Esercizi per il Corso di Algoritmica 2 (a.a. 2011/12)

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

5 dicembre 2011

Sommario

Vengono raccolti i problemi discussi il lunedì e collegati agli argomenti delle lezioni del martedì e del mercoledì della settimana precedente. 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 mentale che conduce alla soluzione (piuttosto che la soluzione stessa), sotto la guida del docente in base alla reazione dei partecipanti. La soluzione non viene qui fornita per i motivi suddetti: è preferibile venire a ricevimento dal docente.

1. [Codifica implicita di bit] Consideriamo un array A di n interi *ordinati*, dove n è pari. Mostrare come permutare tali interi in A per codificare implicitamente $n/2$ bit, garantendo che, per $1 \leq i \leq n$, l'elemento che era originariamente in $A[i]$ possa essere recuperato in tempo costante (per esempio, questo è utile nella ricerca binaria, che deve così costare $O(\log n)$ anche se si è permutato A).

2. [Più lungo prefisso quadrato] Data una stringa S, trovare il suo più lungo prefisso che forma un quadrato, ossia trovare x di lunghezza massima (se esiste) per cui $S = xxy$, dove y può essere una stringa vuota mentre x non è vuota.

3. [Velocità funzione di fallimento KMP] Nell'algoritmo semplificato di Knuth, Morris e Pratt, la funzione φ di fallimento calcola il bordo di ciascun prefisso del pattern. Per esempio, con il pattern $P =$ `ananas`, il bordo del prefisso `anan` è $\varphi(4) = 2$ e corrisponde ad `an`: tuttavia, sia `anan` che `an` sono seguite da una `a`, per cui se nel testo la posizione corrispondente contiene un simbolo diverso da `a`, spostare P di 2 posizioni in avanti *sicuramente* non può dare un match. Descrivere come "accelerare" tale funzione φ per evitare situazioni di questo tipo, che sono prevedibili durante il preprocessing di P.

4. [Stringhe di Fibonacci] Definiamo le stringhe di Fibonacci come $F_0 = \texttt{b}$, $F_1 = \texttt{a}$ e $F_k = F_{k-1}F_{k-2}$ per $k \geq 2$, ottenendo così `b`, `a`, `ab`, `aba`, `abaab`, `abaababa`, `abaababaabaab` e così via. Applicare a tali stringhe la funzione di fallimento φ e la sua versione accelerata descritta nel punto precedente, verificandone le proprietà.

5. [Simulazione NFA con broadword e bit parallelism] Simulare l'automa non deterministico ottenuto dalle espressioni regolari mediate la costruzione di Thompson. Usare le tecniche di broadword e bit parallelism in cui una parola di w bit è associata agli stati dell'automa (ipotizzando che vi siano al massimo w stati). In particolare, l'i-esimo bit è 1 se e soltanto se lo stato i è attivo. Valutare il costo delle soluzioni proposte.

6. [Distanza di edit limitata] Considerare il calcolo della distanza di edit tra due stringhe di lunghezza totale n. Volendo verificare soltanto se tale distanza è al più k, fornire un algoritmo che richiede tempo $O(nk)$ e si basa sulla costruzione parziale della tabella di programmazione dinamica.

7. [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).

8. [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.

9. [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.

10. [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.

11. [BWT e ordinamento suffissi] La trasformata di Burrows-Wheeler (BWT) di una stringa di n simboli (incluso il terminatore \$ di fine stringa) richiede che venga ordinata la matrice $n \times n$ ottenuta mediante n shift circolari della stringa. Collegare questo compito al problema dell'ordinamento dei suffissi della stringa.

12. [Inserimento incrementale di suffissi] Ancora LZ77: occorre effettuare l'inserimento incrementale dei suffissi per trovare la suddivisione in frasi del testo. Discutere come

utilizzare i q-grammi per risolvere il problema e il relativo costo computazionale. Progettare algoritmi per l'inserimento incrementale dei suffissi in un array ordinato (suffix array) o in un trie (suffix trie), focalizzando i colli di bottiglia computazionali.

