Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий

Кафедра вычислительной техники

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №6**

По дисциплине “Алгоритмы и структуры данных”

Вариант 21

|  |  |  |
| --- | --- | --- |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_ | Матковский И.В. |
|  | подпись, дата |  |
| Студент КИ20-07Б, 032052463 | \_\_\_\_\_\_\_\_\_\_ | Базаров А.С. |
|  | подпись, дата |  |
|  |  |  |
|  |  |  |

Красноярск 2021

**СОДЕРЖАНИЕ**

[1 Задание 4](#_Toc81912682)

[2 Исходный код 4](#_Toc81912683)

[3 Результаты 5](#_Toc81912684)

1. Задание

|  |  |
| --- | --- |
| № | Задание |
| 21 | Удалить все узлы со значениями в промежутке от A до B (A и B задаются пользователем). |

1. Исходный код

Программа с двоичным деревом поиска (к программе прикреплен файл):

// Операции с двоичным деревом поиска на C++

#include <iostream>

#include <fstream>

using namespace std;

struct node {

int key;

struct node\* left, \* right;

};

// Создаем узел

struct node\* newNode(int item) {

struct node\* temp = (struct node\*)malloc(sizeof(struct node));

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

// Вставка узла

struct node\* insert(struct node\* node, int key) {

// Возвращаем новый узел, если дерево пустое

if (node == NULL) return newNode(key);

// Проходим в нужное место и вставляет узел

if (key < node->key)

node->left = insert(node->left, key);

else

node->right = insert(node->right, key);

return node;

}

// Поис inorder-преемника

struct node\* minValueNode(struct node\* node) {

struct node\* current = node;

// Ищем крайний левый лист — он и будет inorder-преемником

while (current && current->left != NULL)

current = current->left;

return current;

}

// Удаление узла

struct node\* deleteNode(struct node\* root, int key) {

// Возвращаем, если дерево пустое

if (root == NULL) return root;

// Ищем узел, который хотим удалить

if (key < root->key)

root->left = deleteNode(root->left, key);

else if (key > root->key)

root->right = deleteNode(root->right, key);

else {

// Если у узла один дочерний элемент или их нет

if (root->left == NULL) {

struct node\* temp = root->right;

free(root);

return temp;

}

else if (root->right == NULL) {

struct node\* temp = root->left;

free(root);

return temp;

}

// Если у узла два дочерних элемента

struct node\* temp = minValueNode(root->right);

// Помещаем inorder-преемника на место узла, который хотим удалить

root->key = temp->key;

// Удаляем inorder-преемника

root->right = deleteNode(root->right, temp->key);

}

return root;

}

void print\_Tree(node\* p, int level)

{

if (p)

{

print\_Tree(p->left, level + 1);

for (int i = 0; i < level; i++) cout << " ";

cout << p->key << endl;

print\_Tree(p->right, level + 1);

}

};

void A\_to\_B(struct node\* p, int A, int B, int level) {

if (p)

{

A\_to\_B(p->left, A, B, level + 1);

for (int i = 0; i < level; i++) cout << " ";

if (p->key <= B && p->key >= A)

cout << p->key << endl;

else

cout << "@" << endl;

A\_to\_B(p->right, A, B, level + 1);

}

};

int Search(struct node\* p, int count, int A, int B) {

if (p) {

if (p->key <= B && p->key >= A)

count++;

count = Search(p->left, count, A, B);

count = Search(p->right, count, A, B);

}

return count;

};

void Write(struct node\* p, fstream& FilE, int A, int B) {

if (p) {

if (p->key <= B && p->key >= A) {

FilE << p->key << " ";

}

Write(p->left, FilE, A, B);

Write(p->right, FilE, A, B);

}

};

// Тестим функции

int main() {

setlocale(LC\_ALL, "rus");

struct node\* root = NULL;

int A, B, count;

cout << "Сколько элементов дерева вы хотите ввести?";

cin >> count;

cout << "Введите корень!";

cin >> A;

root = insert(root, A);

cout << "Введите " << count - 1 << " узлов дерева!";;

for (int i = 1; i < count; i++) {

cin >> A;

root = insert(root, A);

