| 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 19-9-2017 – 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, 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.

```
static int[] p(int a) {
    int arr[] = new int[a]; // assumere O(1)
    for(int i = 0; i < a; i++)
        arr[i] = i+1;
    return arr;
}

static int q(int a[]) {
    return q(a, 0, a.length-1);
}
```

```
static int q(int a[], int i, int j) {
    if(i > j) return 0;
    if(i == j) return a[i];
    int d = (j-i)/3;
    return q(a,i,i+d)+q(a,i+d+1,i+2*d)+q(a,i+2*d+1,j);
}

static int r(int a) {
    return q(p(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 `r(int)` in funzione della dimensione dell'input. [4/30]

(b) Determinare il costo spaziale asintotico dell'algoritmo descritto da `r(int)` in funzione della dimensione dell'input. [2/30]

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

Un file system contiene directories (cartelle) e files e viene rappresentato attraverso un albero generale, in cui ciascun nodo può avere un numero arbitrario di figli. I nodi corrispondono a directories o files e la relazione genitore-figlio corrisponde all'appartenenza del figlio (file o directory) al genitore (directory). Da quanto detto segue che ogni nodo interno dell'albero rappresenta una directory e ogni foglia rappresenta un file o una directory vuota.

Nonostante vi siano due differenti tipi di oggetti (directories e file) nel file system impiegheremo un unico tipo di nodo per la loro rappresentazione, descritto dalla struttura/classe `Node`. In particolare un `Node` possiede i seguenti campi:

- `name` nome file o directory in cui non compare `'/'`, max 16 caratteri, di tipo stringa;
- `parent` riferimento al genitore;
- `firstChild` riferimento al primo figlio nel caso di directory, `NULL`/`null` nel caso di file o directory vuota;
- `nextSibling` riferimento al prossimo fratello (directory o file);
- `size` dimensione in bytes nel caso di file, valore fisso 1024 (bytes) nel caso di directory;
- `isDir` flag per denotare una directory, interessante in caso di nodo foglia.

Useremo un `Node` anche per descrivere l'intero file system. In tal caso: `parent`, `nextSibling` contengono sempre `NULL`/`null`; `size` contiene il valore fisso 1024B (come per le directories); `isDir` è convenzionalmente posto a vero.

L'intero albero è descritto dalla struttura/classe `Tree`, che contiene i campi `root` (riferimento al nodo root, che non corrisponde a una directory o file, ma all'intero file system ed è perciò sempre diverso da `NULL`/`null`) e `size` (numero nodi nell'albero, inclusa la radice, che deve essere mantenuto aggiornato ad ogni operazione di inserimento/cancellamento). In particolare ciascun `Tree` contiene sempre una root.

Ciò premesso, si richiede di:

- Realizzare le funzioni/metodi `insertFile` e `insertDir` che dato il nome *nome* del file/directory, la sua dimensione in Byte *size* per il file (la dimensione è di default 1024B per le directory) e un riferimento a un nodo directory $p$ (genitore), crei un nuovo nodo $v$, inserisca $v$ come *primo figlio* di $p$ (l'eventuale primo figlio già presente diventa dunque il primo fratello di $v$) e restituisca un riferimento/puntatore a $v$. La signature esatta è specificata nei file di supporto. [3/30]
- Realizzare una funzione/metodo `findAll` che, dati un `Tree` e una stringa `s`, stampi su standard output, uno per riga, i percorsi assoluti di tutti i file/directories il cui nome è esattamente `s`; usare come separatore nel path il carattere `'/'`. Tutti i path iniziano con `'/'`; i path delle directories terminano con `'/'`. La signature esatta è specificata nei file di supporto. [3/30]
- Realizzare una funzione/metodo `cleanSmallDirs` che, dati un `Tree` e un intero `min`, rimuova dall'albero tutte le directories che non hanno sottodirectory (nodi figlio di tipo directory) e che occupano meno di `min` bytes; la funzione `cleanSmallDirs` deve inoltre restituire il numero totale di bytes eliminati. Ai fini del computo dell'ingombro di una directory va tenuto conto anche il valore 1024B associato alla directory stessa. Se anche tutte le sottodirectory di una cartella venissero cancellate, la cartella corrente non deve essere cancellata, anche se lo spazio occupato è minore di `min` bytes. La signature esatta è specificata nei file di supporto. [4/30]

È necessario implementare i metodi in `tree.c` o `Tree.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 alternativo

Nel paese di NoSortLand è stata approvata una legge che vieta espressamente l'impiego di qualunque algoritmo di ordinamento: è l'unico divieto in materia di algoritmi, essendo invece previsto il libero uso di altri algoritmi e delle strutture dati. I docenti di Algoritmi in quel paese assegnano comunque agli esami problemi che richiedono l'ordinamento di insiemi (di naturali) sulla base del fatto che i legiferanti, non esperti di algoritmi e strutture dati, non si sono resi conto che vietando l'impiego degli algoritmi di ordinamento non avrebbero davvero impedito di risolvere il problema dell'ordinamento.

