Problem set for the Algoritmica 2 class (2015/16)

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

December 3, 2015

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. [Randomized selection] Consider the randomized quicksort, analyzed with the indicator variables, discussed in class (also, paragraph 7.3 in the textbook CLRS - Comern, Leiserson, Rivest, Stein, *Introduction to Algorithms*, 3rd edition, MIT Press). Show how to modify the randomized quicksort so that, given an array A and an integer $1 \leq k \leq |A|$, it finds the kth largest element in A without fully sorting A. Consider the analysis with indicator variables seen in class, and adapt it to show that the selection algorithm thus obtained requires linear expected time. [hint: since the algorithm has A and k as input, define an indicator variable X_{ijk} for z_i and z_j, where $i < j$, with the knowledge that z_k is looked for. The probability will have three cases, according to the relative order of z_k with respect to z_i and z_j.]

2. [Randomized min-cut algorithm] Consider the randomized min-cut algorithm discussed in class. We have seen that its probability of success is about $2/n^2$.

   - Describe how to implement the algorithm when the graph is represented by adjacency lists, and analyze its running time.
   - A weighted graph has a weight $w(e)$ on each edge e, which is a positive real number. The min-cut in this case is meant to be min-weighted cut, where the sum of the weights in the cut edges is minimum. Describe how to extend the algorithm to weighted graphs, and show that the probability of success is still $2/n^2$. [hint: define the weighted degree of a node]

- Show that running the algorithm for $N = cn^2 \ln n$ times, for a constant $c > 0$, and taking the minimum among the N min-cuts thus produced, the probability of success can be made at least $1 - 1/n^c$ (hence, with high probability). [hint: use the fact that $(1 - 1/x)^x \approx e^{-x}$ for small x.]
- Optional. When the graph becomes smaller, the probability of hitting a bad edge is higher. To reduce this chance, what if the algorithm is stopped when the resulting graph contains half of the original vertices? Run it four times independently, starting from the same graph G, and thus obtaining four graphs G_1, G_2, G_3, G_4, each with $n/2$ vertices. Apply recursively this idea to each G_i independently. Each time that two vertices are obtained, return the edges (as before). At the end, choose the best min-cut thus found among all those generated. Show what is the time complexity and the probability of error. [hint: it is a divide-and-conquer approach whose time cost is a recurrence relation; same for the probability]

3. [External memory mergesort] In the external-memory model (hereater EM model), show how to implement the k-way merge (where $(k + 1)B \leq M$), namely, how to simultaneously merge k sorted sequences of total length N, with an I/O cost of $O(N/B)$ where B is the block transfer size. Also, try to minimize and analyze the CPU time cost. Using the above k-way merge, show how to implement the EM mergesort and analyze its I/O complexity and CPU complexity.

4. [Family of uniform hash functions] Show that the family of hash functions $H = \{h(x) = ((ax + b) \% p) \% m\}$ is (almost) "pairwise independent", where $a, b \in [m]$ with $a \neq 0$ and p is a sufficiently large prime number ($m + 1 \leq p \leq 2m$). The notion of pairwise independence says that, for any x_1, x_2 and $c_1, c_2 \in [m]$, we have that $\Pr_{h \in H}[h(x_1) = c_1 \wedge h(x_2) = c_2] = \Pr_{h \in H}[h(x_1) = c_1] \times \Pr_{h \in H}[h(x_2) = c_2]$. In other words, the joint probability is the product of the two individual probabilities.

5. [Deterministic data streaming] Consider a stream of n items, where items can appear more than once in the stream. The problem is to find the most frequently appearing item in the stream (where ties broken arbitrarily if more than one item satisfies the latter). Suppose that only $k = O(\log^c n)$ items can be stored, one item per memory cell, where the available storage is $k + O(1)$ memory cells. Show that the problem cannot be solved deterministically under the following rules: the algorithm can access only $O(\log^c n)$ information for each of the k items that it can store, and can read the next item of the stream; you, the adversarty, have access to all the stream, and the content of the k items stored by the algorithm, and can decide what is the next item that the algorithm reads (please note that you cannot change the past, namely, the items already read by the algorithm) . Hint: it is an adversarial argument based on the k items chosen by the hypothetical determinist streaming algorithm, and the fact that there can be a tie on $> k$ items till the last minute.

