МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №4

по дисциплине
«Структуры и алгоритмы обработки данных**»**

Тема: «Сбалансированные деревья поиска (СДП) и их применение для поиска данных в файле»

Выполнил студент группы ИКБО-20-21 Хитров Н.С.

Принял преподаватель Сорокин А.В.

Лабораторная работа выполнена «__»_____202__ г. *(подпись студента)*

«Зачтено» «__»_____202__ г. *(подпись руководителя)*

Москва 2022

1. Отчет по заданию 1

Разработать приложение, которое использует бинарное дерево поиска (БДП) для поиска записи с ключом в файле, структура которого представлена в задании 2 вашего варианта.

1. Разработать класс «Бинарное дерево поиска». Тип информационной части узла ключ и ссылка на запись в файле (как в практическом задании 2). Методы: включение элемента в дерево, поиск ключа в дереве, удаление ключа из дерева, отображение дерева.

2. Разработать класс управления файлом (если не создали в практическом задании 2). Включить методы: создание двоичного файла записей фиксированной длины из заранее подготовленных данных в текстовом файле; поиск записи в файле с использованием БДП; остальные методы по вашему усмотрению.

Часть 1 – код, реализующий несбалансированное бинарное дерево

```
struct BinTreeNode {
	long long int ISBN;
	int note_id;
	BinTreeNode* left = nullptr;
	BinTreeNode* right = nullptr;
	BinTreeNode(long long int code, int index) {
		this->ISBN = code;
		this->note_id = index;
	}
};
```

Листинг 1 - структура узла несбалансированного бинарного дерева

```
class BinTree {
private:
	BinTreeNode* root;
public:
	BinTree();
	BinTree(BinTreeNode* tree_root);
	BinTreeNode* getRoot() {
		return root;
	}
	void addNode(long long int code, int index, BinTreeNode** node);
	void addNode(long long int code, int index);
	BinTreeNode** seekNode(long long int code, BinTreeNode** node);
	BinTreeNode** seekNode(long long int code);
	void delNode(long long int code);
	void delNode(BinTreeNode** node);
	void printTree(BinTreeNode* node, int level);
	void fillFromFile(string filename, int n);
};
```

Листинг 2 - класс несбалансированного бинарного дерева

```
BinTree::BinTree() {
	this->root = nullptr;
}

BinTree::BinTree(BinTreeNode* tree_root) {
	this->root = tree_root;
}
```

Листинг 3 - конструкторы класса

```
void BinTree::addNode(long long int code, int index, BinTreeNode** node) {
	if (!*node) *node = new BinTreeNode(code, index);
	else if ((* node)->ISBN > code) addNode(code, index, &(*node)->left);
	else if ((*node)->ISBN < code) addNode(code, index, &(*node)->right);
}

void BinTree::addNode(long long int code, int index) {
	addNode(code, index, &root);
}
```

Листинг 4 - методы добавления узла в дерево

```
BinTreeNode** BinTree::seekNode(long long int code,BinTreeNode** node) {
	if (!*node) return nullptr;
	else if ((*node)->ISBN > code) seekNode(code, &(*node)->left);
	else if ((*node)->ISBN < code) seekNode(code, &(*node)->right);
	else return node;
}

BinTreeNode** BinTree::seekNode(long long int code) {
	return seekNode(code, &root);
}
```

Листинг 5 - методы поиска узла в дереве

```
void BinTree::delNode(long long int code) {
	BinTreeNode** node = seekNode(code, &root);
	if (node) {
		delNode(node);
	}
}

void BinTree::delNode(BinTreeNode** node) {
	if (!(*node)->left && !(*node)->right) {
		delete* node;
		*node = nullptr;
	}
	else if (!(*node)->right) {
		BinTreeNode* tmp = *node;
		*node = (*node)->left;
		delete tmp;
	}
	else if (!(*node)->left) {
		BinTreeNode* tmp = *node;
		*node = (*node)->right;
		delete tmp;
	}
}
```

