

*Министерство науки и высшего образования Российской Федерации Федеральное государственное
бюджетное образовательное учреждение высшего образования «Московский государственный
технический университет имени Н. Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)*

ОТЧЕТ

По лабораторной работе №7

По курсу: «Анализ алгоритмов»

Тема: «Алгоритмы поиска подстроки в строке»

Студент:

Керимов А. Ш.

Группа:

ИУ7-54Б

Преподаватели:

Волкова Л. Л.,
Строганов Ю. В.

Москва, 2019

Оглавление

Введение	2
1 Аналитическая часть	3
1.1 Описание задачи	3
1.2 Описание алгоритмов	3
1.2.1 Стандартный алгоритм	3
1.2.2 Алгоритм Кнута — Морриса — Пратта	3
1.2.3 Алгоритм Бойера — Мура	4
2 Конструкторская часть	6
2.1 Разработка алгоритмов	6
3 Технологическая часть	8
3.1 Средства реализации	8
3.2 Листинг кода	8
3.3 Тестирование функций	10
4 Исследовательская часть	12
4.1 Примеры работы	12
4.2 Результаты тестирования	12
Заключение	14
Литература	15

Введение

Поиск подстроки в строке — одна из простейших задач поиска информации. Применяется в виде встроенной функции в текстовых редакторах, СУБД, поисковых машинах, языках программирования и т. п. [1]

Задачи работы

Цель лабораторной работы: изучить алгоритмы поиска подстроки в строке.

В рамках выполнения работы необходимо решить следующие задачи:

- изучить стандартный алгоритм, алгоритмы Кнута — Морриса — Пратта и Бойера — Мура;
- реализовать данные алгоритмы;
- привести подробное описание работы каждого алгоритма.

1 Аналитическая часть

1.1 Описание задачи

Пусть даны строки *source* и *pattern*, обозначим их *s* и *p* соответственно. Необходимо проверить, входит ли строка *p* в *s*, если да, то найти индекс первого вхождения.

1.2 Описание алгоритмов

1.2.1 Стандартный алгоритм

Стандартный алгоритм основан на последовательном сравнении всех подстрок строки *s* с *p*, т. е. будет происходить сравнение всех подстрок размера $|p|$, начиная с индексов $i = 1, 2, \dots, |s| - |p| + 1$.

Пусть $s = \text{"abcabccba"}$, $p = \text{"cab"}$. В таблице 1.1 показаны сравнения символов, выполняемые в ходе работы алгоритма.

№	a	b	c	a	b	c	c	b	a
1	c	a	b						
2		c	a	b					
3			c	a	b				

Таблица 1.1: Сравнение символов в стандартном алгоритме

1.2.2 Алгоритм Кнута — Морриса — Пратта

Алгоритм Кнута — Морриса — Пратта является оптимизацией стандартного алгоритма [2]. Необходимо дать определения префикса, суффикса и префикс-функции.

Алгоритм был разработан Д. Кнутом и В. Праттом и, независимо от них, Д. Моррисом [3]. Результаты своей работы они опубликовали совместно в 1977 году [4].

После частичного совпадения начальной части подстроки *pattern* с соответствующими символами строки *source* мы фактически знаем пройденную часть строки и можем «вычислить» некоторые сведения (на основе самой подстроки *pattern*), с помощью которых потом быстро продвинемся по тексту.

Идея КМП-поиска — при каждом несовпадении двух символов текста и образа образ сдвигается на самое длинное совпадение начала с концом префикса (не учитывая тривиальное совпадение самого с собой) [5].

Рассмотрим пример. Создается массив сдвигов, таблица 1.2.

0	1	2	3	4	5
a	b	c	a	b	d
0	0	0	1	2	0

Таблица 1.2: Массив сдвигов

В таблице 1.3 представлена работа алгоритма.

