|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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 № 5** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Сбалансированные деревья поиска (СДП) и их применение для поиска данных в файле»** | |
|  | |
| Выполнил студент группы ИКБО-33-21 | Зарожина Я.А**.** |
| Принял преподаватель | Муравьева Е.А. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2022

ПРАКТИЧЕСКАЯ РАБОТА № 5.

Цель

- получить навыки в разработки и реализации алгоритмов управления бинарным деревом поиска и сбалансированными бинарными деревьями поиска (АВЛ – деревьями);

- получить навыки в применении файловых потоков прямого доступа к данным файла;

- получить навыки в применении сбалансированного дерева поиска для прямого доступа к записям файла.

ЗАДАНИЯ.

Задание 1.

Разработать приложение, которое использует бинарное дерево поиска (БДП) для поиска записи с ключом в файле.

1. Разработать класс (или библиотеку функций) «Бинарное дерево поиска». Тип информационной части узла дерева: 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 (библиотеку функций) 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.

3. Разработать и протестировать приложение.

4. Подготовить отчет

Задание 2.

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.

5. Оформить отчет

Номер варианта

|  |  |  |
| --- | --- | --- |
| 10 | В-дерево | Англо – русский словарь: английское слово, русское слово |

ОПИСАНИЕ МОДЕЛИ РЕШЕНИЯ ЗАДАЧИ.

Был разработан класс, содержащий узлы дерева, который содержал структуру ячейки таблицы, а также методы: поиск ключа, удаление данных, добавление данных, вывод графа и пр.

|  |
| --- |
| class TreeNode  {  string \*keys;  int \*values;  int t;  TreeNode \*\*C;  int n;  bool leaf;  public:  TreeNode(int temp, bool bool\_leaf);  int findKey(string k);  void insertNonFull(string k, int value);  void splitChild(int i, TreeNode \*y);  void traverse();  void deletion(string k);  void removeFromLeaf(int idx);  void removeFromNonLeaf(int idx);  string getPredecessor(int idx);  string getSuccessor(int idx);  void fill(int idx);  void borrowFromPrev(int idx);  void borrowFromNext(int idx);  void merge(int idx);  int search(string k);  friend class BTree;  };  class BTree  {  TreeNode \*root;  int t;  public:  BTree(int temp)  {  root = NULL;  t = temp;  }  void traverse()  {  if (root != NULL)  root->traverse();  }  int search(string k)  {  return (root == NULL) ? NULL : root->search(k);  }  void insert(string k, int value);  void deletion(string k);  }; |

Метод поиск ключа, осуществляется в файле BTree.cpp и представлен в следующем виде:

|  |
| --- |
| // Find the key  int TreeNode::findKey(string k)  {  int idx = 0;  while (idx < n && keys[idx] < k)  {  ++idx;  }  return idx;  } |

Метод операции удаления, осуществляется в файле BTree.cpp и представлен в следующем виде:

|  |
| --- |
| void TreeNode::deletion(string k)  {  int idx = findKey(k);  if (idx < n && keys[idx] == k)  {  if (leaf)  {  removeFromLeaf(idx);  }  else  {  removeFromNonLeaf(idx);  }  }  else  {  if (leaf)  {  cout << "The key " << k << " is does not exist in the tree\n";  return;  }  bool flag = ((idx == n) ? true : false);  if (C[idx]->n < t)  {  fill(idx);  }  if (flag && idx > n)  {  C[idx - 1]->deletion(k);  }  else  {  C[idx]->deletion(k);  }  }  return;  } |

Метод удаления из списка , осуществляется в файле BTree.cpp и представлен в следующем виде:

|  |
| --- |
| void TreeNode::removeFromLeaf(int idx)  {  for (int i = idx + 1; i < n; ++i)  {  keys[i - 1] = keys[i];  values[i - 1] = values[i];  }  n--;  return;  } |

Метод удаления из узла, осуществляется в файле BTree.cpp и представлен в следующем виде:

|  |
| --- |
| // Delete from non leaf node  void TreeNode::removeFromNonLeaf(int idx)  {  string k = keys[idx];  int v = values[idx];  if (C[idx]->n >= t)  {  string pred = getPredecessor(idx);  keys[idx] = pred;  C[idx]->deletion(pred);  }  else if (C[idx + 1]->n >= t)  {  string succ = getSuccessor(idx);  keys[idx] = succ;  C[idx + 1]->deletion(succ);  }  else  {  merge(idx);  C[idx]->deletion(k);  }  return;  }  string TreeNode::getPredecessor(int idx)  {  TreeNode \*cur = C[idx];  while (!cur->leaf)  {  cur = cur->C[cur->n];  }  return cur->keys[cur->n - 1];  } |

