МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА» - Российский технологический университет»

РТУ МИРЭА

Отчёт по выполнению практического задания № 6.1
Тема:
«Быстрый доступ к данным с помощью хеш-таблиц»

Выполнил студент: Данов Арсений Иванович

Группа: ИКБО-10-23

Москва 2024

Цель:

Освоить приёмы хеширования и эффективного поиска элементов множества.

Задание:

Разработать приложение, которое использует хеш-таблицу (пары «ключ – хеш») для организации прямого доступа к элементам динамического множества полезных данных.

Вариант:

Номер: *6*

Метод хеширования:

Открытая адресация (линейное пробирование)

Структура элемента множества. Ключи записей подчеркнуты:

Специализация вуза: код специальности, название вуза

1.1 Формулировка задачи

Разработайте приложение, которое использует хеш-таблицу (пары «ключ – хеш») для организации прямого доступа к элементам динамического множества полезных данных. Множество реализуйте на массиве, структура элементов (перечень полей) которого приведена в индивидуальном варианте.

Приложение должно содержать класс с базовыми операциями: вставки, удаления, поиска по ключу, вывода. Включите в класс массив полезных данных и хеш-таблицу. Хеш-функцию подберите самостоятельно, используя правила выбора функции.

Реализуйте расширение размера таблицы и рехеширование, когда это требуется, в соответствии с типом разрешения коллизий.

Предусмотрите автоматическое заполнение таблицы 5-7 записями.

Реализуйте текстовый командный интерфейс пользователя для возможности вызова методов в любой произвольной последовательности, сопроводите вывод достаточными для понимания происходящего сторонним пользователем подсказками.

Проведите полное тестирование программы (все базовые операции, изменение размера и рехеширование), тест-примеры определите самостоятельно. Результаты тестирования включите в отчет по выполненной работе.

1.2 Ход решения

По рекомендации, оставленной в методических указаниях к практической работе, было решено воспользоваться хеш-функцией, основанной на делении, а конкретнее - на модальной арифметике. Таким образом индекс каждой записи вычисляется как остаток от деления ключа на текущую длину массива, использующегося в хеш-таблице.

Избежание коллизий достигается путём линейного (открытого) пробирования, таким образом при совпадении вычисленных индексов с уже

имеющимися к ним прибавляется константа до тех пор, пока не будет найдено не занятое другой записью место в массиве.

Были реализованы базовые операции взаимодействия с хеш-таблицей.

В ходе выполнения операции вставки по ключу вычисляется значение хеш-функции, после чего происходит попытка вставить по соответствующему индексу нужную структуру. Если индекс занят — производится линейное пробирование до тех пор, пока не будет найдено свободное место, либо пока не произойдёт превышение длины массива, в случае чего понадобится рехеширование. Также после вставки проверяется условие достаточности необходимого места, после чего при необходимости также проводится рехеширование.

В ходе удаления элемента происходит поиск необходимой записи по индексу хеш-функции, после чего происходит удаление. Аналогично работает функция поиска.

В ходе выполнения рехеширования создаётся новый массив в два раза больше предыдущего, после чего предпринимается попытка рехеширования. В случае неудачи длина увеличивается ещё в два раза, иначе же рехеширование завершается и новый массив заменяет предшествующий.

1.3 Исходный код программ

На рисунках 1-6 представлен исходный коды программы.

```
#ifndef __HASH__H
#define __HASH__H

#include <iostream>
#include <fstream>
#include <vector>
#include <string>

#include "unispec.h"

class dict
{
private:
    std::vector<unispec *> values;
    int count;

public:
    dict();
    void insert(unispec *el);
    void del(std::string code);
    unispec *find(std::string code);
    void show();
    int format(std::string code, int dl);
    void rehash();
};

#endif
```

