МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического занятия 10

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

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент Хвостов В. В.

Группа ИКБО-01-20

Москва 2021

Содержание

Цель работы - получить практический опыт по применению алгоритмов поиска в таблицах данных.

Вариант 10.

Задание 1

1.1. Постановка задачи

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

1.2. Описание подхода к решению

Структура записи файла

Согласно варианту No10 индивидуального задания запись файла представляет собой страхование автосредства и состоит из двух полей: регистрационный номер - шестизначное число и название страховой компании. Сортировка производится по регистрационному номеру.

Размер записи в байтах

Каждая запись представлена целым шестизначным числом и строковой переменной, размер которой зависит от ее длины, поэтому определить точный размер записи не представляется возможным. Однако можно посчитать средний размер записи, зная количество строки вес одного файла с записями. Сгенерировав 10 раз файл размеров из 100 строк, в среднем получился размер 1883 байта, соответственно вес одной записи равен около 19 байтов.

Алгоритмы, реализованные в форме функций

Для создания текстового файла были использованы следующие функции:

```
using DataRow = std :: pair <int , std :: string >;

DataRow getRandomRow() noexcept ;
void generateSortedFile ( const std :: string & file_name , int row_number);
void generateUnsortedFile ( const std :: string & file_name , int row_number);
```

1.3. Код программы

Код программы:

```
#include <algorithm>
#include <fstream>
#include <iostream>
#include <random>
#include <string>
#include <utility>
#include <vector>

constexpr int error_code = 2;
// not work with g++ now =( constexpr std::string standard_file_name = "search_data.txt";
constexpr auto standard_file_name = "search_data.txt"; // auto = char []

const std::vector<std::string> insurance_companies_names = {"Aflac",
  "Acuity_Insurance", "Allianz_life", "Allstate", "Berkshire_Hathaway",
  "CareSource", "Chubb_Corp", "CNO_Financial", "Delta_Dental",
  "Erie_Insurance_Group", "Esurance", "Evergreen", "FM_Global",
  "GAINSCO", "GEICO", "General_Re", "Hanover_Insurance",
  "Ironshore", "K&K_Insurance", "Kemper_Corporation",
  "Knights_of_Columbus", "Lemonade_Inc", "Liberty_Mutual",
  "MassMutual", "MetLife", "Metromile", "Modern_Woodmen_of_America",
  "Omega", "OneBeacon", "Pacific_Life", "PEMCO", "Penn_Mutual",
  "Primerica", "Progressive", "Protective_Life", "Pure_Insurance",
  "QBE", "Safeco", "Society_Insurance", "SquareTrade", "Symetra",
  "The_General", "TIAA-CREF", "Tricare", "Trupanion", "Unum",
  "USAA", "West_Coast_Life", "XL_Catlin", "Zurich_Insurance_Group"};

using DataRow = std::pair<int, std::string>;

DataRow getRandomRow() noexcept;
void generateSortedFile(const std::string& file_name, int row_number);
void generateUnsortedFile(const std::string& file_name, int row_number);

// because it's created many times
// so static allows program not to create objects again
DataRow getRandomRow() noexcept {
    static std::random_device rd;
    static std::mt19937 gen(rd());
    static std::uniform_int_distribution<int> strDist(0, insurance_companies_names.size() - 1);

    static const int lower_bound = 100'000;
    static const int upper_bound = 999'999;
    static std::uniform_int_distribution<int> numDist(lower_bound, upper_bound);

   return {numDist(gen), insurance_companies_names[strDist(gen)]};
}
```

```
void generateSortedFile(const std::string& file_name, int row_number) {
    std::vector<DataRow> rows(row_number); // for std::unique
    for (DataRow& row: rows) {
        row = getRandomRow();
    }

    auto lineComparison = [](const DataRow& first, const DataRow& second)
    { return first.first < second.first; }; // first field - id
    std::sort(rows.begin(), rows.end(), lineComparison);
    auto lineEqualComparison = [](const DataRow& first, const DataRow& second)
    { return first.first == second.first; }; // first field - id
    std::sort(rows.begin(), rows.end(), lineComparison);
    auto it = std::unique(rows.begin(), rows.end(), lineEqualComparison);
    rows.resize(std::distance(rows.begin(), it));

    try {
        std::ofstream fout;
        fout.open(file_name);
        for (const auto& [number, name]: rows) {
            fout << number << " " << name << '\n';
        }
        fout.close();
    } catch(std::exception& ex) {
        std::cerr << ex.what() << std::endl;
        std::exit(error_code);
    }
}

void generateUnsortedFile(const std::string& file_name, int row_number) {
    std::vector<DataRow> rows(row_number); // for std::unique
    for (DataRow& row: rows) {
        row = getRandomRow();
    }

    auto lineComparison = [](const DataRow& first, const DataRow& second)
    { return first.first < second.first; }; // first field - id
    std::sort(rows.begin(), rows.end(), lineComparison);
    auto lineEqualComparison = [](const DataRow& first, const DataRow& second)
    { return first.first == second.first; }; // first field - id
    std::sort(rows.begin(), rows.end(), lineComparison);
    auto it = std::unique(rows.begin(), rows.end(), lineEqualComparison);
    rows.resize(std::distance(rows.begin(), it));

    for (size_t i = 0; i < rows.size(); ++i) { // Swaping elements randomly
        std::random_device rd;
        std::mt19937 gen(rd());
        std::uniform_int_distribution<size_t> dist(0, rows.size() - 1);
```

```
        std::swap(rows[dist(gen)], rows[dist(gen)]);
    }

    try {
        std::ofstream fout;
        fout.open(file_name);
        for (const auto& [number, name]: rows) {
            fout << number << " " << name << '\n';
        }
        fout.close();
    } catch(std::exception& ex) {
        std::cerr << ex.what() << std::endl;
        std::exit(error_code);
    }
}

int main() {
    int row_number;
    std::cout << "Enter the row number of a new file:\n";
    std::cin >> row_number;

    generateUnsortedFile(standard_file_name, row_number);

    std::cout << "Unsorted file was created successfully\n";
    return 0;
}
```

