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

ОЦЕНКА СЛОЖНОСТИ ПРЕФИКС-ФУНКЦИИ, Z-ФУНКЦИИ И АЛГОРИТМА КНУТА-МОРРИСА-ПРАТТА

ОТЧЁТ

студента 2 курса 251 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Карасева Вадима Дмитриевича

Проверено:
доцент, к. ф.-м. н. _____ М. И. Сафрончик

Саратов 2025

СОДЕРЖАНИЕ

1 Префикс-функция

```
1  vector<int> prefix_function(string s) {
2      int n = (int) s.size();  // O(1)
3      vector<int> p(n, 0);     // O(n)
4      for (int i = 1; i < n; i++) {   //O(n)
5          int cur = p[i - 1];  // O(1)
6          while (s[i] != s[cur] && cur > 0) // O(1) в среднем
7              cur = p[cur - 1];  // O(1)
8          if (s[i] == s[cur])    // O(1)
9              p[i] = cur + 1;    // O(1)
10     }
11     return p; // O(1)
12 }
```

Инициализация переменной в строке 2 работает за константу.

Создание вектора в строке 3 работает за линию.

Далее цикл из строки 4 работает за n. Эта сложность умножается на сумму всех сложностей внутри цикла 4.

В худшем случае цикл $while$ из строки 6 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

```
1  vector<int> z_function(string s) {
2      int n = (int)s.size();  // O(1)
3      vector<int> z(n, 0);    // O(n)
4      int l = 0, r = 0;       // O(1)
5      for (int i = 1; i < n; i++) {  // O(n)
6          if (i <= r)   // O(1)
7              z[i] = min(r - i + 1, z[i - l]);  // O(1)
8          while (i + z[i] < n && s[z[i]] == s[i + z[i]]) // O(1) в среднем
9              z[i]++;  // O(1)
10         if (i + z[i] - 1 > r) {  // O(1)
11             r = i + z[i] - 1;    // O(1)
12             l = i;   // O(1)
13         }
14     }
15     z[0] = n;   // O(1)
16     return z;   // O(1)
17 }
```

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 строке 8. 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:

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

3 Алгоритм Кнута-Морриса Пратта (КМП)

```
void KMPSearch(string pat, string txt) { // pat - искомая строка, txt - текст
        int M = pat.length();  // O(1)
        int N = txt.length();  // O(1)
        vector<int> lps(M);    // O(N)
        computeLPSArray(pat, M, lps); // O(M)
        int i = 0;   // O(1)
        int j = 0;   // O(1)
        while ((N - i) >= (M - j)) {  // O(N)
                if (pat[j] == txt[i]) { // O(1)
                        j++;  // O(1)
                        i++;  // O(1)
                }
                if (j == M) {  // O(1)
                        cout << "Found at index " << i - j << '\n';  // O(1)
                        j = lps[j - 1];  // O(1)
                }
                else if (i < N && pat[j] != txt[i]) {  // O(1)
                        if (j != 0)  // O(1)
                                j = lps[j - 1];  // O(1)
                        else
                                i = i + 1;  // O(1)
        } } }

void computeLPSArray(string pat, int M, vector<int> lps) {
        int len = 0; // O(1)
        lps[0] = 0;
        int i = 1;  // O(1)
        while (i < M) {  // O(M)
                if (pat[i] == pat[len]) {  // O(1)
                        len++;  // O(1)
                        lps[i] = len;  // O(1)
                        i++;  // O(1)
                }
                else {
                        if (len != 0)  // O(1)
                                len = lps[len - 1];   // O(1)
                        else {
                                lps[i] = 0;  // O(1)
                                i++;  // O(1)
        } } } }
```

Инициализация переменных в строках 2-3 работает за $O(1)$.

Инициализация вектора работает за $O(N)$.

Функция $computeLPSArray$ работает за $O(M)$ (из-за цикла в строке 28).

Инициализация переменных в строках 6-7 выполняется за константу.

Цикл в строке 8 работает за $O(N)$, все операции внутри него — константные.

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 $T = O(N + M) \approx O(2 * N) = O(N)$.