Given two sticks, cut them into four sticks in order to create the largest possible square. 71](#_Toc151906528)

[43 AngryFrogs 72](#_Toc151906529)

[[Medium / 50 mins] Given blocks in a row, find the distance between the two most distant blocks, such that the heights between them are first decreasing and then increasing. 72](#_Toc151906530)

[44 AssassinsStealth 74](#_Toc151906531)

[[Medium / 50 mins] Given a two-dimensional board with an assassin and guards on it, determine whether the assassin can get to the bottom-right corner undetected. 74](#_Toc151906532)

[45 ChoosingNumbers 76](#_Toc151906533)

[[Medium / 50 mins] Choose the maximum number of integers from an array so that, after sorting, the difference between every adjacent pair is equal. 76](#_Toc151906534)

[46 CleaningRobot 77](#_Toc151906535)

[[Medium / 50 mins] Count the number of squares on a rectangular grid visited by a cleaning robot. The robot only moves forward, turning right if it cannot make a move. 77](#_Toc151906536)

[47 CollectingRainwater 79](#_Toc151906537)

[[Medium / 50 mins] Find the minimum number of water tanks needed to save the rainwater from all houses in a street. 79](#_Toc151906538)

[48 CreateDiverseWord 81](#_Toc151906539)

[[Medium / 50 mins] Join some of the given strings to create the longest possible string that contains neither "AAA" nor "BBB" as a fragment. 81](#_Toc151906540)

[49 DivideIntoGroups 83](#_Toc151906541)

[[Medium / 50 mins] Divide people into three groups so that the largest score difference between the people in each group is as small as possible. 83](#_Toc151906542)

[50 SquareTiles 85](#_Toc151906543)

[[Medium / 50 mins] Given M tiles of size 1x1 and N tiles of size 2x2, calculate the size of the largest square you can create out of these tiles. 85](#_Toc151906544)

[51 StringMisspellingVar (Opal variant) 87](#_Toc151906545)

[[Medium / 50 mins] Given two strings, check whether one of them can be obtained from the other by performing a simple edit operation. 87](#_Toc151906546)

[52 TwoWayRoadRenovation 89](#_Toc151906547)

[[Medium / 50 mins] Given a representation of a two-lane road, select at most two stretches (one in each lane) that cover the maximum number of potholes in total and that allow the road to remain open to traffic. 89](#_Toc151906548)

[53 WordSplit 91](#_Toc151906549)

[[Medium / 20 mins] Given a string consisting only of lowercase letters, split it into a minimal number of substrings such that each substring contains no letter more than once. 91](#_Toc151906550)

[54 RecyclingTrucks 92](#_Toc151906551)

[[Medium / 50 mins] Calculate the time needed by three recycling trucks to collect all trash in a street. 92](#_Toc151906552)

[55 RoadRepair 94](#_Toc151906553)

[[Medium / 50 mins] Find how many potholes in a road can be fixed within a given budget if fixing K consecutive potholes costs K+1. 94](#_Toc151906554)

[56 RoadRoller 96](#_Toc151906555)

[[Medium / 50 mins] Find the minimum number of road roller drives needed to patch all the potholes in a road. 96](#_Toc151906556)

[57 PutThreeTiles 99](#_Toc151906557)

[[Hard / 20 mins] Given an array of integers, calculate the maximum sum of numbers that can be covered using at most three tiles. 99](#_Toc151906558)

[58 FinancialPlan 100](#_Toc151906559)

[[Hard / 60 mins] How many expenses must be rescheduled to the end of the year so that the company doesn't fall into debt? 100](#_Toc151906560)

[59 DecreasePollution 102](#_Toc151906561)

[[Hard / 60 mins] Given an array of integers, find the minimum number of times its elements must be divided by 2 in order to reduce the sum of the array by at least half. 102](#_Toc151906562)

[60 AngleBrackets 104](#_Toc151906563)

[[Hard / 60 mins] Find the maximum possible length of a symmetric fragment which can be obtained after replacing question marks in a given string with "<" or ">". 104](#_Toc151906564)

[61 SameCorners 105](#_Toc151906565)

[[Hard / 60 mins] Given an array, find the subarray with the largest sum in which the leftmost and rightmost elements are the same. 105](#_Toc151906566)

# Task 1

## 01 TheWidestPath

### [Elementary / 30 mins] Given a forest containing N trees, find the width of the widest vertical path that can be built without cutting any tree.

**Task description**

There are **N** trees (numbered from **0** to **N−1**) in a forest. The **K-th** tree is located at coordinates (**X[K], Y[K]**).

We want to build the widest possible vertical path, such that there is no tree on it. The path must be built somewhere between a leftmost and a rightmost tree, which means that the width of the path cannot be infinite.

What is the width of the widest possible path that can be built?

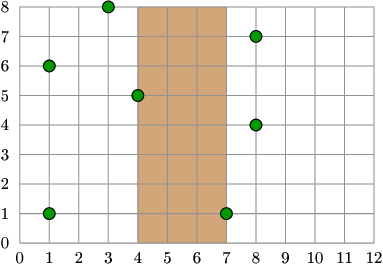
Write a function:

**function solution(X, Y);**

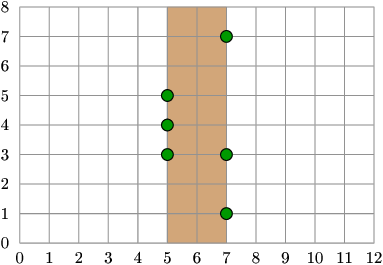
that, given two arrays **X** and **Y** consisting of **N** integers each, denoting the positions of trees, returns the width of the widest possible path that can be built.

**Examples:**

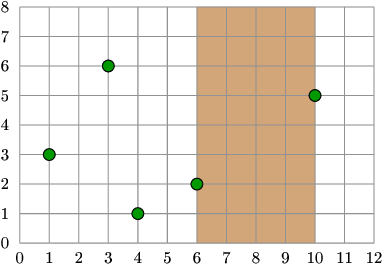
**1.** Given **X = [1, 8, 7, 3, 4, 1, 8], Y=[6, 4, 1, 8, 5, 1, 7]**, the function should return **3**.



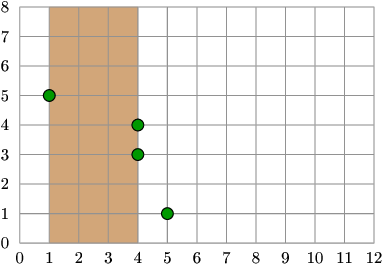
**2.** Given **X = [5, 5, 5, 7, 7, 7], Y=[3, 4, 5, 1, 3, 7]**, the function should return **2**.



**3.** Given **X = [6, 10, 1, 4, 3], Y=[2, 5, 3, 1, 6]**, the function should return **4**.



**4.** Given **X = [4, 1, 5, 4], Y=[4, 5, 1, 3]**, the function should return **3**.



Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[2..100,000]**;
* each element of arrays **X** and **Y** is an integer within the range **[0..1,000,000,000]**;
* there are no two trees with the same coordinates;
* a path of width at least 1 can always be built.

## 02 CardPayments

### [Easy / 40 mins] Given a list of transactions within one year, calculate the final balance of an account. Add a fee for each month that did not include at least three card payments for a total sum of at least 100.

**Task description**

You are given a list of all the transactions on a bank account during the year **2020**. The account was empty at the beginning of the year (the balance was **0**).

Each transaction specifies the amount and the date it was executed. If the amount is negative (less than **0**) then it was a card payment, otherwise it was an incoming transfer (amount at least **0**). The date of each transaction is in **YYYY−MM−DD** format: for example, **2020−05−20** represents **20th May 2020**.

Additionally, there is a fee for having a card (omitted in the given transaction list), which is **5** per month. This fee is deducted from the account balance at the end of each month unless there were at least three payments made by card for a total cost of at least **100** within that month.

Your task is to compute the final balance of the account at the end of the year **2020**.

Write a function:

**function solution(A, D);**

that, given an array **A** of **N** integers representing transaction amounts and an array **D** of **N** strings representing transaction dates, returns the final balance of the account at the end of the year **2020**. Transaction number **K** (for **K** within the range **[0..N-1]**) was executed on the date represented by **D[K]** for amount **A[K]**.

**Examples:**

**1.** Given **A = [100, 100, 100, −10]** and **D = ["2020−12−31", "2020−12−22", "2020−12−03", "2020−12−29"],** the function should return **230**. Total income was equal to **100 + 100 + 100 − 10 = 290** and the fee was paid every month, so **290 - (5 \* 12) = 230**.

**2.** Given **A = [180, -50, -25, -25]** and **D = ["2020−01−01", "2020−01−01", "2020−01−01", "2020−01−31"],** the function should return **25**. The income was equal to **180**, the expenditure was equal to **100** and the fee was applied in every month except January: **180 - 100 - (5 \* 11) = 25**.

**3.** Given **A = [1, -1, 0, -105, 1]** and **D = ["2020−12−31", "2020−04−04", "2020−04−04", "2020−04−14", "2020−07−12"],** the function should return **-164**. The fee is paid every month. **1 - 1 + 0 - 105 + 1 - (5 \* 12) = -164**. Note that in April, even though the total cost of card payments was **106** (more than **100**), there were only two payments made by card, so the fee was still applied. A transaction of value **0** is considered a positive, incoming transfer.

**4.** Given **A = [100, 100, -10, -20, -30]** and **D = ["2020−01−01", "2020−02−01", "2020−02−11", "2020−02−05", "2020−02−08"],** the function should return **80**.

**5.** Given **A = [-60, 60, -40, -20]** and **D = ["2020−10−01", "2020−02−02", "2020−10−10", "2020−10−30"],** the function should return **−115**.

Assume that:

* **N** is an integer within the range **[1..100]**;
* each element of array **A** is an integer within the range **[−1,000..1,000]**;
* **D** contains strings in **YYYY−MM−DD** format, representing dates in the range **2020−01−01** to **2020−12−31**.

In your solution, focus on correctness. The performance of your solution will not be the focus of the assessment.

## 03 CastleBuilding

### [Easy / 60 mins] Find the number of castles that can be built.

**Task description**

Charlemagne, the King of Frankia, is considering building some castles on the border with Servia. The border is divided into **N** segments. The King knows the height of the terrain in each segment of the border. The height of each segment of terrain is stored in array **A**, with **A[P]** denoting the height of the **P-th** segment of the border. The King has decided to build a castle on top of every hill and in the bottom of every valley.

Let **[P..Q]** denote a group of consecutive segments from **P** to **Q** inclusive such that (**0 ≤ P ≤ Q ≤ N−1**). Segments **[P..Q]** form a hill or a valley if all the following conditions are satisfied:

* The terrain height of each segment from **P** to **Q** is the same (**A[P] = A[P+1] = ... = A[Q]**);
* If **P > 0** then **A[P−1] < A[P]** (for a hill) or **A[P−1] > A[P]** (for a valley);
* If **Q < N−1** then **A[Q+1] < A[Q]** (for a hill) or **A[Q+1] > A[Q]** (for a valley);

That is, a hill is higher than its surroundings and a valley is lower than its surroundings. Note that if the surroundings on either side of the hill or valley don't exist (i.e. at the edges of the area under consideration, where **P = 0** or **Q = N−1**), then the condition is considered satisfied for that side of the hill/valley.

The king is wondering how many castles is he going to build. Can you help him?

For example, consider the following array **A = [2, 2, 3, 4, 3, 3, 2, 2, 1, 1, 2, 5]**.

A screenshot of a game

Description automatically generated

There are two hills: **[3..3]** and **[11..11]**. There are also two valleys: **[0..1]** and **[8..9]**. There are no other suitable places for castles.

Write a function:

**function solution(A);**

that, given an array **A** consisting of **N** integers, as explained above, returns the total number of hills and valleys.

For example, given array **A** as described above, the function should return **4**.

Given array **A = [−3, −3]** describing segments with a terrain height below **0**, segment **[0..1]** forms both a hill and a valley, and only one castle can be built, so the function should return **1**.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..100,000];**
* each element of array **A** is an integer within the range **[−1,000,000,000..1,000,000,000].**

## 04 CommonLetter

### [Easy / 60 mins] Given an array of strings, find a pair of strings that share the same letter at the same position.

**Task description**

You are given an array **S** consisting of **N** strings. Every string is of the same length **M**. Your task is to find a pair of strings in array **S**, such that there exists a position in which both of the strings have the same letter. Both the index in array **S** and the positions in the strings are numbered from zero.

For example, given **S = ["abc", "bca", "dbe"]**, string **0 ("abc")** and string **2 ("dbe")** have the same letter **'b'** in position **1**. On the other hand, for strings **"abc"** and **"bca"** there does not exist a position in which they have the same letter.

Write a function:

**function solution(S);**

that, given a zero-indexed array **S** of **N** strings, returns an array describing a pair of strings from **S** which share a common letter at some index. If there is no such pair, the function should return an empty array. If there is more than one correct answer, the function can return any of them.

