Contest 2 : CX

A. Cereal Segments

1 second, 256 megabytes

Problem Description

You are at the grocery store trying to determine which cereal to buy. There is a row of different cereal brands in front of you, represented as an n-element array.

You would like to find a cereal that is not too pricey but also not the most inexpensive. To do this, you scan every subarray of size x, starting from index 0. Subarray are to be contiguous blocks of elements in the inclusive range from index i to index j, where $j - i + 1 = x$ and $0 \leq i \leq j < n$. Then, among each of these subarrays, find the cereal with the minimum price. Return the value of the maximum among these.

For example, given array $arr = [8, 2, 4]$, the subarrays of size $x = 2$ would be $[8, 2]$ and $[2, 4]$. The minimum values of the two subarrays are $[2, 2]$. The maximum of those two minimum values is 2. This is the value you need to determine, which represents the price of the cereal you decide to purchase.

Constraints

- $1 \leq n \leq 10^6$
- $1 \leq arr[i] \leq 10^9$
- $1 \leq x \leq n$

Cereal Segments

You are at the grocery store trying to determine which cereal to buy. There is a row of different cereal brands in front of you, represented as an n-element array.

You would like to find a cereal that is not too pricey but also not the most inexpensive. To do this, you scan every subarray of size x, starting from index 0. Subarrays are to be contiguous blocks of elements in the inclusive range from index i to index j, where j - i + 1 = x and 0 ≤ i ≤ j < n. Then, among each of these subarrays, find the cereal with the minimum price. Return the value of the maximum among these.

For example, given array arr=[8, 2, 4], the subarrays of size x = 2 would be [8, 2] and [2, 4]. The minimum values of the two subarrays are [2, 2]. The maximum of those two minimum values is 2. This is the value you need to determine, which represents the price of the cereal you decide to purchase.

Function Description

Complete the function segment in the editor below. Your function must find the minimum value for each subarray of size x in array arr, then return an integer denoting the maximum of these minima.

segment has the following parameter(s):
x: an integer, the segment length
arr[arr[0],...arr[n-1]]: an array of integers

Constraints
- 1 ≤ n ≤ 10^6
- 1 ≤ arr[i] ≤ 10^9
- 1 ≤ x ≤ n

Input Format for Custom Testing

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

The first line contains an integer, x.
The second line contains an integer, n, which is the size of the array arr.
Each of the next n lines contains the integer arr[i].

Sample Case 0

Sample Input 0

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

Sample Output 1

```
3
```

Explanation 1
The following arguments are passed to your function:
x = 1
arr[5] = [1, 2, 3, 1, 2]

Sample Output 1

```
3
```

Explanation 1
The following arguments are passed to your function:
x = 1
arr[5] = [1, 2, 3, 1, 2]

The subarrays of size x = 1 are [1], [2], [3], [1], and [2]. Because each subarray only contains 1 element, each value is minimal with respect to the subarray it's in. We return the maximum of these values, which is 3.

Sample Case 1

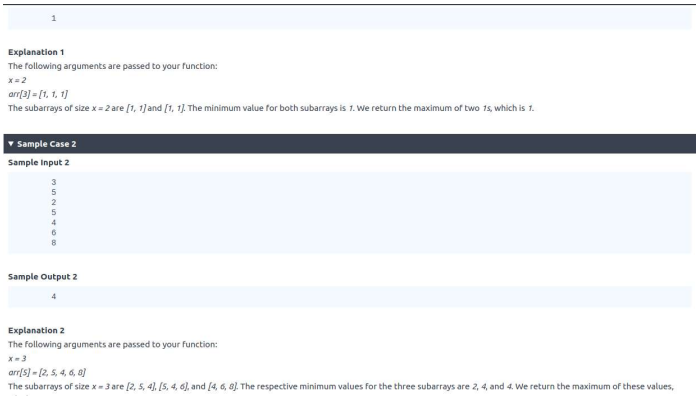
Sample Input 1

```
2
3
1
1
1
```

Sample Output 1

```
1
```

Explanation 1
The following arguments are passed to your function:
x = 2
arr[3] = [1, 1, 1]



```
1
```

Explanation 1
The following arguments are passed to your function:
x = 2
arr[3] = [1, 1, 1]
The subarrays of size x = 2 are [1, 1] and [1, 1]. The minimum value for both subarrays is 1. We return the maximum of two 1s, which is 1.

