1. Twitter new office design

☆ Twitter New Office Design

Twitter is constructing a new office and would like to decorate the interior with popular and cool hashtags. They want to place these hashtags between employee tables of different heights positioned on a line with various amounts of gaps. A unit of gap between tables is referred to as a holder, and each holder can hold one hashtag. Height of a hashtag is simply the number of characters in it. The height of a hashtag placed in a holder cannot exceed *1* unit above the height of an adjacent table or a hashtag. Given the placement of a number of tables and their heights, determine the maximum height hashtag that can be used. If no hashtag can be used, return *0*.

For example, there are *n = 4* tables at *tablePositions = [1, 2, 4, 7]* with *tableHeights = [4, 5, 7, 11]*. There is no space between the first two tables, so there is no holder for the hashtag. Between positions *2* and *4*, there is one unit of gap, so one holder. Heights of the surrounding tables are *5* and *7*, so the maximum possible height of hashtag to place in the holder is *5 + 1 = 6*. Between positions *4* and *7* there are two holders. The heights of surrounding tables are *7* and *11*. The maximum possible height for a hashtag placed next to the table of height *7* is *8*. The maximum height hashtag between a hashtag of height *8* and a table of height *11* is *9*. Hashtag heights are *6, 8* and *9*, and the maximum height is *9*. In the table below, digits are in the columns of tables and *M* is in the Hashtags.

```
      7
      7
     M7
    MM7
   4MM7
  M4MM7
 2M4MM7
12M4MM7
12M4MM7
12M4MM7
12M4MM7
```

Function Description

Complete the function *maxHeight* in the editor below. The function must return an integer, the maximum height hashtag that could be used in the building.

maxHeight has the following parameter(s):

- table*Positions[tablePositions[0],...tablePositions[n-1]]*: an array of integers
- table*Heights[tableHeights[0],...tableHeights[n-1]]*: an array of integers

▼ Sample Case 0

Sample Input For Custom Testing

```
3
1
3
7
3
4
3
3
```

Sample Output

```
5
```

Explanation

```
   M
1M MMM
1M3MMM7
1M3MMM7
1M3MMM7
```

Here table*Positions = [1, 3, 7]* and table*Heights = [4, 3, 3]*. There can be a hashtag of height *4* at position *2* supported by tables of heights *4* and *3*. Between positions *3* and *7*, there can be a hashtag of height *4* at positions *4* and *6*. Between them, a hashtag of height *5* can be used at position *5*.

▼ Sample Case 1

Sample Input For Custom Testing

```
2
1
10
2
1
5
```

Sample Output

```
7
```

Explanation

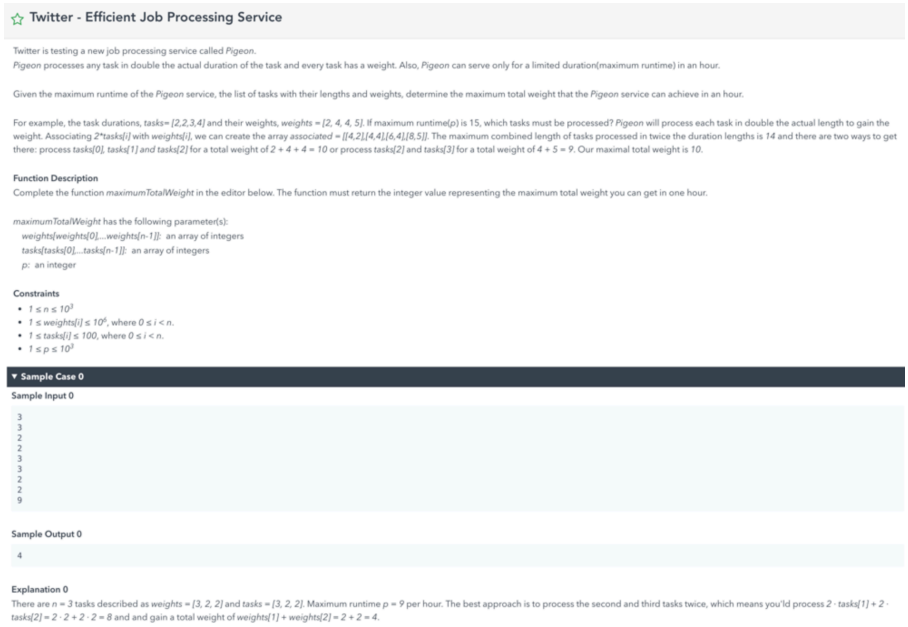
```
     MM
    MMMM
   MMMMM10
  MMMMMM10
 MMMMMMM10
MMMMMMMM10
1MMMMMMM10
```

Here *tablePositions = [1, 10]* and *tableHeights = [1, 5]*. The maximum heights of the hashtags from positions *2* through *9* are *[2, 3, 4, 5, 6, 7, 7, 6]*.

```
1 int calculateHeight( int dist, int height1, int height2 )
2 {
3     int  minH = min( height1, height2);
4     int  maxH = max( height1, height2 );
```

```
5
6       if( dist == 0 ) return 0;
7       if( dist == 1 ) return minH + 1;
8
9       // if both heights are same then meet in middle
10       if(minH == maxH )
11       {
12           int add = ( dist%2 == 0 ) ? dist/2 : dist/2+1;
13           return minH + add;
14       }
15
16       // See if we can make the height same
17       int delta = maxH-minH;
18       if( delta < dist )
19       {
20           dist -= delta;
21           minH += delta;
22           int add = ( dist%2 == 0 ) ? dist/2 : dist/2+1;
23           return minH+add;
24       }
25
26       // for all cases where delta >= dist
27       return minH+dist;
28   }
29
30   int getMaxHeight( const vector<int>& positions, const vector<int>& heights )
31   {
32       if( positions.empty() || heights.empty() || positions.size() != heights.size() )
33       {
34           return 0;
35       }
36
37       int result = 0;
38       for(int i=1; i<positions.size(); ++i )
39       {
40           int currMax = calculateHeight( positions[i]-positions[i-1]-1, heights[i], heights[i-1]
41           result = max( result, currMax);
42       }
43       return result;
44   }
45
46   int main() {
47       cout << getMaxHeight({1,10}, {1,5}) << endl;
48       cout << getMaxHeight({1,3,7},{4,3,3}) << endl;
49        cout << getMaxHeight({1,2,4,7},{4,5,7,11}) << endl;
50   }
```

2. Efficient Job Processing



☆ **Twitter - Efficient Job Processing Service**

Twitter is testing a new job processing service called *Pigeon*.
Pigeon processes any task in double the actual duration of the task and every task has a weight. Also, *Pigeon* can serve only for a limited duration(maximum runtime) in an hour.