Метод borrowFromPrev – «заимствование у предыдущего», осуществляется в файле BTree.cpp и представлен в следующем виде:

|  |
| --- |
| void TreeNode::borrowFromPrev(int idx)  {  TreeNode \*child = C[idx];  TreeNode \*sibling = C[idx - 1];  for (int i = child->n - 1; i >= 0; --i)  {  child->keys[i + 1] = child->keys[i];  child->values[i + 1] = child->values[i];  }  if (!child->leaf)  {  for (int i = child->n; i >= 0; --i)  {  child->C[i + 1] = child->C[i];  }  }  child->keys[0] = keys[idx - 1];  child->values[0] = values[idx - 1];  if (!child->leaf)  {  child->C[0] = sibling->C[sibling->n];  }  keys[idx - 1] = sibling->keys[sibling->n - 1];  values[idx - 1] = sibling->values[sibling->n - 1];  child->n += 1;  sibling->n -= 1;  return;  } |

Метод borrowFromNext– «заимствование у следующего», осуществляется в файле BTree.cpp и представлен в следующем виде:

|  |
| --- |
| void TreeNode::borrowFromNext(int idx)  {  TreeNode \*child = C[idx];  TreeNode \*sibling = C[idx + 1];  child->keys[(child->n)] = keys[idx];  child->values[(child->n)] = values[idx];  if (!(child->leaf))  {  child->C[(child->n) + 1] = sibling->C[0];  }  keys[idx] = sibling->keys[0];  values[idx] = sibling->values[0];  for (int i = 1; i < sibling->n; ++i)  {  sibling->keys[i - 1] = sibling->keys[i];  sibling->values[i - 1] = sibling->values[i];  }  if (!sibling->leaf)  {  for (int i = 1; i <= sibling->n; ++i)  {  sibling->C[i - 1] = sibling->C[i];  }  }  child->n += 1;  sibling->n -= 1;  return;  } |

Метод поглощения– «merge», осуществляется в файле BTree.cpp и представлен в следующем виде:

|  |
| --- |
| // Merge  void TreeNode::merge(int idx)  {  TreeNode \*child = C[idx];  TreeNode \*sibling = C[idx + 1];  child->keys[t - 1] = keys[idx];  child->values[t - 1] = values[idx];  for (int i = 0; i < sibling->n; ++i)  {  child->keys[i + t] = sibling->keys[i];  child->values[i + t] = sibling->values[i];  }  if (!child->leaf)  {  for (int i = 0; i <= sibling->n; ++i)  {  child->C[i + t] = sibling->C[i];  }  }  for (int i = idx + 1; i < n; ++i)  {  keys[i - 1] = keys[i];  values[i - 1] = values[i];  }  for (int i = idx + 2; i <= n; ++i)  {  C[i - 1] = C[i];  }  child->n += sibling->n + 1;  n--;  delete (sibling);  return;  } |

Метод удаления B-дерева, осуществляется в файле BTree.cpp и представлен в следующем виде:

|  |
| --- |
| // Delete Operation  void BTree::deletion(string k)  {  if (!root)  {  cout << "The tree is empty\n";  return;  }  root->deletion(k);  if (root->n == 0)  {  TreeNode \*tmp = root;  if (root->leaf)  {  root = NULL;  }  else  {  root = root->C[0];  }  delete tmp;  }  return;  } |

BinFile.h содержи структуру словаря:

|  |
| --- |
| struct Dictionary  {  char eng\_word[255];  char rus\_word[255];  }; |

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ПРОГРАММЫ.

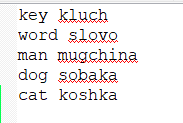


Рис.1 - Текстовый файл 1.txt

При первом заходе в программу, пользователь видит список команд, где необходимо выбрать номер задания.