Sample Case 2

Sample Input 2

```
3
5
2
5
4
6
8
```

Sample Output 2

```
4
```

Explanation 2
The following arguments are passed to your function:
x = 3
arr[5] = [2, 5, 4, 6, 8]
The subarrays of size x = 3 are [2, 5, 4], [5, 4, 6], and [4, 6, 8]. The respective minimum values for the three subarrays are 2, 4, and 4. We return the maximum of these values,

Input

The first line contains an integer , x .

The second line contains an integer n , which is the size of array arr .

Each of the next n lines contains the integer $arr[i]$

Output

Find the minimum value for each subarray of size x in array arr, then print the maximum of these minimas.

input
1 5 1 2 3 1 2
output
3

input
2 3 1 1 1
output
1

In the first sample test case , $x = 1$, and $size\ of\ array(n) = 5, arr[5] = [1, 2, 3, 1, 2]$ .The subarrays of size $x = 1$ are $[1], [2], [3], [1], [2]$.Because each subarray only contains 1 element , each value is minimal with respect to the subarray it's in.Hence , we return the maximum of these values, which is 3.

In the second sample test case, $x = 2$, and $size\ of\ array(n) = 3, arr[3] = [1, 1, 1]$ .The subarrays of size $x = 2$ are $[1, 1], [1, 1]$.The minimum value for both subarrays is 1.Hence, we return the maximum of two $1s$, which is 1.

B. Good Binary Strings

1 second, 256 megabytes

Problem Description

We define the following :

- A **binary string** is a string consisting of 0's or 1's. For example, 01011, 1111 and 00 are all binary strings.
- The **prefix** of a string is any substring of the string that includes the beginning of the string. For example, the prefixes of 11010 are $1, 11, 110, 1101$ and 11010.

We consider a non-empty binary string to be $good$ if the following two conditions are true:

- The number of 0's is equal to the number of 1's.
- For every prefix of the binary string, the number of 1's should not be less than the number of 0's.

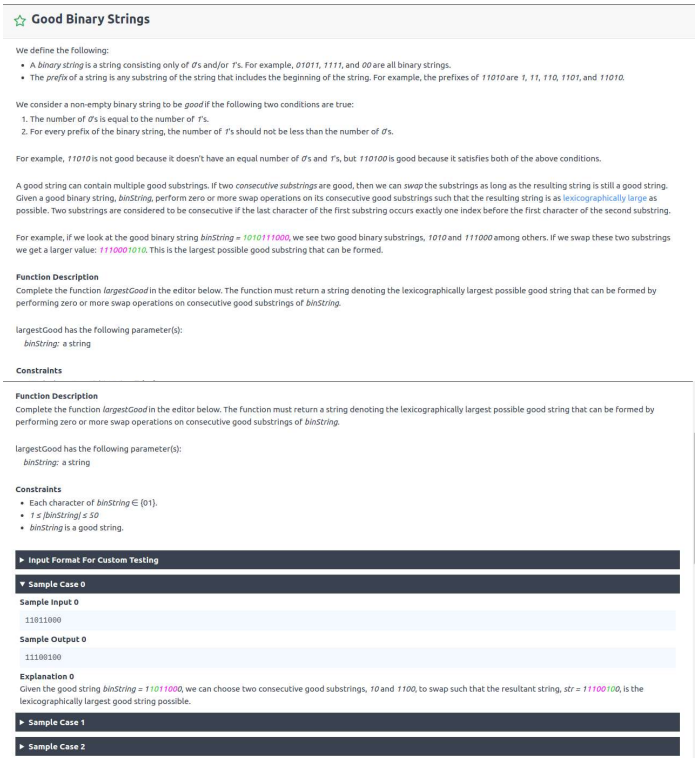
For example, 11010 is not good because it doesn't have an equal number of 0's and 1's, but 110100 is good because it satisfies both of the above conditions.

