|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | | | |
| Институт информационных технологий (ИТ) | |
| Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ) | |

|  |  |  |  |
| --- | --- | --- | --- |
| **ОТЧЁТ ПО ПРАКТИЧЕСКИМ РАБОТАМ** | | | |
| **по дисциплине «Структуры и алгоритмы обработки данных (часть 2/2)»** | | | |
|  | | | |
| Выполнил студент группы ИНБО-21-23 | | Албахтин И.В. | |
|  | |  | |
| Принял  *Ассистент* | | Муравьева Е.А | |
| Практические работы выполнены | «\_\_\_»\_\_\_\_\_\_\_2024 г. | | (подпись студента) | |
| «Зачтено» | «\_\_\_»\_\_\_\_\_\_\_2024 г. | | (подпись преподавателя) | |
|  |  | |  | |

Москва 2023

**ХОД РАБОТЫ**

**Задание №1**

Ответьте на вопросы:

1. **Что такое строка, её префикс и суффикс?**

Строка (слово) – это последовательность символов из некоторого алфавита.

Подстрока X называется префиксом строки Y, если есть такая подстрока Z, что Y=XZ. Причем сама строка является префиксом для себя самой (так как найдется нулевая строка L, что X=XL ). Например, подстрока ab является префиксом строки abcfa.

Подстрока X называется суффиксом строки Y, если есть такая подстрока Z, что Y=ZX. Аналогично, строка является суффиксом себя самой. Например, подстрока bfg является суффиксом строки vsenfbfg.

1. **Объясните идею алгоритма последовательного (наивного) поиска шаблона в строке. Какая асимптотическая сложность наивного поиска подстроки в строке?**

Алгоритм последовательного (наивного) поиска в строке предназначен для нахождения подстроки в строке. Этот алгоритм проходит по исходной строке и последовательно сравнивает подстроку с соответствующими подмассивами (или символами) в строке, пока не найдет совпадение или не дойдет до конца строки.

Время выполнения в худшем случае O(n \* m), где n — длина строки, а m — длина подстроки. Это связано с тем, что для каждой позиции в строке может потребоваться проверить все символы подстроки.

1. **В чём идея поиска образца алгоритмом Бойера–Мура?**

**Основные шаги алгоритма Бойера-Мура:**

Препроцессинг: В этом шаге строится таблица смещений, которая определяет насколько далеко можно сдвинуть образец в случае несовпадения символа. - Для каждого символа алфавита в образце определяется максимальное смещение, на которое можно сдвинуть образец, если этот символ не совпадает с текущим символом в поисковой строке. - Если символ не встречается в образце, то смещение равно длине образца. - Если символ встречается в образце, то смещение равно расстоянию от последнего вхождения символа до конца образца.

Поиск: Начиная с конца строки, сравниваем символы образца с соответствующими символами строки. - Если символы совпадают, продолжаем сравнивать следующие символы образца. - Если символы не совпадают, используя таблицу смещений, выполняем сдвиг образца на максимальное смещение для текущего символа. - Если происходит сдвиг, сравнение начинается снова с последнего символа образца. - Шаги 2 и 3 повторяются, пока не найдено полное совпадение образца или не достигнут конец строки.

1. **В чём идея поиска образца алгоритмом Рабина-Карпа?**

Основная идея заключается в том, чтобы сравнивать "хэш-коды" подстрок с "хэш-кодом" образца, чтобы быстро находить совпадения.

1. **Назовите асимптотическую сложность алгоритма Бойера–Мура, Рабина-Карпа, КнутаМориса-Пратта поиска подстроки в строке по времени и памяти. Что лучше, в каком случае и почему?**

1. Временная сложность алгоритм Бойера–Мура

- Лучший случай: O(n/m), где n — длина текста, а m — длина образца. Происходит очень эффективное сравнение, когда быстро сдвигаемся по тексту.

- Средний случай: O(n) в большинстве случаев.

- Худший случай: O(n m), в ситуациях с большим количеством совпадающих символов.

2. Временная сложность алгоритм Рабина-Карпа

- Средний случай: O(n + m), с использованием хэширования для быстрого сравнения. Если хеш-функция хорошо сбалансирована и количество коллизий минимально.

- Худший случай: O(n m) в случае множества коллизий, когда хеш-коды совпадают, но строки не совпадают.

3. Временная сложность алгоритм Кнута-Мориса-Пратта (КМП)

- Средний и худший случай: O(n + m), так как используется префиксная таблица для ускорения сравнения.

1. **Объясните идею алгоритма Ахо–Корасика. Приведите его вычислительную и ёмкостную сложность.**

**Идея алгоритма**

1. **Построение префиксного дерева (Trie):**

- Вначале алгоритм строит префиксное дерево (также называемое Trie) для всех образцов. Каждая вершина дерева соответствует символу из образца, и путь от корня до вершины представляет собой префикс одного или нескольких образцов.

1. **Добавление "неудачных" переходов:**

- Затем для каждой вершины дерева добавляются "неудачные" переходы (или ссылки на другие вершины), позволяющие ускорить процесс поиска. Эти переходы обозначают, куда переместиться в Trie, если совпадение текущего символа не найдено. Эффектно создается возможность "переключиться" на другой предшествующий префикс, когда текущий символ не совпадает.

1. **Поиск:**

- После того как Trie и переходы готовы, алгоритм проходит по тексту и использует информацию из Trie для одновременного поиска всех образцов. Если текущий символ текста совпадает с символом в Trie, мы просто переходим к следующему символу. Если нет, используем "неудачный" переход, чтобы определить, с какого места в Trie продолжить поиск.

