Informatica 3 – Proff. Ghezzi e Morzenti – Prova del 4 luglio 2003

Cognome e Nome ..... Numero Matricola .....
(in stampatello)

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

NON CONSEGNARE ALTRI FOGLI. (spazio per i docenti)

Esercizio 1

Rispondere alle seguenti domande:

a) Le operazioni di ricerca di un elemento e di inserimento di un elemento in un albero di ricerca binario (BST) hanno la stessa complessità asintotica. E vero o falso? Motivare sinteticamente la risposta, indicando anche la funzione di complessità asintotica per ognuna delle due operazioni.

[R. SI. La complessità è lineare nella profondità dell'albero e quindi, nel caso pessimo, nel numero dei nodi. Occorre attraversare lo stesso numero di nodi per trovare il dato o per determinare dove il nuovo valore debba essere inserito.]

b) Si consideri l'operazione di ricerca di un elemento in un albero di ricerca binario (BST). Trovare due significative funzioni f e g tali che la complessità temporale dell'operazione sia $O(f(n))$ e $\Omega(g(n))$ in funzione del numero n di nodi dell'albero, possibilmente indicando a quali casi particolari corrispondono tempi di ricerca uguali a $f(n)$ e $g(n)$.

[R. f(n)=n e g(n)=log n, corrispondenti alla ricerca in un albero totalmente sbilanciato (di forma lineare) o bilanciato.]

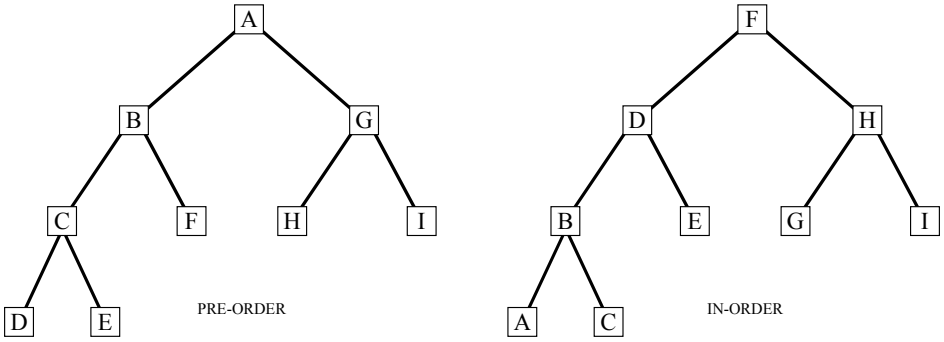
Esercizio 2

Si consideri la seguente stringa, che rappresenta la sequenza dei nodi visitati in pre-ordine per un albero binario ***completo***

A B C D E F G H I

a) Si può ricostruire univocamente l'albero binario?
b) Si disegni l'albero binario (o uno dei possibili alberi binari) corrispondenti alla stringa precedente.
c) Si risponda alla domanda precedente supponendo che la stringa rappresenti la visita in-order.
d) Come cambia la risposta alla domanda (a) nel caso di albero ***pieno*** (e non necessariamente completo)?

[R. L'albero è determinato univocamente per il caso completo, non per il pieno



]

Esercizio 3

Quale delle seguenti operazioni, da effettuare su una sequenza di *n* numeri (assunti per ipotesi tutti diversi) memorizzati in un array, può essere implementata in modo efficiente eseguendo come primo passo un ordinamento dell'array (NB nella valutazione del costo dell'implementazione deve essere incluso anche il costo dell'eventuale ordinamento)?

a) trovare il valore più vicino al valor medio

b) trovare un valore mediano (un valore mediano è uno dei valori memorizzati nell'array, tale che il numero di valori minori di esso e il numero di valori maggiori differiscono al più di uno)

Per ciascuna operazione codificare un algoritmo efficiente (possibilmente ottimale) implementando i metodi Java sotto elencati (nel caso sia opportuno ordinare preventivamente l'array allora la prima istruzione del metodo dovrà essere una chiamata di un metodo di ordinamento ottimale, a esempio heapSort(ar)). In corrispondenza a ogni implementazione indicare la complessità temporale asintotica dell'algoritmo adottato come funzione di *n*.

float vicinoAlValorMedio (float [] ar)

float mediano (float [] ar)

(NB: per codificare i due metodi non è necessario usare esattamente la sintassi e le notazioni di Java, è sufficiente utilizzare uno pseudocodice sufficientemente chiaro e non ambiguo.)

Indicare inoltre, fornendo anche una sintetica spiegazione, quale potrebbe essere la complessità temporale asintotica di un algoritmo che risolva gli stessi problemi proposti, ma sotto l'ipotesi, più restrittiva, che l'array contenente i valori numerici sia già ordinato.