Ciò premesso, si richiede di descrivere i passi logici di una procedura alternativa che permette appunto di risolvere il problema dell'ordinamento senza impiegare algoritmi di ordinamento, determinandone il costo asintotico (che va giustificato e non semplicemente riportato). La descrizione può essere di alto livello concettuale ma deve chiarire come sia possibile ottenere l'effetto dell'ordinamento. Naturalmente, non possono essere impiegati i vari algoritmi "*-sort" presentati/studiati nel corso, né loro varianti, né possono essere progettati algoritmi che hanno la finalità, esplicita o meno, di ordinare. Possono essere invece impiegate soluzioni che hanno finalità ben definite e che consentono di ottenere l'ordinamento come effetto collaterale. [5/30]

**Problema 4 Test di aciclicità**

Descrivere (pseudo-codice o codice) un algoritmo che, dato un grafo diretto $G = (V, E)$, determini se $G$ è aciclico o meno. Analizzarne il costo computazionale. [Algoritmo: 3/30; costo: 2/30]

**Problema 5 Ricerche bibliografiche**

Una prassi consolidata per studiare approfonditamente un determinato argomento di informatica è prendere una pubblicazione scientifica (recente) sull'argomento e da questa ricavare, attraverso la sua bibliografia, un insieme di pubblicazioni rilevanti. Reperite queste ultime, è possibile individuare nelle loro rispettive bibliografie ulteriori pubblicazioni, e così via, ripetendo l'operazione su ogni nuova pubblicazione reperita. Procedendo con questo metodo si arriverà a un punto ove tutte le pubblicazioni presenti nell'unione di tutte le bibliografie saranno state identificate.

La laureanda Alice si reca dal relatore il quale le affida un articolo (*articolo0*), raccomandandole di reperire tutta la bibliografia direttamente o indirettamente da esso derivabile. Alice, appena uscita, si domanda quanti articoli dovrà reperire (quesito A – non va conteggiato l'articolo appena ricevuto dal relatore). Naturalmente, se citato da più fonti, un articolo verrà considerato solo una volta ai fini del conteggio. Pensando a ciò, Alice si domanda anche quale sarà il massimo numero di citazioni dirette di uno stesso articolo (quesito B).

Ciò premesso, si chiede di elaborare quanto segue.

(a) Formulare i problemi posti (quesiti A e B) come problemi 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 ai problemi in esame; successivamente vanno formulati i quesiti A e B come problemi su grafi, utilizzando una terminologia propria del mondo dei grafi, in cui non sono più presenti i termini articolo, citazione ecc. [2/30]
N.B. Qui specificare *quali problemi su grafi* debbono essere risolti, non *come*. Quindi, *non vanno qui indicati algoritmi*.

(b) Descrivere un algoritmo (pseudo-codice) che, preso in input il grafo utilizzato per modellare il quesito A, e l'informazione atta a denotare l'*articolo0*, determini e restituisca in output l'intero positivo che costituisce corretta risposta al quesito A. Valutare il costo dell'algoritmo. [4/30]
N.B. Lo pseudo-codice privo di indentazione riceve penalizzazione del 20%.

(c) Descrivere un algoritmo (pseudo-codice) che, preso in input il grafo utilizzato per modellare il quesito B, e l'informazione atta a denotare l'*articolo0*, determini e restituisca in output l'intero positivo che costituisce corretta risposta al quesito B. Valutare il costo dell'algoritmo. [3/30]
N.B. Lo pseudo-codice privo di indentazione riceve penalizzazione del 20%.