ESERCITAZIONI DI *INTRODUZIONE AGLI ALGORITMI* (A.A. 09/10, CANALE E-O)

APPUNTI ESERCITAZIONE N. 8

Esercizio 1 Definiamo la distanza tra due interi a, b come il valore assoluto della loro differenza, i.e., $d(a, b) = |a - b|$.

a Dato un array di n interi $A[1, \ldots, n]$, restituire il valore minimo della distanza tra due elementi in A, i.e., restituire

$$m = \min\{d(A[i], A[j]), 1 \leq i < j \leq n\}.$$

b Lo stesso di sopra ma partendo da un albero binario di ricerca con n nodi e chiavi intere.

Traccia di soluzione Un approccio per forza bruta costa $\Omega(n^2)$ (calcolare tutte le distanze e confrontarle). L'osservazione fondamentale per risparmiare tempo è questa: se il vettore è ordinato, allora basta calcolare le distanze tra elementi adiacenti. Infatti vale

$$a < b < c \Rightarrow d(a, b) \leq d(a, c).$$

Per il punto (a), ordiniamo il vettore (costo $O(n \cdot \log(n))$) e lo scandiamo calcolando le distanze degli elementi adiacenti e mantenendo il minimo (costo $O(n)$). Il primo costo è dominante. Per il punto (b), visitiamo i nodi dell'albero in ordine crescente e confrontiamo le distanze tra elementi adiacenti in questa visita, mantenendo il minimo. Il costo è $O(n)$.

Esercizio 2 Diciamo che una matrice $n \times n$ è strutturata se contiene interi tutti distinti e in ogni sua riga i valori sono in ordine strettamente crescente da sinistra a destra, e in ogni sua colonna i valori sono in ordine strettamente crescente dal basso verso l'alto. Data una matrice strutturata e un intero k, decidere se k appare nella matrice.

Traccia di soluzione L'approccio per forza bruta costa $\Omega(n^2)$ (visitare tutte le celle della matrice). Possiamo risparmiare con un approccio ricorsivo in cui ad ogni passo l'esito di un confronto ci permette di eliminare dallo spazio di ricerca una porzione della matrice, riducendoci ad una matrice strettamente più piccola. Confrontiamo k con l'elemento nell'angolo sud-est della matrice (al primo passo è $A[n][n]$). Se k è strettamente minore di questo elemento, allora possiamo escludere che k occorra nell'ultima colonna (al primo passo la colonna n-esima della matrice). In questo caso richiamiamo la procedura sulla sottomatrice ottenuta rimuovendo l'ultima colonna (al primo passo $A[1, \ldots, n][1, \ldots, n-1]$). Se invece k è strettamente maggiore di questo elemento, allora possiamo escludere che k occorra nell'ultima riga (al primo passo la riga n-esima). In questo caso richiamiamo la procedura

Note preparate da Lorenzo Carlucci, carlucci@di.uniroma1.it.

sulla sottomatrice ottenuta rimuovendo l'ultima riga (al primo passo è $A[1,\ldots,n-1][1,\ldots,n]$). Il costo è lineare in n (nel caso peggiore facciamo $2n+1$ chiamate).

Esercizio 3 Dato un array di n interi $A[1,\ldots,n]$, diciamo che A è *in crescita* se il suo valore di testa è strettamente inferiore al suo valore di coda, i.e., se $A[1] < A[n]$. Dato un array $A[1,n]$ in crescita, restituire un indice $i \in \{1,\ldots,n-1\}$ tale che $A[i] < A[i+1]$.

Traccia di soluzione Innanzitutto occorre osservare (o dimostrare) che esiste sempre un tale indice. La seconda osservazione è che, se spezziamo in due un array in crescita, allora una delle due metà soddisfa ancora la proprietà di essere in crescita. Per esempio sia $m = (1+n)/2$. Allora

$$A[1,m] \text{ è in crescita, oppure } A[m,n] \text{ è in crescita.}$$

Altrimenti valgono sia $A[1] \geq A[m]$ che $A[m] \geq A[n]$ e dunque $A[1] \geq A[n]$. Ma questo contraddice l'ipotesi che A è in crescita ($A[1] < A[n]$). Ovviamente l'argomento vale per qualunque array in crescita e qualunque modo di spezzarlo in due parti. Abbiamo allora le basi per una soluzione ricorsiva. Ad ogni passo spezziamo l'array corrente, sia $A[t,s]$ (sottoarray dell'array iniziale $A[1,n]$) in due metà, $A[t,(t+s)/2]$ e $A[(t+s)/2,s]$. Se $A[t] < A[(t+s)/2]$, allora richiamiamo la procedura su $A[t,(t+s)/2]$. Altrimenti (e necessariamente vale $A[(t+s)/2] < A[s]$) richiamiamo la procedura su $A[(t+s)/2,s]$. Il costo è $O(\log(n))$.

Esercizio 4 Dati k vettori ordinati $V^1,\ldots,V^k$ di n elementi, fonderli in un vettore ordinato contenente tutti e soli gli elementi di $V^1,\ldots,V^k$.