The result should be represented as an array containing three integers. The first two integers are the indexes in **S** of the strings belonging to the pair. The third integer is the position of the common letter.

For **S = ["abc", "bca", "dbe"]**, as above, the result array should be represented as **[0, 2, 1]**. Another correct answer is **[2, 0, 1]**, as the order of indexes of strings does not matter.

**Examples:**

**1.** Given: **S = ["abc", "bca", "dbe"]**, your function may return **[0, 2, 1]** as described above.

**2.** Given: **S = ["zzzz", "ferz", "zdsr", "fgtd"],** your function may return **[0, 1, 3]**. Both **"zzzz"** and **"ferz"** have **'z'** in position **3**. The function may also return **[1, 3, 0]**, which would reflect strings **"ferz"**, **"fgtd"** and letter **'f'**.

**3.** Given **A = ["gr", "sd", "rg"]**, your function should return **[]**. There is no pair of strings that fulfils the criteria.

**4.** Given **A = ["bdafg", "ceagi"]**, your function may return **[0, 1, 2]**.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..30,000]**;
* **M** is an integer within the range **[1..2,000];**
* each element of **S** consists only of lowercase English letters (**a-z**);
* **N \* M ≤ 30,000**.

## 05 CountBananas

### [Easy / 20 mins] Calculate how many times you can print the word "BANANA" using the letters given in string S.

**Task description**

A string **S** made of uppercase English letters is given. In one move, six letters forming the word "**BANANA**" (one '**B**', three '**A**'s and two '**N**'s) can be deleted from **S**. What is the maximum number times such a move can be applied to **S**?

Write a function:

**function solution(S);**

that, given a string **S** of length **N**, returns the maximum number of moves that can be applied.

**Examples:**

1. Given **S = "NAABXXAN"**, the function should return **1**.

The picture describes the first example test.

2. Given **S = "NAANAAXNABABYNNBZ"**, the function should return **2**.

The picture describes the second example test.

3. Given **S = "QABAAAWOBL"**, the function should return **0**.

The picture describes the third example test.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..100,000]**;
* string **S** is made only of uppercase letters (**A−Z**).

## 06 CreatePalindrome

### [Easy / 20 mins] Replace all question marks in a given string to obtain a palindrome.

**Task description**

Write a function **solution** that, given a string **S** of length **N**, returns any palindrome which can be obtained by replacing all of the question marks in **S** by lowercase letters ('**a**'−'**z**'). If no palindrome can be obtained, the function should return the string "**NO**".

A palindrome is a string that reads the same both forwards and backwards. Some examples of palindromes are: "**kayak**", "**radar**", "**mom**".

**Examples:**

**1.** Given **S = "?ab??a"**, the function should return "**aabbaa**".

**2.** Given **S = "bab??a"**, the function should return "**NO**".

**3.** Given **S = "?a?"**, the function may return "**aaa**". It may also return "**zaz**", among other possible answers.

Assume that:

* **N** is an integer within the range **[1..1,000]**;
* string **S** consists only of lowercases letters (**'a' − 'z'**) or '**?**'.

In your solution, focus on correctness. The performance of your solution will not be the focus of the assessment.

## 07 DiversityString

### [Easy / 40 mins] Find a string of a given length containing as many different lower-case letters as possible, in which each letter occurs an equal number of times.

**Task description**

Write a function **solution** that, given an integer N, returns a string of length N containing as many different lower-case letters ('**a**'-'**z**') as possible, in which each letter occurs an equal number of times.

**Examples:**

**1.** Given **N = 3**, the function may return "**fig**", "**pea**", "**nut**", etc. Each of these strings contains three different letters with the same number of occurrences.

**2.** Given **N = 5**, the function may return "**mango**", "**grape**", "**melon**", etc.

**3.** Given **N = 30**, the function may return "**aabbcc...oo**" (each letter from '**a**' to '**o**' occurs twice). The string contains 15 different letters.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..200,000]**.

## 08 EraseOneLetter

### [Easy / 20 mins] Remove one letter from a word so that resulting string will be as small as possible (alphabetically).

**Task description**

Write a function **solution** that, given a string **S** consisting of **N** characters, returns the alphabetically smallest string that can be obtained by removing exactly one letter from **S**.

**Examples:**

**1.** Given **S = "acb"**, by removing one letter, you can obtain "**ac**", "**ab**" or "**cb**". Your function should return "**ab**" (after removing '**c**') since it is alphabetically smaller than "**ac**" and "**bc**".

**2.** Given **S = "hot"**, your function should return "**ho**", which is alphabetically smaller than "**ht**" and "**ot**".

**3.** Given **S = "codility"**, your function should return "**cdility**", which can be obtained by removing the second letter.

**4.** Given **S = "aaaa"**, your function should return "**aaa**". Any occurrence of '**a**' can be removed.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[2..100,000]**;
* string **S** is made only of lowercase letters **(a−z)**.

## 09 EvenPairsOnCycle

### [Easy / 40 mins] Given numbers on a circle, find the maximum number of neighbouring pairs with an even sum.

**Task description**

You are given **N** numbers on a circle, described by an array **A**. Find the maximum number of neighbouring pairs whose sums are even. One element can belong to only one pair.

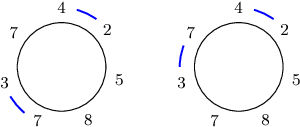
Write a function:

**function solution(A);**

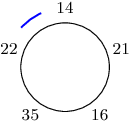
that, given an array **A** consisting of **N** integers, returns the maximum number of neighbouring pairs whose sums are even.

**Examples:**

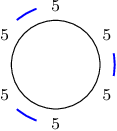
**1.** Given **A = [4, 2, 5, 8, 7, 3, 7]**, the function should return **2**. We can create two pairs with even sums: (**A[0], A[1]**) and (**A[4], A[5]**). Another way to choose two pairs is: (**A[0], A[1]**) and (**A[5], A[6]**).



**2.** Given **A = [14, 21, 16, 35, 22]**, the function should return **1**. There is only one qualifying pair: (**A[0], A[4]**).



**3.** Given **A = [5, 5, 5, 5, 5, 5]**, the function should return **3**. We can create three pairs: (**A[0], A[5]), (A[1], A[2]**) and (**A[3], A[4]**).



Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..100,000]**;
* each element of array **A** is an integer within the range **[0..1,000,000,000]**.

## 10 ForbiddenTriosSwaps

### [Easy / 20 mins] Given a string S consisting of letters 'a' and 'b', return the minimum number of swaps needed to obtain a string with no instances of three identical consecutive letters.

**Task description**

You are given a string **S** consisting of **N** letters **'a'** and/or **'b'**. In one move, you can swap one letter for the other ('**a'** for '**b**' or '**b'** for **'a'**).

Write a function **solution** that, given such a string **S**, returns the minimum number of moves required to obtain a string containing no instances of three identical consecutive letters.

**Examples:**

**1.** Given **S** = "**baaaaa**", the function should return **1**. The string without three identical consecutive letters which can be obtained in one move is "**baabaa**".

**2.** Given **S** = "**baaabbaabbba**", the function should return **2**. There are four valid strings obtainable in two moves: for example, "**bbaabbaabbaa**".

**3.** Given **S** = "**baabab**", the function should return **0**.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[0..200,000]**;
* string **S** is made only of the characters '**a**' and/or '**b**'.

## 11 FormatArray

### [Easy / 75 mins] Given an array of integers, output a string that presents the integers in tabular form.

**Task description**

You have an array of numbers and you would like to print these numbers in a tabular format to make them look more organized. Each cell of the table contains exactly one number and is surrounded by exactly four edges:

**+-+**

**|4|**

**+-+**

**+-----+**

**|12345|**

**+-----+**

As you can see above, each corner of the cell is represented by a "**+**" sign, vertical edges by "**-**" signs and horizontal edges by "**|**" signs. The width of the cell adjusts to accommodate the number of digits of the number written within it. There can be many cells in a row. Adjacent cells share an edge:

**+---+---+---+---+**

**| 4| 35| 80|123|**

**+---+---+---+---+**

Note that each cell has the same width. The width of the cell adjusts to match the width of the longest number in the table. The numbers in cells are aligned to the right, with any unused area in each cell filled with spaces.

The table can consist of many rows, and adjacent rows share an edge:

**+-----+-----+-----+-----+**

**| 4| 35| 80| 123|**

**+-----+-----+-----+-----+**

**|12345| 44| 8| 5|**

**+-----+-----+-----+-----+**

**| 24| 3| 22| 35|**

**+-----+-----+-----+-----+**

Your goal is to output a table containing all the numbers from a given array such that each row contains exactly **K** numbers. The last row can contain fewer numbers.

Write a function:

**function solution(A, K);**

that, given a non-empty array **A** consisting of **N** integers and an integer **K**, prints a string representing the formatted array. The numbers in the table should appear in the same order as the numbers in the array.

For example, given array **A = [4, 35, 80, 123, 12345, 44, 8, 5]** and **K = 10,** the resultant table will contain exactly one row, as shown below:

**+-----+-----+-----+-----+-----+-----+-----+-----+**

**| 4| 35| 80| 123|12345| 44| 8| 5|**

**+-----+-----+-----+-----+-----+-----+-----+-----+**

For **A = [4, 35, 80, 123, 12345, 44, 8, 5, 24, 3], K = 4**, the table would appear as follows:

**+-----+-----+-----+-----+**

**| 4| 35| 80| 123|**

**+-----+-----+-----+-----+**

**|12345| 44| 8| 5|**

**+-----+-----+-----+-----+**

**| 24| 3|**

**+-----+-----+**

Given **A = [4, 35, 80, 123, 12345, 44, 8, 5, 24, 3, 22, 35]** and **K = 4**, the table would appear as follows:

**+-----+-----+-----+-----+**

**| 4| 35| 80| 123|**

**+-----+-----+-----+-----+**

**|12345| 44| 8| 5|**

**+-----+-----+-----+-----+**

**| 24| 3| 22| 35|**

**+-----+-----+-----+-----+**

The function shouldn't return any value.

You can print a string to the output (without or with the end-of-line character) as follows:

**process.stdout.write('sample string'); process.stdout.write('whole line\n');**

Assume that:

* **N** is an integer within the range **[1..200]**;
* **K** is an integer within the range **[1..1,000,000,000]**;
* each element of array **A** is an integer within the range **[0..1,000,000,000]**.

In your solution, focus on correctness. The performance of your solution will not be the focus of the assessment.

## 12 MonitorsDelivery

### [Easy / 40 mins] Given a set of orders, calculate how many of them can be fulfilled.

**Task description**

A technology company announced that a new supply of **P** monitors would soon be available at their store. There were **N** orders (numbered from **0** to **N−1**) placed by customers who wanted to buy those monitors. The **K-th** order has to be delivered to a location at distance **D[K]** from the store and is for exactly **C[K]** monitors.

Now the time has come for the monitors to be delivered. The orders will be fulfilled one by one. To minimize the shipping time, it has been decided that the deliveries will be made in order of increasing distance from the store. If there are many customers at the same distance, they can be processed in any order. Monitors to more distant customers will be delivered only once all orders to customers closer to the store have already been fulfilled.

What is the maximum total number of orders that can be fulfilled?

Write a function:

**function solution(D, C, P);**

that, given two arrays of integers **D** and **C**, and an integer **P**, returns the maximum total number of orders that can be fulfilled.

**Examples:**

**1.** Given **D = [5, 11, 1, 3], C = [6, 1, 3, 2]** and **P = 7**, the function should return **2**. The customers at distances **1** and **3** will have their orders fulfilled and **3 + 2 = 5** monitors will be delivered.

**2.** Given **D = [10, 15, 1], C = [10, 1, 2]** and **P = 3**, the function should return **1**. Only the order for the customer at distance **1** will be fulfilled. There will not be enough monitors in the store for the customer at distance **10**. Therefore, orders for customers at distances **10** and **15** will not be fulfilled.

**3.** Given **D = [11, 18, 1], C = [9, 18, 8]** and **P = 7**, the function should return **0**.

**4.** Given **D = [1, 4, 2, 5], C = [4, 9, 2, 3]** and **P = 19**, the function should return **4**.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..100,000]**;
* each element of arrays **D** and **C** is an integer within the range **[1..1,000,000,000]**;
* **P** is an integer within the range **[0..1,000,000,000]**.

## 13 SameDigitMerge

### [Easy / 30 mins] Count the number of pairs in which the last digit of the first selected number is the same as the first digit of the second selected number.

**Task description**

There is an array **numbers** made of **N** integers. Each number has at least two digits and its first and last digits are different.

You can select a pair of numbers if the last digit of the first selected number is the same as the first digit of the second selected number. Calculate the number of ways in which such a pair of numbers can be selected.

Write a function:

**function solution(numbers);**

that, given an array **numbers** made of **N** integers, returns the number of ways to select a pair of numbers as described above.

