Algoritmos de ordenamiento

Fuentes García Alan Kevin
López Manríquez Ángel
Ruiz Lopez Luis Carlos
3CM3

22 de junio de 2019

1. Planteamiento del problema

2. Implementacion del problema

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <stack>

#include <cassert>

#include "bst.h"

using namespace std;

vector<int> read_from_file(const char * path, const int n) {
    vector<int> loaded;
    ifstream stream(path); // flujo de entrada
    int temp, i = 0;
    assert(stream.is_open());
    while (i < n && stream >> temp) {
        loaded.push_back(temp);
        i++;
    }
    stream.close();
    return loaded;
}

void create_subfile(const char *path, const char *name, size_t lines) {
    int i = 0;
    ifstream istream(path);
    ofstream ostream(name);
    string tmp;
    assert(istream.is_open());
    while (i++ < lines && istream >> tmp) ostream << tmp << endl;
    ostream.close();
    istream.close();
}

void swap(int &a, int &b) {
    int tmp = a;
    a = b;
    b = tmp;
}

void bubblesort(vector<int> &v) {
    size_t len = v.size();
    bool change_presented;
    for (int i = 0; i < len; i++) {
        change_presented = false;
        for (int j = 0; j < len - i - 1; j++) {
            if (v[j] > v[j + 1]) {
                swap(v[j], v[j + 1]);
                change_presented = true;
            }
        }
        if (!change_presented) return;
    }
}

void selectionsort(vector<int> &v) {
    size_t n = v.size();
    for (int i = 0; i < n; i++) {
        int minpos = i;
        for (int j = i + 1; j < n; j++) {
            if (v[j] < v[minpos]) {
                minpos = j;
            }
        }
        swap(v[i], v[minpos]);
    }
}

void insertionsort(vector<int> &v) {
    size_t n = v.size();
    for (int i = 1; i < n; i++) {
        int j = i;
        int val = v[i];
        while (j > 0 && v[j - 1] > val) {
            v[j] = v[j - 1];
            j--;
        }
        v[j] = val;
    }
}

void shellsort(vector<int> &v) {
    size_t n = v.size();
    int gap = 1;
    while (gap < n / 3) gap = gap * 3 + 1;
    while (gap > 0) {
        for (int i = gap; i < n; i++) {
            int tmp = v[i];
            int j = i;
            while (j > gap - 1 && v[j - gap] >= tmp) {
                v[j] = v[j - gap];
                j -= gap;
            }
            v[j] = tmp;
        }
        gap = (gap - 1) / 3;
    }
}

void treesort(vector<int> &v) {
    bst<int> tree(v);
    tree.sort(v);

}

void delete_node_from_bst_test() {
    //vector<int> v = { 3, 8, 0, 7, 4, 5 };
    vector<int> v = read_from_file("10millones.txt", 1e4);
    bst<int> tree(v);

    cout << "antes: " << endl;
    //tree.inorder();
    cout << "\nNumero de nodos: " << tree.numberOfNodes() << endl;

    for (int x: v) {
        cout << "\nElemento actual: " << x << endl;
        tree.remove(x);
        //tree.inorder();
        cout << endl;
    }
    cout << "despues: " << endl;
    tree.inorder();
    cout << "Numero de nodos: " << tree.numberOfNodes() << endl;
}

void sorttest() {
    vector<int> v = read_from_file("10millones.txt", 1e1);
```

```cpp
    cout << "leidos" << endl;
    treesort(v);
    cout << "ordenados" << endl;
    for (int i = 0; i < v.size(); i++) cout << i << " " << v[i] << "\n";
}

int main(int argc, char const *argv[]) {
    vector<int> v = read_from_file("10millones.txt", 1e7);
    bst<int> tree(v);
    //for (int x: v) cout << x << ", ";
    cout << endl;
    cout << tree.contains(856834115) << endl;
    cout << tree.contains(966245083) << endl;
    cout << tree.contains(2045206161) << endl;
    return 0;
}

// create_subfile("10millones.txt", "diezmil.txt", 1e4);
//bubblesort(v);
//selectionsort(v);
//insertionsort(v);
//shellsort(v);
```

```cpp
#ifndef bst_h
#define bst_h

#include <iostream>
#include <vector>
#include <stack>

using namespace std;

template < typename T >
class bst {

private:
    struct BinaryNode {
        T data;
        BinaryNode *left, *right, *parent;