Предусловия и постусловия функций

1. getRandomRow:
    - Предусловие - число записей больше нуля
    - Постусловие - создана строка со случайными параметрами
2. generateSortedFile:
    - Предусловие - число записей больше нуля, строка с именем файла не нулевая
    - Постусловие - создан отсортированный текстовый файл с неповторяющимися ключами с указанным числом записей и с указанным названием
3. generateUnsortedFile:

- Предусловие - число записей больше нуля, строка с именем файла не нулевая
- Постусловие - создан текстовый файл с неповторяющимися ключами с указанным числом записей и с указанным названием

Пример сгенерированного файла приведен на рисунке 1.

Рис. 1 - Пример сгенерированного файла (первые строки)

1.4. Тестирование программы

Для того чтобы убедиться, что генератор записей работает верно произведем тестирование функций (табл. 1).

Таблица 1 - Тестирование задачи 1

Номер теста	Входные данные	Ожидаемый результат	Результат выполнения программы (размер файла)
1	10	≈190 байт	File: search_data.txt Size: 187
2	100	≈1900 байт	File: search_data.txt Size: 1925
3	1000	≈19000 байт	File: search_data.txt Size: 19192

Задание 2

2.1. Постановка задачи

Разработать программу поиска записи по ключу в текстовом файле с применением алгоритма линейного поиска.

2.2. Алгоритм линейного поиска

Параметры функции: file_name - имя файла, в котором будет производиться поиск; key - ключ, по которому будет производиться поиск. Остальные переменные описаны в комментариях алгоритма.

Алгоритм 1 Алгоритм функции линейного поиска

```
function linearSearchFile(file_name, key)
    Let's fin - input file stream                    ▷ Поток чтения из файла
    fin.open(file_name)
    Let's line - string                              ▷ Текущая прочитанная запись
    while std::getline(fin, line) do
        if key in the line equals key in the parameter then
            return line
        end if
    end while
    return "no_line"                                 ▷ Запись не найдена
end function
```

2.3. Код функции поиска

linearSearchFile:

- Предусловие - файл с указанным именем существует и запись с указанным ключом существует (по условию задачи)
- Постусловие - возвращена запись с указанным ключом

```
#include <fstream>
#include <iostream>
#include <string>
#include <vector>

#include "utils.cpp"
```

```
// it pollutes global namespace but not a real code so ...
constexpr auto no_key = "No key";

int parseKey(const std::string& line) {
  auto pos = std::find(line.begin(), line.end(), ' ');
  std::string res_string = std::string(line.begin(), pos);
  return std::stoi(res_string);
}

std::string linearSearchFile(const std::string& file_name, int key) {
  std::ifstream fin;
  fin.open(file_name);
  std::string line;
  while (std::getline(fin, line)) {
    if (parseKey(line) == key) {
      return line;
    }
  }
  return no_key;
}
```

2.4. Код программы

Код файла main_first.cpp:

```
#include "search_algos.cpp"

int main() {
  constexpr auto search_file_name = "search_data.txt";
  size_t key;
  std::cout << "Enter key to find in file" << '\n';
  std::cin >> key;
  std::string res;
  {
    TimeCounter tc;
    res = linearSearchFile(search_file_name, key);
  }
  std::cout << res << '\n';
  return 0;
}
```

2.5. Тестирование программы

Для того чтобы убедиться, что функция линейного поиска работает верно произведем тестирование функции (таблица 2).