A good string can contain multiple good substrings. If two $consecutive\ substrings$ are good, then we can swap the substrings as longs as the resulting string is still a good string. Given a good binary string, $binString$, perform zero or more swap operations on its consecutive good substrings such that the resulting string is as **lexicographically large** as possible. Two substrings are considered to be consecutive if the last character of the first substring occurs exactly one index before the first character of the second substring.

For example, if we look at the good binary string $binString = 1010111000$, we see two good binary substrings, 1010 and 111000 among others. If we swap these two substrings we get a larger value: 1110001010. This is the largest possible good substring that can be formed.

Constraints

- Each character of $binString \in \{0\ ,\ 1\}$
- $1 \leq |binString| \leq 50$
- $binString$ is a good string.


Good Binary Strings

We define the following:
- A binary string is a string consisting only of 0's and/or 1's. For example, 01011, 1111, and 00 are all binary strings.
- The prefix of a string is any substring of the string that includes the beginning of the string. For example, the prefixes of 11010 are 1, 11, 110, 1101, and 11010.

We consider a non-empty binary string to be good if the following two conditions are true:
1. The number of 0's is equal to the number of 1's.
2. For every prefix of the binary string, the number of 1's should not be less than the number of 0's.

For example, 11010 is not good because it doesn't have an equal number of 0's and 1's, but 110100 is good because it satisfies both of the above conditions.

A good string can contain multiple good substrings. If two consecutive substrings are good, then we can swap the substrings as long as the resulting string is still a good string. Given a good binary string, binString, perform zero or more swap operations on its consecutive good substrings such that the resulting string is as lexicographically large as possible. Two substrings are considered to be consecutive if the last character of the first substring occurs exactly one index before the first character of the second substring.

For example, if we look at the good binary string binString = 1010111000, we see two good binary substrings, 1010 and 111000 among others. If we swap these two substrings we get a larger value: 1110001010. This is the largest possible good substring that can be formed.

Function Description
Complete the function largestGood in the editor below. The function must return a string denoting the lexicographically largest possible good string that can be formed by performing zero or more swap operations on consecutive good substrings of binString.

largestGood has the following parameter(s):
binString: a string

Constraints

Function Description
Complete the function largestGood in the editor below. The function must return a string denoting the lexicographically largest possible good string that can be formed by performing zero or more swap operations on consecutive good substrings of binString.

largestGood has the following parameter(s):
binString: a string

Constraints
- Each character of binString ∈ {0,1}.
- 1 ≤ |binString| ≤ 50
- binString is a good string.

▸ Input Format For Custom Testing
▾ Sample Case 0
Sample Input 0
11011000
Sample Output 0
11100100
Explanation 0
Given the good string binString = 11011000, we can choose two consecutive good substrings, 10 and 1100, to swap such that the resultant string, str = 11100100, is the lexicographically largest good string possible.
▸ Sample Case 1
▸ Sample Case 2

Input
Given a good binary string $binString$.

Output
Print the lexicographically largest possible good string that can be formed by performing zero or more swap operations on consecutive good substrings of $binString$.

input
11011000
output
11100100

In first sample test case, given the good binary string $binString = 11011000$, we can choose two consecutive good substrings, 10 and 1100, to swap such that the resultant string $str = 11100100$, is the lexicographically largest good string possible.

