МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Институт №3 – «Системы управления, информатика и электроэнергетика»

Кафедра 307 – «Цифровые технологии и информационные системы»

Отчёт по практическому заданию №3

«Хэш-функции»

Выполнил:

Студент группы М3О-214Б-20

**Цатурьян Константин Артурович**

Принял:

Преподаватель

**Павлов Олег Владимирович**

Москва 2021

## Цель работы

Разработать программу для сравнения всевозможных вариантов строк по их хэш-значениям и посимвольно. Использовать предварительную генерацию строк с записью в файл. Алфавит – a..z, A..Z, 0..9. Сравнить времена обработки строк по хэш-значениям и посимвольно для длин строк от 1 до 6 включительно. Исследовать зависимость времени от длины строки и количества коллизий от длины строки.

## Описание алгоритма

Разработаем программу, которая может принимать пользовательский ввод и в зависимости от введенной команды выполнять одну из операций:

1. Генерация файлов
2. Сравнение по хэш-значениям
3. Сравнение посимвольно

Данные команды будут выполнены для далее введенного количества символов. При генерации файлов – создаются файлы .bin размером 1 Гб для всех вариантов строк. При сравнении по хэш-значениям считывается информация из файлов, заносится в оперативную память, после чего производятся сравнения. Аналогично для посимвольного сравнения.

Итак, алгоритм программы можно представить следующим образом:

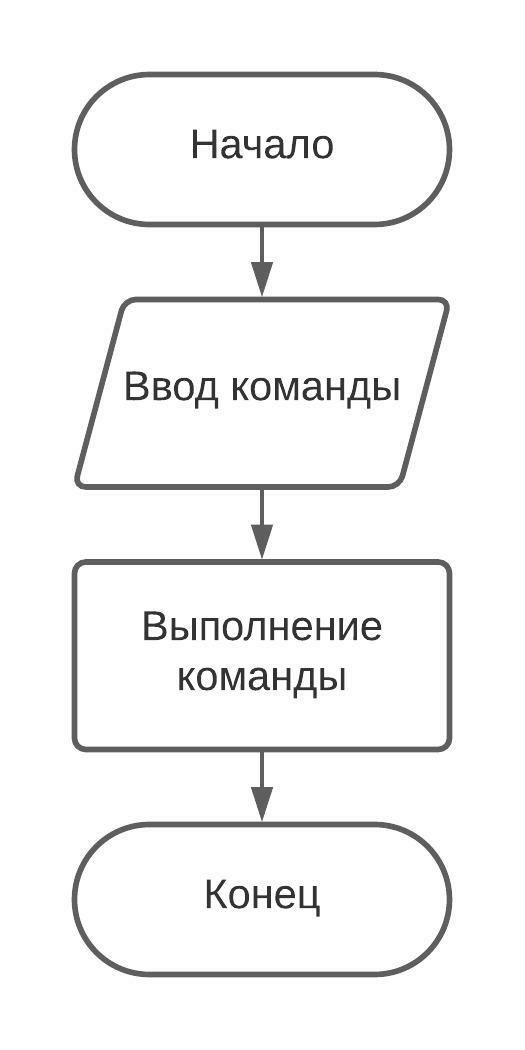
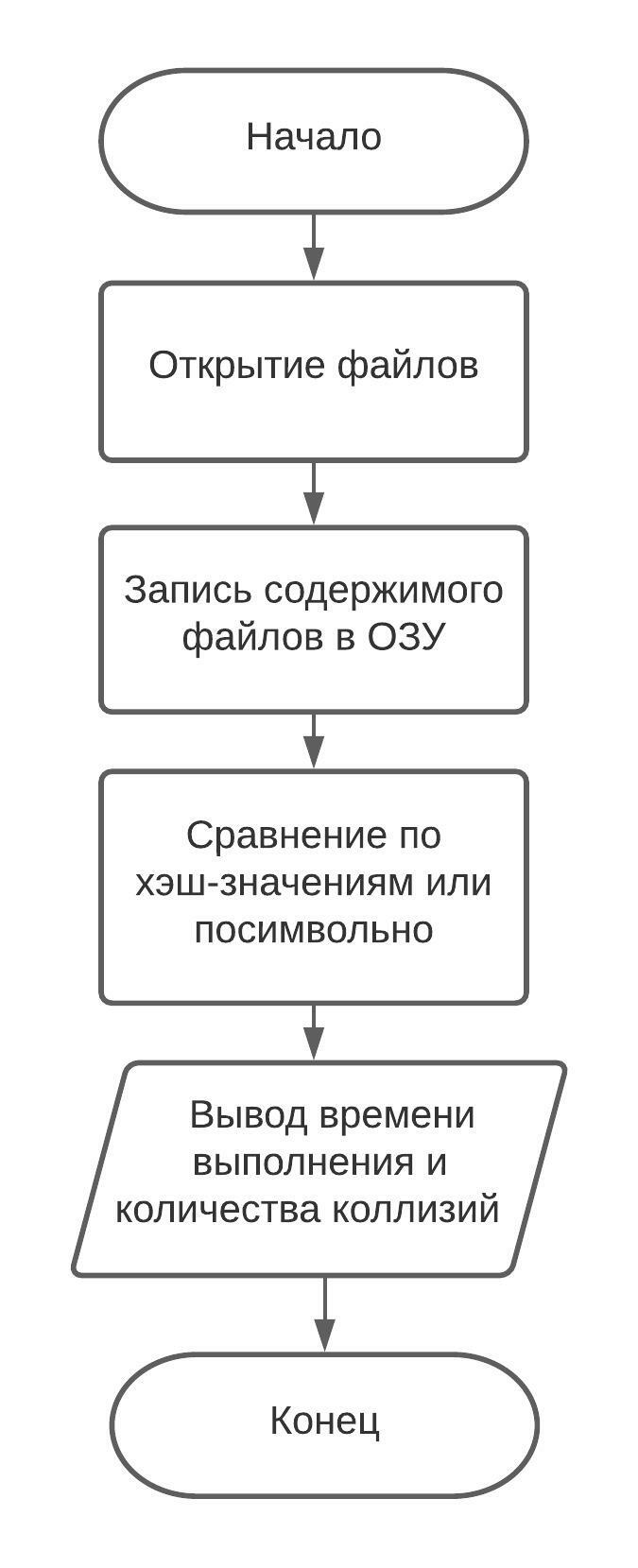


