|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  | https://www.mirea.ru/bitrix/templates/unlimtech/images/logo.png |  | | 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

# 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

# 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

# Тестирование

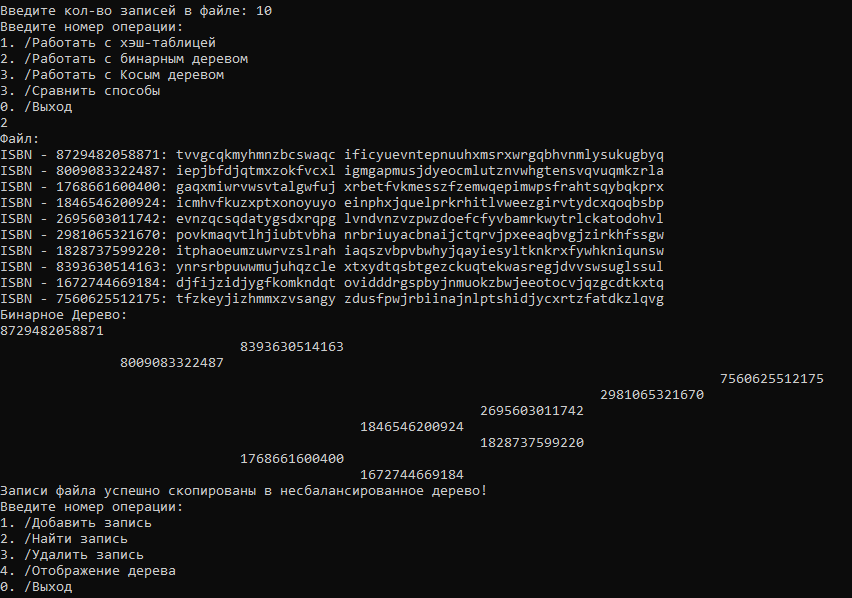


Рисунок - создание несбалансированнго бинарного дерева поиска заполненного из файла

