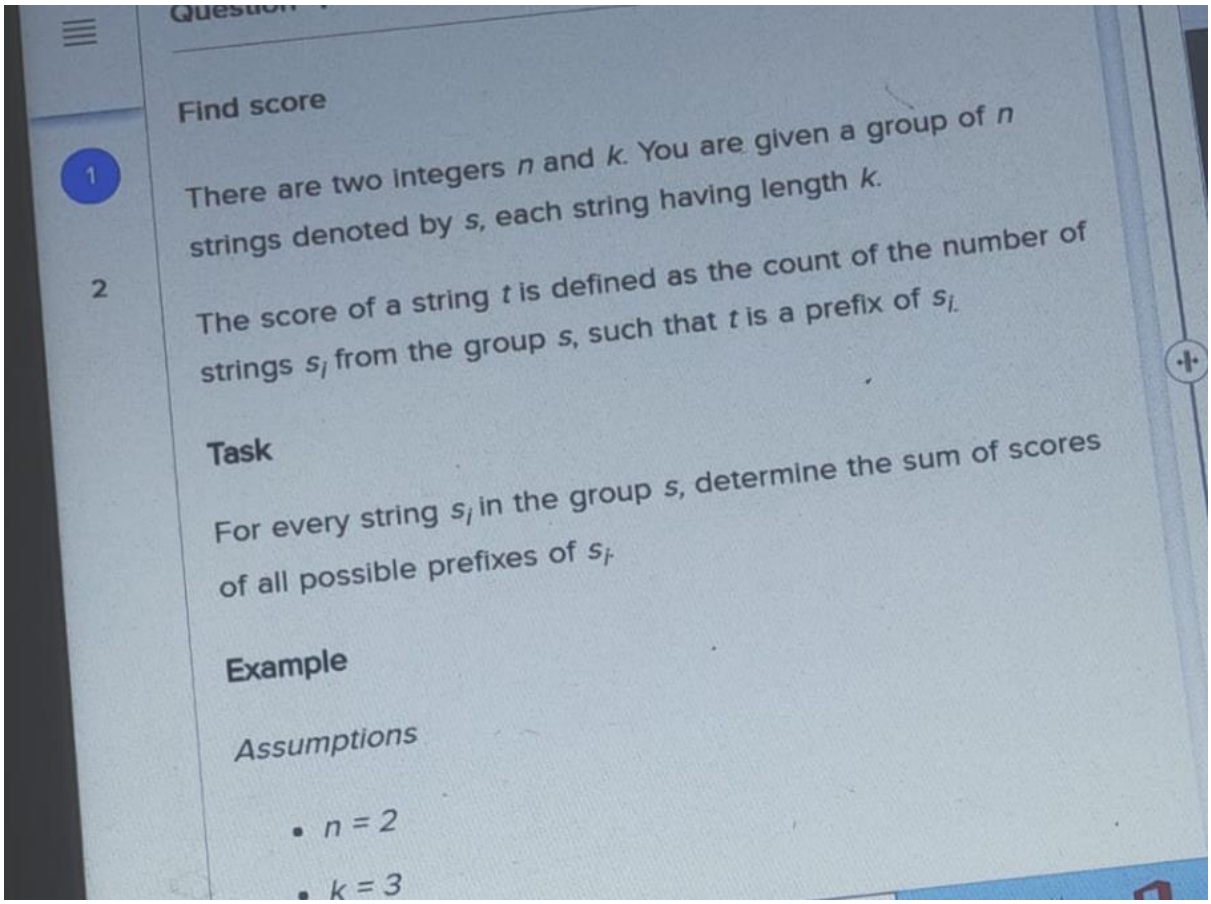
# Google Questions

1)



Find score

There are two integers $n$ and $k$. You are given a group of $n$ strings denoted by $s$, each string having length $k$.

The score of a string $t$ is defined as the count of the number of strings $s_i$ from the group $s$, such that $t$ is a prefix of $s_i$.

