CS 598TMC, Fall 2023
Timothy Chan

Homework 2 (due Oct 13 Friday 10am)

Instructions: see previous homework.

1. [*25 pts*] In this question, we investigate a special case of the problem of maintaining the minimum of a set S of numbers, where all numbers are integers in $[U]$. Specifically, we want a data structure to support the following operations:

   - insert-special(x): insert an element x to S where $x \in [U]$ and x is greater than the current minimum;
   - decrease-key-special(x, k): decrease x's value to k where $k \in [U]$ and k is greater than the current minimum;
   - delete-min(): return the minimum from S and remove this element.

   Note that because of the assumptions in insert-special() and decrease-key-special(), we know that the current minimum *can only increase over time*. (This special case arises, for example, in Dijkstra's shortest path algorithm.)

   (a) [*10 pts*] First give a simple data structure that supports insert-special() and delete-min() in $O(U/N)$ amortized time, and decrease-key-special() in $O(1)$ amortized time,[1] where N denotes the total number of operations.

   (Note: this method is thus better than the methods from class when U is linear in N. Hint: just use an array of size U... You may assume that N and U are known in advance, $U \geq N$, and that all keys are distinct.)

   (b) [*15 pts*] Give a still better data structure that supports insert-special() in $O(1)$ amortized time, delete-min() in $O(\log(U/N))$ amortized time, and decrease-key-special() in $O(1)$ amortized time.

   (Hint: Use an array of N lists of size $O(U/N)$, and store one of the lists (the "active" one) in a Fibonacci heap or quake heap...)

2. [*25 pts*] We want a data structure that supports the following operations on a collection of *not necessarily disjoint* sets of elements (integers):

   - makeset(x): create a set containing one element of value x (there may be other elements having the same value).
   - union(S_1, S_2): create a set $S_1 \cup S_2$, with duplicates removed (S_1 and S_2 may share common values). The new set is now in the collection, but the old sets S_1 and S_2 are not.

[1] $O(U/N)$ amortized time for decrease-key-special() is also acceptable.

- size(S): return the number of *distinct* values in the set S.

Describe a solution that achieves $O(\log n)$ expected amortized time per operation, where n is the number of makesets.

(Hint: we can't directly apply a union-find data structure here, but the weighted-union heuristic idea may still be helpful... Also, use hashing.)

3. [*20 pts*] Give a data structure to support the following operations on a set S of intervals in one dimension:

   - insert(a, b): given two integers $a, b \in \{1, 2, \ldots, U\}$, insert the interval $[a, b]$ to S.
   - query(x): given integer $x \in \{1, 2, \ldots, U\}$, return yes iff x lies in the union of the intervals in S.

   (For example, if S contains the intervals $[1, 4]$, $[6, 10]$, $[8, 13]$, the union of S is $[1, 4] \cup [6, 13]$.)

   Your solution should achieve $O(U\alpha(U))$ amortized preprocessing time, and $O(\alpha(U))$ amortized insert and query time.

   (Hint: use union-find.)

4. [*30 pts*] In class, we presented Fredman, Komlós, and Szemerédi's data structure for the static dictionary problem, for n integers in $[U]$, achieving $O(n)$ space, $O(1)$ worst-case time for queries (i.e., lookups), and $O(n)$ *expected* preprocessing time. In the original FKS paper, they presented a variant of their method achieving $O(n^2 \log U)$ deterministic preprocessing time. In this question, you will explore one way to further reduce the deterministic preprocessing time.[2]

   (a) [*5 pts*] Let $m \leq U$. Consider the following hash function family (which is simpler than the one from class):
   Pick a random prime p in the range $[m/2, m]$. Define $h_p : [U] \to [m]$ by

   $$h_p(x) = x \bmod p.$$

   Prove that for any fixed $x, y \in [U]$ with $x \neq y$, we have $\Pr_p[h_p(x) = h_p(y)] \leq O((\log U)/m)$.
   (Thus, this hash function family is "almost" universal, ignoring the extra $\log U$ factor.)
   (Note : the random variable here is the prime p (unlike the hash function family from class, where the prime was fixed and the random variables were a and b). You may use the prime number theorem, which implies that there are $\Theta(m/\log m)$ primes in the range $[m/2, m]$. Can you upper-bound the number of prime divisors in the range $[m/2, m]$ that the number $x - y$ can have?)

[2] As usual, in multi-part questions, even if you are unable to do one part, you may still be able to do the next part (and get full credit for the next part) by assuming that what is stated in the previous parts is true. That's why the question is divided into parts!

(b) [*3 pts*] For each $x \in S$, define $B_p(x) = \{y \in S - \{x\} : h_p(x) = h_p(y)\}$ (i.e., the "bucket" containing x). Call x *p-bad* if $|B_p(x)| > (n/\sqrt{m}) \log U$. Using (a), prove that $\Pr_p[x \text{ is } p\text{-bad}] \leq O(1/\sqrt{m})$.

(Hint: Markov's inequality.)

(c) [*3 pts*] Using (b), prove that there exists a prime $p \in [m/2, m]$ such that the number of p-bad elements in S is $O(n/\sqrt{m})$.

(d) [*4 pts*] Show that a prime p satisfying the condition in (c) can be found in $O(nm)$ deterministic time.

(Note: you may assume that all primes at most m can be listed in $O(m)$ time by standard algorithms.)

(e) [*12 pts*] Suppose someone has found a data structure for the static dictionary problem with $O(n)$ space, $O(1)$ worst-case query time, and $O(n^{1+\alpha} \log^{\beta} U)$ deterministic preprocessing time for constants $\alpha, \beta > 0$. (E.g., FKS got $\alpha = 1$ and $\beta = 1$.) Using (d), describe an improved data structure for the static dictionary problem with $O(n)$ space, $O(1)$ worst-case query time, and $O(n^{1+\alpha'} \log^{\beta'} U)$ deterministic preprocessing time for new constants $\alpha', \beta' > 0$ with $\alpha' = \frac{2\alpha}{2+\alpha}$.

(f) [*3 pts*] Using (d) and bootstrapping, show that the deterministic preprocessing time can be made $O(n^{1+\varepsilon} \log^{O(1)} U)$ for an arbitrarily small constant $\varepsilon > 0$.