

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	3
1.1	Текст программы	3
1.2	Анализ	3
2	Z-функция	5
2.1	Текст программы	5
2.2	Анализ	6
3	Алгоритм Кнута-Морриса Пратта (КМП)	7
3.1	Текст программы	7
3.2	Анализ	8

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

Алгоритм префикс-функции имеет временную сложность $O(n)$, где n - длина строки.

Доказательство:

— Внешний цикл: $n - 1$ итераций

- Внутренний цикл `while`: кажется $O(n)$ в худшем случае, но амортизированно $O(1)$
 - Ключевое наблюдение: $\pi[i] \leq \pi[i - 1] + 1$
 - Переменная j (текущая длина префикса) увеличивается максимум на 1 за итерацию
 - Каждое уменьшение j в `while` соответствует предыдущему увеличению
 - Суммарно количество уменьшений \leq суммарному количеству увеличений $\leq n$
- Итого: $O(n)$ операций.

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

Алгоритм Z-функции имеет временную сложность $O(n)$.

Доказательство:

- Внешний цикл: $n - 1$ итераций
 - Ключевое наблюдение: указатель R (правая граница Z-блока) движется только вправо
 - На каждой итерации R увеличивается, причем $R \leq n$
 - Суммарное движение R : $O(n)$
 - Все остальные операции - константные
- Итого: $O(n)$ операций.

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

Алгоритм КМП имеет временную сложность $O(n + m)$, где:

- n - длина текста
 - m - длина образца
- Доказательство:**
- Построение префикс-функции: $O(m)$
 - Поиск в тексте: внешний цикл - n итераций
 - Амортизированный анализ: переменная j увеличивается максимум на 1 за итерацию
 - Суммарное количество откатов j в while есть $O(n)$
 - Все операции внутри циклов - константные
- Итого: $O(n + m)$.