Given the maximum runtime of the *Pigeon* service, the list of tasks with their lengths and weights, determine the maximum total weight that the *Pigeon* service can achieve in an hour.

For example, the task durations, *tasks= [2,2,3,4]* and their weights, *weights = [2, 4, 4, 5]*. If maximum runtime(*p*) is 15, which tasks must be processed? *Pigeon* will process each task in double the actual length to gain the weight. Associating *2*tasks[i]* with *weights[i]*, we can create the array *associated = [[4,2],[4,4],[6,4],[8,5]]*. The maximum combined length of tasks processed in twice the duration lengths is *14* and there are two ways to get there: process *tasks[0], tasks[1] and tasks[2]* for a total weight of *2 + 4 + 4 = 10* or process *tasks[2]* and *tasks[3]* for a total weight of *4 + 5 = 9*. Our maximal total weight is *10*.

Function Description
Complete the function *maximumTotalWeight* in the editor below. The function must return the integer value representing the maximum total weight you can get in one hour.

maximumTotalWeight has the following parameter(s):
- *weights[weights[0],...weights[n-1]]*: an array of integers
- *tasks[tasks[0],...tasks[n-1]]*: an array of integers
- *p*: an integer

Constraints
- $1 \le n \le 10^3$
- $1 \le weights[i] \le 10^6$, where $0 \le i < n$.
- $1 \le tasks[i] \le 100$, where $0 \le i < n$.
- $1 \le p \le 10^3$

▾ **Sample Case 0**

Sample Input 0

```
3
3
2
2
3
3
2
2
9
```

Sample Output 0

```
4
```

Explanation 0
There are *n = 3* tasks described as *weights = [3, 2, 2]* and *tasks = [3, 2, 2]*. Maximum runtime *p = 9* per hour. The best approach is to process the second and third tasks twice, which means you'll process *2 · tasks[1] + 2 · tasks[2] = 2 · 2 + 2 · 2 = 8* and and gain a total weight of *weights[1] + weights[2] = 2 + 2 = 4*.

```java
public class EfficientJobProcessingService_14 {
        int[][] dp = new int[tasks.length + 1][p / 2 + 1];      // task will be processed doubl
//        for (int i = 0; i < tasks.length; i++) {
//            tasks[i] *= 2;
//        }

        for (int i = 1; i < dp.length; i++) {
            for (int j = 1; j < dp[0].length; j++) {
                if (j < tasks[i - 1]) {
                    dp[i][j] = dp[i - 1][j];
                } else {
                    dp[i][j] = Math.max(dp[i - 1][j], dp[i - 1][j - tasks[i - 1]] + weights[i
                }
            }
        }

        return dp[tasks.length][p / 2];
    }

    public static void main(String[] args) {
        int[] weights = new int[]{2, 4, 4, 5};
        int[] tasks = new int[]{2, 2, 3, 4};
        System.out.println(new EfficientJobProcessingService_14().maximumTotalWeight(weights,

        weights = new int[]{2, 2, 4, 5};
```

```
        tasks = new int[]{2, 2, 3, 4};
        System.out.println(new EfficientJobProcessingService_14().maximumTotalWeight(weights,

        weights = new int[]{2};
        tasks = new int[]{2};
        System.out.println(new EfficientJobProcessingService_14().maximumTotalWeight(weights,

        weights = new int[]{2};
        tasks = new int[]{2};
        System.out.println(new EfficientJobProcessingService_14().maximumTotalWeight(weights,
    }
}
```

3. Game Events

☆ Game Events

Two teams were playing a football (soccer) match and the record of events for each team is available. There are 4 possible events, Goal (G), Yellow Card (Y), Red Card (R) and Substitution (S). The first three events (G, Y, R) are represented as `player-name time event-name` whereas the last event (S) is represented as `player-name time event-name second-player-name`. The time is represented in minutes from the start of the game. A football game is divided into two halves of 45 minutes each, and sometimes a little extra time is given at the end of each half, which is represented in the time as `time+extra-time`. So, there can be two types of times, for example, `32` and `45+2`. The extra time is always considered to have occurred before the events of the next half, so, `45+2` happened earlier than `46`.

Merge the events for each team into a single game event with teams name in front and sorted by time and event in order of G, Y, R, S. In the case of the same event happening at the same time, output should be sorted lexicographically based on teams name and players name.

Take for example, the first team *team1 = "EDC"* with events recorded as *events1 = ['Name1 12 G','FirstName LastName 43 Y']* and second team *team2 = "CDE"* with events recorded as *events2 = ['Name3 45+1 S SubName','Name4 46 G']*, then the chronological order of events is given by:

EDC Name1 12 G
EDC FirstName LastName 43 Y
CDE Name3 45+1 S SubName
CDE Name4 46 G

Function Description
Complete the function *getEventsOrder* in the editor below. The function must return an array of strings.

