|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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-03-21 | Хречко С.В. |
| Принял преподаватель | Филатов А.С. |

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

Москва 2022

# **Цель работы**

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.

# **Постановка задачи**

Разработать программу, которая продемонстрирует выполнение упражнений варианта. Результаты выполнения упражнения выводить на монитор.

**Задание 1.**

1. Разработать класс управления файлом. Включить методы: создание двоичного файла записей фиксированной длины из заранее подготовленных данных в текстовом файле; поиск и вывод записи в файле с помощью линейного поиска; вывод записи по известному адресу в файле, добавление записи в файл. Структура записи файла определена индивидуальным вариантом задания.

2. Разработать класс «Бинарное дерево поиска». Структура информационной части узла дерева включает ключ и ссылку на запись в файле. Разработать основные методы: построение дерева по существующему файлу; добавление элемента в дерево; поиск по ключу в дереве; удаление элемента из дерева; вывод дерева в форме дерева.

3. Разработать класс «Сбалансированное дерево поиска». Структура информационной части узла дерева включает ключ и ссылку на запись в файле. Разработать основные методы: построение дерева по существующему файлу; добавление элемента в дерево; поиск по ключу в дереве; удаление элемента из дерева; вывод дерева в форме дерева.

4. Разработать приложение, демонстрирующее выполнение всех методов и поиск записей в файле с помощью БДП, СДП и линейного поиска.

5. Провести анализ времени выполнения поиска записей, находящихся в начале, середине и конце большого файла тремя способами.

6. Определить среднее число выполненных поворотов (число поворотов на общее число вставленных ключей) при включении ключей в СДП при его формировании из двоичного файла.

7. Выполнить тестирование.

8. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

Вариант № 4. Условие задания:

|  |  |
| --- | --- |
| Структура записи | Справочник банков по городам страны. Об отдельном банке хранятся данные: наименование, код банка, адрес (город), форма собственности (коммерческий или государственный). |
| Тип СДП | Рандомизированное |

# **Решение**

Бинарное дерево поиска — дерево в котором узлы располагаются таким образом, что каждый узел с меньшим значением (относительно родителя) находится в левой части дерева, соответственно с большим — в правой.

Дерево поиска с минимальной высотой называется сбалансированным, т.е. таким, в котором высота левого и правого поддеревьев отличаются не более чем на единицу. Сбалансированное оно, потому что в таком дереве в среднем требуется наименьшее количество операций для поиска.

## **Работа бинарного дерева поиска**

По условию задания требовалось разработать метод добавления элемента дерево. Метод находит место, в которое должен быть вставлен новый элемент, путем рекурсивного прохода по дереву, аналогично поиску, после чего создает новый объект и вставляет его в дерево.

|  |
| --- |
| void BinarySearchTree::addNode(int id, int index){      if(id < this->id){          if(this->left){              this->left->addNode(id, index);          }else{              this->left = new BinarySearchTree(id, index);              this->left->setParent(this);          }      }else{          if(this->right){              this->right->addNode(id, index);          }else{              this->right = new BinarySearchTree(id, index);              this->right->setParent(this);          }      }  } |

По условию задания требовалось разработать метод нахождения хранимого в дереве значения (индекса) по идентификатору. Метод рекурсивно идет по дереву, сравнивая переданный идентификатор с хранимыми в узле и поддеревьях. Меньшие чем в узле идентификаторы расположены в левом поддереве, большие – в правом.

|  |
| --- |
| int BinarySearchTree::findNodeIndex(int id){      if(id == this->id){          return this->index;      }      if(id < this->id){          if(this->left == nullptr){              return -1;          }          else{              return this->left->findNodeIndex(id);          }      }      else{          if(this->right == nullptr){              return -1;          }          else{              return this->right->findNodeIndex(id);          }      }  } |

По условию задания требовалось разработать метод удаления элемента. Метод находит искомый элемент в дереве, затем производит удаление элемента, вместе с тем просчитывает, как расположены элементы в дереве, чтобы после удаления дерево осталось бинарным деревом поиска.

|  |
| --- |
| void BinarySearchTree::deleteNode(int id){      if(id == this->id){          if(this->left == nullptr && this->right == nullptr){              if(this->parent){                  if(this->parent->getLeft() == this){                      this->parent->setLeft(nullptr);                  }                  else{                      this->parent->setRight(nullptr);                  }              }              delete this;          }          else if(this->left == nullptr){              if(this->parent){                  if(this->parent->getLeft() == this){                      this->parent->setLeft(this->right);                      this->right->setParent(this->parent);                  }                  else{                      this->parent->setRight(this->right);                      this->right->setParent(this->parent);                  }              }              this->right = nullptr;              delete this;          }          else if(this->right == nullptr){              if(this->parent){                  if(this->parent->getLeft() == this){                      this->parent->setLeft(this->left);                      this->left->setParent(this->parent);                  }                  else{                      this->parent->setRight(this->left);                      this->left->setParent(this->parent);                  }              }              this->left = nullptr;              delete this;          }          else{              BinarySearchTree \*temp = this->right;              while(temp->getLeft()){                  temp = temp->getLeft();              }              this->id = temp->id;              this->index = temp->index;              temp->deleteNode(temp->id);          }      }      else if(id < this->id){          if(this->left == nullptr){              return;          }          else{              this->left->deleteNode(id);          }      }      else{          if(this->right == nullptr){              return;          }          else{              this->right->deleteNode(id);          }      }  } |

По условию задания требовалось разработать метод для вывода дерева. Метод рекурсивно проходит по дереву, храня глубину, чтобы вывод был отформатирован в удобном для чтения формате.

|  |
| --- |
| void BinarySearchTree::printTree(int depth){      if(this->right){          this->right->printTree(depth + 1);      }      for(int i = 0; i < depth; i++){          cout << "    ";      }      cout << this->id << endl;      if(this->left){          this->left->printTree(depth + 1);      }  } |

## **Работа рандомизированного бинарного дерева поиска**

По условию задания требовалось разработать метод добавления элемента дерево. Метод в некоторых случаях вызывает метод добавления элемента в корень, а в некоторых добавляет элемент обычным образом. Выбор случаен и зависит от размера дерева, который хранится.

|  |
| --- |
| RandomizedBinarySearchTree \*RandomizedBinarySearchTree::addNode(RandomizedBinarySearchTree \*root, int id, int index, long long &countTurns){      if (!root){          return new RandomizedBinarySearchTree(id, index);      }      if (rand() % (root->getSize() + 1) == 0){          return addNodeToRoot(root, id, index, countTurns);      }      if (id < root->getId()){          root->setLeft(addNode(root->getLeft(), id, index, countTurns));      }      else{          root->setRight(addNode(root->getRight(), id, index, countTurns));      }      fixSize(root);      return root;  } |

Для работы метода добавления элемента требуется метод добавления элемента в корень дерева. Метод создает объект и поднимает его до корня, вызывая для этого методы поворота.

|  |
| --- |
| RandomizedBinarySearchTree \*RandomizedBinarySearchTree::addNodeToRoot(RandomizedBinarySearchTree \*root, int id, int index, long long &countTurns){      if (!root){          return new RandomizedBinarySearchTree(id, index);      }      if (id < root->getId()){          root->setLeft(addNodeToRoot(root->getLeft(), id, index, countTurns));          root = rotateRight(root);          countTurns++;      }      else{          root->setRight(addNodeToRoot(root->getRight(), id, index, countTurns));          root = rotateLeft(root);          countTurns++;      }      //fixsize is done when we rotate      return root;  } |

