| | |
|---|---|
| **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 6-7-2018 – 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`; 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, in cui il parametro `int x` è da intendersi non negativo.

```
static int log2(int x) {
    if(x <= 1) return 0;
    return 1+log2(x/2);
}
```

```
static int loglog2(int x) {
    return log2(log2(x));
}
```

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 `log2(int)` in funzione della dimensione dell'input. [3/30]

(b) Determinare il costo temporale asintotico dell'algoritmo descritto da `loglog2(int)` in funzione della dimensione dell'input. [2/30]

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

In questo problema si fa riferimento a *grafi semplici* e ad *alberi binari*. I grafi semplici sono rappresentati attraverso *liste di incidenza*, ovvero 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`.

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. (A fine testo è disponibile un disegno che mostra la rappresentazione di un semplice grafo di esempio).

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. 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.

Gli alberi binari sono rappresentati attraverso la tradizionale struttura in cui per ogni nodo sono definiti riferimenti espliciti al figlio destro e al figlio sinistro. Classe/struttura `BinTree`/`binTree`. Per il problema posto non sono necessarie le primitive di inserimento/rimozione nodi.

Tutto ciò premesso, risolvere al computer quanto segue, in Java o in C.

2.1 Realizzare una funzione/metodo `isTree` che, dato un grafo semplice $G$, stabilisce se $G$ è un albero.[1] Per l'esatta signature della funzione/metodo fare riferimento ai file ausiliari forniti. [3/30]

2.2 Realizzare una funzione/metodo `getSizeConstrainedPath` che, dati un grafo semplice $G = (V, E)$ e tre vertici *distinti* $x, y, z \in V$, restituisce il costo del cammino minimo da $x$ a $z$, che passi obbligatoriamente per $y$. Attenzione, per soddisfare il vincolo di passaggio obbligato per $y$, il percorso minimo da $x$ a $z$ così vincolato potrebbe dover passare due volte su qualche nodo/arco. Si noti inoltre che il percorso da $x$ a $y$ non deve passare per $z$, mentre è ammissibile un percorso da $y$ a $z$ che passi per $x$; in altre parole, partendo da $x$, si deve incontrare $y$ prima di $z$.

Assumere che i tre vertici forniti in input appartengono effettivamente al grafo. Se il cammino non esiste restituire il valore convenzionale $-1$. Per l'esatta signature della funzione/metodo fare riferimento ai file ausiliari forniti. [3.5/30]

2.3 Realizzare una funzione/metodo `getSizeMaxCompleteSubtree` che, data la radice di un albero binario, restituisce il numero di nodi del suo sottoalbero[2] completo[3] più grande. Per l'esatta signature della funzione/metodo fare riferimento ai file ausiliari forniti. [3.5/30]

È necessario implementare le funzioni in `graph_services.c` e `bintree_services.c` o i metodi in `GraphServices.java` e `BinTreeServices.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) Descrivere (codice o pseudo-codice) un algoritmo (3/30) che preso in input un array di $n = 2m + 1$ interi (non ordinato, tutti i valori sono differenti, $m \geq 0$), determini efficientemente, restituendolo, l'indice del valore mediano (l'$(m + 1)$-esimo in ordine di grandezza crescente). Analizzare (2/30) l'algoritmo nel caso medio e nel caso peggiore. [5/30]

(b) Si vuole ordinare un array di $n$ interi appartenenti all'insieme $\{1, 2, \ldots, 2n\}$, non necessariamente in-place. Descrivere (codice o pseudo-codice) un algoritmo particolarmente efficiente per il caso in questione, fornendone l'analisi di costo. [4/30]

(c) Per il problema $\mathcal{A}$ sono noti due algoritmi differenti: $A_1$, di costo $\Theta(n^2)$; $A_2$, di costo $O(n^2 \log n)$ e $\Omega(n \log n)$. Sulla base di tali elementi, e in assenza di altre informazioni, cosa possiamo dedurre in merito a upper e lower bound di $\mathcal{A}$? Fornire dettagli. [3/30]

[1] Si rammenta che un albero è un grafo semplice, connesso e aciclico, ove aciclico significa che non esistono cicli composti da archi distinti.

[2] Si rammenta che dato un albero $t$ e un suo nodo $v$, il *sottoalbero* di radice $v$ è quel particolare sottoalbero contenente $v$, tutti i suoi discendenti in $t$ e nessun altro nodo.

