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 12-2-2018 – a.a. 2017-18 – 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, memorizzando il file nella cartella `ESAME`;
- 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`); tale cartella va posizionata nella cartella `ESAME`;
- tre altri file `probl1.<matricola>.txt`, `probl3.<matricola>.txt` e `probl4.<matricola>.txt`, contenenti, rispettivamente, gli svolgimenti dei problemi 1, 3 e 4; i tre file vanno posti nella cartella `ESAME`.

Attenzione: i simboli < e > usati nei nomi dei file fanno parte del linguaggio dei metadati e **non debbono essere inclusi nei nomi reali**. 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. In caso contrario, l'esame non è superato.

Problema 1 Analisi algoritmo

Si considerino i metodi Java di seguito illustrati.

```
// assumere a.length > 0
static int g(int[] a) {
  return g(a, 0, a.length-1);
}
```

```
static int g(int[] a, int i, int j) {
  if(i == j) return a[i];
  int onefourth = (j+1-i)/4;
  return g(a, i, i+onefourth) + g(a, i+onefourth+1, j);
}
```

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

(a) Determinare il costo temporale asintotico dell'algoritmo descritto da `ordina(int[])` in funzione della dimensione dell'input (eventuali analisi non asintotiche verranno penalizzate). [4/30]

(b) Come cambierebbe il costo dell'algoritmo se alla variabile `onefourth` venisse assegnato `(j+1-i)/5`, invece di `(j+1-i)/4`? [2/30]

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

Al fine di memorizzare tutte le parole presenti in un libro viene impiegato un BST con chiavi di tipo stringa (in Java: classe `String`; in C: array di `char` terminati da `0`). Ciascuna parola del libro è inserita nel BST dopo essere stata convertita in lettere maiuscole; nel caso di parole presenti più di una volta viene semplicemente incrementato un contatore presente nel nodo. Risolvere al computer quanto segue, in Java o in C. Si impieghi la rappresentazione basata su classe/struttura `BST` e classe/struttura `BinNode`. (Per il superamento della prova al calcolatore è necessario conseguire almeno 5 punti. Si prega di non fare usi di package nel codice Java)

2.1 Realizzare una funzione/metodo *insert* per inserire nel BST una nuova chiave. In caso di possibile presenza di chiavi duplicate si incrementi un contatore nel nodo. Per stabilire l'ordinamento reciproco di due chiavi (stringhe) usi l'operatore `compareTo` in Java o `strcmp` in C (`strcmp(s1, s2)` restituisce un numero negativo se `s1` precede `s2`, 0 se sono eguali, un numero positivo se `s1` segue `s2`). La funzione/metodo deve restituire un riferimento/puntatore al nodo appena inserito, o in cui sia stato incrementato il contatore (3 punti).

2.2 Realizzare una funzione/metodo *find* che, data una chiave, determini se esiste un nodo contenente tale chiave e in tal caso ne restituisca il numero di occorrenze, altrimenti restituisca 0 (3 punti).

2.3 Realizzare una funzione/metodo *mostFrequentString* che restituisca la parola più frequente. In caso di parità una qualsiasi delle parole più frequenti (3 punti).

2.4 Realizzare una funzione/metodo *isBalanced* che determini se l'albero è bilanciato, restituendo 1 se bilanciato, 0 altrimenti (3 punti).

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 metodi e strutture già implementati.* È invece possibile modificare il file `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 *basato sul confronto* che avesse bisogno di impiegare uno spazio di memoria ausiliario di dimensione $\Theta(\sqrt{n})$? Spiegare.

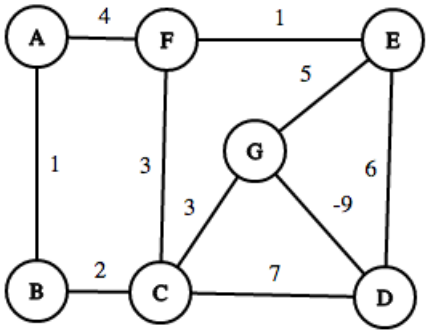
Problema 4 Problema su grafi [(a) 3/30; (b) 5/30; (c) 3/30]

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), determinare l'albero dei cammini minimi radicato in A, descrivendo il procedimento impiegato. Non saranno considerate ammissibili soluzioni prive di descrizione. Opzionale: mostrare lo pseudo-codice dell'algoritmo impiegato.



L'albero in output deve essere descritto attraverso due vettori di sette componenti ciascuno: il primo mostra le distanze da A (la distanza di A da se stesso è ovviamente 0); il secondo descrive, per ogni nodo, il suo predecessore (padre) nell'albero (il predecessore di A ovviamente non esiste).

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

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