****

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 6.1

**Тема: «АЛГОРИТМЫ ПОИСКА»**

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент Комисарик М.А.

группа ИКБО-20-23

**Москва 2024**

**Цель работы:** Освоить приёмы хеширования и эффективного поиска элементов множества.

Задание

Формулировка задачи

Разработать приложение, которое использует хеш-таблицу с открытой адресацией для организации прямого доступа к элементам динамического множества полезных данных. Множество реализовать на массиве, содержащем счета в банке:

1. Номер счета – семизначное число (ключ)
2. ФИО
3. Адрес

Приложение должно содержать класс с операциями:

1. Вставки
2. Удаления
3. Поиска по ключу
4. Вывода
5. Расширение размера таблицы и рехеширование

Включить в класс массив полезных данных и хеш-таблицу. Хеш-функция – остаток от деления. Тип разрешения коллизий – двойное хеширование. Предусмотреть автоматическое заполнение таблицы 5-7 записями. Реализовать текстовый командный интерфейс пользователя для возможности вызова методов в любой произвольной последовательности, сопроводить вывод достаточными для понимания происходящего сторонним пользователем подсказками. Провести полное тестирование программы, тест-примеры определить самостоятельно. Результаты тестирования включить в отчет по выполненной работе.

Модель решения

Хеш функция: h(k) = k % m, где m – размер таблицы.

Каждый элемент таблицы содержит в себе поля:

1. Ключ
2. Значения
3. Признак открытого адреса
4. Признак удаленного значения

Вставка в таблицу: Новый элемент вставляется в ячейку таблицы, индекс которой создала хеш-функция. Если ячейка свободна (открытый адрес), то в нее вставляется значение с ключом. В случае появления коллизии, осуществляется поиск свободной ячейки, путем опробования, следующих ячеек (индекс увеличивается на вычисленное смещение), пока не будет найдена свободная ячейка. Эффективность данного подхода падает, когда коэффициент нагрузки приближается к единице. Следующее значение индекса пробирования вычисляется по принципу двойного хеширования, т.е. от вычисленного смещенного значения опять берется хеш функция. Смещение вычисляется по формуле: o(k) = k / m. Если m делится на o(k) нацело, то вместо вычисленного значения берется 1, во избежание бесконечного цикла пробирования.

Поиск в таблице:

1. Получение ключа поиска К.

2. Определение индекса элемента в таблице по ключу К посредством хеш – функции, которая использовалась при вставке значений в таблицу.

3. Проверка, содержит ли элемент, ключ поиска:

3.1. если этот элемент содержит введенный ключ и элемент не удален, то поиск завершается и возвращаются данные записи;

3.2. если элемент таблицы по этому индексу не содержит заданный ключ и индекс не вышел за границу таблицы, то осуществляется алгоритм подбора нового индекса по тому же алгоритму, который использовался при выполнении операции вставки в случае коллизии, пока не будет найден элемент с таким ключом, или не будет найдена открытая не удаленная ячейка. Если найдена открытая и не удаленная ячейка, то это означает, что такого ключа в таблице нет. Если найдена удаленная ячейка поиск продолжается со следующим смещенным значением.

Удаление из таблицы: При выполнении операции удаления сначала осуществляется поиск записи с заданным ключом в таблице, а затем, в случае удачного поиска, и выполняется удаление записи из таблицы, т.е. ячейка должна стать открытой и поле *Признак удаленного значения* должно стать истинным.

Рехеширование: Рехеширование – пересоздание таблицы с использованием всех неудаленных значений текущей таблицы. Происходит когда признак нагрузки таблицы (количество закрытых и удаленных адресов / m) > 0,75. При рехешировании происходит увеличение таблицы в два раза.

