ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший преподаватель |  |  |  | С. А. Рогачев |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ |
| «ХЕШИРОВАНИЕ ДАННЫХ» |
| по курсу: СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

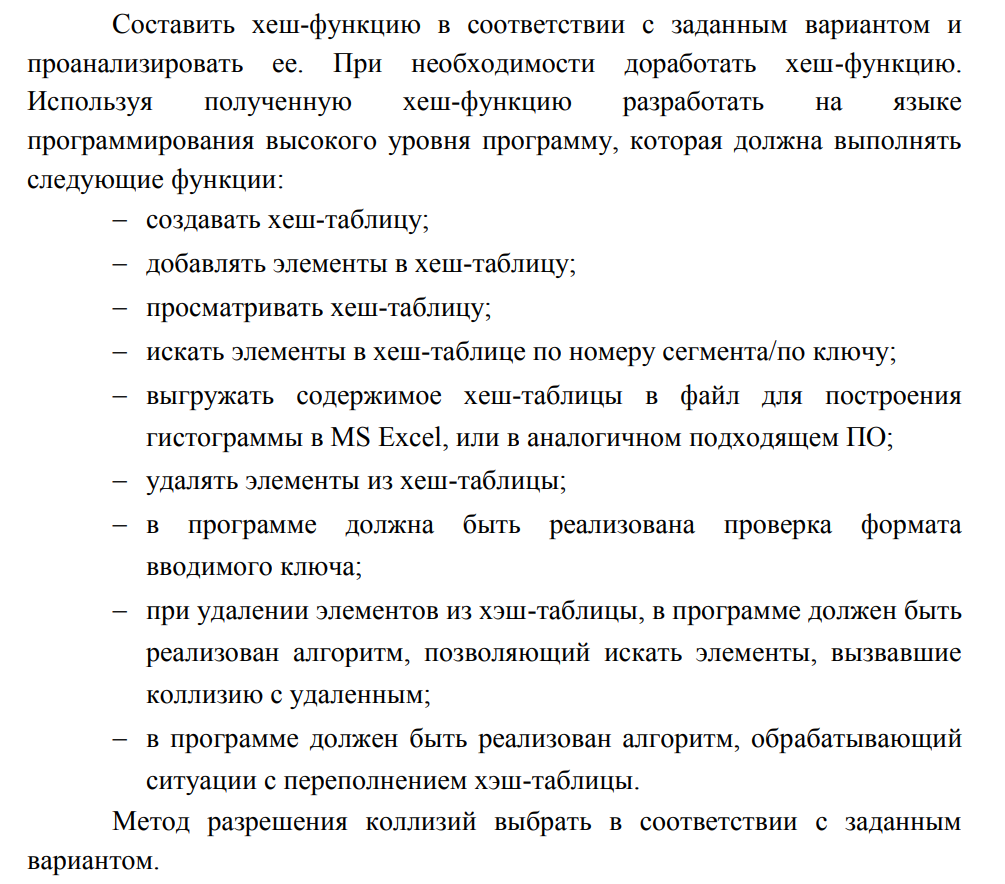
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4131 |  |  |  | Д. А. Кузнецов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

**Цель работы:** Целью работы является изучение методов хеширования данных и получение практических навыков реализации хеш-таблиц.

**Вариант:** 4





**Листинг программы:**

**Main.cpp**

1. // Var 18
3. #include <iostream>
4. #include <string>
6. #include "HashTable.h"
8. using namespace std;
10. int main() {
11. HashTable table(10);
13. table.add("70CK69", "First key");
14. table.add("13ML12", "Second key");
16. table.show();
18. table.add("13ML12", "Second key");
20. table.show();
22. table.remove("70CK69");
24. table.show();
26. cout << "\n\n";
27. table.show("13ML12");
29. // Collision allocation
31. HashTable test\_table(2000);
33. for (int i = 0; i < 4000; i++) {
34. string key = generateKey(getLettersAlphabet(), getNumbersList());
36. test\_table.add(key, "-------");
37. }
39. test\_table.show();
41. allocCollisions(test\_table.getCollArr(), 2000);
42. }

