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:
21:      Rehash(GetNextPrime(m_tableSize * 2));
22:  }
```

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:
19:     delete[] temp;
20: }
```

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:
19:       if (arguments.empty() || arguments[0].empty()) continue;
20:
21:       if (arguments[0] == "find")
22:       {
23:           if (arguments.size() < 2)
24:           {
25:               std::cout << "Предоставьте ключ.\n\n";
26:               continue;
27:           }
28:           key = std::stoi(arguments[1]);
29:           const BankInfo* info = table.Find(key);
30:           if (info == nullptr)
31:           {
32:               std::cout << "Запись не найдена.\n\n";
33:           }
34:           else
35:           {
36:               std::cout << "Найденная запись: " << *info << "\n\n";
37:           }
38:       }
39:       else if (arguments[0] == "insert")
40:       {
41:           if (arguments.size() < 4)
42:           {
43:               std::cout << "Предоставьте ключ, имя и адрес.\n\n";
44:               continue;
45:           }
46:           key = std::stoi(arguments[1]);
47:           table.Insert(key, BankInfo(arguments[2], arguments[3]));
48:           std::cout << table << '\n';
49:       }
50:       else if (arguments[0] == "delete")
51:       {
52:           if (arguments.size() < 2)
53:           {
54:               std::cout << "Предоставьте ключ.\n\n";
55:               continue;
56:           }
57:           key = std::stoi(arguments[1]);
58:           table.Delete(key);
59:           std::cout << table << '\n';
60:       }
61:       else if (arguments[0] == "rehash")
62:       {
63:           if (arguments.size() < 2)
64:           {
65:               std::cout << "Предоставьте новый размер.\n\n";
66:               continue;
67:           }
68:           unsigned int newSize = std::stoi(arguments[1]);
69:           table.Rehash(newSize);
70:           std::cout << table << '\n';
71:       }
72:       else if (arguments[0] == "print")
73:       {
74:           std::cout << table << '\n';
75:       }
76:       else if (arguments[0] == "exit")
77:       {
78:           return;
79:       }
```

```
80:       else
81:       {
82:           std::cout << "Неверная команда.\n\n";
83:       }
84:   }
```

### Результаты тестирования

Результаты тестирования метода Insert():

```
Исходная таблица:
2134567, Джон Брав, Проспект Вернандского 78
1235467, Донни Баво, Проспект Вернандского 78
1234576, Джоннин Бравон, Проспект Вернандского 78
1234567, Джонни Браво, Проспект Вернандского 78
1324567, Джои Брао, Проспект Вернандского 78
Команды:
insert, delete, find, rehash, print, exit
Введите команды:
> insert 1235467 name1 adress1
2134567, Джон Брав, Проспект Вернандского 78
1235467, Донни Баво, Проспект Вернандского 78
1234576, Джоннин Бравон, Проспект Вернандского 78
1234567, Джонни Браво, Проспект Вернандского 78
1324567, Джои Брао, Проспект Вернандского 78

> insert 1235468 name2 adress2
2134567, Джон Брав, Проспект Вернандского 78
1235467, Донни Баво, Проспект Вернандского 78
1234576, Джоннин Бравон, Проспект Вернандского 78
1234567, Джонни Браво, Проспект Вернандского 78
1235468, name2, adress2
1324567, Джои Брао, Проспект Вернандского 78

> insert 1231237 name3 adress3
1231237, name3, adress3
2134567, Джон Брав, Проспект Вернандского 78
1235467, Донни Баво, Проспект Вернандского 78
1234576, Джоннин Бравон, Проспект Вернандского 78
1234567, Джонни Браво, Проспект Вернандского 78
1235468, name2, adress2
1324567, Джои Брао, Проспект Вернандского 78
```

Результаты тестирования метода Delete():

```
> delete 1234567
1231237, name3, adress3
2134567, Джон Брав, Проспект Вернандского 78
1235467, Донни Баво, Проспект Вернандского 78
1234576, Джоннин Бравон, Проспект Вернандского 78
1235468, name2, adress2
1324567, Джои Брао, Проспект Вернандского 78

> delete 1234568
1231237, name3, adress3
2134567, Джон Брав, Проспект Вернандского 78
1235467, Донни Баво, Проспект Вернандского 78
1234576, Джоннин Бравон, Проспект Вернандского 78
1235468, name2, adress2
1324567, Джои Брао, Проспект Вернандского 78
```

Результаты тестирования метода Find():

```
> find 1234567
Запись не найдена.