Рисунок – Общая блок-схема



|  |  |
| --- | --- |
| *Рисунок 2 – Блок-схема алгоритма генерации* | *Рисунок 3 – Блок-схема алгоритма сравнения* |

## Введение

Для начала оценим объем памяти, требуемый, чтобы сохранить все файлы, содержащие строки и их хэш-значения:

Каждый объект состоит из строки и её хэш-значения. Строка – набор символов (char), то есть каждый символ занимает ровно 1 байт. Хэш-значение – будем использовать тип int, который занимает 4 байта. Отсюда получена формула объема всех возможных строк:

,

где А – мощность алфавита (в данном случае 62 – количество уникальных символов), s – количество символов в строке.

Тогда общий объем в Гб будет выражен как:

Это значение является приблизительным, ведь в каждом файле хранится, как минимум, его заголовочная информация, то есть объем практический будет намного больше.

Оценим теоритическую зависимость времени сравнения от количества символов в строке. При сравнении хэш-значений нам требуется взять каждое хэш-значение и сравнить с каждым другим. То есть для s символов это будет сравнений. Или же , где альфа – некий постоянный коэффициент, отражающий скорость ЭВМ при доступе к памяти, сравнении хэш-значений и обработке результата.

Тогда

Аналогичная зависимость при посимвольном сравнении:

. Зависимость от s добавляется, так как при посимвольном сравнении для каждого сравнения мы должны пройти по всей строке и сравнить каждый символ. Это время зависит от скорости работы ЭВМ (бета, которое включает коэффициент альфа из предыдущего уравнения) и от количества символов в строке.

Тогда

Для хэширования используется полиномиальная функция:

где k, d – некие простые числа, k << d.

Оценка коллизий

Чтобы снизить объем хранимой информации было принято использовать .bin файлы. Так как системной единицей хранимой информации является 1 байт, то необходимо, чтобы объем хэш-значения и строки был кратен 1 байту. Действительно, int кодируется 4 байтами, а символьные значения – по 1 байту. Удобство использования binary файла заключается в том, что при записи пары хэш-значение–строка можно не использовать дополнительные символы – пробел, конец строки(‘\0’), перенос строки(‘\n’). А для того, чтобы корректно считать данные, нам необходимо всего лишь знать размер строки и считать 4 + s байт, где s – размер строки.

