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)

Prova del 17-06-2021 – a.a. 2020-21 – Tempo: 2 ore e 30 minuti – somma punti: 32

Istruzioni

1. Per prima cosa si prega di indicare all'inizio del compito i) anno di immatricolazione; ii) se il proprio esame di Fondamenti 2 consiste di Algoritmi e Strutture Dati + Modelli e Linguaggi (*in tal caso scrivere MOD*) oppure Algoritmi e Strutture Dati + Progettazione del Software (*solo studenti che nel periodo che va dal 2014-15 al 2017-18 sono stati iscritti al II anno, estremi inclusi*). *Nel secondo caso scrivere PSW*.
2. **Nota bene.** Per ritirarsi è sufficiente scriverlo all'inizio del file contenente le vostre risposte.

Avviso importante. La risposta al quesito di programmazione va data nel linguaggio scelto (C o Java), usando le interfacce messe a disposizione dal docente. *Il programma va scritto usando l'editor interno di exam.net*.

Mentre non ci aspettiamo che produciate codice compilabile, una parte del punteggio sarà comunque assegnata in base alla leggibilità e chiarezza del codice che scriverete, *a cominciare dall'indentazione*, oltre che rispetto ai consueti requisiti (aderenza alle specifiche ed efficienza della soluzione proposta). Inoltre, errori grossolani di sintassi o semantica subiranno penalizzazioni. *Si noti che l'editor di exam.net consente di indentare i programmi*.

Anche le risposte ai quesiti 2 e 3 vanno scritte usando l'editor interno di exam.net. È possibile integrare i contenuti di tali file con eventuale materiale cartaceo scansionato *unicamente per disegni, grafici, tabelle, calcoli*.

Quesito 1: Progetto algoritmi C/Java [soglia minima per superare la prova: 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 descritte `bst.h` 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 e la funzione dei metodi Java (funzioni C) a disposizione si rimanda all'appendice presente alla fine di questo documento.

Si noti che le primitive messe a disposizione forniscono un insieme base per la manipolazione di BST, ma il problema proposto può essere risolto usandone soltanto un piccolo sottoinsieme.

Per la classe `LinkedList` i principali metodi per la gestione dovrebbero essere noti allo studente, ma alcuni sono riportati in fondo a questo documento per comodità. Analogamente, in fondo a questo documento sono descritti i principali metodi per accedere al tipo `linked_list` in C o per usare iteratori sul tipo `linked_list` .

Le interfacce delle classi/moduli che implementano il BST e i suoi nodi sono descritti nell'appendice di questo documento. Sono riportati alcuni campi/metodi/funzioni utili allo svolgimento degli esercizi proposti. Si noti che molti di questi potrebbero non essere necessari.

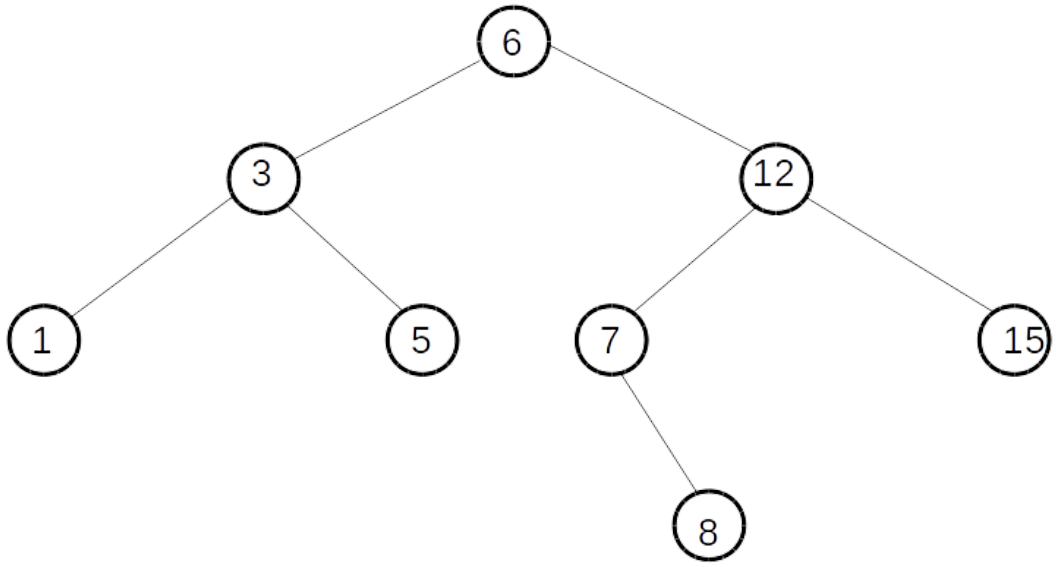


Figura: un albero binario di ricerca. Si noti che nella figura sono mostrate soltanto le chiavi associate ai nodi, rispetto alle quali è definito l'ordinamento.