**Hashing.cpp**

1. #include <iostream>
2. #include <fstream>
3. #include <string>
5. #include "HashTable.h"
7. using namespace std;
9. // Generating random key
10. string generateKey(string\* letters, string\* numbers) {
12. string temp\_str = "";
14. for (int k = 0; k < 6; k++) {
15. if (k < 2 || k > 3) {
16. temp\_str += numbers[std::rand() % 10];
17. }
18. else {
19. temp\_str += letters[std::rand() % 26];
20. }
21. }
23. return temp\_str;
24. }
26. // Get hash key from string and amount of segments
27. int getHashKey(string s, int seg\_amt) {
28. int temp = 0;
30. for (int i = 0; i < s.length(); i++) {
31. temp += s[i] \* s[i];
32. }
33. temp %= seg\_amt;
35. return temp;
36. }
38. string\* getLettersAlphabet() {
39. string\* letters = new string[26];
41. char temp = 'A';
43. for (int i = 0; i < 26; i++) {
44. letters[i] = temp++;
45. }
47. return letters;
48. }
50. string\* getNumbersList() {
51. string\* numbers = new string[10];
53. char temp = '0';
55. for (int i = 0; i < 10; i++) {
56. numbers[i] = temp++;
57. }
59. return numbers;
60. }
62. // Allocating collisions for 2x amount of keys to segments and output to Excel
63. void allocCollisions(int\* coll\_array, int size) {
64. ofstream out;
65. out.open("D:\\Другое\\C++ Scripts\\Structs\_And\_Algo\_SUAI\\Lab\_4\\Kollision\_Allocation.csv");
67. for (int i = 0; i < size; i++) {
68. out << coll\_array[i] << "\t";
69. }
71. out.close();
72. }
74. bool checkKey(string key) {
75. if (key.length() != 6)
76. return false;
78. if (key[0] >= '0' && key[0] <= '9' &&
79. key[1] >= '0' && key[1] <= '9' &&
80. key[2] >= 'A' && key[2] <= 'Z' &&
81. key[3] >= 'A' && key[3] <= 'Z' &&
82. key[4] >= '0' && key[4] <= '9' &&
83. key[5] >= '0' && key[5] <= '9')
84. {
85. return true;
86. }
87. else
88. {
89. return false;
90. }
91. }

