|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
|  | | |
|  | | |
|  | | |
| Практическое задание № 4 | | |
| по дисциплине «Методы построения и анализа алгоритмов» | | |
|  | | |
| **кафедра теоретической и прикладной информатики** | | |
|  | | |
|  | Факультет: | ПМИ |
| Группа: | ПМИ-03 |
| Бригада: | Место для ввода текста. |
| Студенты: | Сидоров Даниил, |
|  | Малыгин Сергей |
|  |  |
| Преподаватели: | Щукин Георгий Анатольевич |
|  |  |
|  | | |
| Новосибирск | | |
| 2021 | | |

**1.Результаты замеров:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | бинарное дерево | | Дерево по убыванию | | Неупорядоченное дерево | | Set | |
| Вставка | Поиск | Вставка | Поиск | Вставка | Поиск | Вставка | Поиск |
| 10 | 3.4e-06 sec | 4e-07 sec | 2.3e-06 sec | 3e-07 sec | 2.1e-06 sec | 4e-07 sec | 9.4e-06 sec | 7e-07 sec |
| 100 | 2.39e-05 sec | 2.4e-06 sec | 2.15e-05 sec | 3.6e-06 sec | 2.17e-05 sec | 2.6e-06 sec | 2.56e-05 sec | 3.6e-06 sec |
| 1 000 | 0.0011764 sec | 2.16e-05 | 0.000943 sec | 2.19e-05 sec | 0.0003956 sec | 2.18e-05 sec | 0.0001221 sec | 1.92e-05 sec |
| 10 000 | 0.0997506 sec | 0.0002411 sec | 0.105182 sec | 0.0003133 sec | 0.0532739 sec | 0.0002141 sec | 0.0012367 sec | 0.0002144 sec |
| 100 000 | 13.6336 sec | 0.0032566 sec | 13.6872 sec | 0.0021283 sec | 8.41472 sec | 0.0039702 sec | 0.0132251 sec | 0.0021773 sec |

**2.Программа:**

**Main.cpp**

#include "binary\_tree.h"

#include<iostream>

#include <chrono>

#include <ctime>

#include <set>

#define CATCH\_CONFIG\_RUNNER

#include "catch.hpp"

#pragma comment(linker, "/STACK:16777216")

using namespace std;

int main(int argc, char \*\*argv)

{

int result = Catch::Session().run(argc, argv);

setlocale(LC\_ALL, "rus");

srand(time(0));

set <int> s;

binary\_tree tree;

cout << "Классическое бинарное дерево" << endl<<endl;

for (int i = 15; i <= 150000; i = i \* 10)

{

tree.destroy();

cout << "N = " << i<<endl;

auto t1 = std::chrono::high\_resolution\_clock::now();

for (int j = 0; j < i; j++)

{

tree.insert(j);

}

auto t2 = std::chrono::high\_resolution\_clock::now();

auto seconds = std::chrono::duration<double>(t2 - t1).count();

cout << "Вставка " << ": " << "Time: " << seconds << " sec." << endl;

t1 = std::chrono::high\_resolution\_clock::now();

for (int j = 0; j < i; j++)

{

tree.find(rand()%i);

}

t2 = std::chrono::high\_resolution\_clock::now();

seconds = std::chrono::duration<double>(t2 - t1).count();

cout << "Поиск " << ": " << "Time: " << seconds << " sec." << endl<<endl;

}

cout << "Дерево по убыванию" << endl << endl;

for (int i = 10; i <= 100000; i = i \* 10)

{

tree.destroy();

cout << "N = " << i << endl;

auto t1 = std::chrono::high\_resolution\_clock::now();

for (int j = 0; j < i; j++)

{

tree.insert\_inverse(j);

}

auto t2 = std::chrono::high\_resolution\_clock::now();

auto seconds = std::chrono::duration<double>(t2 - t1).count();

cout << "Вставка " << ": " << "Time: " << seconds << " sec." << endl;

t1 = std::chrono::high\_resolution\_clock::now();

for (int j = 0; j < i; j++)

{

tree.find\_inverse(rand() % i);

}

t2 = std::chrono::high\_resolution\_clock::now();

seconds = std::chrono::duration<double>(t2 - t1).count();

cout << "Поиск " << ": " << "Time: " << seconds << " sec." << endl << endl;

}

cout << "Неупорядоченное дерево " << endl << endl;

for (int i = 15; i <= 150000; i = i \* 10)

{

tree.destroy();

cout << "N = " << i << endl;

auto t1 = std::chrono::high\_resolution\_clock::now();

for (int j = 0; j < i; j++)

{

tree.insert\_nosort(j);

}

auto t2 = std::chrono::high\_resolution\_clock::now();

auto seconds = std::chrono::duration<double>(t2 - t1).count();

cout << "Вставка " << ": " << "Time: " << seconds << " sec." << endl;

t1 = std::chrono::high\_resolution\_clock::now();

for (int j = 0; j < i; j++)