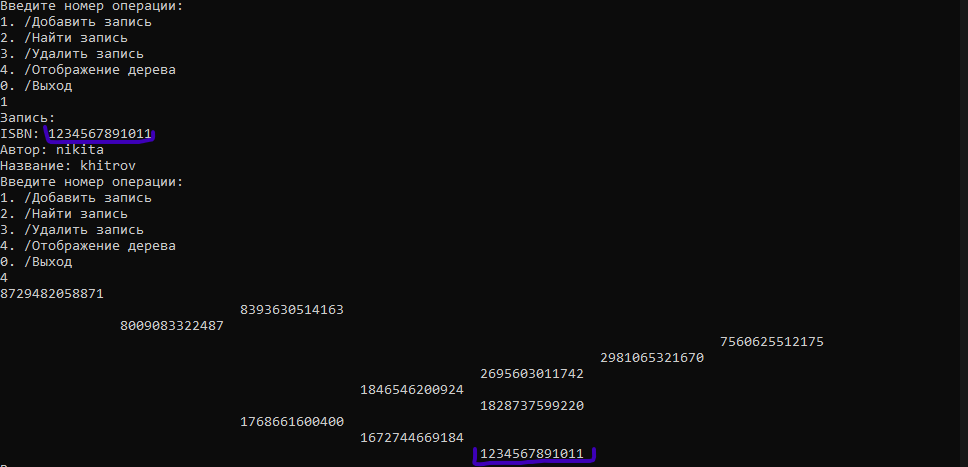


Рисунок - добавление записи в несбалансированное дерево поиска

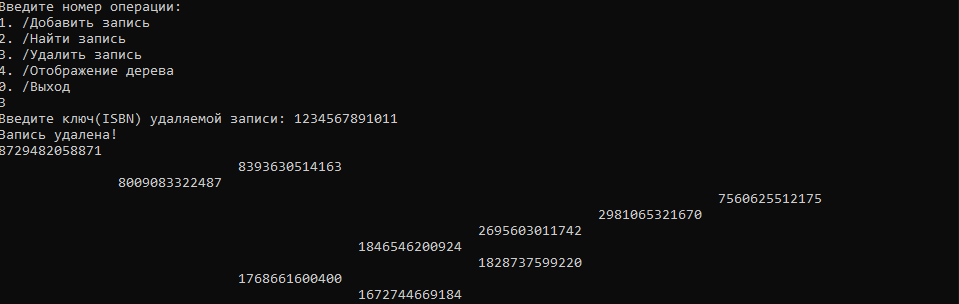


Рисунок - удаление записи из несбалансированного дерева поиска

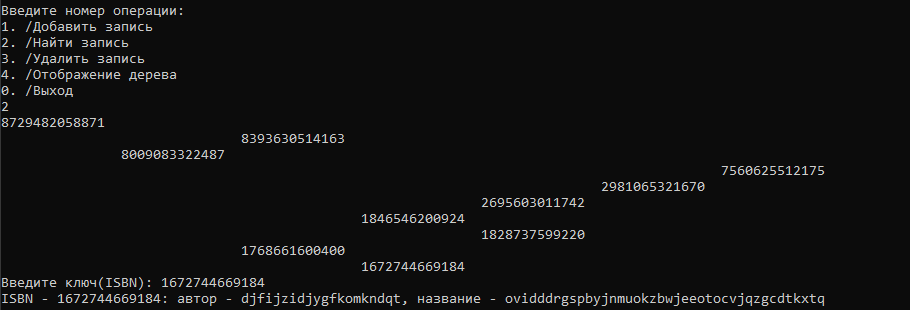


Рисунок - поиск записи в файле по коду узла дерева

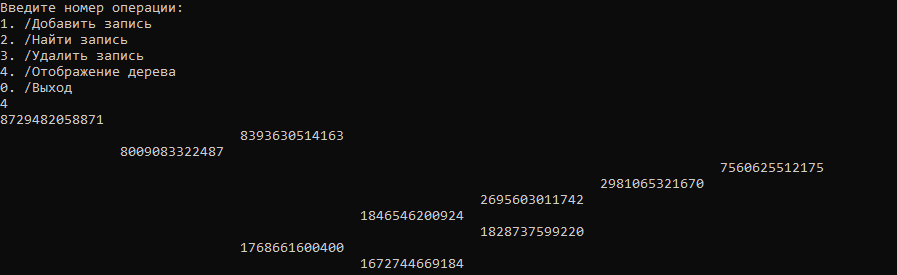


Рисунок - отображение дерева

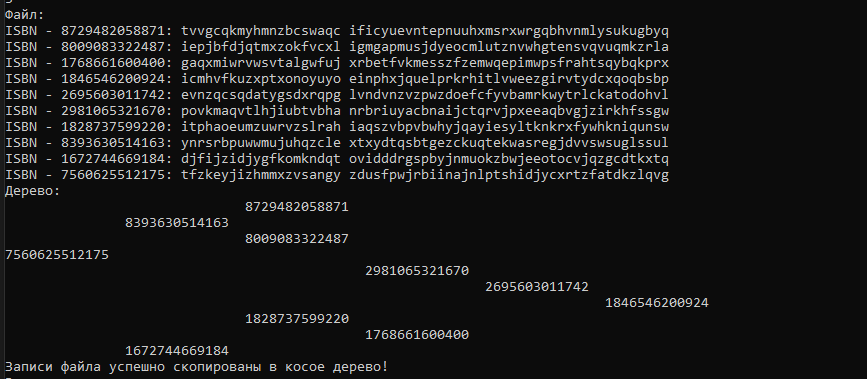


Рисунок - создание косого дерева заполненного из файла

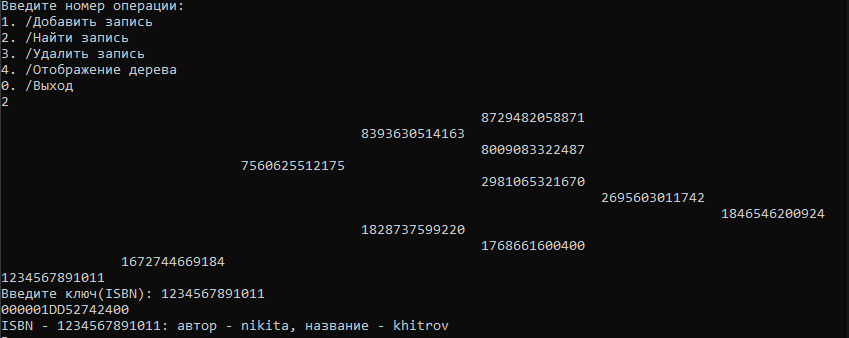


Рисунок - поиск записи в файле по коду узла дерева

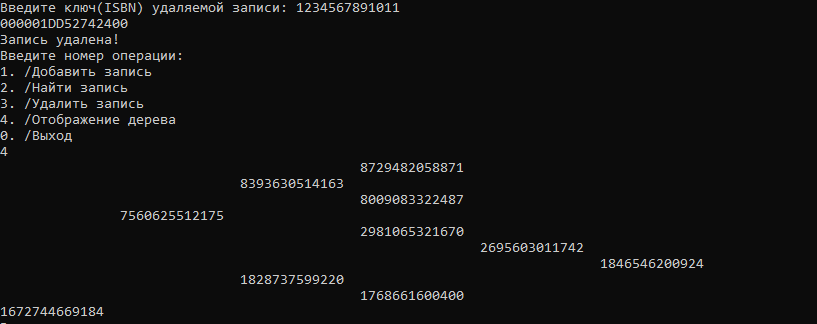