getEventsOrder has the following parameter(s):
- *team1:* a string, the name of the first team
- *team2:* a string, the name of the second team
- *events1[events1[0],...events1[n-1]]:* an array of strings that describe the events of the first team
- *events2[events2[0],...events2[n-1]]:* an array of strings that describe the events of the second team

```
import re


def getEventsOrder(team1, team2, events1, events2):
    # Write your code here
    football = list()
    football.append({"team": team1, "event": events1})
    football.append({"team": team2, "event": events2})

     game_details_list = list()
     original_event = list()
     event_priority = ['G', 'Y', 'R', 'S']

     for f in football:
         for event in f["event"]:
```

```
            original_event.append(f["team"]+" "+event)

            # split events string to get details
            pattern = re.compile("([a-zA-Z\s]*)(\d+)[+]?(\d*).([G,Y,R,S])([a-zA-Z\s]*)")
            split_event = pattern.search(event)

            # create a list of format ["team name", "player name", "time", "extra time", "ever
            record = list()
            record.append(f["team"])  # team name
            if split_event:
                record.append(split_event.group(1).strip())  # player name
                record.append(int(split_event.group(2).strip()))  # time
                record.append(int(split_event.group(3).strip()) if len(split_event.group(3).st
                record.append(event_priority.index(split_event.group(4).strip()))  # event
                record.append(split_event.group(5).strip())  # second player
            game_details_list.append(record)

    # sorting the list to return index position of the sorted list
    new_num_index_sorted = (sorted(range(len(game_details_list)),
                                   key=lambda k: (
                                       game_details_list[k][2],  # time
                                       game_details_list[k][3],  # extra time
                                       game_details_list[k][4],  # event
                                       game_details_list[k][0],  # team name
                                       game_details_list[k][1],  # player name
                                       game_details_list[k][5])))

    # based on the index position, fetching result from original event list and appending in a
    answer = list()
    for i in new_num_index_sorted:
        answer.append(original_event[i])
    return answer
```

4. Unique Twitter User Id Set

☆ Unique Twitter User Id Set

Twitter needs user ids to be unique and while assigning the user ids to its customers, Twitter wants to have the sum of the user ids to be minimum.
Given an array of random user ids for n users, increment any duplicate elements until all the user ids are unique and ensure that their sum is minimum.
Example, if *arr = [3, 2, 1, 2, 7]*, then arr_{unique} *= [3, 2, 1, 4, 7]* and its user ids sum to a minimal value of *3 + 2 + 1 + 4 + 7 = 17*.

Function Description

Complete the *getUniqueUserIdSum* in the editor below to create an array of unique user ids with a minimal sum. Return the user id sum of the resulting unique user ids.

getUniqueUserIdSum has the following parameter(s):

arr: an array of integers to process

Constraints

- $1 \le n \le 2000$
- $1 \le arr[i] \le 3000$ where $0 \le i < n$

online assessment twitter

```python
from collections import Counter


def getUniqueUserIdSum(arr):
    summation = 0
    count = Counter(arr)

    position = dict(zip(arr, arr))

    for i in arr:
        if count[i] > 1:
            temp = position[i] + 1
            while temp in count:
                temp += 1
            summation += temp
            count[temp] = 1
            count[i] -= 1

            position[i] = temp
        else:
            summation += i
    return summation
```

5. Partitioning Array

☆ **Partitioning array**

Given an array of numbers, you are required to check if it is possible to partition the array into some subsequences of length *k* each, such that:

- Each element in the array occurs in exactly one subsequence
- All the numbers in a subsequence are distinct
- Elements in the array having the same value must be in different subsequences

Is possible to partition the array satisfying the above conditions? If it is possible, return *"Yes"*, else return *"No"*.

For example, suppose:

- There are *n* = 4 numbers in the array
- The length of each subsequence needs to be *k* = 2.
- The given array is *{1, 2, 3, 4}*.
- Then one possible way is to choose the first *2* elements of the array *{1, 2}* as the first subsequence, the next *2* elements *{3, 4}* as the next subsequence. So the answer is *Yes*.

Consider another example:

- There are *n* = 4 numbers in the array,
- The length of each subsequence needs to be *k* = 3.
- The given array is *{1, 2, 2, 3}*.
- Here there is no way to partition the array into subsequences such that all subsequences are of length *3* and each element in the array occurs in exactly one subsequence. Hence the answer is *No*.

Function Description

Complete the function *solve* in the editor below. The function has to return one string denoting the answer.

solve has the following parameters:
k: an integer
numbers[0,...,n-1]: an array of integers

online assessment new grad twitter

```java
import java.util.*;

public class Main {
    public static String partArr(int k,int[] num){
        if(num.length == 0 || k==0){
            return "YES";
        }
```

```
        if(num.length%k != 0){
            return "NO";
        }
        HashMap<Integer,Integer> map = new HashMap<>();
        for(int i=0;i<num.length;i++){
            if(!map.containsKey(num[i])){
                map.put(num[i],1);
            }
            else{
                map.put(num[i],map.get(num[i])+1);
            }
        }
        for (Map.Entry<Integer, Integer> entry : map.entrySet()) {
            if(entry.getValue()> num.length/k){
                return "NO";
            }
        }
        return "YES";

    }
    public static void main(String[] args) {
        int[] num1 = new int[]{1,2,3,4};
        System.out.println(partArr(2,num1)); //YES
        int[] num2 = new int[]{1,2,3,3};
        System.out.println(partArr(2,num2));  //YES
        System.out.println(partArr(3,num1)); //NO
        int[] num3 = new int[]{3,3,3,6,6,6,9,9,9};
        System.out.println(partArr(3,num3)); //YES
        int[] num4 = new int[]{};
        System.out.println(partArr(1,num4));//YES
        int[] num5 = new int[]{1};
        System.out.println(partArr(1,num5));//YES
        int[] num6 = new int[]{1,2};
        System.out.println(partArr(2,num6));//YES
    }
}
```

6. Autoscale Policy

☆ Autoscale Policy

A risk modeling system uses a scaling computing system that implements an autoscale policy depending on the current load or *utilization* of the computing system.

The system starts with a number of computing instances given by *instances*. The system polls the instances every second to see the average utilization at that second, and performs scaling as given below. Once any action is taken, the system will stop polling for 10 seconds. During that time, the number of instances does not change.

Average utilization > 60%: Double the number of instances if the doubled value does not exceed $2 * 10^8$. This is an action. If the number of instances exceeds this limit on doubling, perform no action.

Average utilization < 25%: Halve the number of instances if the number of instances is greater than *1* (take ceil if the number is not an integer). This is also an action. If the number of instances is *1*, take no action.

25% ≤ Average utilization ≤ 60%: No action

Given the values of the average utilization at each second for this system as an array determine the number of instances at the end of the time frame.

For example, the system starts with *instances = 2*. Average utilization is given as *averageUtil = [25, 23, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 76, 80].* At the first second, utilization is *25*, so no action is taken. At the second second, *averageUtil[1] = 23 < 25*, so *instances = 2 / 2 = 1*. The next *10* seconds, *averageUtil[2]..averageUtil[11]*, no polling is done. At *averageUtil[12] = 76, 76 > 60* so the number of instances is doubled. There are no more readings to consider and *2* is the final answer.