Рисунок 1 — Заголовок класса hash

```cpp
#include "hash.h"

using namespace std;

dict::dict()
{
    this->count = 0;
    this->values.resize(10, nullptr);

    ifstream file;
    file.open("start.txt");

    string code, name;

    unispec *x = new unispec();
    for (int i = 0; i < 7; ++i)
    {
        file >> code >> name;
        x->code = code;
        x->name = name;
        this->insert(x);
        x = new unispec();
    }

    file.close();
    delete x;
}

int dict::format(string code, int dl)
{
    int cd = 0;
    for (int i = 0; i < code.length(); ++i)
    {
        cd += code[i];
    }
    return cd % dl;
}
```

Рисунок 2 — Конструктор и хеш-функция

```
39  void dict::insert(unispec *el)
40  {
41      int code = this->format(el->code, this->values.size());
42
43      int i = 0;
44      int ind = code;
45      while (true)
46      {
47          if (ind + 7 * i > this->values.size() - 1)
48          {
49              this->rehash();
50              continue;
51          }
52          if (this->values[ind + 7 * i])
53          {
54              if (this->values[ind + 7 * i]->code == el->code)
55              {
56                  this->values[ind + 7 * i]->name = el->name;
57                  this->count += 1;
58                  delete el;
59                  break;
60              }
61              ++i;
62              continue;
63          }
64
65          this->values[ind + 7 * i] = el;
66          this->count += 1;
67          break;
68      }
69      if (((double)this->count) / this->values.size() >= 0.75)
70      {
71          this->rehash();
72      }
73  }
```

Рисунок 3 — Метод вставки

```
75   void dict::del(string code)
76   {
77       int ind = this->format(code, this->values.size());
78       for (int i = ind; i < this->values.size(); i += 7)
79       {
80           if (this->values[i]->code == code)
81           {
82               unispec *x = this->values[i];
83               this->values[i] = nullptr;
84               delete x;
85               return;
86           }
87       }
88   }
89
90   unispec *dict::find(string code)
91   {
92       int ind = this->format(code, this->values.size());
93       for (int i = ind; i < this->values.size(); i += 7)
94       {
95           if (this->values[i]->code == code)
96           {
97               return this->values[i];
98           }
99       }
100      return nullptr;
101  }
```

Рисунок 4 — Методы удаления и поиска

```
103  void dict::rehash()
104  {
105      bool rehashed = false;
106      vector<unispec *> v;
107      for (int i = 2; !rehashed; i *= 2)
108      {
109          v.resize(0);
110          v.resize(this->values.size() * i, nullptr);
111          rehashed = true;
112          for (auto x : this->values)
113          {
114              if (x)
115              {
116                  int code = this->format(x->code, v.size());
117                  if (code > v.size() - 1)
118                  {
119                      rehashed = false;
120                      break;
121                  }
122                  v[code] = x;
123              }
124          }
125      }
126      this->values = v;
127  }
128
129  void dict::show()
130  {
131      for (auto x : this->values)
132      {
133          if (x)
134          {
135              cout << x->code << " " << x->name << '\n';
136          }
137      }
138  }
139
```

Рисунок 5 — Методы рехеширования и вывода

```
int main()
{
    dict d = dict();
    cout << "Элементы хеш-таблицы:\n";
    d.show();
    string code, name;

    unispec *x;

    while (true)
    {
        int choise;
        cout << endl
             << "Что вы хотите сделать?" << endl
             << "1 - Вывод содержимого" << endl
             << "2 - Добавление записи" << endl
             << "3 - Удаление записи" << endl
             << "4 - Поиск записи" << endl
             << "0 - Выход" << endl
             << "Номер действия: ";
        cin >> choise;
        cout << endl;
        if (choise == 1)
        {
            cout << "Элементы хеш-таблицы:\n";
            d.show();
        }
        else if (choise == 2)
        {
            cout << "Введите данные новой записи: ";
            x = new unispec();
            cin >> code >> name;
            x->code = code;
            x->name = name;
            d.insert(x);
        }
        else if (choise == 3)
        {
            cout << "Введите код записи для удаления: ";
            cin >> code;
            d.del(code);
        }
        else if (choise == 4)
        {
            cout << "Введите код записи для поиска: ";
            cin >> code;
            x = d.find(code);
            if (x)
            {
                cout << "Найденная запись:\n"
                     << x->code << " " << x->name;
            }
        }
```