Traccia di soluzione La soluzione banale consiste nel copiare uno per uno i vettori in un vettore di dimensione $k \cdot n$ e poi ordinare questo vettore. La copia costa kn e l'ordinamento costa $O(kn \cdot \log(kn))$, e quest'ultimo costo domina. Una soluzione più efficiente si ottiene generalizzando il *Merge*. Vogliamo fondere k sequenze ordinate, anziché due sequenze. Ad ogni passo scegliamo il minimo tra le teste dei k vettori. Questo costa $O(k)$ con una ricerca lineare, e va ripetuto per quanti sono gli elementi, ossia kn. Il costo è allora $O(k^2 n)$. Una soluzione più efficiente si ottiene usando una struttura che ci permette di trovare il minimo delle teste dei k vettori in tempo logaritmico, ossia uno heap. Costruire uno heap di dimensione k costa $O(k)$, ma basta farlo una volta. Inizializziamo lo heap con i minimi dei k vettori (le teste), e usiamo lo heap per selezionare il minimo (in tempo $O(\log(k))$). Il minimo viene cancellato e inserito nel vettore ordinato in output. Ogni volta che estraiamo il minimo dallo heap dobbiamo scorrere lungo il vettore da cui quel minimo proviene: se tale vettore è non-vuoto, inseriamo la sua nuova testa nello heap (l'inserimento costa $O(\log(k))$. Questa operazione va ripetuta per ogni elemento. Nello heap dobbiamo mantenere l'informazione seguente, per ogni elemento: il valore, l'indice del vettore di provenienza (tra 1 e k), e la posizione dell'elemento nel vettore di provenienza (così da scorrere correttamente lungo ogni vettore). Il costo complessivo è dato da $O(k) + O(kn\log(k))$, i.e., $O(kn\log(k))$.

Esercizio 5 Dati due alberi binari di ricerca T_1, T_2, fonderli in un albero binario di ricerca contenente tutte e sole le chiavi di T_1, T_2. Discutere il caso in cui tutte le chiavi di T_1 sono minori di tutte le chiavi di T_2.

Traccia di soluzione Una prima soluzione consiste nell'inserire i nodi di T_2 uno per uno in T_1. Ogni inserimento ha costo proporzionale all'altezza dell'albero in cui avviene l'inserimento. Questa altezza, nel caso peggiore, aumenta di 1 ad ogni inserimento. Per tanto – denotando con $h(T)$ l'altezza e con $d(T)$ il numero di nodi di un albero T – il costo della procedura è

$$h(T_1)+(h(T_1)+1)+\cdots+h(T_1)+d(T_2) = \sum_{i=0}^{d(T_2)} h(T_1)+i = d(T_2)\cdot h(T_1)+O(d(T_2)^2).$$

Una soluzione alternativa è la seguente, in cui costruiamo dai due alberi un vettore ordinato e da questo un nuovo albero. Fondiamo i due alberi scorrendoli in parallelo, un passo alla volta, usando la funzione successore (che, dato un nodo v dell'albero restituisce un nodo nell'albero con minima chiave $\geq$ alla chiave di v). Ogni operazione di successore ha costo proporzionale all'altezza dell'albero. Ad ogni passo confrontiamo i due valori correnti (uno estratto da T_1 e l'altro da T_2), inseriamo il minimo in un nuovo vettore di dimensione $d(T^1)+d(T^2)$, e continuiamo la scansione degli alberi. La procedura di scansione costa $h(T_1)d(T_1)+h(T_2)d(T_2)$. Gli inserimenti nel vettore aggiungono un addendo trascurabile $d(T_1)+d(T_2)$. Dal vettore ordinato contenente tutte le chiavi di T_1 e di T_2 possiamo ora costruire un albero binario di ricerca con un approccio divide et impera. Ad ogni passo scegliamo come radice il punto centrale dell'array, e richiamiamo la procedura ricorsivamente su entrambe le due metà per costruire i sottoalberi sinistro e destro. Questa procedura costa $T(n) = 2(T(n/2)) + O(1)$, e dunque $\Theta(d(T_1) + d(T_2))$. Il costo complessivo è pertanto

$$O(h(T_1)d(T_1) + h(T_2)d(T_2)) + \Theta(d(T_1) + d(T_2)) = O(h(T_1)d(T_1) + h(T_2)d(T_2)).$$

Vale sempre $h(T) \leq d(T)$, e nel caso di alberi bilanciati $h(T) = O(\log(d(T)))$. Supponendo che $d(T_1) \geq d(T_2)$, il costo di sopra è $O(h(T_1)d(T_1))$, che nel caso pessimo può essere come $O(d(T_1)^2)$ ma che nel caso bilanciato è $O(\log(d(T_1))d(T_1))$.

(Osservazione) La stima di $O(h(T)d(T))$ per il costo di una visita in ordine di un albero binario di ricerca T usando la funzione di successore è una sovrastima. Dimostrare che il costo è $\Theta(d(T))$. Qual è il costo complessivo dell'algoritmo di fusione sopra delineato?

Infine, se tutte le chiavi di T_1 sono minori delle chiavi di T_2, allora basta appendere la radice di T_1 come figlio sinistro del minimo elemento di T_2. Il minimo elemento di un albero binario di ricerca si trova facilmente in tempo $O(h(T_2))$.

Esercizio 6 Diciamo che un indice i in un array A è un auto-indice se vale $i = A[i]$. Dato un array A ordinato di interi ordinato in modo crescente, decidere se contiene un auto-indice.

Traccia di soluzione Basta osservare che un confronto ci permette di ridurre lo spazio di ricerca in modo considerevole. Se vale $A[i] > i$, allora $A[i]$ non è un auto-indice e nessun elemento a destra di $A[i]$ in A è un auto-indice. Infatti vale $A[i+1] \geq A[i]+1 > i+1$ e dunque $A[i+1]$ non è un auto-indice. Se vale $A[i] < i$, allora $A[i]$ non è un auto-indice e nessun elemento a sinistra di $A[i]$ in A è un auto-indice. Vale $A[i-1] \leq A[i]-1 < i-1$ e dunque $A[i-1]$ non è un auto-indice. Per risolvere il problema usiamo allora una variante della ricerca binaria. Ad ogni

passo l'algoritmo si chiede se il centro del vettore è un auto-indice. Se la risposta è no, allora ci si riduce a una metà del vettore. Il costo risultante è $O(\log(n))$.