|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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 | Руденко А.Д. |
| **Проверил:** | ассистент Муравьёва Е.А. |

МОСКВА 2023 г.

**Цель:** получить практический опыт по применению алгоритмов поиска в таблицах данных.

**Задание:** разработать программу поиска записей с заданным ключом в двоичном файле с применением различных алгоритмов.

**Отчет по заданию 1**

Размер записи в байтах: 48

Прямой доступ к записям в бинарном файле означает возможность читать и записывать данные в файле, используя смещения (offsets) в файле для точного определения местоположения каждой записи.

Предусловия:

1. Функция uniqueChecker должна быть реализована и корректно работать для проверки уникальности номеров читательского билета.
2. N - целое положительное число, представляющее количество записей, которые требуется создать в бинарном файле.
3. Вектор records должен быть пустым перед вызовом функции.
4. Вектор companies должен содержать доступные значения для организаций.

Постусловия:

1. Функция создает бинарный файл "test.bin" и записывает в него N записей в бинарном формате.
2. Каждая запись в файле содержит информацию о номере страхового и наименовании страховой.
3. Все страховые номера в файле должны быть уникальными.
4. Функция освобождает память, выделенную для хранения записей в векторе records с использованием delete.
5. Функция выводит сообщение "Файл создан ( n ) записей успешно" после успешного создания и заполнения бинарного файла.
6. Гарантируется, что созданный файл корректно закрывается после записи данных в него.

Код программы:

// Функция для проверки уникальности номера

bool uniqueChecker(const vector<Record\*>& records, int key) {

for (const Record\* record : records) {

if (record->registrationNumber == key)

return false;

}

return true;

}

// Функция для создания файла с записями

map<int, int> createFile(int N) {

srand(time(NULL));

vector<Record\*> records;

string companies[] = { "CompanyA", "CompanyB", "CompanyC", "CompanyD", "CompanyE" };

// Создание уникального номера

for (int i = 0; i < N; i++) {

Record\* newRecord = new Record();

int newKey = 100000 + rand() % 900000; // Генерация случайного 6-значного номера

//Проверка на уникальность уже созданных записей

while (!uniqueChecker(records, newKey))

newKey = 100000 + rand() % 900000;

newRecord->registrationNumber = newKey;

newRecord->companyName = companies[rand() % 5];

records.push\_back(newRecord);

}

map<int, int> codeToRecordMap;

//Запись данных в бинарный файл

ofstream fout("test.bin", ios\_base::binary);

for (int i = 0; i < N; i++) {

size\_t before = fout.tellp(); //сохраняем смещение

//Запись читательского кода

fout.write((char\*)&records[i]->registrationNumber, sizeof(int));

cout << records[i]->registrationNumber << " : " << records[i]->companyName << endl;

//Запись в файл

const char\* companyData = records[i]->companyName.c\_str();

int companyLength = records[i]->companyName.size();

fout.write(companyData, companyLength + 1);

codeToRecordMap[records[i]->registrationNumber] = before;

}

fout.close();

