Министерство науки и высшего образования РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский Авиационный Институт»

Национальный Исследовательский Университет

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №4

по курсу «Дискретный анализ»

Студент:	Хренникова А. С.
Группа:	М8О-208Б-19
Преподаватель:	Капралов Н. С.
Подпись:	
Оценка:	
Дата:	

Москва, 2020

Лабораторная работа №4

Задача: Необходимо реализовать один из стандартных алгоритмов поиска образцов для указанного алфавита.

Вариант алгоритма: Поиск одного образца при помощи алгоритма Бойера-Мура.

Вариант алфавита: Числа в диапазоне от 0 до 2^{32} - 1.

Москва, 2020

1 Описание

Алгоритм сравнивает символы шаблона pattern справа налево, начиная с самого правого, один за другим с символами исходной строки text. Если символы совпадают, производится сравнение предпоследнего символа шаблона и так до конца. Если все символы шаблона совпали с наложенными символами строки, значит, подстрока найдена, и поиск окончен. В случае несовпадения какого-либо символа (или полного совпадения всего шаблона) он использует две предварительно вычисляемых эвристических функций, чтобы сдвинуть позицию для начала сравнения вправо.

Таким образом для сдвига позиции начала сравнения алгоритм Бойера-Мура выбирает между двумя функциями, называемыми эвристиками хорошего суффикса и плохого символа (иногда они называются эвристиками совпавшего суффикса и стоп-символа). Так как функции эвристические, то выбор между ними простой — ищется такое итоговое значение, чтобы мы не проверяли максимальное число позиций и при этом нашли все подстроки равные шаблону.

2 Исходный код:

```
#include <iostream>
#include <algorithm>
#include <cmath>
#include <string>
#include <vector>
#include <sstream>
#include <cstdint>
#include <map>

std::vector<size_t> ZFunction(std::vector<std::uint32_t>& pattern) {
    size_t n = pattern.size(), l = 0, r = 0;
    std::vector<size_t> z(n);
    for (size_t i = 1; i < n; ++i) {
        if (i <= r) {
            z[i] = std::min(r - i + 1, z[i - l]);
        }
        while (i + z[i] < n && pattern[z[i]] == pattern[i + z[i]]) {
            ++z[i];
        }
        if (i + z[i] - 1 > r) {
            l = i;
             r = i + z[i] - 1;
        }
    }
    return z;
}

std::vector<size_t> NFunction(std::vector<std::uint32_t>& pattern) {
    std::reverse(pattern.begin(), pattern.end());
    std::vector<size_t> z = ZFunction(pattern);
    std::vector<size_t> n(z.size());
    for (size_t i = 0; i < pattern.size(); ++i) {
        n[i] = z[pattern.size() - i - 1];
    }
    std::reverse(pattern.begin(), pattern.end());
    return n;
}

std::pair<std::vector<size_t>, std::vector<size_t>> LFunction(std::vector<std::uint32_t>& pattern) {
    std::vector<size_t> n = NFunction(pattern);
    std::vector<size_t> l(n.size());
    std::vector<size_t> L(pattern.size() + 1);
    size_t j = 0;
    for (size_t i = 0; i < pattern.size() - 1; ++i) {
        if (n[i] != 0) {
            j = pattern.size() - n[i];
            l[j] = i;
        }
        if (n[i] == i + 1) {
            L[pattern.size() - i - 1] = i + 1;
        } else {
            L[pattern.size() - i - 1] = L[pattern.size() - i];
        }
    }
    return std::pair<std::vector<size_t>, std::vector<size_t>>(l, L);
}


void BM(std::vector<std::pair<std::pair<size_t, size_t>, std::uint32_t>> const& text,
    std::vector<std::uint32_t>& pattern) {
```

```
    int n = pattern.size() - 1;
    std::vector<int> result;
    std::pair<std::vector<size_t>, std::vector<size_t>> l = LFunction(pattern);
    while (n < text.size()) {
        int i = pattern.size() - 1, j = n;
        while ((i >= 0) && (pattern[i] == text[j].second)) {
            --i;
            --j;
        }
        if (i == -1) {
            result.push_back(n - pattern.size() + 1);
            if (pattern.size() > 2) {
                n += pattern.size() - l.second[1];
            } else {
                ++n;
            }
        } else {
            int suffix = 1, k = 1;
            if (i == pattern.size() - 1) {
                suffix = 1;
            } else {
                if (l.first[i + 1] > 0) {
                    suffix = pattern.size() - l.first[i + 1] - 1;
                } else {
                    suffix = pattern.size() - l.second[i + 1];
                }
            }
            n += std::max(suffix, k);
        }
    }
    for (size_t i = 0; i < result.size(); ++i) {
        std::cout << text[result[i]].first.first << "," << text[result[i]].first.second << "\n";
    }
}

int main() {
    std::cin.tie(nullptr);
    std::ios::sync_with_stdio(false);
    std::vector<std::uint32_t> pattern;
    std::vector<std::pair<std::pair<size_t, size_t>, std::uint32_t>> text;
    char s;
    int row = 1, num = 1, space = 0;
    std::string p, t;
    while (true) {
        s = getchar();
        if (s == ' ') {
            if (!p.empty()) {
                pattern.push_back(static_cast<std::uint32_t>(std::stoi(p)));
            }
            p.clear();
        } else if (s == '\n' || s == EOF) {
            if (!p.empty()) {
                pattern.push_back(static_cast<std::uint32_t>(std::stoi(p)));
            }
            break;
        }
        else {
            p.push_back(s);
        }
    }
    while ((s = getchar()) != EOF) {
        if (s == ' ') {
```

```
            if (!t.empty()) {
                text.push_back({{row, num}, static_cast<std::uint32_t>(std::stoi(t))});
            }
            if (!space) {
                space = 1;
                ++num;
            }
            t.clear();
        } else if (s == '\n') {
            if (!t.empty()) {
                text.push_back({{row, num}, static_cast<std::uint32_t>(std::stoi(t))});
            }
            t.clear();
            space = 0;
            num = 1;
            ++row;
        } else {
            space = 0;
            t.push_back(s);
        }
    }
    BM(text, pattern);
    return 0;
}
```

