Problem set for the Algoritmica 2 class (2016/7)

Roberto Grossi
Dipartimento di Informatica, Università di Pisa
grossi@di.unipi.it

October 20, 2016

Abstract

This is the problem set assigned during class. What is relevant during the resolution of the problems is the reasoning path that leads to their solutions, thus offering the opportunity to learn from mistakes. This is why they are discussed by students in groups, one class per week, under the supervision of the teacher to guide the brainstorming process behind the solutions. The *wrong* way to use this problem set: accumulate the problems and start solving them alone, a couple of weeks before the exam. The correct way: solve them each week in groups, discussing them with classmates and teacher.

1. [Range updates] Consider an array C of n integers, initially all equal to zero. We want to support the following operations:

   - `update`(i, j, c), where $0 \leq i \leq j \leq n-1$ and c is an integer: it changes C such that $C[k] := C[k] + c$ for every $i \leq k \leq j$.
   - `query`(i), where $0 \leq i \leq n-1$: it returns the value of $C[i]$.
   - `sum`(i, j), where $0 \leq i \leq j \leq n-1$: it returns $\sum_{k=i}^{j} C[k]$.

   Design a data structure that uses $O(n)$ space, takes $O(n \log n)$ construction time, and implements each operation above in $O(\log n)$ time. Note that `query`(i) = `sum`(i, i) but it helps to reason.

   [Hint: For the general case, use the segment tree seen in class, which uses $O(n \log n)$ space: prove that its space is actually $O(n)$ when it is employed for this problem.] [Hint to further save space in practice when the only changes are `update`(i, i, c): use an implicit tree such as the Fenwick tree (see wikipedia).]

2. [Depth of a node in a random search tree] A random search tree for a set S can be defined as follows: if S is empty, then the null tree is a random search tree; otherwise, choose uniformly at random a key $k \in S$: the random search tree is obtained by picking k as root, and the random search trees on $L = \{x \in S : x < k\}$ and $R = \{x \in S :$

$x > k\}$ become, respectively, the left and right subtree of the root k. Consider the randomized QuickSort discussed in class and analyzed with indicator variables [CLRS 7.3], and observe that the random selection of the pivots follows the above process, thus producing a random search tree of n nodes. Using a variation of the analysis with indicator variables, prove that the expected depth of a node (i.e. the random variable representing the distance of the node from the root) is nearly $2 \ln n$. Prove that the expected size of its subtree is nearly $2 \ln n$ too, observing that it is a simple variation of the previous analysis.

Prove that the probability that the expected depth of a node exceeds $c2 \ln n$ is small for any given constant $c > 1$. [Note: the latter point can be solved after we see Chernoff's bounds.]

3. [Karp-Rabin fingerprinting on strings] Given a string $S \equiv S[0 \ldots n-1]$, and two positions $0 \leq i < j \leq n-1$, the longest common extension $\texttt{lce}_S(i,j)$ is the length of the maximal run of matching characters from those positions, namely: if $S[i] \neq S[j]$ then $\texttt{lce}_S(i,j) = 0$; otherwise, $\texttt{lce}_S(i,j) = \max\{\ell \geq 1 : S[i \ldots i+\ell-1] = S[j \ldots j+\ell-1]\}$. For example, if $S = \texttt{abracadabra}$, then $\texttt{lce}_S(1,2) = 0$, $\texttt{lce}_S(0,3) = 1$, and $\texttt{lce}_S(0,7) = 4$. Given S in advance for preprocessing, build a data structure for S based on the Karp-Rabin fingerprinting, in $O(n \log n)$ time, so that it supports subsequent online queries of the following two types:

   - $\texttt{lce}_S(i,j)$: it computes the longest common extension at positions i and j in $O(\log n)$ time.
   - $\texttt{equal}_S(i,j,\ell)$: it checks if $S[i \ldots i+\ell-1] = S[j \ldots j+\ell-1]$ in constant time.

   Analyze the cost and the error probability. The space occupied by the data structure can be $O(n \log n)$ but it is possible to use $O(n)$ space. [Note: in this exercise, a one-time preprocessing is performed, and then many online queries are to be answered on the fly.]

4. [Hashing sets] Your company has a database $S \subseteq U$ of keys. For this database, it uses a randomly chosen hash function h from a universal family $\mathcal{H}$ (as seen in class); it also keeps a bit vector B_S of m entries, initialized to zeroes, which are then set $B_S[h(k)] = 1$ for every $k \in S$ (note that collisions may happen). Unfortunately, the database has been lost, thus only B_S and h are known, and the rest is no more accessible. Now, given $k \in U$, how can you establish if k was in S or not? What is the probability of error? Under the hypothesis that $m \geq c|S|$ for some $c > 1$ (note: we do not know the actual values of c and $|S|$...) can you estimate the size $|S|$, i.e. the size of S, looking at just h and B_S? What is the probability of error?

   Later, another database R has been found to be lost: it was using the same hash function h, and the bit vector B_R defined analogously as above. Using h, B_S, and B_R, how can you establish if k was in $S \cup R$ (union), $S \cap R$ (intersection), or $S \setminus R$ (difference)? What is the probability of error?

5. [Family of uniform hash functions] The notion of pairwise independence says that, for any $x_1 \neq x_2$ and $c_1, c_2 \in Z_p$, we have that

$$\Pr_{h \in \mathcal{H}}[h(x_1) = c_1 \wedge h(x_2) = c_2] = \Pr_{h \in \mathcal{H}}[h(x_1) = c_1] \times \Pr_{h \in \mathcal{H}}[h(x_2) = c_2]$$

In other words, the joint probability is the product of the two individual probabilities. Show that the family of hash functions $\mathcal{H} = \{h_{ab}(x) = ((ax + b) \texttt{ mod } p) \texttt{ mod } m : a \in Z_p^*, b \in Z_p\}$ (seen in class) is "pairwise independent", where p is a sufficiently large prime number ($m + 1 \leq p \leq 2m$).