Для работы метода добавления элемента требуются методы поворота влево и вправо. Поворот влево, делает правого потомка корнем, а поворот вправо – левого. Корень вместо правого/левого потомка, который стал новым корнем, после поворота ссылается на левое/правое поддерево нового корня, который больше на него не ссылается, таким образом узел в дереве меняется местами с одним из своих потомков, дерево при этом остается бинарным. Метод fixsize просчитывает размер дерева для затронутых узлов, чтобы не было нарушений.

|  |
| --- |
| RandomizedBinarySearchTree \*RandomizedBinarySearchTree::rotateLeft(RandomizedBinarySearchTree \*root){      RandomizedBinarySearchTree \*newRoot = root->getRight();      root->setRight(newRoot->getLeft());      newRoot->setLeft(root);      fixSize(root);      fixSize(newRoot);      return newRoot;  }  RandomizedBinarySearchTree \*RandomizedBinarySearchTree::rotateRight(RandomizedBinarySearchTree \*root){      RandomizedBinarySearchTree \*newRoot = root->getLeft();      root->setLeft(newRoot->getRight());      newRoot->setRight(root);      fixSize(root);      fixSize(newRoot);      return newRoot;  } |

По условию задания требовалось разработать метод нахождения хранимого в дереве значения (индекса) по идентификатору. Метод рекурсивно идет по дереву, сравнивая переданный идентификатор с хранимыми в узле и поддеревьях. Меньшие чем в узле идентификаторы расположены в левом поддереве, большие – в правом.

|  |
| --- |
| int RandomizedBinarySearchTree::findNodeIndex(int id){      if (this->id == id){          return this->index;      }      if (this->id > id){          if (this->left){              return this->left->findNodeIndex(id);          }          else{              return -1;          }      }      else{          if (this->right){              return this->right->findNodeIndex(id);          }          else{              return -1;          }      }  } |

По условию задания требовалось разработать метод удаления элемента из дерева. Метод ищет элемент с переданным идентификатором, затем вызывает метод слияния для левого и правого поддерева удаляемого элемента, а сам элемент удаляет.

|  |
| --- |
| RandomizedBinarySearchTree \*RandomizedBinarySearchTree::deleteNode(RandomizedBinarySearchTree \*root, int id){      if (!root) return root;      if (root->getId() == id){          RandomizedBinarySearchTree \*newRoot = merge(root->getLeft(), root->getRight());          root->setLeft(nullptr);          root->setRight(nullptr);          delete root;          return newRoot;      }      if (root->getId() > id){          root->setLeft(deleteNode(root->getLeft(), id));      }      else{          root->setRight(deleteNode(root->getRight(), id));      }      return root;  } |

Для работы метода удаления элемента требуется метод слияния. Метод принимает указатели на два дерева, которые необходимо слить воедино. Метод случайным образом выбирает, корень какого поддерева станет новым корнем всего дерева, вероятность зависит от размеров деревьев. Далее метод рекурсивно проводит слияния для поддеревьев, чтобы общее дерево осталось бинарным.

|  |
| --- |
| RandomizedBinarySearchTree \*RandomizedBinarySearchTree::merge(RandomizedBinarySearchTree \*left, RandomizedBinarySearchTree \*right){      if (!left){          return right;      }      if (!right){          return left;      }      if (rand() % (left->getSize() + right->getSize()) < left->getSize()){          left->setRight(merge(left->getRight(), right));          fixSize(left);          return left;      }      else{          right->setLeft(merge(left, right->getLeft()));          fixSize(right);          return right;      }  } |

По условию задания требовалось разработать метод для вывода дерева. Метод рекурсивно проходит по дереву, храня глубину, чтобы вывод был отформатирован в удобном для чтения формате.

|  |
| --- |
| void RandomizedBinarySearchTree::printTree(int depth){      if(this->right){          this->right->printTree(depth + 1);      }      for(int i = 0; i < depth; i++){          cout << "    ";      }      cout << this->id << endl;      if(this->left){          this->left->printTree(depth + 1);      }  } |

## **Создание деревьев из файлов**

По условию задания требовалось создавать деревья поиска, из данных в бинарном файле. Для бинарного дерева, функция открывает файл, проверяет его открытие, затем создает дерево и добавляет последовательно узлы, основываясь на считанных из файла данных.

|  |
| --- |
| BinarySearchTree \*createBinaryTreeOutOfFile(string fileNameBin){      ifstream fin("../../" + fileNameBin, ios::binary);      if (!fin.is\_open()){          cout << "Error opening file " << fileNameBin << endl;          fin.close();          return nullptr;      }      Bank bank;      fin.read((char\*)&bank, sizeof(Bank));      BinarySearchTree \*tree = new BinarySearchTree(bank.id, 0);      int index = 1;      while (fin.good()){          fin.read((char\*)&bank, sizeof(Bank));          if (fin.good()){              tree->addNode(bank.id, index);  index++;          }      }      fin.close();      return tree;  } |

По условию задания требовалось создавать деревья поиска, из данных в бинарном файле. Функция для рандомизированного дерева практически идентична таковой для бинарного дерева. Единственное значимое различие – это подсчет среднего количества поворот, как требовалось по заданию.

|  |
| --- |
| RandomizedBinarySearchTree \*createRandomizedTreeOutOfFile(string fileNameBin, double &averageTurns){      ifstream fin("../../" + fileNameBin, ios::binary);      if (!fin.is\_open()){          cout << "Error opening file " << fileNameBin << endl;          fin.close();          return nullptr;      }      Bank bank;      fin.read((char\*)&bank, sizeof(Bank));      RandomizedBinarySearchTree \*tree = new RandomizedBinarySearchTree(bank.id, 0);      int index = 1;      long long countTurns = 0;      while (fin.good()){          fin.read((char\*)&bank, sizeof(Bank));          if (fin.good()){              tree = RandomizedBinarySearchTree::addNode(tree, bank.id, index, countTurns);  index++;          countTurns++;  }      }      averageTurns = (double)countTurns / (double)index;      fin.close();      return tree;  } |

Интерфейс программы предлагает пользователю протестировать деревья, а затем ввести файлы для тестирования поиска в них с помощью деревьев (рис 1).

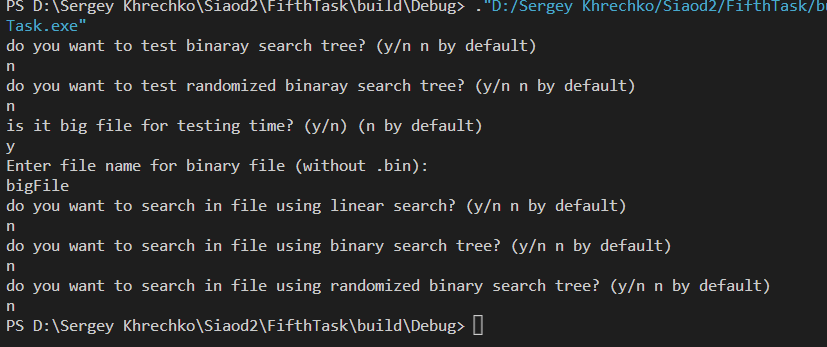


Рисунок 1. Интерфейс программы

# **Тестирование**

Первичное тестирование проводится на заготовленных данных, без вноса данных с клавиатуры. Продвинутое тестирование использует файлы и будет приведено после первичного тестирования обоих деревьев.