6. [Special case of most frequent item in a stream] Suppose to have a stream of n items, so that one of them occurs $> n/2$ times in the stream. Also, the main memory is

limited to keeping just *two* items and their counters, plus the knowledge of the value of n beforehand. Show how to find deterministically the most frequent item in this scenario. [Hint: since the problem cannot be solved deterministically if the most frequent item occurs $\leq n/2$ times, the fact that the frequency is $> n/2$ should be exploited.]

7. [Count-min sketch: extension to negative counters] Check the analysis seen in class, and discuss how to allow $F[i]$ to change by arbirary values read in the stream. Namely, the stream is a sequence of pairs of elements, where the first element indicates the item i whose counter is to be changed, and the second element is the amount v of that change (v can vary in each pair). In this way, the operation on the counter becomes $F[i] = F[i] + v$, where the increment and decrement can be now seen as $(i, 1)$ and $(i, -1)$.

8. [Count-min sketch: range queries] Show and analyze the application of count-min sketch to range queries (i, j) for computing $\sum_{k=i}^{j} F[k]$. Hint: reduce the latter query to the estimate of just $t \leq 2 \log n$ counters $c_1, c_2, \ldots, c_t$. Note that in order to obtain a probability at most δ of error (i.e. that $\sum_{l=1}^{t} c_l > \sum_{k=i}^{j} F[k] + 2\epsilon \log n ||F||$), it does not suffices to say that it is at most δ the probability of error of each counter c_l: while each counter is still the actual wanted value plus the residual as before, it is better to consider the sum V of these t wanted values and the sum X of these residuals, and apply Markov's inequality to V and X rather than on the individual counters.

9. [Succinct data structure for range queries] Borrowing the idea of dyadic intervals employed in the above solution for the count-min sketch for range queries, design a data structure that uses few additional bits to preprocess a bitvector B of length n, such that $B[i]$ is the bit in position i for $0 \leq i < n$. After that, the data structure must support any query of the form $\texttt{xor}(i, j)$ to return the bitwise exclusive or of the bits $B[i], B[i + 1], \ldots, B[j]$ for $0 \leq i \leq j < n$. Analyze the complexity of the proposed solution.

10. [External memory implicit searching] Given a static input array A of N keys in EM, describe how to organize the keys inside A by suitably permuting them during a preprocessing step, so that any subequent search of a key requires $O(\log_B N)$ block transfers using just $O(1)$ blocks of auxiliary storage (besides those necessary to store A). Clearly, the CPU complexity should remain $O(\log N)$. Discuss the I/O complexity of the above preprocessing, assuming that it can uses $O(N/B)$ blocks of auxiliary storage. (Note that the additional $O(N/B)$ blocks are employed only during the preprocessing; after that, they are discarded as the search is implicit and thus cannot use any extra block.)

11. [Number of splits for (a, b)-trees] Consider the (a, b)-trees with $a = 2$ and $b = 3$. Describe an example of an (a, b)-tree with N keys and choose a value of the search key k such that performing a sequence of m operations insert(k), delete(k), insert(k), delete(k), insert(k), delete(k), etc. ..., produces $\Theta(mH)$ split and fuse operations, where $H = O(\log_a N/a)$ is the height. Try to make the example as general as possible.

After that, consider the (a, b)-trees with $a = 2$ and $b = 8$: produce some examples to check that the above situation cannot occur. Try to guess some properties from the examples using the fact that $a = b/4$: if they are convincing, try to prove and use them to show that the situation mentioned above cannot occur; in general, prove that, for any sequence of m updates (arbitrarily chosen from insertions and deletions) starting from an empty $(, b)$-tree, the number of split and fuse operations is $O(m/a)$, which is $O(m/B)$ when $a, b = \Theta(B)$ in external memory.

12. [1-D range query] Describe how to perform a one-dimensional range queries in (a, b)-trees with $a, b = \Theta(B)$. Given two keys $k_1 \leq k_2$, the query asks to report all the keys k in the (a, b)-tree such that $k_1 \leq k \leq k_2$. Give an analysis of the cost of the proposed algorithm, which should be output-sensitive, namely, $O(\log_B N + R/B)$ block transfers, where R is the number of reported keys.