Листинг 6 - методы удаления узла из дерева

```
void BinTree::printTree(BinTreeNode* node, int level)
{
	if (!node) return;
	printTree(node->right, level + 1);
	for (int i = 1; i <= level * 15; i++)
		cout << ' ';
	cout << node->ISBN << "\n";
	printTree(node->left, level + 1);
}

void BinTree::fillFromFile(string filename, int n) {
	ifstream f(filename, ios::binary);

	char* key_str = new char[14]{};
	for (int i = 0; i < n; i++) {
		long long int key = 0;
		f.seekg(i * 73, ios::beg);
		f.read(key_str, 13);
		for (int j = 0; j < 13; j++) {

			key += int(key_str[12 - j] - 48) * pow(10, j);
		}
		addNode(key, i);

	}
	f.seekg(0, ios::beg);
	f.close();
}
```

Листинг 7 - методы вывода и заполнения дерева из файла

Часть 2 – код реализующий управление бинарным файлом

```
class BinFile {
private:
	string name;
	int notes;
public:
	BinFile(string filename);
	int getNotesAm();
	string getFileName();
	void fillManually();
	void fillAuto();
	void seekNote(int x);
	void addNote(long long int key, string auth, string name);
	void printFile();
	~BinFile();
};
```

Листинг 8 - класс бинарного файла

```
BinFile::BinFile(string filename) {
	name = filename;
	notes = 0;
}

string BinFile::getFileName() {
	return name;
}

int BinFile::getNotesAm() {
	return notes;
}

BinFile::~BinFile() {
	ofstream file(name);
	file.clear();
	file.close();
}
```

Листинг 9 - конструктор класса, деструктор класса и геттеры

```
void BinFile::fillManually() {
	int n;
	cout << "Введите кол-во записей в файле: ";
	cin >> n;
	notes = n;
	char* key_stroke = new char[14]{};
	char* auth_stroke = new char[21]{};
	char* name_stroke = new char[41]{};
	ofstream file(name, ios_base::binary | ios_base::app);
	file.clear();
	for (int i = 0; i < n; i++) {
		cout << "Введите " << i + 1 << " запись" << endl;
		cout << "Введите ISBN: ";
		cin >> key_stroke;
		cout << "Введите автора: ";
		cin >> auth_stroke;
		cout << "Введите произведение: ";
		cin >> name_stroke;
		file.write(key_stroke, 13);
		file.write(auth_stroke, 20);
		file.write(name_stroke, 40);
	}
	file.close();
}

void BinFile::fillAuto() {
	int n;
	cout << "Введите кол-во записей в файле: ";
	cin >> n;
	notes = n;
	ofstream file(name, ios_base::binary | ios_base::app);
	file.clear();
	for (int i = 0; i < n; i++) {
		long long int key = rand() % 9 + 1;
		string auth_stroke = "", name_stroke = "";
		random_key(key);
		random_auth(auth_stroke);
		random_name(name_stroke);
		char* key_stroke = new char[14]{};
		for (int j = 0; j < 13; j++) {
			key_stroke[12 - j] = char(key % 10 + 48);
			key /= 10;
		}
		file.write(key_stroke, 13);
		file.write((char*)auth_stroke.c_str(), 20);
		file.write((char*)name_stroke.c_str(), 40);
	}
	file.close();
}
```

Листинг 10 - методы заполнения файла вручную и автоматически

```
void BinFile::seekNote(int x) {
	ifstream file(name, ios::binary);
	string note;
	file.seekg(x*73, ios::beg);
	char* key_stroke = new char[14]{};
	char* auth_stroke = new char[21]{};
	char* name_stroke = new char[41]{};
	file.read(key_stroke,13);
	file.read(auth_stroke, 20);
	file.read(name_stroke, 40);
	note = string(key_stroke) + string(auth_stroke) + string(name_stroke);
	file.seekg(0, ios::beg);
	file.close();
	cout << "ISBN - " << key_stroke << ": автор - " << auth_stroke << ", название - " << name_stroke << endl;
}
```

Листинг 11 - метод поиска записи в бинарном файле по координате символа

```
void BinFile::addNote(long long int key, string auth, string name) {
	ofstream file(this->name, ios_base::binary | ios_base::app);
	char* key_str = new char[14]{};
	char* auth_str = new char[21]{};
	auth_str = (char*)auth.c_str();
	char* name_str = new char[41]{};
	name_str = (char*)name.c_str();
	for (int j = 0; j < 13; j++) {
		key_str[12-j] = char(key % 10 + 48);
		key /= 10;
	}
	file.write(key_str, 13);
	file.write(auth_str, 20);
	file.write(name_str, 40);
	file.close();
	notes++;
}
```

