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 26-1-2018 – a.a. 2017-18 – Tempo a disposizione: *4 ore* – somma punti: 33

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`.

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, che realizzano un algoritmo di ordinamento.

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

static int[] ordina(int[] a, int i) {
    if(i+1 == a.length) return a;
    return infila(i, ordina(a, i+1));
}
```

```
static int[] infila(int i, int[] a) {
    int x = a[i]; int j = i+1;
    if(j == a.length || x <= a[j]) return a;
    a[i] = a[j]; a[j] = x;
    return infila(j, a);
}
```

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) Descrivere, ad alto livello concettuale, l'algoritmo di ordinamento descritto dal metodo `ordina`, valutando se esso non sia semplicemente un'implementazione alternativa di un algoritmo studiato nel corso. [3/30]

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

Un insieme di punti del piano cartesiano deve essere memorizzato all'interno di un apposito BST. Benché ciascun punto sia caratterizzato da una coppia di coordinate `double`, l'ordinamento del BST è basato sulle coordinate polari dei punti, di cui si riporta per comodità la definizione. Dato il punto $p = (x_p, y_p)$ le sue coordinate polari sono una coppia $(r(p), \phi(p))$, ove $r(p)$, detto il raggio (o modulo) di p, è la distanza (reale non negativo) di p dall'origine O; $\phi(p) \in (-\pi, +\pi]$, fase di p, è l'angolo (in radianti) fra la direzione positiva dell'asse x e la

semiretta orientata $\overrightarrow{Op}$ (l'angolo di cui dovrebbe ruotare l'asse x in senso anti-orario per sovrapporsi a $\overrightarrow{Op}$). Vale evidentemente $r(p) = \sqrt{x_p^2 + y_p^2}$. Per quanto riguarda la fase:

$$\phi(p) = \begin{cases} \arctan(y_p/x_p) & x_p > 0 \\ \arctan(y_p/x_p) + \pi & x_p < 0 \wedge y_p \geq 0 \\ \arctan(y_p/x_p) - \pi & x_p < 0 \wedge y_p < 0 \\ \pi/2 & x_p = 0 \wedge y_p > 0 \\ -\pi/2 & x_p = 0 \wedge y_p < 0 \end{cases}$$

Nel caso dell'origine O, la sua fase viene posta convenzionalmente pari a zero radianti.

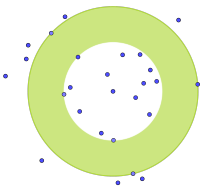
Ciò premesso, si precisa che l'ordinamento del BST è basato sul raggio dei punti; a parità di raggio viene usata la fase. La rappresentazione del BST si basa sulle strutture/classi `BSTNode` e `BST`. Si noti che non sono esplicitamente rappresentate le coordinate polari.

`BSTNode`		`BST`	
`x_coord`	coordinata x di un punto (tipo `double`)	`root`	riferimento alla root
`y_coord`	coordinata y di un punto (tipo `double`)	`size`	numero nodi
`left`	riferimento a figlio sinistro		
`right`	riferimento a figlio destro		

Ciò premesso, si richiede di:

- Realizzare un metodo/funzione `insert` che dato un punto descritto da una coppia di coordinate `double` x ed y, lo inserisca nel BST (rispettandone l'ordinamento basato sulla coordinate polari). Nel caso di punto già presente non va inserito nulla e va restituito `NULL`/`null`, mentre nel caso di punto non presente va restituito un riferimento al nuovo nodo appena inserito. La signature esatta è specificata nei file di supporto. [3/30]
- Realizzare un metodo/funzione `corona` che, dati due raggi r_1 ed r_2 (`double`), essendo $0 \leq r_1 \leq r_2$, determini e restituisca in tempo ottimale il numero di punti appartenenti alla corona circolare di centro O (origine), raggio minore r_1 e raggio maggiore r_2. Debbono funzionare correttamente i tre casi limite $r_1 = 0$, $r_2 = 0$, $r_1 = r_2$. La signature esatta è specificata nei file di supporto. [3/30]
- Realizzare un metodo/funzione `maxCorona` che, dato un BST del tipo descritto, determini e restituisca l'area della massima corona circolare vuota, che non contenga cioè punti al suo interno (sulla frontiera: ammessi). Tale corona circolare deve essere finita e deve esistere un punto di raggio maggiore o eguale al raggio maggiore della corona. La signature esatta è specificata nei file di supporto.

  La figura mostra un possibile esempio. [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 metodi e strutture già implementati.* È invece possibile modificare il file `driver` per poter effettuare ulteriori test.

Problema 3 Scelta struttura dati concreta

Per un fissato parametro intero $k > 0$, progettare una struttura dati per svolgere efficientemente le seguenti operazioni su interi positivi:

- inserimento di un nuovo valore (sono ammesse ripetizioni);
- determinazione del k-esimo valore (in ordine di grandezza) presente nella struttura (se questa contiene meno di k elementi restituire -1);
- rimozione del k-esimo valore (in ordine di grandezza) presente nella struttura (se tale valore esiste);
- determinazione della media dei valori presenti.

La struttura va descritta ad alto livello concettuale, ma occorre spiegare come eseguire le quattro operazioni richieste, determinandone anche il costo computazionale. [6/30]

Problema 4 Determinazione di proprietà di un grafo

Si richiamano due definizioni. Un grafo semplice $G = (V, E)$ è detto *bipartito* se V può essere partizionato in due sottoinsiemi disgiunti V_1 e V_2 in modo tale che ciascuno spigolo in E abbia un estremo in V_1 e l'altro in V_2. Più formalmente: $\forall\{u, v\} \in E : ((u \in V_1) \wedge (v \in V_2)) \vee ((v \in V_1) \wedge (u \in V_2))$, essendo $(V_1 \cap V_2 = \emptyset) \wedge (V_1 \cup V_2 = V)$.

Un grafo *completo* di n nodi, denotato K_n, è un grafo semplice in cui, comunque scelti due nodi distinti x ed y, esiste sempre l'arco $\{x, y\}$.

Ciò premesso, si richiede di

(a) Scrivere un algoritmo (pseudo-codice) che, dato un grafo, stabilisca se questo è bipartito o meno. Calcolarne il costo computazionale. [5/30]

(b) Scrivere un algoritmo (pseudo-codice) che, dato un grafo, stabilisca se questo contiene il sottografo K_5. Calcolarne il costo computazionale. [5/30]