CodeSignal Questions

1. There are some lamps placed on a coordinate line. Each of these lamps illuminates some space around it within a given radius. You are given the coordinates of the lamps on the line and the effective radius of each of the lamps’ light.

In other words, you are given a two-dimensional array *lamps* where *lamps[i]* contains information about the *ith* lamp.

* *Lamps[i][0]* is an integer representing the lamp’s coordindate
* *Lamps[i][1]* is a positive integer representing the lamp’s effective radius. That means the *ith* lamp illuminates everything in a range from *lamps[i][0] – lamps[i][1] to lamps[i][0] + lamps[i][1]* inclusive

Your task is to find the number of integer coordinates that are illuminated by exactly 1 lamp.

Ex. *Lamps = [-2,3], [2,3], [2,1]* the output should be *solution(lamps) = 6*

Answer: Question 1

1. Imagine that you are standing at the starting point of a straight street and are trying to reach the end of the street. This street is represented by a number line starting at 0 and ending at finish ( finish > 0 ). There are electric scooters scattered along the street to help you get to the end. Specially, the scooters are represented by an array scooters, with scooters Lil representing the location of the ith scooter. Each scooter can travel up to 10 points along the number line before its battery is fully discharged and it cannot go further. For example, if a scooter is located at point s, it can travel to points 5 6 6 7 up to point 15 (inclusive), but it cannot get to point 16 or further.

Answer: Question 2

1. You are given a text represented as a string *t*, and a string *s* of length 3. Your task is to count the number of indices *i* such that titi+2ti+4=*s*

For *t = “azcabcab”* and *s = “acb”* the output should be *solution(t,s) = 2*

For *t = “”* and *s = “xyz”* the output should be *solution(t,s) = 0*

Answer: Question 3

1. You are given an array of integers *numbers* and an integer *pivot*. Let *countGreater* denote how many integers wihin *numbers* are greater than *pivot*, and *countless* denote how many integers are less than *pivot*. Your task is to computer *countGreater* and *countless* then return:
   1. *“greater* if *countGreater* is greater than *countless*
   2. *“smaller”* if *countGreater* is less than *countless*
   3. *“tie”* if they are equal

Ex. For *numbers = [1 3 0 -1 1 4 3]* and *pivot = 2*, the output should be *solution(numbers, pivot) = “smaller”*

Answer: Question 4

1. You are given *numbers* an array of non-negative integers. Your task is to perform the following algorithm on this array:
   1. Step 1 – find the index *I* of the leftmost non-zero element *numbers[i] x != 0*. If there is no such element, finish the algorithm.
   2. Step 2 – Start at index *i* and going to the right, attempt to subtract *x* from each element:
      1. If the element is strictly less than *x* move on to step 3
      2. Otherwise subtract *x* from the element and move on to the next element
      3. If you reach the end of the array, move on to step 3
   3. Step 3- Add *x* to the final result
   4. Step 4 – Go back to step 1

Return the resulting sum obtained from step 3. It is guaranteed that the algorithm is finite and will finish at some point.

For *numbers = [3,3,5,2,3]*, the output should be *solution(numbers) = 6.*

Answer: Question 5

1. You are given *n*, an integer representing the length of a binary string *a*, which is all *‘0’*s in the beginning. You are also given *operations*, an array of strings, each representing an operation of one of these two types:
   1. *L* – find the smallest index *i* for which *a[i] = ‘0’* and set *a[i] = ‘1’*. If there is no such index, do nothing.
   2. *C[ind]* – set *a[ind] = ‘0’*. This operation does not depend on the previous value of *a[ind]*. It is guaranteed that *ind* is a valid 0-based index of *a* (i.e *ind < n*)

Given *n* and *operations*, your task is to return *a*, the binary string after all operations have been applied

Ex. *n = 10* and *operations = [L, L, C0, L, C3],* the output should be *solution(n, operations) = 1100000000*

Alternative Wording:

You are given a binary array *state* consisting of integers 0 and 1. You are also given *operations* – an array of strings, each representing an operation of one of two types:

* “L” – find the smallest index *i*, for which *state[i] = 0* and set *state[i] = 1*.
  + If there is no such state, do nothing
* “C[index]” – set *state[index] = 0*.
  + This operation does not depend on the previous value of *state[index].*
  + It is guaranteed that *index* is a valid 0-based index of *state* (i.e *index < state.length*)

Given *state* and *operations*, your task is to return the binary string of state array after all the operations have been applied

Answer: Question 6

1. Given an array of positive integers *numbers* and a positive number *separation.* Find all the minimal difference between elements of *numbers* with indices that are at least *separation* apart. Formally your task is to find *min(|numbers[i] = numbers[j])|)* where *|i-j| >= separation*

Note – assume that *numbers* array indexing is 0-based.

