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 dell'8-6-2018 – a.a. 2017-18 – Tempo a disposizione: *4 ore* – somma punti: 38

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, 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, in cui la matrice `a` è quadrata.

```
static int m1(int a[][]) {
  return m1(a, 0, a.length-1, 0, a[0].length-1);
}

static int m1(int a[][], int l1, int l2, int c1, int c2) {
  if(c1 > c2 || l1 > l2) return 0;
  if(c1 == c2 && l1 == l2) return a[l1][c1];
  int c = (c1+c2)/2, l = (l1+l2)/2;
  return m1(a, l1, l, c1, c)   + m1(a, l1, l, c+1, c2) +
         m1(a, l+1, l2, c1, c) + m1(a, l+1, l2, c+1, c2);
}

static int[] m2(int p) {
  int[] v = new int[p];
  int s = 0;
  for(int i=0; i<p; i++) v[i] = s++;
  return v;
}

static int[] m3(int a[][]) {
  return m2(m1(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 `m1(int[][])` in funzione della dimensione dell'input. [3/30]

(b) Determinare il costo temporale asintotico dell'algoritmo descritto da `m2(int)` in funzione della dimensione dell'input. [2/30]

(c) Il costo temporale asintotico dell'algoritmo descritto da `m3(int[][])` in funzione della dimensione dell'input non è semplice da calcolare. Discutere un metodo di analisi e le relative difficoltà pratiche. [3/30]

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

In questo problema si fa riferimento a *grafi semplici*, rappresentati attraverso *liste di incidenza*, che possono essere pensate come liste di adiacenza in cui appaiono, per ogni nodo, gli archi incidenti invece dei nodi adiacenti. A ciascun nodo u è dunque associata una lista collegata contenente degree(u) elementi, ciascuno dei quali descrive un arco (incidente u); inoltre ad ogni nodo sono associati un numero progressivo (a iniziare da 0) automaticamente assegnato dalla primitiva di inserimento nodo e una label (stringa) fornita come input alla stessa primitiva. Similmente, a ciascun arco è automaticamente assegnato un numero progressivo (a iniziare da 0). Nodi ed archi sono rappresentati dalle classi/strutture `GraphNode`/`graph_node` e `GraphEdge`/`graph_edge`; il grafo è rappresentato dalla classe/struttura `Graph`/`graph`.

In tutto il problema la gestione delle liste (di archi e/o di nodi) deve essere effettuata mediante il tipo `linked_list` (C) o la classe `java.util.LinkedList<E>` (Java); per tali tipi sono già disponibili, nel codice sorgente fornito, o nella classe `java.util.LinkedList<E>`, le primitive di manipolazione. Si noti che gli elementi delle liste sono dei contenitori che includono puntatori ai nodi/archi del grafo.

Sono inoltre già disponibili le primitive di manipolazione grafo: creazione di grafo vuoto, get lista nodi, inserimento di nuovo nodo, get lista archi incidenti un dato nodo, inserimento nuovo arco, get chiave (progressivo) di un dato nodo/arco, get label (stringa) di un dato nodo, get opposite di un nodo per un dato arco, cancellazione arco, cancellazione nodo (e relativi archi incidenti), cancellazione grafo, stampa grafo. Per dettagli sulle signature di tali primitive, così come per dettagli su quali porzioni di memoria allocata vengono liberate dalle varie primitive di cancellazione, si rimanda ai file sorgente distribuiti (`.h` o `.java`). Si noti che le primitive forniscono un insieme base per la manipolazione di grafi, ma i problemi proposti possono essere risolti usandone solo un sottoinsieme.

Tutto ciò premesso, risolvere al computer quanto segue, in Java o in C. (A fine testo è disponibile un disegno che mostra la rappresentazione di un semplice grafo di esempio).