> find 1231237
Найденная запись: name3, adress3

> find 2134567
Найденная запись: Джон Брав, Проспект Вернандского 78
```

Результаты тестирования метода Rehash():

```
> rehash 20
1235467, Донни Баво, Проспект Вернандского 78
2134567, Джон Брав, Проспект Вернандского 78
1235468, name2, adress2
1324567, Джои Брао, Проспект Вернандского 78
1234576, Джоннин Бравон, Проспект Вернандского 78
1231237, name3, adress3

> rehash 101
2134567, Джон Брав, Проспект Вернандского 78
1235467, Донни Баво, Проспект Вернандского 78
1235468, name2, adress2
1231237, name3, adress3
1324567, Джои Брао, Проспект Вернандского 78
1234576, Джоннин Бравон, Проспект Вернандского 78
```

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:
6:      const unsigned int idSize = keySize + groupSize + nameSize;
7:      char* idBuffer = new char[idSize];
8:
9:      vector<unsigned int> keys;
10:     srand(time(0));
11:     GenerateRandomKeys(studentsAmount * 2, studentsAmount, keys);
12:
13:     for (unsigned int key : keys)
14:     {
15:         for (unsigned int i = 0; i < idSize; i++)
16:         {
17:             idBuffer[i] = 0;
18:         }
19:         getline(names, nameBuffer);
20:         *reinterpret_cast<unsigned int*>(idBuffer) = key;
21:         strncpy(idBuffer + keySize, "ИКБО-20-23", groupSize);
22:         strncpy(idBuffer + keySize + groupSize, nameBuffer.c_str(),
    nameBuffer.size());
23:         file.write(idBuffer, idSize);
24:     }
25:
26:     delete[] idBuffer;
27:     names.close();
28: }
29:
30: char* LinearFileSearch(const string& fileName, const unsigned int key, const
    unsigned int entrySize, const unsigned int keyOffset)
31: {
32:     ifstream file(fileName, ios::binary | ios::in);
33:     if (!file)
34:     {
35:         file.close();
36:         throw invalid_argument("No file found with name " + fileName);
37:     }
38:
39:     char* buffer = new char[entrySize];
40:     while (file.read(buffer, entrySize))
41:     {
42:         unsigned int readKey = *reinterpret_cast<unsigned int*>(buffer +
    keyOffset);
43:         if (readKey == key)
44:         {
45:             file.close();
46:             return buffer;
47:         }
48:     }
49:     file.close();
50:     return nullptr;
51: }
52:
53: void PrintStudent(char* student, const unsigned int keySize, const unsigned
    int groupSize, const unsigned int nameSize)
54: {
55:     cout << "Номер зачетной книжки: " << *reinterpret_cast<unsigned
    int*>(student) << '\n';
```

```
56:       cout << "Номер группы: ";
57:       cout.write(student + keySize, groupSize) << '\n';
58:       cout << "ФИО: ";
59:       cout.write(student + keySize + groupSize, nameSize) << '\n';
60:   }
61:
62:   void GenerateKeyTable(vector<unsigned long long>& table, ifstream& file, const
      unsigned int entrySize, const unsigned int keyOffset)
63:   {
64:       table.clear();
65:       char* entry = new char[entrySize];
66:       unsigned int i = 0;
67:       while (file.read(entry, entrySize))
68:       {
69:           unsigned int key = *reinterpret_cast<unsigned int*>(entry +
      keyOffset);
70:           table.push_back(0);
71:           *reinterpret_cast<unsigned int*>(&table[i]) = key;
72:           *(reinterpret_cast<unsigned int*>(&table[i]) + 1) = i;
73:           i++;
74:       }
75:       SortKeyTable(table);
76:   }
77:
78:   void SortKeyTable(vector<unsigned long long>& table)