строка	a	b	c	a	b	e	a	b	c	a	b	c	a	b	d
подстрока	a	b	c	a	b	d									
подстрока				a	b	c	a	b	d						
подстрока						a	b	c	a	b	d				
подстрока							a	b	c	a	b	d			
подстрока										a	b	c	a	b	d

Таблица 1.3: Пример работы алгоритма КМП

1.2.3 Алгоритм Бойера — Мура

Алгоритм поиска строки Бойера — Мура считается наиболее быстрым среди алгоритмов общего назначения, предназначенных для поиска подстроки в строке. Был разработан Бойером и Муром в 1977 году [6]. Преимущество этого алгоритма в том, что ценой некоторого количества предварительных вычислений над шаблоном (но не над строкой, в которой ведётся поиск) шаблон сравнивается с исходным текстом не во всех позициях — часть проверок пропускаются как заведомо не дающие результата.

Основная идея алгоритм — начать поиск не с начала, а с конца подстроки. Наткнувшись на несовпадение, мы просто смещаем подстроку до самого правого вхождения данного символа, не учитывая последний.

Рассмотрим пример. Создаётся массив прыжков, таблица 1.4.

a	b	c	a	b	d
2	1	3	2	1	6

Таблица 1.4: Массив прыжков

В таблице 1.5 представлена работа алгоритма.

строка	a	b	c	a	b	e	a	b	c	a	b	c	a	b	d
подстрока	a	b	c	a	b	d									
подстрока							a	b	c	a	b	d			
подстрока										a	b	c	a	b	d

Таблица 1.5: Пример работы алгоритма БМ

Вывод

В данной работе стоит задача реализации алгоритмов поиска подстроки в строке.

2 Конструкторская часть

2.1 Разработка алгоритмов

На рисунках 2.1 и 2.2 представлены схемы алгоритмов КМП и БМ соответственно.

