Rules :

- Don’t use zip
- File format name : Name_No.extension (sample: Naufal_1.java, Naufal_2.sql, Naufal_3.js)

1. **ALGORITHM**

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

Note: A *substring* is a contiguous group of 1 or more characters within a string.

Example

$s = "abab"$

Substrings in s are { 'a', 'b', 'a', 'b', 'ab', 'ba', 'ab', 'aba', 'bab', 'abab'}. By deleting one "a" and one "b", the string becomes "ab" or "ba" and all of its substrings are distinct. This required 2 deletions.

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 required

Constraints

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

2. **ALGORITM**

In a game, there is a string, *direction,* of length n that consists of characters L and R. L denotes left, R denotes right, and there is a line segment of length 2^n that extends from $[0, 2^n]$, a player takes n turns.

In the i^{th} turn,

- The player places the number i at the center of the current line segment.
- If *direction[i] = 'L'*, the player proceeds in the left direction, eliminating the right half of the current line segment, and vice versa for *direction[i] = 'R'*.

Following this rule, find the final order of numbers on the line segment, starting from the left.

Example

direction = "LRRLLL"

There are $n = 6$ characters in the initial length of the line segment. The segment length is $2^n = 2^6 = 64$ in the range [0, 64].

The game proceeds as follows:

Center	Number	Direction	Remaining Segment
			[0, 64]
32	1	L	[0, 32]
16	2	R	[16, 32]
24	3	R	[24, 32]
28	4	L	[24, 28]
26	5	L	[24, 26]
25	6	L	[24, 25]

The smallest center point is 16 and the value placed is 2. The next smaller center is 24 with a value of 3. When the centers are ordered ascending, their values are [2, 3, 6, 5, 4, 1].

Function Description

Complete the function *findNumberSequence* in the editor below.

findNumberSequence has the following parameter:

string direction: a string of length n where each character denotes the direction of a turn

Returns

int[n] : an integer array that denotes the indices of characters in the string ordered by placement point

Constraints

- $1 \leq n \leq 10^5$
- The string *direction* consists of 'L' or 'R' only.

3. **ALGORITHM**

Given a list of names, determine the number of names in that list for which a given query string is a prefix. The prefix must be at least 1 character less than the entire name string.

Example

names = ['jackson', 'jacques', 'jack']
query = ['jack']

The complete *query* string '*jack*' is a prefix of *jackson* but not of *jacques* or *jack*. The prefix cannot contain the entire *name* string, so *'jack'* does not qualify.

Function Description

Complete the function *findCompletePrefixes* in the editor below. The function must return an array of integers that each denotes the number of *names* strings for which a *query* string is a prefix.

findCompletePrefixes has the following parameter(s):

string names[n]: an array of name strings
string query[q]: an array of query strings

Returns:

int[q]: each *value[i]* is the answer to *query[i]*

Constraints

- *1 ≤ n ≤ 20000*
- *2 ≤ length of names[i], query[i] ≤ 30,*
- *1 ≤ sum of the lengths of all names[i] ≤ 5 x 10^5*
- *1 ≤ q ≤ 200*

4. **SQL**

The locations of the superheroes have been stored in the *SUPERHERO* table. Write a query to print the *ID*s, i.e., *SUPERHERO.ID* of the superheroes whose latitudes and longitudes both have a value smaller than *50*. The order of output does not matter.

Input Format

SUPERHERO		
Name	**Type**	**Description**
ID	Integer	A superhero ID in the inclusive range *[1, 1000]*. This field is the primary key.
NAME	String	A superhero name. This field contains between *1* and *100* characters (inclusive).
LATITUDE	Float	The latitude of the superhero.
LONGITUDE	Float	The longitude of the superhero.

Output Format

The result should contain the *ID*s of the superheroes whose latitudes and longitudes both have a value smaller than *50*.

```
SUPERHERO.ID
```

Sample Input

SUPERHERO			
ID	**NAME**	**LATITUDE**	**LONGITUDE**
1	Batman	50.23	85.45
2	Spiderman	65.43	65.66
3	Thor	45.34	30.89
4	Iron Man	85.34	80.98

5	Deadpool	25.12	69.21	
6	Hulk	30.34	20.98	
7	Doctor Strange	40.45	40.56	
8	Captain America	70.00	75.32	
9	Avengers	81.32	90.84	
10	Superman	85.32	45.78	

Sample Output

```
3
6
7
```

Explanation

- For Batman, latitude = 50.23 and longitude = 85.45. So, neither is less than 50.
- For Thor, latitude = 45.34 and longitude = 30.89. So, both are less than 50.
- For Deadpool, latitude = 25.12 and longitude = 69.21. So, the latitude is less than 50 but the longitude is greater than 50.
- The remaining records are analyzed similarly.

Thor, *Hulk*, and *Doctor Strange* have latitudes and longitudes less than *50* so, their *ID*s are printed.

5. SQL

Company X has a record of its customers and their orders. Find the customer(s) with the highest order price for orders placed within 10 years of the first order (earliest order_date) in the database. Print the customer name and order price. If multiple records are returned, they can be in any order.

CUSTOMERS		
Name	Type	Description
ID	STRING	ID of the customer. This is the primary key.
NAME	STRING	Name of the customer.
ORDER_ID	STRING	ID of the customer's order.

ORDERS		
Name	Type	Description
ID	STRING	ID of the order.
PRICE	INTEGER	Price of the order.
ORDER_DATE	DATE	Date of the order.