**Тестирование бинарного дерева**

Тестирование создания и вывода на печать бинарного дерева приведено на рисунке 3. Данные использованные для тестирования приведены на рисунке 2. Как можно наблюдать, все элементы были успешно добавлены, и дерево является бинарным деревом поиска.

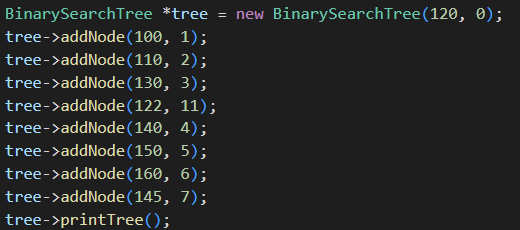


Рисунок 2. Данные для тестирования

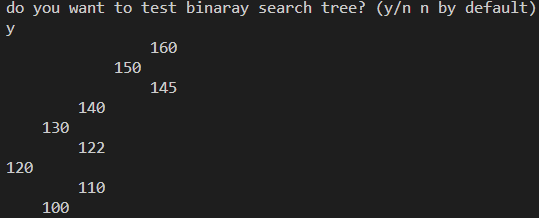


Рисунок 3. Тестирование создания и вывода дерева

Тестирование поиска элемента в дереве. Поиск существующего элемента приведен на рисунке 4, а неудачный - на рисунке 5. Индекс равный -1 означает, что данный элемент не был найден.

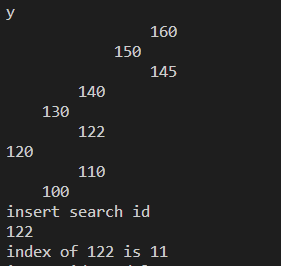


Рисунок 4. Тестирование поиска

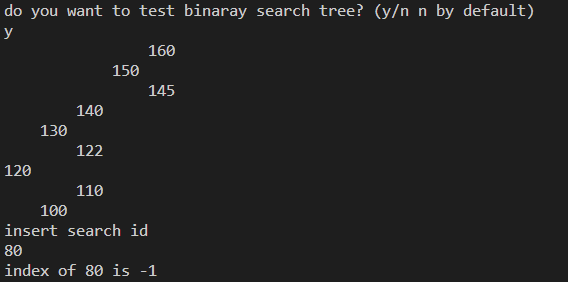


Рисунок 5. Тестирование поиска

Тестирование удаления элемента. Тестирование приведено на рисунках 6 и 7. При удалении несуществующего элемента дерево не было изменено, как и ожидалось.

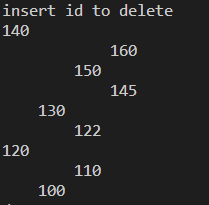


Рисунок 6. Тестирование удаления

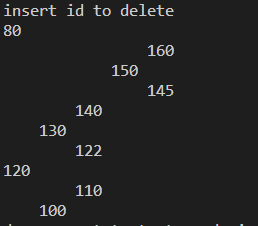


Рисунок 7. Тестирование удаления

**Тестирование бинарного дерева**

Тестирование создания и вывода на печать рандомизированного бинарного дерева приведено на рисунке 9. Данные использованные для тестирования приведены на рисунке 8. Как можно наблюдать, все элементы были успешно добавлены, и дерево является бинарным деревом поиска.

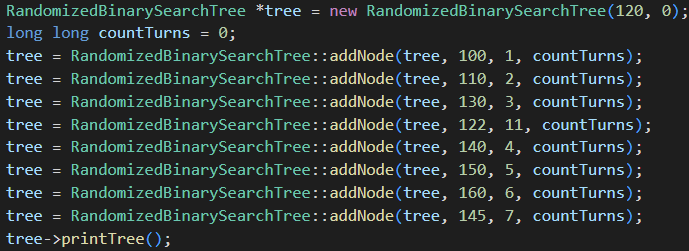


Рисунок 8. Данные для тестирования

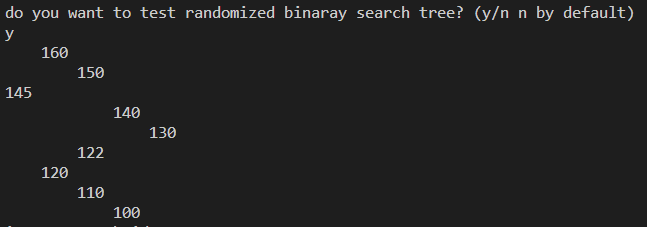


Рисунок 9. Тестирование создания и вывода дерева

Тестирование поиска элемента в дереве. Поиск существующего элемента приведен на рисунке 10, а неудачный - на рисунке 11. Индекс равный -1 означает, что данный элемент не был найден. Также можно заметить, что на рисунке 11 представлено другое дерево из тех же элементов, это происходит в силу случайности, использующейся при создании рандомизированного дерева.

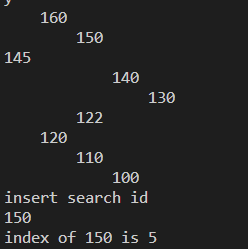


Рисунок 10. Тестирование поиска

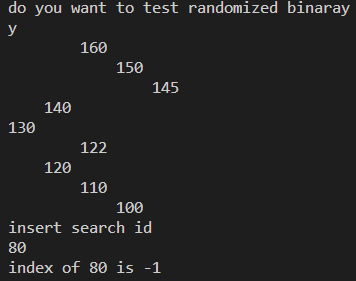


Рисунок 11. Тестирование поиска

Тестирование удаления элемента. Тестирование приведено на рисунках 12 и 13. При удалении несуществующего элемента дерево не было изменено, как и ожидалось.

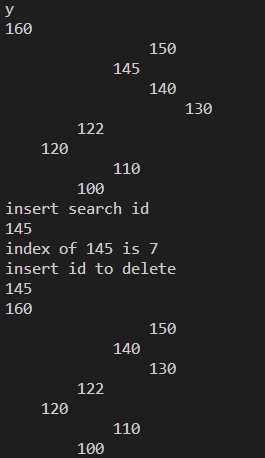


Рисунок 12. Тестирование удаления

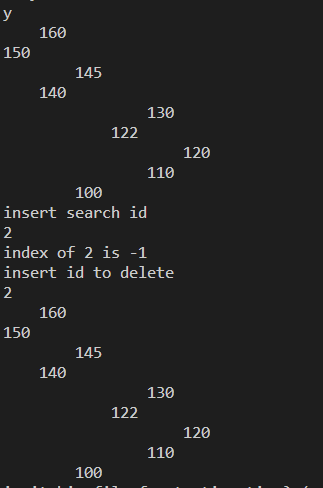


Рисунок 13. Тестирование удаления

**Работа с файлами и тестирование производительности**

Тестирование поиска в файле с помощью бинарного дерева поиска приведено на рисунке 14. Так как поиск успешно выполняется, то значит и дерево успешно построено.

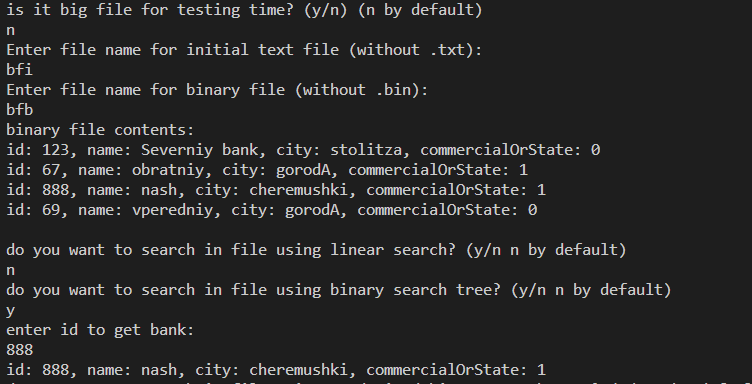


Рисунок 14. Поиск в файле

Тестирование поиска в файле с помощью рандомизированного бинарного дерева поиска приведено на рисунке 15. Тестирование выполнено аналогично тестированию для бинарного дерева. Также получено среднее значение поворотов.

