|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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 №2**

по дисциплине «Структуры и алгоритмы обработки данных»

по теме

«Алгоритмы поиска в таблице (массиве). Применение алгоритмов поиска к поиску по ключу записей в файле».

|  |  |
| --- | --- |
| **Выполнил:**  Студент группыИКБО-13-22 | Козлов Кирилл Игоревич |
| **Проверил:** | ассистент Муравьёва Е.А. |

МОСКВА 2023 г.

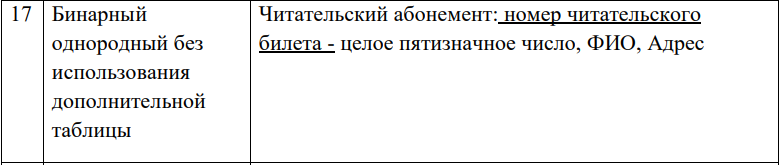
**Практическая работа № 2**

**Цель работы**

Получить практический опыт по применению алгоритмов поиска в

таблицах данных.

**Ход работы**

**Вариант 17** ****

**Задание 1**

**Формулировка задачи:**

Создать двоичный файл из записей. Поле ключа записи в задании варианта подчеркнуто. Заполнить файл данными, используя для поля ключа датчик случайных чисел. Ключи записей в файле уникальны.

**Решение:**

Размер записи: 88 байт



Рисунок 1 – результат тестирования 1

Для решения данной задачи были написаны функции для псевдорандомной генерации записи, а также функции для записи этих записей в обычный и бинарный файл. Также была реализована структура читательского билета согласно условию задания.

|  |
| --- |
| struct readerTicket  {  int ticketNumber;  std::string fio;  std::string adress;  public:  readerTicket(std::string fio, std::string adress, int ticketNumber)  {  this->ticketNumber = ticketNumber;  this->adress = adress;  this->fio = fio;  }  readerTicket()  {  this->ticketNumber = 0;  this->adress = "";  this->fio = "";  }  };int generateRandomReaderTicket()  {  int min = 10000;  int max = 99999;  return(min + rand() % (max - min + 1));  }  bool isUnique(int num, readerTicket\* arr, int length)  {  for (int i = 0; i < length; i++)  {  if (arr[i].ticketNumber == num) { return (false); }  }  return(true);  }  void writeToArrayTickets(readerTicket\* arr, int length)  {  int randNum;  for (int i = 0; i < length; i++)  {  randNum = generateRandomReaderTicket();  if (isUnique(randNum, arr, length))  {  arr[i].ticketNumber = randNum;  arr[i].adress = "Preobrag.street, house num: " + std::to\_string(i);  arr[i].fio = "fio" + std::to\_string(i);  }  }  } |

Листинг 1 – Код создания текстового и бинарного файла

**Задание 2**

**Формулировка задачи:**

Поиск в файле с применением линейного поиска:

1. Разработать программу поиска записи по ключу в бинарном файле,

созданном в первом задании, с применением алгоритма линейного

поиска.

1. Провести практическую оценку времени выполнения поиска на файле объемом 100, 1000, 10 000 записей.
2. Составить таблицу с указанием результатов замера времени.

**Решение:**

Для решения данной задачи были написаны функции чтения бинарного файла содержащая линейный поиск записи по ключу, введённым пользователем.

|  |
| --- |
| bool readFromFileTicketBin(int length, readerTicket\* arr)  {  readerTicket temp;  int userKey;  std::ifstream finBin("readerTicketList.bin", std::ios::binary);  std::cout << "Enter key to linear search" << '\n';  std::cin >> userKey;  auto begin = std::chrono::steady\_clock::now();  while (!finBin.eof())  {  finBin.read(reinterpret\_cast<char\*>(&temp), sizeof(readerTicket));  if (temp.ticketNumber == userKey)  {  std::cout << "ticket has been found!" << '\n';  std::cout << "Adress: " << temp.adress << '\n';  std::cout << "FIO: " << temp.fio << '\n';  std::cout << "Ticket Number: " << temp.ticketNumber << '\n';  break;  }  }  auto end = std::chrono::steady\_clock::now();  std::cout << "Time: " << std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin).count() << " miliseconds" << '\n';  finBin.close();  return(true);  } |

Листинг 2 – функция для решения задания 2

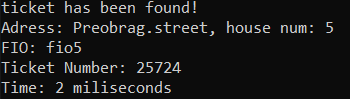


Рисунок 2 – результат тестирования 100 значений

|  |  |
| --- | --- |
| Записи | Время, милисекунд |
| 100 записей | 2 |
| 1000 записей | 2,1 |
| 10000 записей | 3,1 |

Замеры с указанием времени для линейного поиска

**Задание 3**

Формулировка задачи:

Поиск записи в файле с применением дополнительной структуры данных,

сформированной в оперативной памяти (бинарный однородный поиск).

