Compito di esame 8 settembre 08

Tempo: 2h

Problema 1 (6 punti)

Si consideri il metodo visita qui definito

```
static void visita(boolean[][] graph) {
  final int numNodi = graph.length;
  int nodoDaVisitare = 0;
  boolean[] visitato = new boolean[numNodi];
  //INIZIALIZZATO A FALSE AUTOMATICAMENTE
  while (nodoDaVisitare < numNodi) {
      IntQueue daVisitare = new IntQueue(); // CODA DI INTERI
      daVisitare.enqueue(nodoDaVisitare); //COSTO O(1)
      while (!daVisitare.isEmpty()) {
        int nodoCorrente = daVisitare.dequeue(); //COSTO O(1)
        for (int j = 0; j < numNodi; j++)
        if (graph[nodoCorrente][j] && !visitato[j]) {
              visitato[j] = true;
              daVisitare.enqueue(j);
        }
      }
      nodoDaVisitare++;
      while (nodoDaVisitare < numNodi && visitato[nodoDaVisitare]) {
        nodoDaVisitare++;
      }
    }
  }
```

(4 punti) Calcolare il costo computazionale dell'algoritmo descritto dal metodo visita, in funzione dell'ordine della matrice in input.

(2 punti) Esprimere tale costo in funzione della dimensione dell'input.

Soluzione 1

- O(n * n)
  - in funzione del numero di nodi

- O(m)
  - In funzione della dimensione dell'input (m=n*n)

Problema 2 (6 punti)

- Si consideri il problema dell'ordinamento di un insieme di interi.
- **(4 punti)** Con riferimento all'algoritmo QuickSort, scriverne il codice Java e discuterne i costi di esecuzione, sia in termini di tempo che di spazio.
- **(2 punti)** Descrivere il concetto di mediano di un insieme di interi e mostrare come si possa utilizzare un algoritmo di costo lineare per il calcolo del mediano per potenziare il QuickSort. Discutere i costi dell'algoritmo di ordinamento così potenziato.

Problema 3 (6 punti)

- **(2 punti)** Definire le classi Java atte a rappresentare un BST con chiavi di tipo **int**.
- **(4 punti)** Scrivere un metodo Java **potaBST** che riceve in input la radice di un BST e due **int a** e **b**. Il metodo rimuove dall'albero tutti i nodi contenenti una chiave non appartenente all'intervallo [**a**,**b**] e restituisce la radice del BST risultante. Determinare il costo computazionale dell'algoritmo.

Soluzione 3/A

```
public class BSTNode {
    public BSTNode left, right;
    public int key;
    public BSTNode() {
 left = right = null;
 key = 0;
    }
    public int getKey() {
 return key;
    }

    public BSTNode getLeft() {
 return left;
    }

    public BSTNode getRight()
   {
 return right;
    }
}
```

```
public class BST {
    protected BSTNode root;

    public BST() {
 root = null;
    }

    public BST(BSTNode r) {
 root = r;
    }

    public BSTNode getRoot() {
 return root;
    }
}
```

Soluzione 3/B

```
public static BSTNode potaBST(BSTNode r, int a, int b) {
if (r==null)return null;
if (r.key < a) {
    //elimino il nodo e tutti il sottoalbero sinistro
    //(sono sicuramente + piccoli di a)
    return potaBST(r.right, a, b);
}
if (r.key > b) {
    //elimino il nodo e tutti il sottoalbero destro
    // (sono sicuramente + grandi di b)
    return potaBST(r.left, a, b);
}

r.left = potaBST(r.left, a, b);
r.right = potaBST(r.right, a, b);
return r;
   }
```

Problema 4

- Si consideri un albero generico contenente chiavi intere.
- **(2 punti)** Definire le classi Java atte a rappresentare un albero generico.
- **(4 punti)** Scrivere un metodo ricorsivo **sumTree** che, dato un albero generico, restituisca una lista collegata contenente le somme dei valori delle chiavi contenute in ogni sotto-albero dell'albero dato. La lista deve essere ordinata secondo una visita in post-ordine dell'albero ed il costo computazionale deve essere lineare.

Soluzione 4/A

```
public class TreeNode {
    protected int element;
    protected TreeNode firstChild;
    protected TreeNode nextSibling;
    public TreeNode(int element) {
        this.element = element;
    }
    public TreeNode getFirstChild() {
        return firstChild;
    }
    public void setFirstChild(TreeNode firstChild) {
        this.firstChild = firstChild;
    }
    public TreeNode getNextSibling() {
        return nextSibling;
    }
    public void setNextSibling(TreeNode nextSibling) {
        this.nextSibling = nextSibling;
    }
}
```

Soluzione 4/B

```
public static int somma(List<Integer> ris, TreeNode n) {
//controlla tutti i figli...
int somma = 0;
TreeNode child = n.firstChild;
while (child != null) {
    somma += somma(ris, child);
    child = child.nextSibling;//next children
}
//somma dell'elemento del nodo corrente
somma+=n.getElement()
ris.add(somma);
//ritorna la somma
return somma;
}
```

Problema 5 (6 punti)

- **(3 punti)** Con riferimento al concetto di grafo diretto aciclico, illustrare la tecnica di visita in profondità, spiegarne le proprietà e discuterne il costo di esecuzione.
- **(3 punti)** Scrivere lo pseudo-codice dell'algoritmo per l'ordinamento topologico in un grafo diretto aciclico e se ne mostri il funzionamento su un esempio a scelta di almeno 10 nodi con almeno 2 sorgenti e 3 pozzi.