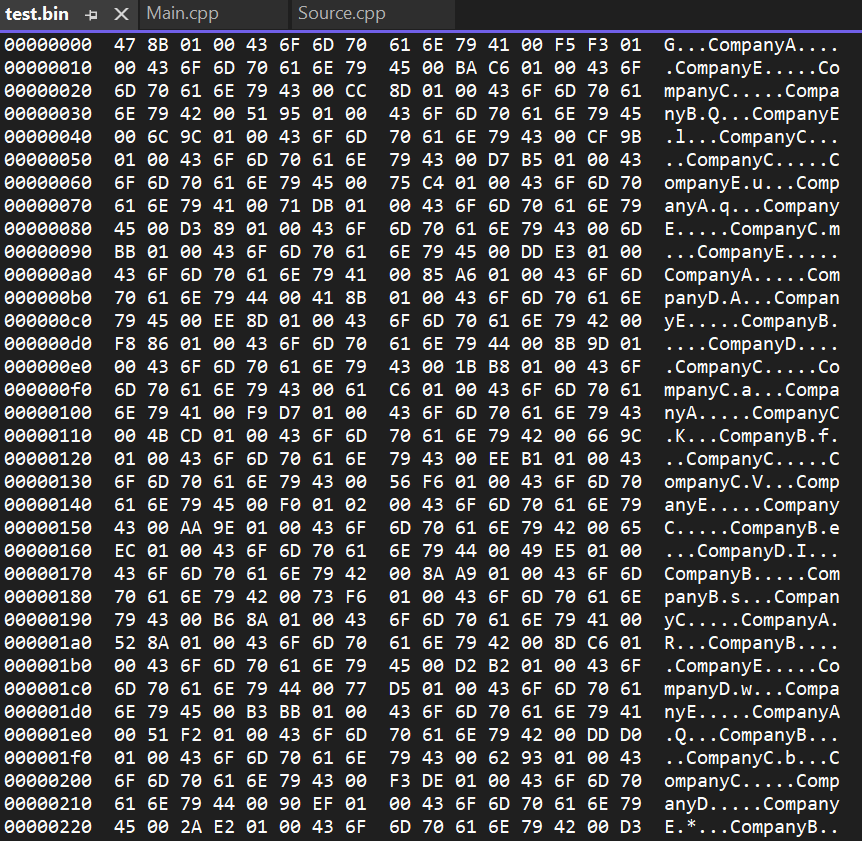
for (Record\* record : records)

delete record;

cout << "Файл на ( " << N << " ) записей создан" << endl;

return codeToRecordMap;

}



Тестирование на 100 элементах

**Отчет по заданию 2**

Псевдокод:

Алгоритм линейного поиска записи с ключом в файле в текстовой форме:

1. Открыть бинарный файл для чтения.
2. Пока не достигнут конец файла:
   1. Прочитать следующую запись из файла во временную переменную "record".
   2. Проверить, соответствует ли ключ записи ключу, который мы ищем.
   3. Если ключи совпадают, закрыть файл и вернуть найденную запись.
3. Если весь файл был прочитан и нужная запись не найдена, закрыть файл и вернуть сообщение "Запись не найдена".

Предусловия:

1. В файле "test.bin" должны быть записаны данные в бинарном формате, и структура данных в файле должна соответствовать ожидаемой структуре Record, чтобы можно было корректно считывать данные.
2. Файл "test.bin" должен существовать и быть доступным для чтения.
3. Переменная num должна быть положительным целым числом, указывающим количество записей, которые могут быть прочитаны из файла.

Постусловия:

1. Функция открывает файл "test.bin" для чтения в бинарном режиме.
2. Пользователь должен ввести значение codeToFind, которое представляет ключ, который нужно найти в записях файла.
3. Функция выполняет линейный поиск записей в файле по ключу codeToFind.
4. Если запись с указанным ключом найдена, функция выводит информацию о ней, включая номер и наименование страховой компании.
5. Если запись не найдена, функция завершает выполнение, и пользователю выводится сообщение "Запись не найдена".
6. Файл "test.bin" закрывается после завершения чтения и поиска записи.

Начало формы

Код программы:

void linearFindRecord(int num) {

// Открытие бинарного файла для чтения

ifstream fin("test.bin", ios\_base::binary);

int code\_to\_find;

cout << "Введите код для нахождения: ";

cin >> code\_to\_find;

for (int i = 0; i < num; i++) {

Record tmp;

// Чтение номера

fin.read((char\*)&tmp.readerTicketNumber, sizeof(int));

string companyData;

char ch;

fin.get(ch);

while (ch != '\0') {

companyData += ch;

fin.get(ch);

}

tmp.fullName = companyData;

string nameData;

fin.get(ch);

while (ch != '\0') {

nameData += ch;

fin.get(ch);

}

tmp.address = nameData;

if (tmp.readerTicketNumber == code\_to\_find) {

cout << "Запись найдена: " << endl;

cout << "Код: " << tmp.readerTicketNumber << endl;

cout << "Фамилия: " << tmp.fullName << endl;

cout << "Адрес: " << tmp.address << endl;

break;