Function Description

Complete the function *finalInstances* in the editor below. The function must return an integer that represents the final number of instances running.

finalInstances has the following parameter(s):

instances: an integer that represents the original number of instances running

averageUtil[averageUtil[0],...averageUtil[n-1]]: an array of integers that represents the average utilization at each second of the time frame

Constraints

- $1 \le instances < 10^5$
- $1 \le n < 10^5$
- $0 \le averageUtil[i] \le 100$

▼ Sample Case 0

Sample Input For Custom Testing

```
1
3
5
10
80
```

Sample Output

```
2
```

Explanation

Here *instances = 1* and *averageUtil = [5, 10, 80].* At the 1^{st} and 2^{nd} seconds of the time period, no action will be taken. Even though the utilization is less than *25%*, the number of instances is *1*. During the 3^{rd} second, the number of instances will be doubled to *2*.

```java
import java.util.*;

public class Main {
    public static int autoPolicy(int[] num,int ins) {
        if (num.length == 0) {
            return ins;
        }
        int i=0;

         while(i<num.length) {
              if(num[i]<=60 && num[i]>=25){
                i ++;
              }
              else if(num[i]>60){
                  if(ins<Math.pow(10,8)){
                      ins = ins*2;
                      i = i+10;
                  }
                  else{
```

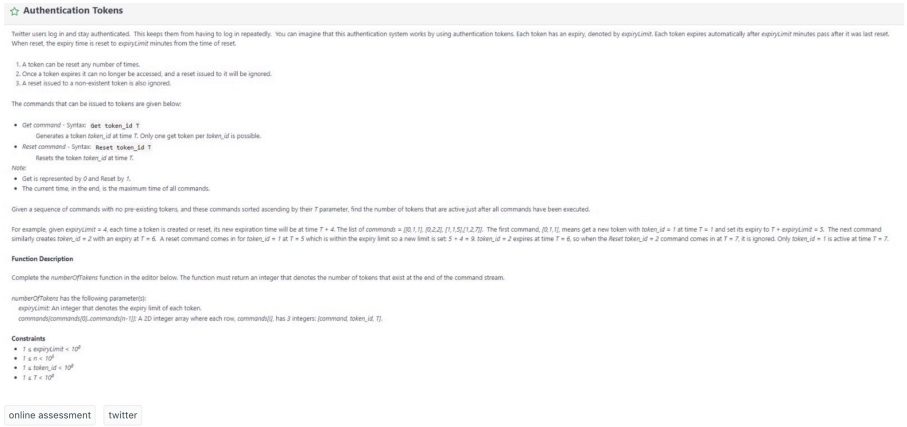
```
                        i++;
                    }

                }
                else{
                    if(ins>1){
                        ins = ins/2;
                        i=i+10;
                    }
                    else{
                        i++;
                    }

                }
            }
            return ins;
        }
        public static void main(String[] args) {
            int tem = 1;
//          int[] funcArr = new int[]{25,23,1,2,3,4,5,6,7,8,9,10,76,80};
            int[] funcArr = new int[]{5,10,80};
            System.out.println(autoPolicy(funcArr,tem));

        }
}
```

7. Authentication Token



☆ **Authentication Tokens**

Twitter users log in and stay authenticated. This keeps them from having to log in repeatedly. You can imagine that this authentication system works by using authentication tokens. Each token has an expiry, denoted by *expiryLimit*. Each token expires automatically after *expiryLimit* minutes pass after it was last reset. When reset, the expiry time is reset to *expiryLimit* minutes from the time of reset.

1. A token can be reset any number of times.
2. Once a token expires it can no longer be accessed, and a reset issued to it will be ignored.
3. A reset issued to a non-existent token is also ignored.

The commands that can be issued to tokens are given below:

- *Get command* - Syntax: `Get token_id T`
  Generates a token *token_id* at time *T*. Only one get token per *token_id* is possible.
- *Reset command* - Syntax: `Reset token_id T`
  Resets the token *token_id* at time *T*.

Note:
- Get is represented by *0* and Reset by *1*.
- The current time, in the end, is the maximum time of all commands.

Given a sequence of commands with no pre-existing tokens, and these commands sorted ascending by their *T* parameter, find the number of tokens that are active just after all commands have been executed.

For example, given *expiryLimit = 4*, each time a token is created or reset, its new expiration time will be at time *T + 4*. The list of *commands = [[0,1,1], [0,2,2], [1,1,5],[1,2,7]]*. The first command, *[0,1,1]*, means get a new token with *token_id = 1* at time *T = 1* and set its expiry to *T + expiryLimit = 5*. The next command similarly creates *token_id = 2* with an expiry at *T = 6*. A reset command comes in for *token_id = 1* at *T = 5* which is within the expiry limit so a new limit is set: *5 + 4 = 9*. *token_id = 2* expires at time *T = 6*, so when the *Reset token_id = 2* command comes in at *T = 7*, it is ignored. Only *token_id = 1* is active at time *T = 7*.

Function Description

Complete the *numberOfTokens* function in the editor below. The function must return an integer that denotes the number of tokens that exist at the end of the command stream.

numberOfTokens has the following parameter(s):
- *expiryLimit*: An integer that denotes the expiry limit of each token.
- *commands[commands[0]..commands[n-1]]*: A 2D integer array where each row, *commands[i]*, has *3* integers: *[command, token_id, T]*.

