

# Esame di

<b>Algoritmi e strutture dati (parte di Fondamenti di informatica II 12 CFU)</b>
<b>Algoritmi e strutture dati (V.O., 5 CFU)</b>
<b>Algoritmi e strutture dati (Nettuno, 6 CFU)</b>

**\*\*Appello del 18-02-2020 – a.a. 2019-20 – 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 seguente codice Java implementa un algoritmo che determina la lunghezza del più lungo sub-array non decrescente di una data porzione di un array di `int` (da indice `i` a indice `j`), ed è basato sul paradigma del divide-et-impera.

```

1 private static int[] sottoseq(int a[], int i, int j) {
2     // assume che i <= j e che a[i] e a[j] esistono
3     int ret[] = new int[4];
4     /* ret[0] il miglior risultato
5      * ret[1] inizio del miglior risultato
6      * ret[2] lungh. prefisso
7      * ret[3] lungh. suffisso
8      */
9     if(i == j) {
10         ret[0] = 1;
11         ret[1] = i;
12         ret[2] = 1;
13         ret[3] = 1;
14         return ret;
15     }
16     int m = (i+j)/2;
17     int s1[] = sottoseq(a, i, m);
18     int s2[] = sottoseq(a, m+1, j);
19     boolean ord1 = s1[0] == m-i+1; // I subarray ordinato
20     boolean ord2 = s2[0] == j-m; // II subarray ordinato
21     boolean cont = a[m] <= a[m+1]; // suffisso1 continua con prefisso2
22     if(cont && (s1[3]+s2[2] > Math.max(s1[0], s2[0]))) {
23         ret[0] = s1[3]+s2[2];
24         ret[1] = m+1-s1[3];
25     } else if(s1[0] > s2[0]) {
26         ret[0] = s1[0];
27         ret[1] = s1[1];
28     } else {
29         ret[0] = s2[0];
30         ret[1] = s2[1];
31     }
32     ret[2] = s1[2]; if(ord1 && cont) ret[2] += s2[2];
33     ret[3] = s2[3]; if(ord2 && cont) ret[3] += s1[3];
34     return ret;
35 }

```

Si risponda ai seguenti quesiti:

1. Determinare il costo temporale asintotico dell'algoritmo `sottoseq`, con particolare riferimento al caso in cui venga invocato con `sottoseq(a, 0, a.length-1)`.

**Punteggio: [4/30]**

2. Determinare il costo spaziale asintotico dell'algoritmo `sottoseq`, con particolare riferimento al caso in cui venga invocato con `sottoseq(a, 0, a.length-1)`.

**Punteggio: [3/30]**

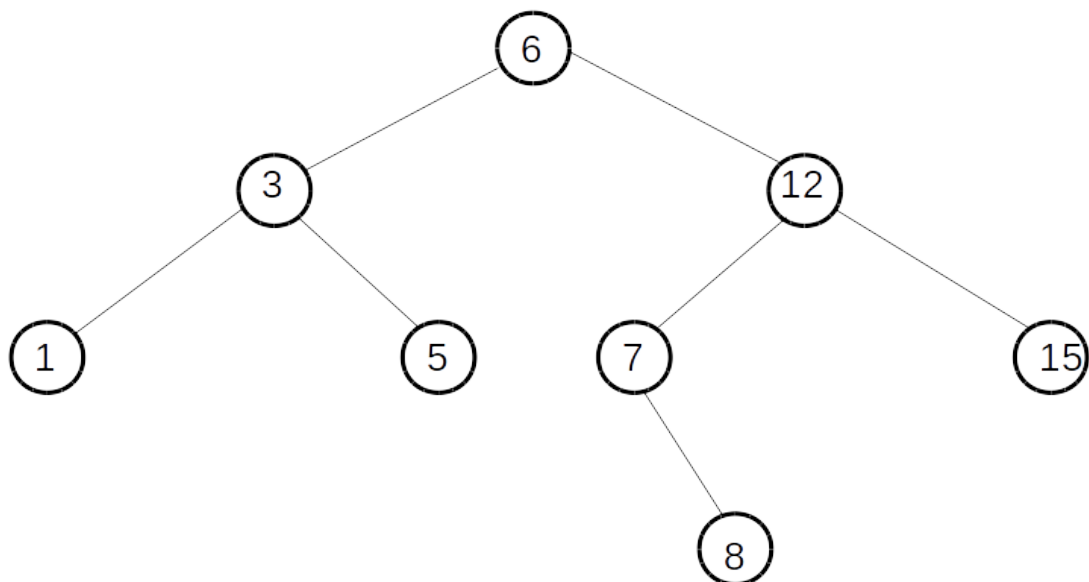
## Quesito 2: Progetto algoritmi C/Java [soglia minima: 5/30]

