Государственное образовательное учреждение высшего профессионального образования
“Московский государственный технический университет имени Н.Э.Баумана”

Дисциплина: Анализ алгоритмов

Лабораторная работа 7

Поиск подстроки в строке

Студент группы ИУ7-55,
Аминов Т.С.
Преподаватели: Волкова Л.Л., Строганов Ю.В.

2019 г.

Содержание

Введение

Целью данной лабораторной работы является изучение алгоритмов для поиска подстроки в строке: алгоритма Кнута-Морриса-Пратта и алгоритма Бойера-Мура.

Задачами данной лабораторной работы являются: реализация обоих алгоритмов на одном из языков программирования и рассмотрение работы каждого из них на примере.

1 Аналитическая часть

В данной части будут рассмотрены теоретические основы задачи поиска подстроки в строке и алгоритмов Кнута-Морриса-Пратта и Бойера-Мура, ее решающих.

1.1 Постановка задачи

Имеются две конечные последовательности символов [2] - substring и text. Требуется узнать, включена ли последовательность substring в text и если да, то определить место первого из включений. Здесь и далее вместо термина "конечная последовательность символов"будет употреблятся слово "строка".

1.2 Простой алгоритм

Простой алгоритм нахождения подстроки в строке заключается в том, что в строке text ищется сначала символ, совпадающий с первым символом строки substring, и далее последующие символы строки text проверяются на совпадение с последующими из substring, и если не все символы совпали, снова ведется поиск на совпадение с первой буквой, и так далее, пока не найдется полное совпадение.

Из алгоритма ясно, что основная операция — сравнение символов, и именно число сравнений и следует подсчитывать. В наихудшем случае при каждом проходе совпадают все символы за исключением последнего. Это может произойти по одному разу на каждый символ текста. Если длина подстроки равна 5, а длина текста равна T, то, в наихудшем случае число сравнений будет равно S(T — S + I) [3].

1.3 Конечные автоматы

Конечные автоматы используются для решения задачи о том, принадлежит ли данное слово данному языку [1]. Конечный автомат представляет собой простое устройство, описываемое текущим состоянием и функцией перехода. Функция перехода формирует новое состояние автомата по текущему состоянию и значению очередного входного символа. Некоторые состояния считаются принимающими, и если после окончания ввода автомат оказывается в принимающем состоянии, то входное слово считается принятым. Конечные автоматы широко используются в алгоритмах, связанных со строками, например, при шифровании текста Машиной Тьюринга[4].

Конечный автомат, настроенный на данный образец, можно использовать для сравнения с образцом; если автомат переходит в принимаю- щее состояние, то это означает, что образец найден в тексте. Использование конечных автоматов очень эффективно, поскольку такой автомат обрабатывает каждый символ однократно. Поэтому поиск образца с помощью конечного автомата требует не более T сравнений. Теперь встает задача создания конечного детерминированного автомата, отвечающего данной подстроке. Это непросто; существующие алгоритмы работают долго. Поэтому конечные автоматы и не дают общепринятого хорошего решения задачи сравнения с образцом.

1.4 Алгоритм Кнута-Морриса-Пратта

При построении конечного автомата для поиска подстроки в тексте легко построить переходы из начального состояния в конечное принимающее состояние: эти переходы помечены символами подстроки. Проблема возникает при попытке добавить другие символы, которые не переводят в конечное состояние.

Алгоритм Кнута-Морриса-Пратта основан на принципе конечного автомата, однако он использует более простой метод обработки неподходящих символов. В этом алгоритме состояния помечаются символами, совпадение с которыми должно в данный момент произойти. Из каждого состояния имеется два перехода: один соответствует успешному сравнению, другой — несовпадению. Успешное сравнение переводит нас в следующий узел автомата, а в случае несовпадения мы попадаем в предыдущий узел, отвечающий образцу.

При всяком переходе по успешному сравнению в конечном автомате Кнута-Морриса—Пратта происходит выборка нового символа из текста. Переходы, отвечающие неудачному сравнению, не приводят к выборке нового символа; вместо этого они повторно используют последний выбранный символ. Если мы перешли в конечное состояние, то это означает, что искомая подстрока найдена [5].

1.5 Алгоритм Бойера-Мура

В отличие от алгоритмов, обсуждавшихся выше, алгоритм Бойера-Мура осуществляет сравнение с образцом справа налево, а не слева направо. Исследуя искомый образец, можно осуществлять более эффективные прыжки в тексте при обнаружении несовпадения.