Ex. numbers = [1,5,4,10,9] and *separation = 3*, the output should be *solution(numbers, separation) = 4*

Answer: Question 7

1. For an array *nums* and an integer *t* (, let’s define a cyclic t-shift operation as carrying *t* elements from the end of the array to the beginning.

For example, applying cyclic t-shift to array *nums*, having value *[nums[0], nums[1], nums[2], … nums[n-1]]* where *n* is the length of *nums:*

* For *t = 0* the cyclic t-shift would be *[nums[0], nums[1], nums[2], … nums[n-1]]*
* For *t = 1* the cyclic t-shift would be *[nums[n-1], nums[0], nums[1], nums[2], … nums[n-2]]*
* For *t = 2* the cyclic t-shift would be *[nums[n-2], nums[n-1], nums[0], nums[1], nums[2], … nums[n-3]]*
* For *t = n – 1*, the *n-1* cyclic will be *[[nums[1], nums[2], …, nums[n-1], nums[0]]*

Given an array of integers *nums*, find such *t*  that cyclic t-shift operation turns *nums* into a reverse sorted array *[n, n-1, …, 1]*. If there is no such t, return -1.

Answer: Question 8

1. You are given two arrays of integers *a* and *b*, and an array *queries* containing the queries you are required to process. Every *queries[i]* can have one of the following two items:
   1. *[0, i, x]*. In this case, you need to add *x* to the current value of *b[i]*
   2. *[i, x]*. In this case, you need to find the total number of pairs of indices *i* and *j* such that *a[i] + b[j] = x*

Perform the given queries in order and return an array containing the results of the queries of the type *[i, x]*

Ex. For *a = [1,2,3], b = [1,4]* and *queries = [[1,5], [0,0,2], [1,5]]*, the output should be *solution(a,b,queries) = [1,2].*

Ex. For *a = [1,2,2], b = [2,3]* and *queries = [[1,4], [0,0,1], [1,5]]* the output should be *solution(a,b,queries) = [3,4].*

Answer: Question 9

1. You are formatting the text on a newspaper page. You have to align the text on the page properly. The text is provided to you in the following format:
   1. *Paragraphs* – is an array of paragraphs, where each paragraph is represented as an array containing portions of text.
   2. *Width* – represents the maximum number of characters each line of the newspaper can include

Your task is to produce a newspaper page according to the following specifications:

* Start a new line for each paragraph *paragraphs[i].* Add all paragraph words *paragraphs[i][j]* in order. When words are together on a line, they should be separated by 1 space.
  + Every paragraph word *paragraphs[i][j]* is guaranteed to contain only lowercase and uppercase English letters.
* You can’t break up words. So, if adding the next word would cause you to exceed *width*, you will need to start a new line to add the word there.
* You may end up with leftover space on a line if the size of the text that fit on the line (including separating spaces) is less than *width*. If this happens, align the text to the cccenter by adding spaces around it:
  + If the amount of leftover space on the line is ***even***, add an equal number of spaces before and after the text.
  + If the amount of leftover space on the line is ***odd***, add an equal number of spaces before and after the text. You’ll have an extra space left over, add it ***after*** the text.
* Include a rectangular border of asterisk (*\** characters) around the top, bottom, left, and right edges of the resulting newspaper page. These characters don’t count towards the width but are added for aesthetic reasons.

Ex. *paragraphs = [*

*[“hello”, “world”],*

*[“How”, “areYou”, “doing”]*

*[“Please”, “look”, “and”, “align”, “to”, “the”, “center”]*

*]*

*Width = 16*

*Output should be solution(paragraphs, width) =*

*[“\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*”,* - 18 \*’s in top/bottom border

*“\* hello world \*”, - 2 spaces around*

*“\*How areYou doing\*”, - 0 spaces around*

*"\* align to the \*", - 2 spaces around*

*"\* center \*", - 5 spaces around*

*“\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*”]*

Answer: Question 10

1. You are given an array of integers *arr*. Consider it’s elements in pairs: *(arr[0], arr[1]), (arr[2], arr[3]),* and so on. Arrange the elements in each pair in ascending order. In other words, swap the position of paired elements if the left element is greater than the right.

If *arr* contains an odd number of elements the last element should be left unchanged.