    After that, for a given set of N keys, describe how to build an (a, b)-tree for them using $O(\text{sort}(N))$ block transfers, where $\text{sort}(N) = \Theta(N/B \log_{M/B} N/B)$ is the optimal bound for sorting N keys in external memory.

13. [External memory (EM) permuting] Given two input arrays A and π, where A contains N elements and π contains a permutation of $\{1, \ldots, N\}$, describe and analyze an optimal external-memory algorithm for producing an output array C of N elements such that $C[\pi[i]] = A[i]$ for $1 \leq i \leq N$.

    After that, extend the lower bound argument given for the sorting problem in the EM model to the permutation problem: given N input elements $e_1, e_2, \ldots, e_N$ and an input array π containing a permutation of the integers in $[1, 2, \ldots, N]$, rearrange in EM the elements according to the permutation in π, so that they appear in the order $e_{\pi[1]}, e_{\pi[2]}, \ldots, e_{\pi[N]}$.

14. [Lower bound for searching] For the best possible comparison-based searching algorithm in a sorted array of N elements stored in external memory, prove that each search requires $\Omega(\log_B N)$ I/Os in the worst case.

15. [Cache-oblivious selection] Consider the linear-time deterministic selection discussed in class (see paragraph 9.3 in the textbook CLRS - Comern, Leiserson, Rivest, Stein, *Introduction to Algorithms*, 3rd edition, MIT Press). Prove that this algorithm is cache-oblivious with complexity $O(N/B)$ block transfers (i.e. cache misses) for N elements stored in an array and *any* unknown block size B.

16. [Implicit navigation in vEB layout] Consider $N = 2^h - 1$ keys where h is a power of 2, and the implicit cache-oblivious vEB layout of their corresponding complete binary tree, where the keys are suitably permuted and stored in an array of length N without using pointers (as it happens in the classical implicit binary heap but the rule here is different). The root is in the first position of the array. Find a rule that, given the position of the current node, it is possible to locate in the array the positions of its

left and right children. Discuss how to apply this layout to obtain (a) a static binary search tree and (b) a heap data structure, discussing the cache complexity.

17. [MapReduce indegree distribution] Use the basic cache-oblivious operations of scan and sort as building blocks for computing the degree distribution of a graph (e.g. a web graph), assuming that this graph is already given and represented by adjacency lists: the output lists all possible indgree values d and, for each such d, the number of nodes having indegree d. Equivalently, see this computation in terms of MapReduce/Haddop: just sketch the code for map and reduce, leaving the rest.

18. [MapReduce prefix sums] Given a huge array A of N entries, define the ith prefix sum of A as $S_i = \sum_{j=1}^{i} A[i]$. Show how to compute simultaneously all S_i, for $i = 1, 2, \ldots, N$, using overall $O(\log N)$ calls to scan/sort/MapReduce/Hadoop and producing $O(N \log N)$ or less pairs $\langle$key,value$\rangle$. [Hint: Easy with $O(N)$ calls. Instead, use partial sums of 2^k entries of A for all suitable values of k.]

19. [Suffix sorting in EM] Using the DC3 algorithm seen in class, and based on a variation of mergesort, design an EM algorithm to build the suffix array for a text of N symbols. The I/O complexity should be the same as that of standard sorting, namely, $O(N/B \log_{M/B} N/B)$ block transfers.

20. [Euler tour] Given a connect graph G with n vertices and m edges, where each node has even degree, design an algorithm to build an Euler tour for G in $O(n + m)$ time.

21. [Wrong greedy for minimum vertex cover] Find an example of (family of) graphs for which the following greedy approach fails to give a 2-approximation for the minimum vertex cover problem (and prove why this is so). Start out with an empty $\tilde{S}$. Choose each time a vertex v with the largest number of incident edges in the current graph. Add v to $\tilde{S}$ and remove its incident edges. Repeat the process on the resulting graph as long as there are edges in it. Return $|\tilde{S}|$ as the approximation of the minimal size of a vertex cover for the original input graph. Generalize your argument to show that the above greedy algorithm cannot actually provide an r-approximation for any given constant $r > 1$.