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

АНАЛИЗ АЛГОРИТМА БОЙЕРА-МУРА

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 Пример программы

```
#include <iostream>
#include <vector>
#include <string>
#include <unordered_map>

using namespace std;

// Функция для создания таблицы "плохого символа"
vector<int> buildBadCharTable(const string& pattern) {
    const int ALPHABET_SIZE = 256;
    vector<int> badCharTable(ALPHABET_SIZE, -1);

    for (int i = 0; i < pattern.size(); i++) {
        badCharTable[(unsigned char)pattern[i]] = i;
    }

    return badCharTable;
}

// Алгоритм Бойера - Мура (только эвристика плохого символа)
vector<int> boyerMooreSearch(const string& text, const string& pattern) {
    vector<int> result;
    int n = text.size();
    int m = pattern.size();

    if (m == 0 || n < m) return result;

    vector<int> badChar = buildBadCharTable(pattern);

    int shift = 0;

    while (shift <= (n - m)) {
        int j = m - 1;

        // Сравнение с конца шаблона
        while (j >= 0 && pattern[j] == text[shift + j]) {
            j--;
        }

        // Если шаблон найден
```

```
41          if (j < 0) {
42              result.push_back(shift);
43              shift += (shift + m < n) ? m - badChar[(unsigned char)text[shift +
                ↪  m]] : 1;
44          } else {
45              // Смещение по эвристике плохого символа
46              shift += max(1, j - badChar[(unsigned char)text[shift + j]]);
47          }
48      }
49
50      return result;
51  }
52
53  int main() {
54      string text, pattern;
55
56      cout << "Введите текст для поиска: ";
57      getline(cin, text);
58
59      cout << "Введите шаблон для поиска: ";
60      getline(cin, pattern);
61
62      vector<int> matches = boyerMooreSearch(text, pattern);  // выполняем поиск
63
64      if (matches.empty()) {  // если совпадений нет
65          cout << "Шаблон не найден в тексте." << endl;
66      } else {  // если совпадения есть
67          cout << "Найденные позиции: ";
68          for (int pos : matches) {
69              cout << pos << " ";
70          }
71          cout << endl;
72      }
73
74      return 0;
75  }
```

2 Анализ сложности

Внутри основного цикла может быть выполнено не более $\frac{n}{m}$ сдвигов (каждый сдвиг хотя бы на одну позицию). Кроме того, при каждом сдвиге мы можем использовать информацию из таблицы "плохих символов"для определения дополнительного сдвига. - В худшем случае для каждого сдвига может потребоваться $O(m)$ времени, чтобы определить этот сдвиг. Таким образом, временная сложность алгоритма Бойера-Мура в худшем случае составляет:

$O(m) + O(n) * O(m) = O(nm)$

Исходный текст: $bb...bb$

Шаблон: $abab...abab$

Из-за того, что все символы b из текста повторяются в шаблоне $m/2$ раз, эвристика хорошего символа будет пытаться сопоставить шаблон в каждой позиции (суммарно, n раз), а эвристика плохого символа в каждой позиции будет двигать строку $m/2$ раз. Итого, $O(n * m)$

где n — длина исходного текста, n — длина шаблона, m — размер алфавита.

Это означает, что в худшем случае алгоритм будет выполняться за время, пропорциональное произведению длины текста на длину шаблона.

В лучшем случае алгоритм использует преимущества таблицы "плохих символов"для значительных сдвигов, что уменьшает количество сравнений и сдвигов: При каждом сдвиге шаблон смещается на значительное расстояние, m позиций сразу, если символ, следующий за совпадением, отсутствует в шаблоне. Количество сдвигов ограничено $\frac{n}{m}$. В лучшем случае:

$O(m) + O(\frac{n}{m}) * O(m) = O(m) + O(n) = O(n)$

Лучший случай:

Исходный текст: $abracadabra$

Шаблон: $hhhh$

В таком случае будет временная сложность будет равняться $O(m) + O(\frac{n}{m}) = O(\frac{n}{m})$.

$O(n)$ - длина исходной строки.

$\frac{n}{m}$ - количество сдвигов шаблона по тексту. $O(m)$ - время на проверку символов шаблона на каждой позиции, но в лучшем случае это практически всегда константное время, так как шаблон сразу сдвигается на m позиций.