**HashTable.cpp**

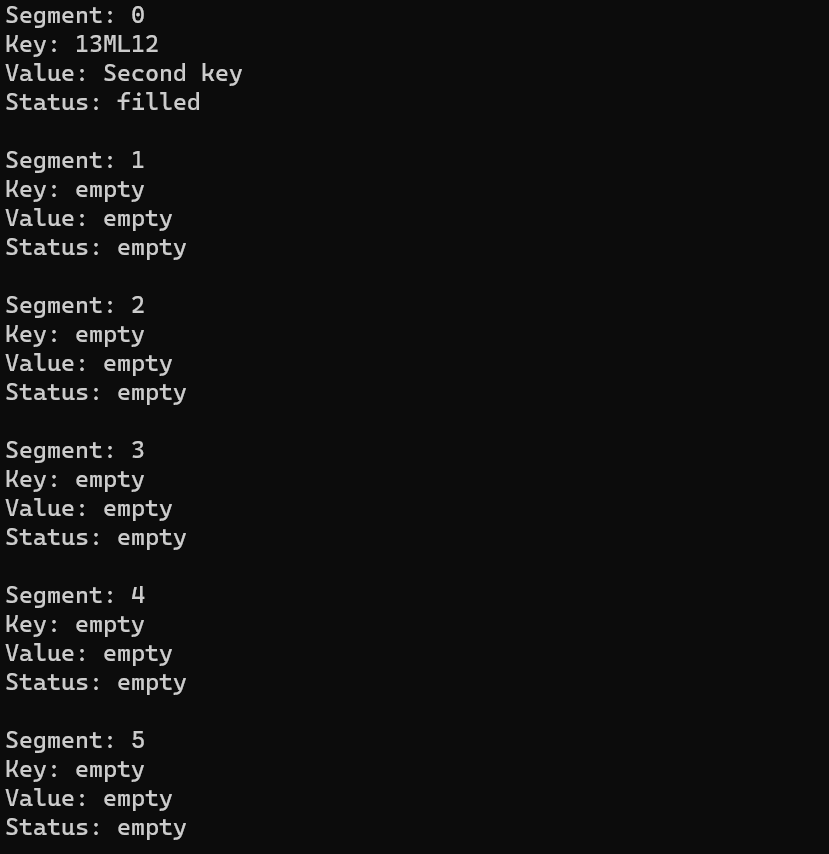
1. #include <iostream>
2. #include <string>
4. #include "HashTable.h"
6. using namespace std;
8. HashTable::HashTable(int size) {
9. table = new HashElem[size];
10. table\_size = size;
11. collisions = new int[size];
13. for (int i = 0; i < size; i++) {
14. collisions[i] = 0;
15. }
16. }
18. bool HashTable::isEmpty(int ind) {
19. if (table[ind].status == "empty" || table[ind].status == "deleted")
20. return true;
21. else
22. return false;
23. }
25. bool HashTable::isExist(int ind, string key) {
26. if (table[ind].status == "filled" && table[ind].key == key)
27. return true;
28. else
29. return false;
30. }
32. void HashTable::add(string key, string value) {
34. if (!checkKey(key)) {
35. cout << "\nKey: " << key << " - Wrong key format!\n\n";
36. return;
37. }
39. int ind;
40. ind = getHashKey(key, table\_size);
42. collisions[ind] += 1;
44. if (isExist(ind, key)) {
45. cout << "\nKey: " << key << " - This key value is already exist!\n\n";
47. return;
48. }
49. else if (isEmpty(ind)) {
50. table[ind].value = value;
51. table[ind].status = "filled";
52. table[ind].key = key;
53. }
54. else {
55. int attemp = 0;
56. int c = 1;
57. int attemp\_ind = ind;
58. bool f = 0;
60. while (attemp < table\_size/2) {
62. // Linear sampling step
63. attemp\_ind += 3 \* c;
65. if (attemp\_ind > table\_size - 1) {
66. attemp\_ind = attemp;
67. attemp += 1;
68. c = attemp;
69. }
71. if (isExist(attemp\_ind, key)) {
72. cout << "\n\nKey: " << key << " - this key value is already exist!\n\n";
73. return;
74. }
75. else if (isEmpty(attemp\_ind)) {
76. table[attemp\_ind].value = value;
77. table[attemp\_ind].status = "filled";
78. table[attemp\_ind].key = key;
80. f = 1;
81. break;
82. }
83. }
85. if (f == 0) {
86. cout << "Trying key: " << key << " - ";
87. cout << "Hash table overflow!\n\n";
88. }
89. }
90. }
92. void HashTable::remove(string key) {
93. if (!checkKey(key)) {
94. cout << "\nKey: " << key << " - Wrong key format!\n\n";
95. return;
96. }
98. int ind;
99. ind = getHashKey(key, table\_size);
101. if (table[ind].status == "filled") {
102. table[ind].status = "deleted";
103. table[ind].value = "empty";
104. table[ind].key = "empty";
106. cout << "\n'" << key << "' - was removed succesfully!\n\n";
107. }
108. else {
109. int attemp = 0;
110. int c = 1;
111. int attemp\_ind = ind;
112. bool f = 0;
114. while (attemp < table\_size / 2) {
116. // Linear sampling step
117. attemp\_ind += 3 \* c;
119. if (attemp\_ind > table\_size - 1) {
120. attemp\_ind = attemp;
121. attemp += 1;
122. c = attemp;
123. }
125. if (table[attemp\_ind].status == "filled" && table[attemp\_ind].key == key) {
126. table[attemp\_ind].status = "deleted";
127. table[attemp\_ind].value = "empty";
128. table[attemp\_ind].key = "empty";
130. cout << "\n\n\'" << key << "' - was removed succesfully!\n\n";
132. f = 1;
133. break;
134. }
136. }
138. if (f == 0) {
139. cout << "Deleting key: " << key << " - ";
140. cout << "This key value does not exist!\n\n";
141. }
142. }
143. }
145. HashElem HashTable::get(string key) {
147. HashElem temp;
149. if (!checkKey(key)) {
150. cout << "\nKey: " << key << " - Wrong key format!\n\n";
151. return temp;
152. }
154. int ind;
155. ind = getHashKey(key, table\_size);
157. if (table[ind].status == "filled" && table[ind].key == key) {
158. return table[ind];
159. }
160. else {
161. int attemp = 0;
162. int c = 1;
163. int attemp\_ind = ind;
164. bool f = 0;
166. while (attemp < table\_size / 2) {
168. // Linear sampling step
169. attemp\_ind += 3 \* c;
171. if (attemp\_ind > table\_size - 1) {
172. attemp\_ind = attemp;
173. attemp += 1;
174. c = attemp;
175. }
177. if (table[attemp\_ind].status == "filled" && table[attemp\_ind].key == key) {
178. return table[attemp\_ind];
179. f = 1;
181. break;
182. }
184. }
185. if (f == 0) {
186. cout << "Key: " << key << " - Element doesn't exist!\n\n";
187. }
188. }
190. return temp;
191. }

194. HashElem HashTable::get(int ind) {
195. return table[ind];
196. }
198. void HashTable::show() {
199. for (int i = 0; i < table\_size; i++) {
200. HashElem temp = table[i];
202. cout << "Segment: " << i << "\n";
203. cout << "Key: " << temp.key << "\n";
204. cout << "Value: " << temp.value << "\n";
205. cout << "Status: " << temp.status << "\n\n";
206. }
207. }
209. void HashTable::show(string key) {
210. show(get(key));
211. }
213. void HashTable::show(int i) {
214. HashElem temp = table[i];
216. cout << "Segment: " << i << "\n";
217. cout << "Key: " << temp.key << "\n";
218. cout << "Value: " << temp.value << "\n";
219. cout << "Status: " << temp.status << "\n\n";
220. }
222. void HashTable::show(HashElem elem) {
224. cout << "Key: " << elem.key << "\n";
225. cout << "Value: " << elem.value << "\n";
226. cout << "Status: " << elem.status << "\n\n";
227. }
229. int\* HashTable::getCollArr() {
230. return collisions;
231. }
233. HashTable::~HashTable() {
234. delete[] table;
235. delete[] collisions;
236. }

