**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе №3**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: «Хэш-таблицы**»**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 7307 | Васильев А.В. Шалугин Е.Д, |  |
| Преподаватель | Колинько П.Г. |  |

Санкт-Петербург

2019

Оглавление

[Цель работы 3](#__RefHeading___Toc188_4150899979)

[Задание 3](#__RefHeading___Toc190_4150899979)

[Обоснование по выбору размера хэш-таблицы и коэффициентов хэш-функции 3](#__RefHeading___Toc302_2376616454)

[Оценка временной сложности 3](#__RefHeading___Toc967_306870553)

[Результаты работы программы 3](#__RefHeading___Toc969_306870553)

[Код программы 6](#__RefHeading___Toc306_2376616454)

[Вывод 9](#__RefHeading___Toc308_2376616454)

# **Цель работы**

Научиться работать с хэш-таблицами и придумывать хэш-функции.

# **Задание**

Составить и отладить программу для вычисления шестого множества по

пяти заданным, представленным в форме хэш-таблиц. Элементы множеств

— целые числа из интервала [0, 99].

Формула для вычислений: (A ∩ B) ⊕ C ∪ D \ E;

средняя мощность множества: 32.

# **Обоснование по выбору размера хэш-таблицы и коэффициентов хэш-функции**

Размер хэш-таблицы равен трехкратной мощности множества. Это обусловлено тем, что в таком случае вероятность коллизии меньше 0,33.

Хэш — функция выбрана такая: h(x) = (a \* x + b) % m,

где m – размер таблицы, а a и b — простые числа.

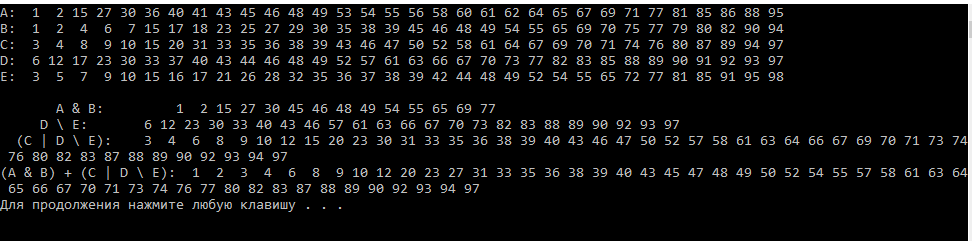
Мы взяли a = 97, b = 3, m = мощность множества \* 3. Такой выбор обеспечивает равномерное использование всех ячеек таблицы в большинстве практических случаев.

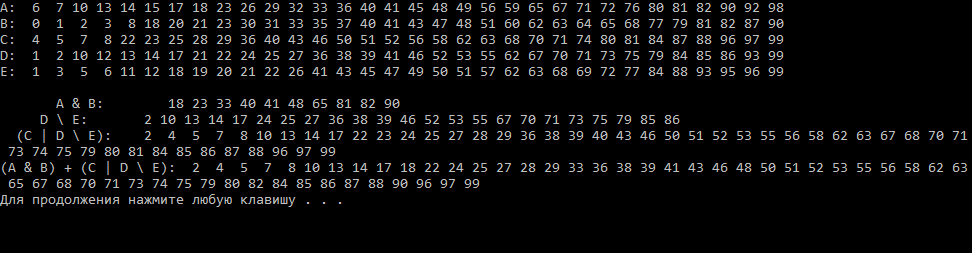
# **Оценка временной сложности**

Если таблица правильно построена и не переполнена, проверка принадлежности элемента множеству, а также вставка и удаление элемента выполняются в ней за постоянное время, примерно такое же, как и в массиве битов. За постоянное время будут выполняться и двуместные операции над множествами: объединение, пересечение и разность.

В худшем случае, когда все данные попадают в одну или несколько ячеек таблицы, образовав неупорядоченные списки, справедливы следующие оценки временной сложности: O(n) — для поиска и удаления элемента; O(n^2) — для двуместной операции над множествами.