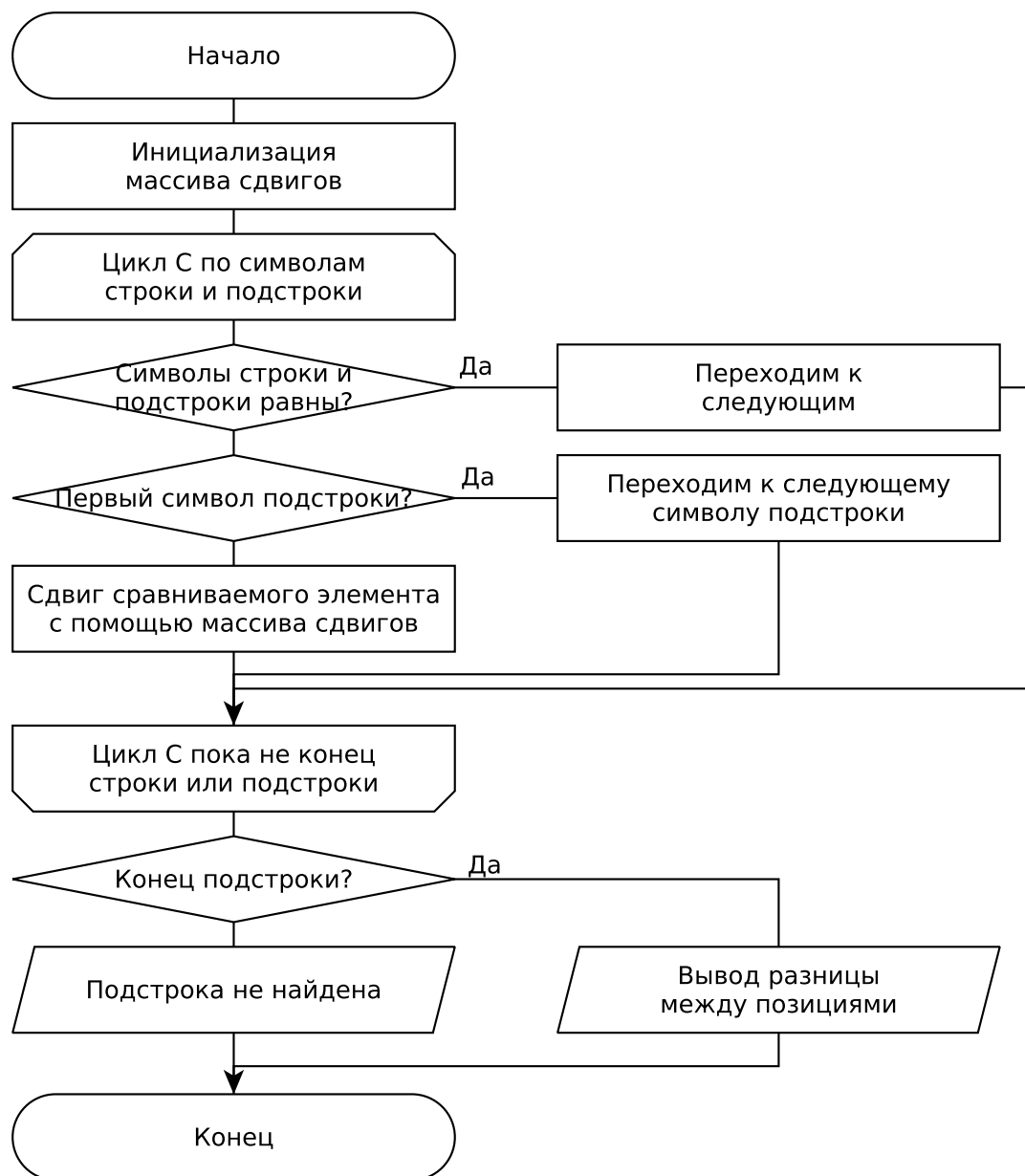


Рис. 2.1: Схема алгоритма Кнута — Морриса — Пратта

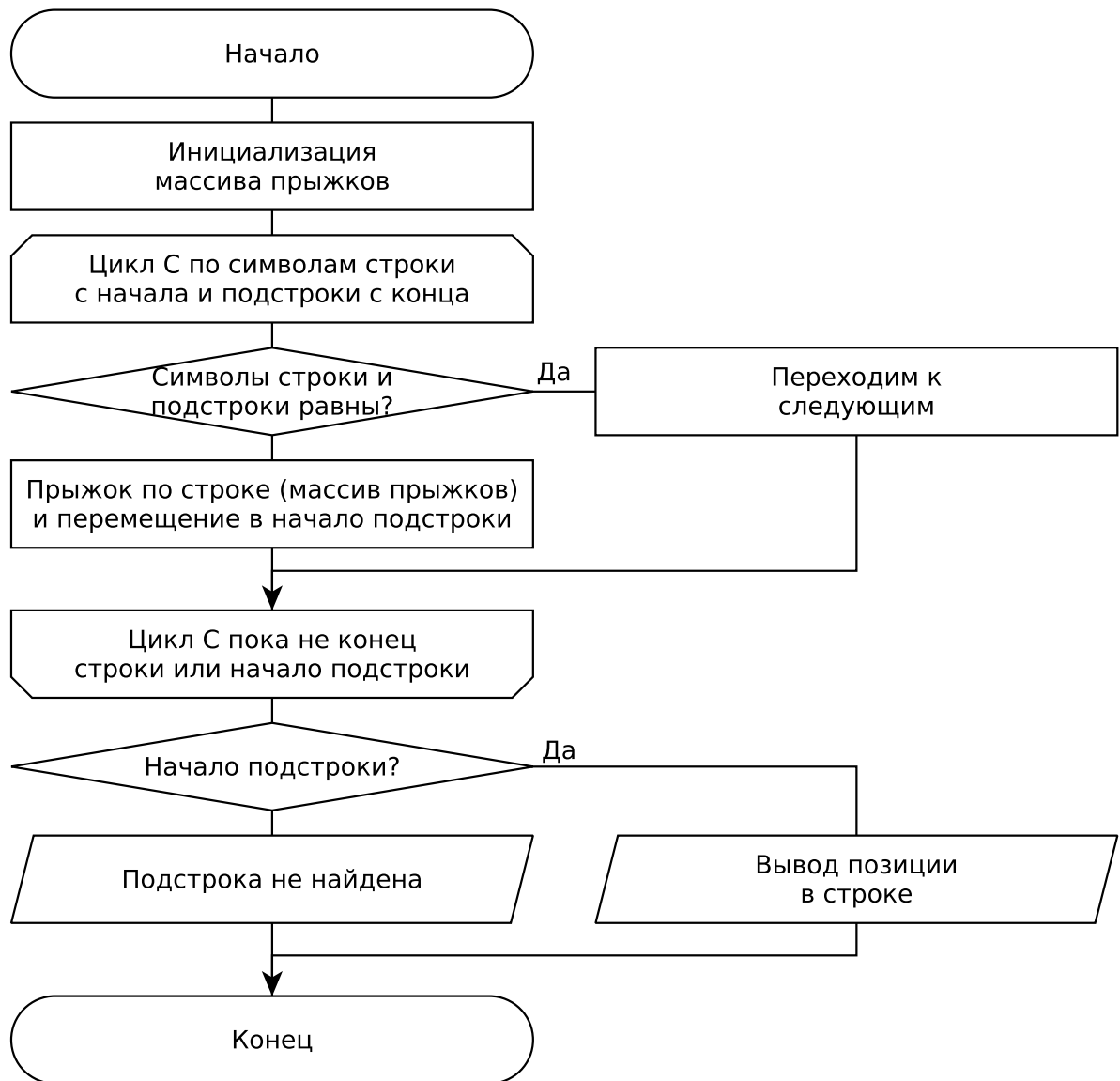


Рис. 2.2: Схема алгоритма Бойера — Мура

Вывод

Были разработаны алгоритмов поиска подстроки в строке.

3 Технологическая часть

В данном разделе приведены средства реализации и листинг кода.

3.1 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран высокопроизводительный язык C++ [7], так как он предоставляет широкие возможности для эффективной реализации алгоритмов.

3.2 Листинг кода

В листингах 3.1, 3.2 и 3.3 приведены листинги алгоритма сортировки пузырьком, вставками и выбором соответственно.

```
int standard(const std::string& str, const std::string& sub) {
    const auto str_size = str.size();
    const auto sub_size = sub.size();

    if (str_size < sub_size) {
        return -1;
    }

    for (size_t i = 0; i <= str_size - sub_size; ++i) {
        for (size_t j = 0, tmp_i = i; j < sub_size; ++j, ++tmp_i) {
            if (sub[j] != str[tmp_i]) {
                break;
            }
            if (j == sub_size - 1) {
                return static_cast<int>(i);
            }
        }
    }

    return -1;
}
```

Листинг 3.1: Стандартный алгоритм

```
std::vector<size_t> get_shift(const std::string& str, size_t size) {
    std::vector<size_t> shift(size);
```

```

    for (size_t i = 1; i < size; ++i) {
        size_t j = shift[i - 1];
        while (j > 0 && str[i] != str[j]) {
            j = shift[j - 1];
        }
        if (str[i] == str[j]) {
            ++j;
        }
        shift[i] = j;
    }

    return shift;
}

int kmp(const std::string& str, const std::string& sub) {
    const auto str_size = str.size();
    const auto sub_size = sub.size();

    if (str_size < sub_size) {
        return -1;
    }

    const auto shift = get_shift(sub, sub_size);

    for (size_t j = 0, i = 0; i < str_size; ++i) {
        while (j > 0 && str[i] != sub[j]) {
            j = shift[j - 1];
        }
        if (str[i] == sub[j]) {
            ++j;
        }
        if (j == sub_size) {
            return static_cast<int>(i - j + 1);
        }
    }

    return -1;
}