}

cout << "Обычное дерево" << endl;

print\_Tree(root, 0);

cout << "Теперь введите промежуток значений от A до B" << endl;

cout << "A = ";

cin >> A;

cout << "B = ";

cin >> B;

fstream FilE;

FilE.open("Tree.txt", ios::out);

FilE << "";

FilE.close();

int i = 0;

i = Search(root, i, A, B);

if (i == 0) {

cout << "Узлов для удаления не найдено так как узлов, попадающих" << endl;

cout << "в заданный диапазон, нет! Дерево не изменилось!";

exit(0);

};

cout << "Всего найдено " << i << " узлов." << endl;

cout << "Найденные узлы" << endl;

A\_to\_B(root, A, B, 0);

FilE.open("Tree.txt");

Write(root, FilE, A, B);

FilE.seekg(0);

FilE.close();

FilE.open("Tree.txt");

cout << endl;

cout << "Удаляем их" << endl;

cout << endl;

int aux;

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

{

FilE >> aux;

deleteNode(root, aux);

}

FilE.close();

cout << "Измененное дерево" << endl;

print\_Tree(root, 0);

}

Программа с двоичным деревом поиска (к программе прикреплен файл):

// Операции с двоичным деревом поиска на C++

#include <iostream>

#include <fstream>

using namespace std;

struct node {

int key;

struct node\* left, \* right;

};

// Создаем узел

struct node\* newNode(int item) {

struct node\* temp = (struct node\*)malloc(sizeof(struct node));

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

// Вставка узла

struct node\* insert(struct node\* node, int key) {

// Возвращаем новый узел, если дерево пустое

if (node == NULL) return newNode(key);

// Проходим в нужное место и вставляет узел

if (key < node->key)

node->left = insert(node->left, key);

else

node->right = insert(node->right, key);

return node;

}

// Поис inorder-преемника

struct node\* minValueNode(struct node\* node) {

struct node\* current = node;

// Ищем крайний левый лист — он и будет inorder-преемником

while (current && current->left != NULL)

current = current->left;

return current;

}

// Удаление узла

struct node\* deleteNode(struct node\* root, int key) {

// Возвращаем, если дерево пустое

if (root == NULL) return root;

// Ищем узел, который хотим удалить

if (key < root->key)

root->left = deleteNode(root->left, key);

else if (key > root->key)

root->right = deleteNode(root->right, key);

else {

// Если у узла один дочерний элемент или их нет

if (root->left == NULL) {

struct node\* temp = root->right;

free(root);

return temp;

}

else if (root->right == NULL) {

struct node\* temp = root->left;

free(root);

return temp;

}

// Если у узла два дочерних элемента

struct node\* temp = minValueNode(root->right);

// Помещаем inorder-преемника на место узла, который хотим удалить

root->key = temp->key;

// Удаляем inorder-преемника

root->right = deleteNode(root->right, temp->key);

}

return root;

}

void print\_Tree(node\* p, int level)

{

if (p)

{

print\_Tree(p->left, level + 1);

for (int i = 0; i < level; i++) cout << " ";

cout << p->key << endl;

print\_Tree(p->right, level + 1);

}

};

void A\_to\_B(struct node\* p, int A, int B, int level) {

if (p)

{

A\_to\_B(p->left, A, B, level + 1);

for (int i = 0; i < level; i++) cout << " ";

if (p->key <= B && p->key >= A)

cout << p->key << endl;

else

cout << "@" << endl;

A\_to\_B(p->right, A, B, level + 1);

}

};

int Search(struct node\* p, int count, int A, int B) {

if (p) {

if (p->key <= B && p->key >= A)

count++;

count = Search(p->left, count, A, B);

count = Search(p->right, count, A, B);