# **Результаты работы программы**





# **Код программы**

1. Файл HashTable.h
2. #pragma once
3. #include <set>
4. #include <iostream>
5. using std::set;
6. struct list
7. {
8. list\* next;
9. int value;
10. list()
11. {
12. next = nullptr;
13. value = 0;
14. }
15. list(list& L)
16. {
17. value = L.value;
18. if (L.next != nullptr)
19. next = new list(\*L.next);
20. else
21. next = nullptr;
22. }
23. ~list()
24. {
25. if (next != nullptr)
26. {
27. delete next;
28. next = nullptr;
29. }
30. }
31. };
32. class HashTable
33. {
34. private:
35. set<int> keys;
36. list\* values[96];
37. int size;
38. int hashfunc(int key) const
39. {
40. return (a \* key + b) % size;
41. }
42. void clear(int s)
43. {
44. size = s;
45. keys.clear();
46. for (int i = 0; i < 96; ++i)
47. if (values[i] != nullptr)
48. {
49. delete values[i];
50. values[i] = nullptr;
51. }
52. }
53. void clear()
54. {
55. clear(0);
56. }
57. public:
58. static int a, b;
59. HashTable(int size) : size(size) {}
60. HashTable(const HashTable& H)
61. {
62. size = H.size;
63. keys = H.keys;
64. for (int i = 0; i < 96; ++i)
65. if (H.values[i] != nullptr)
66. values[i] = new list(\*H.values[i]);
67. else
68. values[i] = nullptr;
69. }
70. HashTable(HashTable&& H)
71. {
72. size = H.size;
73. keys = H.keys;
74. for (int i = 0; i < 96; ++i)
75. {
76. values[i] = H.values[i];
77. H.values[i] = nullptr;
78. }
79. }
80. ~HashTable()
81. {
82. clear();
83. }
84. HashTable& operator=(const HashTable& H)
85. {
86. if (this != &H)
87. {
88. clear(H.size);
89. size = H.size;
90. keys = H.keys;
91. for (int i = 0; i < 96; ++i)
92. if (H.values[i] != nullptr)
93. values[i] = new list(\*H.values[i]);
94. else
95. values[i] = nullptr;
96. }
97. return \*this;
98. }
99. HashTable& operator=(HashTable&& H)
100. {
101. if (this != &H)
102. {
103. clear(H.size);
104. size = H.size;
105. keys = H.keys;
106. for (int i = 0; i < 96; ++i)
107. {
108. values[i] = H.values[i];
109. H.values[i] = nullptr;
110. }
111. }
112. return \*this;
113. }
114. int operator[](const int& key) const
115. {
116. list\* head = values[hashfunc(key)];
117. if (head == nullptr) return -1;
118. if (head->next == nullptr)
119. return head->value;
120. else
121. {
122. list\* temp = head;
123. while (temp != nullptr)
124. {
125. if (temp->value == key)
126. return temp->value;
127. temp = temp->next;
128. }
129. }
130. return -1;
131. }
132. HashTable operator&(const HashTable& B)
133. {
134. HashTable C(\*this);
135. return C &= B;
136. }
137. HashTable& operator&=(const HashTable& B)
138. {
139. HashTable C(\*this);
140. clear(C.size < B.size ? C.size : B.size);
141. set<int> keysC = C.keys;
142. for (auto i : keysC)
143. {
144. if (B[i] == i)
145. add(i);
146. }
147. return \*this;
148. }
149. HashTable operator|(const HashTable& B)
150. {
151. HashTable C(\*this);
152. return C |= B;
153. }
154. HashTable& operator|=(const HashTable& B)
155. {
156. HashTable C(\*this);
157. clear(C.size + B.size);
158. set<int> keysC = C.keys;
159. for (auto i : keysC)
160. add(i);
161. set<int> keysB = B.keys;
162. for (auto i : keysB)
163. add(i);
164. return \*this;
165. }
166. HashTable operator-(const HashTable& B)
167. {
168. HashTable C(\*this);
169. return C -= B;
170. }
171. HashTable& operator-=(const HashTable& B)
172. {
173. HashTable C(\*this);
174. clear(C.size);
175. set<int> keysC = C.keys;
176. for (auto i : keysC)
177. {
178. if (!(B[i] == i))
179. add(i);
180. }
181. return \*this;
182. }
183. HashTable operator+(const HashTable& B)
184. {
185. HashTable C(\*this);
186. return C += B;
187. }
188. HashTable& operator+=(const HashTable& B)
189. {
190. HashTable C(\*this);
191. clear();
192. \*this = (C | B) - (C & B);
193. return \*this;
194. }
195. void add(int keyAndValue)
196. {
197. add(keyAndValue, keyAndValue);
198. }
199. void add(int key, int value)
200. {
201. keys.insert(key);
202. list\* head = values[hashfunc(key)];
203. if (head == nullptr)
204. {
205. head = new list;
206. head->next = nullptr;
207. head->value = value;
208. values[hashfunc(key)] = head;
209. }
210. else
211. {
212. list\* t = head;
213. bool f = false;
214. while (!f && t != nullptr)
215. {
216. f = value == t->value;
217. t = t->next;
218. }
219. if (!f)
220. {
221. list\* t = head;
222. while (t->next != nullptr)
223. t = t->next;
224. list\* insert = new list;
225. insert->next = nullptr;
226. insert->value = value;
227. t->next = insert;
228. }
229. }
230. }
231. set<int> getKeys()
232. {
233. return keys;
234. }
235. bool valueExists(int key)
236. {
237. list\* head = values[hashfunc(key)];
238. list\* t = head;
239. bool f = false;
240. while (!f && t != nullptr)
241. {
242. f = key == t->value;
243. t = t->next;
244. }
245. return f;
246. }
247. int getSize()
248. {
249. return size;
250. }
251. };