Рисунок - удаление узла по его коду

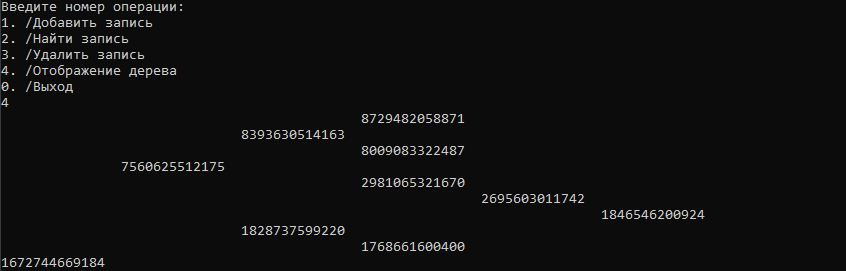
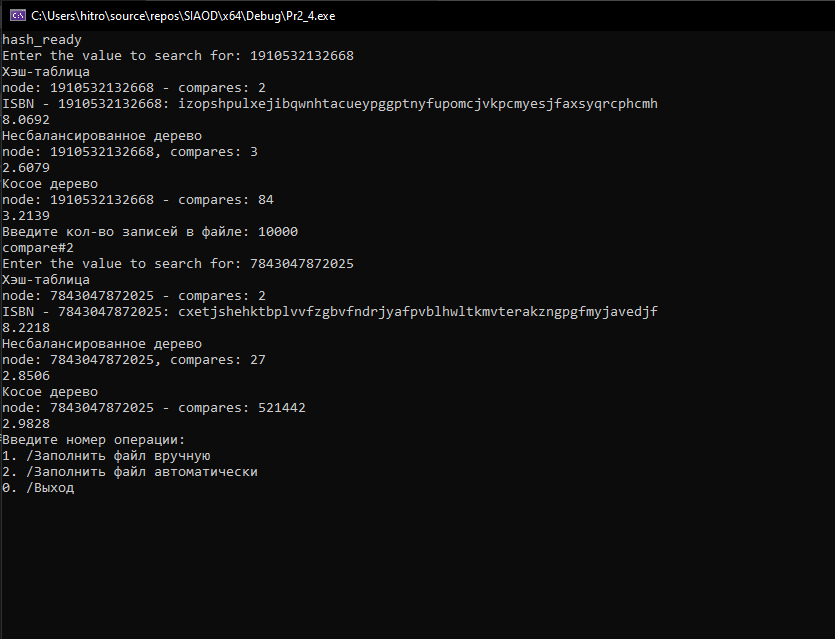


Рисунок - отображение косого дерева

# Задание 3



|  |  |  |  |
| --- | --- | --- | --- |
| 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 | О(10) | 2 сравн.,  8.0692 мс |
| Хэш-таблица | 10000 | О(10000) | 2 сравн.,  2.2218 мс |
| Бинарное дерево | 10 | О(10) | 3 сравн.,  2.6079 мс |
| Бинарное дерево | 10000 | О(10000) | 27 сравн.,  2.8506 мс |
| Косое дерево | 10 | О(10) | 84 сравн.,  3.2139 мс |
| Косое дерево | 10000 | О(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.