Алгоритм Бойера-Мура обрабатывает образец двумя способами. Во-первых, мы можем вычислить величину возможного сдвига при несовпадении очередного символа. Во-вторых, мы вычислим величину прыжка, выделив в конце образца последовательности символов, уже появлявшиеся раньше.

Мы дали общее описание того, как будут использоваться массивы сдвигов и прыжков. Массив сдвигов содержит величины, на которые может быть сдвинут образец при несовпадении очередного символа. В массиве прыжков содержатся величины, на которые можно сдвинуть образец, чтобы совместить ранее совпавшие символы с вновь совпадающими символами строки. При несовпадении очередного символа образца с очередным символом текста может осуществиться несколько возможностей. Сдвиг в массиве сдвигов может превышать сдвиг в массиве прыжков, а может быть и наоборот. Если элемент массива сдвигов больше, то это означает, что несовпадающий символ оказывается «ближе» к началу, чем повторно появляющиеся завершающие символы строки. Если элемент массива прыжков больше, то повторное появление завершающих символов строки начинается ближе к началу образца, чем несовпадающий символ. В обоих случаях нам следует пользоваться большим из двух сдвигов, поскольку меньший сдвиг неизбежно опять приводит к несовпадению из-за того, что мы знаем о втором значении.

1.6 Вывод

В данном разделе были рассмотрены алгоритмы поиска подстроки в строке, такие как: простой алгоритм, алгоритм Кнута-Морриса-Пратта и алгоритм Бойера-Мура. Последний является наиболее выгодным с точки зрения количества сравнений, так как сдвиг при несовпадении при его работе максимальный среди перечисленных алгоритмов.

2 Конструкторская часть

В данном разделе будет рассмотрены схемы алгоритмов и рассмотрена их работа на примере.

2.1 Схемы алгоритма Кнута-Морриса-Пратта

На рисунке 1 приведена схема нахождения подстроки в строке в алгоритме Кнута-Морриса-Пратта.

