****

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

**Тема: «АЛГОРИТМЫ ПОИСКА»**

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент Кузнецов Л. А.

группа ИКБО-20-23

**Москва 2024СОДЕРЖАНИЕ**

[ЧАСТЬ 6.1………………………………………………………………………...3](#а61)

[Условие…………………………………………………………………………...3](#а61условие)

[Метод решения…………………………………………………………………...3](#а61методрешения)

[ЧАСТЬ 6.2……………………………………………………………...………..11](#а62)

[Условие………………………………………………………………………….11](#а62условие)

[ЗАДАНИЕ 1……………………………………………………………………..12](#а621)

[Формулировка задачи…………………………………………………………..12](#а621формулоировка)

[Математическая модель решения……………………………………………...12](#а621модель)

[Код программы с комментариями……………………………………………..12](#а621код)

[Результаты тестирования………………………………………………………13](#а621результат)

[ЗАДАНИЕ 2……………………………………………………………………..15](#а622)

[Формулировка задачи………………………………………………………..…15](#а622формулировка)

[Математическая модель решения……………………………………………...15](#а622модель)

[Код программы с комментариями……………………………………………..15](#а622код)

[Результаты тестирования……………………………………………………….15](#а622результат)

[ВЫВОД…………………………………………………………………………..17](#вывод)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ………………………….17](#список)

**ЧАСТЬ 6.1**

**Условие**

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 (п.3).

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.

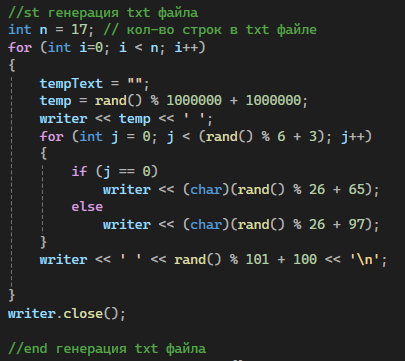
Реализуйте текстовый командный интерфейс пользователя для возможности вызова методов в любой произвольной последовательности, сопроводите вывод достаточными для понимания происходящего сторонним пользователем подсказками.

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

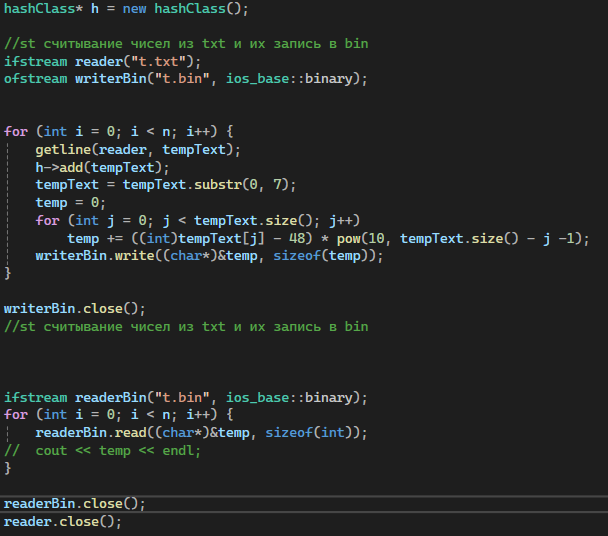
**Метод решения**

Для начала мы создадим генератор случайных значений для нашей хеш-таблицы (рис. 1). Мы создаём сначала шестизначное число (ключ к данным в обычной таблице), а затем переходим с созданию названия товара и его цены.

После генерации данных мы переводим их в бинарный файл, откуда после считываем значения в нашу хеш-таблицу (рис. 2).

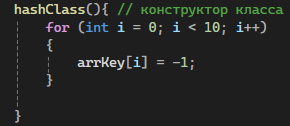


**Рисунок 1 – Генерация данных хеш-таблицы**

****

**Рисунок 2 – Считывание данных в хеш-таблицу**

Теперь же следует более подробно описать создание самой хеш-таблицы: как-никак – это и есть вся данная работа. В самом начале следует понять, какой же конструктор нужен для данной работы хеш-таблицы. В итоге на рис. 3 представлен конечный результат размышлений по данному поводу.