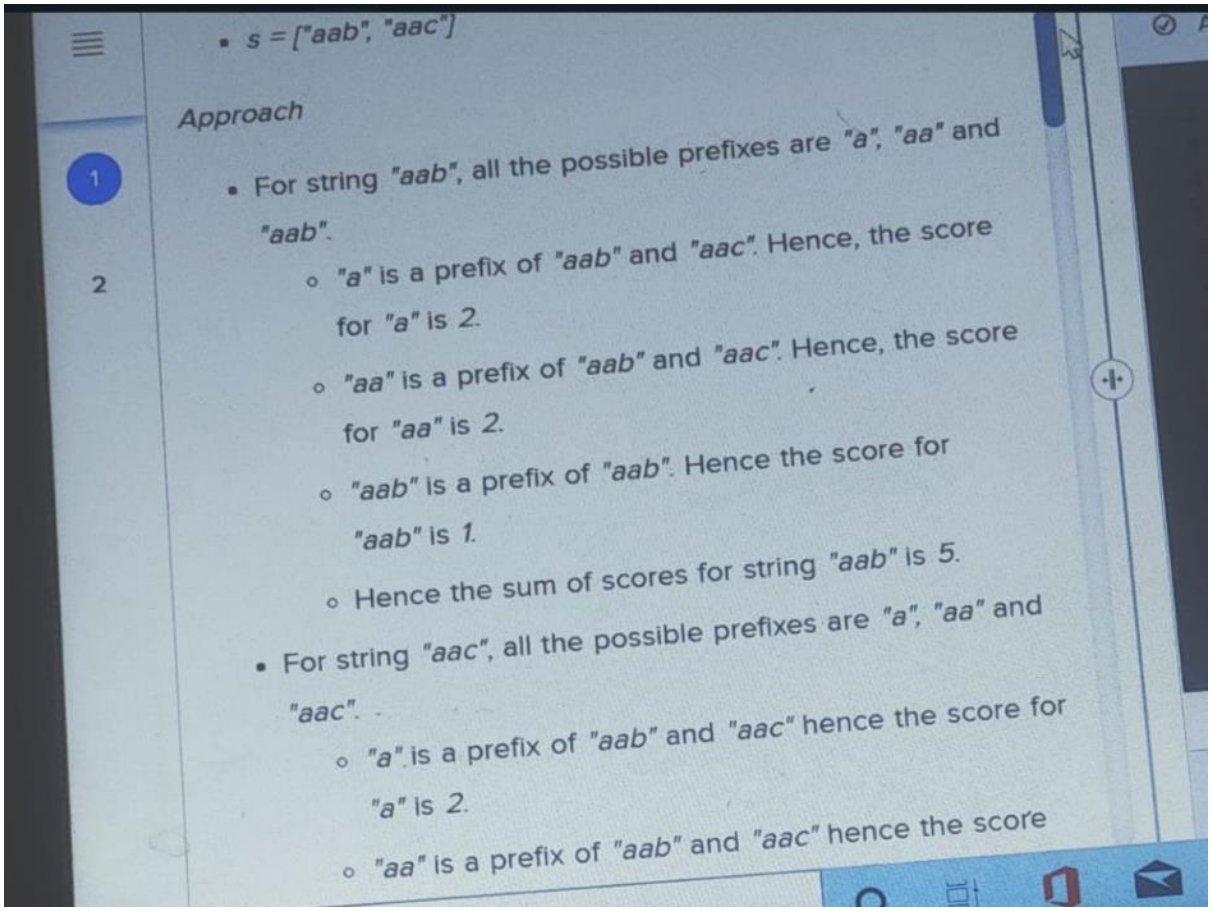
**Task**

For every string $s_i$ in the group $s$, determine the sum of scores of all possible prefixes of $s_i$.

**Example**

*Assumptions*

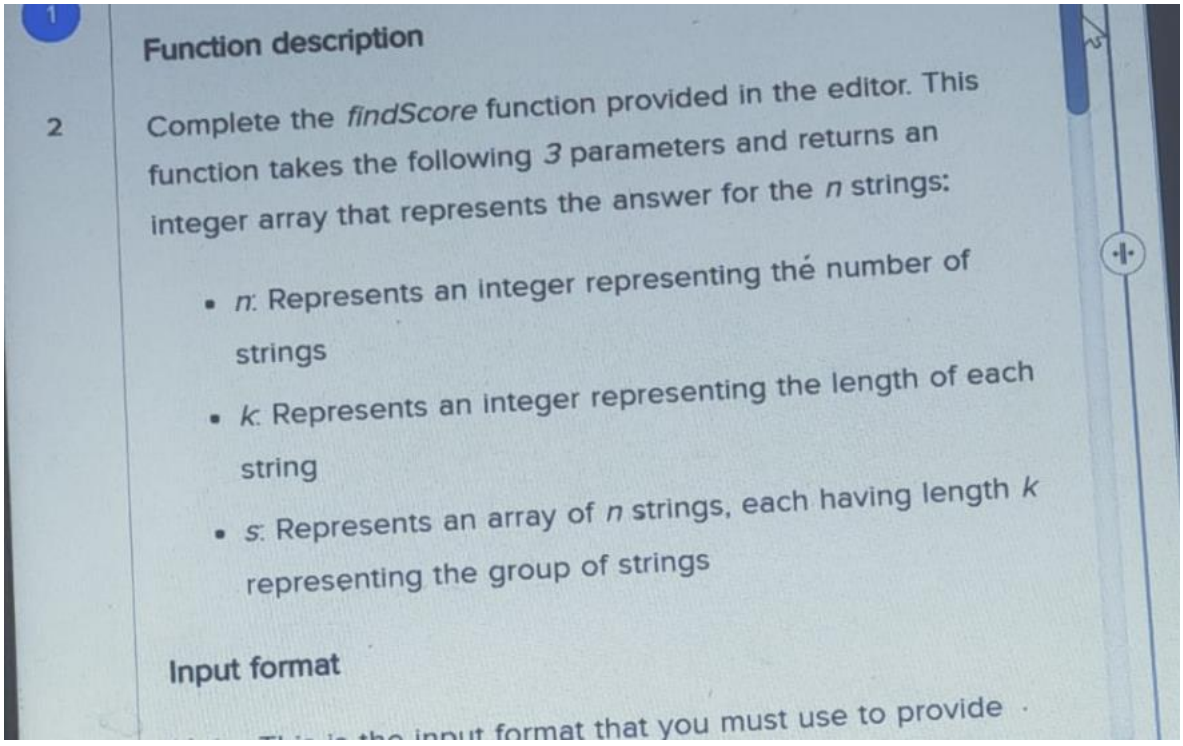
- $n = 2$
- $k = 3$



- $s$ = ["aab", "aac"]