Код программы

Код метода Insert():

1. template<typename T>
2. void HashTable<T>::Insert(const unsigned int key, const T& info)
3. {
4. unsigned int i = GetHash(key);
5. while (m\_table[i].IsTaken)
6. {
7. if (m\_table[i].Key == key) return;
8. i = GetHash(i + GetOffset(key));
9. }
10. m\_table[i].Key = key;
11. m\_table[i].Info = info;
12. m\_table[i].IsTaken = true;
13. ++m\_takenAmount;
14. if (m\_table[i].IsDeleted)
15. {
16. m\_table[i].IsDeleted = false;
17. --m\_deletedAmount;
18. }
19. if (static\_cast<float>(m\_takenAmount + m\_deletedAmount) / m\_tableSize <= 0.75f) return;
20. Rehash(GetNextPrime(m\_tableSize \* 2));
21. }

Код метода Delete():

1. template<typename T>
2. void HashTable<T>::Delete(const unsigned int key)
3. {
4. unsigned int hash = GetHash(key);
5. while (m\_table[hash].IsTaken || m\_table[hash].IsDeleted)
6. {
7. if (m\_table[hash].Key == key)
8. {
9. m\_table[hash].IsTaken = false;
10. --m\_takenAmount;
11. m\_table[hash].IsDeleted = true;
12. ++m\_deletedAmount;
13. return;
14. }
15. hash = GetHash(hash + GetOffset(key));
16. }
17. }

Код метода Find():

1. template<typename T>
2. const T\* HashTable<T>::Find(unsigned int key) const
3. {
4. unsigned int hash = GetHash(key);
5. while (m\_table[hash].IsTaken || m\_table[hash].IsDeleted)
6. {
7. if (m\_table[hash].Key == key && !m\_table[hash].IsDeleted)
8. {
9. return &m\_table[hash].Info;
10. }
11. hash = GetHash(hash + GetOffset(key));
12. }
13. return nullptr;
14. }

Код метода Rehash()

1. template <typename T>
2. void HashTable<T>::Rehash(unsigned int newSize)
3. {
4. TableEntry<T>\* newTable = new TableEntry<T>[newSize];
5. TableEntry<T>\* temp = m\_table;
6. unsigned int tempSize = m\_tableSize;
7. m\_table = newTable;
8. m\_tableSize = newSize;
9. m\_takenAmount = 0;
10. m\_deletedAmount = 0;
11. for (unsigned int i = 0; i < tempSize; ++i)
12. {
13. if (temp[i].IsTaken)
14. {
15. Insert(temp[i].Key, temp[i].Info);
16. }
17. }
18. delete[] temp;
19. }

Код для оператора вывода:

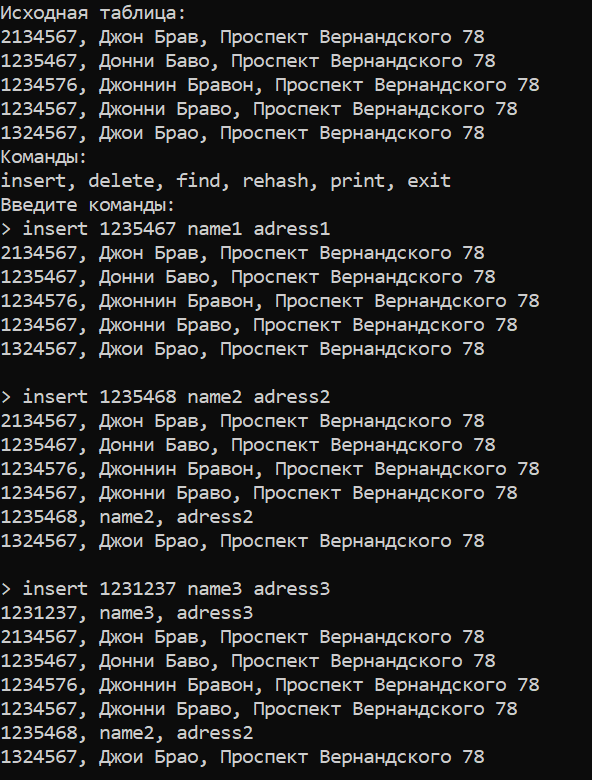
1. template<typename U>
2. std::ostream& operator<<(std::ostream& os, const HashTable<U>& table)
3. {
4. for (unsigned int i = 0; i < table.m\_tableSize; ++i)
5. {
6. if (!table.m\_table[i].IsTaken) continue;
7. os << table.m\_table[i] << '\n';
8. }
9. return os;
10. }