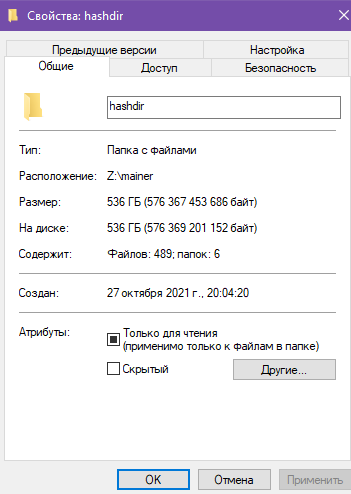


Рисунок 4 – Занимаемое место папкой со всеми файлами

Таким образом, данная конфигурация позволит сэкономить для 6 символов следующее количество памяти:

Итак, приступим к выполнению программы.

## Экспериментальные значения

Итак, после проведения оценки занимаемой памяти и оптимизации кода можно начать производить экспериментальные запуски программы:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 символ | 2 символа | 3 символа | 4 символа | 5 символов | 6 символов |
| t, мин | <<1 | <<1 | 0,003 | 0,1129 | 6,5814 | 159,0114 |
| W, MB | <<1 | <<1 | 1,5 | 38 | 7 683 | 540 627 |

Таблица – Время генерации файлов и их объем

Теперь проведем сравнение по хэш-значениям и посимвольно:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 символ | 2 символа | 3 символа | 4 символа | 5 символов | 6 символов |
| th, мин | 0 | 0,0004 | 1,3847 | 1 373,46 | - | - |
| tv, мин | 1,66\*10-5 | 0,0018 | 9,12 | - | - | - |
| k | 0 | 2 625 | 560 570 | 94 252 730 | - | - |

Таблица – Время сравнения по хэш-значениям и посимвольно

## Теоритические значения

Введем величину скорости сравнения: для каждой строки с номером i надо провести (As - i) сравнение. Тогда зная общее количество строк и среднее время, затрачиваемое на 1 сравнение, можно достаточно точно предсказать общие затраты времени.

Количество сравнений для строки из s символов:

как сумма арифметической прогрессии.

По полученным экспериментальным данным рассчитаем среднее время, требуемое для 1 сравнения:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Сравнение по хэш-значениям | | |
|  | 1 символ | 2 символа | 3 символа |
| t, клок | 0 | 5 | 15 592 |
| n | 1 891 | 7 386 246 | 28 399 998 628 |
| V, сравнений/клок | >>1891 | 1 477 249,2 | 1 821 446,8 |
|  | Сравнение посимвольно | | |
| t, клок | 1 | 109 | 547 406 |
| n | 1891 | 7 386 246 | 28 399 998 628 |
| V, сравнений/клок | ~1891 | 67 764 | 51 881 |

Итак, заметим, что измерение скорости сравнения тем точнее, чем больше было проведено сравнений, поэтому исключим из расчетов значение скорости для одного символа, тогда

Тогда найдем теоритическое время для сравнения строк длиной 4..6 символов:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 4 символа | 5 символов | 6 символов |
| n / 109 | 109 170,045 | 419 649 682, 476 | 1 613 133 381 170, 549 |
| th, мин | 1 302,73 | 4 238 890 | 16 294 276 577 |
| ts, мин | 30 325 | 116 569 360 | 448 092 605 880 |

Так как мы отбрасываем некоторые цифры числа, которые не превосходят 109, то погрешность расчетов составляет около

для сравнений по хэш-значениям.

для сравнений посимвольно.

P.S. примерная зависимость времени f(s) = 622 \* f(s-1) для обоих случаев. Также время сравнения хэш-значений меньше времени посимвольного сравнения примерно в s2 раз, где s – количество символов в строке.

Для построения графиков и поиска зависимостей будем пользоваться теоритическими значениями. Рассчитаем теперь теоритическую зависимость количества коллизий от количества символов. Это достаточно просто.

