Assignment 2

Stacks and Queues
Data Structures and Algorithms
Due Date: 28 April, 2022

Instructions

1. Each of the problems can be solved using stacks, queue and deques.
2. You have to implement Stacks, Queues, and Deques from scratch. How you implement it is up to you, but make sure that your implementations works for the time and space constraints required for the problems.
3. For each ADT, make a header file and its corresponding C file. And write the solution to the problem in *q1.c* for Q1, *q2.c* for Q2, *q3.c* for Q3 and *q4.c* for Q4.
4. After implementing the Stacks, Queues, and Deques, you are only allowed to interact with them using the operations defined for that particular ADT. The operations for the ADTs are:
   - Stack:
     - *Pop*- Remove and return the **element** at the top of the stack.
     - *Push*- Add an **element** to the top of the stack.
     - *Top*- Return the **element** at the top of the stack.
     - *isEmpty*- Returns 1 when stack is empty, and 0 when it is not empty.
     - *Size*- Returns the number of **elements** inside the stack.
   - Queue:
     - *Dequeue*- Remove and return the **element** from the front of the queue.
     - *Enqueue*- Add an **element** to the back of the queue.
     - *Front*- Return the **element** at the front of the queue.
     - *isEmpty*- Returns 1 when queue is empty, and 0 when it is not empty.
     - *Size*- Returns the number of **elements** inside the queue.
   - Deque:
     - *Pop Front*- Remove and return the **element** from the front of the deque.
     - *Pop Back*- Remove and return the **element** from the back of the deque.
     - *Push Front*- Add an **element** to the front of the deque.
     - *Push Front*- Add an **element** to the back of the deque.
     - *Front*- Return the **element** at the front of the deque.
     - *Back*- Return the **element** at the back of the deque.
     - *isEmpty*- Returns 1 when deque is empty, and 0 when it is not empty.
     - *Size*- Returns the number of **elements** inside the deque.

   *Note that in operations where you have to return an element, you **cannot** return the pointer to that element.*
   For example, in an integer stack, *Top* should return an integer and **not** a pointer or struct.

5. You have to submit these problems to an Online Judge (OJ). Details regarding submission will be announced soon.
6. Some problems have subtasks. Grading for the subtasks will be binary, i.e. you either get full marks after passing all the test cases for the subtask, or you get a 0 if your code fails any of the test case.
7. The total marks obtained in the problem is the sum of marks obtained in all the subtasks of that problem.
8. Please write reusable code for ADTs, as you may be required to use the code for an ADT for one or more questions and future assignments too.

1 Amul Gives Too Much Work

Preparation for IIITH Programming Club's main event, i.e Codecraft is in full swing. Amul is coordinating the preparations and he gave Rutvij the following task.

Given an array A of length n having elements $a_1, a_2, \ldots, a_n$. You can perform the following operation any number of times:
For some position x such that $1 \leq x < n$, if $gcd(a_x, a_{x+1}) > 1$, delete a_x and a_{x+1}, and add $lcm(a_x, a_{x+1})$ at position x. Here, *gcd* is the *Greatest Common Divisor* and *lcm* is the *Least Common Multiple*.
For example, consider an array $[2, 4, 6, 9, 10]$ and we choose position $x = 3$. Then, $a_x = a_3 = 6, a_{x+1} = a_4 = 9$, and $gcd(a_3, a_4) = 3 > 1$. Hence, we replace a_3, a_4 with $lcm(a_3, a_4) = 18$, and the array formed after this operation is $[2, 4, 18, 10]$.
Find the array obtained after performing this operation **as many times as possible**.

Rutvij, being the procrastinator that he is, obviously did not finish the task. He wants to go play badminton. So, to get Amul to stop irritating him, Rutvij has passed this task down to you.

1.1 Constraints

1.1.1 Subtask 1 (30 marks)

$1 \leq n \leq 10^3$
$1 \leq a_i \leq 20$

1.1.2 Subtask 2 (70 marks)

$1 \leq n \leq 10^5$
$1 \leq a_i \leq 40$

1.2 Input

The first line contains the integer n, which denotes the size of input array.
The next line contains n integers of the array A.

1.3 Output

The first line of output should contain an integer k, the size of the array after the operation was performed as many times as possible.
The next line should contain k integers, the array after the operation was performed as many times as possible.
To avoid overflow, the output array should be a *long long* array.

1.4 Sample test cases

1.4.1 Test case 1

Input

```
6
2 4 6 9 10 11
```

Output

```
2
180 11
```

Explanation

We perform the operation at:

- $x = 3$, and now the array is $[2, 4, 18, 10, 11]$
- $x = 3$, and now the array is $[2, 4, 90, 11]$
- $x = 1$, and now the array is $[4, 90, 11]$
- $x = 1$, and now the array is $[180, 11]$

After this, we can see that we cannot perform the operation any more.

1.4.2 Test case 2

Input

```
10
3 6 9 4 5 15 7 2 6 5
```

Output

```
4
180 7 6 5
```

2 Kanjoos Pulak

Pulak is giving a treat at JC. He wants to choose the cheapest item to give to his friends. His friends know his intentions, and make a deal with him.

