Question - 1
SQL vs NoSQL data stores

Sometimes it is better to use a standard relational database based on SQL, MySQL for example. Other times, it is better to use data stores that do not rely on relations, commonly called NoSQL data stores. An example is MongoDB. Which of the following are true in general?

- SQL databases are easier to distribute than NoSQL databases.
- SQL databases have better support for complex queries than NoSQL databases.
- NoSQL databases generally have a simpler design than SQL databases.
- NoSQL databases support transactions better than SQL databases.

Question - 2
Practical database partitioning

Various strategies may be used to partition a database. There is a database of users, each having a first name, last name and a unique integer identifier. Nothing is known about the unique identifier except that it is a unique integer for each user. It is known that there will be future writes to the database. Which of the following is/are true?

- A good way to partition the database is to partition the users by the first letter of their names. There will be around 26 partitions, all of roughly equal size.
- A good way to partition the database is to partition the users by their identifiers in such a way that the user with identifier i ends up in partition i modulo m where m is the number of partitions needed. By doing that, all partitions will be roughly equal in size.
- A good way to partition the database is to partition the users by a hash of the first letter of their names, such that the hash is computed using a good hashing function that distributes the data uniformly. By doing that, there will be around 26 partitions, all of roughly equal size.
- A good way to partition the database is to partition the users by a hash of their identifiers. The hashing function chosen will distribute the data uniformly. The hashes are then used to assign partitions in such a way that the user with identifier hash h ends up in partition h modulo m where m is the number of partitions desired. By doing that, all partitions will be roughly equal

Question - 3

Database normalization

Database normalization is a common term for most types of databases. In general, which of the following are true?

Database normalization enforces better data integrity at the cost of greater storage space.

Database normalization enforces better data integrity at the cost of reduced performance in data retrieval.

Database normalization reduces used storage space.

Performing updates to the normalized data in the database is significantly easier and has better performance.

Question - 4

Product Sort

In a warehouse, a manager would like to sort the items in the stock. Given an array of *n* item values, sort the array in ascending order, first by the number of items with a certain value, then by the values themselves.

Example

n = 6

items = [4, 5, 6, 5, 4, 3]

- There are 2 values that occur twice: [4, 4, 5, 5].
- There are 2 values that occur once: [3, 6].
- The array of items sorted by quantity and then by value in ascending order is *[3, 6, 4, 4, 5, 5]*

Function Description

Complete the function *itemsSort* in the editor below.

itemsSort has the following parameter(s):

int items[n]: an array of integers to sort

Returns:

int[n]: An array of integers sorted as described.

Constraints

- $1 \le n \le 2 \times 10^5$
- $1 \le items[i] \le 10^6$

▼ Input Format for Custom Testing

Input from stdin will be processed as follows and passed to the function.

The first line contains an integer *n*, the size of the integer array *items*.

The next *n* lines each contain an integer, *items[i]*.

▼ Sample Case 0

Sample Input 0

```
STDIN         Function
-----         --------
5        →    items[] size n = 5
3        →    items  = [3, 1, 2, 2, 4]
1
2
2
4
```

Sample Output 0

```
1
3
4
2
2
```

Explanation

- There is a quantity of 2 for the item 2 : [2, 2]
- There is a quantity of 1 for the items 1, 3 and 4 : [1], [3], [4]
- The array of items sorted by quantity and then by value in ascending order is *[1, 3, 4, 2, 2]*

▼ Sample Case 1

Sample Input 1

```
STDIN     Function
-----     --------
10    →   items[] size n = 10
8     →   items = [8, 5, 5, 5, 5, 1, 1, 1, 4, 4]
5
5
5
5
1
1
1
4
4
```

Sample Output 1

```
8
4
4
1
1
1
5
5
5
5
```

Explanation

- There is a quantity of 4 for the item 5 :[5, 5, 5, 5]
- There is a quantity of 3 for the item 1 :[1, 1, 1]
- There is a quantity of 2 for the item 4 :[4, 4]
- There is a quantity of 1 for the items 8 : [8]
- The array of items sorted by quantity and then by value in ascending order is *[8, 4, 4, 1, 1, 1, 5, 5, 5, 5]*