Example:

* For *arr = [1, 5, 7, 3, 2, 1]*, the output should be *solution(arr) = [1,5,3,7,1,2]*

Answer: Question 11

1. In Unix, there are two common ways to execute a command:
   1. Enter it’s name, e.g. *“cp”* or *“ls”*
   2. Entering *“!<index>”*. This notation is used to repeat the *indexth* (1 – based) command since the start of the session. For example, suppose that the user has entered the following commands:
      1. ls
      2. cp
      3. mv
      4. mv
      5. mv
      6. !1
      7. !3
      8. !6

*“!1”* would trigger the execution of *“ls”*, *“!3”* would repeat *“mv”*, and *“!6”* would execute *“!1”* which in turn would trigger the execution of *“ls”*.

You are given a sequence of commands *commands* that the user has entered in the terminal since the start of the session. Each command can be one of the following: “cp”, “ls”, “mv”, or “!<index>”. Calculate the number of times each of “cp”, “ls”, and “mv” commands was executed and return an array of three integers in the following form: [# of times for “cp”, # of times for “ls”, # of times for “mv”]

Example:

* For commands = [“ls”, “cp”, “mv”, “mv”, “mv”, “!1”, “!3”, “!6”], the output should be solution(commands) = [1,3,4]

Answer: Question 12

1. Imagine a conversation between two users in a chat app. Given a two-dimensional string array *messages* representing messages from each user, your task if to render the conversation in a messenger window. Specifically, *messages[i]* represents the *ith* message in the conversation in the following format: *[<user>, <text>].* Note that *messages[i][0]* is either “1” or “2” to describe which user sent the message.

To render the messenger window, each message must start on a new line. You are also given two integers *width* and *userWidth*, where *width* represents the width of the entire messenger window, and *userWidth* represents the maximum number of characters that can be rendered on each line of the conversation. Messages which exceed *userWidth* should be rendered on multiple lines. Words in the message are not allowed to be split in the middle, and it is guaranteed that no word in the messages has length exceeding *userWidth*.

In the messenger window, messages from user “1” should be aligned to the left, and messages from the user “2” should be aligned to the right.

To visualize the messenger window, enclose the conversation into a frame with vertical bars and asterisk. The leftmost and rightmost sides of the frame should have vertical bars (the | character), and the topmost and bottommost lines of the frame should have asterisks – a string of \* symbols (with length of *width*) and 2 + symbols on the edges (aligned with the vertical bars |).

Here is an example below:

Given the input array:

Messages = [[“1”, “Hello how r u”], [“2”, “good ty”], [“2”, “u”], [“1”, “me too bro”]]

*Width* = 15, and *userWidth* = 5, the output should be

Solution(messages, width, userWidth) = [“+\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*+”, “|Hello |”, “|how r |”, “|u |”, “| good|”, “| ty|”, “| u|”, “|me |”, “too |”, “|bro |”, “+\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*+”]

Answer: Question 13 -> Need to fix solution still

1. Given an array of integers *target*, return an array *result* consisting of all elements *target[i]* that are greater than both of their neighbors, i.e. that are *target[i] > max(target[i-1], target[i+1])*. Note that *target[0]* and *target[target.length-1]*, which are the first and last elements, are included by default because they do not have neighbors.

For *target = [1,3,1,3,2]*, the output should be solution(target) = [1,3,3,2]

Answer: Question 14

1. Given an infinite number line, you would like to build few blocks and obstacles on it. Specifically, you have to implement code which supports two types operations:
   1. [1,x] – builds an obstacle at coordinate x along the number line. It is guaranteed that coordinate x does not contain any obstacles when the operation is performed.
   2. [2,x,size] – checks whether it’s possible to build a block of size *size* beginning at position x. For example, for size = 2 and x = 0, it will check 0 and 1 on the number line for obstacles. Returns 1 if possible, i.e. there are no obstacles at the occupied coordinates, and return 0 otherwise. Please note that this operation does not actually build the block, it only checks whether a block can be built.

Given an array of *operations* containing both types of operations above, your task is to return a binary string representing the outputs for all [2,x,size] operations.

For example:

Operations = [[1,2], [1,5], [2,3,2], [2,3,3], [2,1,1], [2,1,2]] the output should be *solution(operations) = “1010”*

Answer: Question 15

1. Write a program in Java that solves the following: Given an array of integers numbers and an array pattern representating a comparsion pattern, find how many subarrays of numbers match the given pattern. Pattern can only contain the following integers

*- pattern[i] = 1*, represents that the number of corresponding to this element of the pattern is greater than the previous one

*- pattern[i] = 0*, represents that the number corresponding to this element of the pattern is equal to the previous one

*- pattern[i] = -1*, represents that the number corresponding to this element of the pattern is less than the previous one.

