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
Толстова Роберта Сергеевича

Проверено:

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

Саратов 2025

СОДЕРЖАНИЕ

1 Анализ алгоритма Бойера – Мура

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

using namespace std;

#define alphabetSize 256

void genBadChar(string &needle, int size, int
badChar[alphabetSize]) {
  for (int i = 0; i < alphabetSize; i++)
    badChar[i] = size;
  for (int i = 0; i < size; i++)
    badChar[(int)needle[i]] = i;
}

vector<int> BM(string &haystack, string &needle) {
  int n = haystack.size();
  int m = needle.size();
  int badChar[alphabetSize];
  genBadChar(needle, m, badChar);
  vector<int> result;
  int s = 0;

  while (s <= (n - m)) {
    int i = m - 1;
    while (i >= 0 && needle[i] == haystack[s + i])
      i--;
    if (i < 0) {
```

```
      result.push_back(s);
      s += (s + m < n) ? 1 : 1;
    } else {
      s += max(1, i - badChar[haystack[s + i]]);
    }
  }

  return result;
}

int main() {
  string haystack, needle;
  cout << "String:" << endl;
  cin >> haystack;
  cout << "Substring:" << endl;
  cin >> needle;
  vector<int> ans = BM(haystack, needle);
  for (int pos : ans) {
    for (int i = pos; i < pos + (int)needle.length(); i++)
      cout << haystack[i];
    cout << " " << pos << endl;
  }

  return 0;
}
```

Проанализиируем функцию `genBasChar`: O(m), где m – длина паттерна

Цикл `$(s <= n - m)` — основной цикл сдвигов. В худшем случае количество сдвигов может быть до O(n)O(n).

Внутренний цикл `while (i >= 0 && needle[i] == haystack[s + i]) i--;` — сравнивает символы шаблона и текста справа налево. В худшем случае может сравнить все mm символов.

При несовпадении сдвиг определяется по таблице плохих символов: `s += max(1, i - badChar[haystack[s + i]])`. Это позволяет делать большие сдвиги, пропуская символы.

При полном совпадении добавляем позицию в результат и сдвигаемся на 1, чтобы не пропустить пересекающиеся вхождения.

Построение таблицы: O(m).

Основной цикл:

В худшем случае (например, при повторяющихся символах и плохой таблице сдвигов) алгоритм может сравнивать символы почти на каждом сдвиге, что даёт сложность O(nm).

На практике и в среднем благодаря эвристике плохого символа (и при добавлении эвристики хорошего суффикса) алгоритм работает за O(n).

Поиск всех вхождений не меняет асимптотику — мы просто продолжаем проходить текст и сохраняем найденные позиции.

Таким образом, в среднем случае сложность O(n), в худшем O(nm). Подготовка таблицы требует O(m)