Код программы:

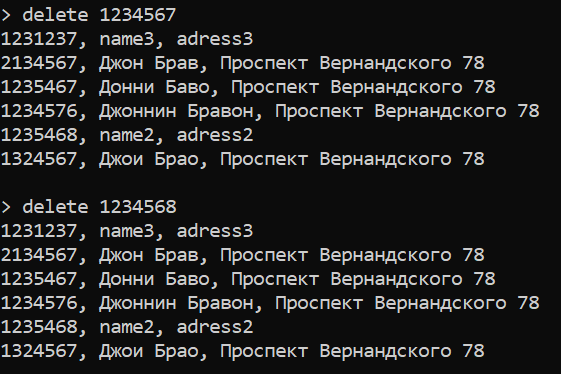
1. HashTable<BankInfo> table(10);
2. table.Insert(1234567, BankInfo("Джонни Браво", "Проспект Вернандского 78"));
3. table.Insert(1234576, BankInfo("Джоннин Бравон", "Проспект Вернандского 78"));
4. table.Insert(2134567, BankInfo("Джон Брав", "Проспект Вернандского 78"));
5. table.Insert(1324567, BankInfo("Джои Брао", "Проспект Вернандского 78"));
6. table.Insert(1235467, BankInfo("Донни Баво", "Проспект Вернандского 78"));
7. std::cout << "Исходная таблица:\n" << table;
8. std::string commandLine;
9. std::vector<std::string> arguments;
10. unsigned int key;
11. std::cout << "Команды:\ninsert, delete, find, rehash, print, exit\n";
12. std::cout << "Введите команды:\n";
13. while (true)
14. {
15. std::cout << "> ";
16. std::getline(std::cin, commandLine);
17. Split(commandLine, " ", arguments);
18. if (arguments.empty() || arguments[0].empty()) continue;
19. if (arguments[0] == "find")
20. {
21. if (arguments.size() < 2)
22. {
23. std::cout << "Предоставьте ключ.\n\n";
24. continue;
25. }
26. key = std::stoi(arguments[1]);
27. const BankInfo\* info = table.Find(key);
28. if (info == nullptr)
29. {
30. std::cout << "Запись не найдена.\n\n";
31. }
32. else
33. {
34. std::cout << "Найденная запись: " << \*info << "\n\n";
35. }
36. }
37. else if (arguments[0] == "insert")
38. {
39. if (arguments.size() < 4)
40. {
41. std::cout << "Предоставьте ключ, имя и адрес.\n\n";
42. continue;
43. }
44. key = std::stoi(arguments[1]);
45. table.Insert(key, BankInfo(arguments[2], arguments[3]));
46. std::cout << table << '\n';
47. }
48. else if (arguments[0] == "delete")
49. {
50. if (arguments.size() < 2)
51. {
52. std::cout << "Предоставьте ключ.\n\n";
53. continue;
54. }
55. key = std::stoi(arguments[1]);
56. table.Delete(key);
57. std::cout << table << '\n';
58. }
59. else if (arguments[0] == "rehash")
60. {
61. if (arguments.size() < 2)
62. {
63. std::cout << "Предоставьте новый размер.\n\n";
64. continue;
65. }
66. unsigned int newSize = std::stoi(arguments[1]);
67. table.Rehash(newSize);
68. std::cout << table << '\n';
69. }
70. else if (arguments[0] == "print")
71. {
72. std::cout << table << '\n';
73. }
74. else if (arguments[0] == "exit")
75. {
76. return;
77. }
78. else
79. {
80. std::cout << "Неверная команда.\n\n";
81. }
82. }