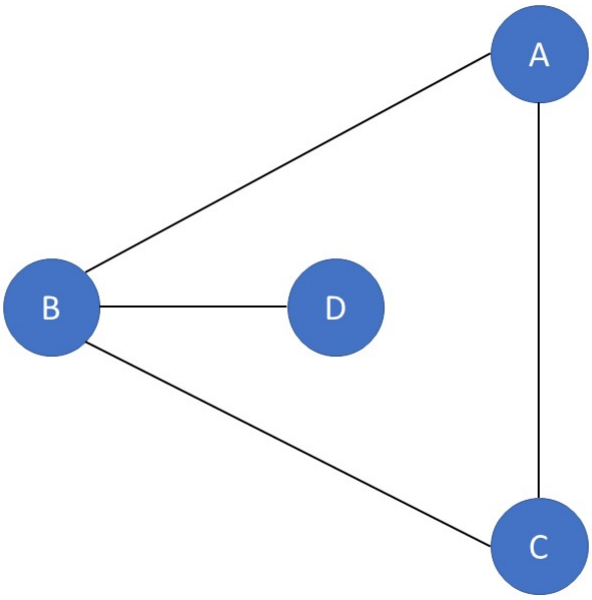
[3] Si rammenta che un albero binario è *completo* se tutti i nodi interni hanno due figli e tutti i rami hanno la stessa lunghezza; equivalentemente, se è vuoto o se la sua radice è genitore di due sottoalberi completi della stessa altezza.

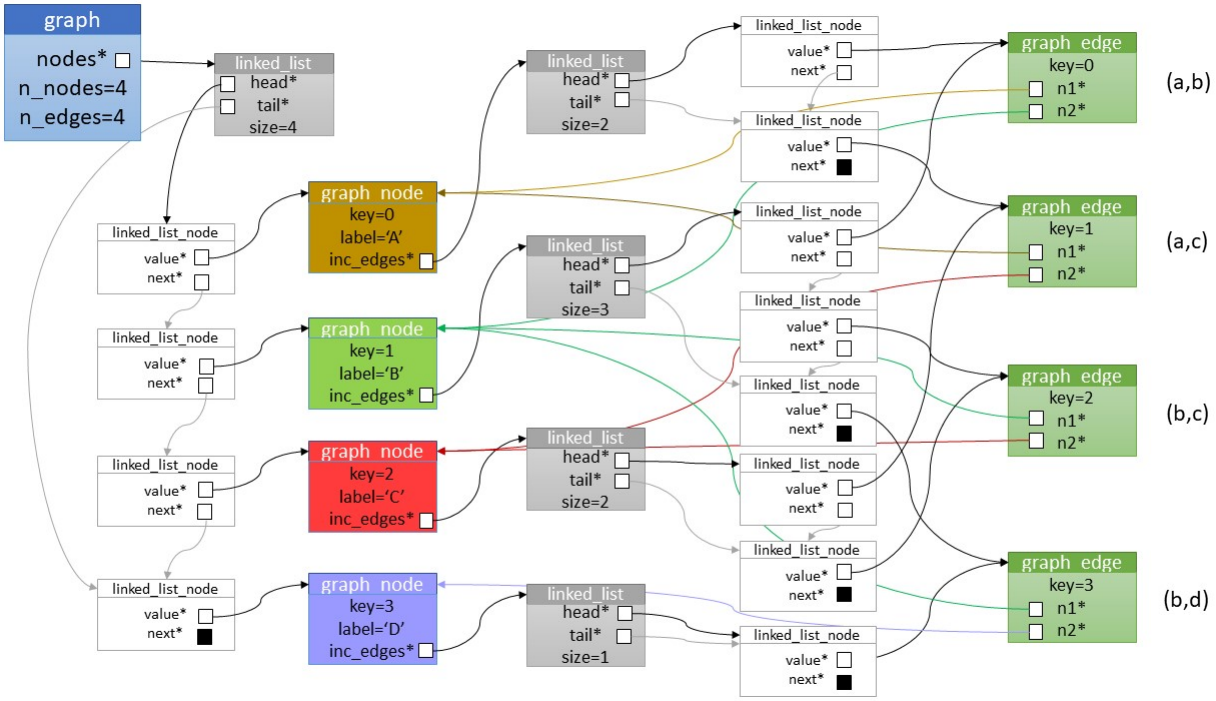
**Problema 4 Strutture complesse**

(a) Definire il concetto di *connessione debole* su grafi diretti. Descrivere un algoritmo (pseudo-codice) che, dato un grafo diretto $D$ rappresentato attraverso *matrice di adiacenza*, determina se $D$ è debolmente connesso. [4/30]

(b) Le parole di un testo sono memorizzate in un BST, secondo il loro ordine alfabetico. In particolare, per ogni parola del testo è stato creato un nodo i cui campi si chiamano `word` (puntatore a stringa), `left` (puntatore a nodo figlio sinistro) e `right` (puntatore a nodo figlio destro); tale nodo è stato poi inserito in un BST inizialmente vuoto, ordinato secondo l'ordine alfabetico delle parole contenute; naturalmente, se una parola è presente $p$ volte nel testo, ci saranno $p$ nodi differenti nel BST associati alla stessa stringa.

Ciò premesso descrivere un algoritmo (codice o pseudo-codice) che, dato un BST del tipo testé descritto, restituisce la parola più frequente nel testo; nel caso di più parole a frequenza massima, restituire la prima trovata. [4/30]


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