Esercizi per il Corso di Algoritmica 2 (a.a. 2014/15)

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

16 ottobre 2014

Sommario

Vengono raccolti i problemi discussi in classe e collegati agli argomenti a lezione. Tali problemi vengono approfonditi e osservati da più punti di vista, anche sbagliati, in quanto l'errore è funzionale all'apprendimento di situazioni complesse. La motivazione risiede nel fatto che interessa sviluppare il percorso che conduce alla soluzione (piuttosto che la soluzione stessa), sotto la guida del docente in base alle idee proposte dagli studenti. La soluzione non viene qui fornita per i motivi suddetti: è preferibile venire a ricevimento dal docente.

1. [Ordinamento in memoria esterna] 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. [Esecuzione della permutazione in memoria esterna] 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. [Limite inferiore per la permutazione] 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. [Ricerca implicita in memoria esterna] 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. [Ordinamento per distribuzione mediante splitter] 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 $k_1 \le k_2$, the query asks to report all the keys k in the (a,b)-tree such that $k_1 \le k \le k_2$. 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 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.