**Задание 2**

Разработайте приложения в соответствии с заданиями в индивидуальном варианте, указанными в таблице

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).

**Задания варианта:**

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

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

**Решение задания 2**

Решение первого задания представлено на рисунке 1.

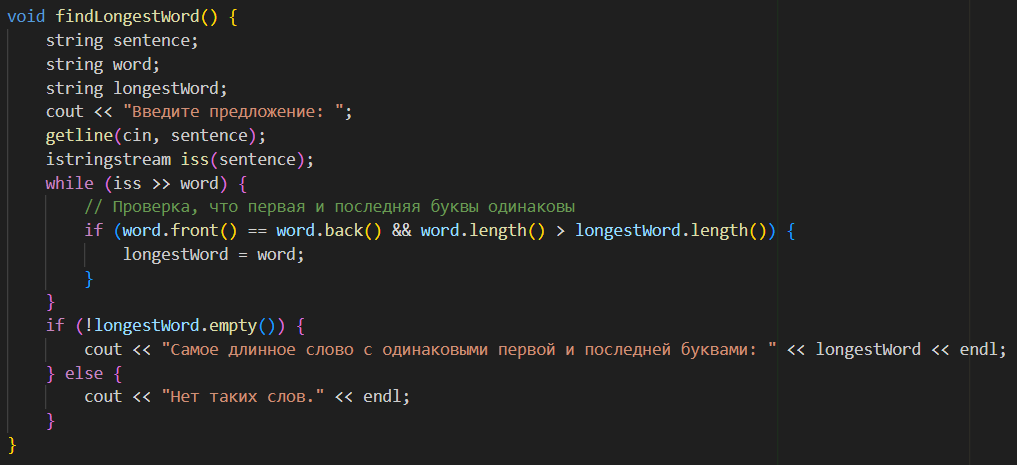


Рисунок 1 – Решение задания 1

Тестирование программы 1 показано на рисунке 2.

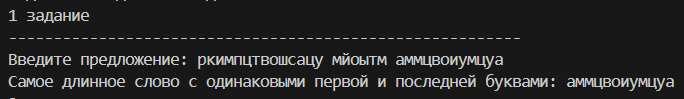


Рисунок 2 – Решение задания 1

Для решения задания 2 с помощью КМП необходимы две дополнительные функции: функция для вычисления префиксного массива и функция для поиска последнего вхождения образца в тексте.

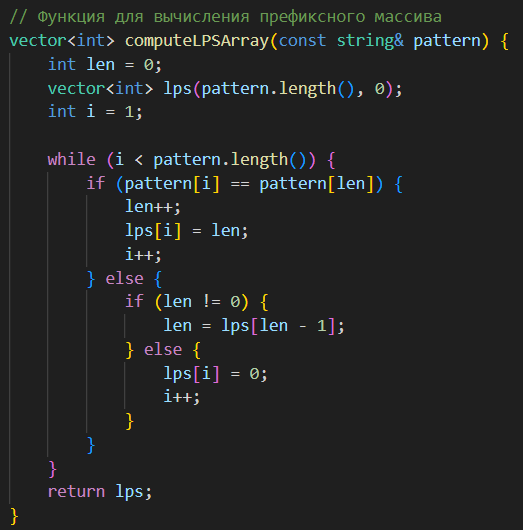


Рисунок 3 – Функция для вычисления префиксного массива



Рисунок 4 – Функция для поиска последнего вхождения образца в тексте

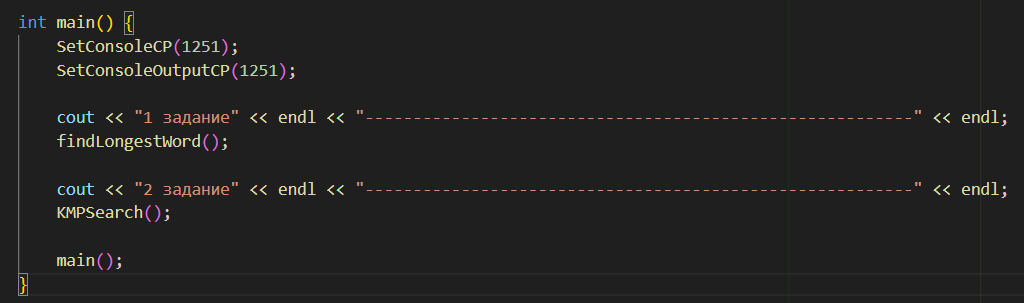


Рисунок 5 – Основная функция

Тестирование программы 2 показано на рисунке 6.

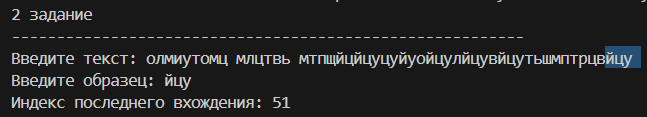


Рисунок 6 – Тестирование программы

**ВЫВОД**

В ходе работы были разработаны два алгоритма: для нахождения самого длинного слова с одинаковыми первой и последней буквами и для поиска последнего вхождения образца в текст с использованием алгоритма Кнута-Морриса-Пратта. Проведены тестирования, на основании которых мы поняли, что все работает корректно.

Анализ количества сравнений подтвердил эффективность КМП. Практическая сложность показала линейную зависимость времени выполнения от длины текста и образца. Тестирование продемонстрировало высокую производительность алгоритмов на разных объемах данных.