*Approach*

- For string *"aab"*, all the possible prefixes are *"a"*, *"aa"* and *"aab"*.
  - *"a"* is a prefix of *"aab"* and *"aac"*. Hence, the score for *"a"* is *2*.
  - *"aa"* is a prefix of *"aab"* and *"aac"*. Hence, the score for *"aa"* is *2*.
  - *"aab"* is a prefix of *"aab"*. Hence the score for *"aab"* is *1*.
  - Hence the sum of scores for string *"aab"* is *5*.
- For string *"aac"*, all the possible prefixes are *"a"*, *"aa"* and *"aac"*.
  - *"a"* is a prefix of *"aab"* and *"aac"* hence the score for *"a"* is *2*.
  - *"aa"* is a prefix of *"aab"* and *"aac"* hence the score



**Function description**

Complete the *findScore* function provided in the editor. This function takes the following *3* parameters and returns an integer array that represents the answer for the *n* strings:

- *n*: Represents an integer representing the number of strings
- *k*: Represents an integer representing the length of each string
- *s*: Represents an array of *n* strings, each having length *k* representing the group of strings

**Input format**

This is the input format that you must use to provide

custom input (available above the Compile and Test button).

2

- The first line contains 2 space-separated integers $n$ and $k$, representing the number of strings and the length of each string.
- Next $n$ lines, each line contains one string each having length $k$.

**Output format**

Print $n$ space-separated integers, where $i^{th}$ integer represents the sum of scores of all possible prefixes of $i^{th}$ string.

**Constraints**

2

**Constraints**

$1 \leq n \leq 10^6$

$1 \leq k \leq 10^6$

$1 \leq n \times k \leq 10^6$

All strings contain lowercase latin alphabets

**Code snippets (also called starter code/boilerplate code)**

2)

## Special paths

You are given the following:

- A Tree with $N$ nodes
- An array $A$ denoting the value of each node

A path is called *special path* if the following conditions are satisfied:

- All nodes in the path are traversed exactly once.
- The value of starting node and terminating node in the path is same and $starting\ node \neq terminating\ node$.
- The values of any node in the path are not greater than the value of starting node.

### Task

Count the number of *special paths* in the tree.

Note: Two paths are different if they contain at least one different node.

### Task

Count the number of *special paths* in the tree.

*Note:* Two paths are different if they contain at least one different node.

### Example

*Assumptions*

- $N = 5$
- *edges = [(1, 2), (1, 3), (3, 4), (3, 5)]*
- *A = [2, 3, 1, 2, 3]*

*Approach*

The following paths are *special paths*:

- *Path 1(2) -> 3(1) -> 4(2)*
- *Path 2(3) -> 1(2) -> 3(1) -> 5(3)*

Thus, the answer is 2.

**Function description**

Complete the function *countSpecialPaths* provided in the editor. The function takes the following parameters and returns the count of *special paths* in the given tree:

- *N*: Represents the number of nodes in the tree
- *edges*: Represents the edges in a tree. It's a 2D-array of size $N \times 2$ where each row denotes edge between the two nodes
- *A*: Represents the value of each node

**Input format**

**Note**: This is the input format that you must use to provide custom input (available above the **Compile and Test** button).

- The first line contains *T* denoting the number of test cases. *T* also specifies the number of times you have to run the *countSpecialPaths* function on a different set of inputs.

- The next *N - 1* line contains 2 space-separated integers denoting an edge between these two nodes.
- The next line contains *N* space-separated integers denoting the value of each node.