It is guaranteed that the *numbers.length > pattern.length*

Example

1. For numbers = [4,1,3,4,4,5,5,1] and pattern = [1,0,-1], that the output should be solution(numbers, pattern) = 1
2. For numbers = [5,7,7,9,9,11,11] and pattern = [1,0,1,0], that the output should be solution(numbers, pattern) = 2
3. For numbers = [5,7,7,9,9,11,11] and pattern = [-1], that the output should be solution(numbers, pattern) = 0
4. For numbers = [3,3,3,3,3,3,5,5,5,4,3,4] and pattern = [0], that the output should be solution(numbers, pattern) = 7
5. For numbers = [1,2,3,4,5,6,7,8] and pattern = [1,1], that the output should be solution(numbers, pattern) = 6
6. For numbers = [3,4,5,6,5,4,7,8] and pattern = [-1,-1,-1], that the output should be solution(numbers, pattern) = 0

Answer: Question 16

1. Imagine you are monitoring changes to user ratings for an online platform. Each user on this platform has an overall rating (an integer between 1 and 2500) and a corresponding level. Rating levels are based on the following rules:
   1. Rating < 1000 = “beginner”
   2. 1000 <= rating < 1500 = “intermediate”
   3. 1500 <= rating < 2000 = “advanced”
   4. 2000 <= rating = “pro”

You are given an *initial* rating value and an array of integers *changes* representing changes to the rating. Your task is to calculate the final rating and return the level corresponding to that rating.

It is guaranteed that changes to the rating value will never result in it becoming less than 1 or greater than 2500.

Ex. For *initial*  *= 1500* and *changes = [-100, -300, 450, 500, -500, -600]*, the output should be *solution(initial, changes) = “beginner”*

Answer: Question 17

1. You are given an array of numbers, with each being a *0* or *1*. All *1*’s are arranged at the beginning of the array and *0*s stand at the end. A process is executed on this array until it halts. You are given a variable *zerosToOne*, which represents the number of 0s that can be deleted and exchanged for one 1.

Each second, one of the three events happen:

1. If there are at least *zerosToOne* zeros, then the last *zerosToOne* zeros are removed and one *1* is added at the beginning of the array.

2. If there is at least one *1*, the last *1* changes to *0*.

3. If neither option 1 nor option 2 can be completed, then the process halts.

Follow the process and compute how many seconds will pass until the process halts.

Ex. For numbers = [1,1,1,0,0,0] and zerosToOne = 2, the output should be solution(numbers, zerosToOne) = 13.

For numbers = [1,1] and zerosToOne = 2, the output should be solution(numbers, zerosToOne) = 4.

Answer: Question 18

1. You are given an array of strings *paths* containing paths to some files or directories. All file paths start with “/“.

Note: File paths may contain special characters “..” to represent parent directory specifiers. For example “/a/b../“ points to the same directory as “/a”. It is guaranteed that all given file paths are valid.

Your task is to find the longest common suffix path without file masks or special characters (e.g. “..”, etc.). If there isn’t any common suffix path, return an empty string as “”.

Example.

For paths = [

“/a/folder1/../folder1/a/leaf.txt”,

“/b/folder2/../folder1/a/leaf.txt”,

“/a/folder3/folder1/folder1/../a/leaf.txt”

]

the output should be solution(paths) = “/folder1/a/leaf.txt”.

For paths = [

"/root/folder1/b/../a",

"/root/folder1/a/leaf",

"/root/folder1/a/b/../../a/branch"

]

the output should be solution(paths) = “”.

For paths = [

"/a/b/../../bb/a/leaf.txt",

"/a/b/../sd/../a/ad/../leaf.txt"

]

the output should be solution(paths) = “/a/leaf.txt”.

Answer: Question 19

1. Suppose you are logged in to some virtual environment, where just two types of commands can be executed:

- goto <bucket\_name> - move to the specified bucket bucket\_name. You can assume the specified bucket always exists.

- create <filename> - create a new file named filename in the current bucket. If the file with the same name already exists in the current bucket, the command does nothing.

Your task is to process all provided commands and return the bucket name that contains the largest number of files at the end. It is guaranteed that there is no tie for the largest bucket name.

