МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания №6

Тема:

АЛГОРИТМЫ ПОИСКА

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Враженко Д.О.

Группа: ИКБО-50-23

Вариант: 6

Москва – 2024

ЦЕЛЬ РАБОТЫ

Часть 6.1. Освоить приёмы хеширования и эффективного поиска элементов множества.

Часть 6.2. Освоить приёмы реализации алгоритмов поиска образца в тексте.

ХОД РАБОТЫ

Часть 6.1. Быстрый доступ к данным с помощью хеш-таблиц

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.

Индивидуальный вариант:

Открытая адресация (линейное пробирование). Специализация вуза: код специальности – (прим.: "09.03.01"), название вуза.

Математическая модель решения (описание алгоритма):

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.

Код программы с комментариями:

На рис. 1 представлен код структуры данных, содержащихся в хеш-таблице, а именно код специальности и название ВУЗа.

```
struct Data
{
    string code; // Код специальности
    string name; // Название ВУЗа
};
```

Рисунок 1 - Структура данных хеш-таблицы

На рис. 2 представлен код полей класса HashTable, в котором описываются методы, используемые для работы с хеш-таблицей.

```
class HashTable
{

private:

    vector<Data*> table; // Хеш-таблица
    int count; // Кол-во элементов в таблице
```

Рисунок 2 - Поля класса HashTable

На рис. 3 представлен код конструктора хеш-таблицы, в котором происходит изначальное создание таблицы и внесение в неё стартовых значений.

```
public:

    HashTable() // Конструктор хеш-таблицы
    {
        count = 0; table.resize(10, nullptr);
        ifstream file;
        file.open("text.txt");
        Data* element = new Data();
        for (int i = 0; i < 7; i++) // Начальное заполненние таблицы
        {
            file >> element->code >> element->name;
            Input(element);
            element = new Data();;
        }
        delete element;
        file.close();
    }
```

Рисунок 3 - Конструктор хеш-таблицы

На рис. 4 представлен код функции для ввода новых данных в хеш-таблицу.

```
void Input(Data* element) // Функция для ввода данных в таблицу
{
    int code = Format(element->code, table.size()); // Определение хеша элемента
    int i = 0, index = code;
    // Цикл будет выполняться до тех пор, пока для элемента не найдется свободное место
    while (true)
    {
        // Если таблица заполнена, то выполняем рехеширование
        if ((index + 7 * i) > (table.size() - 1))
        {
            Rehash(); continue;
        }
        // Если на данном месте уже есть элемент, то...
        if (table[index + 7 * i])
        {
            // Если коды элементов одинаковые, то изменяем название ВУЗа
            if (table[index + 7 * i]->code == element->code)
            {
                table[index + 7 * i]->name = element->name;
                ++count; delete element; break;
            }
            ++i; continue;
        }
        // Присваивание элементу таблицы значения кода специальности и названия ВУЗа
        table[index + 7 * i] = element; ++count; break;
    }
    // Выполняем рехеширование в случае необходимости
    if ((double)count / table.size() >= 0.75) Rehash();
}
```

Рисунок 4 - Функция для ввода новых данных в таблицу

На рис. 5 представлен код для удаления данных из хеш-таблицы по коду специальности.

```
void Delete(string code) // Функция для удаления элемента из таблицы
{
    int index = Format(code, table.size()); // Определение хеша элемента
    // Поиск элемента в таблице по коду специальности
    for (int i = index; i < table.size(); i += 7)
        if (table[i]->code == code)
        {
            Data* element = table[i];
            table[i] = nullptr;
            delete element; return;
        }
}
```

Рисунок 5 - Функция для удаления данных из хеш-таблицы

На рис. 6 представлен код для поиска элемента в хеш-таблице по коду специальности.

```
Data* Search(string code) // Функция для поиск элемента в таблице
{
    int index = Format(code, table.size()); // Определение хеша элемента
    // Поиск элемента в таблице по коду специальности
    for (int i = index; i < table.size(); i += 7)
        if (table[i]->code == code)
            return table[i];
    return nullptr;
}
```

Рисунок 6 - Функция для поиска элемента в хеш-таблице

На рис. 7 представлен код для вывода всех данных из хеш-таблицы на экран.

```
void Print() // Функция для вывода таблицы на экран
{
    // В случае, если элемент таблицы не nullptr, выводим его на экран
    for (Data* element : table) if (element)
        cout << element->code << ' ' << element->name << '\n';
}
```