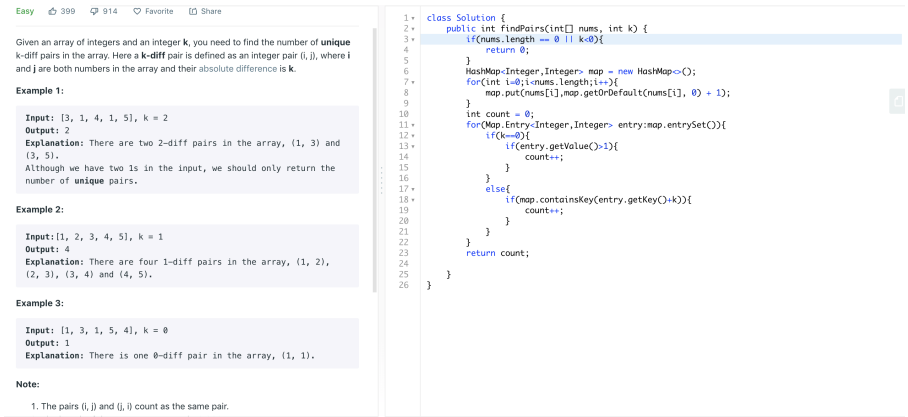
Constraints
- $1 \leq expiryLimit < 10^8$
- $1 \leq n < 10^5$
- $1 \leq token\_id < 10^8$
- $1 \leq T < 10^8$

online assessment twitter

```
public class Main {
    public static int numofToken(int expireTime,List<List<Integer>> commands){
        Map<Integer,Integer> map = new HashMap<>();
        for(List<Integer> l: commands){
            int action = l.get(0);
```

```
            int token_id = l.get(1);
            int time = l.get(2);
            //Get the token
            if(action == 0){
                map.put(token_id,expireTime+time);
             }
             else{
                if(map.containsKey(token_id)){
                    int ori_extime = map.get(token_id);
                    if(ori_extime >= time){
                        int v = map.get(token_id) + expireTime - (expireTime - time);
                        map.put(token_id,v);
                    }
                    else{
                        map.remove(token_id);
                    }
                }
             }
        }
        return map.size();
    }
```

8. K difference



Easy 399 914 Favorite Share

Given an array of integers and an integer **k**, you need to find the number of **unique** k-diff pairs in the array. Here a **k-diff** pair is defined as an integer pair (i, j), where **i** and **j** are both numbers in the array and their absolute difference is **k**.

Example 1:

```
Input: [3, 1, 4, 1, 5], k = 2
Output: 2
Explanation: There are two 2-diff pairs in the array, (1, 3) and (3, 5).
Although we have two 1s in the input, we should only return the number of unique pairs.
```

Example 2:

```
Input:[1, 2, 3, 4, 5], k = 1
Output: 4
Explanation: There are four 1-diff pairs in the array, (1, 2), (2, 3), (3, 4) and (4, 5).
```

Example 3:

```
Input: [1, 3, 1, 5, 4], k = 0
Output: 1
Explanation: There is one 0-diff pair in the array, (1, 1).
```

Note:

1. The pairs (i, j) and (j, i) count as the same pair.

```
class Solution {
    public int findPairs(int[] nums, int k) {
        if(nums.length == 0 || k<0){
            return 0;
        }
        HashMap<Integer,Integer> map = new HashMap<>();
        for(int i=0;i<nums.length;i++){
            map.put(nums[i],map.getOrDefault(nums[i], 0) + 1);
        }
        int count = 0;
        for(Map.Entry<Integer,Integer> entry:map.entrySet()){
            if(k==0){
                if(entry.getValue()>1){
                    count++;
                }
            }
            else{
                if(map.containsKey(entry.getKey()+k)){
                    count++;
                }
            }
        }
        return count;

    }
}
```

9. Buying show tickets

10. Weird Faculty

☆ Weird Faculty

This semester you are taking a course taught by a faculty member who has a weird way of grading tests. In a test, *n* questions will be asked, and the correctness of the answers is already determined. For the i^{th} question, the verdict will be *v[i] (where 0 ≤ i < n)*. If *v[i] = 1*, the answer is correct but if *v[i] = 0*, the answer is wrong.

When a test is given, you have to answer the first *k* questions *(indices 0 to k-1)* where *0 ≤ k < n*, and your friend has to answer the remaining questions *(indices k to n-1)* on the test. At the end, results are calculated as follows:

```
Your results:                   Your friend's results:

int Your_result = 0;            int YourFriend_result = 0;
for(int i=0;i<k;i++)            for(int i=k;i<n;i++)
{                               {
    if(v[i]==1)                     if(v[i]==1)
        Your_result++;                  YourFriend_result++;
    else Your_result--;             else YourFriend_result--;
}                               }
```

Choose the minimum *k* such that *Your_result > YourFriend_result*.

Function Description

Complete the function *exam* in the editor below. The function must return an integer that denotes the minimum number of questions you must answer to have *Your_result > YourFriend_result*.

exam has the following parameter(s):
v[v[0],...v[n-1]]: an array of integers

Example 1:

```
Input: [1, 0, 0, 1, 0]
Output: 0
Explanation:
If you answer 0 questions (k = 0) your_result = 0 and your friend_result = -1 (2 correct answers and 3 wrong answers).
```

Example 2:

```
Input: [1, 0, 0, 1, 1]
Output: 1
```

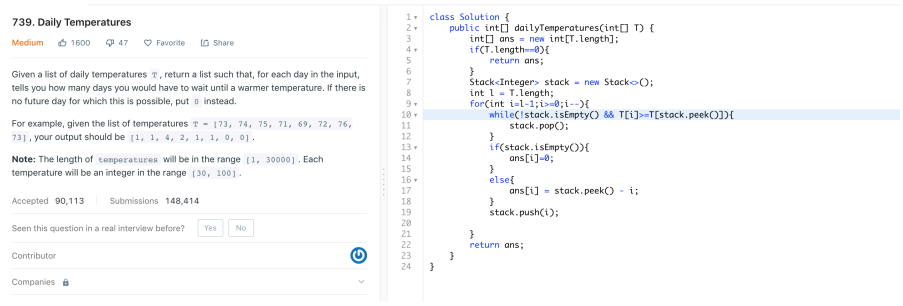
Example 3:

```
Input: [1, 1, 1, 0, 1]
Output: 2
```

```java
public static int exam(List<Integer> v) {
    int totalSum = 0;
    for(int score: v) {
        if (score == 0) totalSum -= 1;
        else totalSum += 1;
    }

    int currSum = 0;
    for(int i =0; i < v.size(); i++) {
         if (currSum > totalSum) return i;
         currSum += v.get(i) == 0 ? -1 : 1;
         totalSum -= v.get(i) == 0 ? -1 : 1;
     }
     return v.size();
}
```

11. Final discounted price




```java
class Solution {
    public int[] dailyTemperatures(int[] T) {
        int[] ans = new int[T.length];
        if(T.length==0){
            return ans;
        }
        Stack<Integer> stack = new Stack<>();
        int l = T.length;
        for(int i=l-1;i>=0;i--){
            while(!stack.isEmpty() && T[i]>=T[stack.peek()]){
                stack.pop();
            }
            if(stack.isEmpty()){
                ans[i]=0;
            }
            else{
                ans[i] = stack.peek() - i;
            }
            stack.push(i);

        }
        return ans;
    }
}
```

12. Reaching Points

780. Reaching Points

Hard 288 57 Favorite Share

A move consists of taking a point `(x, y)` and transforming it to either `(x, x+y)` or `(x+y, y)`.

