ASD (FI2 12 CFU - tutti), ASD (5 CFU) - APPELLO STRAORDINARIO

Appello del 25-10-2016 – a.a. 2015-16 – Tempo a disposizione: *120 minuti* – somma punti: 32

Problema 1 [Punti: (a) 2/30; (b) 2/30; (c) 3/30]

Si considerino i metodi Java di seguito illustrati.

```
static int[] crea(int n) {
  int[] arr = new int[n]; // assumere new O(1)
  for(int i = 0; i < n; i++)
    arr[i] = (int)(Math.random()*n); // assumere Math.random() O(1)
  return arr;
}

static int processa(int[] a, int i, int j) {
  if(i == j) return a[i];
  int p = a[i];
  if((p >= i) && (p < j)) {
    int q = processa(a, i, p) - processa(a, p+1, j);
    return q >= 0 ? q: -q;
  } else return a[j];
}

static int processa(int[] a) {
  return processa(a, 0, a.length-1);
}

static int tutto(int n) {
  return processa(crea(n));
}
```

Sviluppare, *argomentando adeguatamente* (il 50% del punteggio dell'esercizio sarà sulle argomentazioni addotte), quanto segue:

(a) Determinare il costo asintotico dell'algoritmo descritto da `crea(int)` in funzione della dimensione dell'input (denotata z_1).

(b) Determinare il costo asintotico dell'algoritmo descritto da `processa(int[])` in funzione della dimensione dell'input (denotata z_2).

(c) Determinare il costo asintotico dell'algoritmo descritto da `tutto(int)` in funzione della dimensione dell'input (denotata z).

Problema 2 [Punti: 8/30]

Scrivere un algoritmo (codice Java o C) che determini l'intersezione di due BST dati. Più precisamente, dati due BST t_1 e t_2 contenenti chiavi intere, determinare in tempo/spazio lineare l'insieme delle chiavi presenti in entrambi gli alberi e restituendolo attraverso una lista ordinata.

Problema 3 [Punti: 8/30]

1. Con riferimento al tipo astratto *dizionario ordinato* (chiavi `int`), e alle strutture concrete riportate in tabella, indicare i costi computazionali (temporali) di *caso peggiore* delle operazioni elencate. Usare la notazione Θ. [costo corretto: $+0.15$; errato: -0.05; assente: 0]

	put	remove	get	predecessore	findAll
lista concatenata					
array ordinato					
BST					
AVL					

2. Un array di ℓ elementi è denominato *bitonico* se può essere suddiviso in due sotto-array, di lunghezza rispettivamente ℓ_1 ed ℓ_2, essendo $0 \leq \ell_i \leq \ell$ e $\ell_1 + \ell_2 = \ell$, il primo contenente una sequenza monotona non decrescente, il secondo una sequenza monotona non crescente. Esempi di array bitonici: $\{2, 3, 6, 6, 7, 8, 2, 1, 0\}$; $\{3, 6, 8, 11, 11, 11, 10, 8, 4, 2\}$; $\{4, 6, 8, 10\}$.

   Scrivere un algoritmo (pseudo-codice) che, dato un array bitonico, ne ordina il contenuto in tempo lineare. [3/30]

3. Disegnare (a piacere) un Albero di Fibonacci di altezza 5 (senza dimenticare di esplicitarne le chiavi). [2/30]

Problema 4 `[Punti: (a) 6/30; (b) 3/30]`

Una mappa stradale è rappresentata attraverso un *grafo orientato* $\mathcal{G}$ i cui vertici rappresentano incroci e i cui archi rappresentano tratti stradali fra due incroci. Ad ogni vertice è associata un'etichetta di tipo stringa (`id` del vertice, che lo identifica univocamente); a ciascun arco ne sono associate due di tipo reale: `l` (lunghezza in metri del tratto stradale) e `t` (tempo medio di percorrenza, in base a dati storici precedentemente elaborati).

Il grafo $\mathcal{G}$ deve essere utilizzato per determinare percorsi ottimali fra coppie di vertici, di cui sono forniti gli `id` in input. Per percorso ottimale si intende un percorso di lunghezza minima, oppure con tempo di percorrenza medio minimo: la scelta è operata dall'utente ciascuna volta che chiede il calcolo del percorso ottimale.

Ciò premesso, si richiede di:

(a) Scrivere un algoritmo `minPath` (pseudo-codice) che dati un grafo $\mathcal{G} = (V, A)$, due etichette `s1` ed `s2` (che identificano due vertici in V) e un criterio `p` $\in \{\text{SPAZIO}, \text{TEMPO}\}$, determini e restituisca una coppia $(\pi(\texttt{s1}, \texttt{s2}), \texttt{r})$, essendo $\pi(\texttt{s1}, \texttt{s2})$ il percorso ottimale da `s1` a `s2` ed `r` la sua misura (in termini di distanza o di tempo medio totale, a seconda del valore di `p`).

L'algoritmo deve rispettare le specifiche input/output che sono state assegnate e deve anche mostrare come gestisce l'accesso ai vertici date le loro etichette univoche.

(b) Determinare i costi computazionali di `minPath` (tempo e spazio) nel caso di rappresentazione basata sia su *liste di adiacenza* che su *matrice di adiacenza*. I costi debbono includere quelli per l'accesso ai nodi date le etichette. Nel caso in cui l'algoritmo fornito al passo (a) sia logicamente non corretto il punteggio conseguito al presente passo sarà automaticamente dimezzato.