Sprawozdanie z Projektu nr 2

Spis treści

Opis ***2***

Wyniki programów z czasem ***3***

Obserwacje: ***3***

Kod ***4***

Main.cpp **4**

bst.h **5**

bst.cpp **6**

linkedList.h **9**

linkedList.cpp **10**

hashTable.h **12**

hashTable.cpp **13**

Opis

Po ostatnim projekcie spodobała mi się idea programowania obiektowego w takich projektach. Zacząłem więc od ustawienia sobie wszystkich potrzebnych plików i katalogów

- Drzewo BST jest zaprogramowane w katalogu ***/BST*** (nagłówek jak i .cpp)
- Lista dozwiązaniowa jest zaprogramowane w katalogu ***/LinkedList*** (nagłówek jak i .cpp)
- Tablica z hasowaniem jest zaprogramowane w katalogu ***/HashTable*** (nagłówek jak i .cpp)

Na wstępie jeszcze powiem tyle że program kompilowałem własnoręcznie w prostym IDE za pomocą MAKEFILE i konsoli typu bash.

Cały projekt zacząłem od struktury LinkedList – Listy z dowiązaniem. Przypomniałem sobie kształt całej struktury z zajęć. Przekształciłem na obiekt typu klasa, pododawałem metody do wstawienia danych i wypisywania na ekran. Myślę, że ta struktura była najłatwiejsza ze wszystkich.

Drugą w kolei zacząłem robić drzewo BST. Tutaj problemy się zaczęły. Niestety w takiego typu strukturach ważną częścią jest rekurencja, która niestety u mnie jest na poziomie podłogi…

Najwięcej czasu poświęciłem na zrozumienie jak działa jakakolwiek rekurencja a dopiero potem jak wygląda algorytm drzewa w programowaniu. Nad drzewem spędziłem większą część całego projektu.

Na koniec tablica haszująca. Tu nie było nic trudnego. Zwykła tablica o typie Listy z poprzedniego podpunktu. Mając cały algorytm Listy tutaj jedynie musiałem go użyć. Jedynym problemem było wymyślenie w jaki sposób haszować liczby w taki sposób, aby ich index był w przedziale 0 – n i zawsze wskazywał na tą samą liczbę, np. przekazujemy liczbę 980723412 i jej indexem ma być załóżmy liczba 123 która jest indexem tej liczby w tablicy. Więc za każdym razem, gdy wywołujemy tą funkcję dla tej samej liczby ma zwracać tą samą wartość. W moim przypadku rozwiązanie było proste jednak trochę nad tym siedziałem. Funkcja haszująca zwracała resztę z dzielenia liczby przekazywanej z argumentu przez n (ilość elementów w tablicy) Tym sposobem byłem pewny, że index nie będzie ani ujemny ani > n. Nic prostszego.

Aby program miał działać dla różnego n wystarczy w pliku main.cpp zmienić #define n na jakąś inną wartość. Poniżej prezentuję czasy dla odpowiednio n = 10 000, 20 000, 30 000, 100 000.

N = 10 000

Wyniki programów z czasem

N = 10 000

```
Wprowadzanie danych do drzewa BST...
Gotowe! Czas: 0.00279333s
Wprowadzanie danych do listy...
Gotowe! Czas: 0.342374s
Wprowadzanie danych do tablicy z hashowaniem...
Gotowe! Czas: 0.000644417s
Wyszukiwanie danych w drzewie BST...
Gotowe! Czas: 0.00251929s
Wyszukiwanie danych w liscie...
Gotowe! Czas: 0.64285s
Wyszukiwanie danych w tablicy z hashowaniem...
Gotowe! Czas: 0.000388083s
```

N = 20 000

```
Wprowadzanie danych do drzewa BST...
Gotowe! Czas: 0.00586517s
Wprowadzanie danych do listy...
Gotowe! Czas: 1.29826s
Wprowadzanie danych do tablicy z hashowaniem...
Gotowe! Czas: 0.0012015s
Wyszukiwanie danych w drzewie BST...
Gotowe! Czas: 0.00548604s
Wyszukiwanie danych w liscie...
Gotowe! Czas: 2.56211s
Wyszukiwanie danych w tablicy z hashowaniem...
Gotowe! Czas: 0.000823375s
```