**Examples:**

**1.** Given **numbers** = **[30, 12, 29, 91],** the function should return **3**. The pairs are: **(12, 29)**, **(29, 91)** and **(91, 12).**

**2.** Given **numbers** = **[122, 21, 21, 23]**, the function should return **5**. The pairs are: **(122, 21)** occurring twice, **(122, 23)**, and **(21, 122)** occurring twice. Please note that the same pair of numbers can appear multiple times if the pairs of their indices are different.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..100,000]**;
* each element of array **numbers** is an integer within the range **[10..1,000,000,000]**;
* the first and last digits in each element of array **numbers** are different;
* the answer does not exceed **1,000,000,000**.

## 14 ShortestUniqueSubstring

### [Easy / 20 mins] Given a string, count the length of the shortest word which occurs in it exactly once.

**Task description**

Write a function **solution** that, given a string **S** of length **N**, returns the length of the shortest *unique* substring of **S**, that is, the length of the shortest word which occurs in **S** exactly once.

**Examples:**

**1.** Given **S = "abaaba"**, the function should return **2**. The shortest unique substring of **S** is "**aa**".

**2.** Given **S = "zyzyzyz"**, the function should return **5**. The shortest unique substring of **S** is "**yzyzy**". Note that there are shorter words, like "**yzy**", occurrences of which overlap, but they still count as multiple occurrences.

**3.** Given **S = "aabbbabaaa"**, the function should return **3**. All substrings of size 2 occurs in **S** at least twice.

Assume that:

* **N** is an integer within the range **[1..200]**;
* string **S** is made only of lowercase letters **(a−z)**.

In your solution, focus on correctness. The performance of your solution will not be the focus of the assessment.

## 15 SmallestDigitSum

### [Easy / 20 mins] Given an integer N, find the smallest integer whose digits sum to N.

**Task description**

Write a function **solution** that, given integer **N**, returns the smallest non-negative integer whose individual digits sum to **N**.

**Examples:**

**1.** Given **N = 16**, the function should return **79**. There are many numbers whose digits sum to **16** (for example: **79, 97, 808, 5551, 22822**, etc.). The smallest such number is **79**.

**2.** Given **N = 19**, the function should return **199** (the sum of digits is **1 + 9 + 9 = 19**).

**3.** Given **N = 7**, the function should return **7**.

Assume that:

* **N** is an integer within the range **[0..50]**.

In your solution, focus on correctness. The performance of your solution will not be the focus of the assessment.

## 16 ValueOccurrences

### [Easy / 40 mins] Given a sorted array, calculate how many insertion and removal operations are required to make every value X in the array occur exactly X times.

**Task description**

There is an array **A** of **N** integers sorted in non-decreasing order. In one move, you can either remove an integer from A or insert an integer before or after any element of **A**. The goal is to achieve an array in which all values X that are present in the array occur exactly **X** times.

For example, given **A = [1, 1, 3, 4, 4, 4]**, value **1** occurs twice, value **3** occurs once and value **4** occurs three times. You can remove one occurrence each of both 1 and **3**, and insert one occurrence 4, resulting in the array **[1, 4, 4, 4, 4]**. In this array, every element **X** occurs exactly **X** times.

What is the minimum number of moves after which every value **X** in the array occurs exactly **X** times?

Write a function:

**function solution(A);**

that, given an array **A**, returns the minimum number of moves after which every value **X** in the array occurs exactly **X** times. Note that it is permissible to remove some values entirely, if appropriate.

**Examples:**

**1.** Given **A = [1, 1, 3, 4, 4, 4]**, your function should return **3**, as described above.

**2.** Given **A = [1, 2, 2, 2, 5, 5, 5, 8]**, your function should return **4**. You can delete the **8** and one occurrence of **2**, and insert **5** twice, resulting in **[1, 2, 2, 5, 5, 5, 5, 5]** after four moves. Notice that after the removals, there is no occurrence of **8** in the array anymore.

**3.** Given **A = [1, 1, 1, 1, 3, 3, 4, 4, 4, 4, 4]**, your function should return **5**.

**4.** Given **A = [10, 10, 10]**, your function should return **3**. You can remove all elements, resulting in an empty array.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..100,000]**;
* each element of array **A** is an integer within the range **[1..100,000,000]**;
* elements of array **A** are sorted in non-decreasing order.

## 17 AsphaltPatches

### [Easy / 40 mins] Given a description of a road with potholes, find the minimum number of patches needed to repair them all. One patch repairs three consecutive road segments.

**Task description**

There is a road consisting of **N** segments, numbered from **0** to **N-1**, represented by a string **S**. Segment **S[K]** of the road may contain a pothole, denoted by a single uppercase "**X**" character, or may be a good segment without any potholes, denoted by a single dot, "**.**".

For example, string "**.X..X**" means that there are two potholes in total in the road: one is located in segment **S[1]** and one in segment **S[4]**. All other segments are good.

The road fixing machine can patch over three consecutive segments at once with asphalt and repair all the potholes located within each of these segments. Good or already repaired segments remain good after patching them.

Your task is to compute the minimum number of patches required to repair all the potholes in the road.

Write a function:

**function solution(S);**

that, given a string **S** of length **N**, returns the minimum number of patches required to repair all the potholes.

**Examples:**

**1.** Given **S = ".X..X"**, your function should return **2**. The road fixing machine could patch, for example, segments **0-2** and **2-4**.

**2.** Given **S = "X.XXXXX.X."**, your function should return **3**. The road fixing machine could patch, for example, segments **0-2, 3-5** and **6-8**.

**3.** Given **S = "XX.XXX.."**, your function should return **2**. The road fixing machine could patch, for example, segments **0-2** and **3-5**.

**4.** Given **S = "XXXX"**, your function should return **2**. The road fixing machine could patch, for example, segments **0-2** and **1-3**.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[3..100,000]**;
* string **S** is made only of the characters '**.**' and/or '**X**'.

## 18 ABString

### [Easy / 5 mins] Check, whether in a given string all letters 'a' occur before all letters 'b'.

**Task description**

Write a function **solution** that, given a string **S** consisting of **N** letters '**a**' and/or '**b**' returns **true** when all occurrences of letter '**a**' are before all occurrences of letter '**b**' and returns **false** otherwise.

**Examples:**

**1.** Given **S** = "**aabbb**", the function should return **true**.

**2.** Given **S** = "**ba**", the function should return **false**.

**3.** Given **S** = "**aaa**", the function should return **true**. Note that '**b**' does not need to occur in **S**.

**4.** Given **S** = "**b**", the function should return **true**. Note that '**a**' does not need to occur in **S**.

5. Given **S** = "**abba**", the function should return **false**.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..300,000]**;
* string **S** is made only of the characters '**a**' and/or '**b**'.

## 19 EndsTheSame

### [Medium / 20 mins] Count how many times the first and last letters of a string are the same during a sequence of operations.

**Task description**

Initially, string **S** of length **N** is given. Then **N-1** operations are applied to it: move the first letter of **S** to the end. How many times is the first letter of **S** the same as the last letter?

For example, given **S** = "**abbaa**", the obtained sequence of strings is:

abbaa -> bbaaa -> baaab -> aaabb -> aabba

Three of them have the same first and last letter.

Write a function:

**function solution(S);**

that, given a string **S** of length **N**, consisting of letters '**a**' and/or '**b**', returns the number of times the first letter is the same as the last in the obtained sequence of strings.

**Examples:**

**1.** Given **S** = "**abbaa**", the function should return **3**, as described above.

**2.** Given **S** = "**aaaa**", the function should return **4**. The first and last letters are always the same.

**3.** Given **S** = "**abab**", the function should return **0**. The first and last letters are always different.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[2..200,000]**;
* string **S** is made only of the characters '**a**' and/or '**b**'.

# Task 2

## 20 HolidayTrip

### [Easy / 40 mins] Calculate the minimum number of cars needed to take all of the passengers.

**Task description**

A group of friends is going on holiday together. They have come to a meeting point (the start of the journey) using **N** cars. There are **P[K]** people and **S[K]** seats in the **K-th** car for **K** in range **[0..N-1].** Some of the seats in the cars may be free, so it is possible for some of the friends to change the car they are in. The friends have decided that, in order to be ecological, they will leave some cars parked at the meeting point and travel with as few cars as possible.

Write a function:

**function solution(P, S);**

that, given two arrays **P** and **S**, consisting of **N** integers each, returns the minimum number of cars needed to take all of the friends on holiday.

**Examples:**

**1.** Given **P = [1, 4, 1]** and **S = [1, 5, 1]**, the function should return **2**. A person from car number **0** can travel in car number **1** instead. This way, car number **0** can be left parked at the meeting point.

**2.** Given **P = [4, 4, 2, 4]** and **S = [5, 5, 2, 5]**, the function should return **3**. One person from car number **2** can travel in car number 0 and the other person from car number **2** can travel in car number **3**.

**3.** Given **P = [2, 3, 4, 2]** and **S = [2, 5, 7, 2],** the function should return **2**. Passengers from car number **0** can travel in car number **1** and passengers from car number **3** can travel in car number **2**.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..100,000]**;
* each element of arrays **P** and **S** is an integer within the range **[1..9]**;
* every friend had a seat in the car they came in; that is, **P[K] ≤ S[K]** for each **K** within the range **[0..N-1]**.

## 21 Missions

### [Easy / 40 mins] Calculate the minimum number of days required to complete all of the missions in a game.

**Task description**

In order to finish a game, a player has to complete **N** missions. The missions are numbered from **0** to **N-1**. The **K-th** mission has an integer **D[K]** assigned, representing its difficulty level.

During a day, you can perform any number of missions given the two following rules:

* missions should be performed in the specified order, in other words, a mission can be undertaken only if all of the missions preceding it have already been completed;
* the difference between the difficulty levels of any two missions performed on the same day should not be greater than an integer **X**.

Write a function:

**function solution(D, X);**

that, given an array **D** of **N** integers and an integer **X**, returns the minimum number of days required to complete all of the missions in the game.

**Examples:**

**1.** Given **D = [5, 8, 2, 7]** and **X = 3**, your function should return **3**. The first two missions can be performed on the first day, the third mission on the second day and the last mission on the third day. It is not possible to complete all of the missions in fewer days.

**2.** Given **D = [2, 5, 9, 2, 1, 4]** and **X = 4**, your function should return **3**. The first two missions can be performed on the first day, the third mission on the second day and all of the remaining missions on the third day. Note that it is possible to perform the first mission on the first day and the next two missions on the second day. In both of these cases, the minimum number of days required to complete all of the missions is **3**.

**3.** Given **D = [1, 12, 10, 4, 5, 2]** and **X = 2**, your function should return **4**. The first mission can be performed on the first day, the next two missions on the second day, the fourth and fifth missions on the third day, and the last remaining mission on the fourth day. It is not possible to complete all of the missions in fewer days.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..200,000]**;
* **X** is an integer within the range **[0..1,000,000,000]**;
* each element of array **D** is an integer within the range **[1..1,000,000,000]**.

## 22 PlayersMovements

### [Easy / 40 mins] Calculate the number of players that will perform a move.

**Task description**

There are **N** players standing in a row, one player on a field. They are numbered from **0** to **N−1** from left to right.

Players perform moves one by one from left to right, that is, in ascending order of numbers. Each player presses an arrow key in one of the four cardinal directions: left ('**<**'), right ('**>**'), up ('**^**') or down ('**v**'). A key press in the given direction means that the player attempts to move onto the closest field in the direction specified. A move can be performed only if there is no other player already standing on the target field.

Moves are represented as a string **S** of length **N**, where **S[K]** (for **K** within the range **0..N−1**) is the direction of the **K-th** player's move. How many players will actually perform a move successfully?

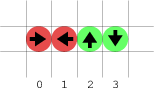
Write a function:

**function solution(S);**

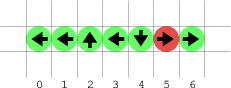
which, given a string **S** of length **N** representing arrow keys pressed by each of the players, returns the number of players that will perform a move successfully.

**Examples:**

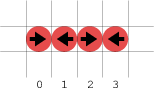
**1.** Given **S = "><^v"**, your function should return **2**. Player **0** cannot move rightwards, because player **1** is standing on the target field. Player **1** cannot move leftwards, because player **0** is standing on the target field. Players **2** and **3** can both perform their moves because there are no other players standing on their target fields. In the pictures below, players that will perform their moves successfully are marked green and players whose moves will fail are marked red.



**2.** Given **S = "<<^<v>>"**, your function should return **6**. Players **0, 1, 2, 3, 4** can all perform their moves because there are no other players standing on their target fields. Player **5** pressed the right-arrow key, but the move cannot be performed as player **6** is already standing on the target field. The move of player **6** can be performed, though, as there is no other player standing on the target field.



**3.** Given **S = "><><"**, your function should return **0**. No player can perform a move.



Assume that:

* **N** is an integer within the range **[1..50]**;
* string **S** is made only of the following characters: **'^', 'v', '<'** and/or **'>'**.

In your solution, focus on correctness. The performance of your solution will not be the focus of the assessment.