Листинг 12 - метод добавления записи в бинарный файл

```
void BinFile::printFile() {
	ifstream file(name, ios::binary);
	for (int i = 0; i < notes; i++) {
		file.seekg(73*i,ios::beg);
		char* key_stroke = new char[14]{};
		char* auth_stroke = new char[21]{};
		char* name_stroke = new char[41]{};
		file.read(key_stroke, 13);
		file.read(auth_stroke, 20);
		file.read(name_stroke, 40);
		cout << "ISBN - " << key_stroke << ": " << auth_stroke << " " << name_stroke << endl;
	}
	file.seekg(0, ios::beg);
	file.close();
}
```

Листинг 13 - метод вывода записей файла

2. Отчет по заданию 1

Разработать приложение, которое использует сбалансированное дерево поиска, предложенное в варианте, для доступа к записям файла.

1. Разработать класс СДП с учетом дерева варианта. Структура информационной части узла дерева включает ключ и ссылку на запись в файле (адрес места размещения). Основные методы: включение элемента в дерево; поиск ключа в дереве с возвратом ссылки; удаление ключа из дерева; вывод дерева в форме дерева (с отображением структуры дерева).

2. Разработать приложение, которое создает и управляет СДП в соответствии с заданием.

3. Выполнить тестирование.

4. Определить среднее число выполненных поворотов (число поворотов на общее число вставленных ключей) при включении ключей в дерево при формировании дерева из двоичного файла.

Вариант №8

Сбалансированное дерево поиска (СДП)	Структура элемента множества (ключ – подчеркнутое поле) остальные поля представляют данные элемента
Косое дерево	Книга: ISBN – двенадцатизначное число, Автор, Название

```
struct TreeNode {
	int note_id;
	long long int ISBN;
	TreeNode* parent;
	TreeNode* left_child;
	TreeNode* right_child;

	TreeNode(long long int code, int index) {
		this->parent = nullptr;
		this->left_child = nullptr;
		this->right_child = nullptr;
		this->ISBN = code;
		this->note_id = index;
	}
};
```

Листинг 14 - структура узла сбалансированного бинарного дерева

```
class SplayTree {
public:
        TreeNode* root;
        int nodes;

        SplayTree();
        SplayTree(TreeNode* tree_root);
        void zig(TreeNode* x);
        void zig_zig(TreeNode* x);
        void zig_zag(TreeNode* x);
        void splay(TreeNode* x);
        TreeNode* search(long long int x);
        void insert(long long int x, int index);
        int remove(long long int x);
        void Clear(TreeNode* x);
        TreeNode* merge(SplayTree* s, SplayTree* t);
        void fillFromFile(string file, int n);
        void prettyPrint(TreeNode* node, int level);
        ~SplayTree();
};
```

Листинг 15 - класс косого дерева

```
SplayTree::SplayTree() {
   this->root = nullptr;
        nodes = 0;
}

SplayTree::SplayTree(TreeNode* tree_root) {
        this->root = tree_root;
        nodes = 1;
}

void SplayTree::Clear(TreeNode* x) {
   if (x != nullptr) {
      if (x->left_child != nullptr || x->right_child != nullptr) {
         if (x->left_child != nullptr && x->right_child == nullptr) {
            Clear(x->left_child);
            delete x;
         }
         else if (x->left_child == nullptr && x->right_child != nullptr) {
            Clear(x->right_child);
            delete x;
         }
         else {
            Clear(x->left_child);
         }
      }
      else {
         delete x;
      }
   }
}

SplayTree::~SplayTree() {
   TreeNode* curr = this->root;
   Clear(curr);
}
```

Листинг 16 - конструкторы и деструктор класса

```
void SplayTree::zig(TreeNode* x) {
        TreeNode* parent_node = x->parent;

   if (parent_node->left_child == x) {
                TreeNode* A = x->right_child;

      x->parent = nullptr;
      x->right_child = parent_node;

      parent_node->parent = x;
      parent_node->left_child = A;

      if (A != nullptr) {
         A->parent = parent_node;
      }
   }
   else {
                TreeNode* A = x->left_child;

      x->parent = nullptr;
      x->left_child = parent_node;

      parent_node->parent = x;
      parent_node->right_child = A;

      if (A != nullptr) {
         A->parent = parent_node;
      }
   }
}
```