N = 30 000

```
Wprowadzanie danych do drzewa BST...
Gotowe! Czas: 0.00971967s
Wprowadzanie danych do listy...
Gotowe! Czas: 2.86745s
Wprowadzanie danych do tablicy z hashowaniem...
Gotowe! Czas: 0.00184233s
Wyszukiwanie danych w drzewie BST...
Gotowe! Czas: 0.010263s
Wyszukiwanie danych w liscie...
Gotowe! Czas: 5.7231s
Wyszukiwanie danych w tablicy z hashowaniem...
Gotowe! Czas: 0.00125396s
```

N = 100 000

```
Wprowadzanie danych do drzewa BST...
Gotowe! Czas: 0.0367459s
Wprowadzanie danych do listy...
Gotowe! Czas: 31.947s
Wprowadzanie danych do tablicy z hashowaniem...
Gotowe! Czas: 0.00668788s
Wyszukiwanie danych w drzewie BST...
Gotowe! Czas: 0.0387339s
Wyszukiwanie danych w liscie...
Gotowe! Czas: 64.3099s
Wyszukiwanie danych w tablicy z hashowaniem...
Gotowe! Czas: 0.00537821s
```

Obserwacje:

Najwięcej czasu na wprowadzenie danych jak i wyszukanie zajmuję liście, nic dziwnego. W liście, aby dodać element musimy przejść po kolei po niej. Gdy lista jest pusta dodajemy na początek, Potem musimy przejść do 2 elementu itd. Aby dodać 100-tny element musimy przejść przez 99 elementów, a ta lista za każdym razem się zwiększa. Jednak to był pikuś w porównaniu to wyszukiwania w liście elementów. Aby wyszukać dany element musimy przejść w najgorszym wypadku od początku listy do jej końca.
Drugie miejsce zajmuję drzewo BST, tutaj wprowadzanie danych trwa krócej, gdyż sprawdzamy tylko gałąź, która nam pasuję.
Najszybciej działa tablica z haszowaniem. Oczywista oczywistość, wprowadzając liczbę do tablicy od razu znamy jej położenie, liczymy sobie jej index funkcją, która wykonuje proste obliczenie, po czym od razu w to miejsce ją dodajemy na koniec listy. Jeśli w danym indexie jest już jakaś wartość idziemy niżej, jeśli jest pusto zapisujemy na „wierzchu”. Wszystko prosto, łatwo i szybko!

Tak jak w poprzednim projekcie zachęcam do githuba: https://github.com/matelko123/ALGO-Projekt2

Kod

Main.cpp

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <vector>
#include <algorithm>

#include "./include/BST/bst.h"
#include "./include/LinkedList/linkedList.h"
#include "./include/HashTable/hashTable.h"

#define mini 1000000000
#define maxi 9999999999
#define ull unsigned long long
#define N 30000

using namespace std;

void generateNumbers(vector<ull> &tab, int amount)
{
    tab.clear();
    while (tab.size() < N)
    {
        ull num = rand() % (maxi - mini) + mini;

        if (find(tab.begin(), tab.end(), num) == tab.end())
            tab.push_back(num);
    }
}

int main()
{
    srand(time(0));
    vector<ull> tab;
    generateNumbers(tab, N);

    BST root;
    LinkedList head;
    HashTable ht(N);

    root.set(tab);
    head.set(tab);
    ht.set(tab);

    generateNumbers(tab, N);
    root.find(tab);
    head.find(tab);
    ht.find(tab);

    return 0;
}
```

bst.h

```
#ifndef __BST_H__
#define __BST_H__
#define ull unsigned long long

#include <iostream>
#include <vector>
#include <chrono>

using namespace std;

struct node
{
    ull data;
    node *left;
    node *right;
};

class BST
{
private:
    node *root;
    void insert(ull, node *);
    void destroy_tree(node *);
    void inorder(node *);

    node *search(ull, node *);

public:
    BST();
    ~BST();

    void destroy_tree();

    void insert(ull);
    void set(vector<ull> &);

    void inorder();

    node *search(ull);

    bool find(ull);
    void find(vector<ull> &);
};