```

Листинг 3.2: Алгоритм Кнута — Морриса — Пратта

```

std::map<char, size_t> get_shift(const std::string& sub) {
    constexpr size_t alphabet_size = 256;

    const auto sub_size = sub.size();
    std::map<char, size_t> shift;

    for (size_t symb = 0; symb < alphabet_size; ++symb) {

```

```

        shift[static_cast<char>(symb)] = sub_size;
    }

    for (size_t symb = 0; symb < sub_size - 1; ++symb) {
        shift[static_cast<char>(sub[symb])] = sub_size - symb - 1;
    }

    return shift;
}

int bm(const std::string& str, const std::string& sub) {
    const auto str_size = static_cast<int>(str.size());
    const auto sub_size = static_cast<int>(sub.size());

    if (str_size < sub_size) {
        return -1;
    }

    const auto shift = get_shift(sub);

    int start = static_cast<int>(sub_size) - 1;
    int k = start;
    for (
        auto i = start, j = start;
        j >= 0 && i < str_size;
        i += shift.at(str[i])
    ) {
        for (
            j = start, k = i;
            j >= 0 && str[k] == sub[j];
            --k, --j
        );
    }

    return k >= str_size - sub_size ? -1 : static_cast<int>(k + 1);
}

```

Листинг 3.3: Алгоритм Бойера — Мура

3.3 Тестирование функций

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы поиска подстроки в строке. Тесты пройдены успешно.

Строка1	Строка2	Ожидаемый результат
aaaaa	a	1
aapo	po	2
poaa	po	0
qwerty	a	-1
poropororo	po	0

Таблица 3.1: Тестирование функций

Вывод

Правильный выбор инструментов разработки позволил эффективно реализовать алгоритмы, настроить модульное тестирование и выполнить исследовательский раздел лабораторной работы.

4 Исследовательская часть

4.1 Примеры работы

На рисунках 4.1 и 4.2 изображены примеры работы программы.

```
/home/user/bmstu/AA/lab07/cmake-build-debug/main
Enter string: substring
Enter substring: string
Standard algorithm result: 3
KMP algorithm result: 3
BM algorithm result: 3

Process finished with exit code 0
```

Рис. 4.1: Пример 1 работы программы

```
/home/user/bmstu/AA/lab07/cmake-build-debug/main
Enter string: string
Enter substring: sub
Standard algorithm result: -1
KMP algorithm result: -1
BM algorithm result: -1

Process finished with exit code 0
```

Рис. 4.2: Пример 2 работы программы

4.2 Результаты тестирования

Проверяем нашу программу на тестах из таблицы 3.1. Полученные результаты представлены в таблице 4.1.

Строка1	Строка2	КМП	БМ
aaaaa	a	0	0
aapo	po	2	2
poaa	po	0	0
qwerty	a	-1	-1
poroporopo	po	0	0

Таблица 4.1: Результаты тестирования программы

Вывод

В данном разделе были приведены примеры работы и протестирована программа.

Заключение

По итогам аналитического раздела были описаны стандартный алгоритм, алгоритм Кнута — Морриса — Пратта и алгоритм Бойера — Мура для нахождения подстроки в строке.

Литература

- [1] Д. Гасфилд. Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология. СПб.: Невский Диалект, 2003. с. 654.
- [2] Б. Смит. Методы и алгоритмы вычислений на строках. М.: Вильямс, 2006. с. 456.
- [3] Кормен Т. Лейзерсон Ч. Ривест Р. Штайн К. Алгоритмы: построение и анализ. М.: Вильямс, 2005. с. 1296.
- [4] Donald Knuth; James H. Morris Jr V. P. Fast pattern matching in strings // SIAM Journal on Computing. 1977. Vol. 6, no. 2. P. 323–350.
- [5] М. Окулов С. Алгоритмы обработки строк. М.: Бином, 2013. с. 255.
- [6] Boyer R. S. M. J. S. A fast string searching algorithm // Communications of the ACM. 1977. Vol. 20, no. 10. P. 762–772.
- [7] Working Draft, Standard for Programming Language C++. URL: <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4713.pdf>. 2017.