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

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

December 9, 2014

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] Nel modello EMM (external memory model), mostrare come implementare il k-way merge ($(k+1)B \leq M$), ossia la fusione di k sequenze individualmente ordinate e di lunghezza totale N, con costo I/O di $O(N/B)$ dove B è la dimensione del blocco. Minimizzare e valutare il costo di CPU. Analizzare il costo del merge sort (I/O complexity, CPU complexity) che utilizza tale k-way merge.

2. [External memory permuting] Dati due array A e π, dove A contiene N elementi (non importa quali) e π contiene una permutazione di $\{1, \ldots, N\}$, descrivere e analizzare nel modello EMM un algoritmo ottimo per costruire un terzo array C di N elementi tale che $C[\pi[i]] = A[i]$ per $1 \leq i \leq N$.

3. [Lower bound for permuting] Estendere l'argomentazione utilizzata per il limite inferiore al problema dell'ordinamento in memoria esterna a quello della permutazione: dati N elementi $e_1, e_2, \ldots, e_N$ e un array π contenente una permutazione degli interi in $[1, 2, \ldots, N]$, disporre 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] Dato un array statico A di N chiavi in memoria esterna, descrivere come organizzare le chiavi dentro A permutandole attraverso un opportuno preprocessing, in modo che sia possibile effettuare successivamente la ricerca di una chiave con $O(\log_B N)$ trasferimenti di blocchi utilizzando soltanto $O(1)$ blocchi di appoggio, oltre a quelli necessari a memorizzare A. (Chiaramente il tempo di CPU

deve rimaner $O(\log N)$.) Discutere il costo di tale preprocessing in memoria esterna, che può utilizzare $O(N)$ spazio aggiuntivo (al contrario della ricerca).

5. [External memory distributin sort using splitters] Utilizzando il fatto che è possibile trovare $\sqrt{M/B}$ splitter per un insieme di N elementi con $O(N/B)$ trasferimenti di blocchi, descrivere come creare $\sqrt{M/B}+1$ sottoinsiemi di chiavi, separati dagli splitter. (Nota: con un solo splitter, questo equive ad avere la classica partizione del quicksort dell'input in due sottoinsiemi). Utilizzando tale distribuzione, progettare un algoritmo di ordinamento (alternativo al mergesort) in memoria esterna che richeda $O(N/B \log_{M/B} N/B)$ trasferimenti di blocchi.

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 have run the DC3 algorithm on the text T of N symbols, so 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 $t \leq 2 \log n$ contatori $c_1, c_2, \ldots, c_t$. 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 thgat 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 be changed 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 $(i, 1)$ e $(i, -1)$.

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 containts 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] Find an example of graphs for which the greedy approach of choosing each time the vertex with the largest residual degree does not give a 2-approximation.