13. [Longest common prefix] Sia SA il suffix array di un testo T di lunghezza n. Per due stringhe x e y, definiamo $lcp(x, y) = \max\{\ell \geq 0 \mid x[1..\ell] = y[1..\ell]\}$ come la lunghezza del loro prefisso comune più lungo. L'array LCP contiene, per $1 \leq i \leq n-1$, la lunghezza $LCP[i] = lcp(T[SA[i]..n], T[SA[i+1]..n])$ del prefisso comune più lungo tra il suffisso in posizione i del suffix array SA e il suffisso nella posizione successiva $i+1$ di SA. Mostrare come modificare la costruzione di SA in modo che si possa calcolare anche LCP con la stessa complessità asintotica in tempo.

14. [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.

15. [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.

16. [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.

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

18. [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.

19. [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.

20. [Paginazione LRU e OPT] Paginazione mediante gli algoritmi LRU e OPT con memoria k. Fornire due sequenze (potenzialmente di lunghezza infinita) con almeno $k+1$ elementi distinti: una che rende uguali i costi di LRU e OPT e una che massimizza il

costo di LRU rispetto a quello di OPT (indicando quale è il fattore di costo raggiunto in tal caso).

21. [Paginazione MARKING e OPT] Paginazione mediante gli algoritmi MARKING e OPT con memoria k. Fornire due sequenze (potenzialmente di lunghezza infinita) con almeno $k + 1$ elementi distinti: una che rende uguali i costi di MARKING e OPT; una che massimizza il costo di MARKING rispetto a quello di OPT (l'ideale sarebbe considerare il costo medio di MARKING, ma va bene mostrare anche cosa succede per alcune scelte randomiche di MARKING).

22. [Selezione randomizzata] Il k-esimo elemento in un insieme non ordinato è l'elemento e tale che ci sono $k - 1$ elementi minori di e nell'insieme. Descrivere e analizzare una variante del quicksort randomizzato che permette la selezione del k-esimo elemento in tempo medio lineare.

23. [Cancellazione da skip list] Descrivere e analizzare l'operazione di cancellazione di una chiave nella struttura di dati skip list.

24. [Cancellazione da cucù hashing] Descrivere e analizzare l'operazione di cancellazione di una chiave nel cucù hashing.

25. [6-Colorazione di un grafo planare] Utilizzando la proprietà di Eulero che ogni grafo planare ha almeno un vertice di grado al più cinque, progettare un algoritmo efficiente per una 6-colorazione di un grafo planare. (Per i temerari: provare anche a trovare una 5-colorazione; la 4-colorazione deriva dal noto teorema dei 4 colori.)

26. [Riduzione polinomiale da HAM a SAT] Il problema del ciclo hamiltoniano (HAM) per un grafo G chiede se è possibile trovare un ciclo in G che attraversa i suoi vertici una e una sola volta. Riduzione HAM al problema del commesso viaggiatore (TSP).

27. [Riduzione polinomiale da ciclo HAM a cammino HAM] Dimostrare che il seguente problema del cammino hamiltoniano è NP-completo: dato un grafo $G = (V, E)$ stabilire se esiste un cammino che attraversa tutti i vertici in V una e una sola volta (e si assuma $|V| > 2$).

28. [Riduzione polinomiale da CLIQUE a SAT] Il teorema di Cook implica che ciascun problema decisionale della classe NP possa essere formulato come un'opportuna formula booleana da soddisfare (SAT). Si consideri il seguente problema CLIQUE: dato un grafo non-orientato $G = (V, E)$ e un intero k, stabilire se G contiene una clique di taglia k. Una clique è un sottoinsieme $C \subseteq V$, tale che ogni coppia di vertici $u, v \in C$ risulta collegata da un arco $(u, v) \in E$. Descrivere una formula booleana F che, costruita a partire da G e k, è soddisfacibile se e solo se G contiene una clique di taglia k. (Nota: in sostanza, si chiede di descrivere una trasformazione polinomiale da CLIQUE a SAT.)

29. [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, ..., x_n le variabili booleane in essa contenute, e c_1, c_2, ... c_m le sue clausole. Scegli i valori booleani casuali b_1, b_2, ..., 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, ..., 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 ..., b_n.

30. [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.