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

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

14 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.

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. [Esecuzione della permutazione in memoria esterna] Dati tre array A, C, D di N elementi, dove A è l'input e C è una permutazione di $\{0, 1, \ldots, n-1\}$, descrivere e analizzare nel modello EMM un algoritmo ottimo per costruire $D = A[C[i]]$ 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 un generico albero binario in blocchi di dimensione B per ottenere un layout in memoria esterna in cui un cammino radice-foglia attraversi $O(h \log B)$ pagine, dove h è l'altezza dell'albero. Opzionale: per chi vuole, esiste una versione più impegnativa di questo esercizio dove un cammino radice-nodo di lunghezza ℓ attraversa $O(\ell / \log B)$ pagine; contattarmi per discutere questa opzione.

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 di N simboli che abbia la stessa complessità del mergesort di N elementi in EMM.

9. [Famiglia di funzioni hash uniformi] Mostrare che la famiglia di funzioni hash $H = \{h(x) = ((ax + b) \% p) \, m\}$ è (quasi) "pairwise independent", 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 $\sum_{k=1}^{n} F_a[k] * F_b[k]$.

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

13. [Elementi distinti] Progettare e analizzare un algoritmo di data streaming che permetta di approssimare il numero di elementi distinti. Nota: utilizzare la rassegna indicata nel programma del corso.

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: l'effetto finale deve essere il medesimo di inserire

prima la chiave in una nuova foglia f (come negli alberi di ricerca tradizionali) e poi di far risalire f verso il nodo interno u opportuno mediante le rotazioni; notare che l'algoritmo richiesto deve attraversare il cammino dalla radice a u, evitando così di visitare il resto del sottoalbero radicato in u.

16. [Lista invertita compressa] Prendere 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: (a) mostrare come utilizzare LZ77 per nascondere dei bit all'interno del file compresso risultante senza aumentare la dimensione del file compresso risultante (nel caso questo sdia possibile); (b) utilizzare il suffix tree per costruire LZ77 per trovare anche 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.

20. [Approssimazione per MIN-VC] Il problema del MIN-VC (minimum vertex-cover) richiede, per un grafo $G = (V, E)$, di trovare un sottoinsieme $S \subseteq V$ di cardinalità minima tale che ogni arco sia incidente ad almeno un vertice di S, cioè per ogni $(u, v) \in E$ vale $u \in S$ oppure $v \in S$. Mostrare come il seguente approccio greedy fornisca una 2-approssimazione: inizializza S all'insieme vuoto e, per ogni arco $(u, v) \in E$, se u e v non sono entrambi marcati, allora marcali e aggiungili a $S := S \cup \{u, v\}$; altrimenti, scarta l'arco.

21. [Approssimazione per MAX-SAT] Per il problema MAX-SAT della soddisfacibilit di una formula booleana, si consideri il seguente algoritmo di approssimazione per massimizzare il numero di clausole soddisfatte in una data formula: Sia F la formula data, x_1, x_2, $\ldots$, x_n le variabili booleane in essa contenute, e c_1, c_2, $\ldots c_m$ le sue clausole. Scegli i valori booleani casuali b_1, b_2, $\ldots$, b_n, ossia ciascun $b_i \in \{0, 1\}$ $(1 \leq i \leq in)$. Calcola il numero m_0 di clausole soddisfatte dall'assegnamento tale che $x_i := b_i$ $(1 \leq i \leq n)$. Calcola il numero m_1 di clausole soddisfatte dall'assegnamento tale che $x_i := \bar{b_i}$ $(1 \leq i \leq n)$, dove $\bar{b_i}$ indica la negazione di b_i. Se $m_0 > m_1$,

restituisci l'assegnamento $x_i := b_i$ $(1 \leq i \leq n)$; altrimenti, restituisci l'assegnamento $x_i := \bar{b_i}$ $(1 \leq i \leq n)$.

Dimostrare che il suddetto algoritmo è una r-approssimazione per MAX-SAT, indicando anche il valore di $r > 1$ (e motivando l'utilizzo di tale valore). Discutere se, in generale, la scelta di $b_1, b_2, \ldots, b_n$ possa influenzare o meno il valore di r, motivando le argomentazioni addotte. Facoltativo: creare un'istanza di MAX-SAT in cui il suddetto algoritmo ottiene un costo che è r volte più piccolo del costo ottimo per una data scelta dei valori di $b_1, b_2 \ldots, b_n$.

22. [Approssimazione per MAX-CUT] Il problema MAX-CUT è NP-hard ed è definito come segue per un grafo non orientato $G = (V, E)$. Una partizione di nodi $(C, V - C)$ con $C \subseteq V$ si chiama "cut" o *taglio*. Un arco $e = (v, w)$ con $v \in C$ e $w \in V - C$ si chiama arco di taglio (ricordando che (v, w) e (w, v) denotano lo stesso arco in un grafo non orientato). Il numero di archi di taglio definisce la dimensione del cut $(C, V - C)$. Poiché cambiando taglio, può cambiare la sua dimensione, il problema richiede di trovare il taglio di dimensione *massima* e quindi gli archi di taglio corrispondenti. Dimostrare che il seguente algoritmo randomizzato è una 2-approssimazione in valore atteso, ossia che il numero medio di archi di taglio così individuati è in media almeno la metà di quelli del taglio massimo. *(1)* Per ogni nodo $v \in V$, lancia una moneta equiprobabile: se viene testa, inserisci v in C; altrimenti (viene croce), inserisci v in $V - C$. *(2)* Inizializza T all'insieme vuoto. Per ogni arco $(v, w) \in E$, tale che $v \in C$ e $w \in V - C$, aggiungi (v, w) all'insieme T. Restituisci C e T come soluzione approssimata.