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 struttura `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 già disponibili le primitive di manipolazione del BST (contenute 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 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, 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 il metodo `LinkedList ordina(BST t)` della classe `BSTServices.java` (in C: la funzione `void ordina (bst *b)` del file `bst_services.c` ) che, ricevuto in ingresso il riferimento a un oggetto di classe `BST` (in C: tipo `bst` ), restituisce una `LinkedList` contenente tutte le chiavi ordinate in senso *decrescente* (stampa l'insieme di tutte le chiavi ordinate in senso *decrescente*). 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 (stampare) la lista

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

**Punteggio: [3/30]**

2. Implementare il metodo `LinkedList outer_range(BST t, int k1, int k2)` della classe `BSTServices` (in C: la funzione `void outer_range(bst *t, int k1, int k2)` del file `bst_services.c` ) che, dato un BST `t` e due chiavi `k1` e `k2` , restituisce una lista (in C: stampa una sequenza) *ordinata in senso crescente*, contenente tutte le chiavi presenti in `t` che *non sono contenute* nell'intervallo  $[k_1, k_2]$ . Se tale intervallo contiene tutte le chiavi presenti, il metodo/funzione restituirà (stamperà) una lista vuota. Ad esempio, con riferimento alla Fig. 1, se  $k_1 = 5$  e  $k_2 = 12$ , il programma deve restituire (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  $[k_1, k_2]$  che sono presenti nel BST.

**Punteggio: [4/30]**

3. Implementare la funzione/metodo `int altezza(BST t)` della classe `BSTServices` (o `int altezza(bst *b)` del file `bst_services.c` in C) che, dato un BST `t` , ne calcola l'altezza, ovvero la massima distanza fra la radice e una foglia. 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 di dimensione iniziale  $N = 7$ . Si supponga (per semplicità) che la dimensione della tabella venga semplicemente raddoppiata e si esegua il conseguente rehashing ogni volta che il fattore di carico diventa pari ad

almeno  $1/2$ . Calcolare il numero di volte che ciò avviene a seguito di  $n$  inserimenti consecutivi (si supponga che non vi siano mai cancellazioni). *Occorre offrire un' argomentazione quantitativa e convincente.*

**Punteggio: [3/30]**

2. Si consideri un heap *minimale* a chiavi intere. Si supponga che vengano inserite, in successione,  $n$  chiavi in ordine *decrescente*. Calcolare il costo asintotico complessivo per la successione degli  $n$  inserimenti. Rispondere alla stessa domanda nel caso le chiavi siano inserite in ordine *crescente*. *In entrambi i casi occorre giustificare adeguatamente 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 *ampiezza* (BFS), *pre-ordine* e *post-ordine*.

**Punteggio: [3/30]**

#### Quesito 4:

Una società di telecomunicazioni deve realizzare una dorsale in fibra ottica destinata a garantire la connettività tra  $n$  località principali. Il costo per stabilire un collegamento tra due località  $u$  e  $v$  è *direttamente proporzionale* alla distanza chilometrica  $w_{uv}$  tra di esse.

L'obiettivo è progettare un algoritmo `backbone` che restituisca le coppie di località da connettere mediante un collegamento in fibra ottica in modo tale che: 1) ogni località sia raggiungibile da ogni altra località mediante una successione di tratte in fibra ottica; 2) il costo complessivo delle tratte in fibra ottica da realizzare sia minimo. Per essere precisi, il costo è pari a  $\sum_{(u,v) \in C} w_{uv}$ , dove  $C$  è l'insieme delle coppie tra ciascuna delle quali verrà realizzato un collegamento in fibra ottica. Ciò premesso si risponda alle domande seguenti:

- Definire con precisione il grafo usato per rappresentare il problema, specificando cosa rappresentano i nodi, qual è l'insieme degli archi e cosa rappresentano gli eventuali pesi sugli archi. Descriverne ulteriori proprietà di interesse (grafo semplice, connesso, orientato o meno, ecc.). Infine, definire in termini di problemi su grafi l'input e l'output dell'algoritmo `backbone`.

**Punteggio: [3/30]**

- Si descriva (è sufficiente lo pseudo-codice o comunque una descrizione dettagliata) l'algoritmo `backbone`. La descrizione può essere anche ad alto livello concettuale ed usare primitive non elementari, ad esempio corrispondenti ad algoritmi noti studiati nel corso. Descrizioni basate su (pseudo) codice C/Java sono comunque considerate accettabili.

**Punteggio: [3/30]**