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 13-09-2019 – a.a. 2018-19 – Tempo: 4 ore – somma punti: 32

Istruzioni

Lanciare la macchina virtuale Oracle VirtualBox e lavorare all'interno della cartella `Esame`. Per prima cosa compilare il file `studente.txt` con le informazioni richieste. In particolare, cognome, nome, matricola, email, esame (vecchio o nuovo), linguaggio in cui si svolge l'esercizio 2 (Java o C). Per quanto riguarda il campo esame (vecchio o nuovo), possono optare per il vecchio esame gli studenti che nel periodo che va dal 2014-15 al 2017-18 (estremi inclusi) sono stati iscritti al II anno.

Nota bene. Per ritirarsi è sufficiente rinominare il file `studente.txt` in `studenteritirato.txt`, *senza* cancellarlo.

Come procedere. Nella cartella `Esame` trovate i) il testo del compito, ii) il file `studente.txt` sopra citato, iii) due sottocartelle compresse `c-aux.zip` e `java-aux.zip`, contenenti il codice di supporto e lo scheletro delle soluzioni per il quesito di programmazione (quesito 2). Svolgere il compito nel modo seguente:

- Per il quesito 2, estrarre la cartella `c-aux` o `java-aux` (a seconda del linguaggio che si intende usare) all'interno della cartella `Esame`, estraendola dal corrispondente file `.zip` . Alla fine la cartella `c-aux` (o `java-aux`, a seconda del linguaggio usato) conterrà le implementazioni delle soluzioni al quesito 2, ottenute completando i relativi metodi (o le relative funzioni) contenuti nei file forniti dai docenti; si ricorda ancora una volta che la cartella `java-aux` (o `c-aux`) deve trovarsi all'interno della cartella `Esame`.
- Per i quesiti 1, 3 e 4, creare tre file `probl1.<matricola>.txt`, `probl3.<matricola>.txt` e `probl4.<matricola>.txt`, contenenti, rispettivamente, gli svolgimenti dei problemi 1, 3 e 4; i tre file devono trovarsi nella cartella `Esame`. È possibile integrare i contenuti di tali file con eventuale materiale cartaceo *unicamente per disegni, grafici, tabelle, calcoli*.

Attenzione: i simboli < e > usati nei nomi dei file fanno parte del linguaggio dei metadati e non debbono essere inclusi nei nomi reali.

Avviso importante. Per svolgere il quesito di programmazione si consiglia caldamente l'uso di un editor di testo (ad esempio Geany) e la compilazione a riga di comando, strumenti più che sufficienti per lo svolgimento del compito. La macchina virtuale mette a disposizione diversi ambienti di sviluppo, quali Eclipse e Javabeans. Gli studenti che li usano lo fanno a proprio rischio. In particolare, *se ne sconsiglia l'uso qualora non se ne abbia il pieno controllo e certamente se non si è già in grado di sviluppare servendosi unicamente di un editor e della compilazione a riga di comando*. Qualora lo studente intenda comunque usare un ambiente di sviluppo integrato, si raccomanda di controllare che i file vengano effettivamente salvati nella cartella `java-aux` (o `c-aux`, a seconda del linguaggio usato), in quanto vi è il rischio concreto di perdere il proprio lavoro.

In generale, file salvati esternamente alla cartella `Esame` andranno persi al termine della prova e quindi non saranno corretti.

Quesito 1: Analisi algoritmo

Il seguende codice Java implementa un algoritmo che determina la lunghezza della più lunga sottosequenza crescente di una sequenza data, ed è basato sul paradigma della programmazione dinamica.

```java
static int CeilIndex(int A[], int l, int r, int key) {
    while (r - l > 1) {
        int m = l + (r - l) / 2;
        if (A[m] >= key) r = m;
        else l = m;
    }
    return r;
}

static int LongestIncreasingSubsequenceLength(int A[], int size) {
    int[] tailTable = new int[size];
    int len;
    tailTable[0] = A[0];
    len = 1;
    for (int i = 1; i < size; i++)
        if (A[i] < tailTable[0])
            tailTable[0] = A[i];
        else if (A[i] > tailTable[len - 1])
            tailTable[len++] = A[i];
        else tailTable[CeilIndex(tailTable, -1, len - 1, A[i])] = A[i];
    return len;
}
```

Si risponda ai seguenti quesiti:

1. Determinare il costo asintotico dell'algoritmo `LongestIncreasingSubsequenceLength`.

   Punteggio: [4/30]

2. Determinare il contenuto finale dell'array `tailTable` sulla seguente sequenza di input :

   ```
   {0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15}
   ```

   Punteggio: [3/30]

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

In questo problema si fa riferimento ad alberi binari di ricerca (Binary Search Tree o **BST**), aventi chiavi intere. A ogni nodo del BST (classe `Node` in Java e `struct _bst_node` in C) sono associati una coppia (*chiave, valore*), dove *chiave* è un intero *positivo*, nonché i riferimenti ai figli sinistro e destro del nodo.