2.1 Realizzare una funzione/metodo `getEdges` che, dato in input un grafo G, restituisca in output una lista dei suoi archi. Se il grafo non ha archi va restituito `null`/`NULL`; naturalmente, ciascun arco deve essere presente una sola volta nella lista in output. Per l'esatta signature della funzione/metodo fare riferimento ai file ausiliari forniti. [3/30]

2.2 Realizzare una funzione/metodo `isConnected` che, dato un grafo semplice G, restituisca 1 se G è connesso, 0 altrimenti (3 punti). Per l'esatta signature della funzione/metodo fare riferimento ai file ausiliari forniti. [3/30]

2.3 Un grafo semplice è detto k-connesso sugli archi, con $k > 0$, se è connesso e se è possibile disconnettere il grafo attraverso la rimozione di un opportuno insieme di k archi. Ciò premesso, realizzare una funzione/metodo `is1Connected` che, dato un grafo semplice G, restituisce 1 se G è 1-connesso, 0 altrimenti. Per l'esatta signature della funzione/metodo fare riferimento ai file ausiliari forniti. [4/30]

È necessario implementare le funzioni in `graph_services.c` o i metodi `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 Miscellanea problemi

(a) Alice e Bob discutono animatamente, mentre stanno considerando il problema di ordinare molti piccoli array. Alice sostiene che, visto che la dimensione di tali array è molto ridotta, i risultati dell'analisi asintotica non sono applicabili, perché basati sull'idea che la dimensione dell'input tende all'infinito, cosa che nel caso in esame non accade: tanto vale usare un algoritmo semplice e robusto come Insertion Sort. Bob risponde che, anche se gli array sono piccoli, visto che si dovrà eseguire un alto numero di operazioni di ordinamento, alla fine l'andamento asintotico si manifesterà, e quindi propende per uno HeapSort. A chi daresti ragione? [2/30]

(b) Occorre scegliere una funzione hash per un tabella di $N = 8000$ elementi. Quali sono le linee guida per effettuare una scelta metodologicamente corretta? [2/30]

(c) È assegnato uno heap di n elementi, rappresentato attraverso un array di n caselle. Sfruttando la nota proprietà secondo cui ogni sotto-albero di uno heap è uno heap, descrivere un algoritmo (codice o pseudo-codice) che dato un array H che rappresenta uno heap e un indice $i \in \{0, 1, \ldots, n-1\}$ costruisca un nuovo heap clone del "sotto-heap" avente radice rappresentata nella casella di posto i e lo restituisca. Sai stimare la dimensione del sotto-heap? [4/30]

