Problem set for the Algoritmica 2 class (2014/15)

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

April 17, 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. [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.

2. [External memory 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 $1 \leq i \leq N$.

3. [Lower bound for permuting] 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 $[1, 2, \ldots, N]$, rdisporre gli elementi secondo la permutazione in π. Dopo tale operazione, la memoria esterna deve contenerli nell'ordine $e_{\pi[1]}, e_{\pi[2]}, \ldots, e_{\pi[N]}$.

4. [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 (differently form the search).

5. [External memory distributin sort using splitters] Using the fact that it is possible to find $\sqrt{M/B}$ splitters for a set S of N elements with $O(N/B)$ block transfers, describe how to partition S into $\sqrt{M/B}+1$ disjoint subsets of keys. (A single single splitter is the classical quicksort partition; here it is a multi-parition in EM). Using this partition algorithm, design an EM distribution sort algorithm (alternative to EM mergesort) that takes $O(N/B \log_{M/B} N/B)$ block transfers.

6. [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 $H = O(\log_b N/b)$ 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, and that the number of split and fuse operations is $\Theta(m)$.

7. [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.

8. [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.

9. [Suffix array search] Suppose to run the DC3 algorithm on the text T of N symbols, so that the suffix array `SA` of N entries is available. Design and analyze efficient algorithms to find all the occurrences of a pattern string P of M symbols in T: we say that position i in T is an occurrence of P if the substring $T[i \ldots i+M-1]$ is equal to P symbol-wise. Note that this is an on-line query, meaning that T and `SA` are fixed, while P is provided each time by the user query and makes use of T and `SA` to list all the occurrences of P. The complexity of the query algorithm thus proposed should be

analyzed and commented in the following settings, assuming that P can be always kept in main memory: (1) both T and SA are in main memory; (2) T is main memory while SA is in external memory; (3) T is in external memory while SA is in main memory; (4) both T and SA are in external memory. Note that (2) and (3) describe a situation in which it is not possible to keep both T and SA in main memory, just only one of them.

10. [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.

11. [LCP, suffix arrays, and suffix trees] Show how to extend the DC3 construction of the suffix array SA so as to compute also the array LCP, where $LCP[i]$ contains the length of the longst common prefx between the two suffixes indicated by $SA[i]$ and $SA[i+1]$, for $1 \leq i \leq N-1$. The complexity of the original DC3 algorithm should not change. After that, show how to build the suffix tree in linear time using arrays SA and LCP for the given input text. [Hint: as seen in class, traversing the suffix tree in preorder, the suffixes represented in the leaves are those in the sequence SA and the skip values in the internal nodes are those in LCP. The idea is to build the suffix tree in preoder in which the current rightmost path is maintained, and the new leaf is assigned the next suffix in SA and is attached to the internal node (which is created if needed) in the path, with the next value in LCP as skip value.

12. [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.

13. [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 has been seen as k-wise independence in class, where $k=2$.

14. [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.

15. [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 $(i,-1)$.

16. [Not all equal 3-SAT] Consider the problem NE-3-SAT, which asks if there is an assignment of the Boolean variables in a conjunctive normal form formula F, where each clause contains three literals, such that each clause of F has at least one literal true and at least one literal false. Hint: use a polynomial reduction from 3-SAT.

17. [Wrong greedy for MAX-CUT] The NP-hard problem MAX-CUT is defined as follows for an undirected (weighted) graph $G=(V,E)$. A partition of V's vertices as $(C, V-C)$ where $C \subseteq V$ is called a cut. The set of cut edges $E(C) = \{(v,w) \in E \mid v \in C \text{ and } w \in V-C\}$ is called cut set. (We adopt the notation so that (v,w) and (w,v) denote the same undirected edge.) We are interested in finding the maximal cutsize, namely, $\max_{C \subseteq V} |E(C)|$. Find an example of graphs for which the following greedy approach fails to give a 2-approximation (and prove why this is so). Start out with an empty C. Choose each time a vertex v with the largest number of incident edges in the current graph. Add v to C and remove its incident edges. Repeat the process as long as there are edges in the graph. Return $|E(C)|$ as the approximation of the maximal cutsize.

18. [Randomized 2-approximation for MAX-CUT] Prove that the following randomized algorithm provided a 2-approximation for MAX-CUT in expectation, namely, the expected cutsize is at least half of the optimal cutsize. Here are the steps. *(1)* For each vertex $v \in V$, toss an unbiased coin: if it is tail, insert v into C; else, insert v into $V-C$. *(2)* Start out with an empty set T. For each esge $(v,w) \in E$, such that $v \in C$ and $w \in V-C$, add (v,w) to the set T. Return $|T|$ as the approximated solution.

19. [[Wrong greedy for minimum vertex cover] For an undirected graph $G=(V,E)$, a subset $S \subseteq V$ of vertices is a vertex cover if each edge $(u,v) \in E$ has $u \in S$ or $v \in S$.

The NP-hard minimum vertex cover (MVC) problem finds the smallest size of a vertex cover. Find an example of graphs for which the following greedy approach fails to give a 2-approximation (and prove why this is so). Start out with an empty S. Choose each time a vertex v with the largest number of incident edges in the current graph. Add v to S and remove its incident edges. Repeat the process as long as there are edges in the graph. Return $|S|$ as the approximation of the minimal size of a vertex cover. Generalize your argument to show that the above greedy algorithm cannot actually provide an r-approximation for any given constant $r > 1$.

20. [Approximation for MAX-SAT] In the MAX-SAT problem, we want to maximize the number of satisfied clauses in a CNF Boolean forumla. Consider the following approximation algorithm for the problem. Let F be the given formula, x_1, x_2, ..., x_n its Boolean variables, and c_1, c_2, ... c_m its clauses. Pick arbirtary Boolean values b_1, b_2, ..., b_n, where $b_i \in \{0, 1\}$ $(1 \leq i \leq n)$. Compute the number m_0 of satisfied clauses by the assignment having $x_i := b_i$ $(1 \leq i \leq n)$. Compute the number m_1 of satisfied clauses by the complement of the assignment, namely, having $x_i := \bar{b_i}$ $(1 \leq i \leq n)$, where $\bar{b_i}$ denotes the negation (complement) of b_i. If $m_0 > m_1$, return the assignment $x_i := b_i$ $(1 \leq i \leq n)$; else, return the assignment $x_i := \bar{b_i}$ $(1 \leq i \leq n)$.

    Show that the above algorithm provides an r-approssimation for MAX-SAT, and specify for which value of $r > 1$ (explaining why). Discuss how the choice of b_1, b_2, ..., b_n can impact the value of r, giving an explanation in your discussion. Optional: create an instance of the MAX-SAT problem where the returned value is exactly $1/r$ of the optimal solution, specifying which values of b_1, b_2 ..., b_n have been employed.