Таблица 2 - Тестирование задачи 2

Номер теста	Входные данные	Ожидаемый результат	Результат выполнения программы
1	673124	673124 Primerica	Enter key to find in file 673124 673124 Primerica
2	451098	451098 Erie_ Insurance_ Group	Enter key to find in file 451098 451098 Erie_Insurance_Group
3	912836	912836 Liberty_Mutual	Enter key to find in file 912836 912836 Liberty_Mutual

2.6. Замера времени работы функции

Чтобы оценить скорость работы функции линейного поиска произведем тестирование функции с замером времени. Тестирование будет производится на файлах из 100 и 1000 записей. В функцию будут передаваться последовательно ключи 3-х записей – записи, расположенной в начале, в середине и в конце файла. Результаты тестирования приведены в таблице 3.

Таблица 3 - Тестирования скорости алгоритма задачи 2

Расположение записи	Время поиска в файле из 100 записей	Время поиска в файле из 1000 записей
Начало	Enter key to find in file 801665 End clock, time = 0.198293 ms 801665 Evergreen	Enter key to find in file 875589 End clock, time = 0.169292 ms 875589 Berkshire_Hathaway
Середина	Enter key to find in file 188526 End clock, time = 0.240528 ms 188526 Penn_Mutual	Enter key to find in file 767918 End clock, time = 0.669998 ms 767918 The_General
Конец	Enter key to find in file 344471 End clock, time = 0.287023 ms 344471 SquareTrade	Enter key to find in file 994423 End clock, time = 1.25966 ms 994423 Ironshore

Задание 3

3.1. Постановка задачи

Разработать функцию интерполяционного поиска записи в файле.

3.2. Описание алгоритма доступа к записи в файле посредством таблицы

Таблица будет представлять собой массив строк в порядке аналогичном записям в файле. Ссылкой на запись будет являться индекс элемента, по которому будет выводится соответствующая запись файла.

3.3. Алгоритм интерполяционного поиска

Параметры функции: rows - вектор из прочитанных записей, в котором будет производиться поиск; key - ключ, по которому будет производиться поиск. Остальные переменные описаны в комментариях алгоритма.

Алгоритм 2 Алгоритм функции интерполяционного поиска

function interpolationSearch(rows, key)
low ← 0 ▷ Нижняя граница поиска
high ← the size of the rows - 1 ▷ Верхняя граница поиска
mid ← 0 ▷ Центр поиска
while rows[high] != rows[low] and key ≥ key of the rows[low] and key ≤ key of the rows[high] **do**
mid ← low + (key - key of the rows[low]) * (high - low) / (key of the rows[high] - key of the rows[low]))
if key of the rows[mid] < key **then**
low ← mid + 1
else if key of the rows[mid] > key **then**
high ← mid - 1
else
return rows[mid]
end if
end while
if key == key of the rows[low] **then**
return rows[low]
else
return "no_key"
end if
end function

3.4. Код функции поиска

interpolationSearch:

- Предусловие - файл с указанным именем существует и запись с указанным ключом существует (по условию задачи)
- Постусловие - возвращена запись с указанным ключом

```
std :: string interpolationSearch (const std :: vector<std :: string>& rows, int key) {
  size_t low = 0;
  size_t high = rows.size() - 1;
  size_t mid = 0;

  while (rows[high] != rows[low] &&key => parseKey(rows[low])
      &&key <= parseKey(rows[high])) {
    mid = low + ((key - parseKey(rows[low])) * (high - low)
        / (parseKey(rows[high]) - parseKey(rows[low])));
```

```
    if (parseKey(rows[mid]) < key) {
      low = mid + 1;
    } else if (key < parseKey(rows[mid])) {
      high = mid - 1;
    } else {
      return rows[mid];
    }
  }
  if (key == parseKey(rows[low])) {
    return rows[low];
  } else {
    return no_key;
  }
}
```

3.5. Код программы

Код файла main_second.cpp:

```
#include "search_algos.cpp"

int main() {
  constexpr auto file_name = "search_data.txt";
  std::ifstream fin;
  fin.open(file_name);
  std::vector<std::string> rows;
  for (std::string line; std::getline(fin, line);) {
    rows.push_back(line);
  }

  std::string result;
  std::cout << "Enter the key of the row:\n";
  int key;
  std::cin >> key;

  {
    TimeCounter tc;
    result = interpolationSearch(rows, key);
  }
  std::cout << result << '\n';

  fin.close();
  return 0;
}
```

3.6. Тестирование программы

Для того чтобы убедиться, что функция интерполяционного поиска работает верно произведем тестирование функции (таблица 4).

