Practice Arena

Practice problems aimed to improve your coding skills.

- PRACTICE-02_SCAN-PRINT
- PRACTICE-03_TYPES
- LAB-PRAC-02_SCAN-PRINT
- LAB-PRAC-01
- PRACTICE-04_COND
- BONUS-PRAC-02
- LAB-PRAC-03_TYPES
- PRACTICE-05_COND-LOOPS
- LAB-PRAC-04_COND
- LAB-PRAC-05_CONDLOOPS
- PRACTICE-07_LOOPS-ARR
- LAB-PRAC-06_LOOPS
- LAB-PRAC-07_LOOPS-ARR
- LABEXAM-PRAC-01_MIDSEM
- PRACTICE-09_PTR-MAT
- LAB-PRAC-08_ARR-STR
- PRACTICE-10_MAT-FUN
- LAB-PRAC-09_PTR-MAT
- LAB-PRAC-10_MAT-FUN
- PRACTICE-11_FUN-PTR
- LAB-PRAC-11_FUN-PTR
    - Name the Clones
    - The Race of the Clones
    - Partial Palindrome
    - Growth Curve
    - The Family Tree of Mr C
    - Timely Tasks
    - Plenty of Palindromes
    - Count and Say Sequence
    - Orbiting Indices
    - Zig-zag Numbers
    - Parent Palindrome
    - Leaderboard
- LAB-PRAC-12_FUN-STRUC
- LABEXAM-PRAC-02_ENDSEM
- LAB-PRAC-13_STRUC-NUM
- LAB-PRAC-14_SORT-MISC

Growth Curve

LAB-PRAC-11_FUN-PTR

Growth Curve [20 marks]

Problem Statement
Mr C is fascinated with non-decreasing functions since they represent growth. A function $f : \mathbb{N} \to \mathbb{N}$ (where $\mathbb{N}$ is the set of natural numbers) is called non-decreasing if whenever $i \geq j$, we have $f(i) \geq f(j)$ where $i, j \in \mathbb{N}$. Thus, as we move from left to right on then number line, the function value may stay the same or go up, but never go down.

In the input, you will be given two non-negative integers n and MAX, separated by a space. As your output, in separate lines, you will have to list all non-decreasing functions of the form

$$f : [0 : n-1] \to [0 : \mathrm{MAX}]$$

by printing the values those functions take on the inputs $0, 1, 2, \ldots, n-1$, separated by a space. However, you have to list these functions in a certain *lexicographic* order, as described below.

Lexicographic ordering
Just as given two digit sequences, say 1923 and 3122, we can say which digit sequence is "smaller" and which is "larger" (by interpreting the digit sequences as numbers), given two sequences of non-negative numbers, say [0 8 11 4] and [1 0 2 3], we can say which digit sequence is "smaller" and which is "larger" in a very intuitive manner.

The rules of doing so are pretty simple if the two sequences are of the same length, which will be the case in this question (both sequences will have only n elements). To compare two sequences, simply look at the first numbers in the two sequences - the sequence with the larger number wins and is declared larger. If both sequences have the same first number, then the second numbers of the two sequences are compared and so on.

Note that we can use exactly these very rules to pronounce 3122 > 1923 but these rules allow us to declare that [0 8 11 4] < [1 0 2 3] since 0 < 1 (The second numbers 8 and 0 do not get compared at all since [1 0 2 3] wins as the larger number when the first numbers get compared). Similarly we have [3 5 9 3] < [3 8 1 1].

In your output, you will have to first output the "smallest function" i.e. the function that corresponds to the lexicographically smallest sequence, then the next smallest function, then the next smallest function and so on.

Caution

1. The integers n and MAX are non-negative. However, they can be zero.
2. Using recursion is not compulsory in this question. However, you will have to write much less code if you use recursion.
3. Do not output functions in incorrect order. The autograder will heavily penalize you if you do this since it will give you marks only if a correct function is output in its correct location.
4. We will not penalize you for stray newlines at the end of your output. However, do not have stray spaces at the end of each line of your output. You will not pass test cases if you have these.

HINTS:
This problem may seem complicated if you try to write a loop to solve the problem but it can be very elegantly solved using recursion as we explain below. Suppose we want to generate all functions of the form $f : [0 : 2] \to [0 : 1]$ (say n = 3 and MAX = 1), in lexicographic order. This can be solved if we first generate all functions that satisfy $f(0) = 0$ and then generate all functions that satisfy $f(0) = 1$ (since functions that have $f(0) = 0$ come before those that have $f(0) = 1$ in lexicographic order).

However, notice that in order to generate all functions with $f(0) = 0$, and that too in lexicographic order, we simply need to find all functions of the form $f : [1 : 2] \to [0 : 1]$. However, this can simply be thought of as a smaller version of the original problem.

Similarly, in order to generate all functions with $f(0) = 1$, and that too in lexicographic order, we simply need to find all functions of the form $f : [1 : 2] \to [1]$ (we are only interested in non-decreasing functions and since $f(0) = 1$, we must have $f(1), f(2) \geq 1$ but since $MAX = 1$, we are stuck). There is only one such function [1 1 1].

Write a function that takes partially filled in integer sequences (corresponding to functions) and recursively calls itself to complete the function descriptions. Write a function genFunctions(int *func, int n, int MAX, int i, int y) which takes in five arguments

1. an integer array func with n elements to describe the function values on 0 : n-1
2. the value of n (will help you know what is the length of the array)
3. the value of MAX (which all numbers are valid outputs of the function)
4. the value of i indicating that we have already decided f(0), f(1), ... f(i-1)
5. the value of y = f(i - 1) (you need this since you need to ensure that $f(j) \geq f(i-1)$ for all $j > i - 1$)

The base case of the recursion can be i = n which indicates that we have filled in the complete sequence corresponding to a function which can simply be printed. To print all functions that satisfy $f(0) = 0$, you could do something like the following:
int func[n];
func[0] = 0;
genFunctions(func, n, MAX, 1, 0); // one function value f(0) = 0 already filled in, rest of function values to be filled in non-decreasing order

To print all functions that satisfy $f(0) = 0$ and $f(1) = 5$ (assuming $MAX \geq 5$), you could do something like the following:
int func[n];
func[0] = 0;
func[1] = 5;
genFunctions(func, n, MAX, 2, 5); // Two values filled in with current max function value 5 rest to be filled in

Use these hints to completely solve the problem by writing the definition of the function genFunctions. Remember that genFunctions will need to call itself with suitable parameters.

EXAMPLE:
INPUT

```
3 1
```

OUTPUT:

```
0 0 0
0 0 1
0 1 1
1 1 1
```

Explanation: Note that the function [0 1 0] is not on the list since it is not a non-decreasing function. Many other such invalid functions (e.g. [1 0 1] etc) are not on the list.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

Start Solving! (/editor/practice/6218)