Distanza successore TUTTO

mercoledì 28 giugno 2023 22:12

```cpp
#include <iostream>
#include <queue>
using namespace std;
template <typename T>
struct Nodo {
    T val;
    Nodo<T>* left;
    Nodo<T>* right;
    Nodo<T>* padre;
};
template <typename T>
class BST {
public:
    BST(): radice(nullptr) {}
    void inserisci (T valore);
    void inOrder() const;
    void stampaLivelli() const;
    int distanzaSuccessore(T x) const;

    template <typename U>
    friend ostream& operator<<(ostream& out, const BST<U>& t);

private:
    Nodo<T>* radice;
    int numElementi = 0;

    void inOrder(Nodo<T>*) const;
    Nodo<T>* minimo(Nodo<T>* x) const;
    Nodo<T>* successore(Nodo<T>* x) const;
};
template <typename T>
void BST<T>::inserisci (T valore){
    Nodo<T>* nuovo = new Nodo<T>;
    Nodo<T>* x = radice;
    Nodo<T>* y = nullptr;

    nuovo->val = valore;
    nuovo->left = nuovo->right = nullptr;

    while(x!=nullptr){
        y = x;
        if(valore < x->val)
            x = x->left;
        else
            x = x->right;
    }

    nuovo->padre = y;
    if(y==nullptr)
        radice = nuovo;
    else if(valore < y->val)
        y->left = nuovo;
    else
        y->right = nuovo;
```

```
    numElementi++;
}
template <typename T>
void BST<T>::inOrder() const { inOrder(radice); }
template <typename T>
void BST<T>::inOrder(Nodo<T>* p) const {
    if(p!=nullptr){
        inOrder(p->left);
        cout << p->val << "\t";
        inOrder(p->right);
    }
}
template <typename T>
Nodo<T>* BST<T>::minimo(Nodo<T>* x) const{
    while(x && x->left != nullptr)
        x = x->left;
    return x;
}
template <typename T>
Nodo<T>* BST<T>::successore(Nodo<T>* x) const{
    if(x->right != nullptr)
        return minimo(x->right);

    Nodo<T>* y = x->padre;
    while(y != nullptr && x == y->right){
        x = y;
        y = y->padre;
    }
    return y;
}
template <typename T>
int BST<T>::distanzaSuccessore(T x) const {
    Nodo<T>* nodo = radice;
    while(nodo != nullptr){
        if(nodo->val == x){
            Nodo<T>* succ = successore(nodo);
            if(succ == nullptr)
                return 0;
            int distanza = 0;
            while(nodo != succ){
                ++distanza;
                nodo = succ;
                if (nodo == nullptr) { // Aggiunto questo controllo
                    break;
                }
                succ = successore(nodo);
            }
            return distanza;
        }
        else if(nodo->val > x)
            nodo = nodo->left;
        else
            nodo = nodo->right;
    }
    return 0; // Valore x non trovato nell'albero
}


template <typename T>
ostream& operator<<(ostream& out, const BST<T>& t) {
    t.inOrder();
    return out;
```

```
}
template <typename T>
void BST<T>::stampaLivelli() const {
    if (radice == nullptr) return;

    queue<Nodo<T>*> q;
    q.push(radice);

    while (!q.empty()) {
        int size = q.size();
        for (int i = 0; i < size; i++) {
            Nodo<T>* node = q.front();
            q.pop();
            cout << node->val << ' ';

            if (node->left) q.push(node->left);
            if (node->right) q.push(node->right);
        }
        cout << endl;
    }
}
int main() {
    BST<int> bstInt;
    bstInt.inserisci(10);
    bstInt.inserisci(5);
    bstInt.inserisci(15);
    bstInt.inserisci(2);
    bstInt.inserisci(7);
    cout << "Albero BST di interi (in ordine): ";
    bstInt.inOrder();
    cout << endl << "Albero BST di interi (livello per livello): " << endl;
    bstInt.stampaLivelli();
    // cout << "Distanza dal successore per il nodo con valore 5: "
<< bstInt.distanzaSuccessore(5) << endl;
    cout << "Distanza dal successore per il nodo con valore 15: "
<< bstInt.distanzaSuccessore(15) << endl;
    BST<char> bstChar;
    bstChar.inserisci('a');
    bstChar.inserisci('c');
    bstChar.inserisci('b');
    cout << "Albero BST di char (in ordine): ";
    bstChar.inOrder();
    cout << endl << "Albero BST di char (livello per livello): " << endl;
    bstChar.stampaLivelli();
    cout << "Distanza dal successore per il nodo con valore 'a': "
<< bstChar.distanzaSuccessore('a') << endl;
    cout << "Distanza dal successore per il nodo con valore 'c': "
<< bstChar.distanzaSuccessore('c') << endl;
    return 0;
}
```

