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

Appello del 19-2-2019 – a.a. 2018-19 – Tempo a disposizione: *4 ore* – somma punti: 34

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`; è possibile aggiungere una quinta riga contenente eventuali informazioni addizionali che intendi porre all'attenzione del docente;
- 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 contenente **equazioni e disegni**, che verrà esaminato solo a condizione che risulti ben leggibile e a completamento di quanto digitalmente redatto.

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, assumento che i parametri `x` ed `y` siano sempre interi non negativi.

```
static int funct1(int x) {
    if(x <= 1) return x;
    return funct1(funct1(x/2)) + 1;
}
```

```
static long funct2(int y) {
    if(y <= 1) return y;
    return funct2(y-1)*funct1(y);
}
```

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 `funct1(int)` in funzione della dimensione dell'input $z_1 = |\mathtt{x}|$ (z_1 è la dimensione dell'input e `x` è l'input). [3/30]

(b) Determinare il costo temporale asintotico dell'algoritmo descritto da `funct2(int)` in funzione della dimensione dell'input $z_2 = |\mathtt{y}|$ (z_2 è la dimensione dell'input e `y` è l'input). [3/30]

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

Si vogliono utilizzare dei cavi in fibra ottica per collegare dei punti nel piano di cui sono note le coordinate (dei numeri **double**). Dato l'alto costo del materiale necessario ad effettuare il collegamento, si vuole minimizzare la quantità (lunghezza) della fibra impiegata. Si può assumere che, comunque considerati due punti, sia sempre teoricamente possibile (ma non è detto che sia conveniente!) collegarli con un cavo in fibra ottica.

Nei punti seguenti si richiederà di lavorare su un grafo che descrive la collocazione dei punti nel piano. Il grafo è rappresentato dalla classe/struttura `Graph`/`graph` che fa a sua volta uso della rappresentazione basata su *matrice di adiacenze*. Sono disponibili le funzioni ausiliarie:

- `double dist(point* a, point* b)` che ritorna la distanza euclidea tra due punti `a` e `b`;
- `int edge_to_code(int x, int y)` che assegna e ritorna un codice univoco all'arco che collega i nodi `x` ed `y` del grafo;
- `int* code_to_edge(int x)` che dato un codice assegnato ad un arco, del tipo ritornato dalla funzione precedente, ritorna un array di 2 elementi contenente i due nodi collegati dall'arco in input.

Sono inoltre disponibili delle implementazioni di `linked_list`, `partition`, e `min_heap`.
Tutto ciò premesso, risolvere al computer quanto segue, in Java o in C.

2.1 Realizzare una funzione/metodo `make_graph` che, dato in input un array di `struct point` (ed il numero dei punti nel caso di C), costruisce e ritorna il grafo pesato, non diretto, i cui nodi sono associati ai punti ed i cui archi rappresentano i collegamenti teoricamente possibili, pesati con la distanza euclidea che separa i nodi che collegano. [3/30]

2.2 Realizzare una funzione/metodo `get_best_connections` che, dato un grafo della tipologia costruito nel punto 2.1, costruisce e ritorna un nuovo grafo i cui archi rappresentano un opportuno albero ricoprente che descrive la cablatura da impiegare per l'interconnessione di tutti i punti, minimizzando la quantità (lunghezza) di cavo impiegato. [4/30]

2.3 Realizzare una funzione/metodo `get_tree_height` che, dato un albero ricoprente come quello costruito al punto precedente, ed un vertice v di tale albero, restituisce l'altezza dell'albero avente v come radice. [3/30]

È necessario implementare le funzioni in `graph_services.c` o i metodi in `GraphServices.java`, identificati dal commento `/*DA IMPLEMENTARE*/`. In tali file è permesso sviluppare nuovi metodi/funzioni ausiliari, se utile. *Non è assolutamente consentito modificare metodi e strutture già implementati.* È invece possibile modificare il file `driver.c`/`Driver.java` per poter effettuare ulteriori test.

Per il superamento della prova al calcolatore è necessario conseguire almeno 5 punti. Si prega di non fare usi di package nel codice Java.

Problema 3 Algoritmi

(a) Il prof. di algoritmi ti chiede all'esame di analizzare il costo computazionale di un algoritmo di cui ti fornisce il codice C. Purtroppo, guardando il codice, non si capisce assolutamente nulla, per cui sembra impossibile eseguire l'analisi. Pur di mostrare un risultato, magari imperfetto, decidi di valutare empiricamente il costo dell'algoritmo eseguendolo molte volte su diversi input. Più in particolare...

Descrivi la metodologia che useresti per stimare il costo asintotico empiricamente.[1] [4/30]

(b) Progettare un algoritmo (codice o pseudo-codice) che, dato un *grafo diretto* G, stabilisca se G è aciclico o meno. [4/30]

(c) Descrivere la rappresentazione di alberi binari mediante array, chiarendone pregi e difetti (70% del punteggio). In quali casi è estremamente conveniente? (30% del punteggio) [4/30]

Problema 4 Conteggio nodi k-ari di un albero

Dato un albero generico t (nessuna restrizione sul numero di figli di un nodo) e dato un intero positivo k, progettare un algoritmo (codice o pseudo-codice) che determini il numero di nodi in t che hanno esattamente k figli. Assumere che t sia rappresentato con una struttura bilineare, in cui ogni nodo possiede un riferimento a `firstChild` (primo figlio) e a `nextSibling` (prossimo fratello). [6/30]

[1] Trattandosi di una stima empirica, è del tutto accettabile che possa anche fornire risultati imprecisi.