## 23 SortedTwoLettersWord

### [Easy / 20 mins] Given a string, count the minimum number of letters that need to be deleted in order to obtain a string in the format "A...AB...B".

**Task description**

We are given a string **S** of length **N** consisting only of letters '**A**' and/or '**B**'. Our goal is to obtain a string in the format "**A...AB...B**" (all letters '**A**' occur before all letters '**B**') by deleting some letters from **S**. In particular, strings consisting only of letters '**A**' or only of letters '**B**' fit this format.

Write a function:

function solution(S);

that, given a string **S**, returns the minimum number of letters that need to be deleted from **S** in order to obtain a string in the above format.

**Examples:**

**1.** Given **S = "BAAABAB"**, the function should return **2**. We can obtain **"AAABB"** by deleting the first occurrence of **'B'** and the last occurrence of **'A'**.

**2.** Given **S = "BBABAA"**, the function should return **3**. We can delete all occurrences of **'A'** or all occurrences of **'B'**.

**3.** Given **S = "AABBBB"**, the function should return **0**. We do not have to delete any letters, because the given string is already in the expected format.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..100,000]**;
* string **S** is made only of the characters **'A'** and/or **'B'**.

# Task 3

## 24 DiceRolls

### [Easy / 40 mins] Given a subset of a series of dice roll results, and the arithmetic mean of all results, find the missing results.

**Task description**

You have just rolled a dice several times. The **N** roll results that you remember are described by an array **A**. However, there are **F** rolls whose results you have forgotten. The arithmetic mean of all of the roll results (the sum of all the roll results divided by the number of rolls) equals **M**.

What are the possible results of the missing rolls?

Write a function:

**function solution(A, F, M);**

that, given an array **A** of length **N**, an integer **F** and an integer **M**, returns an array containing possible results of the missed rolls. The returned array should contain **F** integers from **1** to **6** (valid dice rolls). If such an array does not exist then the function should return **[0]**.

**Examples:**

**1.** Given **A = [3, 2, 4, 3], F = 2, M = 4**, your function should return **[6, 6]**. The arithmetic mean of all the rolls is **(3 + 2 + 4 + 3 + 6 + 6) / 6 = 24 / 6 = 4**.

**2.** Given **A = [1, 5, 6], F = 4, M = 3**, your function may return **[2, 1, 2, 4] or [6, 1, 1, 1]** (among others).

**3.** Given **A = [1, 2, 3, 4], F = 4, M = 6**, your function should return **[0]**. It is not possible to obtain such a mean.

**4.** Given **A = [6, 1], F = 1, M = 1**, your function should return **[0]**. It is not possible to obtain such a mean.

Write an efficient algorithm for the following assumptions:

* **N** and **F** are integers within the range **[1..100,000]**;
* each element of array **A** is an integer within the range **[1..6]**;
* **M** is an integer within the range **[1..6]**.

Remember, all submissions are being checked for plagiarism. Your recruiter will be informed in case suspicious activity is detected.

## 25 DistinctNumbersCount

### [Easy / 40 mins] Count the minimum number of integers that must be deleted from an array so that no two integers occur the same number of times.

**Task description**

An array **A** consisting of **N** integers is given. Our goal is to obtain an array in which every value occurs a unique number of times. We only consider integers that appear at least once in the resulting array. To achieve the goal, we can delete some integers from **A**. What is the minimum number of integers that must be deleted from **A** so that every remaining value occurs a unique number of times?

Write a function:

**function solution(A);**

that, given an array **A** consisting of **N** integers, returns the minimum number of integers that must be deleted from it so that every remaining value occurs a unique number of times.

**Examples:**

**1.** Given **A = [1, 1, 1, 2, 2, 2]**, the function should return **1**. We can delete one occurrence of **1** or one occurrence of **2**. After this operation, one value will occur three times and the other one two times.

**2.** Given **A = [5, 3, 3, 2, 5, 2, 3, 2]**, the function should return **2**. After deleting number **3** twice, the remaining elements of the array are **[5, 2, 5, 2, 3, 2]**. In this array no two numbers occur the same number of times.

**3.** Given **A = [127, 15, 3, 8, 10]**, the function should return **4**. All elements of the array occur exactly once. We have to delete all but one element.

**4.** Given **A = [10000000, 10000000, 5, 5, 5, 2, 2, 2, 0, 0]**, the function should return **4**.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..200,000]**;
* each element of array **A** is an integer within the range **[0..1,000,000,000]**.

## 26 BankTransfers

### [Easy / 40 mins] Given a list of transfers between two banks, determine the minimum initial account balance of each bank required for the given transfers to be carried out.

**Task description**

You are given a list of **N** transfers (numbered from **0** to **N−1**) between two banks: bank **A** and bank **B**. The **K-th** transfer is described by two values:

* **R[K]** (either "**A**" or "**B**") representing the recipient (the bank the transfer is sent to);
* **V[K]** denoting the value sent via the transfer.

All transfers are completed in the order they appear on the list. The banks do not want to go into debt (in other words, their account balance may not drop below **0**). What minimum initial account balance in each bank is necessary in order to complete the transfers?

Write a function:

**function solution(R, V);**

that, given a string **R** and an array of integers **V**, both of length **N**, returns an array of two integers. The integers should represent the minimum initial account balances for banks **A** and **B** in the following order: [bank **A**, bank **B**].

Result array should be returned as an array of integers.

**Examples:**

**1.** Given **R = "BAABA"** and **V = [2, 4, 1, 1, 2]**, the function should return **[2, 4]**. The bank accounts’ balances after each transfer are shown in the following table:

**| A | B**

**------------------------+---+---**

**initial balance | 2 | 4**

**transfer 2 from A to B | 0 | 6**

**transfer 4 from B to A | 4 | 2**

**transfer 1 from B to A | 5 | 1**

**transfer 1 from A to B | 4 | 2**

**transfer 2 from B to A | 6 | 0**

**2.** Given **R = "ABAB"** and **V = [10, 5, 10, 15]**, the function should return **[0, 15]**.

**3.** Given **R = "B"** and **V = [100]**, the function should return **[100, 0]**.

Write an efficient algorithm for the following assumptions:

* string **R** and array **V** are both of length **N**;
* **N** is an integer within the range **[1..100,000]**;
* each element of array **V** is an integer within the range **[1..10,000]**;
* string **R** is made only of the characters **'A'** and/or **'B'**.

## 27 MaxSwitchingSlice

### [Easy / 40 mins] Given an array, find the length of the longest switching slice.

**Task description**

We call an array *switching* if all numbers in even positions are equal and all numbers in odd positions are equal.

For example: **[3, −7, 3, −7, 3]** and **[4, 4, 4]** are switching, but **[5, 5, 4, 5, 4]** and **[−3, 2, 3]** are not switching.

What is the length of the longest switching slice (continuous fragment) in a given array **A**?

Write a function:

**function solution(A);**

that, given an array **A** consisting of **N** integers, returns the length of the longest switching slice in **A**.

**Examples:**

**1.** Given **A = [3, 2, 3, 2, 3]**, the function should return **5**, because the whole array is switching.

**2.** Given **A = [7, 4, −2, 4, −2, −9]**, the function should return **4**. The longest switching slice is **[4, −2, 4, −2]**.

**3.** Given **A = [7, −5, −5, −5, 7, −1, 7]**, the function should return **3**. There are two switching slices of equal length: **[−5, −5, −5]** and **[7, −1, 7]**.

**4.** Given **A = [4]**, the function should return **1**. A single-element slice is also a switching slice.

* Write an efficient algorithm for the following assumptions:
* **N** is an integer within the range **[1..100,000]**;
* each element of array **A** is an integer within the range **[−1,000,000,000..1,000,000,000]**.

## 28 BattleshipRecognition

### [Easy / 40 mins] Given a rectangular board containing ships of three different sizes, find the number of ships of each size.

**Task description**

Battleships is a game played on a rectangular board. You are given a representation of such a board of size **N** (height) **x M** (width) with information about the locations of the ships.

The board is given as an array **B**, whose every element is a string that corresponds to one row of the game board. Each character of each string represents a cell of the board and is either:

* a '**#**' character, marking a part of a ship; or
* a '**.**' character, representing an empty cell.

Two cells that share a side and have a value of '**#**' are parts of the same ship. Cell **(X, Y)** shares a side with cells **(X, Y−1), (X, Y+1), (X−1, Y)** and **(X+1, Y)**.

In the Battleships game there are three types of ships:

**Patrol Boats** of size **1**:



**Submarines** of size **2**, which come in two shapes:



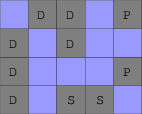
**Destroyers** of size **3**, which come in six shapes:

A black and white square

Description automatically generated with medium confidence

Your task is to find the number of ships of each type occurring on the board.

For example, on the board represented by **B = [".##.#", "#.#..", "#...#", "#.##."]**, there are two patrol boats (marked on the image below as '**P**'), one submarine (**'S'**) and two destroyers (**'D'**).



Write a function:

**function solution(B);**

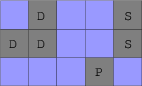
that, given an array **B** consisting of **N** strings of length **M** each, returns an array **R** of three integers, such that:

* **R[0]** represents the number of **Patrol Boats**,
* **R[1]** represents the number of **Submarines**,
* **R[2]** represents the number of **Destroyers**.

**Examples:**

**1.** Given **B = [".##.#", "#.#..", "#...#", "#.##."]**, the function should return **[2, 1, 2]** as explained above.

**2.** Given **B = [".#..#", "##..#", "...#."]**, the function should return **[1, 1, 1]**.



**3.** Given **B = ["##.", "#.#", ".##"]**, the function should return **[0, 0, 2]**.



**4.** Given **B = ["...", "...", "..."]**, the function should return **[0, 0, 0]**.



Assume that:

* **N** is an integer within the range **[1..100]**;
* all strings in **B** are of the same length **M** from the range **[1..100]**;
* every string in **B** consists only of the following characters: '**.**' and/or '**#**';

every ship on the board is either a **Patrol Boat** (size **1**), a **Submarine** (size **2**) or a **Destroyer** (size **3**).

In your solution, focus on correctness. The performance of your solution will not be the focus of the assessment.

## 29 SanatoriumAccommodation

### [Easy / 40 mins] Find the minimum number of rooms needed to accommodate all guests in a sanatorium according to their preferences.

**Task description**

There are **N** guests (numbered from **0** to **N-1**) in a sanatorium waiting to be assigned a room. In each room, any number of guests can be accommodated. However, not all guests like to have a lot of roommates.

You are given an array **A** of **N** integers: the **K-th** guest wants to be in a room that contains at most **A[K]** guests, including themselves.

Write a function:

**function solution(A);**

that, given the array **A**, returns the minimum number of rooms needed to accommodate all guests.

**Examples:**

**1.** Given **A = [1, 1, 1, 1, 1]**, your function should return **5**. Each guest should be accommodated in their own separate room.

**2.** Given **A = [2, 1, 4]**, your function should return **2**. The second guest should be accommodated in one room and the other two guests in another room.

**3.** Given **A = [2, 7, 2, 9, 8]**, your function should return **2**. The first and the third guests should be accommodated in one room and the other three guests in another room.

**4.** Given **A = [7, 3, 1, 1, 4, 5, 4, 9]**, your function should return **4**. The guests can be accommodated as follows: the first two guests in one room, the third and the fourth guests in two single rooms, and the other guests in another room.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..100,000]**;
* each element of array **A** is an integer within the range **[1..100,000]**.

## 30 FixTheTable

### [Easy / 40 mins] Given the distinct positions of holes, find the shortest length of two identical boards that can cover all the holes.

**Task description**

There are **N** holes arranged in a row in the top of an old table. We want to fix the table by covering the holes with two boards. For technical reasons, the boards need to be of the same length.

The position of the **K-th** hole is **A[K]**. What is the shortest length of the boards required to cover all the holes? The length of the boards has to be a positive integer. A board of length **L**, set at position **X**, covers all the holes located between positions **X** and **X+L** (inclusive). The position of every hole is unique.

Write a function:

**function solution(A);**

which, given an array **A** of integers of length **N**, representing the positions of the holes in the table, returns the shortest board length required to cover all the holes.

**Examples:**

**1.** Given **A = [11, 20, 15]**, your function should return **4**. The first board would cover the holes in positions **11** and **15**, and the second board the hole at position **20**.

**2.** Given **A = [15, 20, 9, 11]**, your function should return **5**. The first board covers the holes at positions **9** and **11**, and the second one the holes in positions **15** and **20**.

**3.** Given **A = [0, 44, 32, 30, 42, 18, 34, 16, 35]**, your function should return **18**. The first board would cover the holes in positions between **0** and **18**, and the second the positions between **30** and **44**.

**4.** Given **A = [9]**, your function should return **1**.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..100,000]**;
* each element of array **A** is an integer within the range **[0..1,000,000,000]**;

the elements of **A** are all distinct.