Зависимость коллизий

Итак, тогда конечная таблица имеет вид:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 символ | 2 символа | 3 символа | 4 символа | 5 символов | 6 символов |
| th, час | 0 | 1,2\*10-6 | 0,023 | 23,37 | 70 648 | 2,71 \* 108 |
| tv, час | 2,8\*10-7 | 3\*10-5 | 0,152 | 505,41 | 1 942 822 | 7,46 \* 109 |
| k | 0 | 2 625 | 560 570 |  | - | - |

Итак, на основе полученных данных были построены графики:

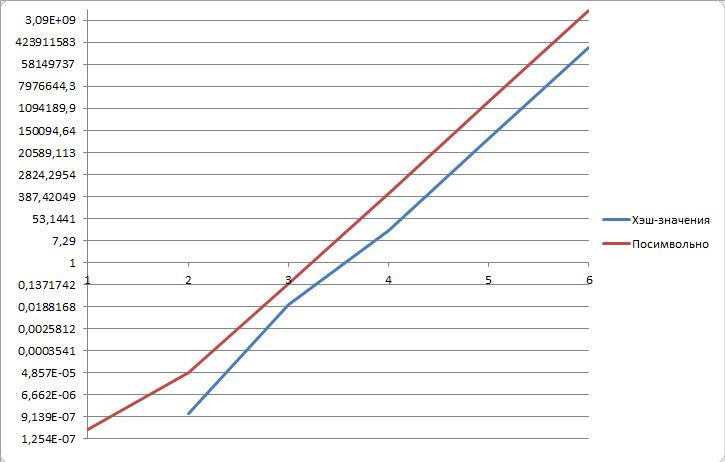


Рисунок 5 – Зависимость времени от количества символов

Отсюда можно сделать вывод, что зависимость времени от количества символов – экспоненциальная.

## Вывод

Была разработана программа, выполняющая генерацию строк и их хэш-значений с записью их в файл, сравнение значений в файлах и вывод затраченного времени в консоль.

## Исходный код

Source.cpp

1. #include <iostream>
2. #include <string>
3. #include "FileAccesser.h"
4. #include "RamAccesser.h"
6. int main(int argc, char\* argv[])
7. {
8. try
9. {
10. string current\_path(argv[0]);
11. while (current\_path.back() != '**\\**') current\_path.pop\_back();
13. bool f = true;
14. char ans, ln;
15. while (f)
16. {
17. f = false;
18. system("cls");
19. cout << "Welcome to the hashzone!" << endl << endl;
20. cout << "1. Generate files" << endl;
21. cout << "2. Compare hashes via disk (not recommended)" << endl;
22. cout << "3. Compare hashes via RAM" << endl;
23. cout << "4. Compare strings via RAM" << endl;
24. //more options
25. cout << "6. DEBUG reading" << endl << endl << ">>";
27. cin >> ans;
28. cin.clear();
29. cin.sync();
30. cout << endl;
32. switch (int(ans - '0'))
33. {
34. case 1:
35. {
36. cout << "Enter length of the string(1..6): ";
37. cin >> ln;
38. cin.clear();
39. cin.sync();
40. cout << endl;
42. ln = ln - '0';
43. if (ln >= 1 && ln <= 6)
44. {
45. auto t\_st = clock();
46. generate\_files(current\_path, int(ln));
47. auto t\_en = clock();
49. cout << "Generation of the files for " << int(ln) << " symbols took " << t\_en - t\_st << " ticks" << endl;
50. cout << "Current TPS is " << CLOCKS\_PER\_SEC << endl << endl;
51. }
52. else cout << "Wrong length!" << endl;
54. break;
55. }
57. case 2:
58. {
59. cout << "Enter length of the string(1..6): ";
60. cin >> ln;
61. cin.clear();
62. cin.sync();
63. cout << endl;
65. ln = ln - '0';
66. if (ln >= 1 && ln <= 6)
67. {
68. clock\_t dt = compare\_hash(current\_path, int(ln));
69. cout << "\nComparison of the files for " << int(ln) << " symbols took " << dt << " ticks" << endl;
70. cout << "Current TPS is " << CLOCKS\_PER\_SEC << endl << endl;
71. }
72. else cout << "Wrong length!" << endl;
74. break;
75. }
77. case 3:
78. {
79. cout << "Enter length of the string(1..6): ";
80. cin >> ln;
81. cin.clear();
82. cin.sync();
83. cout << endl;
85. ln = ln - '0';
86. if (ln >= 1 && ln <= 6)
87. {
88. clock\_t dt = hash\_ram(current\_path, int(ln));
90. cout << "\nComparison(hash) of the files for " << int(ln) << " symbols took " << dt << " ticks" << endl;
91. cout << "Current TPS is " << CLOCKS\_PER\_SEC << endl << endl;
92. }
93. else cout << "Wrong length!" << endl;
95. break;
96. }
98. case 4:
99. {
100. cout << "Enter length of the string(1..6): ";
101. cin >> ln;
102. cin.clear();
103. cin.sync();
104. cout << endl;
106. ln = ln - '0';
107. if (ln >= 1 && ln <= 6)
108. {
109. clock\_t dt = str\_ram(current\_path, int(ln));
111. cout << "**\n**Comparison(string) of the files for " << int(ln) << " symbols took " << dt << " ticks" << endl;
112. cout << "Current TPS is " << CLOCKS\_PER\_SEC << endl << endl;
113. }
114. else cout << "Wrong length!" << endl;
116. break;
117. }
119. case 6:
120. {
121. cout << "Enter length of the string(1..6): ";
122. cin >> ln;
123. cin.clear();
124. cin.sync();
125. cout << endl;
127. ln = ln - '0';
129. debug\_read(current\_path, int(ln));
130. break;
131. }
133. default:
134. {
135. cout << "Fatal error. Wrong command" << endl;
136. cout << "Press any key to try again..." << endl;
137. system("pause");
139. f = true;
140. }
141. }
142. }
143. }
144. catch (exception e)
145. {
146. cout << "Fatal error: " << e.what() << endl;
147. system("pause");
148. return -1;
149. }
151. system("pause");
152. return 0;
153. }