Рисунок 6 — Исходный код функции main

1.4 Результаты тестирования

На рисунках 7-9 представлены результаты тестирования.

```
Элементы хеш-таблицы:
3.52.13 РАНХиГС
4.42.12 МФТИ
41.12.6 МИСИС
9.13.12 РУДН
05.32.12 МАИ
3.63.13 СТАНКИН
09.03.04 МИРЭА

Что вы хотите сделать?
1 - Вывод содержимого
2 - Добавление записи
3 - Удаление записи
4 - Поиск записи
0 - Выход
Номер действия: 2

Введите данные новой записи: 1.3.1313 MGU

Что вы хотите сделать?
1 - Вывод содержимого
2 - Добавление записи
3 - Удаление записи
4 - Поиск записи
0 - Выход
Номер действия: 1

Элементы хеш-таблицы:
3.52.13 РАНХиГС
4.42.12 МФТИ
41.12.6 МИСИС
9.13.12 РУДН
1.3.1313 MGU
05.32.12 МАИ
3.63.13 СТАНКИН
09.03.04 МИРЭА
```

Рисунок 7 — Результат тестирования вставки

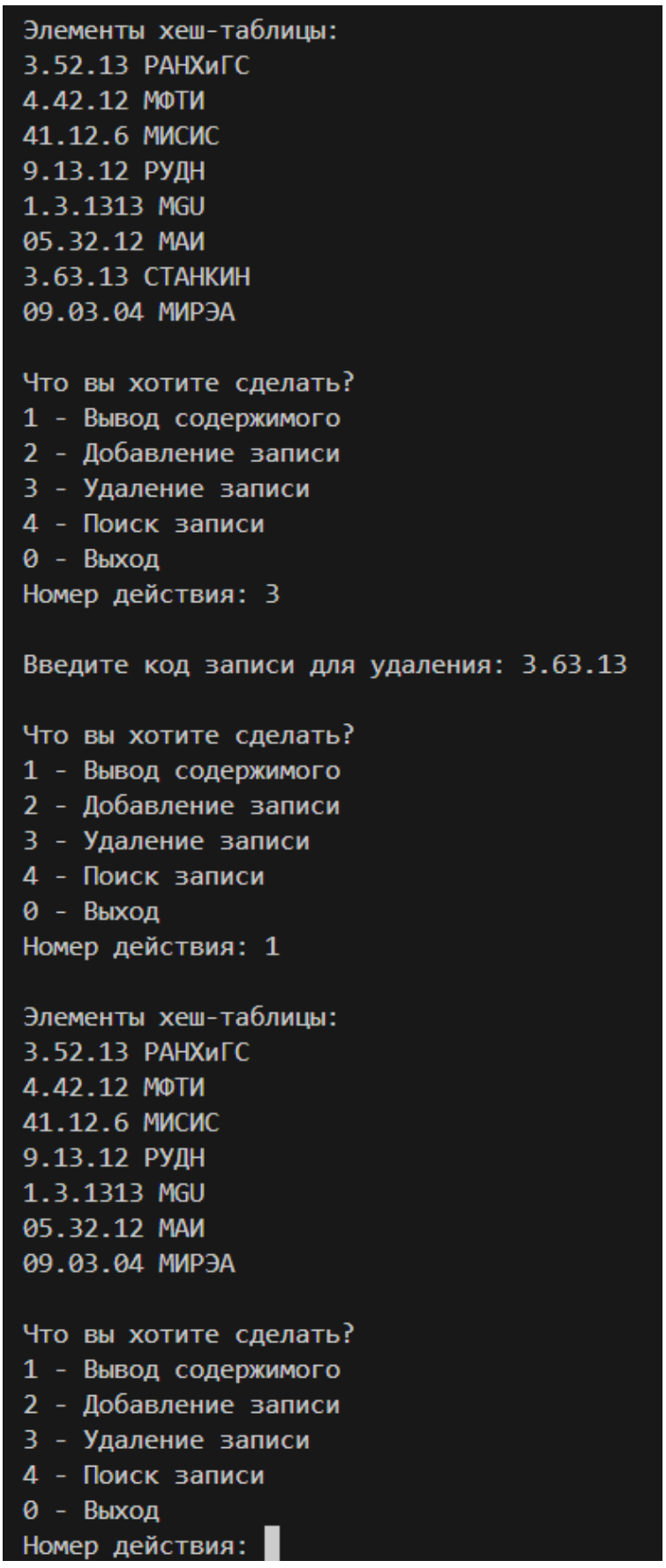

Рисунок 8 — Результат тестирования удаления

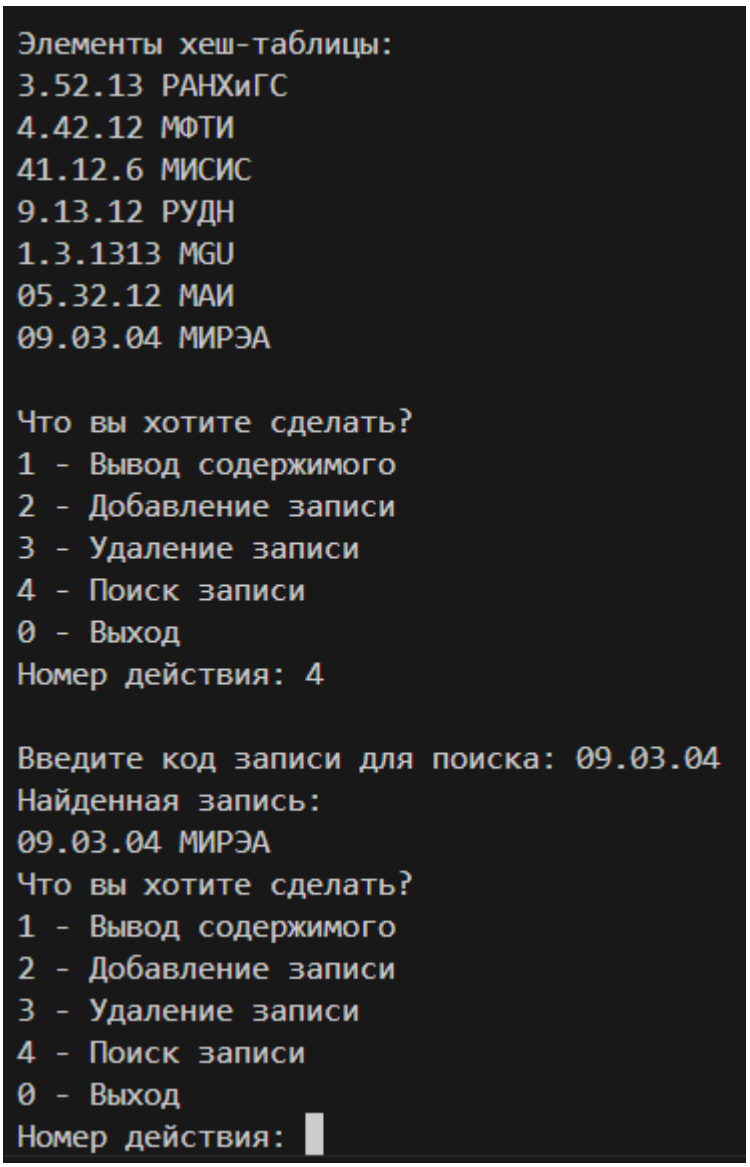

Рисунок 9 — Результат тестирования поиска

2.1 Выводы

В ходе решения поставленных задач были освоены приёмы хеширования и эффективного поиска элементов множества.

Применение хеш-таблиц и хеш-функций позволяет использовать в качестве ключей для значений во множестве любую информацию, при этом не теряя в скорости её считывания.