Sono inoltre già disponibili le primitive di manipolazione del BST (contanute nella classe `BST.java` in Java e nel modulo `bst.c` in C): creazione di un BST (a seguito dell'inserimento della prima coppia (chiave, valore)), inserimento di una coppia (chiave, valore), restituzione del valore associato a una chiave data (se esistente), restituzione della chiave di valore minimo, rimozione

del nodo associato alla chiave di valore minimo, rimozione del nodo associato a una chiave data, se presente nell'albero. Sono infine disponibili metodi/funzioni, che restituiscono la radice del BST e il numero di chiavi in esso presenti.

Per dettagli sulle segnature 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, nei quali i metodi Java (funzioni C) implementati sono preceduti da un breve commento che ne illustra il funzionamento.

Si noti che le primitive forniscono un insieme base per la manipolazione di BST, ma i problemi proposti possono essere risolti usandone solo un sottoinsieme.

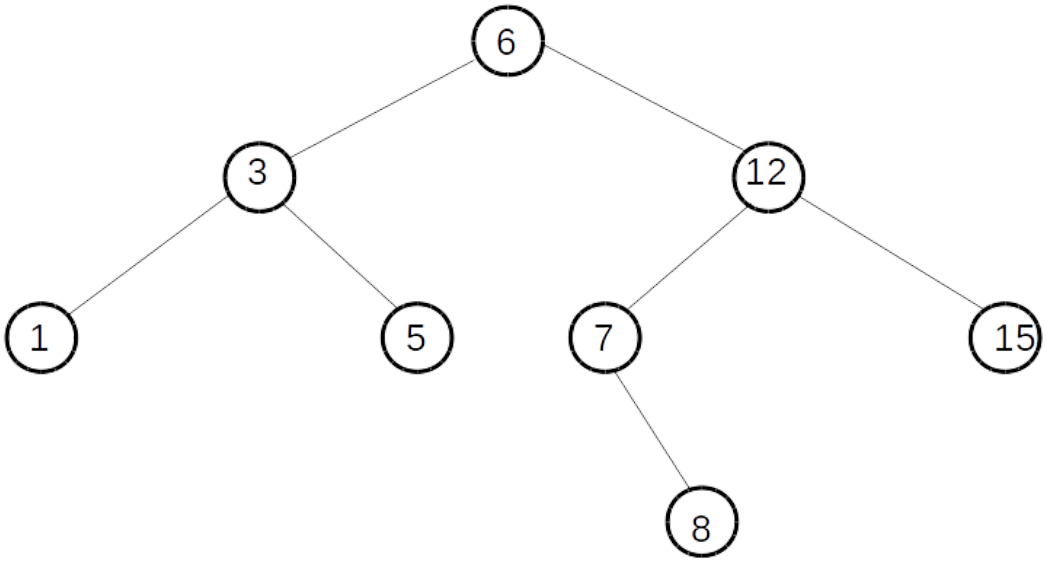

Fig. 1: un albero binario di ricerca. Si noti che nella figura sono mostrate soltanto le chiavi associate ai nodi, rispetto alle quali è definito l'ordinamento.

Tutto ciò premesso, risolvere al calcolatore quanto segue, in Java o C. Gli esempi di seguito riportati fanno riferimento alla Fig. 1, che corrisponde al BST usato nel programma di prova (`Driver.java` o `driver`).

1. Implementare la funzione/metodo `public LinkedList ordina(BST t)` della classe `BSTServices.java` (in C: `void ordina(BST t)` del file `bst_services.c`) che, ricevuto in ingresso il riferimento a un oggetto di classe/tipo `BST`, restituisce una `LinkedList` contenente tutte le chiavi ordinate in senso crescente (in C: stampa su standard output tutte le chiavi ordinate in senso crescente). Si richiede che il costo asintotico nel caso peggiore dell'algoritmo implementato sia *lineare* rispetto al numero di chiavi presenti nel BST. Ad esempio, nel caso della Fig. 1 il programma dovrebbe restituire la lista

   ```
   [1, 3, 5, 6, 7, 8, 12, 15]
   ```

   Punteggio: [3/30]