18 luglio 2008

Problema 2 (6 punti)

(2 punti) Definire una rappresentazione Java per un BST, i cui nodi contengono una chiave di tipo int, nonché un campo int fdb inizializzato a Integer.MIN_VALUE (minimo valore possibile per un int).

(4 punti) Sfruttando la rappresentazione proposta, scrivere un metodo Java static int assegnaFdb(root) che,

- data la radice root di un BST, assegni a tutti i campi fdb il valore del fattore di bilanciamento, espresso come
- altezza1 del sottoalbero destro meno quella del sottoalbero sinistro; non vengono definite specifiche sul valore
- restituito dal metodo, valore che può essere dunque sfruttato come meglio si ritiene. Valutare il costo computazionale dell'algoritmo.

Soluzione 2/A

```
public class BSTNode2 {
    public BSTNode2 left, right;
    public int key;
    public int fdb;
    public boolean isLeaf() {
       return left == null && right == null;
    }
    public int getKey() {
       return key;
    }
    public BSTNode2 getLeft() {
       return left;
    }
    public BSTNode2 getRight() {
       return right;
    }
}
```

Soluzione 2/B

```
public static int fdb(BSTNode2 r) {
if (r != null) {
    final int hsx = r.left == null ? 0 :
            fdb(r.left) + 1;
    final int hdx = r.right == null ? 0 :
            fdb(r.right) + 1;
    r.fdb = hdx - hsx;
    return Math.max(hdx, hsx);
}
return 0;
   }
```

Problema 1 (6 punti)

Problema 1 (6 punti) Si consideri l'algoritmo descritto dal metodo `visita` qui sotto riportato.

```
static void visita(Nodo root) {
	if(root == null) return;
	Stack<Nodo> pila = new Stack<Nodo>();
	pila.push(root);
	Nodo finto = new Nodo();
	while(!pila.isEmpty()) {
		Nodo v = pila.pop();
		if(v == finto) {
			v = pila.pop();
			System.out.println(v.info);
		} else {
			pila.push(v);
			pila.push(finto);
			if(v.dx != null) pila.push(v.dx);
			if(v.sx != null) pila.push(v.sx);
		}
	}
}
class Nodo {
	String info;
	Nodo dx, sx;
}
```

a) **(3 punti)** Determinare il miglior upper bound possibile al costo dell'algoritmo.
b) **(3 punti)** L'algoritmo effettua una visita di un albero binario. Di quale tipologia di visita si tratta (ampiezza, pre-ordine, in-ordine, post-ordine, altro) e perché?

Problema 3 (6 punti)

a) (3 punti) Definire il concetto di grafo semplice e, in tale ambito, quello di grafo connesso. Descrivere la rappresentazione di un grafo semplice tramite matrice di adiacenza.

b) (3 punti) Scrivere un algoritmo (Java o pseudo-codice) che, data la matrice di adiacenza di un grafo semplice, calcoli il numero di componenti connesse del grafo. Valutare il costo computazionale dell'algoritmo.

Problema 4 (6 punti)

a) (3 punti) Definire una rappresentazione Java per un file system, costituito da una cartella radice (root) che contiene zero o più file e/o zero o più sottocartelle. Ogni sottocartella, a sua volta, è una cartella che contiene zero o più file e/o zero o più sottocartelle. File e cartelle sono caratterizzati da identificatori (unici nella cartella in cui esistono ma non necessariamente unici nel file system).

b) (3 punti) Scrivere il metodo moveTo(c1, c2) che, dati due riferimenti c1 e c2 a cartelle effettivamente esistenti nello stesso file system, sposti la cartella associata a c1 dalla posizione in cui si trova, riposizionandola all'interno della cartella associata a c2 (la cartella associata a c1 non può essere root).

c) Discutere la complessità computazionale dell'operazione.

Problema 5 (6 punti)

a) (3 punti) Scrivere un algoritmo (Java o pseudo-codice) che, dato un grafo orientato G, determina se G è aciclico o meno e discuterne la sua complessità computazionale.

b) (3 punti) Eseguire visualmente sul grafo in figura i passi dell'algoritmo che effettua il test di aciclicità.