FileAccesser.h

1. #pragma once
2. #include <fstream>
3. #include <string>
4. using namespace std;
6. //for 1GB file
7. const int STRINGS\_PER\_FILE = 119304647;
9. bool IsPathExist(const string& path\_to\_check);
11. const int k = 31, mod = 1000000007;
12. int encrypt(const char\*);
14. void generate\_files(string dir\_path, int length);
16. clock\_t compare\_hash(string dir\_path, int length);
18. void debug\_read(string dir\_path, int len, int file\_n = 0);
20. void print\_time();

FileAccesser.cpp

1. #include <iostream>
2. #include <fstream>
3. #include <string>
4. #include <sys/stat.h>
5. #include <direct.h>
6. #include <time.h>
7. using namespace std;
9. const int ALPHABET\_POWER = 62;
11. int LENGTH;
12. long long unsigned int NUMBER\_OF\_STRINGS;
14. //for 1GB file
15. const int STRINGS\_PER\_FILE = 119304647;
17. bool IsPathExist(const string& s)
18. {
19. struct stat buffer;
20. return (stat(s.c\_str(), &buffer) == 0);
21. }
23. const int k = 31, mod = 1000000007;
24. int encrypt(const char\* str)
25. {
26. int hash = 0;
27. for (int i = 0; i < LENGTH; i++)
28. hash = (hash \* k + str[i]) % mod;
30. return hash;
31. }
33. void generate\_files(string dir\_path, int len)
34. {
35. LENGTH = len;
36. NUMBER\_OF\_STRINGS = pow(ALPHABET\_POWER, LENGTH);
38. dir\_path += "hashdir";
40. if (!IsPathExist(dir\_path))
41. if(\_mkdir(dir\_path.c\_str()) != 0) throw exception("Error while creating main directory");
43. char\* buf = new char[len];
44. for (int i = 0; i < len; i++) buf[i] = '0';
46. char\* ibuf = new char[4];
47. int hash;
49. int cnt = 0;
50. int file\_cnt = 0;
52. string path = dir\_path + "**\\**test\_" + to\_string(len);
54. if (!IsPathExist(path))
55. if (\_mkdir(path.c\_str()) != 0) throw exception("Error while creating sub directory");
57. path += "**\\**file";
59. fstream gen\_file(path + to\_string(file\_cnt) + ".bin", ios::out | ios::binary);
61. while (buf[len - 1] <= 'z')
62. {
63. if (cnt == STRINGS\_PER\_FILE)
64. {
65. cnt = 0;
66. file\_cnt++;
67. gen\_file.close();
68. gen\_file = fstream(path + to\_string(file\_cnt) + ".bin", ios::out | ios::binary);
69. }
71. gen\_file.write(buf, len);
73. hash = encrypt(buf);
74. memcpy(ibuf, &hash, 4);
75. gen\_file.write(ibuf, 4);
77. cnt++;
78. buf[0]++;
80. for (int i = 0; i < len; i++)
81. {
82. if (buf[i] == '9' + 1) buf[i] = 'A';
83. else if (buf[i] == 'Z' + 1) buf[i] = 'a';
84. else if (buf[i] == 'z' + 1 && i != len - 1)
85. {
86. buf[i] = '0';
87. buf[i + 1]++;
88. }
89. else break;
90. }
91. }
93. gen\_file.close();
94. delete[] buf;
95. }
97. void print\_time()
98. {
99. auto tstmp = time(NULL);
100. auto tm = localtime(&tstmp);
101. cout << tm->tm\_hour << ':' << tm->tm\_min << ':' << tm->tm\_sec << ": ";
102. }
104. clock\_t compare\_hash(string dir\_path, int len)