Рисунок 7 - Функция для вывода информации из хеш-таблицы

На рис. 8 представлен код для определения индекса элемента таблицы.

```
int Format(string code, int dl) // Функция для определения индекса элемента таблицы
{
    int cd = 0;
    for (int i = 0; i < code.length(); i++)
        cd += code[i];
    return cd % dl;
}
```

Рисунок 8 - Функция для определения индекса элемента хеш-таблицы

На рис. 9 представлен код для рехеширования хеш-таблицы в случае необходимости.

```
void Rehash() // Функция для рехеширования таблицы
{
    bool flag = false;
    vector<Data*> Vector; // Объявление новой хеш-таблицы
    for (int i = 2; !flag; i *= 2)
    {
        Vector.resize(0); Vector.resize(table.size() * i, nullptr); flag = true;
        // Занесение данных из старой хеш-таблицы в новую
        for (Data* element : table)
        {
            if (element)
            {
                int code = Format(element->code, Vector.size());
                if (code > Vector.size() - 1)
                {
                    flag = false; break;
                }
                Vector[code] = element;
            }
        }
    }
    // Изменение старой хеш-таблицы на новую
    table = Vector;
}
```

Рисунок 9 - Функция для рехеширования таблицы

Результаты тестирования:

На рис. 10 представлен результат тестирования функции вывода содержимого.

```
Выберите одну из функций:
1 - Вывод содержимого;
2 - Добавление записи;
3 - Удаление записи;
4 - Поиск записи;
0 - Выход.
Номер действия: 1

Элементы хеш-таблицы:
27.03.01 MGTUGA
01.03.02 MFTI
13.03.02 MISIS
12.03.04 MAI
45.03.02 MGLU
09.03.03 RANHiGS
09.03.04 MIREA
```

Рисунок 10 - Результат функции вывода

На рис. 11 представлен результат тестирования функции добавления нового элемента.

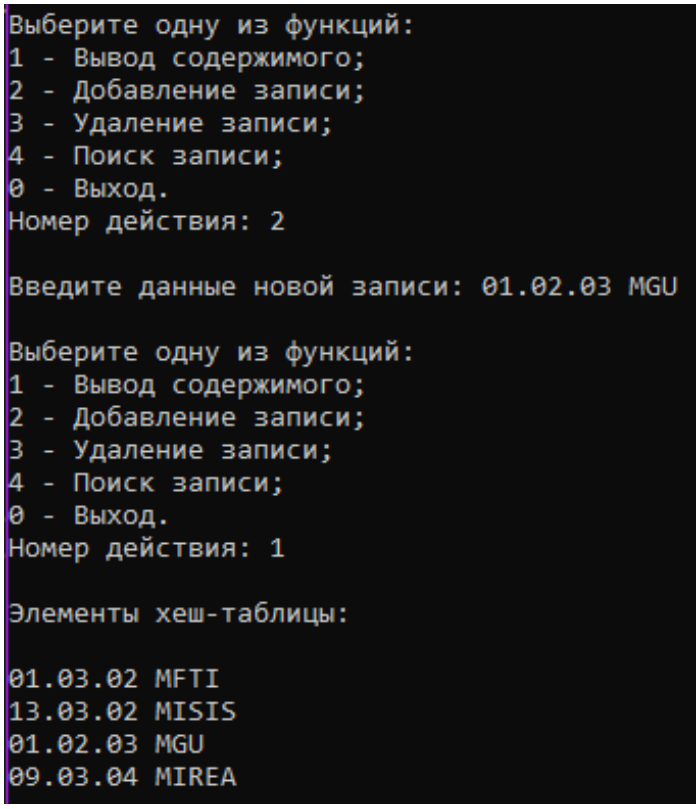

Рисунок 11 - Результат функции добавления

На рис. 12 представлен результат тестирования функции удаления элемента.