1. Для оптимизации поиска в файле создать в оперативной памяти структур данных – таблицу, содержащую ключ и ссылку (смещение) на запись в файле.
2. Программа должна принимать на вход ключ и искать в таблице элемент, содержащий ключ поиска, а возвращать ссылку на запись в файле. Алгоритм поиска определен в варианте.
3. Программа должна принимать эту ссылку на запись в файле, считывать ее, применяя механизм прямого доступа к записям файла и возвращать прочитанную запись как результат.
4. Провести практическую оценку времени выполнения поиска на файле объемом 100, 1000, 10 000 записей.
5. Составить таблицу с указанием результатов замера времени.

**Решение:**

Создаём структуру для таблицы смещений, которая будет содержать ключ (numberTicket) и смещение (pos) в бинарном файле. В качестве смещения используем тип streampos, который хранит позицию курсора в бинарном файле.

Для решения поставленной задачи была реализована функция, которая содержит в себе: создание массива структур(таблица), к которому будут применены реализованные в этой же функции сортировка (встроенная сортировка с применением функции-компаратора для сортировки массива пользовательских структур из таблицы) и бинарный однородный поиск. Бинарный однородный поиск реализован отдельной рекурсивной функцией, что позволяет незначительно снизить время работы. По сути, логика работы однородного бинарного поиска такая же как и обычного бинарного поиска, за исключением того, что на каждой итерации мы не храним начало и конец отрезка, а только его середину.

|  |
| --- |
| table\* binarSearch(std::vector<table>& a, int i, int key, int delta)  {  if (a[i].numberTicket == key) return(&a[i]);  if (delta == 0) return(nullptr);  if (a[i].numberTicket < key) return(binarSearch(a, i + delta / 2 + 1, key, delta / 2));  if (a[i].numberTicket > key) return(binarSearch(a, i - delta / 2 - 1, key, delta / 2));  }  void task3()  {  std::vector<table> testVec;  readerTicket a;  std::streampos i = 0;  std::ifstream finBinOne("readerTicketList.bin", std::ios::binary);  while (finBinOne.read(reinterpret\_cast<char\*>(&a), sizeof(readerTicket)))  {  table b;  b.numberTicket = a.ticketNumber;  b.pos = i;  testVec.push\_back(b);  i = finBinOne.tellg();  }  finBinOne.close();  std::sort(testVec.begin(), testVec.end(), comparator);  int keyO = 0;  std::cout << "Please, enter the key to binSearch" << '\n';  std::cin >> keyO;  table\* result = binarSearch(testVec, testVec.size()-1, keyO, testVec.size() - 1);  readerTicket k;  std::ifstream finBinTwo("readerTicketList.bin", std::ios::binary);  finBinTwo.seekg(result->pos);  finBinTwo.read(reinterpret\_cast<char\*>(&k), sizeof(readerTicket));  std::cout << k.fio << '\n';  finBinTwo.close();  } |

Листинг 3 – листинг кода к заданию 3



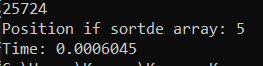




Рисунок 3, 4, 5 – результат тестирования для 100, 1000, 10000 значений

|  |  |
| --- | --- |
| Записи | Время, милисекунд |
| 100 записей | 0.5 |
| 1000 записей | 0.6 |
| 10000 записей | 0.48 |

Замеры с указанием времени для бинарного поиска

Сравнение поисков:

**Линейный поиск:**

• В худшем случае линейный поиск требует просмотра каждой записи в файле. Значит, что его временная сложность составляет O(n), где n - количество записей в файле.

• Преимущества: линейный поиск прост в реализации и может быть эффективным, даже если если файл не отсортирован, так как поиск можно остановить, как только будет найдена запись с ключом.

• Недостатки: он не эффективен для больших файлов, так как его временная сложность линейно зависит от размера файла.

**Бинарный поиск:**

• Однородный бинарный поиск требует в среднем log2(n) операций, где n - количество записей в файле.

• Преимущества: бинарный поиск является эффективным для отсортированных файлов, так как он может быстро сузить область поиска

• Недостатки: он требует, чтобы файл был отсортирован по ключу, и может потребовать дополнительного времени и ресурсов для сортировки файла.