**Output format**

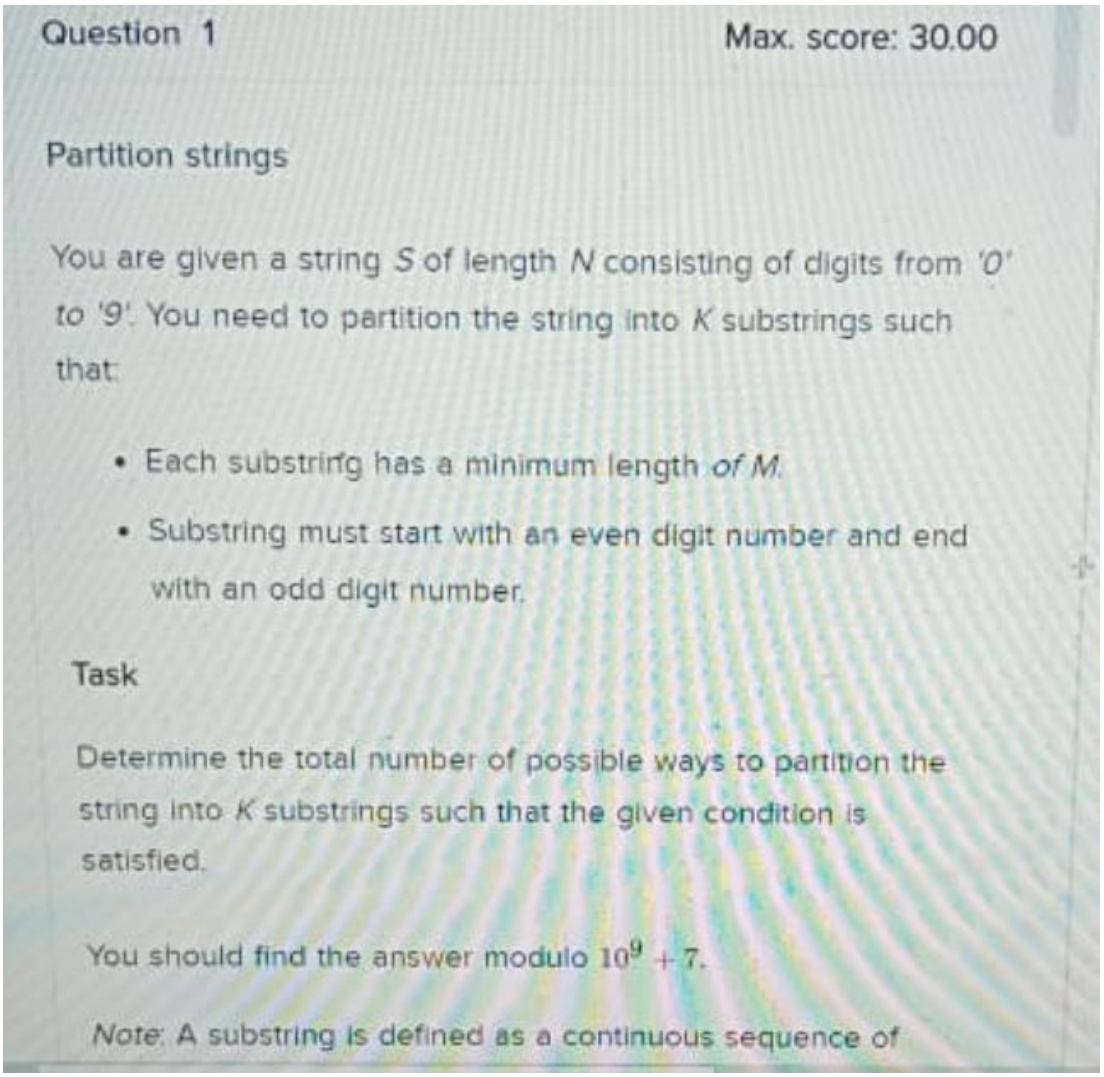
For each test case, print the count of *special paths* in the tree in a new line.

**Constraints**

$1 \leq T \leq 10$
$2 \leq N \leq 10^5$
$1 \leq A_i \leq 10^9 \ \forall i \in [1, N]$

*It is guaranteed that the given input forms a tree.*

3)



Question 1 Max. score: 30.00

Partition strings

You are given a string *S* of length *N* consisting of digits from *'0'* to *'9'*. You need to partition the string into *K* substrings such that:

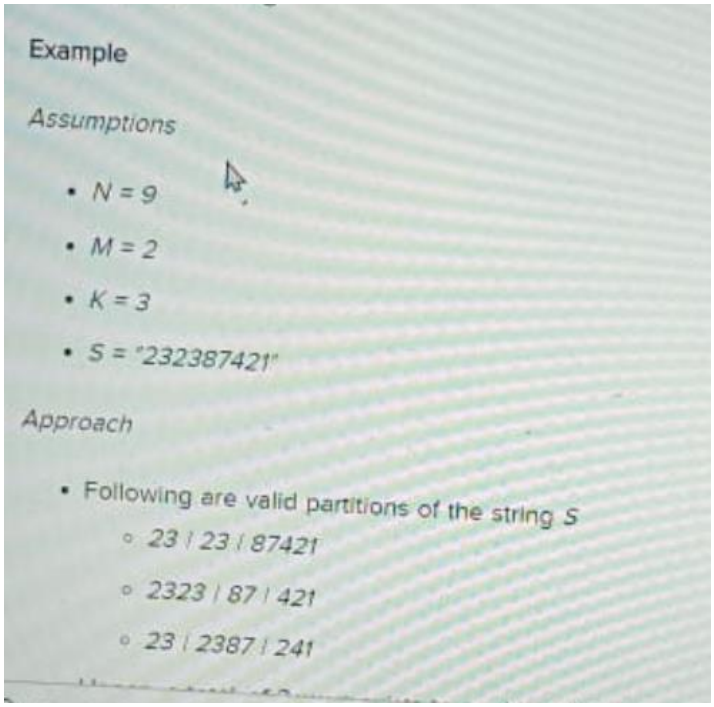
- Each substring has a minimum length *of M*.
- Substring must start with an even digit number and end with an odd digit number.

Task

Determine the total number of possible ways to partition the string into *K* substrings such that the given condition is satisfied.

You should find the answer modulo $10^9 + 7$.

*Note*: A substring is defined as a continuous sequence of



Example

*Assumptions*

- *N = 9*
- *M = 2*
- *K = 3*
- *S = "232387421"*

*Approach*

- Following are valid partitions of the string *S*
  - *23 | 23 | 87421*
  - *2323 | 87 | 421*
  - *23 | 2387 | 241*

- Following are valid partitions of the string *S*
  - *23 | 23 | 87421*
  - *2323 | 87 | 421*
  - *23 | 2387 | 241*
- Hence, a total *of 3* ways exists to partition the string *S*

**Function description**

Complete the solve function provided in the editor. This function takes the following *4* parameters and returns the required answer:

- *N*: Represents the length of string *S*
- *M*: Represents the minimum length of substrings in a partition
- *K*: Represents the number of substrings in the partition
- *S*: Represents the string *S*

**Input format**

- $M$: Represents the minimum length of substrings in a partition
- $K$: Represents the number of substrings in the partition
- $S$: Represents the string $S$

**Input format**

**Note**: This is the input format that you must use to provide custom input (available above the **Compile and Test** button).