Given a starting point `(sx, sy)` and a target point `(tx, ty)`, return `True` if and only if a sequence of moves exists to transform the point `(sx, sy)` to `(tx, ty)`. Otherwise, return `False`.

Examples:
Input: sx = 1, sy = 1, tx = 3, ty = 5
Output: True
Explanation:
One series of moves that transforms the starting point to the target is:
(1, 1) -> (1, 2)
(1, 2) -> (3, 2)
(3, 2) -> (3, 5)

Input: sx = 1, sy = 1, tx = 2, ty = 2
Output: False

Input: sx = 1, sy = 1, tx = 1, ty = 1
Output: True

```
class Solution {
    public boolean reachingPoints(int sx, int sy, int tx, int ty) {
        while(tx>sx && ty>sy){
            if(ty>tx){
                ty %= tx;
            }
            else{
                tx %= ty;
            }
        }
        if(tx==sx && ty>=sy && (ty-sy)%sx == 0){
            return true;
        }
         if(ty==sy && tx>=sx && (tx-sx)%sy == 0){
            return true;
        }
        return false;
    }
}
```

13. twitter scocial network

There are **N** students in a class. Some of them are friends, while some are not. Their friendship is transitive in nature. For example, if A is a **direct** friend of B, and B is a **direct** friend of C, then A is an **indirect** friend of C. And we defined a friend circle is a group of students who are direct or indirect friends.

Given a **N*N** matrix **M** representing the friend relationship between students in the class. If M[i][j] = 1, then the i_{th} and j_{th} students are **direct** friends with each other, otherwise not. And you have to output the total number of friend circles among all the students.

Example 1:

Input:
[[1,1,0],
 [1,1,0],
 [0,0,1]]
Output: 2
Explanation: The 0_{th} and 1_{st} students are direct friends, so they are in a friend circle.
The 2_{nd} student himself is in a friend circle. So return 2.

Example 2:

Input:
[[1,1,0],
 [1,1,1],
 [0,1,1]]
Output: 1
Explanation: The 0_{th} and 1_{st} students are direct friends, the 1_{st} and 2_{nd} students are direct friends,
so the 0_{th} and 2_{nd} students are indirect friends. All of them are in the same friend circle, so return 1.

```
class Solution {
    public static void dfs(int[][] M,int[] visited,int i){
        for(int j=0;j<M.length;j++){
            if(visited[j]==0 && M[i][j] == 1){
                visited[j] = 1;
                dfs(M,visited,j);
            }
        }
    }
    public int findCircleNum(int[][] M) {
        int[] visited = new int[M.length];
        int group = 0;
        for(int i =0;i<M.length;i++){
            if(visited[i]==0){
                dfs(M,visited,i);
                group++;
            }
        }
        return group;
    }
}
```

```
class Solution {
    public static void dfs(int[][] M,int[] visited,int i){
        for(int j=0;j<M.length;j++){
            if(visited[j]==0 && M[i][j] == 1){
                visited[j] = 1;
                dfs(M,visited,j);
            }
        }
    }
     public int findCircleNum(int[][] M) {
         int[] visited = new int[M.length];
         int group = 0;
         for(int i =0;i<M.length;i++){
             if(visited[i]==0){
                 dfs(M,visited,i);
                 group++;
             }
         }
```

```
        return group;
    }
}
```

14. Activate fountain

☆ Activate Fountain

There is a one-dimensional garden of length n. In each position of the n length garden, a fountain has been installed. The fountain at the i^{th} position has a value $a[i]$ (where $1 \le i \le n$) that describes the coverage limit of fountain i. A fountain can cover the range from the position ***max((i - a[i]), 1) to min((i + a[i]), n)***.

For example, if garden length $n = 3$ and $a = [1, 2, 1]$, then:

For position 1: $a[1] = 1$, range = 1 to 2.
For position 2: $a[2] = 2$, range = 1 to 3.
For position 3: $a[3] = 1$, range = 2 to 3.

In the beginning, all the fountains are switched off. Determine the minimum number of fountains you need to activate so that whole n length garden will be covered by water. In the example, the 1 fountain at position $a[2]$ covers the whole garden.

Function Description

Complete the function *fountainActivation* in the editor below. The function must return an integer that denotes the minimum number of fountains that must be activated to cover the entire garden by water.

fountainActivation has the following parameter:
- $a[a[0],...a[n-1]]$: an array of integers

Constraints
- $1 \le n \le 10^5$
- $0 \le a[i] \le min(n,100)$ (where $1 \le i \le 10^5$)

```
int numFountains(vector<int> fountains) {
    int n = fountains.size();
    vector<int> extents(n, 0);

    for (int i = 0; i < n; i++) {
        int left = max(i - fountains[i], 0);
        int right = min(i + (fountains[i] + 1), n);
        extents[left] = max(extents[left], right);
    }

    int num_fountains = 1;
    int right = extents[0], next_right = 0;
    for (int i = 0; i < n; i++) {
        next_right = max(next_right, extents[i]);
        if (i == right) {
            num_fountains++;
            right = next_right;
        }
    }

    return num_fountains;
}

int main() {
    assert(numFountains({0,0,0,3,0,0,2,0,0}) == 2);
    assert(numFountains({3,0,2,0,1,0}) == 2);
    assert(numFountains({3,0,1,0,1,0}) == 2);
    assert(numFountains({3,0,1,0,0,1}) == 2);
    assert(numFountains({2,0,2,0,1,0}) == 2);
    assert(numFountains({2,0,0,0,0}) == 3);
    assert(numFountains({0,0,0,0,0}) == 5);
    assert(numFountains({1,2,1}) == 1);
```

```
33      assert(numFountains({0,1,0}) == 1);
34  }
```

15. Coloring the blocks

There are a row of n houses, each house can be painted with one of the three colors: red, blue or green. The cost of painting each house with a certain color is different. You have to paint all the houses such that no two adjacent houses have the same color.