Tutto ciò premesso, risolvere quanto segue in Java o C (non è ammesso *pseudo-codice*). L'esempio di seguito descritto fa riferimento alla figura in alto.

1. Implementare la funzione/metodo `LinkedList outer_range(BST t, int k1, int k2)` della classe `BSTServices` (o `bst_services.c` in 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 $[k1, k2]$. Se tale intervallo contiene tutte le chiavi presenti, il metodo/funzione restituirà una lista vuota. Ad esempio, con riferimento alla Fig. 1, se $k1 = 5$ e $k2 = 12$, il programma deve restituire 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: [10/30]

Quesito 2: Algoritmi

1. Si consideri una tabella hash di dimensione $N = 7$ con gestione delle collisioni mediante scansione lineare 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 di 5 chiavi $k_1, \ldots, k_5$ così ottenute: per $i = 1, \ldots, 5$, k_i è l'intero a due cifre che ha i nella posizione delle decine e l'i-esima cifra della vostra matricola (a partire

dall'ultima e procedendo verso sinistra) nella posizione delle unità.
Ad esempio se la vostra matricola fosse 1234567, avremmo $k_1 = 17$, $k_2 = 26$ e così via.

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.

Nota: non usare la propria matricola per rispondere al quesito comporta una penalizzazione.

Punteggio: [5/30]

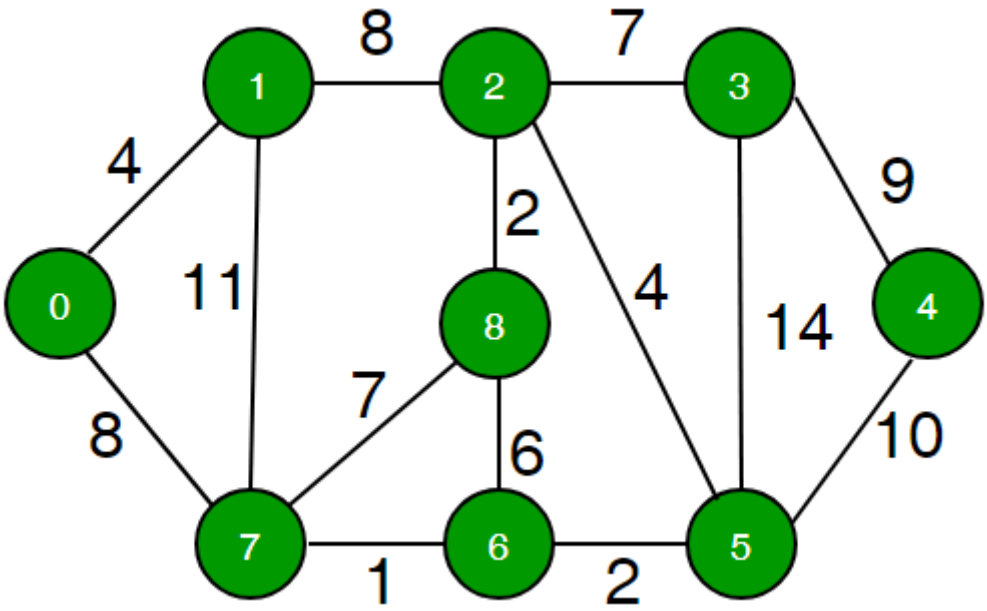
2. Si consideri un heap minimale realizzato con *array*. L'array è inizializzato a una dimensione k ed è inizialmente vuoto. Ogni volta che l'array è pieno, il successivo inserimento nell'heap determina i) l'allocazione di un nuovo array di dimensione *doppia* rispetto alla precedente ii) la copia del contenuto dell'array contenente il vecchio heap nella prima metà del nuovo array e iii) l'inserimento del nuovo elemento nel nuovo heap così creato.

   Si calcoli il costo complessivo asintotico di caso peggiore di una successione di n inserimenti (si supponga che non vi siano mai rimozioni di chiavi).

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

Punteggio: [5/30]

3. Si consideri il grafo non diretto e pesato nella figura sottostante.



   Si consideri la partizione (o taglio) corrispondente al sottoinsieme dei nodi costituito dal nodo v avente etichetta $n \mod 8$ e da tutti i vicini di v, dove n è l'ultima cifra del vostro numero di matricola. Ad esempio, se l'ultima cifra del vostro numero di matricola fosse 7, v sarebbe il nodo avente etichetta 7 e il sottoinsieme da considerare sarebbe l'insieme di 7 e i suoi vicini, ossia $\{0, 1, 6, 7, 8\}$.

