Optimal Towers

A graph representing terrain has been provided. Radio towers need to be placed in an optimal manner on peaks such that a message can reach the end of the terrain.

A tower can be of any height and can only be placed on the top of any peak.

A peak is defined as a point that is greater than both is adjacent points if they exist.

Given that a signal is sent from the left side of the graph towards the right. The signal can only move horizontally towards the right. Any signal can only propagate **k** units without needing a tower to boost the signal again, find the minimum number of towers required for a signal that originates on the left side of the graph to reach the right side. You are given the height values of the train from left to right in form of an array. Note are considered as been weeks not hours

Input Format

The first line contains an integer, k, denoting the propagation limit of the signal.

The next line contains an integer, m, denoting the width of the terrain in the number of units.

Each line i of the m subsequent lines (where 0 <= i <= m) contains the height of the terrain at the i^{th} point in the graph.

Output Format

Print a single integer denoting the minimum number of towers required.

Constraints

- 1 <= k <= 10^5
- 1 <= m <= 10^5
- 1<= height of any terrain <+ 10^5

Cut the pipes for the best flow

A specialized pipe needs to be cut into x pieces.

The pipe is special as every unit length of this pipe has a different flow rate. For example, from unit length 0 to 1, the pipe may have a flow rate of 10 units/second and a flow rate of 20 units/second from 1 to 2 and so on.

It is given that the overall flow rate of fluid flowing through any pipe is equal to the lowest flow rate of any of its unit lengths

This pipe needs to be cut into x pieces such that the sum of the resultant flow rates of each of these new pipe segments is maximum. What is the maximum sum of the flow rates of the new pipe segments?

[Note: There is no wastage when cutting the pipes, i.e sum of the lengths of the x segments is equal to the length of the original pipe. x is always less than or equal to the length of the pipe. Cut can be made only at integral lengths, and the minimum length of a piece is 1]

① Dangerous Script

If you've ever looked at an assembly language dump of an executable file, you know that commands come in the form of hexadecimal digits that are grouped by the processor into instructions. It is important that the parsing be done correctly or code will not be executed as expected. Wrong parsing is the basis for Return Oriented programming, method of causing mayhem.
You have developed a program in a new scripting language. Of course, it requires accurate parsing in order to perform as expected, and it is very cryptic. You want to determine how many valid commands will be in the following format:

- The first letter is a lowercase English letter.
- Next, it contains a sequence of zero or more of the following characters: lowercase English letters, digits, and colons.
- Next, it contains a forward slash '/'.
- Next, it contains a sequence of one or more of the following characters: Lowercase English letters and digits.
- Next, it contains a backward slash '\'.
- Next, it contains a sequence of one or more lowercase English letters.

Given some string, s, we define the following:

- S[i..j] is a substring consisting of all the characters in the inclusive range between index i and index j.
- Two substrings, s[i[1]..j[1]] and s[i[2]..j[2]], are said to be distinct if either i[1] != i[2] or j[1] != j[2].

For example, your command line is abc:/b1c\xy. Valid command substrings are:

```
abc:/b1c\xy
bc:/b1c\xy
c:/b1c\xy
abc:b1c\x
bc:/b1c\x
c:/b1c\x
```

There are six valid commands that may be parsed from that one commands string.

Function description
Complete the function commandCount. The function must return an array of integers, each representing the number of valid command string in command[i].

commandCount has the following parameter(s):
commands[command[0]...commands[n-1]]: an array of strings.

Constraints

- 1<= n <=50
- Each character of commands[i] Є [a-z,0-9,/,\,:]
- Each |command[i]| <= 5 x 10^5

Input format for custom testing
Input from stdin processed as follows and passed to the function.
The first line contains an integer n, the size of the array commands.
Each of the next n lines contains a string commands[i].

Sample case 0

```
6
w\\//a/b
w\\//a\b
w\\/a\b
w::/a\b
w::/a\b
w:/a\bc::/12\xyz
```

Sample output 0

```
0
0
0
0
1
```

2 Longest well performing period

An old fashioned company awards workers for efficiency simply based on number of hours worked. They assume that the worker Performed well on some particular day if he worked more than x hours that day. After they find the longest well-performing period of that worker, which is the longest period at which the number of well-performed days is larger than not well-performed ones.The reward goes to the worker who has the largest " longest well-performing period".

As this task is quite complex and requires a lot of time, you are required write a program to compute the length of " longest well-performing period" for a particular worker.

Given n=5 working days with hours[4,7,5,8,3] and x=5, the " longest well-performing period" is [7,5,8]. In this period the number of well-formed days is larger than not well-performed ones as 7>5 and 8>5. Since there is no longer well-performing period the answer to this example is 3.

Function description
Complete the function findLongest. The function must return the length of the "longest well-performing period" for a particular worker.
findLongest has the following parameter(s):
    Hours[hours[0]...hours[n-1]]: Array of integers
    X: an Integer

Constraints

- 1 <= n <= 10^5
- 0 <= hours[i] <= 24 where 0 <= i <= n-1
- 0 <= x <= 24

Sample Case 0

```
4
5
10
8
9
8
```

Sample Output
3
We have 4 days with hours[5,10,8,9] and x=8. In this example, the "most well-performing period" is [10,9,8]. In this period the number of well-performed days is larger than not well-performed ones as 10>8 and 9>8. Since there is no longer a well-performing period the answer to this example is 3.

