Esame di		
	Fondamenti di informatica II - Algoritmi e strutture dati	**12 CFU**
	Algoritmi e strutture dati V.O.	**5 CFU**
	Algoritmi e strutture dati (Nettuno)	**6 CFU**

Appello del 12-6-2017 – a.a. 2016-17 – Tempo a disposizione: *4 ore* – somma punti: 35

Istruzioni

Lanciare la macchina virtuale Oracle VirtualBox e lavorare all'interno della cartella `ESAME`, avendo cura di creare all'interno della cartella stessa:

- un file `studente.txt` contenente, una stringa per riga, cognome, nome, matricola, email; in tutto quattro righe;
- una cartella `java.<matricola>`, o `c.<matricola>`, ove al posto di `<matricola>` occorrerà scrivere il proprio numero di matricola, contenente i file prodotti per risolvere il Problema 2 (in tale cartella si copi il contenuto dell'archivio `c-aux.zip` o `java-aux.zip`);
- tre altri file `probl1.<matricola>.txt`, `probl3.<matricola>.txt` e `probl4.<matricola>.txt`, contenenti, rispettivamente, gli svolgimenti dei problemi 1, 3 e 4.

È possibile consegnare materiale cartaceo integrativo, che verrà esaminato solo a condizione che risulti ben leggibile.

Per l'esercizio di programmazione (Problema 2) è possibile usare qualsiasi ambiente di sviluppo disponibile sulla macchina virtuale. Si raccomanda però di controllare che i file vengano salvati nella cartella `java.<matricola>`, o `c.<matricola>`. Si consiglia inoltre per chi sviluppa in `c` di compilare da shell eseguendo il comando `make` e poi eseguire `driver` per verificare la correttezza dell'implementazione. Analogamente si raccomanda per chi sviluppa in `java` di compilare da shell eseguendo il comando `javac *.java` e poi eseguire `java Driver` per verificare la correttezza dell'algoritmo.

N.B. Le implementazioni debbono essere compilabili.

Problema 1 Analisi algoritmo [(a) 5/30; (b) 2/30]

Si considerino i metodi Java di seguito illustrati.

```
static int[][] p(int a[], int b[]) {
    if(a.length != b.length) return null;
    int n = a.length;
    int m[][] = new int[n][n];
    for(int i=0; i < n; i++)
        for(int j=0; j<n; j++)
            m[i][j] = a[i]*b[j];
    return m;
}

static int s(int a[][]) {
    if(a == null) return -1;
    int s = 0;
    for(int i = 0; i < a.length; i++)
        for(int j = 0; j < a[i].length; j++)
            s += a[i][j];
    return s;
}

static int c(int a[], int b[]) {
    return s(p(a,b));
}
```

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

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

(b) Discutere se `c` opera *in place* oppure no (risposte del tipo "sì, opera in place" o "no, non opera in place", ma prive di discussione, saranno completamente irrilevanti).

Problema 2 Progetto algoritmo C/Java [soglia minima: 5/30]

Con riferimento agli alberi binari di ricerca (BST), impiegati per realizzare un dizionario con chiavi `int` non negative, risolvere al computer quanto segue, in Java o in C. Si impieghi la rappresentazione basata su classe/struttura BST e classe/struttura BinNode.

- realizzare una funzione/metodo `BST_insert` per inserire in un BST una nuova chiave, avendo cura di gestire correttamente la possibile presenza di chiavi duplicate; la funzione/metodo deve restituire un riferimento/puntatore al nodo appena inserito; [3/30]
- realizzare una funzione/metodo `BST_find` che, data una chiave, determini se esiste un nodo (uno qualsiasi) contenente tale chiave e in tal caso ne restituisca riferimento/puntatore, altrimenti `null`/`NULL`; [3/30]
- realizzare una funzione/metodo `maxOnLevel` che dato un intero k restituisca la più grande chiave presente sul livello k (la radice è a livello 1). Se il livello k non esiste, restituire il valore convenzionale -1. [4/30]

È necessario implementare i metodi in `bst.c` o `bst.java` identificati dal commento `\*DA IMPLEMENTARE*\`. In tali file è permesso sviluppare nuovi metodi se si ritiene necessario. *Non è assolutamente consentito modificare altri metodi già implementati e altri file*, ad eccezione del `driver` per poter effettuare ulteriori test.

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

(a) Spiegare perché il sorting basato sul confronto richiede almeno $\Omega(n \log n)$ operazioni.

(b) Illustrare un algoritmo di sorting (pseudo-codice o codice) che esegue $\Theta(n)$ operazioni nel caso di input già ordinato, spiegando perché l'algoritmo paga $\Theta(n)$.[1]

(c) Qual è il miglior upper bound temporale teoricamente possibile per un eventuale algoritmo di sorting che avesse bisogno di impiegare uno spazio di memoria ausiliario di dimensione $\Theta(n^2)$? Spiegare.

Problema 4 Problema su grafi [(a) 3/30; (b) 5/30; (c) 4/30]

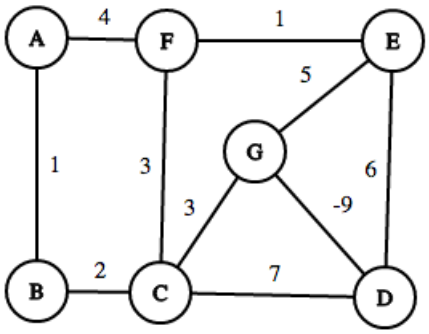
Si considerino i problemi di shortest-path su grafi semplici (non orientati, privi di archi paralleli e di cappi) e pesati sugli archi. Più precisamente, si fa riferimento a un grafo pesato (G, w), ove $G = (V, E)$, con $E = \{\{u, v\} \mid u, v \in V, u \neq v\}$, e w è una funzione di peso a valori interi: $w : E \mapsto \mathrm{Z}$.

Si richiede di sviluppare quanto segue:

(a) Descrivere le tipologie di problemi di shortest-path che è possibile definire su (G, w).

(b) Con riferimento al grafo mostrato in figura (si noti la presenza di pesi negativi), illustrare (tramite pseudo-codice o codice) un algoritmo atto a determinare l'albero dei cammini minimi radicato in A, determinandone il costo (che va giustificato).

[1] Algoritmi basati sull'esame dell'input e che terminano subito in caso di input già ordinato non sono ammissibili.



(c) Descrivere informalmente, attraverso una discussione, come possa essere determinato un albero dei cammini minimi qualora il grafo di cui al punto (b) avesse $w(e) = 1$, $\forall e \in E$. Valutarne il costo.