79:   {
80:       sort(table.begin(), table.end(), [](unsigned long long a, unsigned long
      long b)
81:       {
82:           return *reinterpret_cast<unsigned int*>(&a) <
      *reinterpret_cast<unsigned int*>(&b);
83:       });
84:   }
85:
86:   void GenerateLookupTable(unsigned int len, std::vector<unsigned int>& table)
87:   {
88:       table = vector<unsigned int>(static_cast<unsigned int>(log2(len)) + 3, 0);
89:       unsigned int power = 1;
90:       unsigned int count = 0;
91:
92:       do
93:       {
94:           power <<= 1;
95:           table[count] = (len + (power >> 1)) / power;
96:       }
97:       while (table[count++] != 0);
98:   }
99:
100:  unsigned int BinarySearchInKeyTable(const vector<unsigned long long>&
      keyTable, const unsigned int key, const vector<unsigned int>& lookupTable)
101:  {
102:      unsigned int index = lookupTable[0] - 1;
103:      unsigned int count = 0;
104:      while (lookupTable[count] != 0)
105:      {
106:          if (key == *reinterpret_cast<const unsigned int*>(&keyTable[index]))
107:          {
108:              return *(reinterpret_cast<const unsigned int*>(&keyTable[index]) +
      1);
109:          }
110:
111:          if (key < *reinterpret_cast<const unsigned int*>(&keyTable[index]))
112:          {
113:              index -= lookupTable[++count];
114:          }
```

```
115:         else
116:         {
117:             index += lookupTable[++count];
118:         }
119:     }
120:     return keyTable.size();
121: }
122:
123: char* AccessFileByRef(ifstream& file, const unsigned int ref, const unsigned
     int entrySize)
124: {
125:     file.clear();
126:     file.seekg(ref * entrySize, ios::beg);
127:     char* entry = new char[entrySize];
128:     file.read(entry, entrySize);
129:     return entry;
130: }
131:
132: void GenerateRandomKeys(const unsigned int maxAmount, const unsigned int
     keysAmount, vector<unsigned int>& buffer)
133: {
134:     buffer.clear();
135:     vector<bool> keyTable(maxAmount, false);
136:     for (unsigned int i = 0; i < keysAmount; i++)
137:     {
138:         unsigned int r = rand() % maxAmount;
139:         int k = 0;
140:         while (keyTable[(r + k) % maxAmount])
141:         {
142:             k *= -1;
143:             if (!keyTable[(r + k) % maxAmount])
144:             {
145:                 break;
146:             }
147:             k *= -1;
148:             k++;
149:         }
150:         buffer.push_back((r + k) % maxAmount);
151:         keyTable[(r + k) % maxAmount] = true;
152:     }
153: }
154:
155: void DisplayTimeDuration(std::chrono::steady_clock::duration duration)
156: {
157:     auto time = chrono::duration_cast<chrono::nanoseconds>(duration);
158:     string timeUnit = "наносекунд";
159:     long double convertedTime = time.count();
160:     if (convertedTime >= 1000)
161:     {
162:         convertedTime /= 1000;
163:         timeUnit = "микросекунд";
164:         if (convertedTime >= 1000)
165:         {
166:             convertedTime /= 1000;
167:             timeUnit = "милисекунд";
168:             if (convertedTime >= 1000)
169:             {
170:                 convertedTime /= 1000;
171:                 timeUnit = "секунд";
172:             }
173:         }
174:     }
175:     cout << "Затраченное время: " << convertedTime << ' ' << timeUnit <<
     ".\n";
176: }
```