The cost of painting each house with a certain color is represented by a n x 3 cost matrix. For example, costs[0][0] is the cost of painting house 0 with color red; costs[1][2] is the cost of painting house 1 with color green, and so on... Find the minimum cost to paint all houses.

```
public int minCost(int[][] costs) {
    if(costs==null||costs.length==0)
        return 0;

    for(int i=1; i<costs.length; i++){
        costs[i][0] += Math.min(costs[i-1][1], costs[i-1][2]);
        costs[i][1] += Math.min(costs[i-1][0], costs[i-1][2]);
        costs[i][2] += Math.min(costs[i-1][0], costs[i-1][1]);
    }

     int m = costs.length-1;
     return Math.min(Math.min(costs[m][0], costs[m][1]), costs[m][2]);
}
```

16. Parking Dilemma

17. Get set on

(Java) LeetCode 39. Combination Sum —— 组合总和

Given a set of candidate numbers (`candidates`) (without duplicates) and a target number (`target`), find all unique combinations in `candidates` where the candidate numbers sums to `target`.

The same repeated number may be chosen from `candidates` unlimited number of times.

Note:

- All numbers (including `target`) will be positive integers.
- The solution set must not contain duplicate combinations.

Example 1:

```
Input: candidates = [2,3,6,7], target = 7,
A solution set is:
[
  [7],
  [2,2,3]
]
```

Example 2:

```
Input: candidates = [2,3,5], target = 8,
A solution set is:
[
  [2,2,2,2],
  [2,3,3],
  [3,5]
]
```

经典深度优先搜索+回溯问题。因为candidates数组是没有重复的，且每个数字都可以用无限次，代表每次搜索的时候都要从自己开始。而当所有从自己出发的路径考虑完毕后，要把自己删除出去（即回溯）进而搜索下一个数值。如果用有向图的思想，把每个数组下标都想成一个节点，节点之间的权值是从这个节点下标出发时对应的数组元素（即出权值），那么问题转化为寻找权值为target的路径。每一个节点都可以连通到其他节点，且每个节点又都有自环。模型想象到这里就更容易理解下面的代码。

```
class Solution {
    public static List<List<Integer>> combinationSum(int[] candidates, int target) {
        List<List<Integer>> res = new ArrayList<>();
        if(candidates == null) return res;
```

```
        List<Integer> list = new ArrayList<>(); //存放路径上的节点
        get(candidates, 0, target, list, res);
        return res;
    }

    public static void get(int[] nums, int i, int target, List<Integer> list, List<List<Intege
        if (target < 0) return; //如果路径值小于0，搜索结束，因为所有权值都是大于零，即一旦达到负数没有办
        if (target == 0) { //如果目标为0代表路径权值和已经满为target，满足条件，故添加路径到结果集
            res.add(new ArrayList<>(list)); //
            return;
        }
        for (int p = i; p < nums.length; p++) {
            list.add(nums[p]); //添加节点到路径
            get(nums, p, target - nums[p], list, res); //在已有路径基础上继续查找更新权值后后的路径，
            list.remove(list.size() - 1); //当从当前节点的所有路径都搜索完毕后，将其剔除，之后重新搜索从
        }
    }
}
```

(Java) LeetCode 40. Combination Sum II —— 组合总和 II

Given a collection of candidate numbers (`candidates`) and a target number (`target`), find all unique combinations in `candidates` where the candidate numbers sums to `target`.

Each number in `candidates` may only be used once in the combination.

Note:

- All numbers (including `target`) will be positive integers.
- The solution set must not contain duplicate combinations.

Example 1:

```
Input: candidates = [10,1,2,7,6,1,5], target = 8,
A solution set is:
[
  [1, 7],
  [1, 2, 5],
  [2, 6],
  [1, 1, 6]
]
```

Example 2:

```
Input: candidates = [2,5,2,1,2], target = 5,
A solution set is:
[
  [1,2,2],
  [5]
]
```

LeetCode 39. Combination Sum —— 组合总和的升级版。还是想象成图来帮助理解。和第39题相比本题有两个变化。第一，本题有重复节点；第二，每个节点只能用一次，即没有自环。结合对39代码注释的理解，稍稍更改即可得到本题的解题思路：

如何处理自环问题？每次搜索新路径的时候都从其下一个节点开始，而不是从它本身开始；

如何处理去重问题？每次回溯的时候，刚刚被剔除的节点不能在任何时候再被重新加入到路径上。如何处理这个"任何时候"呢？要么用map标记被剔除的节点直到路径搜索结束，要么应用排序，将所有有相同出权值的节点都放到一起，这样可以方便找到下一个出权值不同的节点。

想完这两个不同点，这道题就解决了。详见代码注释。

```
class Solution {
    public List<List<Integer>> combinationSum2(int[] candidates, int target) {
        List<List<Integer>> res = new ArrayList<>();
        if (candidates == null || candidates.length == 0 || target < 0) return res;
        List<Integer> list = new ArrayList<>();
        Arrays.sort(candidates); //排序，使得寻找相同出权值的节点变得容易
        get(candidates, target, 0, list, res);
```

```
        return res;
    }

    private void get(int[] candidates, int target, int i, List<Integer> list, List<List<Intege
        if (i > candidates.length || target < 0) return; //因为没有自环，所以每次都是从下一个节点开始
        if (target == 0) { //满足条件，添加至结果集
            res.add(new ArrayList<>(list));
            return;
        }
        for (int p = i; p < candidates.length; p++) {
            list.add(candidates[p]); //添加节点到路径
            get(candidates, target - candidates[p], p+1, list, res); //因为没有自环，所以每次搜索更
            list.remove(list.size()-1); //回溯，将当前节点从路径中剔除
            while (p < candidates.length - 1 && candidates[p] == candidates[p+1]) p++; //因为存
        }
    }
}
```

18. Sub Palindrome

647. Palindromic Substrings

Medium 1639 85 Favorite Share

Given a string, your task is to count how many palindromic substrings in this string.