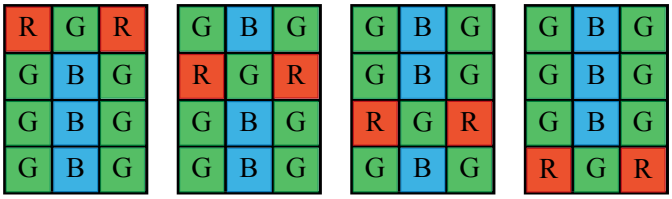
Question - 5
Coloring a Grid

An automated painting system needs a program that can paint an n x 3 grid in red, green, and blue such that no row or column contains cells that are all the same color. Determine the number of valid patterns that can be painted given *n* rows. Since the number of patterns can be large, return the value modulo (10^9+7).

Example

n = 4

Some examples of how the colors can be arranged for *4 x 3* grid are shown below. The total number of valid patterns is *296490* for a grid with 4 rows. Samples with 2 and 3 rows are given below.



Function Description

Complete the *countPatterns* function in the editor below.

countPatterns has only one parameter:

int n: the number of columns of the n *x 3* grid.

Return

int: the number of ways in which the gid can be colored, calculated as a modulo of (10^9+7)

Constraints

- $2 \leq n \leq 20000$

▼ Input Format For Custom Testing

The only line of input contains an integer, *n*, the number of rows of the *n x 3* grid.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
```

```
2      →   number of rows n = 2
```

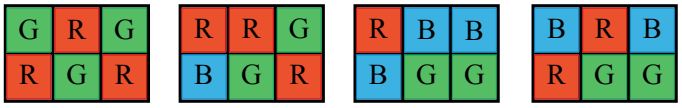
Sample Output

```
174
```

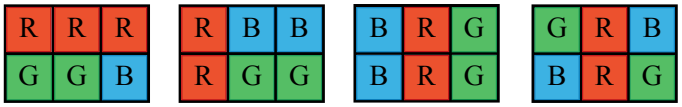
Explanation

Examples of valid patterns are shown below, as well as some invalid arrangements for comparison. There are 174 valid patterns.

Few Valid Arrangements



Few Invalid Arrangements



▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN      Function
-----      --------
3      →   number of rows n = 3
```

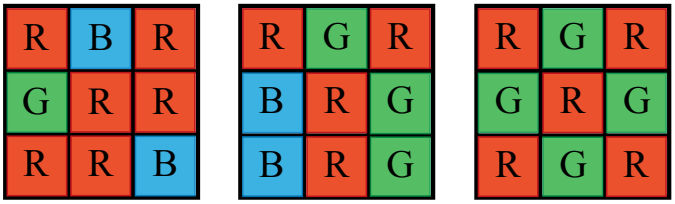
Sample Output

```
9750
```

Explanation

n = 3

Some valid ways to fill the grid:



Question - 6

String Reduction

Given a string, reduce it in such a way that all of its substrings are distinct. To do so, you may delete any character of the string at any index. What is the minimum number of deletions needed in order to complete this task?

Note: A substring is a contiguous sequence of characters within a string. It can be formed by deleting some (0 or more) characters from the left of the string and some (0 or more) characters from the right of the string.

For example, let's say the given string is *s = "abab"*. Currently, the substrings are not distinct—the substring "ab" is found starting at

both index 0 and index 2. By deleting s[2] and s[3], the string becomes "ab", where all substrings are distinct. Therefore, the answer is 2 because this required 2 deletions. (Note that "aba" is not acceptable because the character 'a' counts as a substring. In "aba", there are two instances of the substring "a".)