- The first line contains an integer $N$.
- The second line contains an integer $M$.
- The third line contains an integer $K$.
- The last line contains a string $S$.

**Output format**

Print the answer representing the number of ways of partitioning the string $S$ modulo $10^9 + 7$.

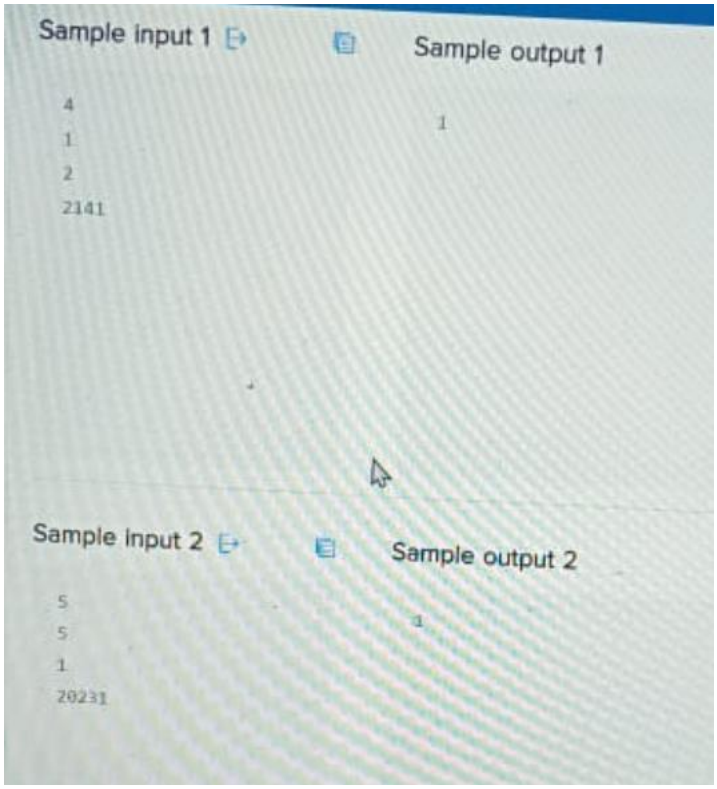
**Constraints**

$1 \leq N \leq 2 \times 10^3$
$1 \leq M \leq N$
$1 \leq K \leq N$
$S[i] \in \{'0' - '9'\}$

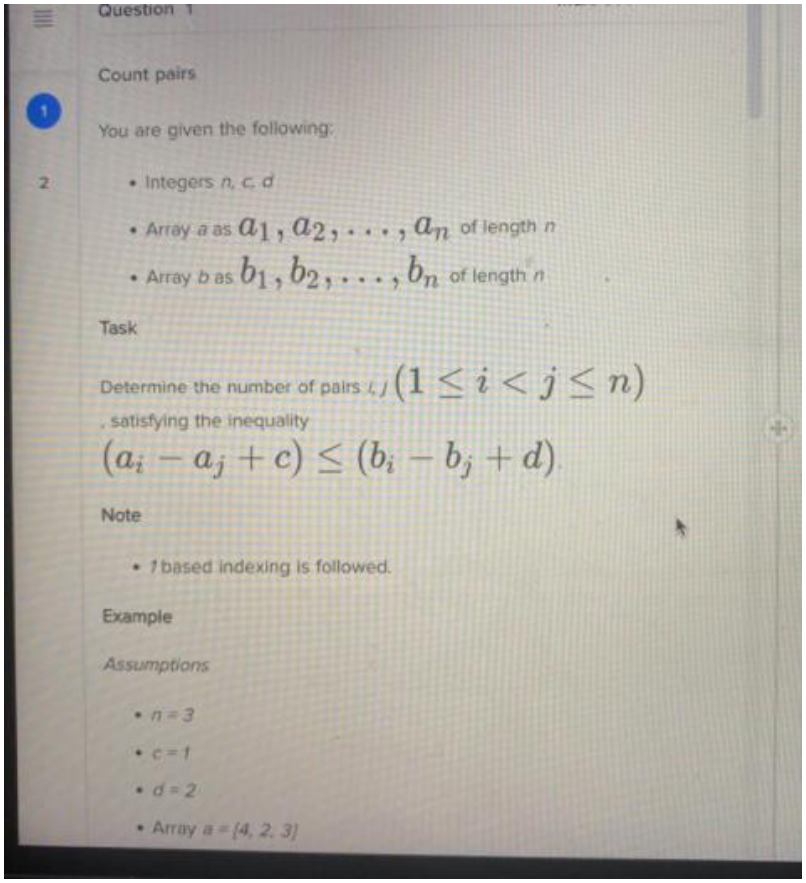
**Code snippets (also called starter code/boilerplate code)**

This question has code snippets for C, CPP, Java, and Python.

| Sample input | Sample output |
| --- | --- |
| 6<br>3<br>2<br>621433 | 1 |


Sample input 1
Sample output 1
4
1
2
2141
1
Sample input 2
Sample output 2
5
5
1
20231
1

4)



Question 1

Count pairs

You are given the following:

- Integers $n, c, d$
- Array $a$ as $a_1, a_2, \ldots, a_n$ of length $n$
- Array $b$ as $b_1, b_2, \ldots, b_n$ of length $n$

Task

Determine the number of pairs $i, j (1 \leq i < j \leq n)$ , satisfying the inequality $(a_i - a_j + c) \leq (b_i - b_j + d)$.