105. {
106. LENGTH = len;
107. NUMBER\_OF\_STRINGS = long long int(pow(ALPHABET\_POWER, LENGTH)) % STRINGS\_PER\_FILE;
109. int collisions = 0;
110. string path = dir\_path + "\\hashdir\\test\_" + to\_string(len);
112. if (!IsPathExist(path))
113. throw exception("Directory not found");
115. path += "\\file";
117. int max\_file = 0;
118. while (IsPathExist(path + to\_string(max\_file + 1) + ".bin")) max\_file++;
120. char\* buf = new char[len]; char\* t\_buf = new char[len];
121. char\* ibuf = new char[4]; char\* t\_ibuf = new char[4];
122. int hs, t\_hs;
124. print\_time();
125. cout << "Init successful! Staring comparison for " << len << "-symbols..." << endl;
127. auto t\_st = clock();
129. //for each file
130. for (int i = 0; i <= max\_file; i++)
131. {
132. fstream f\_file(path + to\_string(i) + ".bin", ios::in | ios::binary);
133. int tmp = ((i == max\_file) ? (NUMBER\_OF\_STRINGS) : (STRINGS\_PER\_FILE));
135. //compare inside first taken file
136. for (int j = 0; j < tmp; j++)
137. {
138. f\_file.seekg((len + 4) \* j, f\_file.beg);
140. f\_file.read(buf, len);
141. f\_file.read(ibuf, 4);
142. memcpy(&hs, ibuf, 4);
144. for (int j1 = j + 1; j1 < tmp; j1++)
145. {
146. f\_file.seekg((len + 4) \* j1, f\_file.beg);
148. f\_file.read(t\_buf, len);
149. f\_file.read(t\_ibuf, 4);
150. memcpy(&t\_hs, t\_ibuf, 4);
152. if (hs == t\_hs)
153. {
154. */\**
155. *//debug collision check*
156. *for (int g = 0; g < len; g++) cout << buf[g];*
157. *cout << ' ' << hs << " & ";*
158. *for (int g = 0; g < len; g++) cout << t\_buf[g];*
159. *cout << ' ' << t\_hs << endl << endl;*
160. *\*/*
161. collisions++;
162. }
163. }
164. }
166. print\_time();
167. cout << "first iteration for file" << i << " completed" << endl;
169. //for each next file
170. for (int i1 = i + 1; i1 <= max\_file; i1++)
171. {
172. fstream s\_file(path + to\_string(i1) + ".bin", ios::in | ios::binary);
173. int tmp1 = ((i1 == max\_file) ? (NUMBER\_OF\_STRINGS) : (STRINGS\_PER\_FILE));
175. f\_file.seekg(0, f\_file.beg);
177. //compare each str in file
178. for (int j = 0; j < tmp; j++)
179. {
180. f\_file.read(buf, len);
181. f\_file.read(ibuf, 4);
182. memcpy(&hs, ibuf, 4);
184. //with each str in file+1
185. for (int j1 = 0; j1 < tmp1; j1++)
186. {
187. s\_file.read(t\_buf, len);
188. s\_file.read(t\_ibuf, 4);
189. memcpy(&t\_hs, t\_ibuf, 4);
191. if (hs == t\_hs) collisions++;
192. }
193. }
195. print\_time();
196. cout << "second iteration for file" << i << " with file" << i1 << " completed" << endl;
197. s\_file.close();
198. }
200. print\_time();
201. cout << "file" << i << " processed..." << endl;
203. f\_file.close();
204. }
205. auto t\_en = clock();
207. cout << "**\n**Found collisions: " << collisions << endl;
209. return (t\_en - t\_st);
210. }