   Dimostrare che l'arco di peso massimo (*tra quelli che attraversano il taglio così individuato*) non può far parte di un albero ricoprente minimo per il grafo considerato. Nell'esempio precedente l'arco di peso massimo è quello di peso 8 associato all'arco $(1, 2)$.

   Si noti che non bisogna calcolare un albero ricoprente minimo, ma semplicemente dimostrare quanto richiesto. Il punteggio dipenderà dal rigore dell'argomentazione proposta e dalla chiarezza espositiva. Non occorre scrivere tanto ma scrivere bene.

   Punteggio: [5/30]

Quesito 3:

Mario è iscritto a un corso di studi universitari che, anziché fornire un piano di studi preciso, presenta il seguente elenco di esami da superare, corredato da una lista di propedeuticità: per ogni esame, è dato un elenco di altri esami (possibilmente vuoto e indicato fra parentesi) che occorre aver superato prima di sostenere l'esame considerato:

Analisi1: ( )

Analisi2: (Analisi1)

Fisica1: (Analisi1, Analisi2)

Fisica2: (Fisica1)

Programmazione1: ( )

Calcolo1: (Analisi2, Fisica2)

Elettronica: (Fisica2, Programmazione1)

Disegno: ( )

Esame_finale: (Analisi1, Analisi2, Fisica1, Fisica2, Programmazione1, Calcolo1, Elettronica, Disegno)

Ciò premesso, proporre un possibile percorso di studi per Mario, ossia una possibile sequenza di esami da sostenere, il cui ordine rispetti i vincoli descritti sopra (si noti che in generale più sequenze potrebbero essere ammissibili). Più precisamente

1. Si descriva come rappresentare l'insieme delle propedeuticità usando un grafo diretto. Occorre disegnare il grafo, in modo che sia chiaro cosa rappresentino i nodi e cosa rappresentino gli archi.

   Punteggio: [4/30]

2. Si proponga un algoritmo per risolvere il problema e lo si applichi al caso in esame, restituendo una sequenza ammissibile di esami da sostenere. Nel caso si riconduca il problema in esame a un problema noto è sufficiente specificare il problema e indicare l'algoritmo usato.

   Punteggio: [3/30]

Nota bene: il punteggio dipenderà dalla chiarezza nell'esposizione e dalla solidità delle argomentazioni proposte.

Appendice: interfacce dei moduli/classi per il quesito 1

Di seguito sono descritti i campi/moduli/funzioni delle classi Java o moduli C che si suppone siano utilizzabili.

Interfacce Java

In Java si suppone sia suppone siano già implementate e disponibili le classi `Node` e `BST`, che rispettivamente implementano il generico nodo e un BST. Ovviamente, in Java si assumono disponibili tutte le classi delle librerie standard, come ad esempio `java.util.LinkedList` ecc.

Classe Node

```
class Node<V> {
    public int key; // Chiave
    public V value; // Valore
    public Node<V> left; // Riferimento al figlio sinistro
    public Node<V> right; // Riferimento al figlio destro

    public Node(int key, V value) {
        this.key = key;
        this.value = value;
    }
}
```

Metodi messi a disposizione dalla classe BST. Di seguito si descrivono lo scheletro e le interfacce dei principali metodi messi a disposizione dalla classe `BST`. Si noti che solo alcuni potrebbero essere utili ai fini dell'esercizio.

```
public class BST<V> {

    private Node<V> root;
    private int size; // Numero di nodi del BST


    public BST(int key, V value) {
        this.root = new Node<V>(key, value);
        this.size = 1;
    }

    // Restituisce un riferimento alla radice del BST
    public Node<V> root()

    // Restituisce il numero di nodi presenti nel BST
    public int size()


    // Metodo che inserisce una coppia (chiave, valore) nel BST
    public void insert(int k, V v)


    // Restituisce il valore associato alla chiave k, se esistente
    public V find(int k)


    // Restituisce il valore della chiave minima
    public int findMin()


    // Rimuove il nodo associato alla chiave di valore minimo
    public void removeMin()


    // Rimuove il nodo associato alla chiave k, se esistente
    public void remove(int k)

}
```

Metodi potenzialmente utili della classe LinkedList.

```
// Appende e alla fine della lista. Restituisce true
boolean add(E e);

// Rimuove e restituisce l'elemento in testa alla lista.
E remove(); // E indica il tipo generico dell'elemento
```

Scheletro della classe BSTServices

Di seguito, lo scheletro della classe `BSTServices` con le segnature dei metodi che essa contiene.

```
public class BSTServices {

    public LinkedList outer_range(BST t, int k1, int k2) {
        /* DA IMPLEMENTARE */
    }
}
```