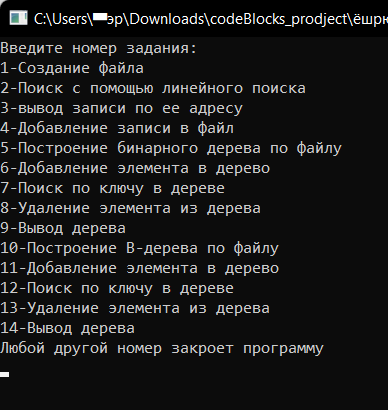


Рис. 0 Пользовательский интерфейс

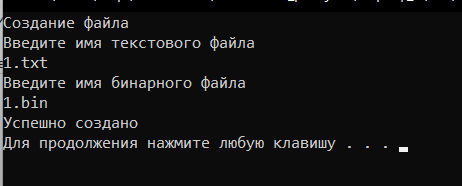


Рис.1 – Создание файла

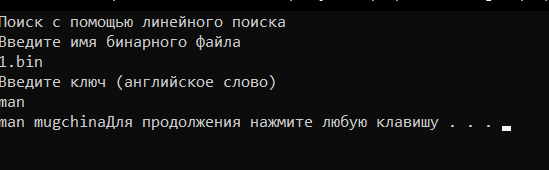


Рис.2 – Поиск с помощью линейного поиска

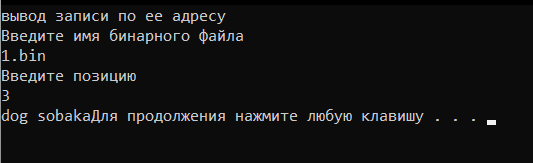


Рис.3 – Вывод записи по ее адресу

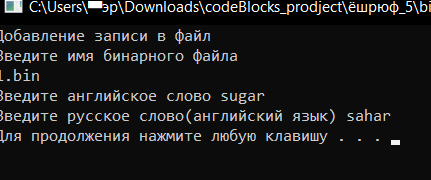




Рис.4 – Добавление записи в файл

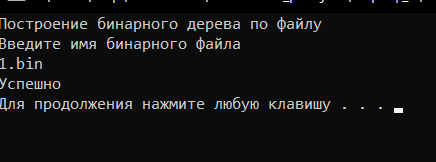


Рис.5 – Построение бинарного дерева по файлу

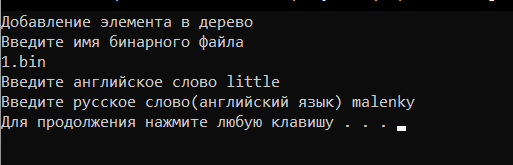


Рис 6. – Добавление элемента в дерево

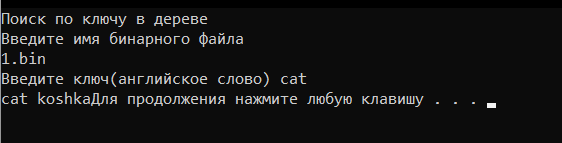


Рис.7 – Поиск по ключу в дереве

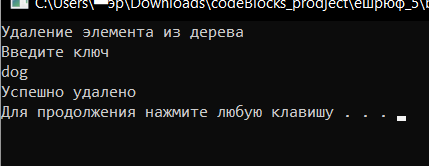


Рис.8 – Удаление элемента из дерева

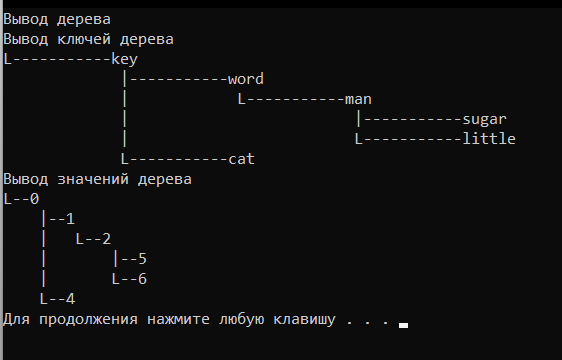


Рис.9 – Вывод дерева

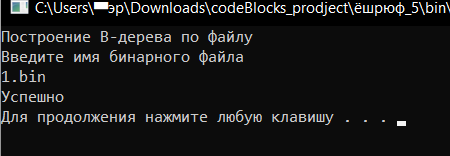


Рис.10 – Построение B-дерева по файлу

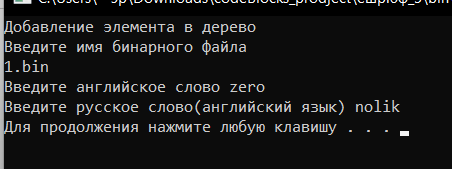


Рис.11 – Добавление элемента в дерево

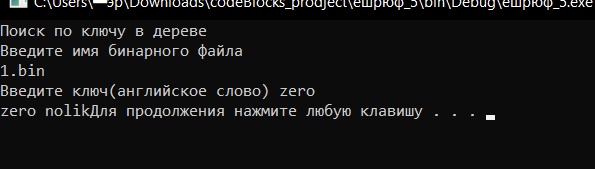


Рис.12 – Поиск по ключу в дереве

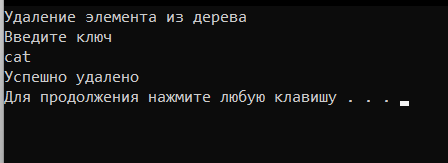


Рис.13 – Удаление элемента из дерева

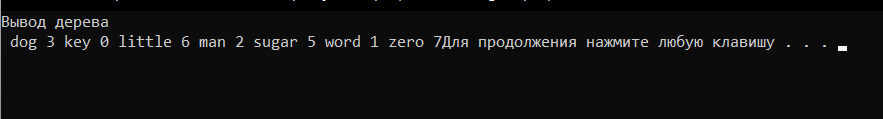


Рис.14 – Вывод дерева

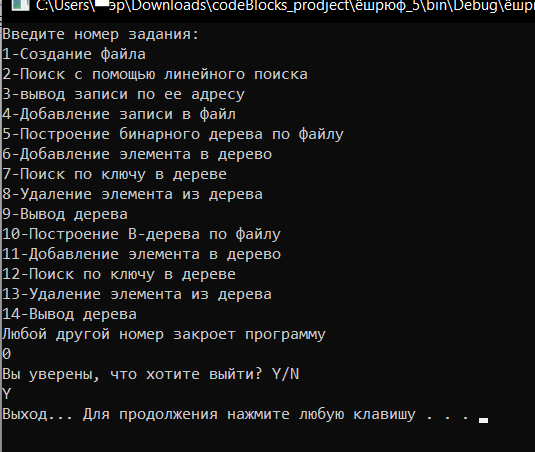


Рис.15 – Выход из программы

# ИСХОДНЫЙ КОД ПРОГРАММЫ.

1. BinFile.h

|  |
| --- |
| #pragma once  #include <iostream>  #include <fstream>  #include <string>  using namespace std;  struct Dictionary  {  char eng\_word[255];  char rus\_word[255];  };  class BinFile  {  public:  static int txt\_to\_bin(string txt\_file, string bin\_file);  static Dictionary get\_class(string filename, int position);  static void AddToBin(ofstream& bin\_file, Dictionary Dictionary);  static Dictionary linear\_search(string txt\_file, string key);  static string\* getwords(string txt\_file);  }; |

2. BinFile.cpp

|  |
| --- |
| #include "BinFile.h"  #include <stdio.h>  #include <string>  #include <sstream>  #include <vector>  #include <iostream>  #include <algorithm>  #include <fstream>  #include <sstream>  #include <vector>  #include <iostream>  #include <algorithm>  #include <fstream>  using namespace std;  int BinFile::txt\_to\_bin(string txt\_file, string bin\_file)  {  