Function Description
Complete the function *getMinDeletions* in the editor below.

getMinDeletions has the following parameter(s):
  string *s:* the given string
Returns:
  int: the minimum number of deletions needed to make *s* have only distinct substrings in it

Constraints

- $1 \leq n \leq 10^5$

▼ Input Format For Custom Testing

The first line contains a string, *s*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN     Function
-----     --------
abcab  => s = "abcab"
```

Sample Output

```
2
```

Explanation
By deleting the first 2 characters, the string becomes "cab", which has only distinct substrings in it. Therefore, the answer is 2.

▼ Sample Case 1

Sample Input For Custom Testing

```
abcabc
```

Sample Output

```
3
```

Explanation
By deleting the characters at indices 0, 4, and 5, the string becomes "bca", which has only distinct substrings in it. Because this required 3 deletions, the answer is 3.

Question - 7
Bus Stand

There are *n* people standing in a queue for bus numbered from left to right, *1* to *n*. Each person has a patience limit, *p* and will only wait until the time *p* expires. If the bus reaches after time *p,* the person will leave the queue and miss the bus. Initially the bus is empty and has a fixed capacity, *k*. Given a number of queries *q,* where each query is a time of

bus arrival, *q[i],* for each query, print the index/number (1-indexed) of the k^{th} person who catches the bus. If all passengers remaining in the queue can board the bus, return *0* because there will be no k^{th} person.

For example, given a bus size *k = 2,* patience limits of *p = [1, 2, 3, 4]*, and queries at times *q = [1, 3, 4]*, there are three scenarios all dealing with the same initial queue. Where the bus arrives at *q[0] = 1*, all passengers are still queued but only the first two will fit on the bus. The last passenger who will fit is number *2*. If the bus arrives at *q[1] = 3,* passengers *1* and *2* have left the queue, the first two remaining *(3* and *4*) get on the bus, filling it to capacity. When *q[2] = 4*, passengers *1, 2* and *3* have left, so passenger *4* can get on. Since the bus is not filled, there is no k^{th} passenger . The returned array of answers is *[2, 4, 0]*.

Function Description

Complete the function *kthPerson* in the editor below. The function must return an array of integers where each integer *i* represents the results of a query, *q[i]*.

kthPerson has the following parameter(s):

k: an integer that represents the size of the bus

p[p[0],...p[n-1]]: an array of integers that represents the patience of *n* people from left to right

q[q[0],...q[j-1]]: an array of integers that represents the queries containing times *q[i]* of the arrival of the bus

Constraints

- *0 < n, k, q[i], p[i], q[j] ≤ 100000*

▼ Input Format For Custom Testing

The first line contains the capacity of the bus, *k*.
The second line contains the number of people in the queue, *n*.
The following *n* lines contain a single integer that represents the patience limit, *p* of each person.
The next line contains the number of queries, *j*.
The following *j* lines contain single integers that represents the time, *q[j]* of bus arrival.

▼ Sample Case 0

Sample Input For Custom Testing

```
3
3
2
5
3
2
1
5
```

Sample Output

```
3
0
```

Explanation

In the first query, the bus arrives at time *1*, so all three people can catch the bus and fill it to capacity. The last person who catches the bus is *3*.

In the second query, the bus arrives at time *5*. By this time first and the third person already leave the queue. So, only the second person can catch the bus. As the bus is not full, the answer is 0.

▼ Sample Case 1

Sample Input For Custom Testing

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

Sample Output

