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 17-06-2021 – a.a. 2020-21 – Tempo: 2 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 1). Svolgere il compito nel modo seguente:

- Per il quesito 1, 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 1, 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 2 e 3, creare due file `probl2.txt` e `probl.txt` contenenti, rispettivamente, gli svolgimenti dei problemi proposti nei quesiti 2 e 3; 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 1. 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.

Avviso importante 2. Il codice *deve compilare* correttamente (warning possono andare, ma niente errori di compilazione), altrimenti riceverà 0 punti. Si raccomanda quindi di scrivere il programma in modo incrementale, ricompilando il tutto di volta in volta.

Quesito 1: Progetto algoritmi C/Java [soglia minima per superare l'esame: 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 (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. In particolare, per il linguaggio C si rimanda agli header file (`.h`)

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.

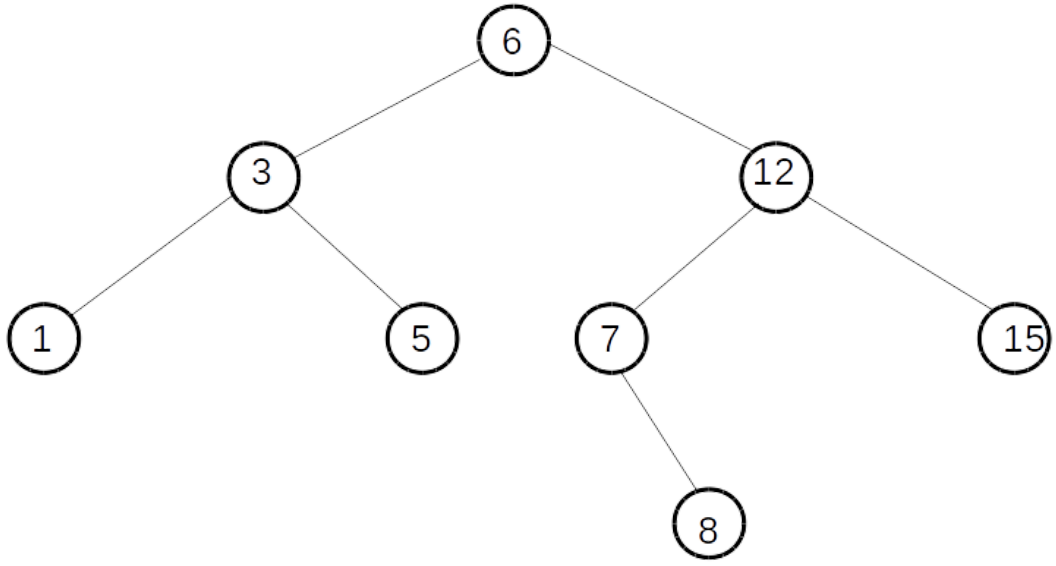

Fig. 1: un albero binario di ricerca. Si noti che nella figura sono mostrate soltanto le chiavi associate ai nodi, rispetto alle quali è definito l'ordinamento.

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 `LinkedList ordina(BST t)` della classe `BSTServices.java` (`bst_services.c` in C) che, ricevuto in ingresso il riferimento a un oggetto di classe `BST`, restituisce una lista (`LinkedList` in Java), contenente 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 la lista

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

   Punteggio: [10/30]

   Nota: potrebbe essere utile implementare una funzione C (o metodo privato Java) ausiliaria, che calcola l'ordinamento per il sotto-albero avente radice nel generico nodo v del BST.

Quesito 2: Algoritmi

1. Si consideri una tabella hash di dimensione $N = 13$ con gestione delle collisioni mediante scansione lineare (*linear probing*) e funzione hash (di compressione) $h(k) = k \mod N$, dove k è la chiave. Descrivere la *successione degli stati* della tabella (ossia l'evoluzione del suo contenuto) a seguito dell'inserimento delle chiavi seguenti nella successione indicata: $10, 26, 52, 76, 13, 8, 3, 33, 60, 42$.

   Segnalare gli inserimenti che danno luogo a collisione, specificando le posizioni coinvolte. Il punteggio dipenderà anche dalla completezza e dalla chiarezza espositiva. Non occorre scrivere tanto ma scrivere bene.

   Punteggio: [8/30]

2. Si consideri una lista non ordinata rappresentata con un array. L'array è inizializzato a una dimensione costante k. Si consideri ora una successione di n inserimenti. Il generico inserimento ha costo costante *a meno che l'array non sia già pieno*. Se l'array è pieno l'inserimento determina i) l'allocazione di un *nuovo* array di dimensione x + k se x era la dimensione precedente dell'array, ii) la copia del contenuto del vecchio array nel nuovo (la copia del *singolo* valore dal vecchio array al nuovo ha costo costante), iii) l'inserimento del nuovo elemento (nel nuovo array). Si calcoli il *costo temporale complessivo asintotico* della successione degli n inserimenti (si supponga che non vi siano mai rimozioni di elementi).

   Occorre dare un'argomentazione quantitativa, rigorosa (prova) e chiara, giustificando adeguatamente la risposta. Il punteggio dipenderà soprattutto da questo.

   Punteggio: [7/30]

Quesito 3:

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 di località tra le 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: [4/30]

- 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]