## 31 XYSplit

### [Easy / 40 mins] Given a string, return the number of ways to split it into two parts, such that at least one of the parts has an equal number of 'x's and 'y's.

**Task description**

You are given a string **S** consisting of **N** lowercase English letters. In how many ways can we split **S** into two non-empty parts, such that in at least one part the letter **'x'** and the letter **'y'** occur the same number of times?

Write a function:

**function solution(S);**

that, given a string **S** of length **N**, returns the number of splits **S** satisfying the condition above.

**Examples:**

**1.** Given **S = "ayxbx"**, the function should return **3**. There are four possible splits of **S: "a/yxbx", "ay/xbx", "ayx/bx"** and **"ayxb/x"**. Only **"ay/xbx"** does not fulfill the condition, so the answer is **3**. Note that in **"a/yxbx"** the left part has **0** occurrences of **'x'** and **'y'**, so it counts as correct split.

**2.** Given **S = "xzzzy"**, the function should return **0**.

**3.** Given **S = "toyxmy"**, the function should return **5**.

**4.** Given **S = "apple"**, the function should return **4**.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[2..200,000]**;
* string **S** is made only of lowercase letters **(a−z).**

## 32 EqualSegments

### [Medium / 50 mins] Find the maximum number of non-intersecting segments of length 2 with equal sums in a given array of integers.

**Task description**

You are given an array **A** of integers. Find the maximum number of non-intersecting segments of length **2** (two adjacent elements), such that segments have an equal sum.

For example, given **A = [10, 1, 3, 1, 2, 2, 1, 0, 4]**, there are three non-intersecting segments, each whose sum is equal to **4**: **(1, 3), (2, 2), (0, 4)**. Another three non-intersecting segments are: **(3, 1), (2, 2), (0, 4)**.

Write a function:

**function solution(A);**

that, given an array **A** of **N** integers, returns the maximum number of segments with equal sums.

**Examples:**

**1.** Given **A = [10, 1, 3, 1, 2, 2, 1, 0, 4]**, the function should return **3**, as explained above.

**2.** Given **A = [5, 3, 1, 3, 2, 3]**, the function should return **1**. Each sum of two adjacent elements is different from the others.

**3.** Given **A = [9, 9, 9, 9, 9]**, the function should return **2**.

**4.** Given **A = [1, 5, 2, 4, 3, 3]**, the function should return **3**. There are three segments: **(1, 5), (2, 4), (3, 3)** whose sums are equal to **6**.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[2..100,000]**;
* each element of array **A** is an integer within the range **[0..1,000,000,000]**.

## 33 FreeingStorageSpace

### [Medium / 50 mins] Compute how many types of items can stay in a storeroom after removing R consecutive items.

**Task description**

A storeroom is used to organize items stored in it on **N** shelves. Shelves are numbered from **0** to **N−1**. The K-th shelf is dedicated to items of only one type, denoted by a positive integer **A[K]**.

Recently it was decided that it is necessary to free **R** consecutive shelves. Shelves cannot be reordered. What is the maximum number of types of items which still can be stored in the storeroom after freeing **R** consecutive shelves?

Write a function:

**function solution(A, R);**

that, given an array **A** of **N** integers representing types of items stored on storeroom shelves, and an integer **R** representing the number of consecutive shelves to be freed, returns the maximum number of different types of items that can be stored in the storeroom after freeing **R** consecutive shelves.

**Examples:**

**1.** Given **A = [2, 1, 2, 3, 2, 2]** and **R = 3**, your function should return **2**. It can be achieved, for example, by freeing shelves **2, 3** and **4** (shelves are numbered from **0**).

**2.** Given **A = [2, 3, 1, 1, 2]** and **R = 2**, your function should return **3**. All three types can still be stored by freeing the last two shelves.

**3.** Given **A = [20, 10, 10, 10, 30, 20]** and **R = 3**, your function should return **3**. It can be achieved by freeing the first three shelves.

**4.** Given **A = [1, 100000, 1]** and **R = 3**, your function should return **0**. All shelves need to be freed.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..100,000]**;
* **R** is an integer within the range **[1..N]**;
* each element of array **A** is an integer within the range **[1..100,000]**.

## 34 GardenArrangement

### [Medium / 50 mins] Calculate the minimum number of trees that must be planted or replanted in order to make the number of them equal in every section of a garden.

**Task description**

A garden is divided into **N** sections numbered from **0** to **N-1**. It is described by an array **A**, where **A[K]** denotes the number of trees in the **K-th** section. To make the garden look more organized, we want the number of trees in every section to be the same. As we don't want to cut any trees down, we can perform either of the following actions:

* planting a new tree in one of the sections;
* replanting an existing tree, moving it from one section to another.

We want to minimize the number of actions performed.

Write a function:

function solution(A);

that, given an array **A** consisting of **N** integers describing the garden, returns the minimum number of actions we need to perform in order to make all sections of the garden contain the same number of trees.

**Examples:**

**1.** Given **A = [1, 2, 2, 4]** the function should return **4**. We can move one tree from **A[3]** to **A[1]** and obtain **A = [1, 3, 2, 3]**. Then we can plant two trees in **A[0]** and one tree in **A[2]** to make every section contain three trees.

**2.** Given **A = [4, 2, 4, 6]**, the function should return **2**. We can move two trees from **A[3]** to **A[1]**. This way, every section in the garden will contain four trees.

**3.** Given **A = [1, 1, 2, 1]**, the function should return **3**. We can plant one tree in **A[0]**, **A[1]** and **A[3]** so that each section in the garden contains two trees.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..100,000]**;
* each element of array **A** is an integer within the range **[1..1,000,000,000]**;
* the answer is always less than or equal to **2,000,000,000**.

## 35 ArraySlicing

### [Medium / 50 mins] Given a sequence of distinct integers, split it into the maximum possible number of slices such that sorting the separate slices sorts the full sequence.

**Task description**

We are given an array **A** consisting of **N** distinct integers. We would like to sort array **A** into ascending order using a simple algorithm. First, we divide it into one or more slices (a slice is a contiguous subarray). Then we sort each slice. After that, we join the sorted slices in the same order. Write a function **solution** that returns the maximum number of slices for which the algorithm will return a correctly sorted array.

**Examples:**

**1.** Given **A = [2, 4, 1, 6, 5, 9, 7]**, the function should return **3**. The array can be split into three slices: **[2, 4, 1], [6, 5]** and **[9, 7]**. Then, after sorting each slice and joining them together, the whole array will be sorted into ascending order.

A green and pink bars

Description automatically generated

**2.** Given **A = [4, 3, 2, 6, 1]**, the function should return **1**. The array cannot be split into smaller slices; it has to be sorted all at once.

A black and pink rectangles

Description automatically generated

**3.** Given **A = [2, 1, 6, 4, 3, 7]**, the function should return **3**.

A black and green rectangle with a black background

Description automatically generated

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..100,000]**;
* each element of array **A** is an integer within the range **[1..1,000,000,000]**;
* the elements of **A** are all distinct.

## 36 LongestEvenCount

### [Medium / 50 mins] Given a string, return the longest substring in which every letter occurs an even number of times.

**Task description**

Write a function:

function solution(S);

that, given a string **S** consisting of **N** lowercase English letters, returns the length of the longest substring in which every letter occurs an even number of times. A substring is defined as a contiguous segment of a string. If no such substring exists, return **0**.

**Examples:**

**1.** Given **S = "bdaaadadb"**, the function should return **6**. Substrings in which every letter occurs an even number of times are **"aa", "adad", "daaada"** and **"aaadad"**. The length of the longest of them is **6**.

**2.** Given **S = "abacb"**, the function should return **0**. There is no non-empty substring in which every letter occurs an even number of times.

**3.** Given **S = "zthtzh"**, the function should return **6**. Every letter in the whole string occurs an even number of times.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..100,000]**;
* string **S** consists only of lowercase letters (**'a'-'z'**).

## 37 LongestTwoDigitFragment

### [Medium / 50 mins] Find the length of the longest fragment in a given array which can be written down using at most two different digits.

**Task description**

Write a function:

function solution(A);

that, given an array **A** of **N** integers, returns the length of the longest consistent fragment of **A** in which all elements can be generated using at most two different digits. You must use the same digits for all elements.

**Examples:**

**1.** Given **A = [23, 333, 33, 30, 0, 505]**, the function should return **4**. Elements **333, 33, 30** and **0** can be generated using only digits **0** and **3**.

**2.** Given **A = [615, 88, 498, 99, 9]**, the function should return **2**. The last two elements can be generated using only digit **9**.

**3.** Given **A = [123, 456]**, the function should return **0**.

Assume that:

* **N** is an integer within the range **[1..100]**;
* each element of array **A** is an integer within the range **[0..1,000,000,000]**.

In your solution, focus on correctness. The performance of your solution will not be the focus of the assessment.

## 38 MinDistinct

### [Medium / 50 mins] Given an array of N numbers within the range [1..N], find the minimum number of increments and decrements of a single element required to make the array a permutation.

**Task description**

You are given an array **A** consisting of **N** integers within the range **[1..N]**. In one move, you can increase or decrease the value of any element by **1**. After each move, all numbers should remain within the range **[1..N]**.

Your task is to find the smallest required number of moves to make all elements in the array pairwise distinct (in other words, no value can appear in the array more than once).

Write a function:

function solution(A);

that, given an array **A** consisting of **N** integers, returns the smallest number of moves required to make all elements in the array pairwise distinct. If the result is greater than **1,000,000,000**, the function should return **-1**.

**Examples:**

**1.** Given **A = [1, 2, 1]**, the function should return 2, because you can increase **A[2]** twice: **[1, 2, 1] −> [1, 2, 2] −> [1, 2, 3]**. In this example, you could also change the array to the following values in two moves: **[3, 2, 1], [1, 3, 2], [2, 3, 1]**.

**2.** Given **A = [2, 1, 4, 4]**, the function should return **1**, as it is sufficient to decrease **A[2]** or **A[3]** by **1**, resulting in **[2, 1, 3, 4] or [2, 1, 4, 3]**.

**3.** Given **A = [6, 2, 3, 5, 6, 3]**, the function should return **4**, because you can achieve the following array in four moves: **[6, 2, 1, 5, 4, 3]**.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..200,000]**;
* each element of array **A** is an integer within the range **[1..N]**.

## 39 NoEqualDigits

### [Medium / 50 mins] Find the next integer with no two identical consecutive digits.

**Task description**

You are given a positive integer **N**. Your task is to find the smallest integer greater than **N** that does not contain two identical consecutive digits.

For example, given **N = 1765**, the smallest integer greater than **N** is **1766**. However, in **1766** the last two digits are identical. The next integer, **1767**, does not contain two identical consecutive digits, and is the smallest integer greater than **1765** that fulfils the condition. Note that the second and fourth digits in **1767** can both be **7** as they are not consecutive.

Write a function:

function solution(N);

that, given an integer **N**, returns the smallest integer gre

ater than **N** that does not contain two identical consecutive digits.

**Examples:**

**1.** Given **N = 55**, the function should return **56**. It is the smallest integer greater than **55** and it does not contain two consecutive digits that are the same.

**2.** Given **N = 1765**, the function should return **1767**, as explained above.

**3.** Given **N = 98**, the answer should be **101**. Both **99** and **100** contain two identical consecutive digits, but **101** does not.

**4.** Given **N = 44432**, the answer should be **45010**.

**5.** Given **N = 3298**, the answer should be **3401**.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..1,000,000,000]**.

## 40 PathDetection

### [Medium / 50 mins] Check whether the given graph contains a path going through all vertices, one by one, in increasing order of their numbers.

**Task description**

You are given an undirected graph consisting of **N** vertices, numbered from **1** to **N**, and **M** edges.

The graph is described by two arrays, **A** and **B**, both of length **M**. A pair (**A[K], B[K]**), for **K** from **0** to **M-1**, describes an edge between vertex **A[K]** and vertex **B[K]**.

Your task is to check whether the given graph contains a path from vertex **1** to vertex **N** going through all of the vertices, one by one, in increasing order of their numbers. All connections on the path should be direct.

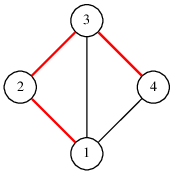
Write a function:

function solution(N, A, B);

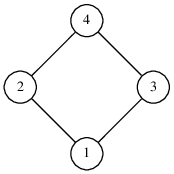
that, given an integer **N** and two arrays **A** and **B** of **M** integers each, returns **true** if there exists a path from vertex **1** to **N** going through all vertices, one by one, in increasing order, or **false** otherwise.

**Examples:**

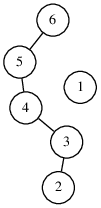
**1.** Given **N = 4, A = [1, 2, 4, 4, 3]** and **B = [2, 3, 1, 3, 1]**, the function should return **true**. There is a path (**1 → 2 → 3 → 4**) using edges (**1, 2**), (**2, 3**) and (**4, 3**).