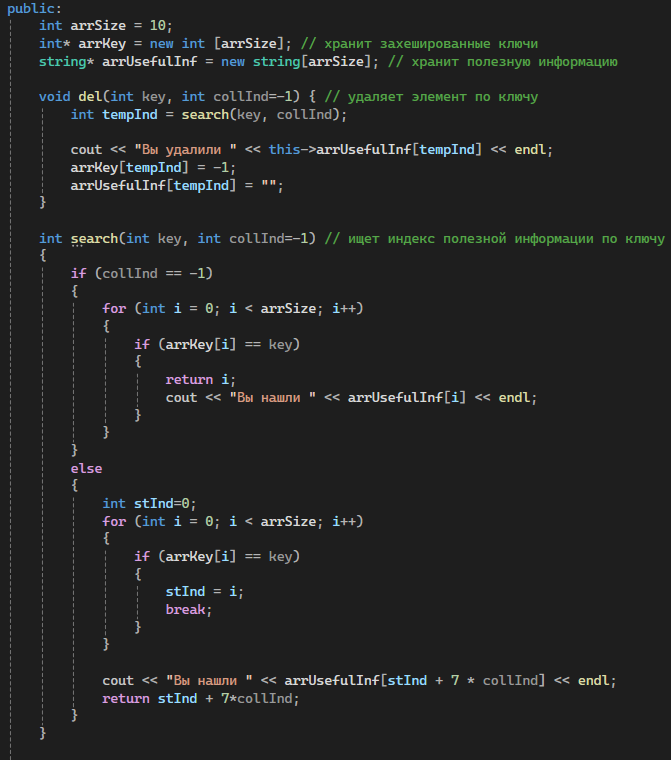


**Рисунок 3 – Конструктор хеш-таблицы**

Для работы с хеш-таблицей был выделен отдельный класс, так как в данной работе это более рациональный подход к решению поставленной задачи: более простая реализация различных методов и возможность создавать множество независимых друг от друга экземпляров данного класса

В первую очередь задаётся длина хеш-таблицы и заполнение заранее пустых ячеек определёнными значениями для отслеживания возможности вставки элементов.

Теперь же распишем все методы нашего класса, которые потребуются в дальнейшем выполнении задания (рис. 4-6).



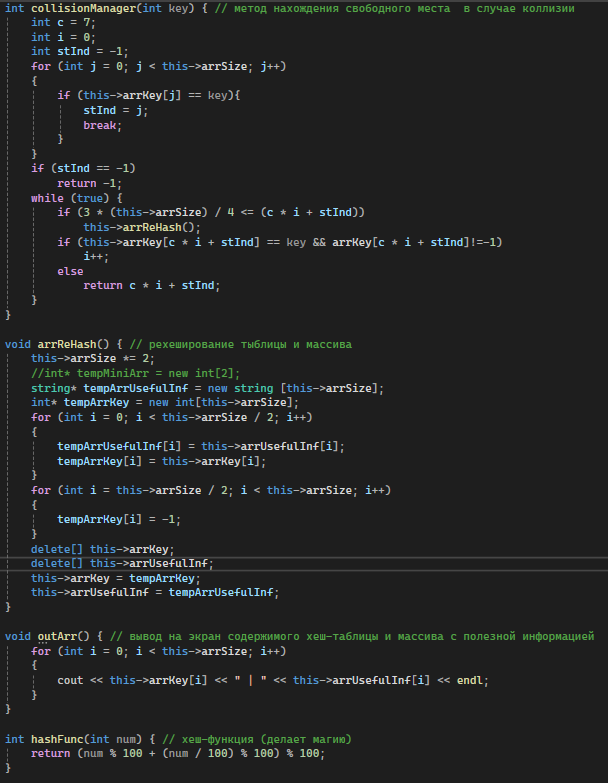
**Рисунок 4 – 1-ая часть всех методов класса хеш-таблицы**

На данной иллюстрации (рис4) изображены все необходимые для правильной работы класса поля, а также методы удаления элементов по ключу (del) и поиска по ключу (search). del и search устроены таким образом, что они будут искать и удалять элементы в зависимости от их коллизии, которую приходится, либо где-то хранить, либо постоянно находить, либо попросту знать.



**Рисунок 5 – 2-ая часть всех методов класса хеш-таблицы**

На текущем же рисунке можно увидеть полноценный метод добавления элементов в нашу хеш-таблицу (add). Метод add получает на вход добавляемую строку и совершает проверку на наличие коллизий, и в самом конце, удостоверившись, что текущая ячейка свободна, добавляет в неё элемент в зависимости от образовавшейся коллизии.



**Рисунок 6 – 3-я часть всех методов класса хеш-таблицы**

И последней частью всех методов является рис. 6, где изображены метод управления коллизией (collisionManager), метод рехеширования таблицы (arrReHash), метод вывода содержимого таблицы на экран (outArr) и метод хеш-функции.

collisionManager лишь вызывается в прочих методах в качестве помощника, ищущего образование различных коллизий.

arrReHash также вызывается исключительно в прочих методах для расширения границ таблицы в зависимости от сложившейся ситуации.