Testo

mercoledì 28 giugno 2023 22:13

Esercizio 1: BST – distanza successore

Si implementi una semplice classe BST, attraverso l'utilizzo dei template, contenente elementi di tipo generico T. La classe dovrà implementare le seguenti funzionalità:

- inserimento di un nuovo elemento
- visualizzare gli elementi del BST
- un metodo 'distanza successore' che prende in input un valore X e restituisce il numero di archi che separano il nodo contenente X dal nodo contenente il suo successore. Se la chiave non ha successore il metodo restituisce il valore zero.

Nel metodo main, istanziare un BST di interi ed un BST di char, successivamente inserire dei valori e testare il corretto funzionamento delle funzionalità implementate.

BILANCIAMENTO DUPLICATI

mercoledì 28 giugno 2023 22:15

```
#include <iostream>
#include <queue>
using namespace std;
template <typename T>
struct Nodo {
    T val;
    Nodo<T>* left;
    Nodo<T>* right;
    Nodo<T>* padre;
    bool lastDupInsertedLeft;
};
template <typename T>
class BST {
    public:
        BST(): radice(nullptr) {}
        void inserisci(T valore);
        void stampaLivelli() const;
        friend ostream& operator<<(ostream& os, const BST<T>& bst) {
            bst.inOrder(bst.radice, os);
            return os;
        }
    private:
        Nodo<T>* radice;
        void inOrder(Nodo<T>*, ostream&) const;
};
template <typename T>
void BST<T>::inserisci(T valore) {
    Nodo<T>* nuovo = new Nodo<T>;
    Nodo<T>* x = radice;
    Nodo<T>* y = nullptr;
    nuovo->val = valore;
    nuovo->left = nuovo->right = nullptr;
    nuovo->lastDupInsertedLeft = false;
    while(x != nullptr) {
        y = x;
        if(valore < x->val)
            x = x->left;
        else if(valore > x->val)
            x = x->right;
        else {
            if(x->lastDupInsertedLeft) {
                x->right = nuovo;
                nuovo->padre = x;
            } else {
                x->left = nuovo;
                nuovo->padre = x;
            }
            x->lastDupInsertedLeft = !x->lastDupInsertedLeft;
            return;
        }
    }
    nuovo->padre = y;
    if(y == nullptr)
        radice = nuovo;
    else if(valore < y->val)
```

```
        y->left = nuovo;
    else
        y->right = nuovo;
}
template <typename T>
void BST<T>::inOrder(Nodo<T>* p, ostream& os) const {
    if(p != nullptr) {
        inOrder(p->left, os);
        os << p->val << " ";
        inOrder(p->right, os);
    }
}
template <typename T>
void BST<T>::stampaLivelli() const {
    if (radice == nullptr) {
        return;
    }
    queue<Nodo<T>*> nodi;
    nodi.push(radice);
    while (!nodi.empty()) {
        int numNodi = nodi.size();
        while (numNodi > 0) {
            Nodo<T>* nodo = nodi.front();
            nodi.pop();
            cout << nodo->val << " ";
            if (nodo->left != nullptr) {
                nodi.push(nodo->left);
            }
            if (nodo->right != nullptr) {
                nodi.push(nodo->right);
            }
            numNodi--;
        }
        cout << endl;
    }
}
int main() {
    BST<int> bstInt;
    bstInt.inserisci(5);
    bstInt.inserisci(5);
    bstInt.inserisci(3);
    bstInt.inserisci(7);
    bstInt.inserisci(7);
    bstInt.inserisci(7);
    cout << "BST<int> in ordine: " << bstInt << endl;
    cout << "BST<int> livello per livello: " << endl;
    bstInt.stampaLivelli();
    BST<char> bstChar;
    bstChar.inserisci('e');
    bstChar.inserisci('e');
    bstChar.inserisci('c');
    bstChar.inserisci('g');
    bstChar.inserisci('g');
    bstChar.inserisci('g');
    cout << "\nBST<char> in ordine: " << bstChar << endl;
    cout << "BST<char> livello per livello: " << endl;
    bstChar.stampaLivelli();
    return 0;
}
```

Testo

mercoledì 28 giugno 2023 22:15

Esercizio 2: BST - Bilanciamento duplicati

Si implementi una semplice classe BST, attraverso l'utilizzo dei template, contenente elementi di tipo generico T. La classe dovrà implementare le seguenti funzionalità:

- inserimento di un nuovo elemento
- visualizzare gli elementi del BST

Inoltre, tale classe BST dovrà essere in grado di gestire le chiavi duplicate. In particolare, le chiavi duplicate dovranno essere inserite nei sottoalberi destro o sinistro della chiave duplicata già presente nel BST in modo alternato.