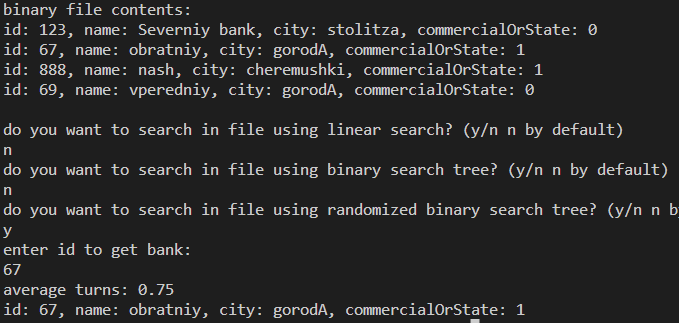


Рисунок 15. Поиск в файле

Тестирование поиска в файле с помощью линейного поиска приведено на рисунке 16. Используется линейный поиск, разработанный в 3-ей практической работе.

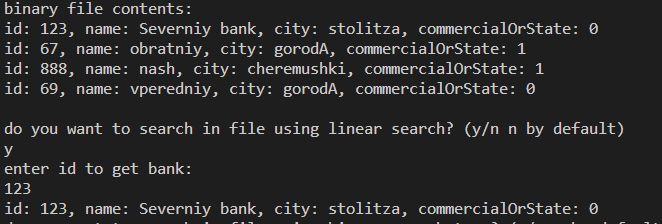


Рисунок 16. Поиск в файле

Сравнение временной эффективности линейного поиска, бинарного дереве и рандомизированного дерева.

Поиск в большом файле линейным поиском в начале, в середине и в конце приведен на рисунке 17. Мы можем наблюдать линейное увеличение временных затрат.

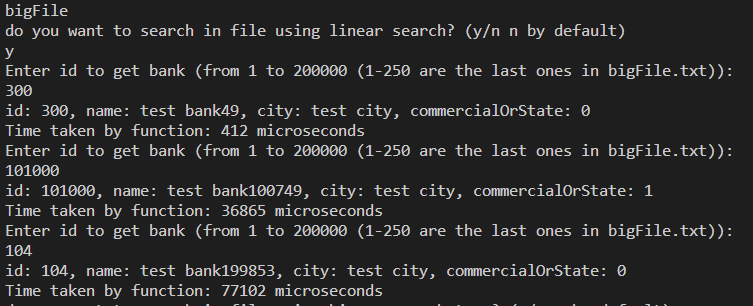


Рисунок 17. Линейный поиск в большом файле

Поиск в большом файле с помощью рандомизированного бинарного дерева в начале, в середине и в конце приведен на рисунке 18. Мы можем наблюдать, что на вне зависимости от положения записи в файле поиск происходит примерно за одно и то же небольшое время .

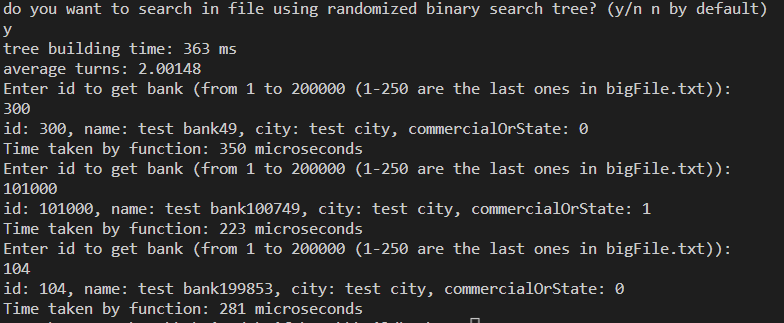


Рисунок 18. Поиск в большом файле рандомизированным деревом

Поиск в большом файле с помощью бинарного дерева привел к ошибке. Ошибка возникла из-за крайне плохих для несбалансированного бинарного дерева входных данных, в файле находится множество упорядоченных возрастающих идентификаторов, из-за этого дерево вырождается в список, по которому проход осуществляется рекурсией, глубина данной рекурсии, может легко быть превышена на больших входных данных. Это является одной из важнейших проблем при работе с несбалансированным бинарным деревом.

Для хотя бы какого-нибудь сравнения всех трех алгоритмов на одинаковых данных, был создан файл на 8000 элементов, в котором сначала идет 4000 элементов от 4000 до 7999, а затем 4000 элементов от 0 до 3999, результаты тестирования всех трех функций представлены на рисунке 19. Можно заметить, что хоть бинарное дерево и быстрее линейного поиска в некоторых случаях, оно по прежнему проигрывает рандомизированному дереву, что подтверждает теорию.

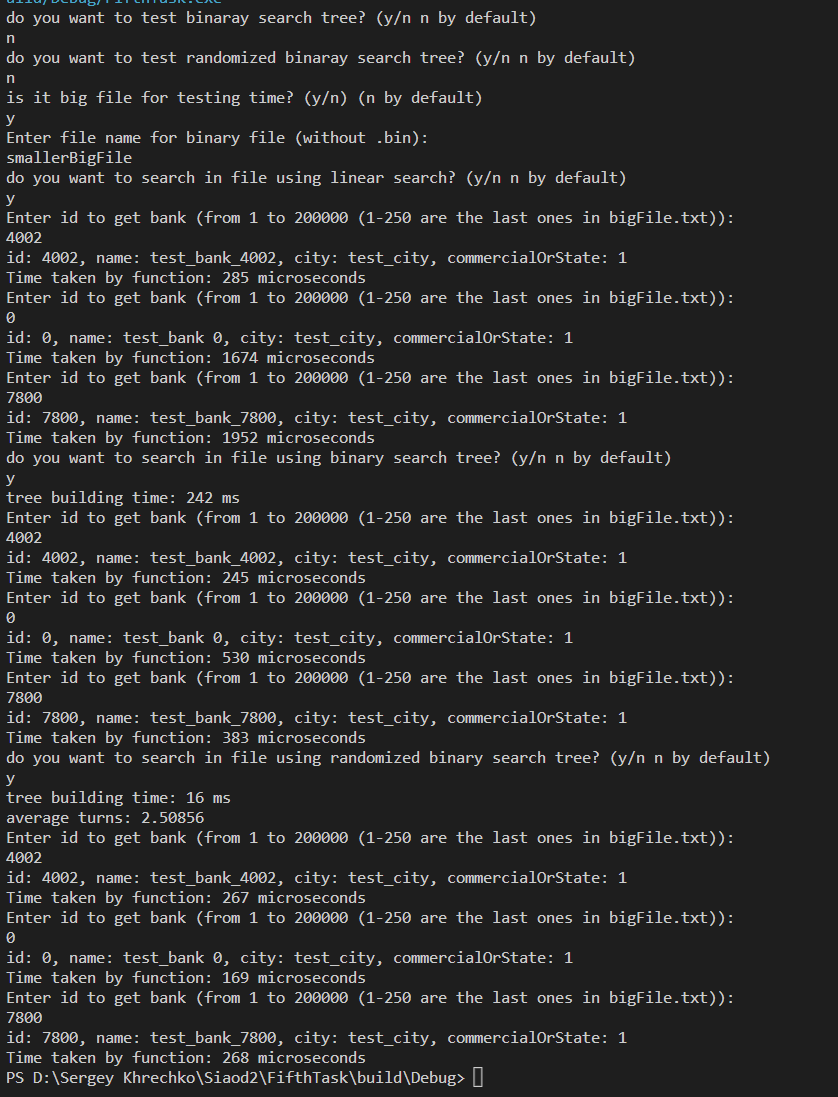


Рисунок 19. Сравнение времени поиска

Из результатов выполнения программы видно:

1. Программа работает корректно.
2. Программа выполняет все поставленные задачи.