#endif // __BST_H__
```

bst.cpp

```cpp
#include "bst.h"

BST::BST()
{
    root = nullptr;
}

BST::~BST()
{
    destroy_tree();
}

void BST::destroy_tree()
{
    destroy_tree(root);
}
void BST::destroy_tree(node *leaf)
{
    if (leaf != nullptr)
    {
        destroy_tree(leaf->left);
        destroy_tree(leaf->right);
        delete leaf;
    }
}

void BST::insert(ull key)
{
    if (root != nullptr)
        insert(key, root);
    else
    {
        root = new node;
        root->data = key;
        root->left = nullptr;
        root->right = nullptr;
    }
}
void BST::insert(ull key, node *leaf)
{
    if (key < leaf->data)
    {
        if (leaf->left != nullptr)
            insert(key, leaf->left);
        else
        {
            leaf->left = new node;
            leaf->left->data = key;
            leaf->left->left = nullptr;
            leaf->left->right = nullptr;
        }
    }
    else
    {
```

```
        if (leaf->right != nullptr)
            insert(key, leaf->right);
        else
        {
            leaf->right = new node;
            leaf->right->data = key;
            leaf->right->left = nullptr;
            leaf->right->right = nullptr;
        }
    }
}

node *BST::search(ull key)
{
    return search(key, root);
}
node *BST::search(ull key, node *leaf)
{
    if (leaf != nullptr)
    {
        if (key == leaf->data)
            return leaf;
        if (key < leaf->data)
            return search(key, leaf->left);
        else
            return search(key, leaf->right);
    }
    else
        return NULL;
}

void BST::inorder()
{
    if(root != nullptr)
        inorder(root);
    cout<<endl;
}
void BST::inorder(node *leaf)
{
    if(leaf == NULL)
        return;

    inorder(leaf->left);
    cout<<leaf->data<<"\t";
    inorder(leaf->right);
}

bool BST::find(ull key)
{
    if(search(key) != nullptr) return true;
    return false;
}
void BST::find(vector<ull> &tab)
{
    cout << "Wyszukiwanie danych w drzewie BST..." << endl;
```

```
    auto start = chrono::steady_clock::now();
    for (int i = 0; i < tab.size(); i++)
        this->find(tab[i]);
    auto end = chrono::steady_clock::now();
    chrono::duration<double> elapsed_seconds = end-start;
    cout << "Gotowe! Czas: " << elapsed_seconds.count() << "s\n";
}

void BST::set(vector<ull> &tab)
{
    cout << "Wprowadzanie danych do drzewa BST..." << endl;
    auto start = chrono::steady_clock::now();
    for (int i = 0; i < tab.size(); i++)
        this->insert(tab[i]);
    auto end = chrono::steady_clock::now();
    chrono::duration<double> elapsed_seconds = end-start;
    cout << "Gotowe! Czas: " << elapsed_seconds.count() << "s\n";
}
```

linkedList.h

```
#ifndef __NODE_H__
#define __NODE_H__
#define ull unsigned long long
#include <iostream>
#include <vector>
#include <chrono>

using namespace std;

struct Node
{
    ull value;
    Node *next;
};

class LinkedList
{
protected:
    Node *root;

public:
    LinkedList();
    ~LinkedList();

    void insert(ull);
    void set(vector<ull> &);
    void print();

    bool find(ull);
    void find(vector<ull> &);

    ull &operator[](int);
};

#endif // __NODE_H__
```

linkedList.cpp

```cpp
#include "linkedList.h"
#include <iostream>

using namespace std;