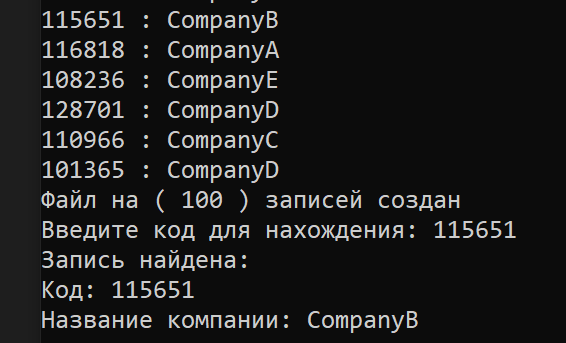
}

}

// Закрытие файла

fin.close();

}



Тестирование линейного поиска на 100 записей

|  |  |
| --- | --- |
| Замеры | Время, милисекунд |
| 100 записей | 16 |
| 1000 записей | 54 |
| 10000 записей | 60 |

Замеры с указанием времени для линейного поиска

**Отчет по заданию 3**

Описание алгоритма доступа к записи в файле посредством таблицы:

Создание бинарного файла: В начале работы программы создается бинарный файл с именем "test.bin", который будет использоваться для хранения записей данных. Этот файл создается с помощью функции createFile, которая генерирует случайные записи данных и сохраняет их в файле, а также создает таблицу codeToRecordMap.

Создание таблицы codeToRecordMap: Таблица codeToRecordMap представляет собой ассоциативный контейнер (map), который отображает ключи (например, номера регистрации) на смещения (offsets) в бинарном файле, где начинаются соответствующие записи данных. Эта таблица создается и заполняется в функции createFile, где каждая запись данных связывается с ее смещением в файле.

Интерполяционный поиск записи: Для поиска записи с определенным ключом (например, codeToFind), используется интерполяционный поиск. Этот поиск выполняется в функции interpolationRecordFind. Основные шаги интерполяционного поиска:

a. Используется таблица codeToRecordMap, чтобы найти смещение (offset) в бинарном файле, где начинается запись с ключом, который ищется. Это делается с помощью функции map::find, которая позволяет быстро найти смещение по ключу.

b. Если ключ найден в таблице, то считывается запись данных из файла по найденному смещению с использованием функции reading, которая выполняет чтение данных из файла и возвращает соответствующую запись Record.

c. Если запись с ключом была найдена, то информация о ней выводится на экран, включая номер регистрации и название компании.

d. Если запись с ключом не была найдена в таблице, выводится сообщение о том, что запись не найдена.

Закрытие файла: После завершения чтения данных из файла, файл закрывается с помощью функции fin.close().

Псевдокод:

1. Найдите смещение (offset) в файле по ключу с помощью таблицы codeToRecordMap.
2. Если ключ найден в таблице:
   1. Считайте запись данных из файла по найденному смещению.
   2. Выведите информацию о найденной записи на экран.3. Если ключ не найден в таблице:
3. Выведите сообщение о том, что запись не найдена в файле.
4. Закройте файл.

Предусловие:

* Входные данные codeToRecordMap должны представлять собой отображение (map), где ключи - это целые числа (int), а значения - целые числа (int), представляющие смещения в бинарном файле.
* Переменная codeToFind должна содержать ключ, который необходимо найти в файле.
* Файл "test.bin" должен существовать и быть доступным для чтения. В противном случае, функция может не работать корректно.

Постусловие:

* Если ключ codeToFind найден в файле, функция выводит на стандартный вывод (cout) информацию о записи с найденным ключом, включая код, и наименование организации.
* Если ключ codeToFind не найден в файле, функция выводит сообщение об этом.
* Функция корректно завершает выполнение и не оставляет открытыми файлы или утечки ресурсов.