C. SGIPC

1 second, 256 megabytes

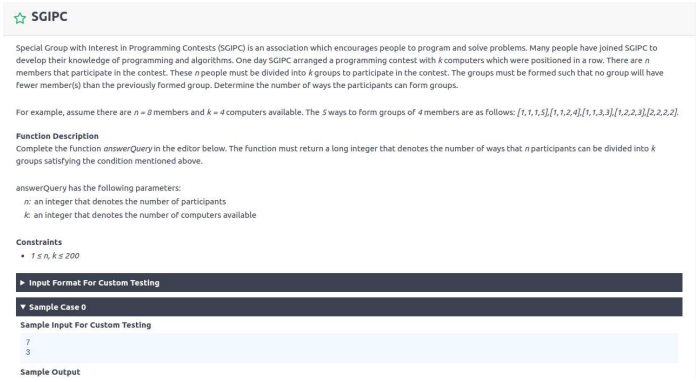
Problem Description

Special Group with interest in Programming Contest(SGIPC) is an association which encourages people to program and solve problems. Many people have joined SGIPC to develop their knowledge of programming and algorithms.One day SGIPC arranged a programming contest with k computers which were positioned in a row. There are n members that participate in the contest. These n people must be divided into k groups to participate in the contest. The groups must be formed such that no group will have fewer member(s) than the previously formed group. Determine the number of ways the participants can form groups. **Output the answer modulo** $10^9 + 7$

For example, assume there are $n = 8$ members and $k = 4$ computers available. The 5 ways to form groups of 4 members are as follows : $[1, 1, 1, 5], [1, 1, 2, 4], [1, 1, 3, 3], [1, 2, 2, 3], [2, 2, 2, 2]$

Constraints

- $1 \leq n, k \leq 200$


SGIPC

Special Group with interest in Programming Contests (SGIPC) is an association which encourages people to program and solve problems. Many people have joined SGIPC to develop their knowledge of programming and algorithms. One day SGIPC arranged a programming contest with k computers which were positioned in a row. There are n members that participate in the contest. These n people must be divided into k groups to participate in the contest. The groups must be formed such that no group will have fewer member(s) than the previously formed group. Determine the number of ways the participants can form groups.

For example, assume there are n = 8 members and k = 4 computers available. The 5 ways to form groups of 4 members are as follows: [1,1,1,5],[1,1,2,4],[1,1,3,3],[1,2,2,3],[2,2,2,2].

Function Description
Complete the function answerQuery in the editor below. The function must return a long integer that denotes the number of ways that n participants can be divided into k groups satisfying the condition mentioned above.

answerQuery has the following parameters:
n: an integer that denotes the number of participants
k: an integer that denotes the number of computers available

Constraints
- 1 ≤ n, k ≤ 200

▸ Input Format For Custom Testing
▾ Sample Case 0
Sample Input For Custom Testing
7
3
Sample Output

Input
The first line contains an integer n denoting the number of participants .

The second line contain an integer k denoting the number of computers available .

Output
You should print an integer that denotes the number of ways that n participants can be divided into k groups satisfying the condition mentioned above . **Output the answer modulo** $10^9 + 7$

input
7 3
output
4

input
8 4
output
5

D. SGIPC (Hard)

1 second, 256 megabytes

Note: The only difference from the easy version is the constraints.

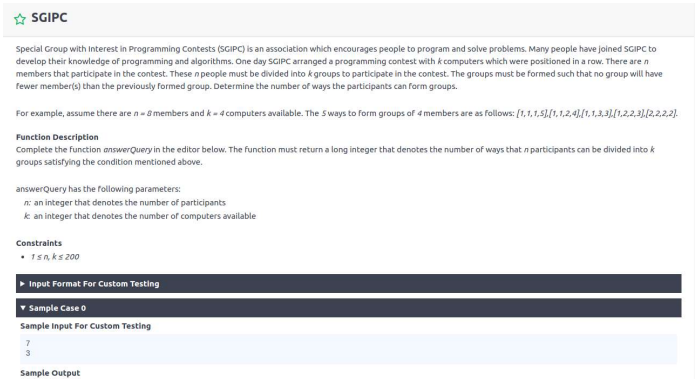
Problem Description

Special Group with interest in Programming Contest(SGIPC) is an association which encourages people to program and solve problems. Many people have joined SGIPC to develop their knowledge of programming and algorithms.One day SGIPC arranged a programming contest with k computers which were positioned in a row. There are n members that participate in the contest. These n people must be divided into k groups to participate in the contest. The groups must be formed such that no group will have fewer member(s) than the previously formed group. Determine the number of ways the participants can form groups. **Output the answer modulo** $10^9 + 7$