**Вывод**

Для небольшого объема файла лучше подходит линейный поиск, так как на небольшом наборе значений сложность этого алгоритма не дает критично большого времени поиска, однако для больших файлов и файлов, отсортированных по ключу, бинарный поиск будет значительно более эффективным и быстрым способом поиска.

Листинги полного кода:

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <string>  #include <ctime>  #include <vector>  #include <set> //для набора уникальных элементов  #include <chrono> // Для измерения времени  using namespace std;  //Структура записи - Книга: уникальный ISBN, Автор, Название  struct BookRecord  {  long long ISBN = 0;  string author;  string name;  };  //ISBN, состоящий из 12 чисел  unsigned long long Randomizer()  {  long long min = 100000000000;  long long max = 999999999999;  return min + rand() % (max - min + 1);  }  int main()  {  setlocale(LC\_ALL, "Rus");  srand(static\_cast<unsigned int>(time(nullptr)));  ofstream fileText("text.txt"); //текстовый файл  if (!fileText)  {  cout << "Текстовый файл не удалось открыть для записи." << endl;  return 1;  }  cout << "Текстовый файл успешно открыт для записи." << endl << endl;  //открываем файл для записи в бинарном режиме (флаг - ios::binary)  ofstream fileBinaryOUT("text.bin", ios::binary); //бинарный файл  if (!fileBinaryOUT) {  cout << "Бинарный файл не удалось открыть для записи." << endl;  return 1;  }  cout << "Бинарный файл успешно открыт для записи." << endl << endl;  int quant = 1000; //количество записей  set<int> unique; //контейнер для уникальных значений  BookRecord record;  for (int i = 0; i < quant; ++i)  {  do {  record.ISBN = Randomizer();  } while (unique.count(record.ISBN) > 0); //если такой записи нет в контейнере  unique.insert(record.ISBN); //добавление значения в контейнер  record.author = "Автор\_" + to\_string(i);  record.name = "Название\_" + to\_string(i);  fileText << record.ISBN << ' ' << record.author << ' ' << record.name << '\n'; //запись записи в тексовый файл  fileBinaryOUT.write(reinterpret\_cast<const char\*>(&record), sizeof(record)); //запись записи в бинарный файл  }  cout << "Генерация текстового и бинарного файла завершена." << endl << endl;  fileText.close();  fileBinaryOUT.close();  ifstream fileBinaryIN("text.bin", ios::binary); //открытие бинарного файла для чтения  if (!fileBinaryIN)  {  cout << "Бинарный файл не удалось открыть для чтения." << endl;  return 1;  }  cout << "Бинарный файл успешно открыт для чтения." << endl << endl;  cout << "----Поиск книги по ISBN----" << endl << endl;  long long key;  cout << "Введите ISBN-номер книги (12 чисел): ";  cin >> key;    auto start = chrono::high\_resolution\_clock::now(); //для отсчёта времени  while (fileBinaryIN.read(reinterpret\_cast<char\*>(&record), sizeof(record))) //линейный поиск записи  {  if (record.ISBN == key)  {  cout << endl << "Запись книги найдена." << endl;  cout << "Номер ISBN: " << record.ISBN << endl;  cout << "Автор книги: " << record.author << endl;  cout << "Название книги: " << record.name << endl;  auto end = chrono::high\_resolution\_clock::now();  chrono::duration<double> duration = end - start;  cout << "Время выполнения поиска: " << duration.count() << " секунд." << endl;  fileBinaryIN.close();  return 0;  }  }  cout << "Запись с номером " << key << " не была найдена." << endl;  auto end = chrono::high\_resolution\_clock::now();  chrono::duration<double> duration = end - start;  cout << "Время выполнения поиска: " << duration.count() << " секунд." << endl;  fileBinaryIN.close();  return 0;  } |