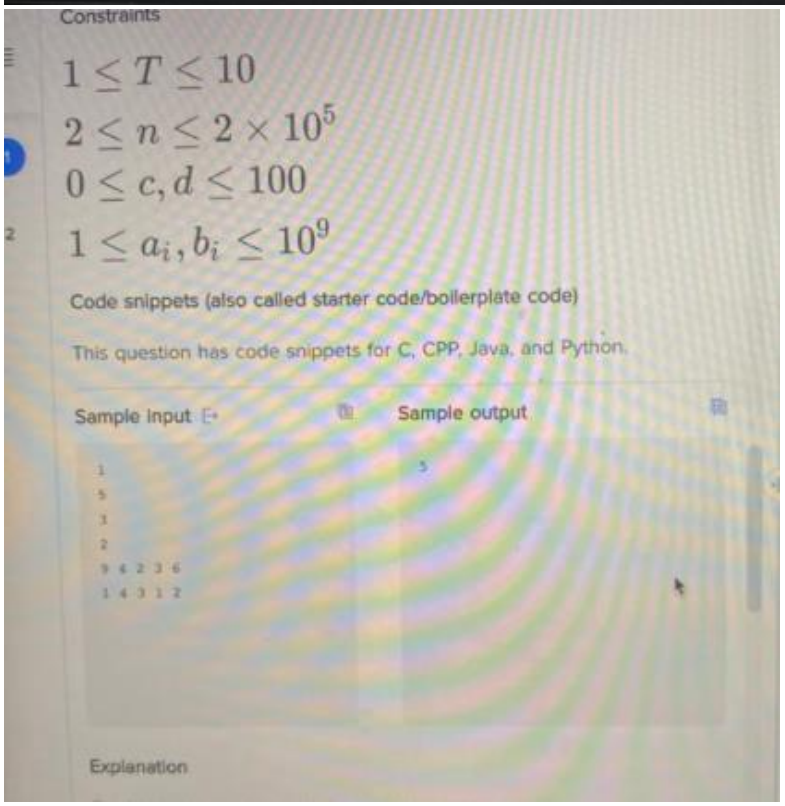
Note

- 1 based indexing is followed.

Example

*Assumptions*

- $n = 3$
- $c = 1$
- $d = 2$
- Array $a = [4, 2, 3]$



Constraints

$1 \leq T \leq 10$

$2 \leq n \leq 2 \times 10^5$

$0 \leq c, d \leq 100$

$1 \leq a_i, b_i \leq 10^9$

Code snippets (also called starter code/boilerplate code)

This question has code snippets for C, CPP, Java, and Python.

Sample Input

```
1
5
1
2
9 4 2 3 6
1 4 3 1 2
```

Sample output

```
5
```

Explanation

## Explanation

The first line represents the number of test cases, $T = 1$.

**For test case 1**

*Given*

- $n = 5$
- $c = 3$
- $d = 2$
- Array $a = [9, 4, 2, 3, 6]$
- Array $b = [1, 4, 3, 1, 2]$

*Approach*

5 pairs of $(i, j) \in \{(2, 4), (2, 5), (4, 5), (3, 5), (3, 4)\}$ satisfying the given inequality

$$(a_i - a_j + c) \leq (b_i - b_j + d)$$

Therefore, the answer is 5.

## Sample input 2

```
3
4
1
2
3 2 3 10
10 10 2 9
6
1
2
7 2 2 7 7 8
```

## Sample output 2

```
6
14
32
```

View more

## Sample input 1

```
2
10
0
1
10 7 3 8 7 4 9 9 3 4
1 9 5 6 8 5 10 6 6 1
5
1
2
9 1 1 2 3
```

## Sample output 1

```
28
4
```

5)

Question 2 Max. score: 30.00

Good path

You are given the following:

- A tree of $n$ nodes numbered from *1* to $n$ and $n - 1$ edges
- An array $C$ of length $n$

The element $C_i$ for each $(1 \leq i \leq n)$ represents the value associated with the $i_{th}$ node. Now, let's call a *simple path* good if the frequency of the value of any one of the nodes in the path is at least half of the length of the path rounded down to the next greatest integer. The length of a path from node $A$ to node $B$ is the number of nodes you encounter in that unique path while going from $A$ to $B$.

Task

Determine the count of the total good path starting from node *1*.

*Notes*

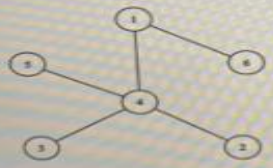
- *1*-based indexing is followed.
- In graph theory, a simple path is a path in a graph that does not have repeating vertices
- A tree is an undirected graph in which any two vertices are connected by exactly one path, or equivalently a connected acyclic undirected graph.

- In graph theory, a simple path is a path in a graph that does not have repeating vertices
- A tree is an undirected graph in which any two vertices are connected by exactly one path, or equivalently a connected acyclic undirected graph.