ifstream inp;  inp.open(txt\_file, ios::binary | ios::app);  ofstream outp;  outp.open(bin\_file, ios::binary | ios::app);  if (inp.good() && outp.good())  {  Dictionary tt;  while (true)  {  inp >> tt.eng\_word;  if (!inp.eof())  {  inp >> tt.rus\_word;  outp.write((char\*)&tt, sizeof(Dictionary));  }  else break;  }  inp.close();  outp.close();  return 1;  }  else  {  inp.close();  outp.close();  return 0;  }  }  Dictionary BinFile::get\_class(string filename, int position)  {  Dictionary tt;  ifstream bin\_file(filename, ios::binary | ios::in);  bin\_file.seekg((position) \* sizeof(tt));  bin\_file.read((char\*)&tt, sizeof(Dictionary));  bin\_file.close();  return tt;  }  void BinFile::AddToBin(ofstream& bin\_file, Dictionary Dictionary)  {  bin\_file.seekp(0, ios::end);  bin\_file.write((char\*)&Dictionary, sizeof(Dictionary));  }  Dictionary BinFile::linear\_search(string bin\_file, string key)  {  ifstream inp;  inp.open(bin\_file, ios::binary | ios::app);  Dictionary tt;  tt.eng\_word[0] = ' ';  while (true)  {  inp.read((char\*)&tt, sizeof(tt));  if (!inp.eof())  {  if (tt.eng\_word == key)  {  return tt;  }  }  else  {  break;  }  }  inp.close();  return tt;  }  string\* BinFile::getwords(string txt\_file)  {  ifstream inp;  inp.open(txt\_file, ios::binary | ios::app);  Dictionary tt;  string \*words= new string[10000];  int i = 0;  while (true)  {  inp.read((char\*)&tt, sizeof(tt));  if (!inp.eof())  {  words[i] = tt.eng\_word;  i++;  }  else  {  break;  }  }  inp.close();  return words;  } |

3. BST.h

|  |
| --- |
| #pragma once  #include <iostream>  #include <string>  #include <cstring>  using namespace std;  class node  {  public:  int value;  char key[255];  node \*left, \*right;  node(string a, int b)  {  strcpy(key, a.c\_str());  value = b;  left = nullptr;  right = nullptr;  }  static bool exists(node \* root, string a);  static int search(node \* root, string a);  static node \* insert(node \* root, string a, int b);  static class node \*minValueNode(class node \*node);  static class node \*deleteNode(class node \*root, string key);  static void printBT(const string& prefix, const node\* node, bool isLeft, bool key);  }; |