С outArr всё гораздо проще – он выводит на экран всё содержимое имеющейся хеш-таблицы (выводятся даже пустые ячейки).

А hashFunc – это всего-навсего фантазия автора данного кода (то есть меня), которая позволяет создавать уникальные ключи с желательной периодичностью.

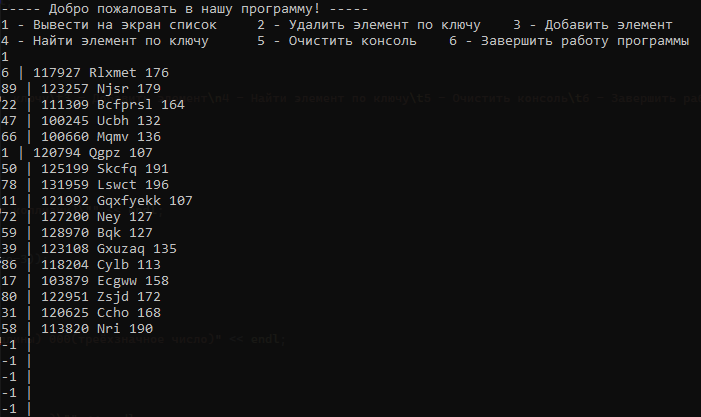
Единственное, что осталось добавить, так это UI и на рис. 7 как раз-таки представлен код реализующий тот самый UI.



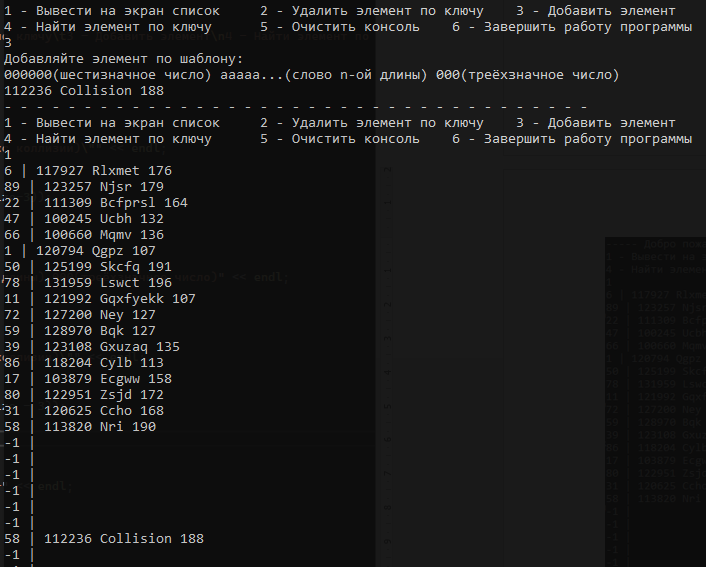
**Рисунок 7 – Реализация в коде UI**

В данном интерфейсе предусмотрены 6 различных ситуаций и, конечно же, если условия в них не будут соблюдены то программа даст сбой, поэтому мы предполагаем, что работаем с заранее обработанными данными.

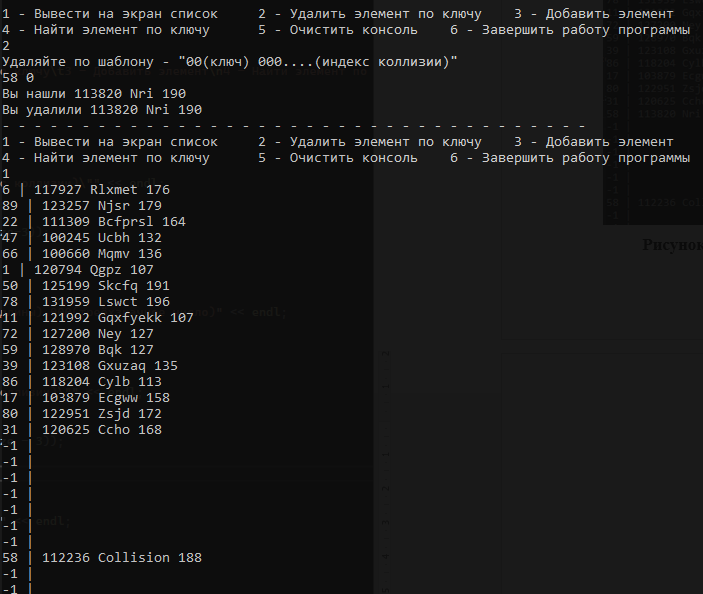
Теперь стоит перейти к тестированию нашей программы. Результаты тестов будут приведены на рис. 8 – 11.



