Esame di Programmazione II

Appello di giorno 5 Ottobre 2016
Università degli Studi di Catania - Corso di Laurea in Informatica
- Prova A -

Testo della Prova

Definizione Iniaziale.
Un *Foresta Crescente* di grado k (IF, *Increasing Forest*) è un insieme di alberi, ordinati in senso non decrescente rispetto alla loro altezza, ogni albero ha un'altezza massima pari a k. Dato un albero t indicheremo con $h(t)$ la sua altezza e con $m(t)$ la sua chiave più piccola. Gli alberi contenuti all'interno di una Foresta Crescente, sono ordinati in base ai seguenti criteri:

- se $h(t_1) > h(t_2)$ allora $t_1 > t_2$
- se $h(t_1) = h(t_2)$ e $m(t_1) > m(t_2)$ allora $t_1 > t_2$
- se $h(t_1) = h(t_2)$ e $m(t_1) = m(t_2)$ allora $t_1 = t_2$

Una Foresta crescente è implementata mediante una lista concatenata di alberi. Se $t_1 = t_2$ e t_1 è stato inserito prima di t_2 allora t_1 appare prima di t_2 nella lista concatenata. Quando un elemento x viene inserito nella struttura dati esso va a finire nel primo albero della lista (quello con altezza più piccola), a meno che esso non abbia altezza pari a k. In quest'ultimo caso verrà creato un nuovo albero vuoto in cui l'elemento x verrà inserito.

1. Si fornisca una classe `C++`, denominata `LList<H>`, che implementi una lista linkata e ordinata, contenente (almeno) i seguenti metodi.
   (a) `LList<H>* ins(H x)` aggiunge un nuovo elemento x alla lista. La funzione restituisce un puntatore ad un oggetto di tipo `LList<H>`;
   (b) `LList<H>* canc(H x)` elimina l'elemento x dalla lista, se presente. La funzione restituisce un puntatore ad un oggetto di tipo `LList<H>`;
   (c) `H* search(H x)` restituisce il puntatore all'elemento x, se esso è presente nella struttura dati. Restituisce `NULL` altrimenti;
   (d) `void print()` è una procedura che stampa in output gli elementi della lista. La stampa procede dall'elemento più piccolo all'elemento più grande della lista.

2. Si fornisca una classe `C++`, denominata `BST<H>`, che implementi un albero binario di ricerca, contenente (almeno) i seguenti metodi.
   (a) `BST<H>* ins(H x)` aggiunge un nuovo elemento x all'albero. La funzione restituisce un puntatore ad un oggetto di tipo `BST<H>`;
   (b) `BST<H>* canc(H x)` elimina l'elemento x dall'albero, se presente. La funzione restituisce un puntatore ad un oggetto di tipo `BST<H>`;
   (c) `H* search(H x)` restituisce il puntatore all'elemento x, se esso è presente nella struttura dati. Restituisce `NULL` altrimenti;
   (d) `void print()` è una procedura che stampa in output gli elementi dell'albero. La stampa procede dall'elemento più piccolo all'elemento più grande della lista.
   (e) `H* minimum()` restituisce il puntatore all'elemento più piccolo dell'albero. Restituisce `NULL` se l'albero non contiene elementi;
   (f) `H* successor(H x)` restituisce il puntatore al successore dell'elemento x, se presente. Restituisce `NULL` se l'elemento x è il più grande tra quelli presenti nella struttura o se x non è presente nella struttura;

3. Si fornisca una classe `C++`, denominata `IF<H>`, che implementi una Foresta Crescente. Il costruttore della classe dovrà prendere in input il grado k della struttura. La classe dovrà contenere (almeno) i seguenti metodi.
   (a) `IF<H>* ins(H` x`)` aggiunge un nuovo elemento x alla struttura dati. La funzione restituisce un puntatore ad un oggetto di tipo `IF<H>`;
   (b) `IF<H>* canc(H` x`)` elimina tutti gli elementi con valore x dalla struttura dati, se presenti. La funzione restituisce un puntatore ad un oggetto di tipo `IF<H>`;
   (c) `void print()` è una procedura che stampa in output gli elementi della struttura dati. La stampa procede dall'albero più piccolo all'albero più grande della lista. per ogni albero la stampa procede dall'elemento più piccolo all'elemento più grande in esso contenuto.

Valutazione.
La prova d'esame verrà valutata anche in base all'ouput generato dal proprio programma su specifici input. Nello specifico lo studente, per la verifica del codice, dovrà eseguire il programma utilizzando il seguente main.

```
int main() {
        /* valutazione del primo esercizio : 8 punti*/
        LList<int>* l = new LList();
        l->ins(3)->ins(7)->ins(1)->ins(8)->ins(2)->ins(4)->print();
        l->canc(3)->canc(9)->canc(5)->canc(1)->ins(10)->ins(5)->print();
        if(l->search(5)) cout << "elemento 5 presente"; else cout << "elemento 5 non presente";
        if(l->search(3)) cout << "elemento 3 presente"; else cout << "elemento 3 non presente";
        /* output:  1 2 3 4 7 8
                    2 4 5 7 8 10
                    elemento 5 presente
                    elemento 3 non presente */
        /* valutazione del secondo esercizio : 10 punti */
        BST<int>* t = new BST();
        t->ins(3)->ins(7)->ins(1)->ins(8)->ins(2)->ins(4)->print();
        t->canc(3)->canc(9)->canc(5)->canc(1)->ins(10)->ins(5)->print();
        if(l->search(5)) cout << "elemento 5 presente"; else cout << "elemento 5 non presente";
        if(l->search(3)) cout << "elemento 3 presente"; else cout << "elemento 3 non presente";
        int *r = t->minimum();
        if(r) cout << "il valore più piccolo è " << *r;
        if(r=t->successor(5)) cout << "il successore di 5 è " << *r;
        if(r=t->successor(3)) cout << "il successore di 3 è " << *r;
        /* output:  1 2 3 4 7 8
                    2 4 5 7 8 10
                    elemento 5 presente
                    elemento 3 non presente
                    il valore più piccolo è 2
                    il successore di 5 è 7 */
        /* valutazione del terzo esercizio : 12 punti */
        IF<int>* b = new IF(3);
        b->ins(3)->ins(14)->ins(2)->ins(5)->ins(8)->ins(9)->print();
        b->ins(4)->ins(7)->ins(11)->ins(5)->ins(6)->ins(12)->print();
        b->canc(11)->canc(3)->canc(2)->canc(9)->ins(9)->print();
        /* output:
                    9 2 3 5 8 14
                    6 12 2 3 5 8 14 4 5 7 9 11
                    4 5 7 5 8 14 6 9 12 */
}
```