Листинг 4 – Полный код программы, реализующий задачи 1 и 2

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <string>  #include <ctime>  #include <vector>  #include <set> //для набора уникальных элементов  #include <chrono> // Для измерения времени  #include <algorithm>  using namespace std;  //Структура записи - Книга: уникальный ISBN, Автор, Название  struct BookRecord  {  long long ISBN = 0;  string author;  string name;  };  //ISBN, состоящий из 12 чисел  unsigned long long Randomizer()  {  long long min = 100000000000;  long long max = 999999999999;  return min + rand() % (max - min + 1);  }  struct OffsetTableEntry  {  long long ISBN = 0;  streampos offset; //streampos хранит позицию(смещение) в бинарном файле  };  //функция для сортировки таблицы смещений по ключу  bool CompareByISBN(const OffsetTableEntry& a, const OffsetTableEntry& b)  {  return a.ISBN < b.ISBN;  }  int main()  {  setlocale(LC\_ALL, "Rus");  srand(static\_cast<unsigned int>(time(nullptr)));  ofstream fileText("text.txt"); //текстовый файл  if (!fileText)  {  cout << "Текстовый файл не удалось открыть для записи." << endl;  return 1;  }  cout << "Текстовый файл успешно открыт для записи." << endl << endl;  //открываем файл для записи в бинарном режиме (флаг - ios::binary)  ofstream fileBinaryOUT("text.bin", ios::binary); //бинарный файл  if (!fileBinaryOUT) {  cout << "Бинарный файл не удалось открыть для записи." << endl;  return 1;  }  cout << "Бинарный файл успешно открыт для записи." << endl << endl;  int quant = 10000; //количество записей  set<int> unique;  BookRecord record;  for (int i = 0; i < quant; ++i)  {  do {  record.ISBN = Randomizer();  } while (unique.count(record.ISBN) > 0);  unique.insert(record.ISBN);  record.author = "Автор\_" + to\_string(i);  record.name = "Название\_" + to\_string(i);  fileText << record.ISBN << ' ' << record.author << ' ' << record.name << '\n';  fileBinaryOUT.write(reinterpret\_cast<const char\*>(&record), sizeof(record));  }  cout << "Генерация текстового и бинарного файла завершена." << endl << endl;  fileText.close();  fileBinaryOUT.close();  vector<OffsetTableEntry> offsetTable; // Таблица смещений  ifstream fileBinaryIN("text.bin", ios::binary);  if (!fileBinaryIN)  {  cout << "Бинарный файл не удалось открыть для чтения." << endl;  return 1;  }  cout << "Бинарный файл успешно открыт для чтения." << endl << endl;  streampos currentOffset = 0; //изначальная позиция смещения  while (fileBinaryIN.read(reinterpret\_cast<char\*>(&record), sizeof(record)))  {  OffsetTableEntry entry; //создание записи в таблице смещений  entry.ISBN = record.ISBN;  entry.offset = currentOffset;  offsetTable.push\_back(entry);  currentOffset = fileBinaryIN.tellg(); //Обновление позиции чтения для следующей записи  }  //сортировка массива по ключу(для бинарного поиска)  sort(offsetTable.begin(), offsetTable.end(), CompareByISBN);  fileBinaryIN.close();  //бинарный однородный поиск по ключу  cout << "----Поиск книги по ISBN----" << endl << endl;  streampos offset = -1; //индикатор того что запись не найдена  long long key;  cout << "Введите ISBN-номер книги (12 чисел): ";  cin >> key;  auto start = chrono::high\_resolution\_clock::now();  //установка индексов для бинарного поиска  size\_t left = 0;  size\_t right = offsetTable.size() - 1;  //однородный бинарный поиск  while (left <= right)  {  size\_t mid = left + (right - left) / 2;  if (offsetTable[mid].ISBN == key)  {  offset = offsetTable[mid].offset;  break;  }  else if (offsetTable[mid].ISBN < key)  {  left = mid + 1;  }  else  {  right = mid - 1;  }  }  //если запись нашлась  if (offset != -1)  {  ifstream fileBinaryIN("text.bin", ios::binary);  if (fileBinaryIN)  {  fileBinaryIN.seekg(offset); //установка позиции чтения на место найденной записи  if (fileBinaryIN.read(reinterpret\_cast<char\*>(&record), sizeof(record))) //считывание записи  {  cout << endl << "Запись книги найдена." << endl;  cout << "Номер ISBN: " << record.ISBN << endl;  cout << "Автор книги: " << record.author << endl;  cout << "Название книги: " << record.name << endl;  auto end = chrono::high\_resolution\_clock::now();  chrono::duration<double> duration = end - start;  cout << "Время выполнения поиска: " << duration.count() << " секунд." << endl;  fileBinaryIN.close();  return 0;  }  }  }  cout << "Запись с номером " << key << " не была найдена." << endl;  auto end = chrono::high\_resolution\_clock::now();  chrono::duration<double> duration = end - start;  cout << "Время выполнения поиска: " << duration.count() << " секунд." << endl;  fileBinaryIN.close();  return 0;  } |

Листинг 5 – Полный код программы, реализующий задачу 3