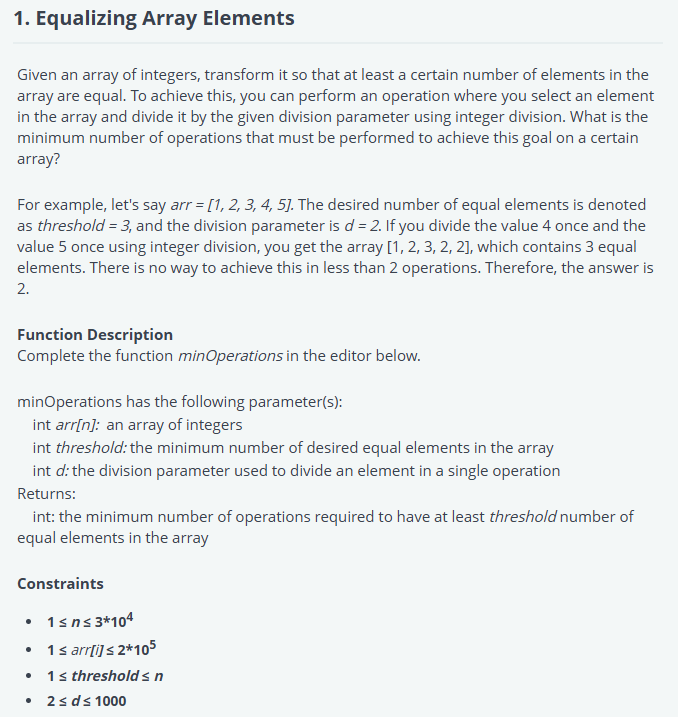
# Equalizing Array Elements



import java.io.\*;

import java.util.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.toList;

class Result {

    /\*

     \* Complete the 'minOperations' function below.

     \*

     \* The function is expected to return an INTEGER.

     \* The function accepts the following parameters:

     \*  1. INTEGER\_ARRAY arr

     \*  2. INTEGER threshold

     \*  3. INTEGER d

     \*/

    public static int minOperations(List<Integer> arr, int threshold, int d) {

        // Map to store the number of operations needed to make each value

        Map<Integer, List<Integer>> operationCounts = new HashMap<>();

        // Process each number in the array

        for (int num : arr) {

            int currentNum = num;

            int operations = 0;

            // Track how many operations are required to reduce the number

            while (currentNum > 0) {

                operationCounts.putIfAbsent(currentNum, new ArrayList<>());

                operationCounts.get(currentNum).add(operations);

                // Divide by d for the next operation

                currentNum /= d;

                operations++;

            }

            // Don't forget to add the final zero state

            operationCounts.putIfAbsent(0, new ArrayList<>());

            operationCounts.get(0).add(operations);

        }

        // Find the minimum number of operations for at least 'threshold' equal elements

        int minOperations = Integer.MAX\_VALUE;

        // Go through each value in the map and check if it can satisfy the threshold condition

        for (Map.Entry<Integer, List<Integer>> entry : operationCounts.entrySet()) {

            List<Integer> operationsList = entry.getValue();

            if (operationsList.size() >= threshold) {

                // Sort the operation counts to find the minimum operations to get `threshold` equal elements

                Collections.sort(operationsList);

                int totalOps = 0;

                // Add the minimum operations needed for the threshold number of elements

                for (int i = 0; i < threshold; i++) {

                    totalOps += operationsList.get(i);

                }

                // Update the minimum operations

                minOperations = Math.min(minOperations, totalOps);

            }

        }

        return minOperations;

    }

}

public class Solution {

    public static void main(String[] args) throws IOException {

        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

        BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

        int arrCount = Integer.parseInt(bufferedReader.readLine().trim());

        List<Integer> arr = IntStream.range(0, arrCount).mapToObj(i -> {

            try {

                return bufferedReader.readLine().replaceAll("\\s+$", "");

            } catch (IOException ex) {

                throw new RuntimeException(ex);

            }

        })

            .map(String::trim)

            .map(Integer::parseInt)

            .collect(toList());

        int threshold = Integer.parseInt(bufferedReader.readLine().trim());

        int d = Integer.parseInt(bufferedReader.readLine().trim());

        int result = Result.minOperations(arr, threshold, d);

        bufferedWriter.write(String.valueOf(result));

        bufferedWriter.newLine();