# **Вывод**

В результате выполнения работы я:

1. Освоил алгоритмы работы с бинарными деревьями поиска.
2. Научился создавать рандомизированное бинарное дерево поиска.

# **Исходный код программы**

|  |
| --- |
| #ifndef \_\_BINFILE\_H\_\_  #define \_\_BINFILE\_H\_\_  #include <iostream>  #include <fstream>  #include <time.h>  #include <string>  #include <vector>  #include <cstring>  using namespace std;  struct Bank{      int id;      char name[61];      char city[61];      bool commercialOrState; //true - commercial, false - state  };  int toBinary(string fileName, string fileNameBin);//read from text file and write to binary file  int getBank(string fileNameBin, Bank &bank, int ind);//get bank by index from binary file  int linearSearch(string fileNameBin, Bank &bank, int id);  int deleteBankById(string fileNameBin, int id);  int deleteBankByIndex(string fileNameBin, int ind);  int addBank(string fileNameBin, const Bank &bank, int ind = -1);  void testBinF();  #endif  #include "binFile.h"  int toBinary(string fileName, string fileNameBin){//read from text file and write to binary file      ifstream fin("../../" + fileName);      fstream fout("../../" + fileNameBin, ios::out|ios::binary|ios::trunc);      if(!fin.is\_open() || !fout.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          fout.close();          return 1;      }      while(fin.good()){          Bank bank;          vector<string> tokens;          for (string each; getline(fin, each, ';'); tokens.push\_back(each));          for(int i = 0; i < tokens.size(); i++){              //cout << tokens.size() << " " << tokens[i] << endl;              if(i%4 == 0) bank.id = stoi(tokens[i]);              else if(i%4 == 1) strcpy(bank.name, tokens[i].c\_str());              else if(i%4 == 2) strcpy(bank.city, tokens[i].c\_str());              else if(i%4 == 3){                  bank.commercialOrState = (tokens[i] == "true") ? true : false;                  fout.write((char\*)&bank, sizeof(bank));              }          }        }      fin.close();      fout.close();      return 0;  }  int addBank(string fileNameBin, const Bank &bank, int ind){      fstream fout("../../" + fileNameBin, ios::in|ios::out|ios::binary);      if(!fout.is\_open()){          cout << "file doesn't exist" << endl;          fout.close();          return 1;      }      if(ind == -1){          fout.seekp(0, ios::end);          fout.write((char\*)&bank, sizeof(bank));          fout.close();          return 0;      }      fout.seekp((ind)\*sizeof(bank), ios::beg);      // if(fout.write((char\*)&bank, sizeof(bank))){      //  fout.close();      //  return 0;      // }      fout.write((char\*)&bank, sizeof(bank));      fout.close();      return 0;  }  int getBank(string fileNameBin, Bank &bank, int ind){//get bank by index from binary file starting from 1      //returns 0 if bank was found, 1 if file doesn't exist, 2 if bank wasn't found      fstream fin("../../" + fileNameBin, ios::in|ios::binary);      if(!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return 1;      }        fin.seekg(sizeof(Bank)\*(ind), ios::beg);      Bank readBank;      fin.read((char\*)&readBank, sizeof(Bank));      if(!fin.good()){          return 2;      }      bank = readBank;      fin.close();      return 0;  }  int linearSearch(string fileNameBin, Bank &bank, int id){      fstream fin("../../" + fileNameBin, ios::in|ios::binary);      if(!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return 1;      }      int ind = 0;      while(fin.good()){          Bank readBank;          fin.read((char\*)&readBank, sizeof(Bank));          if(readBank.id == id){              bank = readBank;              fin.close();              return ind;          }          ind++;      }      fin.close();      return -1;  }  int deleteBankById(string fileNameBin, int id){//delete bank by id from binary file      //returns 0 if bank was deleted, 1 if file doesn't exist, 2 if bank wasn't found      fstream fin("../../" + fileNameBin, ios::in|ios::out|ios::binary);      if(!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return 1;      }      Bank bank;      fin.seekg(0, ios::beg);      while(fin.read((char\*)&bank, sizeof(bank))){          if(bank.id == id){              bank.id = -1; //means that bank was deleted              fin.seekp(-sizeof(bank), ios::cur);              fin.write((char\*)&bank, sizeof(bank));              fin.close();              return 0;          }      }      fin.close();      return 2;  }  int deleteBankByIndex(string fileNameBin, int ind){      //returns 0 if bank was deleted, 1 if file doesn't exist, 2 if bank wasn't found      fstream fin("../../" + fileNameBin, ios::in|ios::out|ios::binary);      if(!fin.is\_open()){          cout << "file doesn't exist" << endl;          fin.close();          return 1;      }      fin.seekg(sizeof(Bank)\*(ind), ios::beg);      Bank readBank;      fin.read((char\*)&readBank, sizeof(Bank));      if(!fin.good()){          return 2;      }      readBank.id = -1; //means that bank was deleted      fin.seekp(-sizeof(readBank), ios::cur);      fin.write((char\*)&readBank, sizeof(readBank));      fin.close();      return 0;  }  void testBinF(){      //test toBinary      toBinary("bfi.txt", "bfb.txt");      cout << "binary file contents:" << endl;      for(int i = 1; i <= 4; i++){          Bank bank;          getBank("bfb.txt", bank, i-1);          cout << bank.id << " " << bank.name << " " << bank.city << " " << bank.commercialOrState << endl;      }      //test getBank      cout << "insert index from 1 to pick bank" << endl;      int ind;      cin >> ind;      Bank bank;      getBank("bfb.txt", bank, ind-1);      cout << bank.id << " " << bank.name << " " << bank.city << " " << bank.commercialOrState << endl;      //test deleteBank by id      cout << "insert id to delete bank" << endl;      int id;      cin >> id;      deleteBankById("bfb.txt", id);      cout << "binary file contents:" << endl;      for(int i = 1; i <= 4; i++){          Bank bank;          getBank("bfb.txt", bank, i-1);          cout << bank.id << " " << bank.name << " " << bank.city << " " << bank.commercialOrState << endl;      }      //test deleteBank by index      cout << "insert index to delete bank" << endl;      cin >> ind;      deleteBankByIndex("bfb.txt", ind-1);      cout << "binary file contents:" << endl;      for(int i = 1; i <= 4; i++){          Bank bank;          getBank("bfb.txt", bank, i-1);          cout << bank.id << " " << bank.name << " " << bank.city << " " << bank.commercialOrState << endl;      }      //test adding      bank.id = 1337;      string tmpName = "test bank";      string tmpCity = "test city";      strcpy(bank.name, tmpName.c\_str());      strcpy(bank.city, tmpCity.c\_str());      bank.commercialOrState = true;      cout << endl << "bank that is being added " << bank.id << ";" << bank.name << ";" << bank.city << ";" << bank.commercialOrState << endl;      cout << "adding in the end and to pos 2" << endl;      addBank("bfb.txt", bank, 2-1);      addBank("bfb.txt", bank);      cout << "binary file contents:" << endl;      for(int i = 1; i <= 5; i++){          Bank bank;          getBank("bfb.txt", bank, i-1);          cout << bank.id << " " << bank.name << " " << bank.city << " " << bank.commercialOrState << endl;      }  }  #ifndef \_\_BINARYSEARCHTREE\_H\_\_  #define \_\_BINARYSEARCHTREE\_H\_\_  #include <iostream>  #include <fstream>  #include <time.h>  #include <string>  #include <cstring>  using namespace std;  //binary search tree class for int  class BinarySearchTree{      private:          int id;          int index;          BinarySearchTree \*parent;          BinarySearchTree \*left;          BinarySearchTree \*right;      public:          BinarySearchTree(int id, int index);          ~BinarySearchTree();           void setParent(BinarySearchTree \*parent);          void setLeft(BinarySearchTree \*left);          void