Результаты тестирования

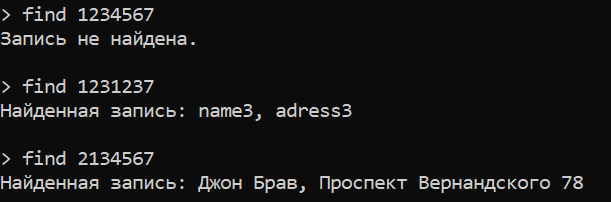
Результаты тестирования метода Insert():

****

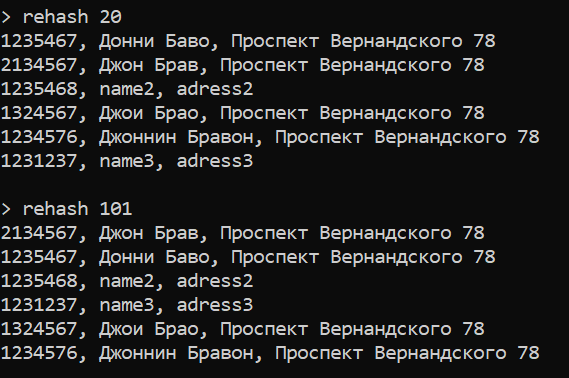
Результаты тестирования метода Delete():



Результаты тестирования метода Find():



Результаты тестирования метода Rehash():



Функция вывода проверена имплицитно при проверке других методов.

ВЫВОДЫ

В рамках задания была реализована структура данных хеш таблицы с открытым адресом и двойным хешированием. Были освоены навыки работы с хеш таблицами.

Приложения

Приложение 1 – Все использованные функции.