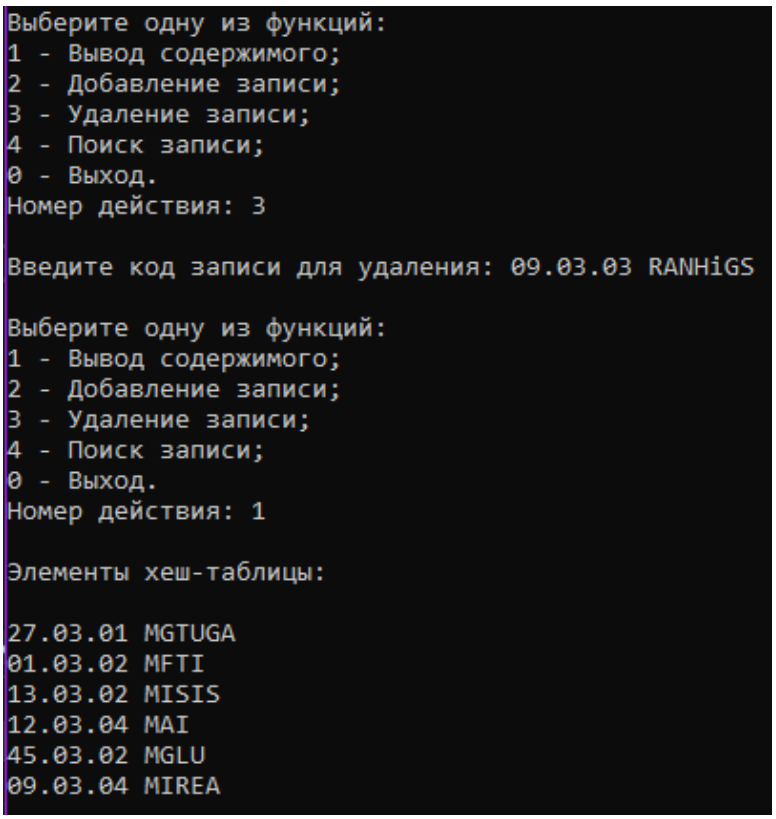

Рисунок 12 - Результат тестирования функции удаления

На рис. 13 представлен результат тестирования функции поиска элемента.

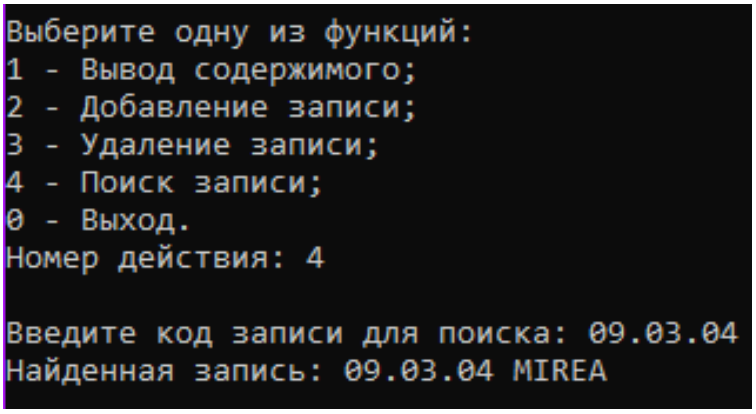

Рисунок 13 - Результат тестирования функции поиска

Часть 6.2. Алгоритмы поиска в таблице при работе с данными из файла

Формулировка задачи:

Разработайте приложения в соответствии с заданиями в индивидуальном варианте.

Индивидуальный вариант:

1. Дан произвольный текст, состоящий из слов, разделенных знаками препинания. Отредактировать его, оставив между словами по одному пробелу, а между предложениями по два.

2. Дана непустая строка S, длина которой N не превышает 10^6. Считать, что элементы строки нумеруются от 1 до N. Требуется для всех i от 1 до N вычислить $\pi[i]$ – префикс функцию.

Задание 1:

Математическая модель решения (описание алгоритма):

Для выполнения первой задачи варианта необходимо отредактировать текст, исключив из него все знаки препинания и заменив их нужным количеством пробелов.

Всего в письменной речи используется не очень большой набор знаков препинания. Для разграничения слов используют запятые, точки с запятой, двоеточия, тире, слэшы, кавычки, скобки, а также пробелы. Для разграничения предложений используются точки, вопросительный и восклицательный знаки.

Поэтому для выполнения задания достаточно линейно проходиться по строке, из которой все символы не знаки препинания переписывать в новую строку, знаки препинания разделения слов заменять одним пробелом, а разделения предложений – двумя. Также при вставке пробелов необходимо учитывать их количество, чтобы не вставить два пробела между словами из-за двух или трёх знаков препинания подряд.