        // constructores con iniciadores miembro
        BinaryNode(T mdata, BinaryNode *mleft, BinaryNode *mright, BinaryNode *mparent)
            : data{ mdata }, left{ mleft }, right{ mright }, parent { mparent } { }

        BinaryNode(T mdata)
            : data{ mdata }, left{ nullptr }, right{ nullptr }, parent { nullptr } { }

        BinaryNode()
            : left{ nullptr }, right{ nullptr }, parent { nullptr } { }
    };

    BinaryNode *root;

    BinaryNode *add(BinaryNode *root, T data) {
        if (root == nullptr) { // no se ha insertado nada previamente
            root = new BinaryNode(data);
        } else {
            BinaryNode *parent, *current = root; // empezamos estando en la raiz del
             ↪  arbol
            BinaryNode *toInsert = new BinaryNode(data);
            bool lastLeft = false; // nos indicara si somos izq. o der. de nuestro
             ↪  padre
            while (current != nullptr) {
                parent = current;
                lastLeft = data < current->data;
                if (data < current->data) current = current->left;
                else if (data > current->data) current = current->right;
                else return root; // no permitimos duplicados
            }

            // empezamos a conectar el nuevo nodo
            toInsert->parent = parent;
            if (lastLeft) parent->left = toInsert;
            else parent->right = toInsert;
        }
        return root;
    }

    void inorder(BinaryNode *self) {
        if (self == nullptr) return;
        inorder(self->left);
        cout << self->data << ", ";
        inorder(self->right);
    }

    BinaryNode *min(BinaryNode *root) {
        if (root != nullptr)
            while (root->left != nullptr)
                root = root->left;
        return root;
    }


    inline bool leaf(BinaryNode *node) {
        return node->right == nullptr && node->left == nullptr;
    }

    int numberOfNodes(BinaryNode *root) {
        if (root == nullptr) return 0;
        return numberOfNodes(root->left) + numberOfNodes(root->right) + 1;
    }

    void remove(BinaryNode *&self, BinaryNode *parent, const T &x) {
        if (self == nullptr) return; // no encontrado
        if (x < self->data) remove(self->left, self, x);
        else if (self->data < x) remove(self->right, self, x);
        // encontrado!
        else if (self->left != nullptr && self->right != nullptr) { // dos hijos
            self->data = min(self->right)->data;
            remove(self->right, self, self->data); // eliminamos el sucesor duplicado
        } else { // el nodo tiene a lo mas 1 hijo
            BinaryNode *old = self; // guardamos la direccion del nodo a ser borrado
            self = (self->left != nullptr) ? self->left: self->right;
            if (self != nullptr) self->parent = parent;
            delete old;
        }
    }

    void destroy(BinaryNode *&root) {
        if (root != nullptr) {
            destroy(root->left);
            destroy(root->right);
            delete root;
        }
        root = nullptr;
    }

    bool contains(const BinaryNode *self, const T x) {
        if (self == nullptr) return false;
        if (self->data == x) return true;
        if (x < self->data) return contains(self->left, x);
        else return contains(self->right, x);
    }

public:

    bool contains(const T toFind) {
        return contains(root, toFind);
    }

    inline void remove(T x) {
        remove(root, nullptr, x);
    }

    inline ~bst() {
        destroy(root);
    }

    inline bst() { root = nullptr; }

    bst(vector<T> v) {
        root = nullptr;
        for (T x: v)
            root = add(root, x);
    }

    int numberOfNodes() {
        return numberOfNodes(root);
    }

    inline void add(T data) {
        root = add(root, data); // actualizamos la nueva raiz
    }

    inline void inorder() { inorder(root); }

    void sort(vector<T> &v) {
        for (T x: v) add(x);

        // recorrido en inorden
        stack<BinaryNode *> s;
        BinaryNode *curr = root;
        T data;
        int i = 0;
        while (true) {
            if (curr != nullptr) {
                s.push(curr);
                curr = curr->left;
            } else {
                if (s.empty()) break; // el nodo actual es nulo y la pila esta vacia
                curr = s.top();
                s.pop();
                data = curr->data;
                v[i++] = data;
                curr = curr->right;
            }
        }
    }
};

#endif
```

3. Actividades y pruebas

4. Anexo

5. Bibliografia