**2.** Given **N = 4, A = [1, 2, 1, 3]** and **B = [2, 4, 3, 4]**, the function should return **false**. There is no path (**1 → 2 → 3 → 4**), as there is no direct connection from vertex **2** to vertex **3**.



**3.** Given **N = 6, A = [2, 4, 5, 3]** and **B = [3, 5, 6, 4]**, the function should return **false**. There is no direct connection from vertex **1** to vertex **2**.



**4.** Given **N = 3, A = [1, 3]** and **B = [2, 2]**, the function should return **true**. There is a path (**1 → 2 → 3**) using edges (**1, 2**) and (**3, 2**).

Picture illustrates the fourth example test

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[2..100,000]**;
* **M** is an integer within the range **[0..100,000]**;
* all elements of arrays **A** and **B** are integers within the range **[1..N]**;
* there are no self-loops (edges with **A[K] = B[K]**) in the graph;
* there are no multiple edges between the same vertices.

## 41 PriceFluctuation

### [Medium / 40 mins] For a historical record of assets values, calculate the maximum income you could have made.

**Task description**

You are given a record of the historical prices of an investment asset from the last **N** days. Analyze the record in order to calculate what could have been your maximum income. Assume you started with one asset of this type and could hold at most one at a time. You could choose to sell the asset whenever you held one. If you did not hold an asset at some moment, you could always afford to buy an asset (assume you had infinite money available).

What is the maximum income you could make?

Write a function:

function solution(A);

that, given an array **A** of length **N** representing a record of prices over the last **N** days, returns the maximum income you could make. As the result may be large, return its last nine digits without leading zeros (return the result modulo **1,000,000,000**).

**Examples:**

**1.** Given **A = [4, 1, 2, 3]**, the function should return **6**. You could sell the product on the first day (for **4**), buy it on the second (for **1**) and sell it again on the last day (for **3**). The income would be equal **4 − 1 + 3 = 6**.

**2.** Given **A = [1, 2, 3, 3, 2, 1, 5]**, the function should return **7**. You could sell the product when its value was **3**, buy it when it changed to **1**, and sell it again when it was worth **5**.

**3.** Given **A = [1000000000, 1, 2, 2, 1000000000, 1, 1000000000]**, the function should return **999999998**. The maximum possible income is **2999999998**, whose last 9 digits are **999999998**.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..200,000]**;
* each element of array **A** is an integer within the range **[0..1,000,000,000]**.

## 42 Sticks

### [Medium / 10 mins] Given two sticks, cut them into four sticks in order to create the largest possible square.

**Task description**

There are two wooden sticks of lengths **A** and **B** respectively. Each of them can be cut into shorter sticks of integer lengths. Our goal is to construct the largest possible square. In order to do this, we want to cut the sticks in such a way as to achieve four sticks of the same length (note that there can be some leftover pieces). What is the longest side of square that we can achieve?

Write a function:

function solution(A, B);

that, given two integers **A, B,** returns the side length of the largest square that we can obtain. If it is not possible to create any square, the function should return **0**.

**Examples:**

**1.** Given **A = 10, B = 21**, the function should return **7**. We can split the second stick into three sticks of length **7** and shorten the first stick by **3**.

**2.** Given **A = 13, B = 11**, the function should return **5**. We can cut two sticks of length 5 from each of the given sticks.

**3.** Given **A = 2, B = 1**, the function should return **0**. It is not possible to make any square from the given sticks.

**4.** Given **A = 1, B = 8**, the function should return **2**. We can cut stick **B** into four parts.

Write an efficient algorithm for the following assumptions:

* **A** and **B** are integers within the range **[1..1,000,000,000]**.

## 43 AngryFrogs

### [Medium / 50 mins] Given blocks in a row, find the distance between the two most distant blocks, such that the heights between them are first decreasing and then increasing.

**Task description**

There are **N** blocks, numbered from **0** to **N-1**, arranged in a row. A couple of frogs were sitting together on one block when they had a terrible quarrel. Now they want to jump away from one another so that the distance between them will be as large as possible. The distance between blocks numbered **J** and **K**, where **J ≤ K**, is computed as **K − J + 1**. The frogs can only jump up, meaning that they can move from one block to another only if the two blocks are adjacent and the second block is of the same or greater height as the first. What is the longest distance that they can possibly create between each other, if they also chose to sit on the optimal starting block initially?

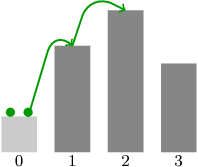
Write a function:

function solution(blocks);

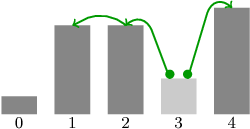
that, given an array **blocks** consisting of N integers denoting the heights of the blocks, returns the longest possible distance that two frogs can make between each other starting from one of the blocks.

**Examples:**

**1.** Given **blocks = [2, 6, 8, 5]**, the function should return **3**. If starting from **blocks[0]**, the first frog can stay where it is and the second frog can jump to **blocks[2]** (but not to **blocks[3]**).



**2.** Given **blocks = [1, 5, 5, 2, 6]**, the function should return **4**. If starting from **blocks[3]**, the first frog can jump to **blocks[1]**, but not **blocks[0]**, and the second frog can jump to **blocks[4]**.



**3.** Given **blocks = [1, 1]**, the function should return **2**. If starting from **blocks[1]**, the first frog can jump to **blocks[0]** and the second frog can stay where it is. Starting from **blocks[0]** would result in the same distance.

"Graphical representation of example 3."

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[2..200,000]**;
* each element of array **blocks** is an integer within the range **[1..1,000,000,000]**.

## 44 AssassinsStealth

### [Medium / 50 mins] Given a two-dimensional board with an assassin and guards on it, determine whether the assassin can get to the bottom-right corner undetected.

**Task description**

We are given a two-dimensional board of size **N × M** (**N** rows and **M** columns). Each field of the board can be empty ('**.**'), may contain an obstacle ('**X**') or may have a character in it. The character might be either an assassin ('**A**') or a guard. Each guard stands still and looks straight ahead, in the direction they are facing.

Every guard looks in one of four directions (up, down, left or right on the board) and is represented by one of four symbols. A guard denoted by '**<'** is looking to the left; by '**>**', to the right; '**^**', up; or '**v**', down. The guards can see everything in a straight line in the direction in which they are facing, as far as the first obstacle ('**X**' or any other guard) or the edge of the board.

The assassin can move from the current field to any other empty field with a shared edge. The assassin cannot move onto fields containing obstacles or enemies.

Write a function:

function solution(B);

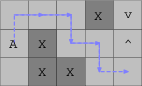
that, given an array **B** consisting of **N** strings denoting rows of the array, returns **true** if is it possible for the assassin to sneak from their current location to the bottom-right cell of the board undetected, and **false** otherwise.

**Examples:**

**1.** Given **B = ["X.....>", "..v..X.", ".>..X..", "A......"]**, your function should return **false**. All available paths lead through a field observed by a guard.



**2.** Given **B = ["...Xv", "AX..^", ".XX.."]**, your function should return **true**. The guard in the second row is blocking the other one from watching the bottom-right square.



**3.** Given **B = ["...", ">.A"]**, your function should return **false**, as the assassin gets spotted right at the start.

Graphical representation of the third example

**4.** Given **B = ["A.v", “...”]**, your function should return **false**. It's not possible for the assassin to enter the bottom-right cell undetected, as the cell is observed.

Graphical representation of the fourth example

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..500]**;
* all strings in **B** are of the same length **M** from range **[1..500]**;
* there is exactly one assassin on the board;
* there is no guard or wall on **B[N−1][M−1]**;
* every string in **B** consists only of the following characters **'.', 'X', '<', '>', 'v', '^'** and/or '**A**'.

## 45 ChoosingNumbers

### [Medium / 50 mins] Choose the maximum number of integers from an array so that, after sorting, the difference between every adjacent pair is equal.

**Task description**

There is an array **A** made of **N** integers. Your task is to choose as many integers from **A** as possible so that, when they are put in ascending order, all of the differences between all pairs of consecutive integers are equal.

For example, for **A = [4, 3, 5, 1, 4, 4]**, you could choose **1, 3** and **5** (with differences equal to **2**) or **4, 4** and **4** (with differences equal to **0**).

What is the maximum number of integers that can be chosen?

Write a function:

function solution(A);

that, given an array **A** made of **N** integers, returns the maximum number of integers that can be chosen following the rules described above.

**Examples:**

**1.** For **A = [4, 7, 1, 5, 3]**, the function should return **4**. It is possible to choose four integers (**7, 1, 5** and **3**). When put in ascending order, the difference between all consecutive integers is **2**.

**2.** For **A = [12, 12, 12, 15, 10]**, the function should return **3**. It is optimal to choose all integers with a value of **12**.

**3.** For **A = [18, 26, 18, 24, 24, 20, 22]**, the function should return **5**. Five integers (**18, 20, 22, 24, 26**) can be chosen. Notice that we cannot pick any other integers, even though they occur more than once.

Assume that:

* **N** is an integer within the range **[2..50]**;
* each element of array **A** is an integer within the range **[1..100]**.

In your solution, focus on correctness. The performance of your solution will not be the focus of the assessment.

## 46 CleaningRobot

### [Medium / 50 mins] Count the number of squares on a rectangular grid visited by a cleaning robot. The robot only moves forward, turning right if it cannot make a move.

**Task description**

There is a cleaning robot which is cleaning a rectangular grid of size **N x M**, represented by array **R** consisting of **N** strings. Rows are numbered from **0** to **N−1** (from top to bottom) and columns are numbered from **0** to **M−1** (from left to right).

The robot starts cleaning in the top-left corner, facing rightwards. It moves in a straight line for as long as it can, in other words, while there is an unoccupied grid square ahead of it. When it cannot move forward, it rotates 90 degrees clockwise and tries to move forward again until it encounters another obstacle, and so on. Dots in the array ("**.**") represent empty squares and "**X**"s represent occupied squares (ones the robot cannot move through). Each square that the robot occupied at least once is considered clean. The robot moves indefinitely.

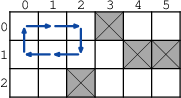
Write a function:

function solution(R);

that, given an array **R** consisting of **N** strings, each of length **M**, representing the grid, returns the number of clean squares.

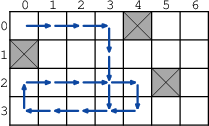
**Examples:**

**1.** Given **A = ["...X..", "....XX", "..X..."]**, your function should return **6**.

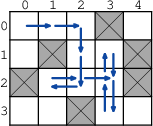


The robot starts at **(0,0)**, facing rightwards, and moves to **(0,2)**, where it turns due to the obstacle at **(0,3)**. Then it goes down from **(0,2)** to **(1,2)**, where it changes direction again due to another obstacle. Next it goes left from **(1, 2)** to **(1,0)**, where it turns once because of the grid boundary, then it moves once and turns once more, which makes it stand again at position **(0,0)** facing rightwards, just as at the beginning, which means it will now repeat the loop indefinitely. The total number of cleaned squares is **6**.

**2.** Given **A = ["....X..", "X......", ".....X.", "......."]**, your function should return **15**.



**3.** Given **A = ["...X.", ".X..X", "X...X", "..X.."]**, your function should return **9**.



**4.** Given **A = ["."]**, your function should return **1**, because there is only one square on the grid and it is cleaned in the first move.

Assume that:

* **N** and **M** are integers within the range **[1..20]**;
* top-left cell is empty;
* each string in **R** consists only of the following characters: "**.**" and/or "**X**".

In your solution, focus on correctness. The performance of your solution will not be the focus of the assessment.

## 47 CollectingRainwater

### [Medium / 50 mins] Find the minimum number of water tanks needed to save the rainwater from all houses in a street.

**Task description**

Today our world is approaching an ecological crisis. Due to global warming, the sea level is rising. At the same time, the amount of drinkable water is decreasing. One idea about preventing the loss of drinkable water is the propagation of rainwater storage, in other words, equipping houses with a water tank for rainwater.

You are given a string **S** describing a street, in which '**H**' denotes a house and '**−**' denotes an empty plot. You may place water tanks in empty plots to collect rainwater from nearby houses. A house can collect its own rainwater if there is a tank next to it (on either the left or the right side).

Your task is to find the minimum number of water tanks needed to collect rainwater from all of the houses.

For example, given **S = "−H−HH−−"**, you can collect rainwater from all three houses by using two water tanks. You can position one water tank between the first and second houses and the other after the third house. This placement of water tanks can be represented as **"−HTHHT−"**, where **'T'** denotes a water tank.

Write a function:

function solution(S);

that, given a string **S** of length **N**, returns the minimum number of water tanks needed.

If there is no solution, return **-1**.

**Examples:**

**1.** Given **S = "−H−HH−−"**, the function should return **2**, as explained above.

**2.** Given **S = "H"**, the function should return **-1**. There is no available plot on which to place a water tank.

**3.** Given **S = "HH−HH"**, the function should return **-1**. There is only one plot to put a water tank, and it is impossible to collect rainwater from the first and last houses.