Код программы с комментариями:

На рис. 14 представлен код для замены знаков препинания на нуное количество пробелов.

```
string Replace(string str) // Функция для замены знаков на пробелы
{
    string new_str = ""; // Новая строка
    string end = "!?."; // Знаки между предложениями
    string between = ",;:-\"()/\\«»–[]"; // Знаки между словами
    bool flag = false; // Флаг для запоминания "состояния" пробелов
    for (int i = 0; i < str.length(); ++i) // Проход по встроке
    {
        // Проверка на русскую букву
        if (str[i] >= 'а' && str[i] <= 'я' || str[i] >= 'А' && str[i] <= 'Я' || str[i] == 'ё' || str[i] == 'Ё')
        {
            if (flag) flag = false;
            new_str += str[i];
        }
        // Проверка на пробел
        else if (str[i] == ' ' && !flag)
        {
            new_str += ' ';
            flag = true;
        }
        else
            // Проверка на символ между словами
            if (between.find(str[i]) != string::npos)
            {
                if (!flag)
                {
                    new_str += ' ';
                    flag = true;
                }
            }
            // Проверка на символ между предложениями
            else if (end.find(str[i]) != string::npos)
            {
                if (!flag)
                {
                    new_str += "  ";
                    flag = true;
                }
            }
    }
    return new_str; // Вовзрат новой строки
}
```

Рисунок 14 - Функция для замены символов и вставки пробелов

Результаты тестирования:

На рис. 15 представлен результат тестирования программы.

```
Исходный текст:
На рассвете мы отправились в лес – аромат свежести и хвои наполнял воздух. Вдруг, среди деревьев, мы заметили: дикие цве
ты, искрящиеся росой, привлекли наше внимание."Как красиво!" – воскликнула Мария, указывая на ярко-красные маки. Мы реши
ли остановиться, чтобы сделать несколько фотографий. В тишине слышался лишь шёпот листьев и пение птиц, словно лес пригл
ашал нас в своё волшебное царство.
Измененный текст:
На рассвете мы отправились в лес аромат свежести и хвои наполнял воздух  Вдруг среди деревьев мы заметили дикие цветы ис
крящиеся росой привлекли наше внимание  Как красиво  воскликнула Мария указывая на ярко красные маки  Мы решили останови
ться чтобы сделать несколько фотографий  В тишине слышался лишь шёпот листьев и пение птиц словно лес приглашал нас в св
оё волшебное царство
```

Рисунок 15 - Результат тестирования программы

Задание 2:

Математическая модель решения (описание алгоритма):

Для нахождения префикс функции строки сначала необходимо создать массив из нулевых значений, которые мы будем динамически заполнять по ходу выполнения. Изначально префикс функция является препроцессингом для алго-

ритмов поиска вхождения подстроки в строку. В ней каждое значение является длиной наибольшего собственного префикса части образца, одновременно являющееся его суффиксом.

Расчёт функции происходит динамически для каждого элемента образца. Значение функции в точке 0 полагается равным 0, а далее для каждого символа происходит сравнение: если он равен символу, чей индекс определяется значением префикс функции предыдущего элемента, значит он также является продолжением префикса строки и его значение префиксной функции будет на единицу больше, чем у предыдущего элемента. Иначе же проверяется значение на единицу меньше прошлого значения префиксной функции до тех пор, пока не будет найдено совпадение или значение не обратится в ноль, после чего результат сохраняется.

Код программы с комментариями:

На рис. 16 представлен код программы.

```
vector<int> PrefixFunction(string s) // Функция для поиска префикс функции
{
    int n = s.length();
    vector<int> pi(n); // Создание вектора для значений префикс функции
    for (int i = 1; i < n; ++i) // Заполнение вектора
    {
        int cur = pi[i - 1];
        while (cur > 0 && s[i] != s[cur])
            cur = pi[cur - 1];
        if (s[i] == s[cur])
            ++cur;
        pi[i] = cur;
    }
    return pi; // Вовзрат значения вектора
}
```

Рисунок 16 - Исходынй код программы

Результаты тестирования:

На рис. 17 представлен вывод программы.

```
Исходная строка:
good morning, i want to say google

Префикс-функция для строки:
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 1, 0, 0 ]
```

Рисунок 17 - Вывод программы

ВЫВОД

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.

В ходе решения поставленных задач были освоены приёмы реализации алгоритмов поиска образца в тексте.

В случае применения линейного поиска сложность алгоритма довольно высока, но благодаря применению более эффективных алгоритмов можно снизить сложность до окололинейной. В частности использование префикс функции позволяет значительно снизить количество выполняемых операций.

СПИСОК ЛИТЕРАТУРЫ

1. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих, 2017. – С. 100-126.
2. Кормен Т.Х. и др. Алгоритмы. Построение и анализ, 2013. – С. 285-318.
3. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.
4. Документация по языку C++ [Электронный ресурс]. URL: https://docs.microsoft.com/ruru/cpp/cpp/ (дата обращения 01.09.2021).
5. Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. URL: https://online-edu.mirea.ru/course/view.php?id=4020 (дата обращения 01.09.2021).