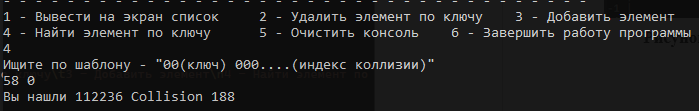
**Рисунок 8 – Вывод содержимого хеш-таблицы на экран**

****

**Рисунок 9 – Добавление элемента в таблицу с учётом коллизии**

****

**Рисунок 10 – Удаление элемента из таблицы с учётом коллизии**

****

**Рисунок 11 – Нахождение элемента с учётом коллизии**

Тесты проведены и, как можно заметить, успешно – программы работает исправно и выдаёт желаемый результат без сбоев, что и требовалось доказать в качестве работоспособности данной программы.

**ЧАСТЬ 6.2**

**Условие**

Разработайте приложения в соответствии с заданиями в индивидуальном варианте (п.2).

В отчёте в разделе «Математическая модель решения (описание алгоритма)» разобрать алгоритм поиска на примере. Подсчитать количество сравнений для успешного поиска первого вхождения образца в текст и безуспешного поиска.

Определить функцию (или несколько функций) для реализации алгоритма поиска. Определить предусловие и постусловие.

Сформировать таблицу тестов с указанием успешного и неуспешного поиска, используя большие и небольшие по объему текст и образец, провести на её основе этап тестирования.

Оценить практическую сложность алгоритма в зависимости от длины текста и длины образца и отобразить результаты в таблицу (для отчета).

**ЗАДАНИЕ 1**

**Формулировка задачи**

Дано предложение, состоящее из слов. Найти самое длинное слово предложения, первая и последняя буквы которого одинаковы.

**Математическая модель решения**

Мы будем получать на вход строку с разными разделителями, от которых мы впоследствии избавимся при помощи нашего собственного метода, который будет искать разделители между словами.

В конечном итоге вся работа будет заключатся в прохождении по строке слева направо в поиске слов, чей первый и последний символ одинаковы. Мы добьёмся распознавания этих символов при помощи отдельных полей, отвечающих за начало и конец слов.

**Код программы с комментариями**

На рис. 12 представлен код программы для решения задания 1.

Сначала идёт метод find, осуществляющий поиск разделителей и возвращает значение индекса этого разделителя.

Метод же search1 ответственен за поиск самого длинного слова с повторяющимися первым и последним символами. Поля longest и tempStr нужны для хранения длиннейшего слова и временного слова, которое будет выведено проверку, соответственно. Поля start и end служат ограничителями начала и конца слова.

Работа метода следующая: до тех пор, пока мы не дошли до конца строки, идёт проверка первого и последнего символа слов, с последующим нахождением их длины в удовлетворительном случае, а также с поиском следующего разделителя. В случае нахождения конца строки идет простая проверка первого и последнего символа слова. И в конце возвращается самое длинное слово с повторяющимися первым и последним символами.

****

**Рисунок 12 – Код программы для задания 1**

**Результаты тестирования**

Сделаем таблицу 1 с результатами всех вычислений.

Таблица 1 – Результаты тестирования с маленьким и большим текстами

|  |  |  |
| --- | --- | --- |
|  | Кол-во итераций | |
| Маленький текст | Большой текст |
| Худший случай | 24 | 350 |
| Лучший случай | 14 | 50 |

Всевозможные случаи будут представлены на рис. 13-16.

Для маленького текста возьмём худший случай как «jij somotos ege sfadds abba dfdffgdd», где мы будем обрабатывать каждое слово, ведь все они имеют одинаковые первый и последний символы, а также увеличиваются с каждым последующим словом. А лучший случай – «jii somotoo egg sfaddd abbb dfdffgdd», где только одно слово является искомым.

****

**Рисунок 13 – Результат работы первой программы в худшем случае с маленьким текстом**

****

**Рисунок 14 – Результат работы первой программы в лучшем случае с маленьким текстом**

Как можно заметить, кол-во итераций уменьшилось ровно на длину проверяемых слов, это говорит нам о том, что в худшем случае мы проходимся по всем словам, а в лучшем – всего по одному.