[R. Sì per (b), ma non per (a).

```
float vicinoAlValorMedio (float [] ar) {
    float m=0;    float diff, minDiff;    int i, minIndex;
    for (i = 0; ar.length; i++) m += ar[i];
    m = m / ar.length;
    minDiff = diff = abs(ar[0]-m);  minIndex = 0;
    for (i = 1; ar.length; i++) {
        diff = abs(ar[i]-m);
        if (diff < minDiff) { minDiff = diff; minIndex = i; }
    }
    return ar[minIndex];
}

float mediano (float [] ar) { heapSort(ar); return ar[(ar.length+1)/2]; }
```

la complessità di *vicinoAlValorMedio* è $\Theta(n)$, quella di *mediano* è $\Theta(n \log n)$.

Partendo da un array ordinato, la complessità sarebbe lineare per il problema (a) (occorre comunque calcolare la media) e costante per il problema (b) (basta prendere il valore al centro dell'array).

]

Esercizio 4

Si indichi, motivando brevemente la risposta, in quali tra i casi seguenti sia preferibile la codifica di un grafo orientato G = (V, E) mediante matrice delle adiacenze o liste delle adiacenze. (Nota: Nel seguito due vertici *u* e *v* si dicono adiacenti se sono collegati da un arco, cioè se $(u, v) \in E$; si assuma inoltre che nei grafi considerati i vertici siano etichettati con numeri interi compresi tra 0 e |V|-1 e non contengano altra informazione, e gli archi siano distribuiti in modo uniforme tra i vari vertici.)

a) Il grafo ha |V| = 10.000 e |E| = 20.000, ed è prioritario minimizzare l'occupazione di memoria;

b) Il grafo ha |V| = 10.000 e |E| = 40.000.000, ed è prioritario minimizzare l'occupazione di memoria;

c) Il grafo ha |V| = 10.000 e |E| = 60.000.000, ed è prioritario minimizzare l'occupazione di memoria;

d) Il grafo ha |V| = 10.000 e |E| = 20.000.000, ed è prioritario poter determinare velocemente se due vertici sono adiacenti;

e) Il grafo ha |V| = 10.000 e |E| = 10.000.000, ed è prioritario poter determinare velocemente se due vertici sono collegati da un cammino di lunghezza 2 (cioè se esiste un vertice *w* tale che *u* e *w* sono adiacenti e lo sono anche *w* e *v*);

f) Il grafo ha |V| = 10.000 e |E| = 10.000.000, ed è prioritario poter determinare velocemente se due vertici sono collegati da un cammino di lunghezza 3.

Per argomentare la risposta ai punti (e) ed (f) si suggerisce di tratteggiare gli algoritmi considerati e indicarne esplicitamente la complessità temporale.

[R. Assumendo che un'etichetta di tipo int e un puntatore occupino la stessa quantità di memoria, a esempio una cella:

a) conviene la rappresentazione a liste; ($|V|^2$=100.000.000 celle contro |V|+2|E|=50.000)
b) idem c.s. ($|V|^2$=100.000.000 celle contro |V|+2|E|≅80.000.000);
c) conviene la matrice ($|V|^2$=100.000.000 celle contro |V|+2|E|≅120.000.000);
d) conviene la matrice (costo costante);
e) conviene la matrice (costo proporzionale a |V| contro $(|E|/|V|)^2$);

algoritmo per la matrice

```
boolean isEdge(int i, int j) {
    for (k = 0; k < |V|; k++)
        if (M[i,k] && M[k,j])
            return true;
    return false;
}
```

algoritmo per le liste di adiacenza

```
boolean isEdge(int i, int j) {
    for (tutti i vertici k adiacenti a i)
        for (tutti i vertici m adiacenti a k)
            if (m == j)  return true;
    return false;
}
```

f) conviene ancora la matrice (costo proporzionale a |V|+|E| contro $(|E|/|V|)^3$).

algoritmo per la matrice: nota che il ciclo più interno, essendo soggetto alla condizione M[i,k], viene eseguito |E|/|V| volte

```
boolean isEdge(int i, int j) {
    for (k = 0; k < |V|; k++)
        if (M[i,k])
            for (m = 0; m < |V|; m++)
                if (M[k, m] && M[m, j])
                    return true;
    return false;
}
```

algoritmo per le liste di adiacenza

```
boolean isEdge(int i, int j) {
    for (tutti i vertici k adiacenti a i)
        for (tutti i vertici m adiacenti a k)
            for (tutti i vertici n adiacenti a m)
                if (m == j)  return true;
    return false;
}
```

]