МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра информатики и программирования

ОЦЕНКА СЛОЖНОСТИ ПРЕФИКС-ФУНКЦИИ, Z-ФУНКЦИИ И АЛГОРИТМА КМП

ОТЧЕТ

студентки 2 курса 211 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета компьютерных наук и информационных технологий
Никитенко Яны Валерьевны

Саратов 2025

СОДЕРЖАНИЕ

1 Префикс-функция

1.1 Текст программы

```
    // Префикс функция
vector<int> Prefix_Function(const string& s) {
    int n = s.length();
    vector<int> pi(n, 0); //вектор для хранения значений префикс-функции

    for (int i = 1; i < n; ++i) {
        int j = pi[i - 1]; //длина предыдущего префикса

        //пока символы не совпадают и j больше 0, уменьшаем j
        while (j > 0 && s[i] != s[j]) {
            j = pi[j - 1];
        }

        //если символы совпадают, увеличиваем j
        if (s[i] == s[j]) {
            j++;
        }

        pi[i] = j; //записываем значение префикс-функции
    }

    return pi;
}
//
```

1.2 Анализ

Инициализация переменной в строке 2 работает за константу. Создание вектора в строке 3 работает за линию. Далее цикл работает за n.

Эта сложность умножается на сумму всех сложностей внутри цикла.

В худшем случае цикл while из строки может работать O(n) раз за одну итерацию, но в среднем каждый while работает за O(1). Остальные операции

внутри внешнего цикла выполняются за константу. Префикс-функция каждый шаг возрастает максимум на единицу и после каждой итерации while уменьшается хотя бы на единицу. Значит, суммарное количество операций:

$$T = O(1) + O(n) + O(n) * (O(1) + O(1) + O(1) + O(1) + O(1)) \approx O(n)$$

2 Z-функция

2.1 Текст программы

```
    // Z функция
vector<int> Z_function(const std::string& s) {
    int n = s.length();
    std::vector<int> Z(n, 0);
    int L = 0, R = 0;

    for (int i = 1; i < n; ++i) {
        if (i > R) {
            L = R = i;
            while (R < n && s[R] == s[R - L]) {
                R++;
            }
            Z[i] = R - L;
            R--;
        }
        else {
            int k = i - L;
            if (Z[k] < R - i + 1) {
                Z[i] = Z[k];
            }
            else {
                L = i;
                while (R < n && s[R] == s[R - L]) {
                    R++;
                }
                Z[i] = R - L;
                R--;
            }
        }
    }
    return Z;
}
```

//

2.2 Анализ

Инициализация переменных в 2 и 4 строки происходит за константу. Инициализация вектора в строке 3 работает за O(n). Цикл в строке 5 работает за O(n), каждая операция внутри домножается на O(n) по правилу умножения вложенных циклов.

В основном сложность алгоритма зависит от цикла .

В алгоритме мы делаем столько же действий, сколько раз сдвигается правая граница z-блока — а это O(n) в сумме по всем проходам цикла. Таким образом, в среднем на каждой итерации этот цикл будет работать за какую-то константу. Общая сложность алгоритма таким образом составит: $T = O(1) + O(n) + O(1) + O(n)(O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1)) + O(1) + O(1) = O(1) + O(n) + O(n)O(1) + O(1) \approx O(n)$

3 Алгоритм Кнута-Морриса Пратта (КМП)

3.1 Текст программы

```
    // Префикс функция
vector<int> Prefix_Function(const string& s) {
    int n = s.length();
    vector<int> pi(n, 0); //вектор для хранения значений префикс-функции

    for (int i = 1; i < n; ++i) {
        int j = pi[i - 1]; //длина предыдущего префикса

        //пока символы не совпадают и j больше 0, уменьшаем j
        while (j > 0 && s[i] != s[j]) {
            j = pi[j - 1];
        }

        //если символы совпадают, увеличиваем j
        if (s[i] == s[j]) {
            j++;
        }

        pi[i] = j; //записываем значение префикс-функции
    }

    return pi;
}
//

    //
void KnutMorrisPratt(const string& text, const string& pattern) {
    vector<int> pi = Prefix_Function(pattern);
    int n = text.length();
    int m = pattern.length();
    int j = 0; // индекс для pattern
```

```
    for (int i = 0; i < n; i++) {
        while (j > 0 && text[i] != pattern[j]) {
            j = pi[j - 1]; // откат к предыдущему префиксу
        }
        if (text[i] == pattern[j]) {
            j++;
        }
        if (j == m) { // найдено совпадение
            cout << (i - m + 1) << endl;
            j = pi[j - 1]; // продолжаем поиск
        }
    }
}
//
```

3.2 Анализ

Инициализация переменных в функции KnutMorrisPratt в строках 3-4 работает за $O(1)$.

Инициализация вектора работает за $O(N)$.

Функция $Prefix_Function$ работает за линейное время $(O(N))$.

Инициализация переменной в строке 5 выполняется за константу.

Цикл for работает за O(N).

Тогда общая временная сложность $T = O(1) + O(N) + O(M) + O(N)O(1) = O(N) + O(M) = O(N + M)$. Так как $M \leq N$, то можно сказать, что алгоритм работает не больше чем за $T = O(N + M) \approx O(2 * N) = O(N)$.