Файл HashTable.h

#pragma once

#include "HashTable.h"

class generator

{

public:

static HashTable GenerateSet(int size, int maxValue)

{

HashTable table(size);

maxValue++;

while (table.getKeys().size() < size)

table.add(rand() % maxValue);

return table;

}

};

1. *Файл main.c*
2. #include "pch.h"
3. #include "HashTable.h"
4. #include "Generator.h"
5. #include <iostream>
6. #include <ctime>
7. #include <windows.h>
8. int HashTable::a = 97;
9. int HashTable::b = 3;
10. using std::cin;
11. using std::cout;
12. using std::endl;
13. void DisplayHashTable(HashTable& table)
14. {
15. set<int> s = table.getKeys();
16. for (auto i : s)
17. if (table[i] != -1)
18. {
19. cout.width(3);
20. cout << i;
21. }
22. cout << endl;
23. }
24. int main()
25. {
26. std::srand(unsigned(std::time(0)));
27. HashTable setA = generator::GenerateSet(32, 99);
28. cout << "A:";
29. DisplayHashTable(setA);
30. HashTable setB = generator::GenerateSet(32, 99);
31. cout << "B:";
32. DisplayHashTable(setB);
33. HashTable setC = generator::GenerateSet(32, 99);
34. cout << "C:";
35. DisplayHashTable(setC);
36. HashTable setD = generator::GenerateSet(32, 99);
37. cout << "D:";
38. DisplayHashTable(setD);
39. HashTable setE = generator::GenerateSet(32, 99);
40. cout << "E:";
41. DisplayHashTable(setE);
42. cout << endl;
43. HashTable set1 = setA & setB;
44. cout << " A & B: ";
45. DisplayHashTable(set1);
46. HashTable set2 = setD - setE;
47. cout << " D \\ E: ";
48. DisplayHashTable(set2);
49. HashTable set3 = set2 | setC;
50. cout << " (C | D \\ E): ";
51. DisplayHashTable(set3);
52. HashTable set4 = set3 + set1;
53. cout << "(A & B) + (C | D \\ E):";
54. DisplayHashTable(set4);
55. system("pause");
56. return 0;
57. }

# **Вывод**

1. В данной работе мы изучили хэш – таблицы, работу с ними, грамотное создание хэш – функций.
2. Сами хэш — таблицы -- это хороший способ хранения данных, с возможностью быстрого их добавления, удаления, считывания, поиск и т. д.
3. Благодаря использованию set и list сильно облегчилась работа с классом.