4. BST.cpp

|  |
| --- |
| #include "BST.h"  #include <sstream>  #include <vector>  #include <iostream>  #include <algorithm>  #include <fstream>  using namespace std;  bool node::exists(node \* root, string a)  {  if (root == nullptr)  {  return false;  }  if (root->key == a)  {  return true;  }  if (a < root->key)  {  return exists(root->left, a);  }  return exists(root->right, a);  }  int node::search(node \* root, string a)  {  if (root == nullptr)  {  return -1;  }  if (root->key == a)  {  return root->value;  }  if (a < root->key)  {  return search(root->left, a);  }  return search(root->right, a);  }  node \* node::insert(node \* root, string a, int b)  {  if (exists(root, a))  {  return root;  }  if (root == nullptr)  {  return new node(a, b);  }  if (a < root->key)  {  root->left = insert(root->left, a, b);  }  if (a > root->key)  {  root->right = insert(root->right, a, b);  }  return root;  }  node \* node::minValueNode(class node \*node)  {  class node \*current = node;  while (current && current->left != NULL)  {  current = current->left;  }  return current;  }  node \* node::deleteNode(node \* root, string key)  {  if (root == NULL)  {  return root;  }  if (key < root->key)  {  root->left = deleteNode(root->left, key);  }  else if (key > root->key)  {  root->right = deleteNode(root->right, key);  }  else  {  if (root->left == NULL)  {  class node \*temp = root->right;  free(root);  return temp;  }  else if (root->right == NULL)  {  class node \*temp = root->left;  free(root);  return temp;  }  class node \*temp = minValueNode(root->right);  strcpy(root->key, temp->key);  root->value = temp->value;  root->right = deleteNode(root->right, temp->key);  }  return root;  }  void node::printBT(const string& prefix, const node\* node, bool isLeft, bool key)  {  if (key)  {  if (node != nullptr)  {  std::cout << prefix;  std::cout << (isLeft ? "|-----------" : "L-----------");  std::cout << node->key << endl;  printBT(prefix + (isLeft ? "| " : " "), node->right, true,1);  printBT(prefix + (isLeft ? "| " : " "), node->left, false,1);  }  }  else  {  if (node != nullptr)  {  std::cout << prefix;  std::cout << (isLeft ? "|--" : "L--");  std::cout << node->value << endl;  printBT(prefix + (isLeft ? "| " : " "), node->right, true,0);  printBT(prefix + (isLeft ? "| " : " "), node->left, false,0);  }  }  } |

5. BTree.h

|  |
| --- |
| #pragma once  #include <iostream>  #include <string>  #include <fstream>  #include <cstring>  using namespace std;  class TreeNode  {  string \*keys;  int \*values;  int t;  TreeNode \*\*C;  int n;  bool leaf;  public:  TreeNode(int temp, bool bool\_leaf);  int findKey(string k);  void insertNonFull(string k, int value);  void splitChild(int i, TreeNode \*y);  void traverse();  void deletion(string k);  void removeFromLeaf(int idx);  void removeFromNonLeaf(int idx);  string getPredecessor(int idx);  string getSuccessor(int idx);  void fill(int idx);  void borrowFromPrev(int idx);  void borrowFromNext(int idx);  void merge(int idx);  int search(string k);  friend class BTree;  };  class BTree  {  TreeNode \*root;  int t;  public:  BTree(int temp)  {  root = NULL;  t = temp;  }  void traverse()  {  if (root != NULL)  root->traverse();  }  int search(string k)  {  return (root == NULL) ? NULL : root->search(k);  }  void insert(string k, int value);  void deletion(string k);  }; |

