Esercizi per il Corso di Algoritmica 2 (a.a. 2012/13)

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

4 dicembre 2012

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.

NOTA: la lista è ancora incompleta.

1. [Ordinamento in memoria esterna] Nel modello EMM (external memory model), mostrare come implementare il k-way merge, ossia la fusione di n 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 (I/O complexity, CPU complexity) che utilizza tale k-way merge.

2. [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 $e_{\pi[1]}, e_{\pi[2]}, \ldots, e_{\pi[N]}$.

3. [Permutazione in memoria esterna] Dati tre array A, B, C di N elementi, dove A l'input e C una permutazione di $\{0, 1, \ldots, n-1\}$. descrivere e analizzare nel modello EMM un algoritmo ottimo per costruire $C = A[C[i]]$ per $0 \leq i \leq n-1$.

4. [Multi-selezione in memoria esterna] Scrivere tutti i passaggi dell'analisi del costo e della dimostrazione di correttezza dell'algoritmo di multi-selezione per la memoria esterna visto a lezione.

5. [MapReduce] Utilizzare il paradigma scan&sort mediante la MapReduce per calcolare la distribuzione dei gradi in ingresso delle pagine Web. In particolare, specificare quanti passi di tipo MapReduce sono necessari e quali sono le funzioni Map e Reduce impiegate. Ipotizzare di avere già tali pagine a disposizione.

6. [Navigazione implicita in vEB] Dato un albero completo memorizzato secondo il layout di van Emde Boas (vEB) in modo implicito, ossia senza l'ausilio di puntatori (come succede nello heap binario implicito), trovare la regola per navigare in tale albero senza usare puntatori espliciti.

7. [Layout di alberi binari] Proporre una paginazione di alberi binari in blocchi di dimensione B per realizzare un loro layout in memoria esterna: valutare se un qualunque cammino minimo radice-nodo di lunghezza ℓ attraversa sempre $O(\ell/\log B)$ pagine. NOTA: per chi vuole, esiste una versione più impegnativa di questo esercizio, basta contattarmi per averla.

8. [Suffix array in memoria esterna] Utilizzando la costruzione del suffix array basata sul mergesort e la tecnica DC3 vista a lezione, progettare un algoritmo per EMM per costruire il suffix array di un testo che abbia la stessa complessità del mergesort in EMM.

9. [Famiglia di funzioni hash uniformi] Mostrare che la famiglia di funzioni hash $H = \{h(x) = ((ax + b)\%\, p)\, m\}$ è (quasi) uniforme, dove $a, b \in [m]$ con $a \neq 0$ e p è un numero primo sufficientemente grande.

10. [Count-min sketch: estensione] Estendere l'analisi vista a lezione permettendo di incrementare e decrementare i contatori con valori arbitrati.

11. [Count-min sketch: prodotto scalare] Mostrare come utilizzare il paradigma del count-min sketch per approssimare il prodotto scalare (i.e., approssimare $\sum_{k=1}^{n} F_a[k] * F_b[k]$).

12. [Count-min sketch: prodotto scalare] Mostrare come utilizzare il paradigma del count-min sketch per rispondere alle interval query (i.e., approssimare $\sum_{k=i}^{j} F[k]$).

13. [Elementi distinti] Progettare e analizzare un algoritmo di data streaming che permetta di approssimare il numero di elementi distinti.

14. [Cuckoo hashing] Scrivere tutti i passaggi dell'analisi del costo dell'inserimento di un elemento in una tabella di cuckoo hashing. Discutere anche della cancellazione e della sua complessità.

15. [Random search tree] Scrivere l'algoritmo per inserire una chiave in un random search tree con una sola discesa dalla radice (i.e., senza dover risalire poi dalla foglia appena inserita verso la radice mediante le rotazioni).

16. [Lista invertita compressa] Prendiamo una sequenza ordinata crescente di n interi $i_1, i_2, \ldots, i_n$, come per esempio una lista invertita. La rappresentazione compressa differenziale è la sequenza S di $|S|$ bit ottenuti concatenando $\gamma(i_1)$, $\gamma(i_2 - i_1)$, $\gamma(i_3 - i_2)$, $\ldots$, $\gamma(i_n - i_{n-1})$, dove $\gamma(x)$ rappresenta il gamma code di Elias per la codifica istantanea dell'intero $x \geq 1$ in due $2\lfloor \log_2 x \rfloor + 1$ bits. Mostrare come aggiungere un'opportuna directory di spazio $O(|S|)$ bit (meglio ancora, di $o(|S|)$ bit) per poter accedere velocemente, dato $j \in [2..n]$, alla codifica $\gamma(i_j - i_{j-1})$. Estendere tale approccio per accedere velocemente a i_j (e quindi poter eseguire una ricerca binaria sugli interi della lista invertita compressa).

17. [Prefix tree del codice di Huffman] Impostare un algoritmo per costruire il prefix tree del codice di Huffman. Dimostrare l'ottimalità di tale albero in termini di numero di bit utilizzati per codici prefix free dei simboli.

18. [Applicazioni di LZ77] Sfruttando le caratteristiche dell'algoritmo LZ77 di Lempel e Ziv per suddividere un testo in una sequenza di frasi, mostrare come utilizzare LZ77 per (a) nascondere dei bit all'interno del file compresso risultante e per (b) trovare la più lunga sottostringa che si ripete, ossia che appare almeno due volte nel testo.

19. [Dizionario di LZ78] Progettare una struttura di dati per memorizzare e interrogare velocemente il dizionario delle frasi ottenute incrementalmente con l'algoritmo LZ78. Valutare il costo delle soluzioni proposte.