Листинг 17 - балансировка, когда родитель узла корень поддерева

```
void SplayTree::zig_zig(TreeNode* x) {
            TreeNode* parent_node = x->parent;
            TreeNode* grandparent_node = parent_node->parent;

    if (parent_node->left_child == x) {
                        TreeNode* A = x->right_child;
                        TreeNode* B = parent_node->right_child;

        x->parent = grandparent_node->parent;
        x->right_child = parent_node;

        parent_node->parent = x;
        parent_node->left_child = A;
        parent_node->right_child = grandparent_node;

        grandparent_node->parent = parent_node;
        grandparent_node->left_child = B;

        if (x->parent != nullptr) {
            if (x->parent->left_child == grandparent_node) {
                x->parent->left_child = x;
            }
            else {
                x->parent->right_child = x;
            }
        }
        if (A != nullptr) {
            A->parent = parent_node;
        }
        if (B != nullptr) {
            B->parent = grandparent_node;
        }
    }
    else {
                        TreeNode* A = parent_node->left_child;
                        TreeNode* B = x->left_child;

        x->parent = grandparent_node->parent;
        x->left_child = parent_node;

        parent_node->parent = x;
        parent_node->left_child = grandparent_node;
        parent_node->right_child = B;
        grandparent_node->parent = parent_node;
        grandparent_node->right_child = A;

        if (x->parent != nullptr) {
            if (x->parent->left_child == grandparent_node) {
                x->parent->left_child = x;
            }
            else {
                x->parent->right_child = x;
            }
        }
        if (A != nullptr) {
            A->parent = grandparent_node;
        }
        if (B != nullptr) {
            B->parent = parent_node;
        }
    }
}
```

Листинг 18 - балансировка когда прародитель узла корень поддерева и родитель сонаправлен с узлом (оба левые сыновья родителей или правые)

```
void SplayTree::zig_zag(TreeNode* x) {
            TreeNode* parent_node = x->parent;
            TreeNode* grandparent_node = parent_node->parent;
   if (parent_node->right_child == x) {
                        TreeNode* A = x->left_child;
                        TreeNode* B = x->right_child;
      x->parent = grandparent_node->parent;
      x->left_child = parent_node;
      x->right_child = grandparent_node;

      parent_node->parent = x;
      parent_node->right_child = A;
      grandparent_node->parent = x;
      grandparent_node->left_child = B;
      if (x->parent != nullptr) {
         if (x->parent->left_child == grandparent_node) {
            x->parent->left_child = x;
         }
         else {
            x->parent->right_child = x;
         }
      }
      if (A != nullptr) {
         A->parent = parent_node;
      }
      if (B != nullptr) {
         B->parent = grandparent_node;
      }
   }
   else {
                        TreeNode* A = x->left_child;
                        TreeNode* B = x->right_child;

      x->parent = grandparent_node->parent;
      x->left_child = grandparent_node;
      x->right_child = parent_node;

      parent_node->parent = x;
      parent_node->left_child = B;
      grandparent_node->parent = x;
      grandparent_node->right_child = A;
      if (x->parent != nullptr) {
         if (x->parent->left_child == grandparent_node) {
            x->parent->left_child = x;
         }
         else {
            x->parent->right_child = x;
         }
      }
      if (A != nullptr) {
         A->parent = grandparent_node;
      }
      if (B != nullptr) {
         B->parent = parent_node;
      }
   }
}
```

Листинг 19 - Листинг 18 - балансировка когда прародитель узла корень поддерева и родитель противоположно направлен с узлом (родитель левый ребенок, искомый узел правый ребенок и наоборот)

```
void SplayTree::splay(TreeNode* x) {
```

```
    while (x->parent != nullptr) {
                        TreeNode* parent_node = x->parent;
                        TreeNode* grandparent_node = parent_node->parent;
        if (grandparent_node == nullptr) {
            zig(x);
        }
        else if (grandparent_node->left_child == parent_node && parent_node->left_child == x) {
            zig_zig(x);
        }
        else if (grandparent_node->right_child == parent_node && parent_node->right_child == x) {
            zig_zig(x);
        }
        else {
            zig_zag(x);
        }
    }
    this->root = x;
}
```