6. BTree.cpp

|  |
| --- |
| #include "BTree.h"  using namespace std;  TreeNode::TreeNode(int t1, bool leaf1)  {  t = t1;  leaf = leaf1;  keys = new string[2 \* t - 1];  values = new int[2 \* t - 1];  C = new TreeNode \*[2 \* t];  n = 0;  }  // Find the key  int TreeNode::findKey(string k)  {  int idx = 0;  while (idx < n && keys[idx] < k)  {  ++idx;  }  return idx;  }  // Deletion operation  void TreeNode::deletion(string k)  {  int idx = findKey(k);  if (idx < n && keys[idx] == k)  {  if (leaf)  {  removeFromLeaf(idx);  }  else  {  removeFromNonLeaf(idx);  }  }  else  {  if (leaf)  {  cout << "The key " << k << " is does not exist in the tree\n";  return;  }  bool flag = ((idx == n) ? true : false);  if (C[idx]->n < t)  {  fill(idx);  }  if (flag && idx > n)  {  C[idx - 1]->deletion(k);  }  else  {  C[idx]->deletion(k);  }  }  return;  }  // Remove from the leaf  void TreeNode::removeFromLeaf(int idx)  {  for (int i = idx + 1; i < n; ++i)  {  keys[i - 1] = keys[i];  values[i - 1] = values[i];  }  n--;  return;  }  // Delete from non leaf node  void TreeNode::removeFromNonLeaf(int idx)  {  string k = keys[idx];  int v = values[idx];  if (C[idx]->n >= t)  {  string pred = getPredecessor(idx);  keys[idx] = pred;  C[idx]->deletion(pred);  }  else if (C[idx + 1]->n >= t)  {  string succ = getSuccessor(idx);  keys[idx] = succ;  C[idx + 1]->deletion(succ);  }  else  {  merge(idx);  C[idx]->deletion(k);  }  return;  }  string TreeNode::getPredecessor(int idx)  {  TreeNode \*cur = C[idx];  while (!cur->leaf)  {  cur = cur->C[cur->n];  }  return cur->keys[cur->n - 1];  }  string TreeNode::getSuccessor(int idx)  {  TreeNode \*cur = C[idx + 1];  while (!cur->leaf)  cur = cur->C[0];  return cur->keys[0];  }  void TreeNode::fill(int idx)  {  if (idx != 0 && C[idx - 1]->n >= t)  {  borrowFromPrev(idx);  }  else if (idx != n && C[idx + 1]->n >= t)  {  borrowFromNext(idx);  }  else  {  if (idx != n)  {  merge(idx);  }  else  {  merge(idx - 1);  }  }  return;  }  // Borrow from previous  void TreeNode::borrowFromPrev(int idx)  {  TreeNode \*child = C[idx];  TreeNode \*sibling = C[idx - 1];  for (int i = child->n - 1; i >= 0; --i)  {  child->keys[i + 1] = child->keys[i];  child->values[i + 1] = child->values[i];  }  if (!child->leaf)  {  for (int i = child->n; i >= 0; --i)  {  child->C[i + 1] = child->C[i];  }  }  child->keys[0] = keys[idx - 1];  child->values[0] = values[idx - 1];  if (!child->leaf)  {  child->C[0] = sibling->C[sibling->n];  }  keys[idx - 1] = sibling->keys[sibling->n - 1];  values[idx - 1] = sibling->values[sibling->n - 1];  child->n += 1;  sibling->n -= 1;  return;  }  // Borrow from the next  void TreeNode::borrowFromNext(int idx)  {  TreeNode \*child = C[idx];  TreeNode \*sibling = C[idx + 1];  child->keys[(child->n)] = keys[idx];  child->values[(child->n)] = values[idx];  if (!(child->leaf))  {  child->C[(child->n) + 1] = sibling->C[0];  }  keys[idx] = sibling->keys[0];  values[idx] = sibling->values[0];  for (int i = 1; i < sibling->n; ++i)  {  sibling->keys[i - 1] = sibling->keys[i];  sibling->values[i - 1] = sibling->values[i];  }  if (!sibling->leaf)  {  for (int i = 1; i <= sibling->n; ++i)  {  sibling->C[i - 1] = sibling->C[i];  }  }  child->n += 1;  sibling->n -= 1;  return;  }  // Merge  void TreeNode::merge(int idx)  {  TreeNode \*child = C[idx];  TreeNode \*sibling = C[idx + 1];  child->keys[t - 1] = keys[idx];  child->values[t - 1] = values[idx];  for (int i = 0; i < sibling->n; ++i)  {  child->keys[i + t] = sibling->keys[i];  child->values[i + t] = sibling->values[i];  }  if (!child->leaf)  {  for (int i = 0; i <= sibling->n; ++i)  {  child->C[i + t] = sibling->C[i];  }  }  for (int i = idx + 1; i < n; ++i)  {  keys[i - 1] = keys[i];  values[i - 1] = values[i];  }  for (int i = idx + 2; i <= n; ++i)  {  C[i - 1] = C[i];  }  child->n += sibling->n + 1;  n--;  delete (sibling);  return;  }  void TreeNode::traverse()  {  int i;  for (i = 0; i < n; i++)  {  if (leaf == false)  {  C[i]->traverse();  }  cout << " " << keys[i];  cout << " " << values[i];  }  if (leaf == false)  {  C[i]->traverse();  }  }  int TreeNode::search(string k)  {  int i = 0;  while (i < n && k > keys[i])  {  i++;  }  if (keys[i] == k)  {  return values[i];  }  if (leaf == true)  {  return NULL;  }  return C[i]->search(k);  }  void BTree::insert(string k, int value)  {  if (root == NULL)  {  root = new TreeNode(t, true);  root->keys[0] = k;  root->values[0] = value;  root->n = 1;  }  else  {  if (root->n == 2 \* t - 1)  {  TreeNode \*s = new TreeNode(t, false);  s->C[0] = root;  s->splitChild(0, root);  int i = 0;  if (s->keys[0] < k)  {  i++;  }  s->C[i]->insertNonFull(k, value);  root = s;  }  else  root->insertNonFull(k, value);  }  }  void TreeNode::insertNonFull(string k, int value)  {  int i = n - 1;  if (leaf == true)  {  while (i >= 0 && keys[i] > k)  {  keys[i + 1] = keys[i];  values[i + 1] = values[i];  i--;  }  keys[i + 1] = k;  values[i + 1] = value;  n = n + 1;  }  else  {  while (i >= 0 && keys[i] > k)  {  i--;  }  if (C[i + 1]->n == 2 \* t - 1)  {  splitChild(i + 1, C[i + 1]);  if (keys[i + 1] < k)  {  i++;  }  }  C[i + 1]->insertNonFull(k, value);  }  }  void TreeNode::splitChild(int i, TreeNode \*y)  {  TreeNode \*z = new TreeNode(y->t, y->leaf);  z->n = t - 1;  for (int j = 0; j < t - 1; j++)  {  z->keys[j] = y->keys[j + t];  z->values[j] = y->values[j + t];  }  if (y->leaf == false)  {  for (int j = 0; j < t; j++)  {  z->C[j] = y->C[j + t];  }  }  y->n = t - 1;  for (int j = n; j >= i + 1; j--)  {  C[j + 1] = C[j];  }  C[i + 1] = z;  for (int j = n - 1; j >= i; j--)  {  keys[j + 1] = keys[j];  values[j + 1] = values[j];  }  keys[i] = y->keys[t - 1];  values[i] = y->values[t - 1];  n = n + 1;  }  // Delete Operation  void BTree::deletion(string k)  {  if (!root)  {  cout << "The tree is empty\n";  return;  }  root->deletion(k);  if (root->n == 0)  {  TreeNode \*tmp = root;  if (root->leaf)  {  root = NULL;  }  else  {  root = root->C[0];  }  delete tmp;  }  return;  } |