**HashTable.h**

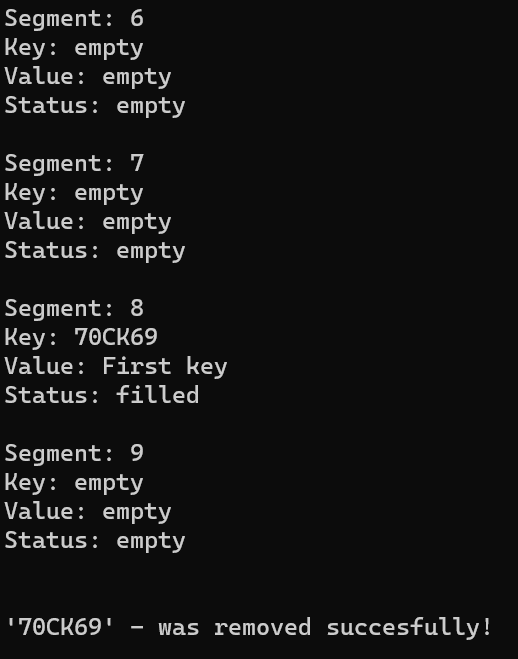
1. #pragma once
3. #include <string>
5. using namespace std;
7. string generateKey(string\* letters, string\* numbers);
9. int getHashKey(string s, int seg\_amt);
11. string\* getLettersAlphabet();
12. string\* getNumbersList();
14. void allocCollisions(int\* coll\_array, int size);
16. bool checkKey(string key);
18. struct HashElem {
19. string key = "empty";
20. string value = "empty";
22. string status = "empty";
23. };
25. class HashTable {
26. HashElem\* table;
27. int table\_size;
29. bool isEmpty(int ind);
30. bool isExist(int ind, string key);
32. int\* collisions;
34. public:
36. HashTable(int size);
38. void add(string key, string value);
39. void remove(string key);
40. HashElem get(string key);
41. HashElem get(int ind);
43. void show();
44. void show(string key);
45. void show(int seg);
46. void show(HashElem elem);
47. int\* getCollArr();
49. ~HashTable();
50. };

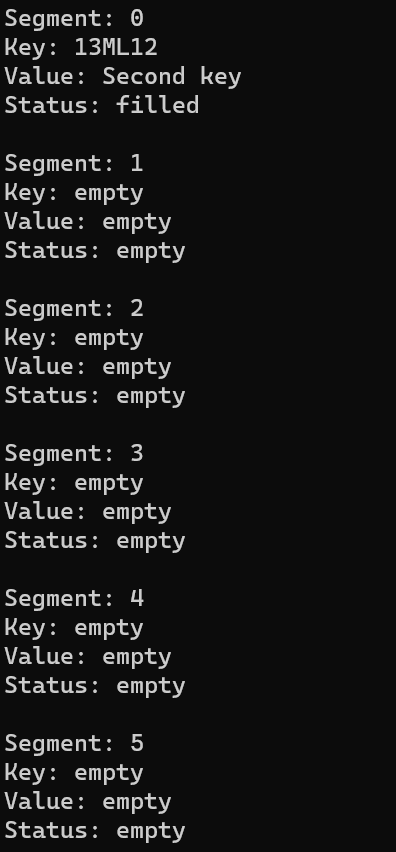
**Контрольные примеры:**

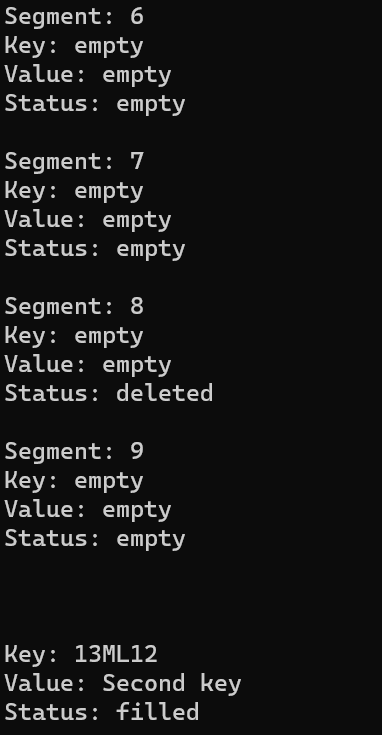
****

****

****

****

****

****

**Вывод:**

В ходе лабораторной работы я изучил методы хеширования данных и полученил практические навыкои реализации хеш-таблиц.