Interfacce C

bst.h (solo tipi principali)

```
#ifndef BST_H
#define BST_H

#ifdef  __cplusplus
extern "C" {
#endif

typedef struct _bst_node {
    int key; // Chiave
    void * value; // Valore
    struct _bst_node * left; // Figlio sinistro
    struct _bst_node * right; // Figlio destro
} _bst_node;

struct bst {
    _bst_node * root;
};

typedef struct bst bst;

// Crea albero vuoto
bst * bst_new(int k, void * v);

// Inserisce una coppia (chiave, valore) nel BST
void bst_insert(bst * t, int k, void * v);

// Restituisce il valore associato alla chiave k
void * bst_find(bst * t, int k);

// Rimuove il nodo associato alla chiave k
void bst_remove(bst * t, int k);
```

```
// Restituisce la chiave minima
int bst_find_min(bst * n);

// Rimuove l'elemento avente chiave minima
void bst_remove_min(bst * n);

// DA IMPLEMENTARE
void outer_range(bst *, int, int);


#ifdef  __cplusplus
}
#endif

#endif  /* BST_H */
```

Scheletro del modulo C bst_services

Di seguito lo scheletro del metodo `bst_services.c` e le segnature delle funzioni da implementare.

```
#include <stdio.h>
#include <limits.h>
#include "bst.h"

void outer_range(bst *t, int k1, int k2) {
    /* DA IMPLEMENTARE */
    return;
}
```

linked_list.h (solo parte)

```
typedef struct linked_list_node {
    void *value;
    struct linked_list_node *next;
    struct linked_list_node *pred;
} linked_list_node;

typedef struct linked_list {
    linked_list_node *head;
    linked_list_node *tail;
    int size;
} linked_list;

typedef struct linked_list_iterator linked_list_iterator;

/**********************
      linked_list
**********************/
/**
Crea una nuova lista.
*/
linked_list * linked_list_new();
```

```c
/**
Aggiunge in testa alla lista ll, un nodo che punta a value.
*/
void linked_list_insert_head(linked_list* ll, void* value);

/**
Aggiunge in coda alla lista ll, un nodo che punta a value.
*/
void linked_list_insert_tail(linked_list* ll, void* value);

/**
Come linked_list_insert_tail(linked_list* ll, void* value).
*/
void linked_list_add(linked_list * ll, void * value);


/**
Aggiunge alla lista ll un nodo che punta a value, subito dopo predec
*/
void linked_list_insert_after(linked_list * ll, linked_list_node *predec, void
* value);


/**
Rimuove dalla lista ll il nodo in testa e ritorna il valore puntato da tale
nodo.
*/
void *linked_list_remove_head(linked_list* ll);

/**
Rimuove dalla lista ll il nodo in coda e ritorna il valore puntato da tale
nodo.
*/
void* linked_list_remove_tail(linked_list * ll);

/**
Ritorna un puntatore al valore puntato dal nodo in input.
*/
void *linked_list_node_getvalue(linked_list_node* node);

/**
Ritorna la dimensione della lista ll.
*/
int linked_list_size(linked_list *ll);


/**
Ritorna 1 se la linked list contiene value, 0 altrimenti.
*/
int linked_list_contains(linked_list *ll, void *value);


/**
Stampa a video una rappresentazione della lista ll.
*/
void linked_list_print(linked_list *ll);
```

```c
/**
Distrugge la lista ll e libera la memoria allocata per i suoi nodi. Nota che la funzione
non libera eventuale memoria riservata per i valori puntati dai nodi della lista.
*/
void linked_list_delete(linked_list *ll);


/**********************
  linked_list_iterator
**********************/
/**
Crea un nuovo iteratore posizionato sul primo elemento della lista ll.
*/
linked_list_iterator * linked_list_iterator_new(linked_list *ll);

/**
Ritorna 1 se l'iteratore iter ha un successivo, 0 altrimenti.
*/
int linked_list_iterator_hasnext(linked_list_iterator* iter);


/**
Muove l'iteratore un nodo avanti nella lista e ritorna il valore puntato dal nodo
appena oltrepassato, o NULL se l'iteratore ha raggiunto la fine della lista.
*/
//void * linked_list_iterator_next(linked_list_iterator * iter);
linked_list_node * linked_list_iterator_next(linked_list_iterator * iter);

/**
Rimuove dalla lista il nodo ritornato dall'ultima occorrenza della funzione linked_list_iterator_next.
*/
void *linked_list_iterator_remove(linked_list_iterator * iter);

/**
Ritorna il valore puntato dal nodo su cui si trova attualmente l'iteratore iter.
*/
//void * linked_list_iterator_getvalue(linked_list_iterator *iter);

/**
Distrugge l'iteratore e libera la memoria riservata. Nota che questa operazione non ha nessun effetto sulla lista puntata dall'iteratore.
*/
void linked_list_iterator_delete(linked_list_iterator* iter);
```