Notes:

- It is guaranteed that the first command is goto

- It is guaranteed that there is at least one create command

Example.

For commands = [

goto bucketA

create fileA

create fileB

create fileA

goto bucketB

goto bucketC

create fileA

create fileB

create fileC

]

the output should be solution(commands) = “bucket”

Answer: Question 20

1. You are monitoring the building density in a district of houses. The district is represented as a number line, where a house can be built at each numbered point on the line if at least one of the neighboring points is not occupied. Initially, there are no houses in the district.

You are given queries, an array of integers representing the locations of new houses in the order in which they will be built. After each house is built, your task is to find the longest segment of contiguous houses in the district.

Return an array of integers representing the longest segment of contiguous houses after each respect house from queries is built.

NOTE: It is guaranteed that all of the house locations in queries are unique and no house was built at a point with existing houses on both left and right adjacent points. Assume that array indices are 0-based.

Example:

queries = [2,1,3], the output should be solution(queries) = [1,2,3].

Answer: Question 21

1. You need to get to the mall to buy some new shoes, but you’re not sure how to get there. Your city is a n x m rectangular grid of blocks, where your home is located at the coordinates (x1, y1) and that mall’s location is (x2, y2). It is guaranteed that (x1, y1) and (x2, y2) are not the same.

Since you are not sure exactly how to get to the mall you follow a movement strategy based on these rules:

1. Move diagonally on each move, starting in the direction (+1, +1). It means that standing at a cell with coordinates (x,y) you’ll move to the cell with coordinates (x + 1, y + 1) unless the new cell is outside the city grid.

2. If the current move would take you outside the city grid, come back and reverse the direction that was leading outside the grid (e.g. if the x coordinate is outside the grid, reverse the x movement direction)

3. If the current move would escape the city grid outside of a corner, reverse both directions.

Your task is to determine how many steps it will take to reach the mall at (x2, y2). Return -1 if it’s not possible to reach the mall using this strategy.

Example. For n = 5, m = 5, x1 = 2, y1 = 1, x2 = 1, and y2 = 2, the output should be solution(n,m,x1,y1,x2,y2) = 7

Answer: Question 22

1. You are given a string of digits *panel* and an array of strings *codes*. Each string in the *codes* array consists of digits only and represents a code in the following format: *“<index><pattern>”,* where both index and pattern should consist of at least one digit. Since there are several ways to split the code, let’s consider them all in ascending order of index length and call them split-cases. For instance, for the code = “1324”, the split cases are:

case 1 - index = “1” and pattern = “324”

case 2 - index = “13” and pattern = “24”

case 3 - index = “132” and pattern = “4”.

For each code in *codes* and for every split-case of this code, check whether a string pattern is present at the index position in the *panel* string. Return a string array consisting of results of these checks, where each element is either pattern, if this pattern is present in *panel*, or otherwise “not found”.

Guaranteed constraints:

String panel : 1 <= panel.length <= 10^3

String array codes : 1<= codes.length <= 100 and 2 <= codes[i].length <= 10 and consists of ONLY digits between 0-9 inclusive

Answer: Question 23

1. You are given an array of integers *arr*. Your task is to determine whether each sequence of three elements in the array (*arr[i], arr[i+1], arr[i+2])* are monotonic. Three consecutive elements are monotonic if their values are in strictly increasing or strictly decreasing order.

Return an array of integers of length *arr.length – 2*, where the *ith* element is equal to *1*. If *arr[i] < arr[i+1] < arr[i+2]* or *arr[i] > arr[i+1] > arr[i+2]*, and *0* otherwise.

Example:

* For *arr = [1,2,1,-4,5,10]* the output should be *solution(arr) = [0,1,0,1]*

Answer: Question 24

1. You are given an array of positive integers *numbers* and a positive integer *x*. Your task is to find the number of ways that *x* can be obtained by combining pairs of integers in the array *numbers* together. In other words, find the number of pairs *(I,j)* such that *x* is the concatenation of *numbers[i] + numbers[j]*

Answer: Question 25

1. You are developing a simple version control system. Your system can support branches, and you can switch between branches or update the files in a branch using the following actions:
   1. *Switch < branch name>* - switches to the branch with given name
   2. *Push <file name>* - pushes a file to the current branch. Note that if the file was pushed to the current branch before, then the number of files should not change.

Your task is to process all the logs and return the branch name that contains the largest number of files at the end. It is guaranteed that there is no tie.