LinkedList::LinkedList()
{
    root = NULL;
}

LinkedList::~LinkedList()
{
    delete root;
}

void LinkedList::insert(ull key)
{
    Node *tmp = new Node;
    tmp->value = key;
    tmp->next = nullptr;

    if (root == NULL)
    {
        root = tmp;
        return;
    }

    Node *t = root;
    while (t->next != nullptr)
        t = t->next;

    t->next = tmp;
}

void LinkedList::print()
{
    Node *t = root;
    while (t != nullptr)
    {
        cout << t->value << " -> ";
        t = t->next;
    }
    cout << "NULL" << endl;
}

bool LinkedList::find(ull key)
{
    Node *t = root;
    while (t != nullptr)
    {
        if (t->value == key)
            return true;
        t = t->next;
    }
```

```
    return false;
}

ull &LinkedList::operator[](int i)
{
    Node *t = root;
    int j=0;
    while (t != nullptr && j < i)
        t = t->next;

    if(i == j) return t->value;

    return root->value;
}

void LinkedList::find(vector<ull> &tab)
{
    cout << "Wyszukiwanie danych w liscie..." << endl;
    auto start = chrono::steady_clock::now();
    for (int i = 0; i < tab.size(); i++)
        this->find(tab[i]);
    auto end = chrono::steady_clock::now();
    chrono::duration<double> elapsed_seconds = end-start;
    cout << "Gotowe! Czas: " << elapsed_seconds.count() << "s\n";
}

void LinkedList::set(vector<ull> &tab)
{
    cout << "Wprowadzanie danych do listy..." << endl;
    auto start = chrono::steady_clock::now();
    for (int i = 0; i < tab.size(); i++)
        this->insert(tab[i]);
    auto end = chrono::steady_clock::now();
    chrono::duration<double> elapsed_seconds = end-start;
    cout << "Gotowe! Czas: " << elapsed_seconds.count() << "s\n";
}
```

hashTable.h

```
#ifndef __HASHTABLE_H__
#define __HASHTABLE_H__
#define ull unsigned long long
#include <iostream>
#include <sstream>
#include <cstdlib>
#include <vector>
#include <chrono>
#include "../LinkedList/linkedList.h"

using namespace std;

class HashTable
{
private:
    LinkedList *tab;
    unsigned int amount;

    ull hash(ull);

public:
    HashTable(int);
    // ~HashTable();

    void print();
    void insert(ull);
    void set(vector<ull>);
    bool find(ull);
    void find(vector<ull> &);

    LinkedList &operator[](int);
};

#endif // __HASHTABLE_H__
```

hashTable.cpp

```cpp
#include "hashTable.h"

HashTable::HashTable(int _amount): amount(_amount)
{
    tab = new LinkedList[amount];
}

void HashTable::print()
{
    for(int i = 0; i < amount; i++)
    {
        tab[i].print();
        cout<<"\t";
    }
    cout<<endl;
}

ull HashTable::hash(ull val)
{
    return val%amount;
}

void HashTable::insert(ull val)
{
    ull hash_val = hash(val);
    tab[hash_val].insert(val);
}

LinkedList &HashTable::operator[](int i)
{
    if(i > amount)
        return tab[0];

    return tab[i];
}

bool HashTable::find(ull key)
{
    ull hash_val = hash(key);
    return tab[hash_val].find(key);
}

void HashTable::find(vector<ull> &tab)
{
    cout << "Wyszukiwanie danych w tablicy z hashowaniem..." << endl;
    auto start = chrono::steady_clock::now();
    for (int i = 0; i < tab.size(); i++)
        this->find(tab[i]);
    auto end = chrono::steady_clock::now();
    chrono::duration<double> elapsed_seconds = end-start;
    cout << "Gotowe! Czas: " << elapsed_seconds.count() << "s\n";
}

void HashTable::set(vector<ull> tab)
```

```
{
    cout << "Wprowadzanie danych do tablicy z hashowaniem..." << endl;
    auto start = chrono::steady_clock::now();
    for (int i = 0; i < tab.size(); i++)
        this->insert(tab[i]);
    auto end = chrono::steady_clock::now();
    chrono::duration<double> elapsed_seconds = end-start;
    cout << "Gotowe! Czas: " << elapsed_seconds.count() << "s\n";
}
```