Problema 4 Alberi

Un file system è rappresentato attraverso un albero secondo le seguenti regole:

- la radice è un nodo associato all'intero file system e contiene informazioni sulla sua capacità;
- i nodi interni (esclusa la radice) rappresentano cartelle non vuote e contengono i nomi delle cartelle;
- le foglie rappresentano cartelle vuote oppure file e contengono i nomi delle cartelle o file;
- ogni foglia che rappresenta un file contiene anche la dimensione in byte del file stesso.

Sul file system si desidera effettuare efficientemente alcune operazioni, per cui si rende necessario scegliere sia gli opportuni algoritmi, sia la modalità di rappresentazione del file system.

Ciò premesso, di chiede di sviluppare i seguenti punti. Descrivere:

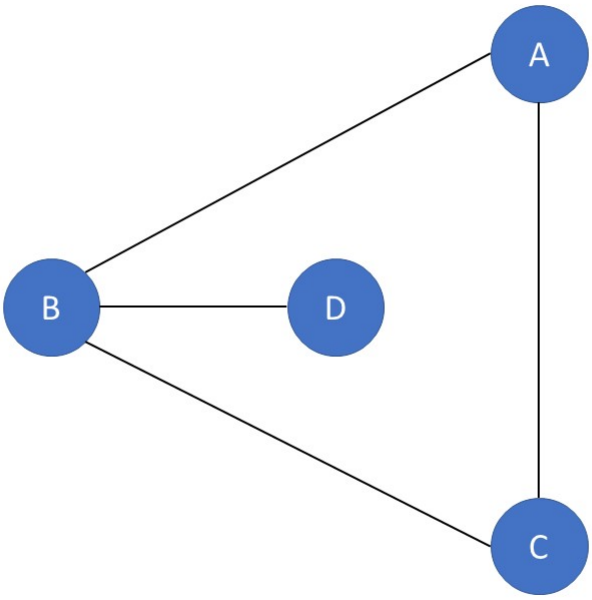
(a) un algoritmo (codice o pseudo-codice) che, dato un file system f, restituisce il path assoluto del file di maggiore dimensione presente nel file system; in caso di file non presenti restituire `NULL`; in caso di più file aventi la dimensione massima fare riferimento a uno qualunque di essi; [3/30]

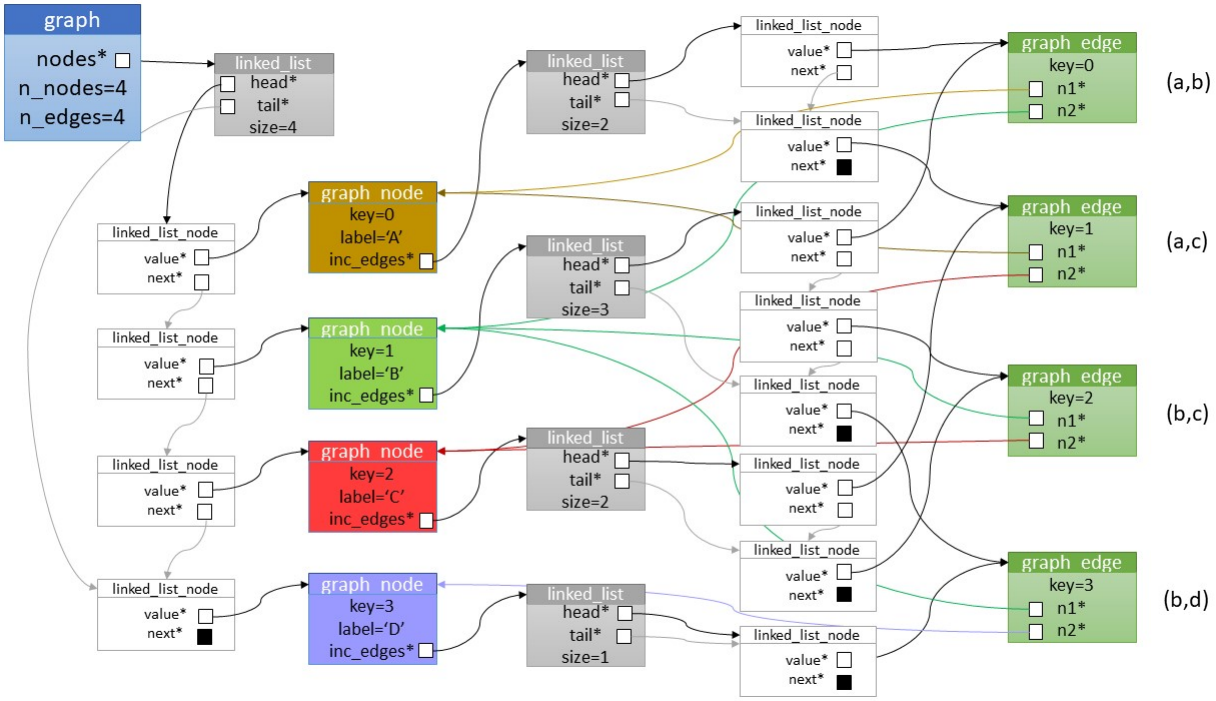
(b) un algoritmo (codice o pseudo-codice) che, dato un file system f, stampi su standard output, per ogni cartella, il suo nome e la sua occupazione in byte (data dalla somma delle dimensioni di tutti file presenti nel sottoalbero di cui è radice; si noti che i nodi che rappresentano cartelle non contengono tale informazione); [3/30]

(c) un algoritmo (codice o pseudo-codice) che, dato un file system f e un naturale $d > 0$, restituisce il numero di file che si trovano a profondità[1] non superiore a d; [3/30]

(d) un'opportuna struttura dati concreta per la rappresentazione del file system, in modo da favorire l'esecuzione efficiente degli algoritmi di cui ai punti precedenti; si noti che data una foglia deve essere possibile capire se trattasi di cartella vuota o di file. [3/30]

[1] Distanza dalla radice.


A
B
D
C


graph
nodes*
n_nodes=4
n_edges=4
linked_list
head*
tail*
size=4
linked_list_node
value*
next*
graph_node
key=0
label='A'
inc_edges*
key=1
label='B'
key=2
label='C'
key=3
label='D'
graph_edge
key=0
key=1
key=2
key=3
n1*
n2*
(a,b)
(a,c)
(b,c)
(b,d)
size=3
size=2
size=1