МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Отчет по выполнению практического задания №10

Тема: Применение алгоритмов поиска по ключу

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент Антонов А.Д.

Группа ИКБО-01-20

Москва 2021

Содержание

1. Задание 1

1.1. Постановка задачи

Создать текстовый файл из записей. Заполнить файл данными, используя для поля ключа генератор случайных чисел.

1.2. Описание подхода к решению

Структура записи файла

Согласно варианту №2 индивидуального задания запись файла представляет собой: счет в банке: номер счета – 7 разрядное число, ФИО, адрес. Сортировка производится по номеру счета.

Размер записи в байтах

Каждая запись представлена строковой переменной и её размер зависит отдлины строки, поэтому определить точный размер записи не представляется возможным. Однако можно посчитать средний размер записи, зная количество строки вес одного файла с записями. Так, для 200 записей вес файла составляет около 5400 байт (рис. 1), соответственно вес одной записи около 27 байт.

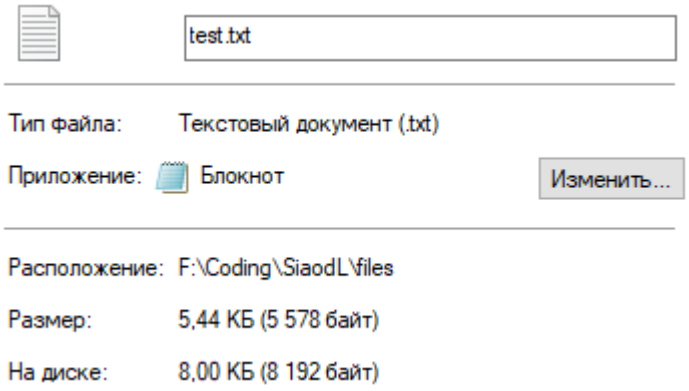


Рис. 1. Свойства файла, содержащего 200 записей

Алгоритмы, реализованные в форме функций

Для генерации текстового файла были созданы следующие функции:

```
11  string bankAccount(long int*& plate, int amount)
12  {
13      srand(time(NULL));
14      long int account = rand() % amount;
15      while (plate[account] != 0)
16          account = rand() % amount;
17      plate[account] = 1;
18      string strNum = to_string(account);
19      return strNum;
20  }
```

```
22  string* generateInfo()
23  {
24      ifstream infoTxt("F:/Coding/SiaodL/files/info.txt");
25      string info;
26      auto* bank = new string[n];
27      for (int i = 0; i < n; ++i)
28          getline(infoTxt, bank[i]);
29      infoTxt.close();
30      return bank;
31  }
```

```
33  string* generateAddress()
34  {
35      ifstream addressTxt("F:/Coding/SiaodL/files/address.txt");
36      string address;
37      auto* owners = new string[n];
38      for (int i = 0; i < n; ++i)
39          getline(addressTxt, owners[i]);
40      addressTxt.close();
41      return owners;
42  }
```

```
string* generateStr(int amount)
{
    srand(time(NULL));
    string* strComp = generateInfo();
    string* strOwn = generateAddress();
    auto* strArr = new string[amount];
    auto* keys = new long int[10 * amount]{};
    for (int i = 0; i < amount; ++i)
    {
        string num = bankAccount(keys, 10000 * amount);
        string str = num + ' ' + strComp[(rand()) % n] + ' ' + strOwn[(rand()) % n];
        strArr[i] = str;
    }
    return strArr;
}
```

```
void sortList(int amount, string*& strArr)
{
    int i, j;
    string k;
    for (j = 1; j < amount; ++j)
    {
        k = strArr[j];
        i = j - 1;
        while ((i >= 0) && (stoi(strArr[i]) > stoi(k)))
            { strArr[i + 1] = strArr[i]; --i; }
        strArr[i + 1] = k;
    }
    ofstream test("F:/Coding/SiaodL/files/out.txt");
    for (i = 0; i < amount; ++i)
        test << strArr[i] << endl;
    test.close();
}
```

```
void generateList(int amount, string*& strArr)
{
    strArr = generateStr(amount);
    ofstream test("F:/Coding/SiaodL/files/test.txt");
    for (int i = 0; i < amount; ++i)
        test << (*(strArr + i)) << endl;
    test.close();
}
```