Answer: Question 20 (It’s a duplicate of 20, just worded differently but same test case and example)

1. Given two array of numbers, *firstArray* and *secondArray*. Return the length of the longest common prefix (LCP) between any pair of numbers from different arrays or 0 if no common prefix exists.

Note: A prefix of number is a number formed by one or more of its digits, starting from its highest-order digit. For example, 123 is a prefix of the number 12345 and 2 is a prefix of the number 234. A common prefix of two numbers if a number, which is a prefix of both. For instance, the longest common prefix (LCP) of 5655359 and 56554 is 5655 ad there is no common prefix of 123 and 456.

Ex. Given the following arrays:

First = [25,288,2655,54546,54,555] and

Second = [2,255,266,244,26,5,54547]

the output should be solution(firstArray, secondArray) = 4

Answer: Question 27

1. You are given a binary array *state* consisting of integers *0* and *1*. You are also given *operations* – an array of strings, each representing an operation of one of two types:
   1. *L* – find the smallest index *i,* for which *state[i] = 0*, and set *state[i] = 1*. If there is no such index, do nothing.
   2. *C[index]* – set *state[index] = 0*
      1. This operation does not depend on the previous value of *state[index]*.
      2. It is guaranteed that *index* is a valid 0-based index of *state* (i.e *index < state.length*)

Given *state* and *operations*, your task is to return the binary string of *state* array after all the operations have been applied.

Ex. For *state = [0,0,0,0,0,0,0,0,0,0]* and *operations = [“L”, “L”, “C0”, “L”, “C3”]*, the output should be *solution(state, operations) = “1100000000”*.

Answer: Question 28

Poppa, pompom,

1. You are given a string *word* consisting of lowercase English letters, and a list of strings *skeletons* consisting of *‘-‘* characters and lowercase English letters. Every *skeleton* will always be the same length as *word.*

Your task is to return a list of skeletons that can form the given *word*. A skeleton can form a word if all ***–*** characters can be replaced with other characters taken from the same skeleton to make the string equal to the *word*. If no strings within *skeletons* can form the given *word* by doing this, return an empty list. The matching skeletons should be returned in the same order they appear in *skeletons* and the list of skeletons may not all be unique.

*Example:* For word = “hello” and skeletons = [“he-lo”, he--o”, “-ell-“, “hello”], the output should be solution(word, skeletons) = [“he-lo”, “hello”]

Answer: Question 29

1. Image that there are several lamps placed on a number line, each of which illuminates some segment of the line. Specifically, the lamps are represented in a two-dimensional array *lamps*, where the *ith* lamp covers the segment from *lamps[i][0]* to *lamps[i][1]*, inclusive.

Additionally, you are given a list of control points on this number line, represented by an array *points*. Your task is to find the number of lamps that illuminate each control point. Specifically, for each control point *points[j]* in the array, your task is to find the number of lamps *lamps[i]* which include this point within it’s covered segment – when *points[j]* lies inside the segment *lamps[i][0], lamps[i][1]*.

As a result, return the array of integers, where *ith* integer corresponds to the answer for the *ith* control point.

Example:

*Lamps = [[1,7], [5,11], [7,9]]* and *points = [7,1,5,10,9,15].* The output should be solution(lamps, points) = [3,1,2,1,2,0].

Answer : Question 30

1. Let’s imagine objects located on the canvas at a certain moment in time. You are given an array of integer pairs *centers* representing the coordinates of those objects. Each object has a collision box – a square area around it’s center with a side equal to 2. Two objects are supposed to collide if their collision boxes have at least one common point. Calculate the number of object pairs that collide.

Note: Object collision boxes intersect if the distance **in each coordinate** between the object centers does not exceed 2.

If x1, y1 are the coordinates of one object and x2, y2 are coordinates of the second object, then the collision condition for them can be written in the form |x[j] – x[i]| <= 2 and |y[j] – y[i] | <= 2.

Example: For centers = [[1,1], [2,2], [0,4]] the output should be solution(centers) = 2.

Answer: Question31

1. Given an array of strings *schedule* that represents the schedule of bus arrival times and a string *time* that represents the current time, find out how long you have to wait for the next bus. If there are no more buses scheduled for the day, return *“-1”.*

Time is represented as a string in the form of *HH:MM* (in the 24-hour format). Bus arrival times are sorted in chronological order.

Please assume that the bus leaves at the same time that it arrives. So, if a bus was scheduled to come before the current time, it has already left.

Example. For *schedule = [“12:30”, “14:00”, “19:55”]* and *time = “14:30”*, the output should be *solution(schedule, time) = “05:25”.*

Answer: Question 32