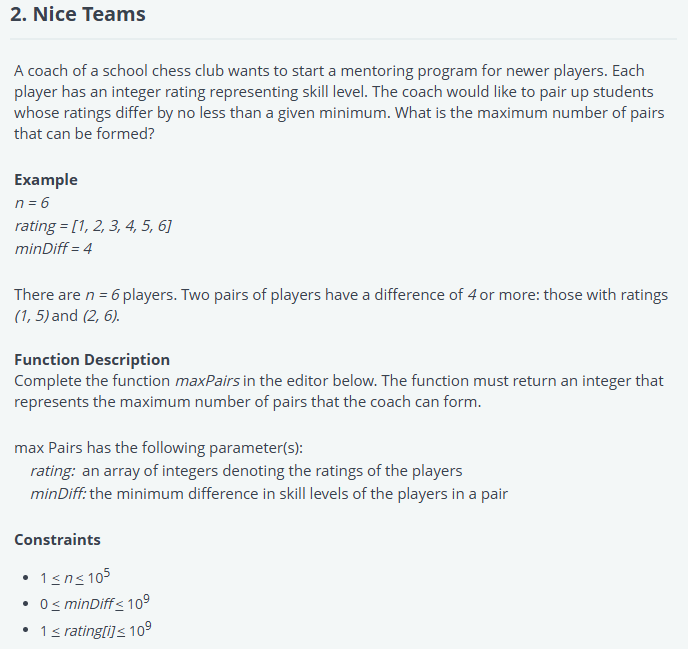
        bufferedReader.close();

        bufferedWriter.close();

    }

}

# Nice Teams



import java.io.\*;

import java.util.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.toList;

class Result {

    /\*

     \* Complete the 'maxPairs' function below.

     \*

     \* The function is expected to return an INTEGER.

     \* The function accepts the following parameters:

     \*  1. INTEGER\_ARRAY skillLevel

     \*  2. INTEGER minDiff

     \*/

    public static int maxPairs(List<Integer> skillLevel, int minDiff) {

        // Sort the skill levels in ascending order

        Collections.sort(skillLevel);

        int pairs = 0;

        int i = 0;

        int j = 1;

        // Loop through and form valid pairs

        while (j < skillLevel.size()) {

            // Check if the difference between skillLevel[i] and skillLevel[j] is >= minDiff

            if (skillLevel.get(j) - skillLevel.get(i) >= minDiff) {

                // If valid, form a pair and increment pair count

                pairs++;

                // Move both pointers forward as we formed a valid pair

                i++;

                j++;

            } else {

                // If not valid, just move the second pointer forward

                j++;

            }

        }

        return pairs;

    }

}

public class Solution {

    public static void main(String[] args) throws IOException {

        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

        BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

        int skillLevelCount = Integer.parseInt(bufferedReader.readLine().trim());

        List<Integer> skillLevel = IntStream.range(0, skillLevelCount).mapToObj(i -> {

            try {

                return bufferedReader.readLine().replaceAll("\\s+$", "");

            } catch (IOException ex) {

                throw new RuntimeException(ex);

            }

        })

            .map(String::trim)

            .map(Integer::parseInt)

            .collect(toList());

        int minDiff = Integer.parseInt(bufferedReader.readLine().trim());

        int result = Result.maxPairs(skillLevel, minDiff);

        bufferedWriter.write(String.valueOf(result));

        bufferedWriter.newLine();

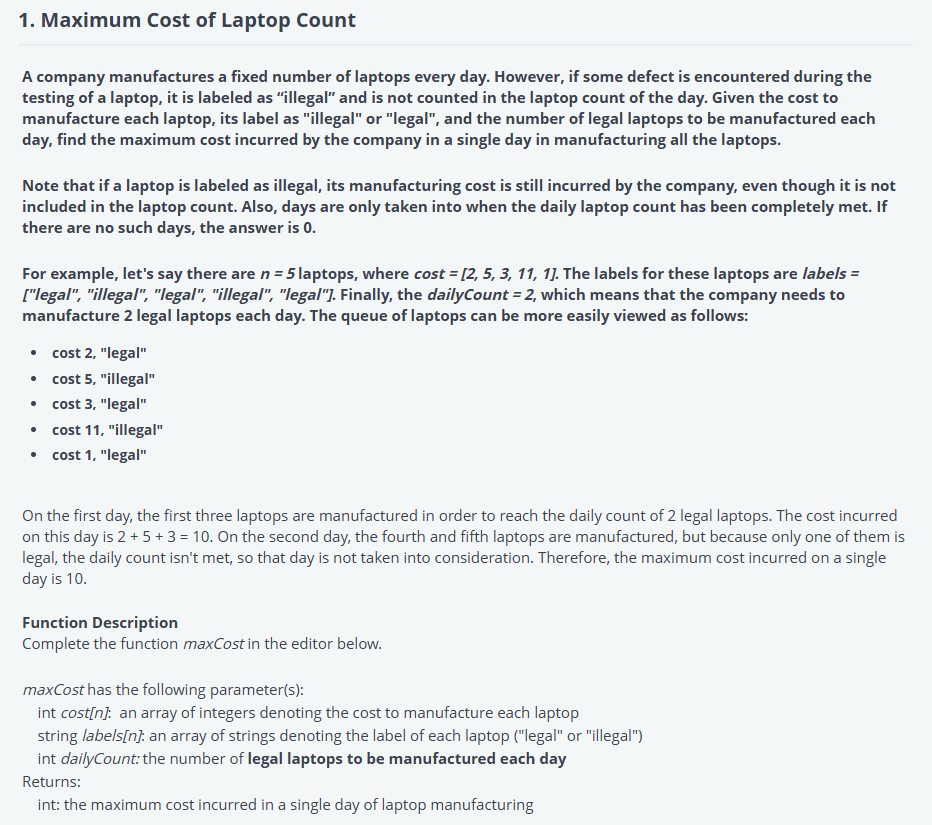
        bufferedReader.close();

        bufferedWriter.close();