216. void debug\_read(string path, int len, int f = 0)
217. {
218. LENGTH = len;
219. NUMBER\_OF\_STRINGS = pow(ALPHABET\_POWER, LENGTH);
221. cout << "Reading file" << f << " from " << len << "-symbols. Press any key to confirm" << endl;
222. cin.clear();
223. cin.sync();
224. getchar();
226. path += "**\\**hashdir**\\**test\_" + to\_string(len) + "**\\**file" + to\_string(f) + ".bin";
228. fstream read\_file(path, ios::in | ios::binary);
230. char\* buf = new char[len];
231. char\* ibuf = new char[4];
232. int hs;
234. for (int k = 0; k < STRINGS\_PER\_FILE && k < NUMBER\_OF\_STRINGS; k++)
235. {
236. read\_file.read(buf, len);
237. read\_file.read(ibuf, 4);
238. memcpy(&hs, ibuf, 4);
240. for (int i = 0; i < len; i++) cout << buf[i];
241. cout << ' ' << hs << endl;
242. }
244. read\_file.close();
245. }

RamAccesser.h

1. #pragma once
2. #include <string>
3. #include <fstream>
4. using namespace std;
6. const int STRINGS\_PER\_RAM = 119304647;
7. bool IsPathExist(const string& path\_to\_check);
8. void print\_time();
10. time\_t hash\_ram(string dir\_path, int len);
11. time\_t str\_ram(string dir\_path, int len);
13. struct unit
14. {
15. char\* str;
16. int hash;
18. unit() = delete;
20. };
22. class ram\_unit
23. {
24. private:
25. int size;
26. unit\* cache;
27. int l;
29. public:
30. ram\_unit() = delete;
31. ram\_unit(string file\_path, int str\_in\_file);
32. int compare();
33. void compare\_str();
35. ~ram\_unit();
36. friend int operator+(const ram\_unit& ru1, const ram\_unit& ru2);
37. friend void operator\*(const ram\_unit& ru1, const ram\_unit& ru2);
38. };

RamAccesser.cpp

1. #include <string>
2. #include <fstream>
3. #include <iostream>
4. using namespace std;
6. const int STRINGS\_PER\_RAM = 119304647;
7. bool IsPathExist(const string& path\_to\_check);
8. void print\_time();
10. struct unit
11. {
12. char\* str;
13. int hash;
14. };
16. class ram\_unit
17. {
18. private:
19. int size;
20. unit\* cache;
21. int l;
23. public:
24. ram\_unit() = delete;
25. ram\_unit(string path, int nos, int len)
26. {
27. l = len;
28. size = nos;
29. fstream f\_file(path, ios::in | ios::binary);
31. cache = new unit[size];
33. char\* ibuf = new char[4];
35. for (int i = 0; i < size; i++)
36. {
37. cache[i].str = new char[len];
39. f\_file.read(cache[i].str, len);
40. f\_file.read(ibuf, 4);
41. memcpy(&cache[i].hash, ibuf, 4);
42. }
44. f\_file.close();
45. }
47. void compare\_str()
48. {
49. int colls = 0;
51. for (int i = 0; i < size; i++)
52. for (int j = i + 1; j < size; j++)
53. for (int k = 0; k < l; k++)
54. if (cache[i].str[k] == cache[j].str[k]) colls++;
55. }
57. int compare()
58. {
59. int colls = 0;
61. for (int i = 0; i < size; i++)
62. for (int j = i + 1; j < size; j++)
63. if (cache[i].hash == cache[j].hash) colls++;
65. return colls;
66. }
68. ~ram\_unit()
69. {
70. delete[] cache;
71. }
73. friend int operator+ (const ram\_unit& ru1, const ram\_unit& ru2)
74. {
75. int colls = 0;
77. for (int i = 0; i < ru1.size; i++)
78. for (int j = 0; j < ru2.size; j++)
79. if (ru1.cache[i].hash == ru2.cache[j].hash) colls++;
81. return colls;
82. }
84. friend void operator\* (const ram\_unit& ru1, const ram\_unit& ru2)
85. {
86. int colls = 0;
88. for (int i = 0; i < ru1.size; i++)
89. for (int j = 0; j < ru2.size; j++)
90. for (int k = 0; k < ru1.l; k++)
91. if (ru1.cache[i].str[k] == ru2.cache[j].str[k]) colls++;
92. }
93. };