Алгоритм бинарного поиска:

void interpolationFindRecord(const map<int, int>& codeToRecordMap, int codeToFind) {

map<int, int>::const\_iterator iterator;

int n = codeToRecordMap.size();

int left = 0;

int right = n - 1;

while (left <= right && codeToFind >= codeToRecordMap.begin()->first && codeToFind <= codeToRecordMap.rbegin()->first) {

iterator = codeToRecordMap.begin();

// Интерполяционная формула для приближенной позиции

int pos = left + ((codeToFind - iterator->first) \* (right - left)) / (codeToRecordMap.rbegin()->first - iterator->first);

advance(iterator, pos);

if (iterator->first == codeToFind) {

Record tmp = reading(iterator->second);

cout << "Запись найдена: " << endl;

cout << "Код: " << tmp.registrationNumber << endl;

cout << "Название компании: " << tmp.companyName << endl;

return;

}

else if (iterator->first < codeToFind) {

left = pos + 1; // Искать в правой половине

}

else {

right = pos - 1; // Искать в левой половине

}

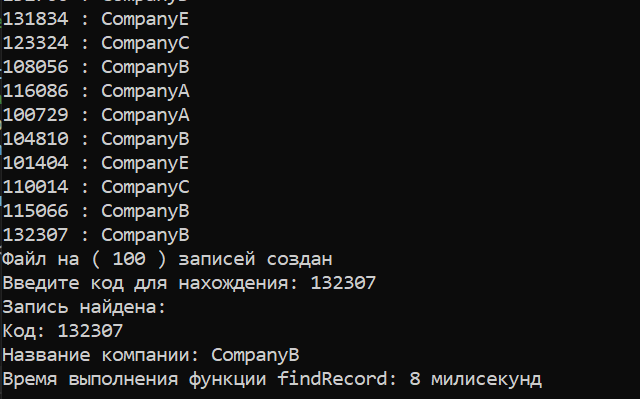
}

cout << "Запись не найдена!";

}

|  |  |
| --- | --- |
| Замеры | Время, милисекунд |
| 100 записей | 8 |
| 1000 записей | 7 |
| 10000 записей | 7 |

Замеры с указанием времени для интерполяционного поиска



Результаты бинарного поиска для 100 записей

Сравнение поисков:

Линейный поиск:

* Временная сложность: В худшем случае линейный поиск требует просмотра каждой записи в файле. Это означает, что его временная сложность составляет O(N), где N - количество записей в файле.
* Преимущества: Линейный поиск прост в реализации и может быть эффективным, если файл отсортирован, так как поиск можно остановить, как только будет найдена запись с ключом (или ключ станет больше).
* Недостатки: Он не эффективен для больших файлов, так как его временная сложность линейно зависит от размера файла.

Интерполяционный поиск:

* Временная сложность: Интерполяционный поиск также требует log2(N) операций в среднем. Однако, его производительность может быть лучше адаптирована к распределению ключей в файле, и в некоторых случаях, он может быть даже более быстрым, чем бинарный поиск.
* Преимущества: Интерполяционный поиск работает лучше, когда ключи в файле имеют равномерное распределение, и он способен быстро сузить область поиска, основываясь на значении ключа, что может привести к ускорению поиска.
* Недостатки: Он также требует отсортированного файла по ключу, и если ключи в файле распределены неравномерно, его производительность может быть хуже, чем у бинарного поиска.

Вывод:

* Если файл содержит малое количество записей или не требует сортировки, линейный поиск может быть простым и достаточно эффективным решением.
* Для больших файлов и файлов, отсортированных по ключу, интерполяционный поиск будет значительно более эффективным и быстрым способом поиска.
* Важно выбирать алгоритм в зависимости от размера и организации файла, а также от требований к скорости поиска

Весь код:

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <cstring>

#include <ctime>

#include <map>

#include <chrono>

using namespace std;

using namespace chrono;

//Структура записи

struct Record {

int registrationNumber;

string companyName;

};

// Функция для проверки уникальности номера

bool uniqueChecker(const vector<Record\*>& records, int key) {

for (const Record\* record : records) {

if (record->registrationNumber == key)

return false;

}

return true;

}

// Функция для создания файла с записями

map<int, int> createFile(int N) {

srand(time(NULL));

vector<Record\*> records;

string companies[] = { "CompanyA", "CompanyB", "CompanyC", "CompanyD", "CompanyE" };

// Создание уникального номера

for (int i = 0; i < N; i++) {

Record\* newRecord = new Record();

int newKey = 100000 + rand() % 900000; // Генерация случайного 6-значного номера

//Проверка на уникальность уже созданных записей

while (!uniqueChecker(records, newKey))

newKey = 100000 + rand() % 900000;

newRecord->registrationNumber = newKey;

newRecord->companyName = companies[rand() % 5];

records.push\_back(newRecord);