There are N items available at JC, and their price is denoted by an array $P = [p_1, p_2, \ldots, p_N]$ of length N where p_i is the price of the i^{th} item. For a fixed integer M ($M \leq N$), Pulak's friends will randomly choose an integer i ($1 \leq i \leq N - M + 1$) and Pulak will choose the cheapest item of i^{th},$(i+1)^{th}$,...,$(i+M-1)^{th}$ items to give to his friends.

Since the integer i is randomly chosen, Pulak wants to know the price of the cheapest item for **all possible values of** i.

He has an upcoming compilers deadline, and asks for your help. Please help Pulak, and you might get a treat too :)

2.1 Constraints

2.1.1 Subtask 1 (30 marks)

$1 \leq N \leq 10^3$
$1 \leq M \leq N$
$1 \leq p_i \leq 10^9$

2.1.2 Subtask 2 (70 marks)

$1 \leq N \leq 10^6$
$1 \leq M \leq N$
$1 \leq p_i \leq 10^9$

2.2 Input

The first line contains the integers N and M.
The next line contains N integers of the array P.
All A[i] can be stored inside an int variable

2.3 Output

The output should consist of a single line with (N-M+1) numbers, where the i^{th} number denotes the price of the cheapest of i^{th},$(i+1)^{th}$,...,$(i+M-1)^{th}$ items.

2.4 Sample test cases

2.4.1 Test case 1

Input

```
5 3
15 32 27 21 33
```

Output

```
15 21 21
```

2.4.2 Test case 2

Input

```
10 4
14 1 2 31 74 23 84 62 17 1999
```

Output

```
1 1 2 23 23 17 17
```

3 Tejas on the Rooftop

IIIT's Research Street has N buildings with distinct heights denoted by the array $H = [h_1, h_2, \ldots, h_N]$ of length N ($i \neq j \Rightarrow h_i \neq h_j$). Say Tejas is on the roof of building i, then he can go to roof of building j ($j > i$) with a ladder of length $d(i, j) = j - i + 1$ only if $h_{i+1}, \ldots, h_{j-1} < \min(h_i, h_j)$, i.e. if all the buildings between i and j have height less than both of these buildings.

Tejas is busy playing Elden Ring. So he asks you to compute the **sum of distances** $d(i, j)$ between all pairs (i, j) ($i < j$) of locations such that it is possible to go from the roof of building i to the roof of building j.

3.1 Constraints

3.1.1 Subtask 1 (30 marks)

$1 \leq N \leq 10^3$
$1 \leq h_i \leq 10^9$
$i \neq j \Rightarrow h_i \neq h_j$, *OR*, the elements of the array are distinct.

3.1.2 Subtask 2 (70 marks)

$1 \leq N \leq 10^5$
$1 \leq h_i \leq 10^9$
$i \neq j \Rightarrow h_i \neq h_j$, *OR*, the elements of the array are distinct.

3.2 Input

The first line contains the integer N, which denotes the size of input array.
The next line contains N integers denoting the heights of the buildings.

3.3 Output

Output contains a single integer denoting the sum of distances of all pairs of locations of buildings such that Tejas can go from the roof of the first building to the other.
To avoid overflow, store the output integer in a *long long* variable.

3.4 Sample test cases

3.4.1 Test case 1

Input

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

Output

```
24
```

Explanation

The pairs of required locations in this example are:
$(1, 2), (1, 5), (2, 3), (2, 4), (2, 5), (3, 4), (4, 5), (5, 6), (6, 7)$
And their respective distances are: $2, 5, 2, 3, 4, 2, 2, 2, 2$

4 (Ir)Responsible Shreyas

After fracturing his arm after falling from his wave-board, Shreyas took up the responsibility of creating a safe environment for newbies to learn how to ride skateboards and wave-boards. IIITH Administration gave him permission to make a skateboard park in the forest near the main gate, under the only condition that he cannot cut any trees.

The forest can be described by a grid G having N rows and M columns, where the element at the i^{th} row and j^{th} column is denoted by $G_{i,j}$. $G_{i,j}$ is ”*” if there is a tree at that location, and ”.” if it is empty.

Shreyas wants to find **the maximum area of a rectangular field** that can be placed in the forest so that **it does not have any trees in it**. Shreyas is busy (watching anime), so he asks for your help.
Note that area of a rectangle is calculated by counting the number of grid elements in the rectangle.

4.1 Constraints

4.1.1 Subtask 1 (30 marks)

$1 \leq N, M \leq 50$
$G_{i,j} =$ ”*” or ”.” (without the quotes)

4.1.2 Subtask 2 (70 marks)

$1 \leq N, M \leq 1000$
$G_{i,j} =$ ”*” or ”.” (without the quotes)

4.2 Input

The first line contains two integers N and M, which denotes the size of the forest grid.
The next N line contains M character strings describing the forest. Each character is ”.” (empty) or ”*” (has a tree).

4.3 Output

Output contains a single integer denoting the maximum area of a rectangular field that can be placed in the forest so that it does not have any trees in it.

4.4 Sample test cases

4.4.1 Test case 1

Input

```
4 7
...*.*.
.*.....
.......
......*
```

Output

12

Explanation

The rectangle with corners at $(2, 3)$ and $(4, 6)$ has the largest area.
Thus, the area is 12.