1. void GenerateStudentsFile(ofstream& file, unsigned int studentsAmount, unsigned int keySize, unsigned int groupSize, unsigned int nameSize)
2. {
3. ifstream names("names.txt");
4. string nameBuffer;
5. const unsigned int idSize = keySize + groupSize + nameSize;
6. char\* idBuffer = new char[idSize];
7. vector<unsigned int> keys;
8. srand(time(0));
9. GenerateRandomKeys(studentsAmount \* 2, studentsAmount, keys);
10. for (unsigned int key : keys)
11. {
12. for (unsigned int i = 0; i < idSize; i++)
13. {
14. idBuffer[i] = 0;
15. }
16. getline(names, nameBuffer);
17. \*reinterpret\_cast<unsigned int\*>(idBuffer) = key;
18. strncpy(idBuffer + keySize, "ИКБО-20-23", groupSize);
19. strncpy(idBuffer + keySize + groupSize, nameBuffer.c\_str(), nameBuffer.size());
20. file.write(idBuffer, idSize);
21. }
22. delete[] idBuffer;
23. names.close();
24. }
25. char\* LinearFileSearch(const string& fileName, const unsigned int key, const unsigned int entrySize, const unsigned int keyOffset)
26. {
27. ifstream file(fileName, ios::binary | ios::in);
28. if (!file)
29. {
30. file.close();
31. throw invalid\_argument("No file found with name " + fileName);
32. }
33. char\* buffer = new char[entrySize];
34. while (file.read(buffer, entrySize))
35. {
36. unsigned int readKey = \*reinterpret\_cast<unsigned int\*>(buffer + keyOffset);
37. if (readKey == key)
38. {
39. file.close();
40. return buffer;
41. }
42. }
43. file.close();
44. return nullptr;
45. }
46. void PrintStudent(char\* student, const unsigned int keySize, const unsigned int groupSize, const unsigned int nameSize)
47. {
48. cout << "Номер зачетной книжки: " << \*reinterpret\_cast<unsigned int\*>(student) << '\n';
49. cout << "Номер группы: ";
50. cout.write(student + keySize, groupSize) << '\n';
51. cout << "ФИО: ";
52. cout.write(student + keySize + groupSize, nameSize) << '\n';
53. }
54. void GenerateKeyTable(vector<unsigned long long>& table, ifstream& file, const unsigned int entrySize, const unsigned int keyOffset)
55. {
56. table.clear();
57. char\* entry = new char[entrySize];
58. unsigned int i = 0;
59. while (file.read(entry, entrySize))
60. {
61. unsigned int key = \*reinterpret\_cast<unsigned int\*>(entry + keyOffset);
62. table.push\_back(0);
63. \*reinterpret\_cast<unsigned int\*>(&table[i]) = key;
64. \*(reinterpret\_cast<unsigned int\*>(&table[i]) + 1) = i;
65. i++;
66. }
67. SortKeyTable(table);
68. }
69. void SortKeyTable(vector<unsigned long long>& table)
70. {
71. sort(table.begin(), table.end(), [](unsigned long long a, unsigned long long b)
72. {
73. return \*reinterpret\_cast<unsigned int\*>(&a) < \*reinterpret\_cast<unsigned int\*>(&b);
74. });
75. }
76. void GenerateLookupTable(unsigned int len, std::vector<unsigned int>& table)
77. {
78. table = vector<unsigned int>(static\_cast<unsigned int>(log2(len)) + 3, 0);
79. unsigned int power = 1;
80. unsigned int count = 0;
81. do
82. {
83. power <<= 1;
84. table[count] = (len + (power >> 1)) / power;
85. }
86. while (table[count++] != 0);
87. }
88. unsigned int BinarySearchInKeyTable(const vector<unsigned long long>& keyTable, const unsigned int key, const vector<unsigned int>& lookupTable)
89. {
90. unsigned int index = lookupTable[0] - 1;
91. unsigned int count = 0;
92. while (lookupTable[count] != 0)
93. {
94. if (key == \*reinterpret\_cast<const unsigned int\*>(&keyTable[index]))
95. {
96. return \*(reinterpret\_cast<const unsigned int\*>(&keyTable[index]) + 1);
97. }
98. if (key < \*reinterpret\_cast<const unsigned int\*>(&keyTable[index]))
99. {
100. index -= lookupTable[++count];
101. }
102. else
103. {
104. index += lookupTable[++count];
105. }
106. }
107. return keyTable.size();
108. }
109. char\* AccessFileByRef(ifstream& file, const unsigned int ref, const unsigned int entrySize)
110. {
111. file.clear();
112. file.seekg(ref \* entrySize, ios::beg);
113. char\* entry = new char[entrySize];
114. file.read(entry, entrySize);
115. return entry;
116. }
117. void GenerateRandomKeys(const unsigned int maxAmount, const unsigned int keysAmount, vector<unsigned int>& buffer)
118. {
119. buffer.clear();
120. vector<bool> keyTable(maxAmount, false);
121. for (unsigned int i = 0; i < keysAmount; i++)
122. {
123. unsigned int r = rand() % maxAmount;
124. int k = 0;
125. while (keyTable[(r + k) % maxAmount])
126. {
127. k \*= -1;
128. if (!keyTable[(r + k) % maxAmount])
129. {
130. break;
131. }
132. k \*= -1;
133. k++;
134. }
135. buffer.push\_back((r + k) % maxAmount);
136. keyTable[(r + k) % maxAmount] = true;
137. }
138. }
139. void DisplayTimeDuration(std::chrono::steady\_clock::duration duration)
140. {
141. auto time = chrono::duration\_cast<chrono::nanoseconds>(duration);
142. string timeUnit = "наносекунд";
143. long double convertedTime = time.count();
144. if (convertedTime >= 1000)
145. {
146. convertedTime /= 1000;
147. timeUnit = "микросекунд";
148. if (convertedTime >= 1000)
149. {
150. convertedTime /= 1000;
151. timeUnit = "милисекунд";
152. if (convertedTime >= 1000)
153. {
154. convertedTime /= 1000;
155. timeUnit = "секунд";
156. }
157. }
158. }
159. cout << "Затраченное время: " << convertedTime << ' ' << timeUnit << ".\n";
160. }