Листинг 20 - метод балансировки дерева

```
int SplayTree::remove(long long int x) {
            int num;
            TreeNode* del = search(x);
    if (del == nullptr) {
        return -1;
    }
            TreeNode* L = del->left_child;
    if (L == nullptr) {
        root = del->right_child;
        if (root != nullptr) {
            root->parent = nullptr;
        }
                        num = del->note_id;
        delete del;
                        nodes--;
        return num;
    }
    while (L->right_child != nullptr) {
        L = L->right_child;
    }
    if (del->right_child != nullptr) {
        L->right_child = del->right_child;
        del->right_child->parent = L;
    }
    root = del->left_child;
    root->parent = nullptr;
            num = del->note_id;
    delete del;
            return num;
            nodes--;
}
```

Листинг 21 - метод удаления узла из дерева

```
void SplayTree::insert(long long int x, int index) {
        nodes++;
    if (root == nullptr) {
        root = new TreeNode(x, index);
        return;
    }
        TreeNode* curr = this->root;
    while (curr != nullptr) {
        if (x < curr->ISBN) {
            if (curr->left_child == nullptr) {
                                        TreeNode* new_node = new TreeNode(x, index);
                curr->left_child = new_node;
                new_node->parent = curr;
                splay(new_node);
                this->root = new_node;
                return;
            }
            else {
                curr = curr->left_child;
            }
        }
        else if (x >= curr->ISBN) {
            if (curr->right_child == nullptr) {
                                        TreeNode* new_node = new TreeNode(x, index);
                curr->right_child = new_node;
                new_node->parent = curr;
                splay(new_node);
                this->root = new_node;
                return;
            }
            else {
                curr = curr->right_child;
            }
        }
        else {
            splay(curr);
            return;
        }
    }
}
```

Листинг 22 - метод вставки узла в дерево

```
TreeNode* SplayTree::search(long long int x) {
        TreeNode* res = nullptr;
        TreeNode* curr = this->root;
        TreeNode* prev = nullptr;
   while (curr != nullptr) {
      prev = curr;
      if (x < curr->ISBN) {
         curr = curr->left_child;
      }
      else if (x > curr->ISBN) {
         curr = curr->right_child;
      }
      else {
                              res = curr;
         break;
      }
   }
   if (res != nullptr) {
      splay(res);
   }
   else if (prev != nullptr) {
      splay(prev);
   }
        cout << res << endl;
   return res;
}
```

Листинг 23 - метод поиска узла по ключу

```
void SplayTree::fillFromFile(string file, int n) {
        ifstream f(file, ios::binary);

        char* key_str = new char[14]{};
        for (int i = 0; i < n; i++) {
                long long int key = 0;
                f.seekg(i*73, ios::beg);
                f.read(key_str, 13);
                for (int j = 0; j < 13; j++) {

                        key += int(key_str[12-j]-48) * pow(10, j);
                }
                insert(key, i);

        }
        f.seekg(0, ios::beg);
        f.close();
}

void SplayTree::prettyPrint(TreeNode* node, int level) {
        if (!node) return;
        prettyPrint(node->right_child, level + 1);
        for (int i = 1; i <= level * 15; i++)
                cout << ' ';
        cout << node->ISBN << "\n";
        prettyPrint(node->left_child, level + 1);
}
```

Листинг 24 - метод заполнения дерева из бинарного файла

3. Тестирование

```
Введите кол-во записей в файле: 10
Введите номер операции:
1. /Работать с хэш-таблицей
2. /Работать с бинарным деревом
3. /Работать с Косым деревом
3. /Сравнить способы
0. /Выход
2
Файл:
ISBN - 8729482058871: tvvgcqkmyhmnzbcswaqc ificyuevntepnuuhxmsrxwrgqbhvnmlysukugbyq
ISBN - 8009083322487: iepjbfdjqtmxzokfvcxl igmgapmusjdyeocmlutznvwhgtensvqvuqmkzrla
ISBN - 1768661600400: gaqxmiwrvwsvtalgwfuj xrbetfvkmesszfzemwqepimwpsfrahtsqybqkprx
ISBN - 1846546200924: icmhvfkuzxptxonoyuyo einphxjquelprkrhitlvweezgirvtydcxqoqbsbp
ISBN - 2695603011742: evnzqcsqdatygsdxrqpg lvndvnzvzpwzdoefcfyvbamrkwytrlckatodohvl
ISBN - 2981065321670: povkmaqvtlhjiubtvbha nrbriuyacbnaijctqrvjpxeeaqbvgjzirkhfssgw
ISBN - 1828737599220: itphaoeumzuwrvzslrah iaqszvbpvbwhyjqayiesyltknkrxfywhkniqunsw
ISBN - 8393630514163: ynrsrbpuwwmujuhqzcle xtxydtqsbtgezckuqtekwasregjdvvswsuglssul
ISBN - 1672744669184: djfijzidjygfkomkndqt ovidddrgspbyjnmuokzbwjeeotocvjqzgcdtkxtq
ISBN - 7560625512175: tfzkeyjizhmmxzvsangy zdusfpwjrbiinajnlptshidjycxrtzfatdkzlqvg
Бинарное Дерево:
8729482058871
                              8393630514163
              8009083322487
                                                                                                  7560625512175
                                                                                  2981065321670
                                                                  2695603011742
                                                1846546200924
                                                                  1828737599220
                              1768661600400
                                                1672744669184
Записи файла успешно скопированы в несбалансированное дерево!
Введите номер операции:
1. /Добавить запись
2. /Найти запись
3. /Удалить запись
4. /Отображение дерева
0. /Выход
```