    }

}

# Maximum Cost of Laptop Count



#!/bin/python3

import math

import os

import random

import re

import sys

#

# Complete the 'maxCost' function below.

#

# The function is expected to return an INTEGER.

# The function accepts following parameters:

#  1. INTEGER\_ARRAY cost

#  2. STRING\_ARRAY labels

#  3. INTEGER dailyCount

#

def maxCost(cost, labels, dailyCount):

    # Write your code here

    ans = 0

    cur\_cnt = 0

    cur\_cost = 0

    for c, l in zip(cost, labels):

        cur\_cost += c

        if l == "illegal":

            continue

        cur\_cnt += 1

        if cur\_cnt == dailyCount:

            ans = max(ans, cur\_cost)

            cur\_cnt = 0

            cur\_cost = 0

    return ans

if \_\_name\_\_ == '\_\_main\_\_':

    fptr = open(os.environ['OUTPUT\_PATH'], 'w')

    cost\_count = int(input().strip())

    cost = []

    for \_ in range(cost\_count):

        cost\_item = int(input().strip())

        cost.append(cost\_item)

    labels\_count = int(input().strip())

    labels = []

    for \_ in range(labels\_count):

        labels\_item = input()

        labels.append(labels\_item)

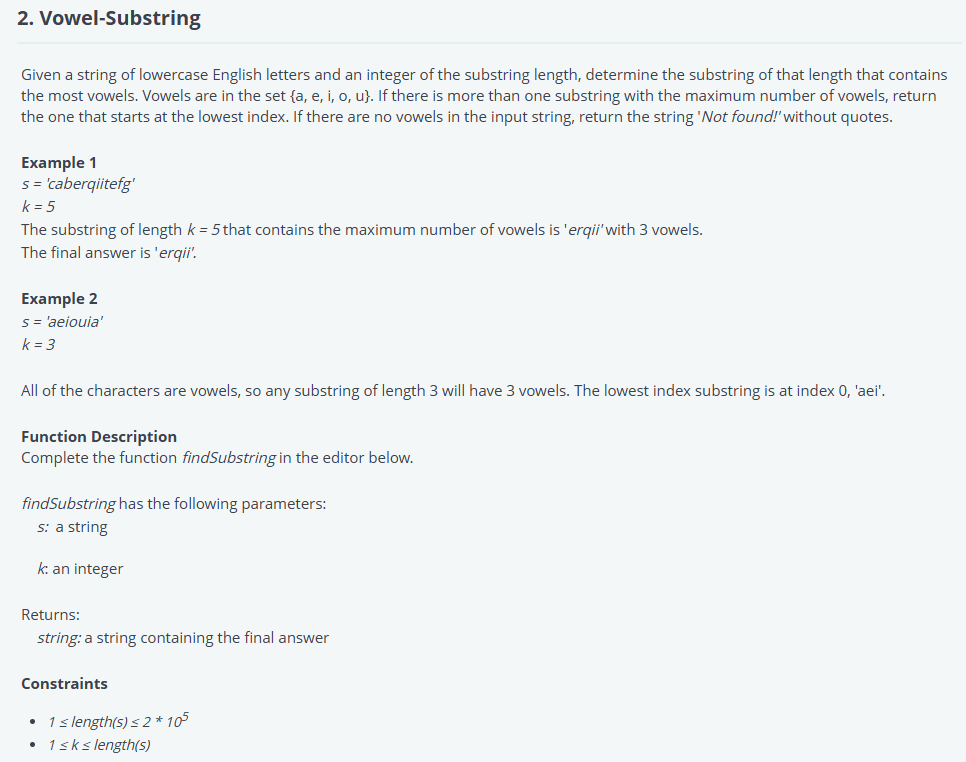
    dailyCount = int(input().strip())

    result = maxCost(cost, labels, dailyCount)

    fptr.write(str(result) + '\n')

    fptr.close()

# Vowel-Substring



#!/bin/python3

import math

import os

import random

import re

import sys

#

# Complete the 'findSubstring' function below.

#

# The function is expected to return a STRING.

# The function accepts following parameters:

#  1. STRING s

#  2. INTEGER k

#

def findSubstring(s, k):

    # Write your code here

    vowels = ["a", "e", "i", "o", "u"]

    cur = best = sum([c in vowels for c in s[:k]])

    ans = 0

    for i in range(k, len(s)):

        cur += s[i] in vowels

        cur -= s[i - k] in vowels

        if cur > best:

            best = cur

            ans = i - k + 1

    if best > 0:

        return s[ans:(ans+k)]

    else:

        return "Not found!"

if \_\_name\_\_ == '\_\_main\_\_':

    fptr = open(os.environ['OUTPUT\_PATH'], 'w')

    s = input()

    k = int(input().strip())

    result = findSubstring(s, k)

    fptr.write(result + '\n')

    fptr.close()