**Example**

*Assumptions*

- $n = 6$
- $C = [1, 2, 1, 3, 1, 1]$
- Tree with edges as follow:



*Approach*

All paths starting from $node_1$:

- $C_1$ value in the path from *1* to *1* is ($node_1 = 1$) and the frequency of $C_1$ satisfies the given condition so it is a good path.
- $C_i$ values in path from *1* to *2* are ($node_1 = 1$, $node_4 = 3$, $node_2 =$

- $C_i$ values in path from *1* to *3* are ($node_1 = 1$, $node_4 = 3$, $node_3 = 1$) and the frequency of $C_1$ satisfies the given condition so it is a good path.
- $C_i$ values in path from *1* to *4* are ($node_1 = 1$, $node_4 = 3$) and the frequency of $C_1$ satisfies the given condition so it is a good path.
- $C_i$ values in path from *1* to *5* are ($node_1 = 1$, $node_4 = 3$, $node_5 = 1$) and the frequency of $C_1$ satisfies the given condition so it is a good path.
- $C_i$ values in path from *1* to *6* are ($node_1 = 1$, $node_6 = 1$) and the frequency of $C_1$ satisfies the given condition so it is a good path.

Hence, the final answer will be *5*.

**Function description**

Complete the *solve* function provided in the editor. This function takes the following *3* parameters and returns an integer:

- *n*: Represents the number of nodes in a tree
- *C*: Represents an array denoting the node values
- *edges*: Represents a 2d array of edges

**Input format**

**Note**: This is the input format that you must use to provide custom input (available above the **Compile and Test** button)

- The first line contains an integer *T* denoting the number of test

- $C_i$ value in the path from 1 to 1 is ($node_1 = 1$) and the frequency of $C_1$ satisfies the given condition so it is a good path.
- $C_i$ values in path from 1 to 2 are ($node_1 = 1$, $node_2 = 2$) and the frequency of $C_1$ satisfies the given condition so it is a good path.
- $C_i$ values in path from 1 to 3 are ($node_1 = 1$, $node_2 = 2$ , $node_3 = 3$) since no $C_i$ satisfies the given condition so it is not a good path.
- $C_i$ values in path from 1 to 4 are ($node_1 = 1$, $node_4 = 4$) and the frequency of $C_1$ satisfies the given condition so it is a good path.
- $C_i$ values in path from 1 to 5 are ($node_1 = 1$, $node_2 = 2$ , $node_5 = 5$) since no $C_i$ satisfies the given condition so it is not a good path.

Hence, the final answer is *3*.

**The second test case**

- Explained in the problem statement example above.

---

The following test cases are the actual test cases of this question that may be used to evaluate your submission.

**Sample input 1**

```
3
10
1 2 9 1 6 9 1 10 9 1
4 3
6 5
5 10
4 9
7 5
4 6
1 7
```

**Sample output 1**

```
10
5
3
```

Sample input

```
2
5
1 2 3 4 5
1 2
2 3
1 4
2 5
6
1 2 1 3 1 1
1 4
```

Sample output

```
3
5
```

View more

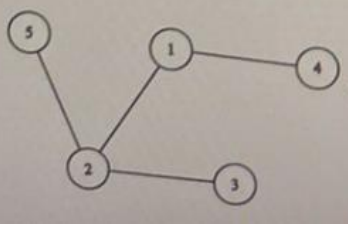
## Explanation

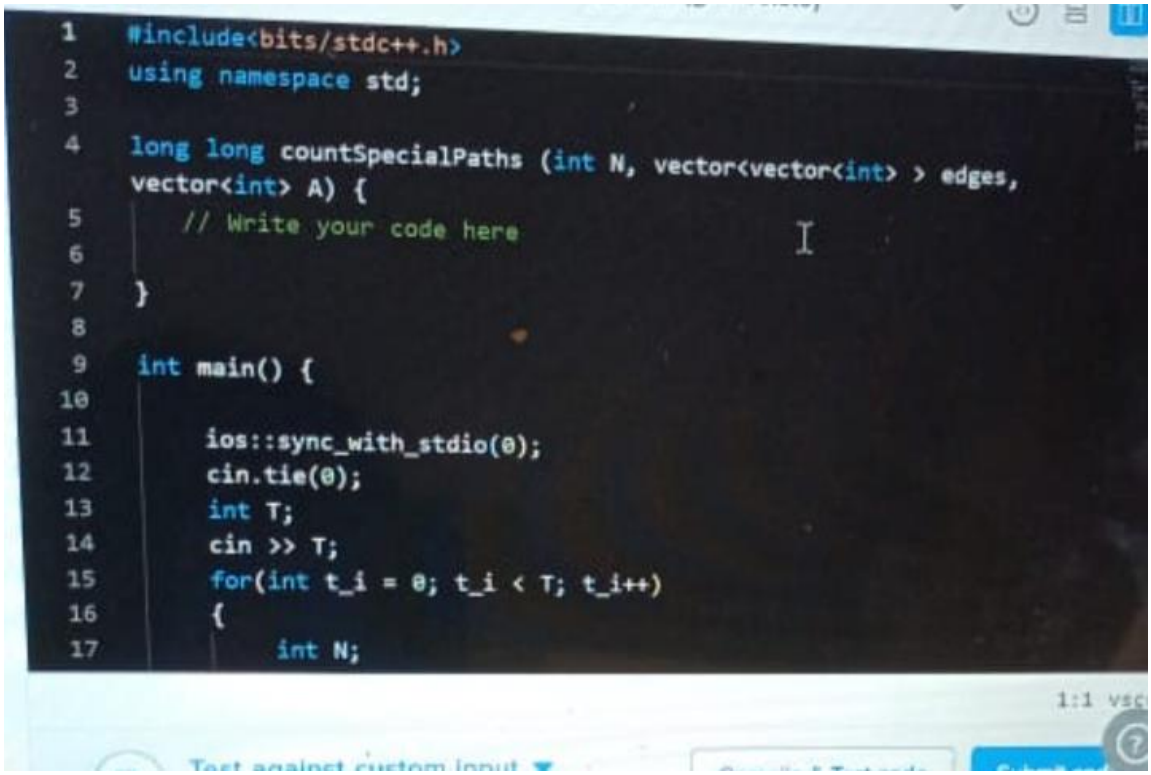
The first line contains the number of test cases, $T = 2$.