96. time\_t hash\_ram(string dir\_path, int len)
97. {
98. long long int NUMBER\_OF\_STRINGS = pow(62, len);
99. long long int collisions = 0;
100. string path = dir\_path + "\\hashdir\\test\_" + to\_string(len);
102. path = "d:**\\**hashdir**\\**test\_" + to\_string(len); //!!!!!!!
104. if (!IsPathExist(path))
105. throw exception("Directory not found");
107. path += "**\\**file";
109. int max\_file = 0;
110. while (IsPathExist(path + to\_string(max\_file + 1) + ".bin")) max\_file++;
112. print\_time();
113. cout << "Init successful! Staring comparison for " << len << "-symbols..." << endl;
115. auto t\_st = clock();
117. for (int i = 0; i <= max\_file; i++)
118. {
119. int tmp = ((i == max\_file) ? (NUMBER\_OF\_STRINGS % STRINGS\_PER\_RAM) : (STRINGS\_PER\_RAM));
121. ram\_unit rm(path + to\_string(i) + ".bin", tmp, len);
123. //compare with file self
124. collisions += rm.compare();
126. //compare with others
127. for (int i1 = i + 1; i1 <= max\_file; i1++)
128. {
129. int tmp1 = ((i1 == max\_file) ? (NUMBER\_OF\_STRINGS % STRINGS\_PER\_RAM) : (STRINGS\_PER\_RAM));
131. ram\_unit rm1(path + to\_string(i1) + ".bin", tmp1, len);
133. collisions += (rm + rm1);
134. }
136. print\_time();
137. cout << "Second iteration for file" << i << " completed" << endl;
138. }
140. auto t\_en = clock();
142. cout << "**\n**Found collisions: " << collisions << endl;
144. return (t\_en - t\_st);
145. }
147. time\_t str\_ram(string dir\_path, int len)
148. {
149. long long int NUMBER\_OF\_STRINGS = pow(62, len);
150. string path = dir\_path + "\\hashdir\\test\_" + to\_string(len);
152. path = "d:**\\**hashdir**\\**test\_" + to\_string(len); //!!!!!!!
154. if (!IsPathExist(path))
155. throw exception("Directory not found");
157. path += "**\\**file";
159. int max\_file = 0;
160. while (IsPathExist(path + to\_string(max\_file + 1) + ".bin")) max\_file++;
162. print\_time();
163. cout << "Init successful! Staring comparison for " << len << "-symbols..." << endl;
165. auto t\_st = clock();
167. for (int i = 0; i <= max\_file; i++)
168. {
169. int tmp = ((i == max\_file) ? (NUMBER\_OF\_STRINGS % STRINGS\_PER\_RAM) : (STRINGS\_PER\_RAM));
171. ram\_unit rm(path + to\_string(i) + ".bin", tmp, len);
173. //compare with file self
174. rm.compare\_str();
176. //compare with others
177. for (int i1 = i + 1; i1 <= max\_file; i1++)
178. {
179. int tmp1 = ((i1 == max\_file) ? (NUMBER\_OF\_STRINGS % STRINGS\_PER\_RAM) : (STRINGS\_PER\_RAM));
181. ram\_unit rm1(path + to\_string(i1) + ".bin", tmp1, len);
183. rm \* rm1;
184. }
186. print\_time();
187. cout << "Second iteration for file" << i << " completed" << endl;
188. }
190. auto t\_en = clock();
192. return (t\_en - t\_st);
193. }