Algoritmi e Strutture Dati

Appello del 2 Settembre 2008

Modulo Teoria

Attenzione: Scrivere nome, cognome e numero di matricola sul foglio protocollo.

Esercizio 1 [Punti 10]

Dato un albero binario T i cui nodi contengono chiavi intere, progettare un algoritmo ricorsivo che stabilisce se le chiavi nei nodi di T soddisfano la proprietà di *Heap di massimo* e analizzarne la complessità.

Esercizio 2 [Punti 10]

Data la sequenza di chiavi $S = 7, 2, 6, 413, 19, 32, 9$ da inserire in una tabella hash di dimensione m,

1. mostrate il contenuto della tabella supponendo che le collisioni siano gestite con liste concatenate e che la funzione hash sia $h_1(k) = k \ mod \ m$, con $m = 7$;

2. mostrate il contenuto della tabella supponendo che le collisioni siano gestite con indirizzamento aperto utilizzando lo hash doppio come metodo di scansione, con h_1 definita come sopra, $h_2(k) = [(k \times 7) \ mod \ 3] + 1$ e $m = 11$.

Esercizio 3 [Punti 8]

Dato un grafo orientato memorizzato con le seguenti liste di adiacenza:

$$\begin{array}{lcl} 1 & \rightarrow & 2, 4, 7 \\ 2 & \rightarrow & 6 \\ 3 & \rightarrow & 1 \\ 4 & \rightarrow & 7 \\ 5 & \rightarrow & 2, 6 \\ 6 & \rightarrow & 3, 5 \\ 7 & \rightarrow & 5, 6 \end{array}$$

eseguire una visita DFS a partire dal vertice 1 indicando la sequenza dei vertici incontrati e l'albero DFS.

Esercizio 4 [Punti 2]

Dare le definizioni di albero **1-bilanciato** e albero di **Fibonacci**.

Modulo Laboratorio

Esercizio 1 [Punti 14]

Si dia un'implementazione in codice java dell'algoritmo di ordinamento Merge-Sort.

Esercizio 2 [Punti16]

1. Definire il problema di trovare un albero di copertura minimo, e spiegare a parole l'algoritmo risolutivo che è stato implementato durante il corso.

2. Si commenti con javadoc il seguente codice, che implementa un algoritmo di visita pre-ordine di un albero.

```
void visitaPreOrdine(Node nodo) {

  Pila pila = new PilaArray();

  Node n;

  pila.push(nodo);

  while (!pila.eVuota()) {

    n = (Node) pila.pop();

    if (n != null) {

       visita(n);
```

```
        Pila pilaFigli = new PilaArray();
        Node son = n.figlio;
        while (son != null) {
         pilaFigli.push(son);
         son = son.fratello;
        }
      while (! pilaFigli.eVuota()) {
       pila.push(pilaFigli.pop());
      }
     }
   }
}
```