setRight(BinarySearchTree \*right);          BinarySearchTree \*getParent();          BinarySearchTree \*getLeft();          BinarySearchTree \*getRight();          void addNode(int id, int index);          int findNodeIndex(int id);          void deleteNode(int id);          void printTree(int depth = 0);  };  #endif  #include "binarySearchTree.h"  BinarySearchTree::BinarySearchTree(int id, int index){      this->id = id;      this->index = index;      this->parent = nullptr;      this->left = nullptr;      this->right = nullptr;  }  BinarySearchTree::~BinarySearchTree(){      if(this->left){          delete this->left;      }      if(this->right){          delete this->right;      }  }  void BinarySearchTree::setParent(BinarySearchTree \*parent){      this->parent = parent;  }  void BinarySearchTree::setLeft(BinarySearchTree \*left){      this->left = left;  }  void BinarySearchTree::setRight(BinarySearchTree \*right){      this->right = right;  }  BinarySearchTree \*BinarySearchTree::getParent(){      return this->parent;  }  BinarySearchTree \*BinarySearchTree::getLeft(){      return this->left;  }  BinarySearchTree \*BinarySearchTree::getRight(){      return this->right;  }  void BinarySearchTree::addNode(int id, int index){      if(id < this->id){          if(this->left){              this->left->addNode(id, index);          }else{              this->left = new BinarySearchTree(id, index);              this->left->setParent(this);          }      }else{          if(this->right){              this->right->addNode(id, index);          }else{              this->right = new BinarySearchTree(id, index);              this->right->setParent(this);          }      }  }  int BinarySearchTree::findNodeIndex(int id){      if(id == this->id){          return this->index;      }      if(id < this->id){          if(this->left == nullptr){              return -1;          }          else{              return this->left->findNodeIndex(id);          }      }      else{          if(this->right == nullptr){              return -1;          }          else{              return this->right->findNodeIndex(id);          }      }  }  void BinarySearchTree::deleteNode(int id){      if(id == this->id){          if(this->left == nullptr && this->right == nullptr){              if(this->parent){                  if(this->parent->getLeft() == this){                      this->parent->setLeft(nullptr);                  }                  else{                      this->parent->setRight(nullptr);                  }              }              delete this;          }          else if(this->left == nullptr){              if(this->parent){                  if(this->parent->getLeft() == this){                      this->parent->setLeft(this->right);                      this->right->setParent(this->parent);                  }                  else{                      this->parent->setRight(this->right);                      this->right->setParent(this->parent);                  }              }              this->right = nullptr;              delete this;          }          else if(this->right == nullptr){              if(this->parent){                  if(this->parent->getLeft() == this){                      this->parent->setLeft(this->left);                      this->left->setParent(this->parent);                  }                  else{                      this->parent->setRight(this->left);                      this->left->setParent(this->parent);                  }              }              this->left = nullptr;              delete this;          }          else{              BinarySearchTree \*temp = this->right;              while(temp->getLeft()){                  temp = temp->getLeft();              }              this->id = temp->id;              this->index = temp->index;              temp->deleteNode(temp->id);          }      }      else if(id < this->id){          if(this->left == nullptr){              return;          }          else{              this->left->deleteNode(id);          }      }      else{          if(this->right == nullptr){              return;          }          else{              this->right->deleteNode(id);          }      }  }  void BinarySearchTree::printTree(int depth){      if(this->right){          this->right->printTree(depth + 1);      }      for(int i = 0; i < depth; i++){          cout << "    ";      }      cout << this->id << endl;      if(this->left){          this->left->printTree(depth + 1);      }  }  #ifndef \_\_RANDOMIZEDBINARYSEARCHTREE\_H\_\_  #define \_\_RANDOMIZEDBINARYSEARCHTREE\_H\_\_  #include <iostream>  #include <fstream>  #include <time.h>  #include <random>  #include <ctime>  #include <string>  #include <cstring>  using namespace std;  class RandomizedBinarySearchTree{  private:      int id;      int index;      int size;      RandomizedBinarySearchTree \*left;      RandomizedBinarySearchTree \*right;  public:      RandomizedBinarySearchTree(int id, int index);      ~RandomizedBinarySearchTree();      void setLeft(RandomizedBinarySearchTree \*left);      void setRight(RandomizedBinarySearchTree \*right);      void setSize(int size);      void setId(int id);      RandomizedBinarySearchTree \*getLeft();      RandomizedBinarySearchTree \*getRight();      int getSize();      int getId();      static void fixSize(RandomizedBinarySearchTree \*node);      static RandomizedBinarySearchTree \*rotateLeft(RandomizedBinarySearchTree \*root);      static RandomizedBinarySearchTree \*rotateRight(RandomizedBinarySearchTree \*root);      static RandomizedBinarySearchTree \*addNodeToRoot(RandomizedBinarySearchTree \*root, int id, int index, long long &countTurns);      static RandomizedBinarySearchTree \*addNode(RandomizedBinarySearchTree \*root, int id, int index, long long &countTurns);      int findNodeIndex(int id);      static RandomizedBinarySearchTree \*merge(RandomizedBinarySearchTree \*left, RandomizedBinarySearchTree \*right);      static RandomizedBinarySearchTree \*deleteNode(RandomizedBinarySearchTree \*root, int id);      void printTree(int depth = 0);  };  #endif  #include "randomizedBinarySearchTree.h"  RandomizedBinarySearchTree::RandomizedBinarySearchTree(int id, int index){      this->id = id;      this->index = index;      this->size = 1;      this->left = nullptr;      this->right = nullptr;  }  RandomizedBinarySearchTree::~RandomizedBinarySearchTree(){      if (this->left){          delete this->left;      }      if (this->right){          delete this->right;      }  }  void RandomizedBinarySearchTree::setLeft(RandomizedBinarySearchTree \*left){      this->left = left;  }  void RandomizedBinarySearchTree::setRight(RandomizedBinarySearchTree \*right){      this->right = right;  }  void RandomizedBinarySearchTree::setSize(int size){      this->size = size;  }  void RandomizedBinarySearchTree::setId(int id){      this->id = id;  }  RandomizedBinarySearchTree \*RandomizedBinarySearchTree::getLeft(){      return this->left;  }  RandomizedBinarySearchTree \*RandomizedBinarySearchTree::getRight(){      return this->right;  }  int RandomizedBinarySearchTree::getSize(){      return this->size;  }  int RandomizedBinarySearchTree::getId(){      return this->id;  }  void RandomizedBinarySearchTree::fixSize(RandomizedBinarySearchTree \*node){      if(node){          int leftSize = 0;          int rightSize = 0;          if (node->getLeft()){              leftSize = node->getLeft()->getSize();          }          if (node->getRight()){              rightSize = node->getRight()->getSize();          }          node->setSize(leftSize + rightSize + 1);      }  }  RandomizedBinarySearchTree \*RandomizedBinarySearchTree::rotateLeft(RandomizedBinarySearchTree \*root){      RandomizedBinarySearchTree \*newRoot = root->getRight();      root->setRight(newRoot->getLeft());      newRoot->setLeft(root);      fixSize(root);      fixSize(newRoot);      return