

Esercizio 2.

Siano dati due alberi binari di ricerca, tali che le chiavi di un albero siano tutte minori delle chiavi dell'altro.

Definire un algoritmo per concatenare i due alberi in un unico albero binario di ricerca tale che, se h è l'altezza dell'albero più alto, l'algoritmo abbia tempo di esecuzione $O(h)$ nel caso peggiore.

Soluzione

Siano T_1 e T_2 i due alberi binari di ricerca da concatenare. Nel caso in cui le chiavi di T_1 siano tutte minori delle chiavi di T_2 , appendiamo T_2 come sottoalbero destro del massimo di T_1 , altrimenti appendiamo T_1 come sottoalbero destro del massimo di T_2 . In entrambi i casi, la proprietà di ricerca nell'albero risultante viene mantenuta.

Poiché le chiavi di uno dei due alberi sono tutte minori delle chiavi dell'altro, per capire in quale caso ci si trova, è sufficiente confrontare le radici dei due alberi. Il problema quindi si riduce alla ricerca del massimo in uno dei due alberi.

In un albero binario di ricerca, la ricerca del massimo richiede tempo non superiore all'altezza dell'albero. Nel caso peggiore potrebbe essere eseguita sull'albero più alto. Tutte le altre operazioni hanno tempo costante. Il tempo per la concatenazione è quindi $O(h)$, con $h = \max\{\text{altezza}(T_1), \text{altezza}(T_2)\}$.

Esercizio 3

Dato un insieme di $n > 0$ elementi, non necessariamente distinti tra di loro, l'elemento di maggioranza è l'elemento tra questi, se esiste, che si ripete per un numero maggiore di $n/2$ volte. Immaginiamo che l'insieme sia formato da numeri interi positivi, e che sia memorizzato in un array.

1. Tratteggiare un semplice algoritmo che permetta di stabilire se esiste un elemento di maggioranza e in caso affermativo ne trovi il valore, e si valuti la sua complessità asintotica.

(Suggerimento: ordinare l'array e scanderlo sequenzialmente ...).

2. Nell'ipotesi che l'array non possa essere modificato (né copiato in un altro array modificabile...) tratteggiare un algoritmo di complessità $n \log n$, basato su una strategia *divide et impera*, per risolvere lo stesso problema.

3. Tratteggiare un algoritmo di complessità lineare per risolvere lo stesso problema.

4. Per ogni algoritmo descritto, fornire una spiegazione il più possibile semplice e convincente della sua correttezza.

Soluzione

1. Si ordina l'array (in tempo $\theta(n \log n)$) e con una scansione sequenziale si cerca il più lungo segmento di valori consecutivi uguali; se tale segmento ha lunghezza $> n/2$ allora esso contiene l'elemento di maggioranza, altrimenti tale elemento non esiste, si restituisce "nessun elemento". La complessità è quindi $\theta(n \log n)$.

2. L'algoritmo divide et impera divide ripetutamente l'array per due chiamandosi ricorsivamente su ognuna delle due metà, in modo analogo all'algoritmo di mergesort. Al livello di segmenti di un unico elemento restituisce il suo valore come elemento di maggioranza di quel segmento. Quando avvengono due chiamate ricorsive vengono esaminati i valori restituiti. **È importante notare che se esiste un elemento di maggioranza del segmento di array intero esso lo è anche in almeno una delle due metà in cui il segmento è stato diviso.** Ci sono quindi quattro casi

I. entrambe le chiamate ricorsive restituiscono "nessun elemento di maggioranza": allora viene restituito lo stesso valore;

II. il segmento sinistro ha un elemento di maggioranza e quello destro no: si verifica se l'elemento di maggioranza trovato lo è anche per il segmento combinato dei due (si noti che ciò si può fare in tempo lineare); se lo è si restituisce il suo valore, altrimenti "nessun elemento";

III. caso simmetrico al precedente (il segmento destro ha un elemento di maggioranza, quello sinistro no): si agisce in modo analogo;

IV. entrambe le chiamate ricorsive restituiscono un elemento: si verifica (in tempo lineare) se uno dei due è elemento di maggioranza del segmento combinato; se sì, si restituisce il suo valore, altrimenti "nessun elemento".

La complessità $T(n)$ di questo algoritmo soddisfa le seguenti equazioni:

$$T(1) = c$$

$$T(n) = 2 \cdot T(n/2) + 2 \cdot n$$

quindi l'algoritmo è $\Theta(n \log n)$.

3. Si scandisce l'array sequenzialmente dall'inizio alla fine (con complessità lineare), mantenendo due variabili: un valore candidato a essere quello di maggioranza e un contatore intero. Inizialmente il contatore è nullo e il candidato non è assegnato. A ogni passo della scansione,

- se il contatore è 0 esso viene posto a 1 e il valore corrente dell'array diventa candidato;
- altrimenti se il contatore è >0 viene incrementato o decrementato di 1 a seconda che il valore corrente nell'array sia uguale al candidato o diverso da esso.

Al termine della scansione si verifica, con una seconda scansione sequenziale (anch'essa a complessità lineare), se il valore del candidato è effettivamente elemento di maggioranza (nel qual caso lo si restituisce come risultato) altrimenti si restituisce "nessun elemento".

Si noti che se l'array possiede un elemento di maggioranza il valore del candidato al termine della prima scansione è certamente uguale a esso; se invece non esiste un elemento di maggioranza può accadere che alla fine della scansione il contatore sia maggiore di 0; il valore del candidato viene in ogni caso verificato con la seconda scansione.