**The first test case**

*Given*

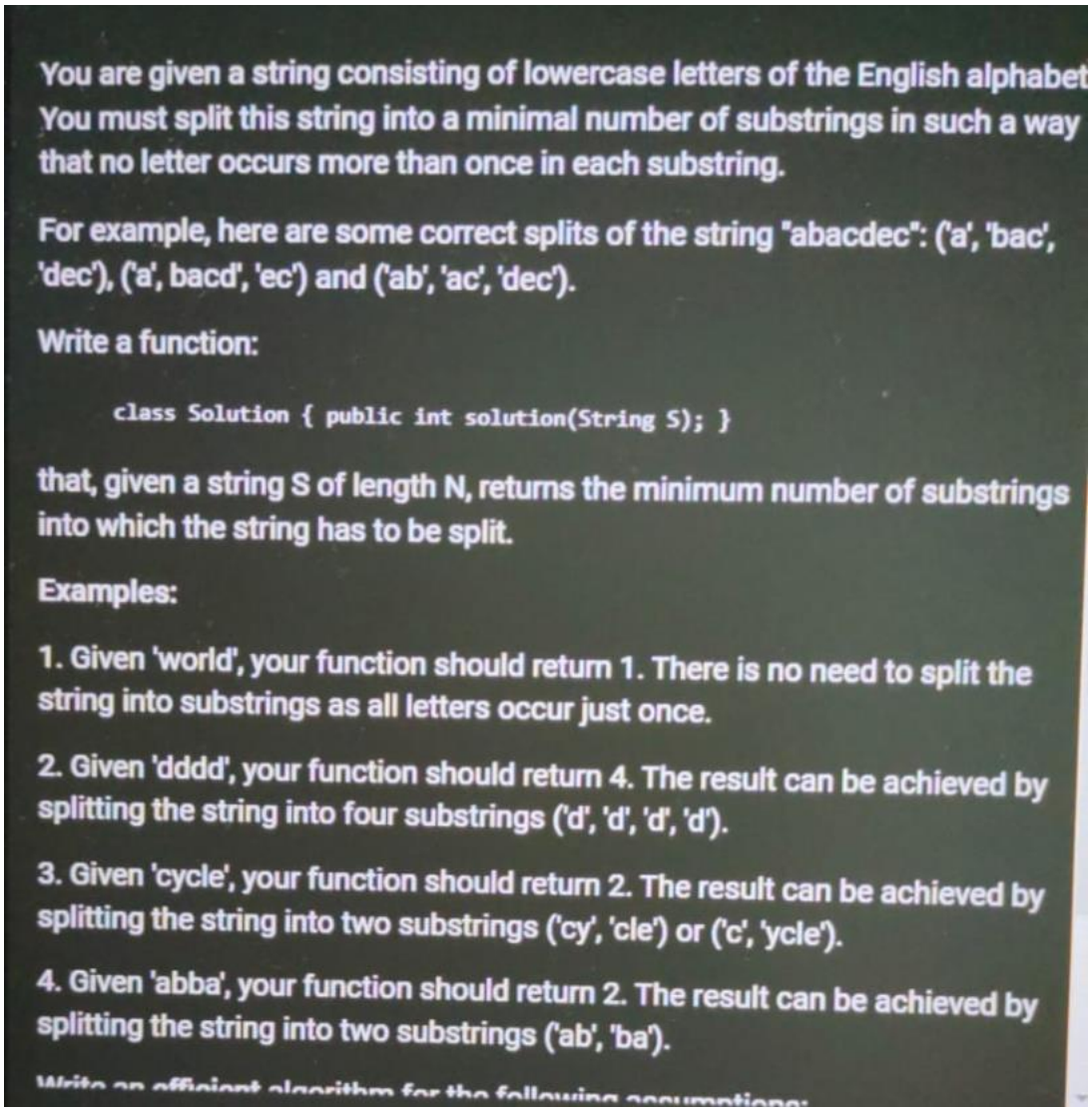
- $n = 5$
- $C = [1, 2, 3, 4, 5]$
- Tree:





```cpp
#include<bits/stdc++.h>
using namespace std;

long long countSpecialPaths (int N, vector<vector<int> > edges,
vector<int> A) {
    // Write your code here

}

int main() {

    ios::sync_with_stdio(0);
    cin.tie(0);
    int T;
    cin >> T;
    for(int t_i = 0; t_i < T; t_i++)
    {
        int N;
```

Microsoft →



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:

```
class Solution { public int solution(String 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:

Write a function:

```
class Solution { public int solution(String 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 consists only of lowercase letters (a−z).