1.3. Код программы

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <ctime>
#include <algorithm>
#include <chrono>

using namespace std;
const int n = 60;

int main()
{
    int amount;
    cout << "Enter number of bank accounts: ";
    cin >> amount;
    auto* strArr = new string[amount]{};
    generateList(amount, strArr);
    sortList(amount, strArr);
}
```

1.4. Тестирование программы

Таблица 2. Тестирование генератора записей

Номер теста	Входные данные	Ожидаемый результат	Результат (вес файла)
1	10	~260 байт	Расположение: F:\Coding\SiaodL\files Размер: 264 байт (264 байт)
2	100	~2600 байт	Расположение: F:\Coding\SiaodL\files Размер: 2,57 КБ (2 640 байт)
3	1000	~26000 байт	Расположение: F:\Coding\SiaodL\files Размер: 25,7 КБ (26 400 байт)

Как видно из таблицы 2 – все функции работают и генерируют файл необходимого размера.

2. Отчет по заданию 2

2.1. Постановка задачи

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

2.2. Алгоритм

Таблица 3. Алгоритм линейного поиска на псевдокоде

key – ключ записи, *amount* – количество записей, *str* – конкретная строка (запись) из файла

search(key, amount)
{
fstream test ("F:/Coding/SiaodL/files/test.txt");
string str;
for i←0 to amount **do**
getline(test, str);
if (key == stoi(str)) then
return str;
od
}

2.3. Код функции поиска

Функция линейного поиска открывает файл, в который были записаны данные и после этого производит последовательное сравнение ключа каждой записи с ключом, переданным в функцию.

```
87  string search(int key, int amount)
88  {
89      fstream test( s: "F:/Coding/SiaodL/files/out.txt");
90      string str;
91      for (int i = 0; i < amount; ++i)
92      {
93          getline( &: test,  &: str);
94          if (key == stoi(str)) return str;
95      }
96  }
```

2.4. Код программы линейного поиска записи по ключу

Генерируется файл с записями и после этого вводится ключ, по которому нужно найти запись. Ключ и количество записей передаются в функцию поиска. Найденная запись печатается в консоль.

```
98  int main()
99  {
100     int amount, z;
101     cout << "Enter number of bank accounts: ";
102     cin >> amount;
103     auto* strArr = new string[amount]{};
104     generateList(amount, strArr);
105     sortList(amount, strArr);
106     cout << "Enter search key: ";
107     cin >> z;
108     cout << "Search result:" << endl;
109     cout << search(z, amount);
110 }
```

2.5. Результат тестирования программы

Таблица 4. Тестирование функции линейного поиска

Номер теста	Входные данные	Ожидаемый результат	Результат (вес файла)
1	Кол-во: 10 Ключ: 1234567	1234567 Pupkin Vasily Ivanovich Pushkina, 6	Number of accounts: 10 Enter search key: 1234567 Search result: 1234567 Pupkin Vasily Ivanovich Pushkina, 6
2	Кол-во: 100 Ключ: 5553535	5553535 John Smith Jr. Park Avenue, 2A	Number of accounts: 100 Enter search key: 5553535 Search result: 5553535 John Smith Jr. Park Avenue, 2A
3	Кол-во: 1000 Ключ: 2234456	6969696 Buyanov Nikita Geneburn Tarkov, Labs	Number of accounts: 1000 Enter search key: 6969696 Search result: 6969696 Buyanov Nikita Geneburn Tarkov, Labs

Как видно из таблицы 4 – функция линейного поиска работает верно и находит запись согласно введенному ключу.

2.6. Замеры времени поиска записи по заданному ключу

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.

Код программы:

```
98  void search(int key, int amount)
99  {
100     fstream test("F:/Coding/SiaodL/files/out.txt"); string str;
101     auto begin = chrono::steady_clock::now();
102     for (int i = 0; i < amount; ++i)
103     {
104         getline(test, str); if (key == stoi(str))
105         {
106             auto end = chrono::steady_clock::now();
107             auto elapsed_ms = chrono::duration_cast<chrono::microseconds>(end - begin);
108             cout << str << endl << "Time: " << elapsed_ms.count() << " mcs";
109         }
110     }
111 }
```

```
115 int main()
116 {
117     int amount, z;
118     cout << "Enter number of bank accounts: ";
119     cin >> amount;
120     string* strArr = new string[amount]{};
121     generatelist(amount, strArr);
122     sortlist(amount, strArr);
123     cout << "Enter search key: ";
124     cin >> z;
125     cout << "Search result:" << endl;
126     search(z, amount);
127 }
```

Таблица 5. Замеры времени поиска записи по ключу

Расположение записи	100 записей	1000 записей
Начало	123 мкс	1562 мкс
Середина	742 мкс	6132 мкс
Конец	1659 мкс	11585 мкс

Как видно из таблицы 5 время поиска соответствует линейной сложности алгоритма: при увеличении значения в два раза время поиска по файлу увеличивается аналогично в два раза.

3. Отчет по заданию 3

3.1. Постановка задачи

Разработать функцию бинарного поиска записи в файле с применением дополнительной структуры данных, сформированной в оперативной памяти.

3.2. Алгоритм

Алгоритм бинарного поиска записи с ключом в файле на псевдокоде представлен в таблице 6.

Таблица 6. Алгоритм бинарного поиска на псевдокоде

key – ключ записи, *left* – индекс левой границы, *strArr* – массив строк, *right* – индекс правой границы

```
binarysearch(strArr, left, right, key)
 {
middle←0;
while (true) do
   middle←((left+right)/2);
   if (key < stoi(strArr[middle])) then right←middle-1;
   else if (key > std::stoi(strArr[middle])) then left←middle+1;
        else return middle;
        if (left> right) then return -1;
    od
 }
```

3.3. Код функции бинарного поиска

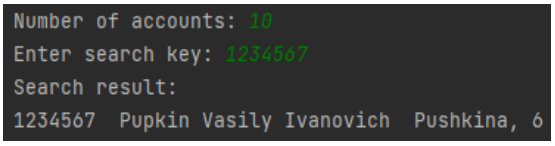
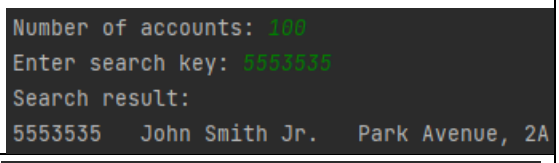
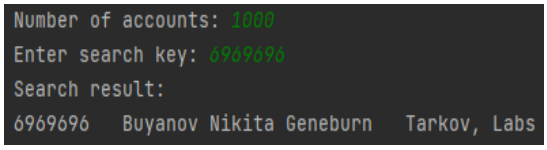
В функцию бинарного поиска передается таблица записей (массив со строками), индексы левой и правой границы, а также ключ, по которому ищется запись.

```
98  int binarysearch(string*& strArr, int left, int right, int key)
99  {
100     int middle = 0;
101     while (true)
102     {
103         middle = (left + right) / 2;
104         if (key < stoi(strArr[middle]))
105             right = middle - 1; else
106         if (key > stoi(strArr[middle]))
107             left = middle + 1;
108         else return middle;
109         if (left > right) return -1;
110     }
111 }
```

3.4. Результат тестирования программы для 100 записей

Для того чтобы убедиться, что функция бинарного поиска работает верно произведем тестирование функции (табл. 7).

Таблица 7. Тестирование функции бинарного поиска

Номер теста	Входные данные	Ожидаемый результат	Результат (вес файла)
1	Кол-во: 10 Ключ: 1234567	1234567 Pupkin Vasily Ivanovich Pushkina, 6	
2	Кол-во: 100 Ключ: 5553535	5553535 John Smith Jr. Park Avenue, 2A	
3	Кол-во: 1000 Ключ: 6969696	6969696 Buyanov Nikita Geneburn Tarkov, Labs	

Как видно из таблицы 7 – функция бинарного поиска работает верно инаходит запись согласно введенному ключу.

3.5. Таблица с замерами времени бинарного поиска записи

Чтобы оценить скорость работы функции бинарного 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.

Код программы:

```
98  int binarysearch(string *&strArr, int left, int right, int key)
99  {
100     int middle = 0;
101     auto begin = chrono::steady_clock::now();
102     while (true)
103     {
104         middle = (left + right) / 2;
105         if (key < stoi(strArr[middle])) right = middle - 1;
106         else if (key > stoi(strArr[middle])) left = middle + 1;
107         else
108         {
109             auto end = chrono::steady_clock::now();
110             auto elapsed_ms = chrono::duration_cast<chrono::microseconds>(end - begin);
111             cout << strArr[middle] << endl << "Time: " << elapsed_ms.count() << " mcs";
112             return 0;
113         }
114         if (left > right) return -1;
115     }
116 }
```

```
118 int main()
119 {
120     int amount, z;
121     cout << "Enter number of bank accounts: ";
122     cin >> amount;
123     string* strArr = new string[amount]{};
124     generatelist(amount, strArr);
125     sortlist(amount, strArr);
126     cout << "Enter search key: ";
127     cin >> z;
128     cout << "Search result:" << endl;
129     search(z, amount);
130     binarysearch(strArr, 0, 1000, z);
131 }
```

Таблица 8. Замеры времени бинарного поиска записи по ключу

Расположение записи	Время поиска в файле из 100 записей	Время поиска в файле из 1000 записей
Начало	8 мкс	7 мкс
Середина	5 мкс	8 мкс
Конец	11 мкс	14 мкс

Как видно из таблицы 8, время бинарного поиска не зависит от количества записей и их расположения линейно.

4. Анализ эффективности рассмотренных алгоритмов поиска в файле

Произведем анализ эффективности рассмотренных алгоритмов поиска в файле. Как видно из результатов замера времени для каждого алгоритма (табл. 5, табл. 8) – алгоритм бинарного поиска с использованием дополнительной памяти гораздо быстрее, чем алгоритм линейного поиска. Алгоритм бинарного поиска меньше зависит от длины массива и во многих частных случаях показывает себя быстрее (если элемент расположен в самой середине списка).

Таблица 9. Сравнение результатов

Кол-во записей	Алгоритм поиска	Расположение		
		Начало	Середина	Конец
100	Линейный	123 мкс	742 мкс	1659 мкс
	Бинарный	8 мкс	5 мкс	11 мкс
1000	Линейный	1562 мкс	6132 мкс	11585 мкс
	Бинарный	7 мкс	8 мкс	14 мкс

ВЫВОДЫ

В ходе данной практической работы были получены знания и практические навыки по разработке и реализации алгоритмов поиска в таблице. В первом задании был разработан генератор таблицы данных в соответствии с вариантом индивидуального задания, описаны прототипы используемых функций и приведены результаты тестирования алгоритма. Во втором задании был разработан алгоритм линейного поиска, приведены результаты тестирования алгоритма и произведен замер времени для разных случаев. В третьем задании был разработан алгоритм бинарного поиска, приведены результаты тестирования алгоритма и произведен замер времени для разных случаев. На основе замеров времени были проанализированы результаты эффективности изученных алгоритмов (время их выполнения) и сделан вывод, что алгоритм бинарного поиска является наиболее эффективным, из всех представленных в данной работе алгоритмов.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Теоретический материал по структурам и алгоритмам обработки данных
2. Мейерс, С. Эффективный и современный C++[Текст] / Скотт Мейерс. – O’REILLY, 2016. – 306 с.