{

tree.find\_nosort(rand() % i);

}

t2 = std::chrono::high\_resolution\_clock::now();

seconds = std::chrono::duration<double>(t2 - t1).count();

cout << "Поиск " << ": " << "Time: " << seconds << " sec." << endl << endl;

}

cout << "Set" << endl << endl;

for (int i = 15; i <= 150000; i = i \* 10)

{

cout << "N = " << i << endl;

auto t1 = std::chrono::high\_resolution\_clock::now();

for (int j = 0; j < i; j++)

{

s.insert(j);

}

auto t2 = std::chrono::high\_resolution\_clock::now();

auto seconds = std::chrono::duration<double>(t2 - t1).count();

cout << "Вставка " << ": " << "Time: " << seconds << " sec." << endl;

t1 = std::chrono::high\_resolution\_clock::now();

for (int j = 0; j < i; j++)

{

s.find(rand() % i);

}

t2 = std::chrono::high\_resolution\_clock::now();

seconds = std::chrono::duration<double>(t2 - t1).count();

cout << "Поиск " << ": " << "Time: " << seconds << " sec." << endl << endl;

s.clear();

}

}

**Binary\_tree.h**

#pragma once

#include <iostream>

class binary\_tree

{

private:

struct tree\_elem

{

int m\_data;

tree\_elem \*m\_left;

tree\_elem \*m\_right;

tree\_elem(int val)

{

m\_left = nullptr;

m\_right = nullptr;

m\_data = val;

}

};

tree\_elem \*m\_root =nullptr;

int m\_size = 0;

bool flag = false;

void print\_tree(tree\_elem \* curr)

{

if (curr)

{

print\_tree(curr->m\_left);

std::cout << curr->m\_data << " ";

print\_tree(curr->m\_right);

}

}

void delete\_tree(tree\_elem \* curr)

{

if (curr)

{

delete\_tree(curr->m\_left);

delete\_tree(curr->m\_right);

delete curr;

}

}

bool fnsort(tree\_elem \*curr,int key)

{

if (curr && curr->m\_data != key)

{

fnsort(curr->m\_left, key);

fnsort(curr->m\_right, key);

}

return curr != NULL;

}

public:

binary\_tree()

{

}

~binary\_tree()

{

delete\_tree(m\_root);

flag = false;

}

void destroy()

{

delete\_tree(m\_root);

flag = false;

}

void print()

{

print\_tree(m\_root);

std::cout << std::endl;

}

void find\_nosort(int key)

{

fnsort(m\_root,key);

}

bool find(int key)

{

tree\_elem \*curr = m\_root;

while (curr && curr->m\_data != key)

{

if (curr->m\_data > key)

curr = curr->m\_left;

else

curr = curr->m\_right;

}

return curr != NULL;

}

bool find\_inverse(int key)

{

tree\_elem \*curr = m\_root;

while (curr && curr->m\_data != key)

{

if (curr->m\_data < key)

curr = curr->m\_left;

else

curr = curr->m\_right;

}

return curr != NULL;

}

void insert(int key)

{

if (flag)

{

tree\_elem \*curr = m\_root;

while (curr && curr->m\_data != key)

{

if (curr->m\_data > key && curr->m\_left == NULL)

{

curr->m\_left = new tree\_elem(key);

++m\_size;

return;

}

if (curr->m\_data < key && curr->m\_right == NULL)

{

curr->m\_right = new tree\_elem(key);

++m\_size;

return;

}

if (curr->m\_data > key)

curr = curr->m\_left;

else

curr = curr->m\_right;

}

}else

{

m\_root = new tree\_elem(key);

m\_size = 1;

flag = true;

}

}

void insert\_inverse(int key)

{

if (flag)

{

tree\_elem \*curr = m\_root;

while (curr && curr->m\_data != key)

{

if (curr->m\_data < key && curr->m\_left == NULL)

{

curr->m\_left = new tree\_elem(key);

++m\_size;

return;

}

if (curr->m\_data > key && curr->m\_right == NULL)

{

curr->m\_right = new tree\_elem(key);

++m\_size;

return;

}

if (curr->m\_data < key)

curr = curr->m\_left;

else

curr = curr->m\_right;

}

}

else

{

m\_root = new tree\_elem(key);

m\_size = 1;

flag = true;

}

}

void insert\_nosort(int key)

{

if (flag)

{

tree\_elem \*curr = m\_root;

while (curr && curr->m\_data != key)

{

if (curr->m\_left == NULL)

{

curr->m\_left = new tree\_elem(key);

++m\_size;

return;

}

if (curr->m\_right == NULL)

{

curr->m\_right = new tree\_elem(key);

++m\_size;

return;

}

if (curr->m\_left != NULL)

curr = curr->m\_left;

else if (curr->m\_right != NULL)

curr = curr->m\_right;

}

}

else

{

m\_root = new tree\_elem(key);

m\_size = 1;

flag = true;

}

}

int size()

{

return m\_size;

}

};