For example, assume there are $n = 8$ members and $k = 4$ computers available. The 5 ways to form groups of 4 members are as follows : $[1, 1, 1, 5], [1, 1, 2, 4], [1, 1, 3, 3], [1, 2, 2, 3], [2, 2, 2, 2]$

Constraints

- $1 \leq n, k \leq 4000$


SGIPC

Special Group with interest in Programming Contests (SGIPC) is an association which encourages people to program and solve problems. Many people have joined SGIPC to develop their knowledge of programming and algorithms. One day SGIPC arranged a programming contest with k computers which were positioned in a row. There are n members that participate in the contest. These n people must be divided into k groups to participate in the contest. The groups must be formed such that no group will have fewer member(s) than the previously formed group. Determine the number of ways the participants can form groups.

For example, assume there are n = 8 members and k = 4 computers available. The 5 ways to form groups of 4 members are as follows: [1,1,1,5],[1,1,2,4],[1,1,3,3],[1,2,2,3],[2,2,2,2].

Function Description
Complete the function answerQuery in the editor below. The function must return a long integer that denotes the number of ways that n participants can be divided into k groups satisfying the condition mentioned above.

answerQuery has the following parameters:
n: an integer that denotes the number of participants
k: an integer that denotes the number of computers available

Constraints
- 1 ≤ n, k ≤ 200

▸ Input Format For Custom Testing
▾ Sample Case 0
Sample Input For Custom Testing
7
3
Sample Output

Input

The first line contains an integer n denoting the number of participants .

The second line contain an integer k denoting the number of computers available .

Output

You should print an integer that denotes the number of ways that n participants can be divided into k groups satisfying the condition mentioned above . **Output the answer modulo** $10^9 + 7$

input
7 3
output
4

input
8 4
output
5

E. Good Binary Strings (Hard)

1 second, 256 megabytes

Note : The only difference from the easy version is the constraints.

Problem Description

We define the following :

- A **binary string** is a string consisting of 0's or 1's. For example, 01011, 1111 and 00 are all binary strings.
- The **prefix** of a string is any substring of the string that includes the beginning of the string. For example, the prefixes of 11010 are $1, 11, 110, 1101$ and 11010.

We consider a non-empty binary string to be $good$ if the following two conditions are true:

- The number of 0's is equal to the number of 1's.
- For every prefix of the binary string, the number of 1's should not be less than the number of 0's.

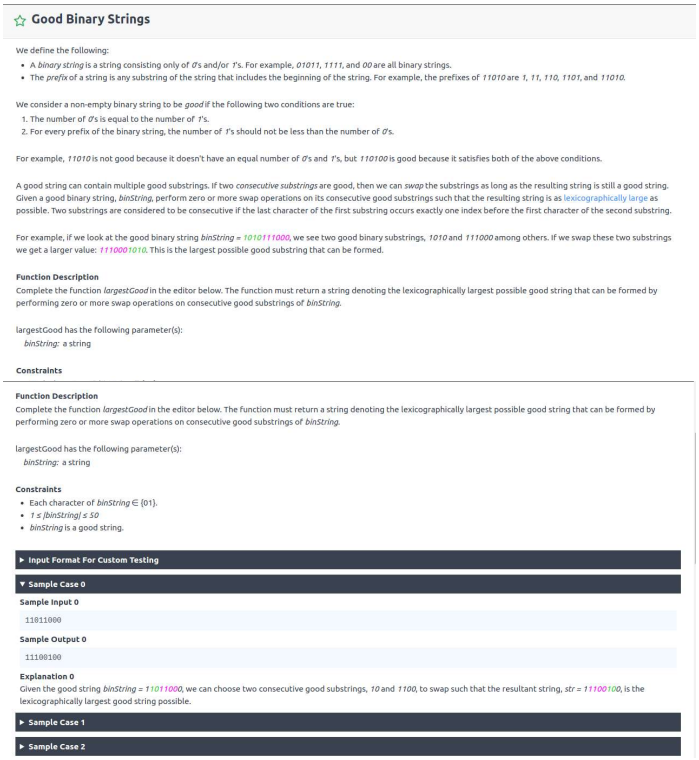
For example, 11010 is not good because it doesn't have an equal number of 0's and 1's, but 110100 is good because it satisfies both of the above conditions.

