Mathematics 5365 (Analysis of Algorithms)

Assignment 1: Arrays

Submit your solutions typeset in TEX no later than Tuesday, September 4.

Acceptable modes of collaboration: discussing problems with your classmates orally or using a blackboard. You must indicate your collaborators in your submissions.

Unacceptable modes of collaboration:

- looking at or copying from a written solution of your classmate or somebody else;
- writing down something that you collaborator told you, but you do not understand.

1 Background

Abstract data structures:

- Set: binary relation $=$ such that $x = y$ and $y = z$ implies $x = z$; $x = x$ for all x; $x = y$ implies $y = x$;
- Poset: binary relation $\leq$ such that $x \leq y$ and $y \leq z$ implies $x \leq z$; $x \leq x$ for all x; $x \leq y$ and $y \leq x$ implies $x = y$;
- Order: a poset such that $x \leq y$ or $y \leq x$ for any x and y;
- abelian group (Ab): nullary operation 0, unary operation $-$, binary operation $+$ such that $x+(y+z) = (x+y)+z$, $0+x = x+0 = x$, $x+(-x) = (-x)+x = 0$, $x+y = y+x$;
- Ring: abelian group equipped with multiplication, i.e., nullary operation 1, binary operation $\cdot$ such that $x \cdot (y \cdot z) = (x \cdot y) \cdot z$, $1 \cdot x = x \cdot 1 = x$, $x \cdot (y+z) = x \cdot y + x \cdot z$, $(x+y) \cdot z = x \cdot z + y \cdot z$.

Each of the above operations uses $O(1)$ time.

If D denotes an abstract data structure, then $D[m]$ denotes the type of an array of m elements of type D indexed by integers in $[0, m)$. We say that an array $x : D[m]$ is *increasing* if D is equipped with a structure of a poset (and possibly other structures) and $x[i] \leq x[j]$ whenever $i \leq j$. The type of increasing arrays is denoted $D[m]_{\leq}$. We say that x is *strictly increasing* if $i < j$ implies $x[i] < x[j]$.

We denote $\mathbf{N} = \{0, 1, 2, \ldots\}$, $\mathbf{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$, and $\mathbf{B} = \{0, 1\}$.

2 Problems

1. Input data: $n : \mathbf{N}$, $x : \mathbf{Z}[n]$ such that $x[i] \in [0, k)$ for some fixed $k : \mathbf{N}$. Output data: $m : \mathbf{N}$. Design an $O(n + k)$ algorithm that computes the number m of distinct elements in x.

2. Input data: $A : \mathsf{Poset}$, $n : \mathbf{N}$, $x : A[n]_{\leq}$. Output data: $m : \mathbf{N}$. Design an $O(n)$ algorithm that computes the number m of distinct elements in x.

3. Input data: $m, n : \mathbf{N}$, $x : \mathbf{B}[m][n]$. Output data: $r : \mathbf{N}$. Interpret x as a two-dimensional array that represents an image where 1 is a black pixel and 0 is a white pixel. Several black rectangles of arbitrary width and height with disjoint sides are painted on an initially white image. The result is given to you as an input. Design an $O(mn)$ algorithm that computes the number r of black rectangles.

4. Input data: $A : \mathsf{Set}$, $m, n : \mathbf{N}$, $x : A[m + n]$. Output data: $x : A[m + n]$ (transformed as described). Design an $O(m + n)$ algorithm that transforms x in such a way that the segments $x[0, m)$ and $x[m, m + n)$ are exchanged, whereas the order of elements in each segment is preserved, using $O(1)$ additional memory. (The naive solution that uses an auxiliary array of the same size does not work because it uses $O(m + n)$ additional memory.)

5. Input data: $R : \mathsf{Ring}$, $n : \mathbf{N}$, $p : R[n]$, $x : R$. Output data: $u : R$, $v : R$. Design an $O(n)$ algorithm that computes $u = p(x) = \sum_{0 \leq k < n} p_k x^k$ and $v = p'(x)$ (the derivative of p evaluated at x) using $O(1)$ additional memory.