Рисунок 1 - создание несбалансированнго бинарного дерева поиска заполненного из файла

```
Введите номер операции:
1. /Добавить запись
2. /Найти запись
3. /Удалить запись
4. /Отображение дерева
0. /Выход
1
Запись:
ISBN: 1234567891011
Автор: nikita
Название: khitrov
Введите номер операции:
1. /Добавить запись
2. /Найти запись
3. /Удалить запись
4. /Отображение дерева
0. /Выход
4
8729482058871
                              8393630514163
              8009083322487
                                                                                                  7560625512175
                                                                                  2981065321670
                                                                  2695603011742
                                                1846546200924
                                                                  1828737599220
                              1768661600400
                                                1672744669184
                                                                  1234567891011
```

Рисунок 2 - добавление записи в несбалансированное дерево поиска

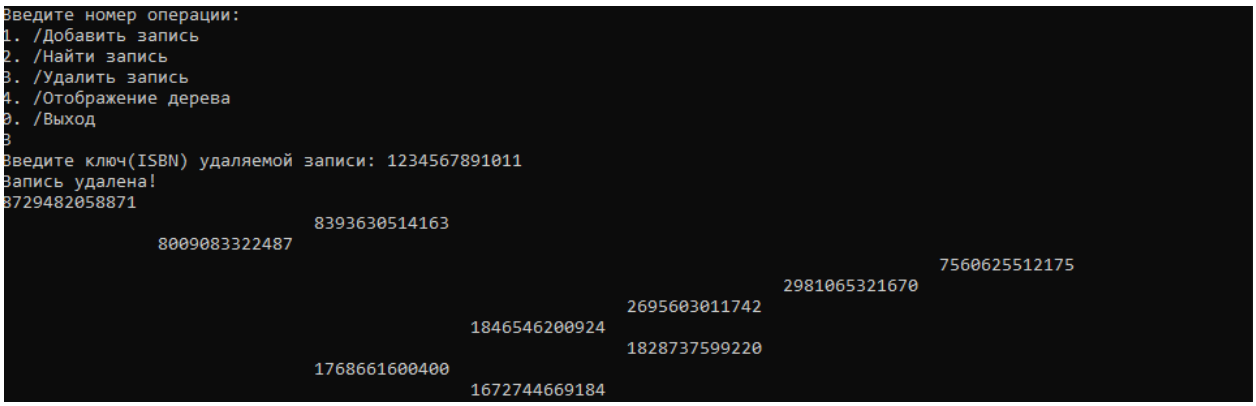

Рисунок 3 - удаление записи из несбалансированного дерева поиска

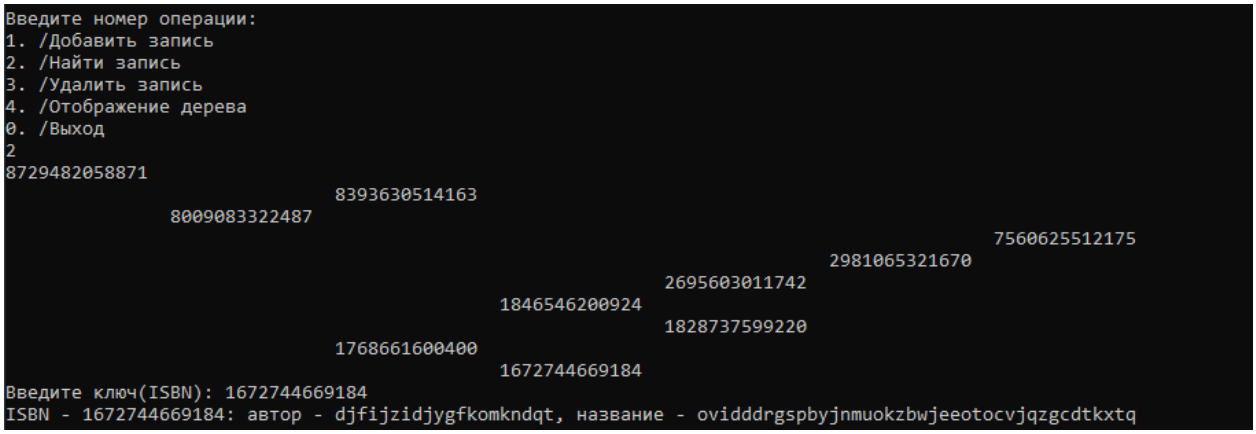

Рисунок 4 - поиск записи в файле по коду узла дерева

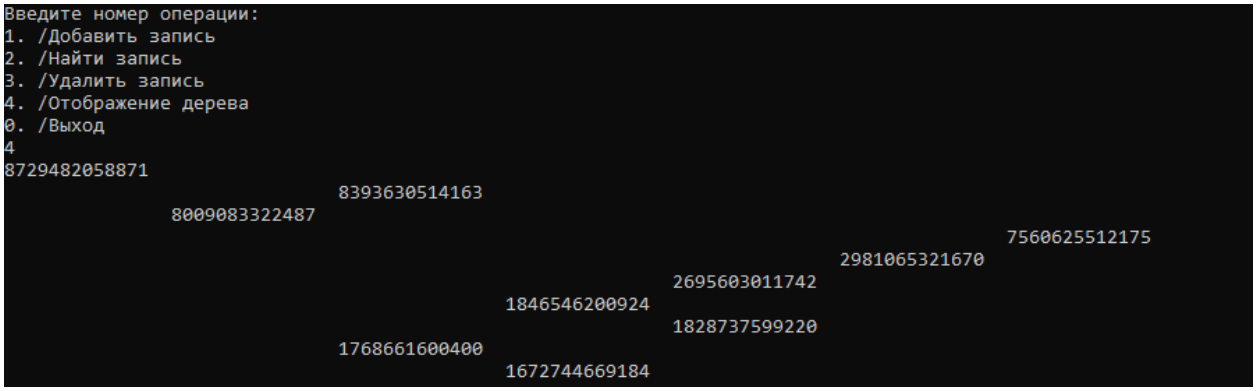