Nel metodo main, istanziare un BST di interi ed un BST di char, successivamente inserire dei valori e testare il corretto funzionamento delle funzionalità implementate.

main.cpp tutto

mercoledì 28 giugno 2023 22:27

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
using namespace std;
template <class H> class Node{
public:
    H key;
    Node<H> *parent, *left, *right;
    Node(H key){
        this->key = key;
        parent = left = right = NULL;
    }
};
template <class H> class BST{
private:
    Node<H> *root;
    Node<H>* _search(Node<H>* x, H k){
        if(!x) return NULL;
        if(k==x->key) return x;
        if(k<x->key) return _search(x->left, k);
        return _search(x->right, k);
    }
    Node<H>* _minimum(Node<H>* x){
        if(!x || !x->left) return x;
        return _minimum(x->left);
    }
    void _inorder(Node<H>* x, ostream& os){
        if(!x) return;
        _inorder(x->left, os);
        os << x->key << " ";
        _inorder(x->right, os);
    }
    int _distanzaChiavi(Node<H>* x, H k, H h){
        if(!x) return -1;
        if(k > h){
            return _distanzaChiavi(x, h, k);
        }
        if(k < x->key && h < x->key){
            return _distanzaChiavi(x->left, k, h);
        }
        if(k > x->key && h > x->key){
            return _distanzaChiavi(x->right, k, h);
        }
        return _depth(x, k) + _depth(x, h);
    }
    int _depth(Node<H>* x, H k){
        if(k == x->key){
            return 0;
        }
        if(k < x->key){
            return 1 + _depth(x->left, k);
        }
        return 1 + _depth(x->right, k);
    }
public:
    BST(){
```

```
        root = NULL;
    }
    BST<H>* insert(H k){
        Node<H>* x = root;
        Node<H>* p = NULL;
        Node<H>* y = new Node<H>(k);
        while(x){
            p = x;
            if(k<=x->key) x = x->left;
            else x = x->right;
        }
        y->parent = p;
        if(!p) root = y;
        else if(k<=p->key) p->left = y;
        else p->right = y;
        return this;
    }
    void inorder(ostream& os){
        _inorder(root, os);
        os << endl;
    }
    int distanzaChiavi(H k, H h){
        return _distanzaChiavi(root, k, h);
    }
};
int main(){
    ifstream in("input.txt");
    ofstream out("output.txt");
    string type, op;
    int n;
    while(in >> type >> n){
        if(type == "int"){
            BST<int>* t = new BST<int>();
            for(int i=0; i<n; i++){
                in >> op;
                if(op[0] == 'i'){
                    stringstream ss(op.substr(4, op.size()));
                    int x; ss >> x;
                    t->insert(x);
                }
            }
            int k, h; in >> k >> h;
            t->inorder(cout);
            out << t->distanzaChiavi(k, h) << endl;
        }
        // extend for other types
    }
    return 0;
}
```

Testo

mercoledì 28 giugno 2023 22:27

BST – distanze

Si implementi una classe BST, attraverso l'utilizzo dei template, contenente elementi di tipo generico H. La classe dovrà implementare la procedura di inserimento di un nuovo elemento, una visita che permette di visualizzare il contenuto del BST, ed una procedura `int distanzaChiavi(H k, H h)` che, prese in input due chiavi k e h, sia in grado di calcolare la distanza tra i nodi che contengono k e h rispettivamente.

L'input è suddiviso in 6 task, uno per ogni riga. Ogni riga del file di input è formata da 3+N elementi. Il primo elemento è una stringa che identifica il tipo di dato che dovrà essere contenuto all'interno dell'albero. Il secondo elemento è un valore intero N, il quale rappresenta il numero delle operazioni di inserimento che dovranno essere effettuate nella struttura dati. Seguono N stringhe che rappresentano, nell'ordine dato, le operazioni da svolgere all'interno dell'albero. Nell specifico un inserimento di un valore "v" viene identificato dall stringa "ins:" seguita dal valore "v". Gli ultimi due elementi della sequenza sono i valori k ed h, i quali rappresentano le chiavi contenute nei nodi per i quali si vuole calcolare la distanza.

Il metodo main deve leggere il contenuto del file input.txt, istanziare un BST per ciascun task, visualizzarne il contenuto ed eseguire l'operazione

Esempio:

```
Input:     int 6 ins:10 ins:5 ins:15 ins:13 ins:22 ins:7 7 15

Output:    3
```

Ritaglio schermata acquisito: 28/06/2023 22:28

main TUTTO

mercoledì 28 giugno 2023 22:34

```
#include <iostream>
#include <string>
#include <fstream>
#include <sstream>
using namespace std;
class Film {
    string titolo;
    int anno;
    float valutazione;
public:
    Film(string titolo, int anno, float valutazione) : titolo(titolo),
anno(anno), valutazione(valutazione) {}
    string getTitolo() const { return titolo; }
    int getAnno() const { return anno; }
    float getValutazione() const { return valutazione; }
    friend ostream &operator<<(ostream &os, const Film &film) {
        os << "Titolo: " << film.getTitolo() << ", Anno: " << film.getAnno()
<< ", Valutazione: " << film.getValutazione();
        return os;
    }
};
struct Nodo {
    Film film;
    Nodo *next;
    Nodo(Film film, Nodo *next = NULL) : film(film), next(next) {}
};
class Lista {
    Nodo *head;
public:
    Lista() : head(NULL) {}
    void insert(Film film) {
        Nodo *nodo = new Nodo(film, head);
        head = nodo;
    }
    friend ostream &operator<<(ostream &os, const Lista &lista) {
        Nodo *nodo = lista.head;
        while (nodo != NULL) {
            os << nodo->film << endl;
            nodo = nodo->next;
        }
        return os;
    }
    Nodo* getHead() { return head; }
};
struct NodoBST {
    Film film;
    NodoBST *left, *right;
    NodoBST(Film film) : film(film), left(NULL), right(NULL) {}
};
class BST {
    NodoBST *root;
    void inOrder(NodoBST *nodo) {
        if (nodo == NULL) return;
        inOrder(nodo->left);
        cout << nodo->film << endl;
```

```
        inOrder(nodo->right);
    }
public:
    BST() : root(NULL) {}
    void insert(Film film) {
        NodoBST *nodo = new NodoBST(film);
        if (root == NULL) {
            root = nodo;
            return;
        }
        NodoBST *current = root, *parent = NULL;
        while (current != NULL) {
            parent = current;
            current = (film.getTitolo() < current->film.getTitolo()) ?
current->left : current->right;
        }
        if (film.getTitolo() < parent->film.getTitolo()) parent->left = nodo;
        else parent->right = nodo;
    }
    void inOrder() {
        inOrder(root);
    }
};

int main() {
    ifstream file("film.txt");
    string line;
    Lista lista;
    while (getline(file, line)) {
        stringstream ss(line);
        string titolo, anno, valutazione;
        getline(ss, titolo, ';');
        getline(ss, anno, ';');
        getline(ss, valutazione, ';');
        Film film(titolo, stoi(anno), stof(valutazione));
        lista.insert(film);
    }
    cout << "Lista di Film:\n" << lista << endl;
    BST bst;
    Nodo* nodo = lista.getHead();
    while (nodo != NULL) {
        bst.insert(nodo->film);
        nodo = nodo->next;
    }
    cout << "BST di Film:\n";
    bst.inOrder();
    return 0;
}
```

Testo

mercoledì 28 giugno 2023 22:35

Programmazione 2

Esame di Laboratorio – 14/07/2022

Esercizio 1

Nel file film.txt sono presenti dei film. In ogni riga sono presenti le seguenti informazioni:

TITOLO;ANNO;VALUTAZIONE;

Ad esempio:

TITANIC;1997;7.9

Si legga il file e si inseriscano gli elementi in una lista semplicemente linkata, modellando gli elementi con una opportuna classe, che ridefinisca anche l'operatore <<.

Si rimuovano quindi gli elementi dalla lista e si inseriscano in un BST, che dovrà implementare le funzionalità di:

- Inserimento di un nuovo nodo
- Visualizzazione degli elementi

Scrivere un main che permetta di eseguire tutte le operazioni previste.