}

return count;

};

void Write(struct node\* p, fstream& FilE, int A, int B) {

if (p) {

if (p->key <= B && p->key >= A) {

FilE << p->key << " ";

}

Write(p->left, FilE, A, B);

Write(p->right, FilE, A, B);

}

};

// Тестим функции

int main() {

setlocale(LC\_ALL, "rus");

struct node\* root = NULL;

int A, B, count;

cout << "Сколько элементов дерева вы хотите ввести?";

cin >> count;

cout << "Введите корень!";

cin >> A;

root = insert(root, A);

cout << "Введите " << count - 1 << " узлов дерева!";;

for (int i = 1; i < count; i++) {

cin >> A;

root = insert(root, A);

}

cout << "Обычное дерево" << endl;

print\_Tree(root, 0);

cout << "Теперь введите промежуток значений от A до B" << endl;

cout << "A = ";

cin >> A;

cout << "B = ";

cin >> B;

fstream FilE;

FilE.open("Tree.txt", ios::out);

FilE << "";

FilE.close();

int i = 0;

i = Search(root, i, A, B);

if (i == 0) {

cout << "Узлов для удаления не найдено так как узлов, попадающих" << endl;

cout << "в заданный диапазон, нет! Дерево не изменилось!";

exit(0);

};

cout << "Всего найдено " << i << " узлов." << endl;

cout << "Найденные узлы" << endl;

A\_to\_B(root, A, B, 0);

FilE.open("Tree.txt");

Write(root, FilE, A, B);

FilE.seekg(0);

FilE.close();

FilE.open("Tree.txt");

cout << endl;

cout << "Удаляем их" << endl;

cout << endl;

int aux;

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

{

FilE >> aux;

deleteNode(root, aux);

}

FilE.close();

cout << "Измененное дерево" << endl;

print\_Tree(root, 0);

}

Программа с АВЛ-деревом (к программе прикреплен файл):

#include<algorithm>

#include<iomanip>

#include<iostream>

#include <fstream>

using namespace std;

struct Node

{

int key;

char height;

Node\* right;

Node\* left;

Node(int k) { key = k; height = 1; left = right = 0; }

};

char height(Node\* p)

{

if (p) return p->height;

else return 0;

}

int BF(Node\* p)

{

return height(p->right) - height(p->left);

}

void OverHeight(Node\* p)

{

char hleft = height(p->left);

char hright = height(p->right);

p->height = (hleft > hright ? hleft : hright) + 1;

}

Node\* RightRotation(Node\* x)

{

Node\* y = x->left;

x->left = y->right;

y->right = x;

OverHeight(x);

OverHeight(y);

return y;

}

Node\* LeftRotation(Node\* y)

{

Node\* x = y->right;

y->right = x->left;

x->left = y;

OverHeight(y);

OverHeight(x);

return x;

}

Node\* Balance(Node\* x)

{

OverHeight(x);

if (BF(x) == 2)

{

if (BF(x->right) < 0) x->right = RightRotation(x->right);

return LeftRotation(x);

}

if (BF(x) == -2)

{

if (BF(x->left) > 0) x->left = LeftRotation(x->left);

return RightRotation(x);

}

return x;

}

struct Node\* Insert(Node\* x, int k)

{

if (!x) return new Node(k);

if (k < x->key) x->left = Insert(x->left, k);

else x->right = Insert(x->right, k);

return Balance(x);

}

Node\* SearchMin(Node\* x)

{

if (x->left) return SearchMin(x->left);

else return x;

}

Node\* DeleteMin(Node\* x)

{

if (x->left == 0) return x->right;

x->left = DeleteMin(x->left);

return Balance(x);

}

Node\* Delete(Node\* x, int k)

{

if (!x) return 0;

if (k < x->key) x->left = Delete(x->left, k);

else if (k > x->key) x->right = Delete(x->right, k);

else

{

Node\* y = x->left;

Node\* z = x->right;

delete x;

if (!z) return y;