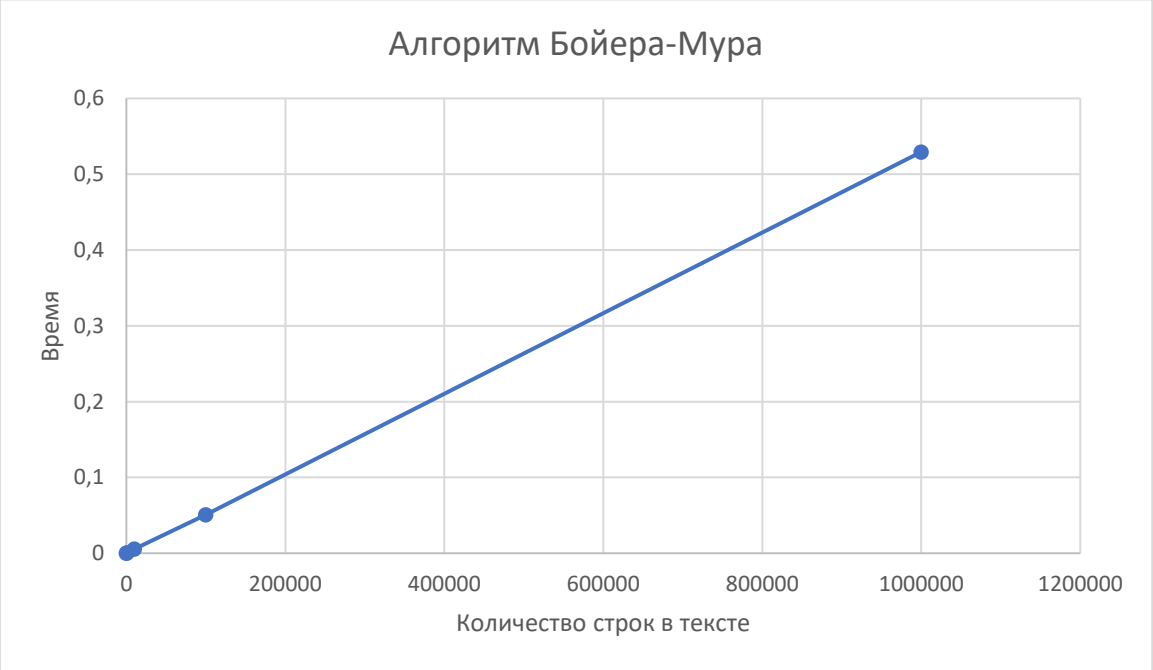
3 Консоль:

```
lina_tucha@LAPTOP-44CRFC1U:~/labs/da/lab4$ cat test.txt
11 45 11 45 90
0011 45 011 0045 11 45 90    11
45 11 45 90
lina_tucha@LAPTOP-44CRFC1U:~/labs/da/lab4$ ./678 < test.txt
1,3
1,8
lina_tucha@LAPTOP-44CRFC1U:~/labs/da/lab4$ cat test1.txt
74418 654 871 1 52
84174418 654 871 1 52 0 817 1 25744
18 654 871 1 52 744 18 654 871 1 52 74418
654 871 1 52 14560 840 80
80 8408944 465484 44 1 2 3 5 4
7 4 4 1 8 6 5 4 8 7 1 1 5 2
lina_tucha@LAPTOP-44CRFC1U:~/labs/da/lab4$ ./678 < test1.txt
2,12
lina_tucha@LAPTOP-44CRFC1U:~/labs/da/lab4$ cat test3.txt
1 2 3 4 5
0 01 0002 34 5 18 5 01
23 450 01 2 3 04 5 068 12
3 4 05 780 112 34504
1 2 3 4 5 1 2 3 4 5 1 2
3 4
5
lina_tucha@LAPTOP-44CRFC1U:~/labs/da/lab4$ ./678 < test3.txt
2,3
4,1
4,6
4,11
```

4 Тест производительности:

Тесты представляли собой строки по 5 чисел от 0 до 9, которые генерируются рандомно, шаблон 3 числа.

	BM(sec.)	Std::string(sec.)
100	0.0001103	0.0001391
1000	0.0006218	0.0004986
10000	0.0064316	0.0046973
100000	0.0569433	0.0362724
1000000	0.529397	0.489345



- Фаза предварительных вычислений требует $O(m^2+\sigma)$ времени и памяти.
- В худшем случае поиск требует $O(m \cdot n)$ сравнений.
- В лучшем случае требует $\Omega(n/m)$ сравнений.

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

Москва, 2020

5 Выводы:

Выполняя данную лабораторную работу, я познакомилась с таким алгоритмом поиска подстроки в строке как алгоритм Бойера-Мура.

Алгоритм Бойера-Мура на хороших данных очень быстр, а вероятность появления плохих данных крайне мала. Поэтому он оптимален в большинстве случаев, когда нет возможности провести предварительную обработку текста, в котором проводится поиск. Таким образом, данный алгоритм является наиболее эффективным в обычных ситуациях, а его быстродействие повышается при увеличении подстроки или алфавита. В наихудшем случае трудоемкость рассматриваемого алгоритма O(m+n). В среднем же алгоритм показывает линейную зависимость от исходной строки.

Москва, 2020