3

Math Homework

A teacher gives a student a few math problems to complete. After solving the ith problem the student must then complete either then next problem i.e. (i+1)th, or skip that one and solve the (i+2)th problem. Each problem has happiness points associated and they are sorted ascending. A student needs to solve problems until the difference between maximum and minimum point solved question reaches or exceed threshold. If this cannot be done, must solve all of the the problems. Determine the maximum number of problems the student can solve.

For example, if threshold = 4 and questions have happiness points happy=[1,2,3,5],the student can solve happy[0]=1, happy[2]=3 and happy[3]=5 because 5 -1=4, the threshold. if threshold>4, the student would have to solve all the problems as there is no way to meet the threshold value. in this case, the student must solve three problems: 0,2, and 3, so the return value 3.

Function description
Complete the function minNum. The function must return an integer denoting the minimum number of problems the student must solve to satisfy the teacher's requirements.

minNum has the following parameters:
Threshold: an Integer
Happy[happy[0]...happy[n-1]]: an array of Integers

4

Arbitrary shopping

An avid shopper goes to a clothing store and picks any arbitrary outfit. Later the shopkeeper buys all consecutive outfit picked up as long as there is money to pay for them up to the nth outfit. For example after first selecting outfit 'i' the shopper will continue to outfit i+1, i+2 and so on until there is not enough money for another outfit. Determine the maximum number of outfits the shopper can buy.

For example, assume outfits prices are outfits = [2,3,5,1,1,2,1]And the money available, money=5. There are three subarrays of prices that sum to less than or equal to money: [2,3],[5],[1,1,2,1]. The longest of these, that is the maximum number of outfits that can be bought is 4.

Function description

Complete the function getMaximumOutfits. The function must return an integer that denotes the maximum number of outfits that can be bought.
getMaximumOutfits The following parameters:
money: an integer, which denotes the amount of money.
outfits: an array of integers, which denotes the outfits.

Constraints

- 1 <= n <= 10^5
- 1 <= outfits[i] <= 100
- 1 <= money <= 10^6

Sample Input for Custom Testing

```
3
10
10
10
5
```

Sample Output

```
0
```

Explanation

There are 3 outfits each costing 10. With Money = 5 , there is not enough money to buy any of the outfit

5

Tag Identification Number

The tag identification number of every employee in a company has length 11 and fits the pattern "8xxxxxxxxxx", where each X is replaced by a digit. For example: "80123456789" and "80000000000" are TIDs, while "8012345678" and "79000000000" are not. The automatic machine that generates these TIDs has malfunctioned and now has a pool of n digits. Find out the maximum number of TIDs that can be formed from this pool of digits simultaneously. If at least one TID can be made from these cards, output the maximum number of words that can be made. Otherwise, output zero. Note that TIDs do not have to be unique. For example: "8111111111181111111111" can make two TIDs each being "81111111111"

As an example, given a pool of digits pool=8012345678981234567899, there are 22 digits available. It is possible to make 2 TIDs simultaneously from this many digits. Two examples are 80123456789 and 81234567899. They use all digits in the pool one time.

Function Description
Complete the function numOftds. It must return an integer that represents the total number of TIDs that can be generated from the pool of digits.

numOftds has the following parameter(s):
pool: a string of n digits

Constraints
1 <= length of pool <=100

6

Slowest Key Press

Alex wants to be a faster typist and is taking a typing test to find out which key takes longest time to press. Given the results of the test, determine which key takes the longest to press. For example, given keyTimes =[[0,2],[1,5],[0,9],[2,15]], interpret each keyTimes[i][0] as a encoded character in the range ascii[a-z] where a=0, b = 1,...z = 25. The second element, keyTimes[i][1] represents the time the key is pressed since the start of the test. In the example, keys pressed, in order are abac at times 2,5,9,15.
From the start time, it took 2-0 =0 to press the first key, 5-2 = 3 to press the second, and so on. The longest time it took to press a key was key 2 or 'c', at 15-9 = 6.

Function Description
Complete the function slowestKey. The function must return a character, the slowest key that Alex presses.
slowestKey has the following parameter(s):
keyTimes[keyTimes[0]...keyTimes[n-1]]: an array of two integers: the character Alex pressed(encoded) and the time at which it was pressed.

Constraints

- 1 <=n <= 10^5
- 0 <= keyTimes[i][0] <= 25 (0 <= i <n)
- 1 <=keyTimes[i][1] <= 10^8 (0 <= i < n)
- It is guaranteed that there will only be one key with the worst time.

Sample Input for Custom Testing

```
3
2
0  2
1  3
0  7
```

Sample Output

```
a
```

Explanation
The time taken to press 'a' in the worst case is 7-3=4. To press worst case is 3-2=1. Alex performed worst on 'a'.