Node\* min = SearchMin(z);

min->right = DeleteMin(z);

min->left = y;

return Balance(min);

}

return Balance(x);

}

struct Node\* FindKey(struct Node\* node, int k) {

if (node != NULL)

{

if (node->key = k) return node;

if (node->key > k) return FindKey(node->left, k);

else return FindKey(node->right, k);

}

return NULL;

}

void A\_to\_B(struct Node\* p, int A, int B, int level) {

if (p)

{

A\_to\_B(p->left, A, B, level + 1);

for (int i = 0; i < level; i++) cout << " ";

if (p->key <= B && p->key >= A)

cout << p->key << endl;

else

cout << "@" << endl;

A\_to\_B(p->right, A, B, level + 1);

}

};

void print\_Tree(Node\* p, int level)

{

if (p)

{

print\_Tree(p->left, level + 1);

for (int i = 0; i < level; i++) cout << " ";

cout << p->key << endl;

print\_Tree(p->right, level + 1);

}

};

int Search(Node\* p, int count, int A, int B) {

if (p) {

if (p->key <= B && p->key >= A)

count++;

count = Search(p->left, count, A, B);

count = Search(p->right, count, A, B);

}

return count;

};

void Write(Node\* p, fstream& FilE, int A, int B) {

if (p) {

if (p->key <= B && p->key >= A) {

FilE << p->key << " ";

}

Write(p->left, FilE, A, B);

Write(p->right, FilE, A, B);

}

};

int main()

{

setlocale(LC\_ALL, "Russian");

int A, B, count;

cout << "Сколько элементов дерева вы хотите ввести?";

cin >> count;

cout << "Введите корень!";

cin >> A;

struct Node\* root = NULL;

root = Insert(root, A);

cout << "Введите " << count - 1 << " узлов дерева!";;

for (int i = 1; i < count; i++) {

cin >> A;

root = Insert(root, A);

}

cout << "АВЛ-дерево" << endl;

print\_Tree(root, 0);

cout << endl;

cout << "Теперь введите промежуток значений от A до B" << endl;

cout << "A = ";

cin >> A;

cout << "B = ";

cin >> B;

fstream FilE;

FilE.open("AVLTree.txt", ios::out);

FilE << "";

FilE.close();

int i = 0;

i = Search(root, i, A, B);

if (i == 0) {

cout << "Узлов для удаления не найдено так как узлов, попадающих" << endl;

cout << "в заданный диапазон, нет! Дерево не изменилось!";

exit(0);

};

cout << "Всего найдено " << i << " узлов." << endl;

cout << "Найденные узлы" << endl;

A\_to\_B(root, A, B, 0);

FilE.open("AVLTree.txt");

Write(root, FilE, A, B);

FilE.seekg(0);

FilE.close();

FilE.open("AVLTree.txt");

cout << endl;

cout << "Удаляем их" << endl;

cout << endl;

int aux;

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

{

FilE >> aux;

root = Delete(root, aux);

}

FilE.close();

cout << "Измененное дерево" << endl;

print\_Tree(root, 0);

return 0;

}

1. Теоретические оценки сложности

Теоретическая сложность первой программы при использовании сортировки вставками составляет , (где n – массив чисел) так как мы создаем дерево, открываем и считываем в файл удаляемые узлы, а затем удаляем узлы дерева.

Теоретическая сложность второй программы при использовании сортировки вставками составляет , (где n – массив чисел) так как мы создаем дерево, открываем и считываем в файл удаляемые узлы, а затем удаляем узлы дерева.

1. Результаты

Результаты выполнения алгоритма приведены в таблице 1

Таблица 1

|  |  |  |
| --- | --- | --- |
| Количество элементов | Двоичное дерево поиска, время в мс. | АВЛ-дерево, время в мс. |
| 10 | 0.032 | 0.014 |
| 100 | 0.302 | 0.11 |
| 500 | 1.502 | 0.5073 |