6. Input data: $R : \mathsf{Ring}$, $a : R$, $b : R$, $a' : R$, $b' : R$. Output data: $u : R$, $v : R$, $w : R$. Design an algorithm that computes $u = aa'$, $v = bb'$, and $w = ab' + ba'$ using only three multiplications and any (constant) number of additions.

7. Input data: $S : \mathsf{Order}$, $x : S[m]_{\leq}$, $y : S[n]_{\leq}$. Output data: $u : \mathbf{N}$, $v : \mathbf{N}$. Design an $O(m + n)$ algorithm that computes the number u of pairs (i, j) such that $x[i] = y[j]$, as well as the number v of distinct elements among the elements of x and y.

8. Input data: $S : \mathsf{Order}$, $m, n : \mathbf{N}$, $x : S[m]_{\leq}$, $y : S[n]_{\leq}$. Output data: $U : S[m + n]$, $k : \mathbf{N}$, $I : S[k]$. Design an $O(m + n)$ algorithm that produces a new increasing array $U : S[m + n]$ (U for union) that contains all elements of x and y (with the same multiplicities), rearranged in an increasing order. as well as an array $I : S[k]$ (I for intersection) (here $k \leq \min(m, n)$) that contains all elements that are common to x and y, with the multiplicity of such an element e being the minimum of multiplicities of e in x and y (e.g., if e occurs 5 times in x and 9 times in y, then the multiplicity of e in I is 5).

9. Input data: $S : \mathsf{Order}$, $m, n : \mathbf{N}$, $x : S[m][n]$, where each row $x[h][-]$ is an increasing array for any h (i.e., $x[h][i] \leq x[h][j]$ whenever $i \leq j$ and the relevant indices denote an element in x). Output data: $k : \mathbf{N}$, $I : S[k]$. Design an $O(mn)$ algorithm that computes the intersection I (with multiplicities; see the previous problem) of all rows $x[i][-]$.

10. Input data: $S : \mathsf{Order}$, $a : S$, $m, n : \mathbf{N}$, $x : S[m][n]$, each row $x[i][-]$ and column $x[-][j]$ is an increasing array for any i and j. Output data: $q : \mathbf{B}$, if q is true, also $i : \mathbf{N}$, $j : \mathbf{N}$. Design an $O(m + n)$ algorithm that produces a pair (i, j) such that $a = x[i][j]$ (the output q must be set to true in this case) or verifies that such a pair does not exist (the output q must be set to false in this case).

11. Input data: $S : \mathsf{Order}$, $a : S$, $m : \mathbf{N}$, $x : S[m]$. Output data: $x : S[m]$ (transformed as described). Design an $O(m)$ algorithm that permutes the elements of x in such a way that elements strictly less than a come first, elements equal to a come next, and elements strictly greater than a come last. Additional memory $O(1)$.

12. Input data: $A : \mathsf{Ab}$, $m, n, a, b : \mathbf{N}$, $x : A[m][n]$, $a \leq m$, $b \leq n$. Output data: $y : A[m + 1 - a][n + 1 - b]$. Design an $O(mn)$ algorithm that computes a new array $y : A[m + 1 - a][n + 1 - b]$ such that $y[i][j] = \sum_{i \leq k < i+a, j \leq l < j+b} x[k][l]$. (Hint: it may be helpful to solve the one-dimensional version of this problem first.)

3 General conventions

All input is supplied on the standard input and all output must go to the standard output.

The standard input contains zero or more datasets, each dataset is formatted according to the conventions below. The standard output must contain an answer for each dataset, in the format described below.

The following formatting conventions apply to both input and output, i.e., the input data is guarantee to be in this format, whereas the output data *must* be in this format in order to pass validation.

$b : \mathbf{B} = \{0, 1\}$: a single character, 0 or 1.

$n : \mathbf{N}$: a sequence of zero or more decimal digits, if $n > 0$, the first digit must be nonzero, but $n = 0$ is presented as a single digit 0. Unless specified otherwise, $n < 2^{31}$.