2. Implementare il metodo Java `LinkedList outer_range(BST t, int k1, int k2)` (in C: la funzione `void outer_range(BST t, int k1, int k2)`) della classe `BSTServices` (in C: del file `bst_services.c`) che, dato un BST `t` e due chiavi `k1` e `k2`, restituisce una lista ordinata in senso crescente, contenente tutte le chiavi presenti in `t` *che non sono contenute* nell'intervallo $[k1, k2]$ (in C: stampa su standard output, ordinatamente, tutte le chiavi presenti in `t` *che non sono contenute* nell'intervallo $[k1, k2]$). Se tale intervallo contiene tutte le chiavi presenti, il metodo restituirà una lista vuota (in C: la funzione non stamperà nulla)

Ad esempio, con riferimento alla Fig. 1, se $k1 = 5$ e $k2 = 12$, il programma deve restituire (o stampare) la lista `[1, 3, 15]`.

Il metodo deve avere costo asintoticamente ottimo rispetto alla profondità dell'albero e al numero di chiavi non contenute nell'intervallo $[k1, k2]$ che sono presenti nel BST.

Punteggio: [4/30]

3. Implementare la funzione/metodo `int altezza(BST t)` della classe `BSTServices` (o `bst_services.c` in C) che, dato un BST `t`, ne determina l'altezza. Ad esempio, per il BST in figura il programma dovrebbe restituire il valore 3. Il costo deve essere asintoticamente ottimo.

   Punteggio: [3/30]

Quesito 3: Algoritmi

1. Si consideri una tabella hash a *indirizzamento aperto* di dimensione $N = 11$ e a chiavi intere. Si supponga che la funzione di compressione sia $f(x) = x \mod N$ e che le collisioni siano risolte mediante *esplorazione lineare*. Dare un esempio di aggregazione (clustering) *primaria*, generata dall'inserimento in successione di 4 coppie (chiave, valore). *Occorre spiegare cos'è l'aggregazione e perché si verifica nell'esempio proposto*.

   Punteggio: [3/30]

2. Si mostri come si può usare un heap minimale per individuare, con un costo $O(n + k \log n)$, i k valori più piccoli tra quelli contenuti in un *array non ordinato* contenente n chiavi intere. *Occorre giustificare la risposta.*

   Punteggio: [3/30]

3. Con riferimento alla Fig. 1, descrivere la sequenza con in cui vengono visitati i nodi dell'albero binario, per ciascuna delle visite in pre-ordine, post-ordine e simmetrico.

   Punteggio: [3/30]

Quesito 4:

Una società di web analytics ha collezionato dati sui "like" attribuiti da utenti (che chiameremo "soggetti") a prodotti/servizi (che chiameremo "oggetti"). Tali dati vengono descritti attraverso un opportuno grafo i cui nodi descrivono utenti e/o prodotti/servizi, mentre gli archi descrivono l'attribuzione dei "like". L'azienda tuttavia nutre dei dubbi sulla qualità dei dati poiché ritiene che alcuni utenti abbiano usato il pulsante "like" a casaccio, senza dunque caratterizzare davvero le loro preferenze. Poiché non esiste un algoritmo che consenta di individuare tale categoria di utenti, l'azienda decide di fare alcune analisi, approntando alcuni semplici strumenti.

Uno di tali strumenti, detto `contaAnaloghi`, è teso a valutare l'unicità di un utente: in particolare, dato un utente u, si vuole determinare il numero di utenti (diversi da u, e detti *analoghi* ad u) i cui like definiscono insiemi di oggetti che sono superinsiemi di quello definito dai like di u. Se ad esempio u ha attribuito il like agli oggetti $\{a, d\}$ e

- v ha messo like su $\{a, b, e\}$ (intersezione non nulla, non superinsieme)
- w su $\{a, c, d\}$ (superinsieme)
- x su $\{a, d\}$ (superinsieme, uguale)
- y su $\{d\}$, (intersezione non nulla, sottoinsieme, non superinsieme)

allora gli analoghi di u saranno 2 (w ed x).

Si risponda alle domande seguenti:

- Definire con precisione il grafo usato per rappresentare i "like", descrivendone caratteristiche e proprietà (semplice, connesso, orientato, bipartito, ecc.). Definire in termini di problemi su grafi l'input e l'output dell'algoritmo implementato da `contaAnaloghi`.

  Punteggio: [3/30]

- Si descriva (è sufficiente lo pseudo-codice o comunque una descrizione dettagliata) l'algoritmo usato per implementare `contaAnaloghi`. La descrizione può essere anche ad alto livello concettuale ed usare primitive non elementari, purché quest'ultime appartengano al corredo standard del tipo/classe `Graph`. Descrizioni basate su codice C/Java sono comunque considerate accettabili.

  Punteggio: [3/30]