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

Appello del 9-9-2016 – a.a. 2015-16 – Tempo a disposizione: *120 minuti* – somma punti: 33

Problema 1

`[Punti: (a) 5/30; (b) 2/30]]`

Si considerino i metodi Java di seguito illustrati.

```
static int sum(int[] a, int i, int j) {
    if(i==j) return a[i];
    else if(i<j) return a[i]+a[j]+sum(a, i+1, j-1);
    else return 0;
}

static int balance(int[] a) {
    int diff, best=Integer.MAX_VALUE, ibest=-1;
    for(int i = 0; i < a.length-1; i++) {
        diff = sum(a, 0, i) - sum(a, i+1, a.length-1);
        diff = diff >= 0 ? diff : -diff;
        if(diff < best) {
            best = diff;
            ibest = i;
        }
    }
    return ibest;
}
```

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

(a) Determinare il costo asintotico dell'algoritmo descritto da `balance(int[])` in funzione della dimensione dell'input.

(b) Determinare l'output del metodo `balance(int[])` quando riceve in input l'array `{1, 4, -1, 4}`.

Problema 2

`[Punti: 8/30]`

Scrivere un algoritmo (codice Java o C) che, dato un albero binario i cui nodi contengono interi non negativi, determini in tempo/spazio lineare il più piccolo intero non negativo che non è contenuto nell'albero e lo restituisca.

[In classe è stata data l'opportunità di scegliere fra input costituito da albero binario generico, come detto nel testo del problema, aggiungendo +1 punti, o da BST, togliendo 2 punti. In altre parole, risolvendo correttamente il problema posto si ottengono 9 punti; risolvendo correttamente il problema assumendo l'ipotesi semplificativa ci sia un BST in input si ottengono 6 punti.]

Problema 3

1. Con riferimento al tipo astratto *mappa* (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: +0.2; costo errato: −0.1; costo assente: 0)

	put	remove	get	range query[1]
array disordinato				
array ordinato				
BST				
AVL				
tavola hash (linear probing)				

[1] Dato un intervallo $[a, b]$, restituire una lista delle chiavi che appartengono all'intervallo.

2. Definire il concetto di connessione forte e descrivere (pseudo-codice) un algoritmo efficiente (tempo e spazio) per determinare se un grafo diretto è fortemente connesso. (3 punti)

3. A piacere: disegnare due grafi diretti di 8 nodi ciascuno, entrambi debolmente connessi, ma uno solo fortemente connesso. (2 punti)

Problema 4

`[Punti: (a) 2/30; (b) 4/30; (c) 3/30]`

Il social network LinkedIn permette come è noto di stabilire relazioni (bidirezionali) di amicizia fra i partecipanti al network. Quando un membro visualizza il profilo di un altro membro LinkedIn indica se i due sono amici e l'elenco degli eventuali amici comuni ai due. Se non ci sono neanche amici comuni viene fornita la distanza (intesa come numero di "presentazioni" necessarie per diventare amici). Dato l'alto numero di consultazioni di profili, i gestori della piattaforma vogliono ottimizzare l'algoritmo utilizzato per realizzare le informative di cui sopra.

Ciò premesso, si richiede di:

(a) Formulare il problema posto come un problema su grafi. A tal fine occorre dapprima definire i grafi a cui si farà riferimento, definendone vertici ed archi e chiarendo come essi siano collegati al problema in esame; successivamente va formalizzato il problema in esame come problema su grafi, utilizzando una terminologia appropriata.

(b) Definire un algoritmo (pseudo-codice) per la soluzione generale del problema: dati in input due membri x e y di LinkedIn, con $x \neq y$, fornire in output una coppia, come specificato:

- `(0, <lista_amici_comuni>)` se x e y sono amici;
- `(1, <lista_amici_comuni>)` se x e y non sono amici, ma hanno amici comuni;
- `(<n_presentazioni>, NULL)` se x e y non sono amici e non hanno amici comuni.

Per maggiore chiarezza, si precisa che la coppia in output è composta da numero naturale che esprime il numero di presentazioni necessarie per poter raggiungere lo stato di amicizia (0 nel caso di amicizia già stabilita) e da un riferimento alla lista degli eventuali amici comuni.

(c) Determinare i costi computazionali dell'algoritmo (tempo e spazio) nel caso di rappresentazione basata sia su *liste di adiacenza* che su *matrice di adiacenza*.