newRoot;  }  RandomizedBinarySearchTree \*RandomizedBinarySearchTree::rotateRight(RandomizedBinarySearchTree \*root){      RandomizedBinarySearchTree \*newRoot = root->getLeft();      root->setLeft(newRoot->getRight());      newRoot->setRight(root);      fixSize(root);      fixSize(newRoot);      return newRoot;  }  RandomizedBinarySearchTree \*RandomizedBinarySearchTree::addNodeToRoot(RandomizedBinarySearchTree \*root, int id, int index, long long &countTurns){      if (!root){          return new RandomizedBinarySearchTree(id, index);      }      if (id < root->getId()){          root->setLeft(addNodeToRoot(root->getLeft(), id, index, countTurns));          root = rotateRight(root);          countTurns++;      }      else{          root->setRight(addNodeToRoot(root->getRight(), id, index, countTurns));          root = rotateLeft(root);          countTurns++;      }      //fixsize is done when we rotate      return root;  }  RandomizedBinarySearchTree \*RandomizedBinarySearchTree::addNode(RandomizedBinarySearchTree \*root, int id, int index, long long &countTurns){      if (!root){          return new RandomizedBinarySearchTree(id, index);      }      if (rand() % (root->getSize() + 1) == 0){          return addNodeToRoot(root, id, index, countTurns);      }      if (id < root->getId()){          root->setLeft(addNode(root->getLeft(), id, index, countTurns));      }      else{          root->setRight(addNode(root->getRight(), id, index, countTurns));      }      fixSize(root);      return root;  }  int RandomizedBinarySearchTree::findNodeIndex(int id){      if (this->id == id){          return this->index;      }      if (this->id > id){          if (this->left){              return this->left->findNodeIndex(id);          }          else{              return -1;          }      }      else{          if (this->right){              return this->right->findNodeIndex(id);          }          else{              return -1;          }      }  }  RandomizedBinarySearchTree \*RandomizedBinarySearchTree::merge(RandomizedBinarySearchTree \*left, RandomizedBinarySearchTree \*right){      if (!left){          return right;      }      if (!right){          return left;      }      if (rand() % (left->getSize() + right->getSize()) < left->getSize()){          left->setRight(merge(left->getRight(), right));          fixSize(left);          return left;      }      else{          right->setLeft(merge(left, right->getLeft()));          fixSize(right);          return right;      }  }  RandomizedBinarySearchTree \*RandomizedBinarySearchTree::deleteNode(RandomizedBinarySearchTree \*root, int id){      if (!root) return root;      if (root->getId() == id){          RandomizedBinarySearchTree \*newRoot = merge(root->getLeft(), root->getRight());          root->setLeft(nullptr);          root->setRight(nullptr);          delete root;          return newRoot;      }      if (root->getId() > id){          root->setLeft(deleteNode(root->getLeft(), id));      }      else{          root->setRight(deleteNode(root->getRight(), id));      }      return root;  }  void RandomizedBinarySearchTree::printTree(int depth){      if(this->right){          this->right->printTree(depth + 1);      }      for(int i = 0; i < depth; i++){          cout << "    ";      }      cout << this->id << endl;      if(this->left){          this->left->printTree(depth + 1);      }  }  #include <iostream>  #include <chrono>  #include "libraries/binFile.h"  #include "libraries/binarySearchTree.h"  #include "libraries/randomizedBinarySearchTree.h"  using namespace std;  BinarySearchTree \*createBinaryTreeOutOfFile(string fileNameBin){      ifstream fin("../../" + fileNameBin, ios::binary);      if (!fin.is\_open()){          cout << "Error opening file " << fileNameBin << endl;          fin.close();          return nullptr;      }      Bank bank;      fin.read((char\*)&bank, sizeof(Bank));      BinarySearchTree \*tree = new BinarySearchTree(bank.id, 0);      int index = 1;      while (fin.good()){          fin.read((char\*)&bank, sizeof(Bank));          if (fin.good()){              tree->addNode(bank.id, index);              index++;          }      }      fin.close();      return tree;  }  RandomizedBinarySearchTree \*createRandomizedTreeOutOfFile(string fileNameBin, double &averageTurns){      ifstream fin("../../" + fileNameBin, ios::binary);      if (!fin.is\_open()){          cout << "Error opening file " << fileNameBin << endl;          fin.close();          return nullptr;      }      Bank bank;      fin.read((char\*)&bank, sizeof(Bank));      RandomizedBinarySearchTree \*tree = new RandomizedBinarySearchTree(bank.id, 0);      int index = 1;      long long countTurns = 0;      while (fin.good()){          fin.read((char\*)&bank, sizeof(Bank));          if (fin.good()){              tree = RandomizedBinarySearchTree::addNode(tree, bank.id, index, countTurns);              index++;              countTurns++;          }      }      averageTurns = (double)countTurns / (double)index;      fin.close();      return tree;  }  int main() {      srand(time(NULL));      cout << "do you want to test binaray search tree? (y/n n by default)" << endl;      char c;      cin >> c;      if (c == 'y'){          BinarySearchTree \*tree = new BinarySearchTree(120, 0);          tree->addNode(100, 1);          tree->addNode(110, 2);          tree->addNode(130, 3);          tree->addNode(122, 11);          tree->addNode(140, 4);          tree->addNode(150, 5);          tree->addNode(160, 6);          tree->addNode(145, 7);          tree->printTree();          cout << "insert search id" << endl;          int id;          cin >> id;          cout << "index of " << id << " is " << tree->findNodeIndex(id) << endl;          cout << "insert id to delete" << endl;          cin >> id;          tree->deleteNode(id);          tree->printTree();          delete tree;      }      cout << "do you want to test randomized binaray search tree? (y/n n by default)" << endl;      cin >> c;      if (c == 'y'){          RandomizedBinarySearchTree \*tree = new RandomizedBinarySearchTree(120, 0);          long long countTurns = 0;          tree = RandomizedBinarySearchTree::addNode(tree, 100, 1, countTurns);          tree = RandomizedBinarySearchTree::addNode(tree, 110, 2, countTurns);          tree = RandomizedBinarySearchTree::addNode(tree, 130, 3, countTurns);          tree = RandomizedBinarySearchTree::addNode(tree, 122, 11, countTurns);          tree = RandomizedBinarySearchTree::addNode(tree, 140, 4, countTurns);          tree = RandomizedBinarySearchTree::addNode(tree, 150, 5, countTurns);          tree = RandomizedBinarySearchTree::addNode(tree, 160, 6, countTurns);          tree = RandomizedBinarySearchTree::addNode(tree, 145, 7, countTurns);          tree->printTree();          cout << "insert search id" << endl;          int id;          cin >> id;          cout << "index of " << id << " is " << tree->findNodeIndex(id) << endl;          cout << "insert id to delete" << endl;          cin >> id;          tree = RandomizedBinarySearchTree::deleteNode(tree, id);          tree->printTree();          delete tree;      }        cout << "is it big file for testing time? (y/n) (n by default)" << endl;      char bigFile;      cin >> bigFile;      cin.ignore(INT\_MAX, '\n');      if(bigFile != 'y' && bigFile != 'n') bigFile = 'n';      if(bigFile == 'n'){          cout << "Enter file name for initial text file (without .txt): " << endl;          string fileName;          getline(cin, fileName);          fileName += ".txt";          ifstream fin("../../" + fileName);          if(!fin.is\_open()){              cout << "Error opening file" << endl;              return 1;          }          fin.close();          cout << "Enter file name for binary file (without .bin): " << endl;          string fileNameBin;          getline(cin, fileNameBin);          fileNameBin += ".txt";          