The substrings with different start indexes or end indexes are counted as different substrings even they consist of same characters.

Example 1:

```
Input: "abc"
Output: 3
Explanation: Three palindromic strings: "a", "b", "c".
```

Example 2:

```
Input: "aaa"
Output: 6
Explanation: Six palindromic strings: "a", "a", "a", "aa", "aa", "aaa".
```

```
class Solution {
    public int countSubstrings(String s) {
        if(s.length() == 0){
            return 0;
        }
        int count = 0;
        for(int i=0;i<s.length();i++){
            count+=extendFunc(s,i,i);
            count+=extendFunc(s,i,i+1);
        }
        return count;
    }

    public static int extendFunc(String s,int left,int right){
        int count = 0;
        while(left>=0 && right<s.length() && s.charAt(left) == s.charAt(right)){
            count++;
            left--;
            right++;
        }
        return count;
    }
}
```

19.Restocking the Warehouse

20.Balanced Sales Array

21. university career fair

University Career Fair

Sam is part of the organizing team arranging the university's career fair and has list of companies and their respective arrival times and durations. Due to university-wide budget cuts, there is only one stage/dais available on the entire campus so only one event can occur at a time. Given each company's arrival time and the duration they will stay, determine the maximum number of promotional events that can be hosted during the career fair.

For example, there are *n = 5* companies that will arrive at times *arrival = [1, 3, 3, 5, 7]* and will stay for *duration = [2, 2, 1, 2, 1]*. The first company arrives at time *1* and stays for *2* hours. At time *3*, two companies arrive, but only *1* can stay for either *1* or *2* hours. The next companies arrive at times *5* and *7* and do not conflict with each other. In total, there can be a maximum of *4* promotional events.

Function Description

Complete the function *maxEvents* in the editor below. It must return an integer that represents the maximum number of promotional events that can be hosted.

maxEvents has the following parameter(s):

- *arrival[arrival[0],...arrival[n-1]]*: an array of integers where i^{th} element is the arrival time of the i^{th} company.
- *duration[duration[0],...duration[n-1]]*: an array of integers where i^{th} element is the duration that the i^{th} company's stay at the career fair.

Constraints

- $1 \le n \le 50$
- $1 \le arrival[i] \le 1000$
- $1 \le duration[i] \le 1000$
- *Both 'arrival' array and 'duration' array will have equal number of elements*

是greedy interval schedule的问题

435. Non-overlapping Intervals

Medium 579 26 Favorite Share

Given a collection of intervals, find the minimum number of intervals you need to remove to make the rest of the intervals non-overlapping.

Example 1:

```
Input: [[1,2],[2,3],[3,4],[1,3]]
Output: 1
Explanation: [1,3] can be removed and the rest of intervals are non-overlapping.
```

Example 2:

```
Input: [[1,2],[1,2],[1,2]]
Output: 2
Explanation: You need to remove two [1,2] to make the rest of intervals non-overlapping.
```

Example 3:

```
Input: [[1,2],[2,3]]
Output: 0
Explanation: You don't need to remove any of the intervals since they're already non-overlapping.
```

```java
class Solution {
    public int eraseOverlapIntervals(int[][] intervals) {
        if(intervals.length == 0 || intervals[0].length == 0){
            return 0;
        }
        int num = intervals.length;
        int count = 1;
        Arrays.sort(intervals,new Comparator<int[]>(){
            public int compare(int[] a,int[] b){
                return a[1]-b[1];
            }
        });
        int endTime = intervals[0][1];
        for(int i=0;i<num;i++){
            if(intervals[i][0]>=endTime){
                count++;
                endTime = intervals[i][1];
            }
        }
        return num-count;
    }
}
```

22. Anagram Difference

Two words are anagrams of one another if their letters can be rearranged to form the other word.

In this challenge, you will be given a string. You must split it into two contiguous substrings, then determine the minimum number of characters to change to make the two substrings into anagrams of one another.

For example, given the string 'abccde', you would break it into two parts: 'abc' and 'cde'. Note that all letters have been used, the substrings are contiguous and their lengths are equal. Now you can change 'a' and 'b' in the first substring to 'd' and 'e' to have 'dec' and 'cde' which are anagrams. Two changes were necessary.

Function Description

Complete the anagram function in the editor below. It should return the minimum number of characters to change to make the words anagrams, or -1 if it's not possible.

anagram has the following parameter(s):

- s: a string

Input Format

The first line will contain an integer, q, the number of test cases.
Each test case will contain a string s which will be concatenation of both the strings described above in the problem.
The given string will contain only characters in the range ascii[a-z].

Constraints

- $1 \leq q \leq 100$
- $1 \leq |s| \leq 10^4$
- s consists only of characters in the range ascii[a-z].

Output Format

For each test case, print an integer representing the minimum number of changes required to make an anagram. Print -1 if it is not possible.

. .

```
6
aaabbb
ab
abc
mnop
xyyx
xaxbbbxx
```

Sample Output

```
3
1
-1
2
0
1
```

Explanation

Test Case #01: We split s into two strings $S1$='aaa' and $S2$='bbb'. We have to replace all three characters from the first string with 'b' to make the strings anagrams.

Test Case #02: You have to replace 'a' with 'b', which will generate "bb".

Test Case #03: It is not possible for two strings of unequal length to be anagrams of one another.

Test Case #04: We have to replace both the characters of first string ("mn") to make it an anagram of the other one.

Test Case #05: $S1$ and $S2$ are already anagrams of one another.

Test Case #06: Here S1 = "xaxb" and S2 = "bbxx". You must replace 'a' from S1 with 'b' so that S1 = "xbxb".

```java
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int t = sc.nextInt();
        while (t-->0){
            String str = sc.next();
             int len = str.length(), count = 0;
             if (len%2!=0){
                 System.out.println(-1);
                 continue;
             }
             String s1 = str.substring(0,len/2);
             String s2 = str.substring(len/2,len);
             char[] s1Chars = s1.toCharArray();
             for (char c : s1Chars){
                 int index = s2.indexOf(c);
                 if (index == -1){
                     count++;
```

```
                } else {
                    s2 = s2.substring(0,index)+s2.substring(index+1);
                }
            }
            System.out.println(count);
        }
    }
}
```