Рисунок 5 - отображение дерева

```
Файл:
ISBN - 8729482058871: tvvgcqkmyhmnzbcswaqc ificyuevntepnuuhxmsrxwrgqbhvnmlysukugbyq
ISBN - 8009083322487: iepjbfdjqtmxzokfvcxl igmgapmusjdyeocmlutznvwhgtensvqvuqmkzrla
ISBN - 1768661600400: gaqxmiwrvwsvtalgwfuj xrbetfvkmesszfzemwqepimwpsfrahtsqybqkprx
ISBN - 1846546200924: icmhvfkuzxptxonoyuyo einphxjquelprkrhitlvweezgirvtydcxqoqbsbp
ISBN - 2695603011742: evnzqcsqdatygsdxrqpg lvndvnzvzpwzdoefcfyvbamrkwytrlckatodohvl
ISBN - 2981065321670: povkmaqvtlhjiubtvbha nrbriuyacbnaijctqrvjpxeeaqbvgjzirkhfssgw
ISBN - 1828737599220: itphaoeumzuwrvzslrah iaqszvbpvbwhyjqayiesyltknkrxfywhkniqunsw
ISBN - 8393630514163: ynrsrbpuwwmujuhqzcle xtxydtqsbtgezckuqtekwasregjdvvswsuglssul
ISBN - 1672744669184: djfijzidjygfkomkndqt ovidddrgspbyjnmuokzbwjeeotocvjqzgcdtkxtq
ISBN - 7560625512175: tfzkeyjizhmmxzvsangy zdusfpwjrbiinajnlptshidjycxrtzfatdkzlqvg
Дерево:
                            8729482058871
              8393630514163
                            8009083322487
7560625512175
                                        2981065321670
                                                    2695603011742
                                                                1846546200924
                            1828737599220
                                        1768661600400
              1672744669184
Записи файла успешно скопированы в косое дерево!
```