$n : \mathbf{Z}$: if $n < 0$, a minus sign $(-)$; followed by the representation of $|n|$. Unless specified otherwise, $n \in [-2^{31}, 2^{31})$.

If n is known (always for the input, sometimes for the output), an array $x : S[n]$ (S is an arbitrary data type than we know how to format) must be presented as $x[0] \ldots x[n-1]$, where consecutive elements are separated by a single space, and the last element must be followed by a single newline character. If $n = 0$, the representation consists of a single newline character.

If n is not known (i.e., n is computed by the algorithm), an array $x : S[n]$ must be presented as n $x[0]$ $\ldots$ $x[n-1]$, where each $x[i]$ is preceded by a single space, and there is a newline character at the end.

A two-dimensional array $x : S[m][n]$ is presented as a sequence of lines (one for each $i \in [0, m)$), each line contains the one-dimensional array $x[i][-]$. If m and n are not known (always for the input, sometimes for the output), then the whole presentation must be preceded by a line that contains m and n separated by a single space.

4 Format

1. $n \in [0, 2^{20}]$, $k \in [0, 2^{20}]$.

```
2 2 2 3 3 5 7 8
0 1
5
2
```

2. $A = \mathbf{Z}$.

```
7 9 1000000000
9
3
1
```

3.

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

4. $A = \mathbf{Z}$. Input: m x.

```
4 0 1 2 3 4 5 6 7 8 9
2 6 9 11
4 5 6 7 8 9 0 1 2 3
11 6 9
```

5. $R = \mathbf{Z}/(m)$ (elements are represented by integers in $[0, m)$), $m \in (0, 2^{31})$. Input: m x p.

7 2 1 2 3
3

6. $R = \mathbf{Z}$.

2 3 4 5
6 15 22

7. $S = \mathbf{N}$.

2 2 3 3 3
2 2 2 3
1 2 3 4
2 3 4 5
9 2
3 5

8. $S = \mathbf{N}$.

2 2 2 3 3 5
2 2 3
0 1
2 3
1
1
2 2 2 2 2 3 3 3 5
2 2 3
0 1 2 3
1 1
1

9. Input data: $S : \mathsf{Order}$, $m, n : \mathbf{N}$, $x : S[m][n]$, where each row $x[h][-]$ is an increasing array for any h (i.e., $x[h][i] \le x[h][j]$ whenever $i \le j$ and the relevant indices denote an element in x). Output data: $k : \mathbf{N}$, $I : S[k]$. Design an $O(mn)$ algorithm that computes the intersection I (with multiplicities; see the previous problem) of all rows $x[i][-]$.

10. Input data: $S : \mathsf{Order}$, $a : S$, $m, n : \mathbf{N}$, $x : S[m][n]$, each row $x[i][-]$ and column $x[-][j]$ is an increasing array for any i and j. Output data: $q : \mathbf{B}$, if q is true, also $i : \mathbf{N}$, $j : \mathbf{N}$. Design an $O(m + n)$ algorithm that produces a pair (i, j) such that $a = x[i][j]$ (the output q must be set to true in this case) or verifies that such a pair does not exist (the output q must be set to false in this case).

11. Input data: $S : \mathsf{Order}$, $a : S$, $m : \mathbf{N}$, $x : S[m]$. Output data: $x : S[m]$ (transformed as described). Design an $O(m)$ algorithm that permutes the elements of x in such a way that elements strictly less than a come first, elements equal to a come next, and elements strictly greater than a come last. Additional memory $O(1)$.

12. Input data: $A : \mathsf{Ab}$, $m, n, a, b : \mathbf{N}$, $x : A[m][n]$, $a \le m$, $b \le n$. Output data: $y : A[m + 1 - a][n + 1 - b]$. Design an $O(mn)$ algorithm that computes a new array $y : A[m + 1 - a][n + 1 - b]$ such that $y[i][j] = \sum_{i \le k < i+a, j \le l < j+b} x[k][l]$. (Hint: it may be helpful to solve the one-dimensional version of this problem first.)