7. Source.cpp

|  |
| --- |
| #include"BST.h"  #include "BinFile.h"  #include "BTree.h"  #include <sstream>  #include <vector>  #include <iterator>  #include <iostream>  #include <algorithm>  #include <fstream>  using namespace std;  extern double counter;  int main()  {  setlocale(LC\_ALL, "rus");  class node \*root = NULL;  BTree t(2);  string txt\_file, bin\_file;  string key;  Dictionary temp;  int adress;  string \*words/\* = new string[10000]\*/;  int i = 0;  while (true)  {  system("cls");  cout << "Введите номер задания:\n1-Создание файла\n2-Поиск с помощью линейного поиска\n3-вывод записи по ее адресу\n4-Добавление записи в файл\n5-Построение бинарного дерева по файлу\n6-Добавление элемента в дерево\n7-Поиск по ключу в дереве\n8-Удаление элемента из дерева\n9-Вывод дерева\n10-Построение B-дерева по файлу\n11-Добавление элемента в дерево\n12-Поиск по ключу в дереве\n13-Удаление элемента из дерева\n14-Вывод дерева\nЛюбой другой номер закроет программу" << endl;  int k = 0;  cin >> k;  if (k == 1)  {  system("cls");  cout << "Создание файла" << endl;  cout << "Введите имя текстового файла" << endl;  cin >> txt\_file;  cout << "Введите имя бинарного файла" << endl;  cin >> bin\_file;  if (BinFile::txt\_to\_bin(txt\_file, bin\_file) == 1)  {  cout << "Успешно создано" << endl;  }  else  {  cout << "Файл не существует" << endl;  }  system("pause");  }  if (k == 2)  {  system("cls");  cout << "Поиск с помощью линейного поиска" << endl;  cout << "Введите имя бинарного файла" << endl;  cin >> bin\_file;  cout << "Введите ключ (английское слово)" << endl;  cin >> key;  temp=BinFile::linear\_search(bin\_file, key);  if (temp.eng\_word[0] != ' ')  {  cout << temp.eng\_word << " " << temp.rus\_word;  }  else  {  cout <<"Запись с таким ключом не найдена" << endl;;  }  system("pause");  }  if (k == 3)  {  system("cls");  cout << "вывод записи по ее адресу" << endl;  cout << "Введите имя бинарного файла" << endl;  cin >> bin\_file;  cout << "Введите позицию" << endl;  cin >> adress;  temp=BinFile::get\_class(bin\_file, adress);  if (temp.eng\_word[0] != ' ')  {  cout << temp.eng\_word << " " << temp.rus\_word;  }  else  {  cout << "Запись с таким ключом не найдена" << endl;;  }  system("pause");  }  if (k == 4)  {  system("cls");  cout << "Добавление записи в файл" << endl;  cout << "Введите имя бинарного файла" << endl;  cin >> bin\_file;  ofstream outp;  outp.open(bin\_file, ios::binary|ios::app);  cout << "Введите английское слово ";  cin >> temp.eng\_word;  cout << "Введите русское слово(английский язык) ";  cin >> temp.rus\_word;  if (outp.good())  {  BinFile::AddToBin(outp, temp);  }  else  {  cout << "Такого файла не существует" << endl;  }  outp.close();  system("pause");  }  if (k == 5)  {  system("cls");  cout << "Построение бинарного дерева по файлу" << endl;  cout << "Введите имя бинарного файла" << endl;  cin >> bin\_file;  ifstream inp;  inp.open(bin\_file, ios::binary | ios::app);  if (inp.good())  {  inp.close();  words = BinFile::getwords(bin\_file);  while (words[i] != "\0")  {  root = node::insert(root, words[i], i);  i++;  }  cout << "Успешно" << endl;  //delete[] words;  }  else  {  cout << "Файл не существует" << endl;  }  system("pause");  }  if (k == 6)  {  system("cls");  cout << "Добавление элемента в дерево" << endl;  cout << "Введите имя бинарного файла" << endl;  cin >> bin\_file;  ofstream outp;  outp.open(bin\_file, ios::binary | ios::app);  cout << "Введите английское слово ";  cin >> temp.eng\_word;  cout << "Введите русское слово(английский язык) ";  cin >> temp.rus\_word;  if (outp.good())  {  BinFile::AddToBin(outp, temp);  root = node::insert(root, temp.eng\_word, i);  i++;  }  else  {  cout << "Такого файла не существует" << endl;  }  outp.close();  system("pause");  }  if (k == 7)  {  system("cls");  cout << "Поиск по ключу в дереве" << endl;  cout << "Введите имя бинарного файла" << endl;  cin >> bin\_file;  ofstream outp;  outp.open(bin\_file, ios::binary | ios::app);  if (outp.good())  {  cout << "Введите ключ(английское слово) ";  cin >> key;  adress = node::search(root, key);  if (adress!