Politecnico di Milano
Facoltà di Ingegneria dell'Informazione
Informatica 3
Proff. Campi, Ghezzi, Matera e Morzenti
Appello del 14 Settembre 2006
Recupero II Parte

COGNOME E NOME (IN STAMPATELLO)

MATRICOLA

Risolvere i seguenti esercizi, scrivendo le risposte ed eventuali tracce di soluzione negli spazi disponibili.
Non consegnare altri fogli.

Spazio riservato ai docenti

Esercizio 1.

Si definisca la codifica di Huffman di un set di caratteri con le seguenti frequenze relative:

A: 100
E: 89
D: 67
C: 45
F: 23
G: 12
H: 9

Soluzione
A 11
E 10
D 00
C 011
F 0101
H 01000
G 01001

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 T1 e T2 i due alberi binari di ricerca da concatenare. Nel caso in cui le chiavi di T1 siano tutte minori delle chiavi di T2, appendiamo T2 come sottoalbero destro del massimo di T1, altrimenti appendiamo T1 come sottoalbero destro del massimo di T2. 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{altezza(T1), altezza(T2)}.

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 scandirlo sequenzialmente ...).

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

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

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

Soluzione

1. Si ordina l'array (in tempo $\theta(n \cdot \log n)$) 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 $\theta(n \cdot \log n)$.

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 $\Theta(n \cdot \log n)$.

3. Si scandisce l'array sequenzialmente dall'ínizio 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;
- 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.

Politecnico di Milano
Facoltà di Ingegneria dell'Informazione
Informatica 3
Proff. Campi, Ghezzi, Matera e Morzenti
Appello del 14 Settembre 2006
Recupero I Parte

COGNOME E NOME (IN STAMPATELLO)

MATRICOLA

Risolvere i seguenti esercizi, scrivendo le risposte ed eventuali tracce di soluzione negli spazi disponibili.
Non consegnare altri fogli.

Spazio riservato ai docenti

Esercizio 1

Si consideri un linguaggio di programmazione L eseguito da un computer dedicato (il computer esegue un singolo programma L alla volta). Viene rilevato il seguente fatto che appare inizialmente inspiegabile: due chiamate successive a una stessa funzione (che non effettua operazioni di I/O), con gli stessi valori dei parametri e delle variabili non locali, impiegano tempi di esecuzione diversi. Dopo qualche tempo, uno dei progettisti attribuisce questo fenomeno al fatto che L alloca dati nella memoria heap e impiega un garbage collector. È giustificata questa affermazione del progettista? Perchè si o no?

Soluzione

Si. Potrebbe accadere che alla seconda chiamata di funzione la memoria occupata dal programma la memoria heap sia tutta riempita e debba essere attivato il garbage collector per poter procedere con l'esecuzione. Pertanto non vengono eseguite solo le operazioni della funzione, ma anche quelle del garbage collector.

Esercizio 2

Si consideri un linguaggio che adotta regole di visibilità dinamica (dynamic scope). Per accedere a una variabile di nome XXX, l'interprete della macchina SIMPLESEM chiama una funzione `fps(CURRENT, XXX)` che opera sulla memoria `D` e che ha il compito di calcolare l'indirizzo di base (indice della memoria `D`) dove è memorizzato il record di attivazione corrispondente alla più recente chiamata attiva di una funzione nella quale è dichiarata `XXX` e che ha portato all'attivazione della funzione in esecuzione.
Esprimere `fps` come funzione ricorsiva, utilizzando la funzione di servizio `isLocal (p, y)` che calcola valore vero se la variabile `y` è presente nel record di attivazione il cui indirizzo di base è `p`.
Nello scrivere la funzione, si definisca esplicitamente come si assume che sia strutturato il record di attivazione, e cioè a quale offset siano allocate le varie grandezze (return point, link dinamico, ecc.)

Soluzione
Ipotizzando che a offset 0 sia memorizzato il return point e a offset 1 il link dinamico, la funzione deve discendere lungo la catena dinamica. Dunque:

```
if isLocal(CURRENT, XXX) then CURRENT
    else fps(D[CURRENT+1], XXX)
```

Esercizio 3

Si consideri un programma concorrente nel quale due task possono eseguire concorrentemente un frammento che contiene le seguenti istruzioni

```
x = x+y;
y = x-y;
```

Si ipotizzi che ciascuna delle due istruzioni venga eseguita come una singola operazione atomica della macchina virtuale, mentre invece non esista mutua esclusione nell'esecuzione della sequenza. Ipotizzando che inizialmente x valga 10 e y valga 5, spiegare che cosa potrebbe succedere dopo che i due task hanno eseguito il frammento in maniera concorrente.

Soluzione

Si considerino due task, A e B. Nel caso in cui prima esegua A e poi B si ottiene x=25 e y=15. Nel caso in cui A esegua la prima istruzione e poi B esegua la prima istruzione, alla fine si ottiene x=25 e y=5. Negli altri mix di esecuzione tra A e B si ottiene uno dei due risultati citati.