ifstream finBin("../../" + fileNameBin);          if(!finBin.is\_open()){              cout << "Error opening file" << endl;              return 1;          }          finBin.close();          toBinary(fileName, fileNameBin);          cout << "binary file contents: " << endl;          Bank bank;          int iter = 0;          while(getBank(fileNameBin, bank, iter) == 0){              //print bank              cout << "id: " << bank.id;              cout << ", name: " << bank.name;              cout << ", city: " << bank.city;              cout << ", commercialOrState: " << bank.commercialOrState << endl;              iter++;          }          cout << endl;          cout << "do you want to search in file using linear search? (y/n n by default)" << endl;          cin >> c;          if (c == 'y'){              cout << "enter id to get bank: " << endl;              int id;              cin >> id;              Bank bank;              bank.id = -1;              linearSearch(fileNameBin, bank, id);              if(bank.id == -1){                  cout << "no bank with such id" << endl;              }              else{                  cout << "id: " << bank.id;                  cout << ", name: " << bank.name;                  cout << ", city: " << bank.city;                  cout << ", commercialOrState: " << bank.commercialOrState << endl;              }          }            cout << "do you want to search in file using binary search tree? (y/n n by default)" << endl;          cin >> c;          if (c == 'y'){              cout << "enter id to get bank: " << endl;              int id;              cin >> id;              BinarySearchTree \*tree = createBinaryTreeOutOfFile(fileNameBin);              int index = tree->findNodeIndex(id);              Bank bank;              bank.id = -1;              getBank(fileNameBin, bank, index);              if(bank.id == -1){                  cout << "no bank with such id" << endl;              }              else{                  cout << "id: " << bank.id;                  cout << ", name: " << bank.name;                  cout << ", city: " << bank.city;                  cout << ", commercialOrState: " << bank.commercialOrState << endl;              }          }          cout << "do you want to search in file using randomized binary search tree? (y/n n by default)" << endl;          cin >> c;          if (c == 'y'){              cout << "enter id to get bank: " << endl;              int id;              cin >> id;              double averageTurns = 0;              RandomizedBinarySearchTree \*tree = createRandomizedTreeOutOfFile(fileNameBin, averageTurns);              cout << "average turns: " << averageTurns << endl;              int index = tree->findNodeIndex(id);              Bank bank;              bank.id = -1;              getBank(fileNameBin, bank, index);              if(bank.id == -1){                  cout << "no bank with such id" << endl;              }              else{                  cout << "id: " << bank.id;                  cout << ", name: " << bank.name;                  cout << ", city: " << bank.city;                  cout << ", commercialOrState: " << bank.commercialOrState << endl;              }          }        }      else{ //big file for testing time          cout << "Enter file name for binary file (without .bin): " << endl;          string fileNameBin;          getline(cin, fileNameBin);          fileNameBin += ".txt";          ifstream finBin("../../" + fileNameBin);          if(!finBin.is\_open()){              cout << "Error opening file" << endl;              return 1;          }          finBin.close();          cout << "do you want to search in file using linear search? (y/n n by default)" << endl;          cin >> c;          if(c == 'y'){              for(int i = 0; i < 3; i++){                  cout << "Enter id to get bank (from 1 to 200000 (1-250 are the last ones in bigFile.txt)): " << endl;                  int id;                  cin >> id;                  Bank bank;                  auto start = chrono::high\_resolution\_clock::now();                  linearSearch(fileNameBin, bank, id);                  auto end = chrono::high\_resolution\_clock::now();                  auto duration = chrono::duration\_cast<chrono::microseconds>(end - start);                  cout << "id: " << bank.id;                  cout << ", name: " << bank.name;                  cout << ", city: " << bank.city;                  cout << ", commercialOrState: " << bank.commercialOrState << endl;                  cout << "Time taken by function: " << duration.count() << " microseconds" << endl;              }          }          cout << "do you want to search in file using binary search tree? (y/n n by default)" << endl;          cin >> c;          if(c == 'y'){              auto treeBuildingStart = chrono::high\_resolution\_clock::now();              BinarySearchTree \*tree = createBinaryTreeOutOfFile(fileNameBin);              auto treeBuildingEnd = chrono::high\_resolution\_clock::now();              auto treeBuildingTime = chrono::duration\_cast<chrono::milliseconds>(treeBuildingEnd - treeBuildingStart);              cout << "tree building time: " << treeBuildingTime.count() << " ms" << endl;              for(int i = 0; i < 3; i++){                  cout << "Enter id to get bank (from 1 to 200000 (1-250 are the last ones in bigFile.txt)): " << endl;                  int id;                  cin >> id;                  Bank bank;                  auto start = chrono::high\_resolution\_clock::now();                  int index = tree->findNodeIndex(id);                  bank.id = -1;                  getBank(fileNameBin, bank, index);                  auto end = chrono::high\_resolution\_clock::now();                  auto duration = chrono::duration\_cast<chrono::microseconds>(end - start);                  cout << "id: " << bank.id;                  cout << ", name: " << bank.name;                  cout << ", city: " << bank.city;                  cout << ", commercialOrState: " << bank.commercialOrState << endl;                  cout << "Time taken by function: " << duration.count() << " microseconds" << endl;              }          }          cout << "do you want to search in file using randomized binary search tree? (y/n n by default)" << endl;          cin >> c;          if(c == 'y'){              auto treeBuildingStart = chrono::high\_resolution\_clock::now();              double averageTurns = 0;              RandomizedBinarySearchTree \*tree = createRandomizedTreeOutOfFile(fileNameBin, averageTurns);              auto treeBuildingEnd = chrono::high\_resolution\_clock::now();              auto treeBuildingTime = chrono::duration\_cast<chrono::milliseconds>(treeBuildingEnd - treeBuildingStart);              cout << "tree building time: " << treeBuildingTime.count() << " ms" << endl;              cout << "average turns: " << averageTurns << endl;              for(int i = 0; i < 3; i++){                  cout << "Enter id to get bank (from 1 to 200000 (1-250 are the last ones in bigFile.txt)): " << endl;                  int id;                  cin >> id;                  Bank bank;                  auto start = chrono::high\_resolution\_clock::now();                  int index = tree->findNodeIndex(id);                  bank.id = -1;                  getBank(fileNameBin, bank, index);                  auto end = chrono::high\_resolution\_clock::now();                  auto duration = chrono::duration\_cast<chrono::microseconds>(end - start);                  cout << "id: " << bank.id;                  cout << ", name: " << bank.name;                  cout << ", city: " << bank.city;                  cout << ", commercialOrState: " << bank.commercialOrState << endl;                  cout << "Time taken by function: " << duration.count() << " microseconds" << endl;              }          }      }      return 0;  } |