=-1)  {  temp = BinFile::get\_class(bin\_file, adress);  cout << temp.eng\_word << " " << temp.rus\_word;  }  else  {  cout << "Запись с таким ключом не найдена" << endl;  }  }  else  {  cout << "такого файла не существует" << endl;  }  outp.close();  system("pause");  }  if (k == 8)  {  system("cls");  cout << "Удаление элемента из дерева" << endl;  cout << "Введите ключ" << endl;  cin >> key;  if (node::search(root, key) != -1)  {  root = node::deleteNode(root, key);  cout << "Успешно удалено" << endl;  }  else  {  cout << "Запись с таким ключом не найдена" << endl;  }  system("pause");  }  if (k == 9)  {  system("cls");  cout << "Вывод дерева" << endl;  cout << "Вывод ключей дерева" << endl;  node::printBT("", root, false, 1);  cout << "Вывод значений дерева" << endl;  node::printBT("", root, false, 0);  system("pause");  }  if (k == 10)  {  system("cls");  cout << "Построение B-дерева по файлу" << endl;  cout << "Введите имя бинарного файла" << endl;  cin >> bin\_file;  ifstream inp;  i = 0;  inp.open(bin\_file, ios::binary | ios::app);  if (inp.good())  {  inp.close();  words = BinFile::getwords(bin\_file);  while (words[i] != "\0")  {  t.insert(words[i], i);  i++;  }  cout << "Успешно" << endl;  }  else  {  cout << "Файл не существует" << endl;  }  system("pause");  }  if (k == 11)  {  system("cls");  cout << "Добавление элемента в дерево" << endl;  cout << "Введите имя бинарного файла" << endl;  cin >> bin\_file;  ofstream outp;  outp.open(bin\_file, ios::binary |ios::app);  cout << "Введите английское слово ";  cin >> temp.eng\_word;  cout << "Введите русское слово(английский язык) ";  cin >> temp.rus\_word;  if (outp.good())  {  BinFile::AddToBin(outp, temp);  t.insert(temp.eng\_word, i);  i++;  }  else  {  cout << "Такого файла не существует" << endl;  }  outp.close();  system("pause");  }  if (k == 12)  {  system("cls");  cout << "Поиск по ключу в дереве" << endl;  cout << "Введите имя бинарного файла" << endl;  cin >> bin\_file;  ofstream outp;  outp.open(bin\_file, ios::binary | ios::app);  if (outp.good())  {  cout << "Введите ключ(английское слово) ";  cin >> key;  adress = t.search(key);  if (adress != NULL)  {  temp = BinFile::get\_class(bin\_file, adress);  cout << temp.eng\_word << " " << temp.rus\_word;  }  else  {  cout << "Запись с таким ключом не найдена" << endl;  }  }  else  {  cout << "такого файла не существует" << endl;  }  outp.close();  system("pause");  }  if (k == 13)  {  system("cls");  cout << "Удаление элемента из дерева" << endl;  cout << "Введите ключ" << endl;  cin >> key;  if (t.search(key) != NULL)  {  t.deletion(key);  cout << "Успешно удалено" << endl;  }  else  {  cout << "Запись с таким ключом не найдена" << endl;  }  system("pause");  }  if (k == 14)  {  system("cls");  cout << "Вывод дерева" << endl;  t.traverse();  system("pause");  }  if (k < 1 || k>14)  {  cout << "Вы уверены, что хотите выйти? Y/N" << endl;  char j;  cin >> j;  if (j == 'Y')  {  cout << "Выход... ";  system("pause");  exit(0);  }  }  }  } |

# ВЫВОД

В результате выполнения работы я:

1. Получить навыки в разработки и реализации алгоритмов управления бинарным деревом поиска и сбалансированными бинарными деревьями поиска (АВЛ – деревьями);

2. Получила навыки в применении файловых потоков прямого доступа к данным файла;

3. Получила навыки в применении сбалансированного дерева поиска для прямого доступа к записям файла.