}

map<int, int> codeToRecordMap;

//Запись данных в бинарный файл

ofstream fout("test.bin", ios\_base::binary);

for (int i = 0; i < N; i++) {

size\_t before = fout.tellp(); //сохраняем смещение

//Запись читательского кода

fout.write((char\*)&records[i]->registrationNumber, sizeof(int));

cout << records[i]->registrationNumber << " : " << records[i]->companyName << endl;

//Запись в файл

const char\* companyData = records[i]->companyName.c\_str();

int companyLength = records[i]->companyName.size();

fout.write(companyData, companyLength + 1);

codeToRecordMap[records[i]->registrationNumber] = before;

}

fout.close();

for (Record\* record : records)

delete record;

cout << "Файл на ( " << N << " ) записей создан" << endl;

return codeToRecordMap;

}

// Функция для поиска записи линейным поиском

void linearFindRecord(int num, int code\_to\_find) {

// Открытие бинарного файла для чтения

ifstream fin("test.bin", ios\_base::binary);

for (int i = 0; i < num; i++) {

Record tmp;

// Чтение номера

fin.read((char\*)&tmp.registrationNumber, sizeof(int));

string companyData;

char ch;

fin.get(ch);

while (ch != '\0') {

companyData += ch;

fin.get(ch);

}

tmp.companyName = companyData;

if (tmp.registrationNumber == code\_to\_find) {

cout << "Запись найдена: " << endl;

cout << "Код: " << tmp.registrationNumber << endl;

cout << "Название компании: " << tmp.companyName << endl;

break;

}

}

// Закрытие файла

fin.close();

}

Record reading(int offset) {

ifstream fin("test.bin", ios\_base::binary);

fin.seekg(offset); // Смещение в файле

Record tmp;

// Чтение данных из файла

fin.read((char\*)&tmp.registrationNumber, sizeof(int));

string companyName;

char ch;

fin.get(ch);

while (ch != '\0') {

companyName += ch;

fin.get(ch);

}

tmp.companyName = companyName;

fin.close();

return tmp;

}

void interpolationFindRecord(const map<int, int>& codeToRecordMap, int codeToFind) {

map<int, int>::const\_iterator iterator;

int n = codeToRecordMap.size();

int left = 0;

int right = n - 1;

while (left <= right && codeToFind >= codeToRecordMap.begin()->first && codeToFind <= codeToRecordMap.rbegin()->first) {

iterator = codeToRecordMap.begin();

// Интерполяционная формула для приближенной позиции

int pos = left + ((codeToFind - iterator->first) \* (right - left)) / (codeToRecordMap.rbegin()->first - iterator->first);

advance(iterator, pos);

if (iterator->first == codeToFind) {

Record tmp = reading(iterator->second);

cout << "Запись найдена: " << endl;

cout << "Код: " << tmp.registrationNumber << endl;

cout << "Название компании: " << tmp.companyName << endl;

return;

}

else if (iterator->first < codeToFind) {

left = pos + 1; // Искать в правой половине

}

else {

right = pos - 1; // Искать в левой половине

}

}

cout << "Запись не найдена!";

}

int main() {

setlocale(LC\_ALL, "ru");

//cout << " == " << sizeof(Record) << endl; //Вывод кол-ва байт занимаемых одной записью

int N = 100;

map<int, int> codeToRecordMap = createFile(N); // Создание файла и таблицы

int code\_to\_find;

cout << "Введите код для нахождения: ";

cin >> code\_to\_find;

//linearFindRecord(N, code\_to\_find); // Поиск записи по номеру

// Замер выполнения метода

auto start\_time = high\_resolution\_clock::now();

//linearFindRecord(N, code\_to\_find);

interpolationFindRecord(codeToRecordMap, code\_to\_find);

auto end\_time = high\_resolution\_clock::now();

auto duration = duration\_cast<milliseconds>(end\_time - start\_time);

cout << "Время выполнения функции findRecord: " << duration.count() << " милисекунд" << endl;

return 0;

}