Рисунок 6 - создание косого дерева заполненного из файла

```
Введите номер операции:
1. /Добавить запись
2. /Найти запись
3. /Удалить запись
4. /Отображение дерева
0. /Выход
2
                                                    8729482058871
                                        8393630514163
                                                    8009083322487
                            7560625512175
                                                    2981065321670
                                                                2695603011742
                                                                            1846546200924
                                        1828737599220
                                                    1768661600400
              1672744669184
1234567891011
Введите ключ(ISBN): 1234567891011
000001DD52742400
ISBN - 1234567891011: автор - nikita, название - khitrov
```

Рисунок 7 - поиск записи в файле по коду узла дерева

```
Введите ключ(ISBN) удаляемой записи: 1234567891011
000001DD52742400
Запись удалена!
Введите номер операции:
1. /Добавить запись
2. /Найти запись
3. /Удалить запись
4. /Отображение дерева
0. /Выход
4
                                      8729482058871
                            8393630514163
                                      8009083322487
              7560625512175
                                      2981065321670
                                                  2695603011742
                                                              1846546200924
                            1828737599220
                                      1768661600400
1672744669184
```

Рисунок 8 - удаление узла по его коду

```
Введите номер операции:
1. /Добавить запись
2. /Найти запись
3. /Удалить запись
4. /Отображение дерева
0. /Выход
4
                                        8729482058871
                                8393630514163
                                        8009083322487
                        7560625512175
                                        2981065321670
                                                        2695603011742
                                                                        1846546200924
                                1828737599220
                                        1768661600400
1672744669184
```

Рисунок 9 - отображение косого дерева

4. Задание 3

```
C:\Users\hitro\source\repos\SIAOD\x64\Debug\Pr2_4.exe
hash_ready
Enter the value to search for: 1910532132668
Хэш-таблица
node: 1910532132668 - compares: 2
ISBN - 1910532132668: izopshpulxejibqwnhtacueypggptnyfupomcjvkpcmyesjfaxsyqrcphcmh
8.0692
Несбалансированное дерево
node: 1910532132668, compares: 3
2.6079
Косое дерево
node: 1910532132668 - compares: 84
3.2139
Введите кол-во записей в файле: 10000
compare#2
Enter the value to search for: 7843047872025
Хэш-таблица
node: 7843047872025 - compares: 2
ISBN - 7843047872025: cxetjshehktbplvvfzgbvfndrjyafpvblhwltkmvterakzngpgfmyjavedjf
8.2218
Несбалансированное дерево
node: 7843047872025, compares: 27
2.8506
Косое дерево
node: 7843047872025 - compares: 521442
2.9828
Введите номер операции:
1. /Заполнить файл вручную
2. /Заполнить файл автоматически
0. /Выход
```

Вид поисковой структуры	Количество элементов, загруженных в структуру в момент выполнения поиска	Емкостная сложность: объем памяти для струтуры	Количество выполненных сравнений, время на поиск ключа в структуре
Хэш-таблица	10	O(10)	2 сравн., 8.0692 мс
Хэш-таблица	10000	O(10000)	2 сравн., 2.2218 мс
Бинарное дерево	10	O(10)	3 сравн., 2.6079 мс
Бинарное дерево	10000	O(10000)	27 сравн., 2.8506 мс
Косое дерево	10	O(10)	84 сравн., 3.2139 мс
Косое дерево	10000	O(10000)	521442 сравн., 2.9828 мс

Исходя из данных полученных в результате тестирования работы программы, можно сделать вывод, что сбалансированное дерево и несбалансированное дерево поиска как правило выполняют задачу поиска быстрее, чем хэш таблица, хотя и не на много при любых объемах данных. Поиск в сбалансированном дереве занял больше времени нежели в несбалансированном, так как помимо поиска необходимо проводить балансировку и при повторном поиске его скорость будет возрастать.

Вывод

В результате выполнения работы были получены навыки по разработке и реализации алгоритмов управления бинарным деревом поиска и сбалансированным бинарным деревом поиска, а также навыки применения файловых потоков прямого доступа к данным файла, получил навыки в применении сбалансированного дерева поиска для прямого доступа к записям.