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 КМП

ОТЧЁТ

студента 2 курса 251 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Толстова Роберта Сергеевича

Проверено:

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

Саратов 2025

СОДЕРЖАНИЕ

1 Префикс-функция

```
std::vector<int> prefix(std::string s) {
  std::vector<int> result(s.length());
  result[0] = 0;

  for (int i = 1; i < s.length(); i++) {
    int k = result[i - 1];

    while (k > 0 && s[i] != s[k]) {
      k = result[k - 1];
    }
    if (s[i] == s[k]) {
      k++;
    }
    result[i] = k;
  }
  return result;
}
```

Временная сложность данного алгоритма:
T = O(n) + O(1) + O(n) · (O(1) + O(1) + O(1) + O(1) + O(1)) ≈ O(n), где n — длина строки s

Казалось бы, мы имеем вложенность и следует ждать $O(n^2)$. Но на деле `while` выполнится не более n раз за всю работу процедуры. Получим O(n + n) = O(n)

2 Z-функция

```
std::vector<int> z(std::string s) {
  std::vector<int> result(s.length());

  int left = 0, right = 0;

  for (int i = 1; i < s.length(); i++) {
    result[i] = std::max(0, std::min(right - i, result[i - left]));

    while (i + result[i] < s.length() && s[result[i]] == s[i +
result[i]]) {
      result[i]++;
    }
    if (i + result[i] > right) {
      left = i;
      right = i + result[i];
    }
  }
  return result;
}
```

Поскольку каждый символ строки рассматривается не более двух раз: один раз при увеличении i и один раз внутри цикла while, то временная сложность алгоритма Z-функции:
$T = O(n) + O(n) \cdot (O(1) + O(1) + O(1) + O(1)) + O(1) = O(n) + O(n) \cdot O(1) \approx O(n)$, где n — длина строки s.

3 КМП

```
std::vector<int> kmp(std::string s, std::string pattern) {
  int patternLength = pattern.length();
  std::vector<int> result(s.length());
  std::vector<int> p = prefix(pattern + "#" + s);

  int count = 0;
  for (int i = patternLength; i < p.size(); i++) {
    if (p[i] == patternLength) {
      result[count++] = i - 2 * patternLength + 1;
    }
  }
  return result;
}
```

Пусть m – длина искомого паттерна, n – длина строки.

Формула временной сложности: T = O(1) + O(n) + O(m + n + 1) + O(1) + O(n + m + 1) · (O(1) + O(1) + O(1) + O(1) + O(1)) + O(1) = O(1) + O(n) + O(n + m + 1) + O(1) + O(m + n + 1) · O(1) + O(1) = O(m + n)

Таким образом, алгоритм работает линейно по сумме длины текста и шаблона, что делает его одним из самых эффективных для поиска подстрок.