A good string can contain multiple good substrings. If two $consecutive\ substrings$ are good, then we can swap the substrings as longs as the resulting string is still a good string. Given a good binary string, $binSring$, perform zero or more swap operations on its consecutive good substrings such that the resulting string is as **lexicographically large** as possible. Two substrings are considered to be consecutive if the last character of the first substring occurs exactly one index before the first character of the second substring.

For example, if we look at the good binary string $binString = 1010111000$, we see two good binary substrings, 1010 and 111000 among others. If we swap these two substrings we get a larger value: 1110001010. This is the largest possible good substring that can be formed.

Constraints

- Each character of $binString \in$ {0 , 1}
- $1 \leq |binString| \leq 4000$
- $binString$ is a good string.


Good Binary Strings

We define the following:
- A binary string is a string consisting only of 0's and/or 1's. For example, 01011, 1111, and 00 are all binary strings.
- The prefix of a string is any substring of the string that includes the beginning of the string. For example, the prefixes of 11010 are 1, 11, 110, 1101, and 11010.

We consider a non-empty binary string to be good if the following two conditions are true:
1. The number of 0's is equal to the number of 1's.
2. For every prefix of the binary string, the number of 1's should not be less than the number of 0's.

For example, 11010 is not good because it doesn't have an equal number of 0's and 1's, but 110100 is good because it satisfies both of the above conditions.

A good string can contain multiple good substrings. If two consecutive substrings are good, then we can swap the substrings as long as the resulting string is still a good string. Given a good binary string, binString, perform zero or more swap operations on its consecutive good substrings such that the resulting string is as lexicographically large as possible. Two substrings are considered to be consecutive if the last character of the first substring occurs exactly one index before the first character of the second substring.

For example, if we look at the good binary string binString = 1010111000, we see two good binary substrings, 1010 and 111000 among others. If we swap these two substrings we get a larger value: 1110001010. This is the largest possible good substring that can be formed.

Function Description
Complete the function largestGood in the editor below. The function must return a string denoting the lexicographically largest possible good string that can be formed by performing zero or more swap operations on consecutive good substrings of binString.

largestGood has the following parameter(s):
binString: a string

Constraints

Function Description
Complete the function largestGood in the editor below. The function must return a string denoting the lexicographically largest possible good string that can be formed by performing zero or more swap operations on consecutive good substrings of binString.

largestGood has the following parameter(s):
binString: a string

Constraints
- Each character of binString ∈ {01}.
- 1 ≤ |binString| ≤ 50
- binString is a good string.

▸ Input Format For Custom Testing
▾ Sample Case 0
Sample Input 0
11011000
Sample Output 0
11100100
Explanation 0
Given the good string binString = 11011000, we can choose two consecutive good substrings, 10 and 1100, to swap such that the resultant string, str = 11100100, is the lexicographically largest good string possible.
▸ Sample Case 1
▸ Sample Case 2

Input

Given a good binary string $binString$.

Output

Print the lexicographically largest possible good string that can be formed by performing zero or more swap operations on consecutive good substrings of $binString$.

input
11011000
output
11100100

input
1010111000
output
1110001010

In first sample test case, given the good binary string $binString = 11011000$, we can choose two consecutive good substrings, 10 and 1100, to swap such that the resultant string $str = 11100100$, is the lexicographically largest good string possible.