Теперь же рассмотрим текст побольше: «a aa aaa aaaa aaaaa aaaaaa aaaaaaa aaaaaaaa aaaaaaaaa aaaaaaaaaa aaaaaaaaaaa aaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaaaaaa baaaaaaaaaaaaaaaaaaaaaaab», а за лучший вариант возьмём « ab ab aab aaab aaaab aaaaab aaaaaab aaaaaaab aaaaaaaab aaaaaaaaab aaaaaaaaaab aaaaaaaaaaab aaaaaaaaaaaab aaaaaaaaaaaaab aaaaaaaaaaaaaab aaaaaaaaaaaaaaab aaaaaaaaaaaaaaaab aaaaaaaaaaaaaaaaab aaaaaaaaaaaaaaaaaab aaaaaaaaaaaaaaaaaaab aaaaaaaaaaaaaaaaaaaab aaaaaaaaaaaaaaaaaaaaab aaaaaaaaaaaaaaaaaaaaaab aaaaaaaaaaaaaaaaaaaaaaab baaaaaaaaaaaaaaaaaaaaaaab».

****

**Рисунок 15 – Результат работы первой программы в худшем случае с большим текстом**

****

**Рисунок 16 – Результат работы первой программы в лучшем случае с большим текстом**

Всё аналогично предыдущим тестам: лучший случай = кол-во слов + длина самого большого слова, худший = кол-во слов + длина всех слов.

**ЗАДАНИЕ 2**

**Формулировка задачи**

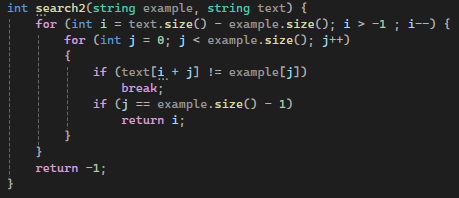
Используя алгоритм Кнута-Мориса-Пратта, найти индекс последнего вхождения образца в текст.

**Математическая модель решения**

Мы идём по каждому элементу нашего текста и поочерёдно сравниваем символы образца и элемента текста, в конечном итоге либо получая, либо нет искомый образец из текста.

**Код программы с комментариями**

На рис. 17 показан код программы для задания 2.



**Рисунок 17 – Код алгоритма Кнута-Мориса-Пратта**

Как и говорилось в математической модели, мы поочерёдно проходимся по всем словам нашего текста в поиске желанного образца и, если в какой-то момент текущее слово не совпадает с образцом, мы тут же прекращаем проверку этого слова. В случае, если мы ничего не находим, то программа возвращает -1, как индекс несуществующего образца.

**Результаты тестирования**

Сделаем таблицу 2 с результатами всех вычислений.

Таблица 2 – Результаты тестирования с маленьким и большим текстами

|  |  |  |
| --- | --- | --- |
|  | Кол-во итераций | |
| Маленький текст | Большой текст |
| Худший случай | 45 | 829 |
| Лучший случай | 5 | 50 |

Результаты всех тестов представлены на рис. 18-21.

В качестве примера худшего случая для маленького текста примем текст «slov slov slov slov slov slov»с образцом«slovo».



**Рисунок 18 – Результат работы программы в худшем случае с маленьким текстом**

А в качестве лучшего случая примем текст «slov slov slov slov slov slovo».



**Рисунок 19 – Результат работы программы в лучшем случае с маленьким текстом**

Как можно заметить, кол-во итерация составило всего длину образца, а также мы получили индекс начала искомого слова.

В качестве большого текста возьмём условие практики 6. А в качестве образца укажем слово «рыба».



**Рисунок 20 – Результат работы программы в худшем случае с большим текстом**

Данный случай является приблизительно худшим, так как не состоит исключительно из слов «рыб», что вызвало бы дополнительные итерации.



**Рисунок 21 – Результат работы программы в лучшем случае с большим текстом**

Как видно из предыдущего образца кол-во итераций составило всего длину самого образца.

Из выше перечисленных примеров можно сделать вывод, что в худшем случае кол-во итерация = (длина образца\*2 - 1)\*(кол-во слов – 1). А лучшем случаем = длина образца.

**ВЫВОД**

В практической работе 6.1 изучили методы работы с хеш-таблицей, а также выработали навыки необходимые для создания подобного рода таблиц, решив поставленную задачу.

В практической работе 6.2 разработали методы поиска слов в тексте при помощи различных инструментов, а также осуществили оценку полученных результатов в ходе пробных тестов программ.

**СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ**

1. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.

2. Документация по языку С++ [Электронный ресурс]. URL: https://docs.microsoft.com/ruru/cpp/cpp/ (дата обращения 01.09.2021).

3. 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).