Mathematics 5365 (Analysis of Algorithms)

Assignment 2: Divide et impera

Submit your solutions typeset in TeX or calligraphed no later than Tuesday, September 11.

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 $\le$ such that $x \le y$ and $y \le z$ implies $x \le z$; $x \le x$ for all x; $x \le y$ and $y \le x$ implies $x = y$;
- Order: a poset such that $x \le y$ or $y \le x$ 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 $x\cdot(y\cdot z) = (x\cdot y)\cdot z$, $1\cdot x = x\cdot 1 = x$, $x\cdot(y+z) = x\cdot y + x\cdot z$, $(x+y)\cdot z = x\cdot z + y\cdot z$;
- monoid (Monoid): nullary operation 1 and binary operation $\cdot$ such that $x\cdot(y\cdot z) = (x\cdot y)\cdot z$ and $x\cdot 1 = 1\cdot x = x$. Example: $\mathbf{Z}\cup\{\infty\}$ with ∞ and min as the nullary and binary operation.
- commutative monoid ($\mathsf{CommMonoid}$): a monoid such that $x\cdot y = y\cdot x$.
- ordered abelian group ($\mathsf{Ab}_{\le}$): an abelian group equipped with a compatible order structure: $a \le b$ implies $a+c \le b+c$ for all c. Examples: $\mathbf{Z}$, $\mathbf{Q}$.

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 $x[i] \le x[j]$ whenever $i \le j$. The type of increasing arrays is denoted $D[m]_{\le}$. 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: $S : \mathsf{Order}$, $n : \mathbf{N}$, $x : S[n]_{\le}$, $a : S$. Output data: $k : \mathbf{N}$, $l : \mathbf{N}$ such that elements with indices in $[0, k)$ are strictly less than a, elements with indices in $[k, l)$ are equal to a, and elements with indices in $[l, n)$ are strictly greater than a. Worst-case runing time: $O(\log n)$.

2. Input data: $A : \mathsf{Ab}$, $n : \mathbf{N}$, $x : A[n]$. Output data: x (transformed as described below). Worst-case running time: $O(n)$. Additional memory: $O(1)$. Transform the array x in place so that the new value y of x satisfies $y[k] = \sum_{k-2^a<i\le k} x[i]$, where 2^a is the largest power of 2 that divides $k+1$. (For example, if $n = 4$, then $x = [a, b, c, d]$ would be replaced by $[a, a+b, c, a+b+c+d]$.)

3. Input data: $A : \mathsf{Ab}$, $n : \mathbf{N}$, $y : A[n]$, $k, l : \mathbf{N}$, $0 \le k \le l \le n$. Output data: $r : A$, where $r = \sum_{k\le i<l} x[i]$, where x denotes the array from which y was obtained as in the previous problem. (The algorithm is only allowed to use y, not x.) Worst-case running time: $O(\log n)$. Additional memory: $O(1)$.

4. Input data: $M : \mathsf{Monoid}$, $n : \mathbf{N}$, $x : M[n]$, $q : \mathbf{N}$, $a, b : \mathbf{N}[q]$, $0 \le a[i] \le b[i] \le n$. Output data: $r : M[q]$, where $r[p] = \sum_{i\in[a[p],b[p])} x[i]$. (Remember that M is not necessarily a group, only a monoid, so there is no subtraction. A good example to keep in mind is $M = (\mathbf{Z} \cup \{\infty\}, \infty, \min)$, so $r[p]$ is the minimum of a on the interval $[a[p], b[p])$.) Worst-case running time: $O(n + q \log n)$.

5. Input data: $M : \mathsf{CommMonoid}$, $n : \mathbf{N}$, $q : \mathbf{N}$, $a, b : \mathbf{N}[q]$, $w : M[q]$, $0 \le a[i] \le b[i] \le n$. Output data: $r : A[q]$. Worst-case running time: $O(q \log n)$. The algorithm should compute the following: define $x : M[n]$, assign $x[j] \leftarrow 1$ for all $j \in [0, n]$. Then for each $i \in [0, q)$ do the following: (1) Assign $r[i] \leftarrow \prod_{j\in[a[i],b[i])} x[j]$; (2) Assign $x[j] \leftarrow x[j] \cdot w[i]$ for all $j \in [a[i], b[i])$. (Of course, interpreting these formulas as is would produce an algorithm with running time $O(qn)$, which is too slow, so instead you should seek to emulate these operations in a different way.)

6. Input data: $R : \mathsf{Ring}$, $n : \mathbf{N}$, $p : R[n]$, $x : R$, $x^n = 1$, $n = 2^a$ for some $a : \mathbf{N}$. Output data: $u : R[n]$, where $u[i] = p(x^i) = \sum_{j\in[0,n)} p[j]x^{i\cdot j}$. Worst-case running time $O(n \log n)$. (Don't forget that R need not be commutative: $x \cdot y \ne y \cdot x$, e.g., for the ring of matrices.) Hint: in the expression $\sum_{0\le j<n} p[j]x^{i\cdot j}$ group together terms with even and odd j respectively. At this point the fact that $n = 2^a$ becomes crucial because $n/2$ is an integer and $x^n = (x^2)^{n/2} = 1$, so one can solve a similar problem with parameters $n/2$ and x^2 instead of n and x.

7. Input/output data: same as in the previous problem, but u and p exchange their roles. In the ring R all elements $n \cdot 1_R$, where $n \in \mathbf{Z}$, $n \ne 0$, are invertible. Explain how to recover p from u, in the same time. Hint: what happens when you apply the previous algorithm twice, i.e., apply it the output data u? Using this algorithm, explain how to compute $p \cdot q$ for two polynomials p and q, assuming R is commutative ($x \cdot y = y \cdot x$), with the worst-case running time $O(n \log n)$.

8. Input data: $n : \mathbf{N}$, $x : \mathbf{N}[n]$, $b : \mathbf{N}$, $x[i] \in [0, 2^b)$. Output data: x (transformed as described below). Reorder the elements of x so that $i \le j$ implies $x[i] \le x[j]$. Worst-case running time $O(bn)$. Additional memory: $O(1)$.