```
2
3
3
3
0
0
0
```

Explanation

In this case, *k = 2, p = [1, 4, 4, 3, 1, 2, 6] and q = [1, 2, 3, 4, 5, 6, 7]*.

In *q[0]*, the bus arrives at time *1*, so person number *2* will be the last person to catch the bus.

In *q[1]*, the bus arrives at time *2*, so person number *3* will be the last person to catch the bus as *k = 2* and the person number *1* will already have expired his patience by then.

In *q[2]*, the bus arrives at time *3*, so person number *3* will be the last person to catch the bus as *k = 2* and the person number *1* will already have expired his patience by then.

In *q[3]*, the bus arrives at time *4*, so person number *3* will be the last person to catch the bus as *k = 2* and the person number *1* will already have expired his patience by then.

In *q[4]*, the bus arrives at time *5*, everyone except the last person would have expired their patience. Hence, only *1* person boards the bus, it does not meet capacity so the answer is *0*.

And so on for *q[5]* and *q[6]*.

Question - 8
Palindrome Counter

A palindrome is a string that reads the same from the left and from the right. For example, *mom* and *tacocat* are palindromes, as are any single-character strings. Given a string, determine the number of its substrings that are palindromes.

Example
The string is *s = 'tacocat'*. Palindromic substrings are *['t', 'a', 'c', 'o', 'c', 'a', 't', 'coc', 'acoca', 'tacocat']*. There are *10* palindromic substrings.

Function Description
Complete the *countPalindromes* function in the editor.

countPalindromes has the following parameter:
string s: the string to analyze

Returns:
int: an integer that represents the number of palindromic substrings in the given string

Constraints

- $1 \leq |s| \leq 5 \times 10^3$
- *each character of s, s[i] ∈ {'a'-'z'}.*

▼ Input Format For Custom Testing

The first line contains a string, *s*.

▼ Sample Case 0

Sample Input

```
STDIN     Function
-----     -----
aaa    →  s = 'aaa'
```

Sample Output

```
6
```

Explanation
There are *6* possible substrings of *s*: *{'a', 'a', 'a', 'aa', 'aa', 'aaa'}*. All of them are palindromes, so return *6*.

▼ Sample Case 1

Sample Input

```
STDIN      Function
-----      -----
abccba  →  s = 'abccba'
```

Sample Output

```
9
```

Explanation
There are *21* possible substrings of *s*, the following *9* of which are palindromes: *{'a', 'a', 'b', 'b', 'c', 'c', 'cc', 'bccb', 'abccba'}*.

▼ Sample Case 2

Sample Input

```
STDIN     Function
-----     -----
daata  →  s = 'daata'
```

Sample Output

```
7
```

Explanation

There are *15* possible substrings of *s*, the following *7* of which are palindromes: *{'a', 'a', 'a', 'aa', 'ata', 'd', 't'}*.

Question - 9

Fun With Vowels

A *subsequence* is a sequence of letters in a string, in order, but with any number of characters removed. For example, *3* letter subsequences of *abcd* are *abc, abd, acd,* and *bcd. Vowels* in order are the letters in the string *aeiou*. Given a string, determine the length of the longest subsequence that contains all of the vowels, in order, and no vowels out of order.

For example, the string *aeeiiouu* contains all vowels in order. The string *aeeiiaouu* does not because of the *a* at position *5,* (*0* based indexing). The first string is the longest subsequence of the second string that contains all vowels in order.

Function Description

Complete the function *longestVowelSubsequence* in the editor below. The function must return the length of the longest subsequence within the input string that contains all of the vowels in order. If one does not occur in the string, return *0*

longestVowelSubsequence has the following parameter(s):

s: the string to analyze

Constraints

- $5 < |s| < 5 \times 10^5$
- Each character of string $s \in \{a, e, i, o, u\}$.

▼ Input Format for Custom Testing

Input from stdin will be processed as follows and passed to the function.

There is only one line containing the string *s*.

▼ Sample Case 0

Sample Input 0

```
STDIN                Function Parameters
-----                -------------------
aeiaaioooaauuaeiu →  s = "aeiaaioooaauuaeiu"
```

Sample Output 0

```
10
```

Explanation 0

In the table below, the characters making the longest vowel subsequence are in red:

a	e	i	a	a	i	o	o	o	a	a	u	u	a	e	i	u

▼ Sample Case 1

Sample Input 1

```
STDIN          Function Parameters
-----          -------------------
aeiaaiooaa →   s = "aeiaaiooaa"
```

Sample Output 1

```
0
```

Explanation 1

The string *s* does not contain all of the vowels.