Таблица 4 - Тестирование задачи 3

Номер теста	Входные данные	Ожидаемый результат	Результат выполнения программы
1	132343	132343 FM_Global	Enter the key of the row: 132343 132343 FM_Global
2	574851	574851 Primerica	Enter the key of the row: 574851 574851 Primerica
3	988153	988153 Allstate	Enter the key of the row: 988153 988153 Allstate

3.7. Замера времени работы функции

Чтобы оценить скорость работы функции интерполяционного поиска произведем тестирование функции с замером времени. Тестирование будет производится на файлах из 100 и 1000 записей. В функцию будут передаваться последовательно ключи 3-х записей – записи, расположенной в начале, в середине и в конце файла. Результаты тестирования приведены в таблице 5.

Таблица 5 - Тестирования скорости алгоритма задачи 3

Расположение записи	Время поиска в файле из 100 записей	Время поиска в файле из 1000 записей
Начало	Enter the key of the row: 103004 End clock, time = 0.021278 ms 103004 Chubb_Corp	Enter the key of the row: 100329 End clock, time = 0.016553 ms 100329 MetLife
Середина	Enter the key of the row: 509160 End clock, time = 0.032532 ms 509160 Hanover_Insurance	Enter the key of the row: 544427 End clock, time = 0.047611 ms 544427 CareSource
Конец	Enter the key of the row: 996771 End clock, time = 0.026477 ms 996771 CNO_Financial	Enter the key of the row: 997905 End clock, time = 0.019507 ms 997905 CareSource

Анализ эффективности алгоритмов поиска

Произведем анализ эффективности рассмотренных алгоритмов поиска в файле. Как видно из результатов замера времени для каждого алгоритма (см. таблицу 6 - общая таблица скоростей, составленная по таблицам 3 и 5), алгоритм интерполяционног поиска гораздо быстрее, чем алгоритм линейного поиска. Алгоритм интерполяционного поиска меньше зависит от длины массива и во многих частных случаях показывает себя быстрее (если элемент расположен в самой середине списка).

Таблица 6 - Таблица сравнения скоростей алгоритмов

Кол-во записей	Алгоритм поиска	Расположение		
		Начало	Середина	Конец
100	Линейный	0.198 ms	0.241 ms	0.287 ms
	Интерполяционный	0.021 ms	0.033 ms	0.026 ms
1000	Линейный	0.169 ms	0.670 ms	1.260 ms
	Интерполяционный	0.017 ms	0.048 ms	0.020 ms

Выводы

В ходе данной практической работы были получены знания и практические навыки по разработке и реализации алгоритмов поиска в таблице данных. Также были опробованы применения алгоритмов поиска к поиску по ключу записей в файле, а также получен практический опыт по применению алгоритмов поиска в таблицах данных. В первом задании был разработан генератор таблицы данных в соответствии с вариантом индивидуального задания, а также описаны прототипы используемых функций и протестирован алгоритм. Во втором задании был разработан алгоритм линейного поиска, описаны функции, протестирован алгоритм и произведён замер времени для разных случаев поиска. В третьем задании был разработан алгоритм интерполяционного поиска, протестирован и произведен замер времени для разных случаев. На основе замеров времени были проанализированы эффективности изученных алгоритмов и получен вывод, что алгоритм интерполяционного поиска является более эффективным, чем алгоритм линейного поиска.

Список информационных источников

1. Thomas H. Cormen, Clifford Stein и другие: Introduction to Algorithms, 3rd Edition. Сентябрь 2009. The MIT Press.

2. N. Wirth: Algorithms and Data Structures. Август 2004. https://people.inf.ethz.ch/wirth/AD.pdf.

3. Interpolation search // Wikipedia [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/Interpolation_search (Дата обращения: 08.05.2021)

4. Курс Algorithms, part 2 // Coursera [Электронный ресурс]. URL: https://www.coursera.org/learn/algorithms-part2 (Дата обращения: 08.05.2021)