**4.** Given **S = "−H−H−H−H−H"**, the function should return **3**. One possible way of placing water tanks is **"−HTH−HTHTH"**.

Assume that:

* **N** is an integer within the range **[1..20]**;
* string **S** is made only of the characters **'-'** and/or **'H'**.

In your solution, focus on correctness. The performance of your solution will not be the focus of the assessment.

## 48 CreateDiverseWord

### [Medium / 50 mins] Join some of the given strings to create the longest possible string that contains neither "AAA" nor "BBB" as a fragment.

**Task description**

There are two-letter strings, **“AA”, “AB”** and **“BB”**, which appear **AA, AB** and **BB** times respectively. The task is to join some of these strings to create the longest possible string which does not contain **“AAA”** or **“BBB”**.

For example, having **AA = 5, AB = 0** and **BB = 2**, it is possible to join five strings by taking both of the **"BB"** strings and three of the **"AA"** strings. Then they can be joined into **"AA-BB-AA-BB-AA" → "AABBAABBAA"**.  
Note that it is not possible to add another **"AA"** string as the result would then contain **"AAA"**.

Write a function:

function solution(AA, AB, BB);

that, given three integers **AA**, **AB** and **BB**, returns the longest string that can be created according to the rules described above. If there is more than one possible answer, the function may return any of them.

**Examples:**

**1.** Given **AA = 5, AB = 0** and **BB = 2**, the function should return **"AABBAABBAA"**, as explained above.

**2.** Given **AA = 1, AB = 2** and **BB = 1**, possible results are **"BBABABAA", "ABAABBAB", "ABABAABB"** or **"AABBABAB"**.

**3.** Given **AA = 0**, **AB = 2** and **BB = 0**, the function should return **"ABAB"**.

**4.** Given **AA = 0**, **AB = 0** and **BB = 10**, the function should return **"BB"**.

Assume that:

* **AA, AB** and **BB** are integers within the range **[0..10]**;
* the resulting string will not be empty.

In your solution, focus on correctness. The performance of your solution will not be the focus of the assessment.

## 49 DivideIntoGroups

### [Medium / 50 mins] Divide people into three groups so that the largest score difference between the people in each group is as small as possible.

**Task description**

There is an array **A** consisting of **N** integers. Divide them into three non-empty groups. In each group we calculate the difference between the largest and smallest integer. Our goal is to make the maximum of these differences as small as possible.

For example, given **A = [11, 5, 3, 12, 6, 8, 1, 7, 4]**, we can divide the elements into three groups:

* **[3, 1, 4]** − the difference between elements is **3**;
* **[5, 6, 8, 7]** − the difference is also **3**;
* **[11, 12]** − the difference is **1**.

The maximum difference equals **3**, which is the minimum possible result.

Write a function:

function solution(A);

that, given an array **A**, returns the minimum possible result as explained above.

**Examples:**

**1.** For **A = [11, 5, 3, 12, 6, 8, 1, 7, 4]**, the function should return **3**, as explained above.

**2.** For **A = [10, 14, 12, 1000, 11, 15, 13, 1]**, the function should return **5**. The elements of A should be divided into three groups as follows:

* **[1];**
* **[10, 14, 12, 11, 15, 13];**
* **[1000].**

**3.** For **A = [4, 5, 7, 10, 10, 12, 12, 12]**, the function should return **2**. The elements of **A** could be divided into these three groups:

* **[4, 5];**
* **[7];**
* **[10, 10, 12, 12].**

**4.** For **A = [5, 10, 10, 5, 5]**, the function should return **0**. The first group may contain all elements with value **5**; the second and the third groups may each contain one element with value **10**.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[3..100,000]**;
* each element of array **A** is an integer within the range **[0..1,000,000,000]**.

## 50 SquareTiles

### [Medium / 50 mins] Given M tiles of size 1x1 and N tiles of size 2x2, calculate the size of the largest square you can create out of these tiles.

**Task description**

You have **M** square tiles of size **1×1** and **N** square tiles of size **2×2**. Your task is to create the largest possible square using these tiles. Tiles may not overlap, and the resulting square should be filled (it should not contain empty spaces).

Write a function:

function solution(M, N);

that, given two integers **M** and **N**, returns the length of the side of the largest square you can create. If no square can be created, your function should return **0**.

**Examples:**

**1.** Given **M = 8** and **N = 0**, your function should return **2**. You can use four out of eight tiles to arrange them into **2×2** square. There are not enough tiles to create **3×3** square.

**2.** Given **M = 4** and **N = 3**, your function should return **4**. You can obtain a **4×4** square by arranging four **1×1** tiles into a **2×2** square, and surrounding it by **2×2** tiles:

A black background with a black square

Description automatically generated with medium confidence

**3.** Given **M = 0** and **N = 18**, your function should return **8**. You need to use sixteen **2×2** tiles to create the square. Note that not all the tiles are used.

**4.** Given **M = 13** and **N = 3**, your function should return **5**. One of the possible arrangements is shown in the following image:

A black background with a black square

Description automatically generated with medium confidence

Write an efficient algorithm for the following assumptions:

* **M** and **N** are integers within the range **[0..1,000,000,000]**.

## 51 StringMisspellingVar (Opal variant)

### [Medium / 50 mins] Given two strings, check whether one of them can be obtained from the other by performing a simple edit operation.

**Task description**

Write a function:

function solution(S, T);

that, given two strings **S** and **T** consisting of **N** and **M** characters, respectively, determines whether string **T** can be obtained from string **S** by at most one simple operation from the set specified below. The function should return a string:

* "**INSERT c**" if string **T** can be obtained from string **S** by inserting a single character "**c**" at the beginning of the string;
* "**REMOVE c**" if string **T** can be obtained from string **S** by deleting a single character "**c**" from the end of the string;
* "**SWAP c d**" if string **T** can be obtained from string **S** by swapping two adjacent characters "**c**" and "**d**" (these characters should be distinct and in the same order as in string **S**; exactly one swap is performed);
* "**EQUAL**" if no operation is needed (strings **T** and **S** are equal);
* "**IMPOSSIBLE**" if none of the above works.

Note that by characters "**c**" and "**d**" from the operations above, we mean any English alphabet lowercase letters .

For example:

* given **S = "gain"** and **T = "again"**, the function should return **"INSERT a"**;
* given **S = "parks"** and **T = "park"**, the function should return **"REMOVE s"**;
* given **S = "form"** and **T = "from"**, the function should return **"SWAP o r"**;
* given **S = "o"** and **T = "odd"**, the function should return **"IMPOSSIBLE"**;
* given **S = "fift"** and **T = "fifth"**, the function should return **"IMPOSSIBLE"**.

Assume that:

* **N** and **M** are integers within the range **[1..100]**;
* strings **S** and **T** are made only of lowercase letters **(a−z)**.

In your solution, focus on correctness. The performance of your solution will not be the focus of the assessment.

## 52 TwoWayRoadRenovation

### [Medium / 50 mins] Given a representation of a two-lane road, select at most two stretches (one in each lane) that cover the maximum number of potholes in total and that allow the road to remain open to traffic.

**Task description**

You are given a description of a two-lane road in which two strings, **L1** and **L2**, respectively represent the first and the second lane, each lane consisting of **N** segments of equal length.

The **K-th** segment of the first lane is represented by **L1[K]** and the **K-th** segment of the second lane is represented by **L2[K]**, where **'.'** denotes a smooth segment of road and **'x'** denotes a segment containing potholes.

Cars can drive over segments with potholes, but it is rather uncomfortable. Therefore, a project to repair as many potholes as possible was submitted. At most one contiguous stretch of each lane may be repaired at a time. For the time of reparation those stretches will be closed to traffic.

How many road segments with potholes can be repaired given that the road must be kept open (in other words, stretches of roadworks must not prevent travel from one end of the road to the other)?

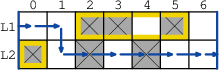
Write a function:

function solution(L1, L2);

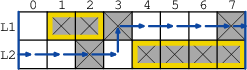
that, given two strings **L1** and **L2** of length **N**, returns the maximum number of segments with potholes that can be repaired.

**Examples:**

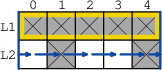
**1.** Given **L1 = "..xx.x."** and **L2 = "x.x.x.."**, your function should return **4**. It is possible to repair three potholes in the first lane and the first pothole in the second lane without closing the road to traffic.



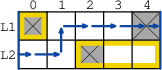
**2.** Given **L1 = ".xxx...x"** and **L2 = "..x.xxxx"**, your function should return **6**.



**3.** Given **L1 = "xxxxx"** and **L2 = ".x..x"**, your function should return **5**.



**4.** Given **L1 = "x...x"** and **L2 = "..x.."**, your function should return **2**.



Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..200,000]**;
* strings **L1** and **L2** consist only of the characters **"."** and/or **"x"**.

## 53 WordSplit

### [Medium / 20 mins] Given a string consisting only of lowercase letters, split it into a minimal number of substrings such that each substring contains no letter more than once.

**Task description**

You are given a string consisting of lowercase letters of the English alphabet. You must split this string into a minimal number of substrings in such a way that no letter occurs more than once in each substring.

For example, here are some correct splits of the string **"abacdec"**: (**'a', 'bac', 'dec'**), (**'a', bacd', 'ec'**) and (**'ab', 'ac', 'dec'**).

Write a function:

function solution(S);

that, given a string **S** of length **N**, returns the minimum number of substrings into which the string has to be split.

**Examples:**

**1.** Given **'world'**, your function should return **1**. There is no need to split the string into substrings as all letters occur just once.

**2.** Given **'dddd'**, your function should return **4**. The result can be achieved by splitting the string into four substrings (**'d', 'd', 'd', 'd'**).

**3.** Given **'cycle'**, your function should return **2**. The result can be achieved by splitting the string into two substrings (**'cy', 'cle'**) or (**'c', 'ycle'**).

**4.** Given **'abba'**, your function should return **2**. The result can be achieved by splitting the string into two substrings (**'ab', 'ba'**).

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..1,000,000]**;
* string **S** is made only of lowercase letters **(a−z)**.

## 54 RecyclingTrucks

### [Medium / 50 mins] Calculate the time needed by three recycling trucks to collect all trash in a street.

**Task description**

There are **N** houses (numbered from **0** to **N-1**) along a street. In each of them, recyclable trash (plastic, glass, metal) is collected into separate bags.

There are three trucks that collect the trash. Each of them collects a separate type of trash (the first collects plastic, the second, glass and the third, metal). All the trucks begin and end their jobs at the starting end of the street. Passing from the starting point to house number **0** takes **D[0]** minutes. Passing between houses number **K-1** and **K** (for **K** in the range **1** to **N-1**) takes **D[K]** minutes. Loading one bag onto the truck takes one minute.

For example, **D = [2, 5]** means that passing between the starting point and house number **0** takes **2** minutes and passing between houses number **0** and **1** takes **5** minutes.

Each of the houses has already collected some bags (or possibly no bags) of recyclable trash. The number of bags that house number **K** has collected is recorded in string **T[K]**, composed of letters **'P'** (plastic), **'G'** (glass) and **'M'** (metal). For example, **T[1]** = **"GMG"** means that house number **1** has collected two bags of glass and one bag of metal. Each house may collect more than one bag of each type.

All of the trucks start their jobs simultaneously. Each finishes its job after collecting all of the bags of the given type of trash and returning back to the starting point. What is the minimum number of minutes that will pass before all the trucks finish all the jobs?

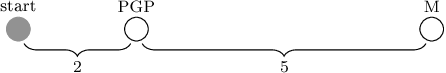
Write a function:

function solution(D, T);

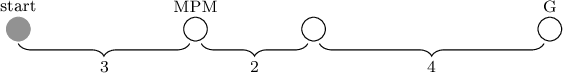
that, given array **D** of **N** integers and array **T** of **N** strings, returns the minimum number of minutes needed by the trucks to finish all the jobs.

**Examples:**

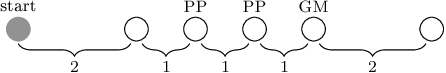
**1.** Given **D = [2, 5], T = ["PGP", "M"]**, the function should return **15**. The truck collecting plastic needs to go to house number **0**, collect two bags and go back, which takes **2 + 1 + 1 + 2 = 6** minutes. The truck collecting glass needs **5** minutes: **2** minutes to go to house number **0, 1** minute to collect a bag and **2** minutes to return to the starting point. The truck collecting metal will go straight to house number **1**, collect a bag and go back in **7 + 1 + 7 = 15** minutes. After **15** minutes all of the trash will have been collected and all trucks will have returned to the starting point.



**2.** Given **D = [3, 2, 4], T = ["MPM", "", "G"]**, the function should return **19**. The truck collecting glass needs the most time: **3 + 2 + 4 + 1 + 4 + 2 + 3 = 19**.



**3.** Given **D = [2, 1, 1, 1, 2], T = ["", "PP", "PP", "GM", ""]**, the function should return **12**. The truck collecting plastic needs **12** minutes, whereas the trucks collecting glass and metal both need **11** minutes to finish their jobs.



Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..100,000]**;
* each element of array **D** is an integer within the range **[1..100]**;
* each element of array **T** is a string consisting of letters **'P', 'G' or 'M'**;
* **S**, the total length of all strings, is an integer within the range **[0..100,000]**.

## 55 RoadRepair

### [Medium / 50 mins] Find how many potholes in a road can be fixed within a given budget if fixing K consecutive potholes costs K+1.

**Task description**

You are given a task to fix potholes in a road. The road is described by a string **S** consisting of **N** characters. Each character represents a single fragment of the road. Character **'.'** denotes a smooth surface and **'x'** denotes a pothole. For example, **S = "...xxx..x"** means that the road starts with three smooth fragments, followed by three potholes, followed by two smooth fragments and ending with one pothole.

You can choose any number of consecutive potholes and fix all of them. Fixing a segment consisting of **K** consecutive potholes costs **K + 1**. In the example above, fixing the first two consecutive potholes costs **2 + 1 = 3** and fixing the last pothole costs **1 + 1 = 2**. After these fixes, the road would look like this: **".....x..."**.

You are given a budget **B**. You can fix multiple segments containing potholes as long as you fit in the budget. What is the maximum number of potholes you can fix?

Write a function:

function solution(S, B);

that, given the string **S** of length **N** and the integer **B**, returns the maximum number of potholes that can be fixed.

**Examples:**

**1.** Given **S = "...xxx..x....xxx."** and **B = 7**, the function should return **5**. You can start by fixing the first three consecutive potholes for a cost of **4**, obtaining the road: **"........x....xxx."**. Then, you can fix the last two potholes for a cost of 3, obtaining the road: **"........x....x..."**. The total cost is **7**, which fits in the budget, and you fix **5** potholes in total.

**2.** Given **S = "..xxxxx"** and **B = 4**, the function should return **3**. One way is to fix the middle three potholes, which costs the whole budget and makes the road look as follows: **"..x...x"**. Alternatively, you could fix the first three potholes or the last three potholes.

**3.** Given **S = "x.x.xxx...x"** and **B = 14**, the function should return **6**. You can fix all the potholes, which costs **2 + 2 + 4 + 2 = 10**, leaving you with the spare budget of **4**. This fixes the entire road.

**4.** Given **S = ".."** and **B = 5**, the function should return **0**. There are no potholes to fix.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..100,000]**;
* **B** is an integer within the range **[0..200,000]**;
* string **S** consists only of characters **'.'** and **'x'**.

## 56 RoadRoller

### [Medium / 50 mins] Find the minimum number of road roller drives needed to patch all the potholes in a road.

**Task description**

There are **N** potholes on a straight road parallel to the **y-axis**. The positions of the potholes are described by two arrays of integers, **X** and **Y**. The **K-th** pothole (for **K** within the range **[0..N-1]**) is located at coordinates **(X[K], Y[K])**.

In order to fix the potholes, a road roller of width **W** will be used. The road roller can only drive along the street (parallel to the **y-axis**). It can start driving from any x coordinate at the beginning of the road (a point whose **y** coordinate is equal to **0**). During one drive, for a chosen starting point **(x, 0)** of the road roller's left end, the road roller drives upwards and patches all potholes on its way that fall within the width of the road roller, **W**, and are to the right of its starting position, **x**. In other words, it patches all the potholes whose first coordinate is between **x** and **x+W** inclusive.

What is the minimum number of road roller drives required to patch all the potholes?

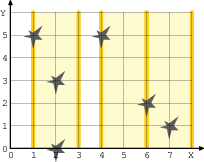
Write a function:

int solution(X, Y, W);

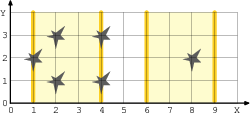
that, given two arrays of integers X and Y, describing the positions of the N potholes, and an integer W, specifying the width of the road roller, returns the minimum number of drives needed to patch all the potholes.

**Examples:**

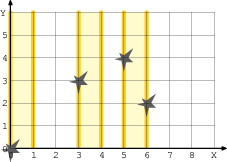
**1.** Given **X = [2, 4, 2, 6, 7, 1], Y = [0, 5, 3, 2, 1, 5]** and **W = 2**, the answer is **3**. At least three drives will be needed to patch all the potholes. For example, the drives could start respectively at points **(1, 0), (4, 0)** and **(6, 0)**. During the first drive, the road roller would patch the potholes located at **(2, 0), (2, 3)** and **(1, 5)**. During the second drive, the potholes located at **(4, 5)** and **(6, 2)** would be patched. Finally, in the third drive, the pothole located at **(7, 1)** would be patched.



**2.** Given **X = [4, 8, 2, 2, 1, 4], Y = [1, 2, 3, 1, 2, 3]** and **W = 3**, the answer is **2**. If the first drive of the road roller started at point **(1, 0)** and the second drive at the point **(6, 0)**, all the potholes would be patched.



**3.** Given **X = [0, 3, 6, 5], Y = [0, 3, 2, 4]** and **W = 1**, the answer is **3**. The first drive of the road roller could start at point **(0, 0)**, the second drive at point **(3, 0)** and the third at point **(5, 0)**.



Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..100,000]**;
* each element of arrays **X** and **Y** is an integer within the range **[0..1,000,000,000]**;
* **W** is an integer within the range **[1..1,000,000,000]**.

## 57 PutThreeTiles

### [Hard / 20 mins] Given an array of integers, calculate the maximum sum of numbers that can be covered using at most three tiles.

**Task description**

There is an array **A** of **N** integers and three tiles. Each tile can cover two neighbouring numbers from the array but cannot intersect with another tile. It also cannot be placed outside the array, even partially.

Write a function:

function solution(A);

that, given an array **A** of **N** integers, returns the maximum sum of numbers that can be covered using at most three tiles.

**Examples:**

**1.** Given **A = [2, 3, 5, 2, 3, 4, 6, 4, 1]**, the function should return **25**. There is only one optimal placement of tiles: **(3, 5), (3, 4), (6, 4)**.

**2.** Given **A = [1, 5, 3, 2, 6, 6, 10, 4, 7, 2, 1]**, the function should return **35**. One of the three optimal placements of tiles is **(5, 3), (6, 10), (4, 7)**.

**3.** Given **A = [1, 2, 3, 3, 2]**, the function should return **10**. There is one optimal placement of tiles: **(2, 3), (3, 2)**. Only two tiles can be used because **A** is too small to contain another one.

**4.** Given **A = [5, 10, 3]**, the function should return **15**. Only one tile can be used.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[2..100,000]**;
* each element of array **A** is an integer within the range **[0..1,000,000]**.

## 58 FinancialPlan

### [Hard / 60 mins] How many expenses must be rescheduled to the end of the year so that the company doesn't fall into debt?

**Task description**

A company has a list of expected revenues and payments for the upcoming year in chronological order. The problem is that at some moments in time the sum of previous payments can be larger than the total previous revenue. This would put the company in debt. To avoid this problem the company takes a very simple approach. It reschedules some expenses to the end of the year.

You are given an array of integers, where positive numbers represent revenues and negative numbers represent expenses, all in chronological order. In one move you can relocate any expense (negative number) to the end of the array. What is the minimum number of such relocations to make sure that the company never falls into debt? In other words: you need to make sure that there is no consecutive sequence of elements starting from the beginning of the array, that sums up to a negative number.

You can assume that the sum of all elements in **A** is nonnegative.

Write a function:

function solution(A);

that, given an array **A** of **N** integers, returns the minimum number of relocations, so that company never falls into debt.

**Examples:**

**1.** Given **A = [10, -10, -1, -1, 10]**, the function should return **1**. It is enough to move **-10** to the end of the array.

**2.** Given **A = [-1, -1, -1, 1, 1, 1, 1]**, the function should return **3**. The negative elements at the beginning must be moved to the end to avoid debt at the start of the year.

**3.** Given **A = [5, -2, -3, 1]**, the function should return **0**. The company balance is always nonnegative.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..100,000]**;
* each element of array **A** is an integer within the range **[−1,000,000,000..1,000,000,000]**;
* sum of all elements in **A** is greater than or equal to **0**.

## 59 DecreasePollution

### [Hard / 60 mins] Given an array of integers, find the minimum number of times its elements must be divided by 2 in order to reduce the sum of the array by at least half.

**Task description**

An industrial company has **N** factories, each producing some pollution every month. The company has decided to reduce its total fume emissions by equipping some of the factories with one or more filters.

Every such filter reduces the pollution of a factory by half. When a second (or subsequent) filter is applied, it again reduces by half the remaining pollution emitted after fitting the existing filter(s). For example, a factory that initially produces **6** units of pollution will generate **3** units with one filter, **1.5** units with two filters and **0.75** units with three filters.

You are given an array of **N** integers describing the amount of pollution produced by the factories. Your task is to find the minimum number of filters needed to decrease the total sum of pollution by at least half.

Write a function:

function solution(A);

which, given an array of integers **A** of length **N**, returns the minimum number of filters needed to reduce the total pollution by at least half.

**Examples:**

**1.** Given **A = [5, 19, 8, 1]**, your function should return **3**. The initial total pollution is **5 + 19 + 8 + 1 = 33**. We install two filters at the factory producing **19** units and one filter at the factory producing **8** units. Then the total pollution will be **5 + ((19 / 2) / 2) + (8 / 2) + 1 = 5 + 4.75 + 4 + 1 = 14.75** which is less than **33 / 2 = 16.5**, so we have achieved our goal.

**2.** Given **A = [10, 10]**, your function should return **2**, because we may install one filter at each factory.

**3.** Given **A = [3, 0, 5]**, your function should return **2**, because it is sufficient to install one filter at each factory producing a non-zero amount of pollution.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[1..30,000]**;
* each element of array **A** is an integer within the range **[0..70,000]**.

## 60 AngleBrackets

### [Hard / 60 mins] Find the maximum possible length of a symmetric fragment which can be obtained after replacing question marks in a given string with "<" or ">".

**Task description**

A string made of an even number of characters (**"<"** and/or **">"**) is called *symmetric* if all characters in its first half are **"<"** and all characters in its second half are **">"**. Examples of symmetric strings are: **""** (empty string), **"<>"**, **"<<>>"**, **"<<<>>>"**, etc.

Write a function:

function solution(S);

that, given a string **S** made of **N** characters (**"<"**, **">"** and/or **"?")"**, returns the length of the longest symmetric substring that can be obtained after replacing question marks with **"<"** or **">"** characters.

**Examples:**

**1.** For **S = "<><??>>"**, after replacing all question marks with **"<"**, the string **"<><<<>>"** is obtained. Its longest symmetric substring is **"<<>>"**, so the function should return **4**.

**2.** For **S = "??????"**, the optimal option is to replace the first three question marks with **"<"** and the next three question marks with **">"**, thus obtaining the string **"<<<>>>"**. The function should return **6**.

**3.** For **S = "<<?"**, the function should return **2**.

Write an efficient algorithm for the following assumptions:

* the length of string **S** is within the range **[1..200,000];**
* string **S** is made only of the following characters: **'<', '>'** and/or **'?'**.

## 61 SameCorners

### [Hard / 60 mins] Given an array, find the subarray with the largest sum in which the leftmost and rightmost elements are the same.

**Task description**

You are given an array **A** consisting of **N** positive integers. Consider subarrays of **A**, with at least two elements, whose first and last elements have the same value. Your task is to find the largest possible sum of such a subarray.

For example, for array **A = [1, 3, 6, 1, 6, 6, 9, 9]**, the following subarrays meet the requirements:

* **[1, 3, 6, 1]**: first and last elements are equal to **1**; the sum of the subarray is **11**.
* **[6, 1, 6]**: first and last elements are equal to **6**; the sum of the subarray is **13**.
* **[6, 1, 6, 6]**: first and last elements are equal to **6**; the sum of the subarray is **19**.
* **[6, 6]**: first and last elements are equal to **6**; the sum of the subarray is **12**.
* **[9, 9]**: first and last elements are equal to **9**; the sum of the subarray is **18**.

The subarray with the largest sum is **[6, 1, 6, 6]** and its sum is **19**.

Write a function:

function solution(A);

that, given an array **A** of **N** positive integers, returns the largest sum of a subarray whose first and last elements have the same value.

If there is no such subarray, return **-1**.

**Examples:**

**1.** Given **A = [1, 3, 6, 1, 6, 6, 9, 9]**, the function should return **19**, as explained above.

**2.** Given **A = [5, 1, 4, 3]**, function should return **-1**. There is no subarray satisfying the requirements.

**3.** Given **A = [2, 2, 2, 3, 2, 3]**, function should return **11**. The subarray with the largest sum is **[2, 2, 2, 3, 2]**.

Write an efficient algorithm for the following assumptions:

* **N** is an integer within the range **[2..100,000]**;
* each element of array **A** is an integer within the range **[1..10,000].**