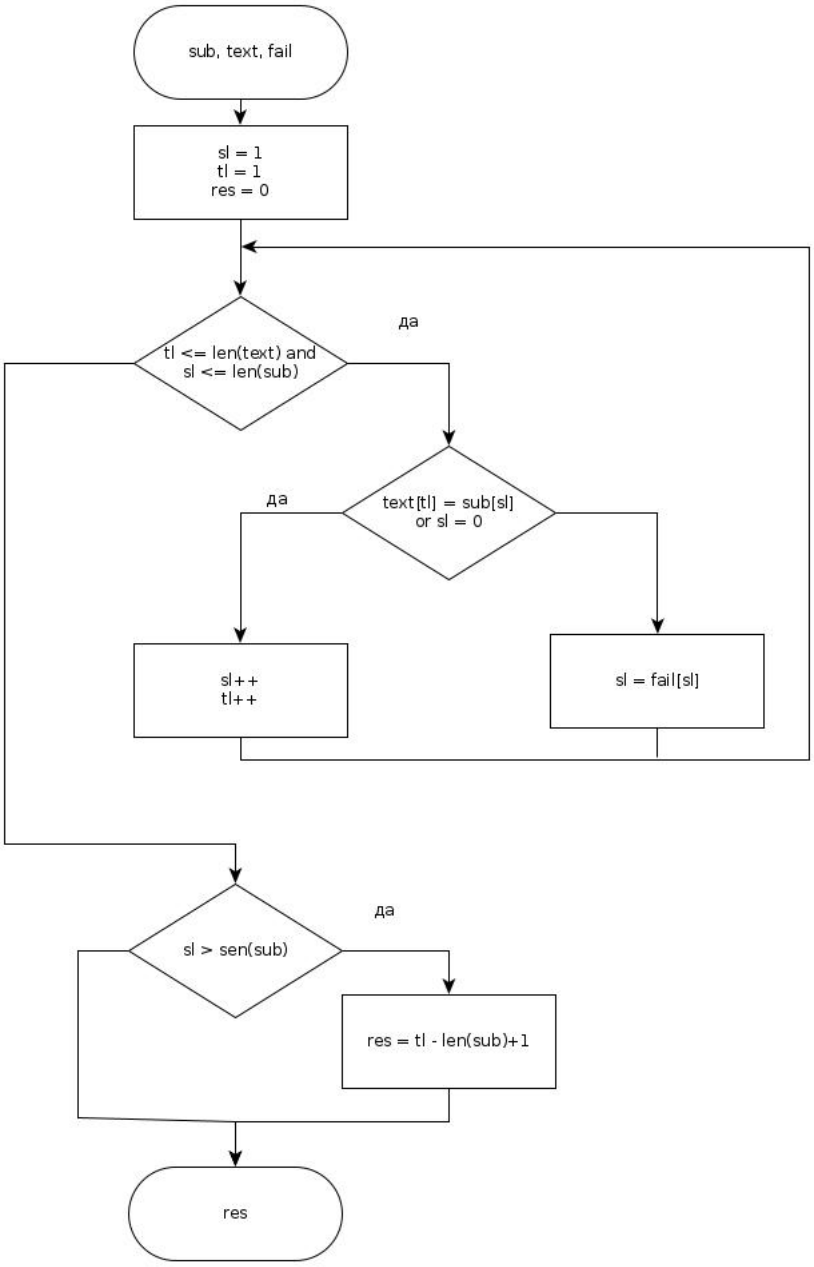

Рис. 1: Схема нахождения подстроки в строке в алгоритме Кнута-Морриса-Пратта

На рисунке 2 приведена схема составления массива fail в алгоритме Кнута-Морриса-Пратта.

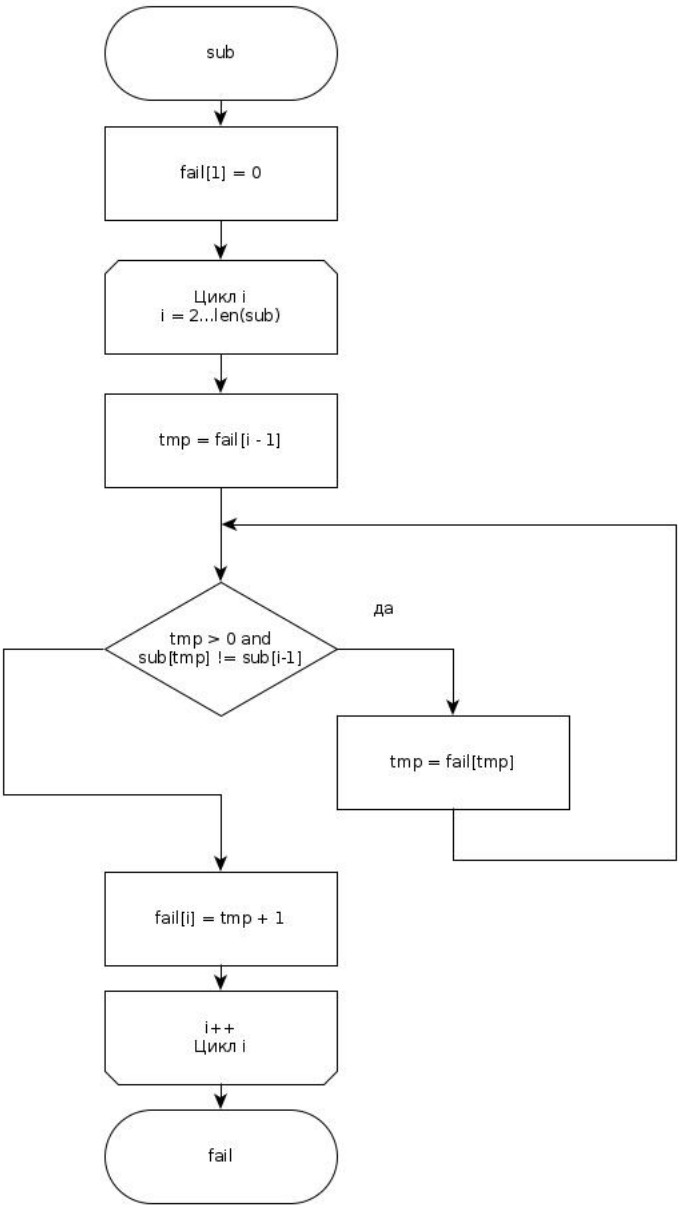

Рис. 2: Схема составления массива fail в алгоритме Кнута-Морриса-Пратта

2.2 Схемы алгоритма Бойера-Мура

На рисунке 4 приведена схема нахождения подстроки в строке в алгоритме Бойера-Мура.

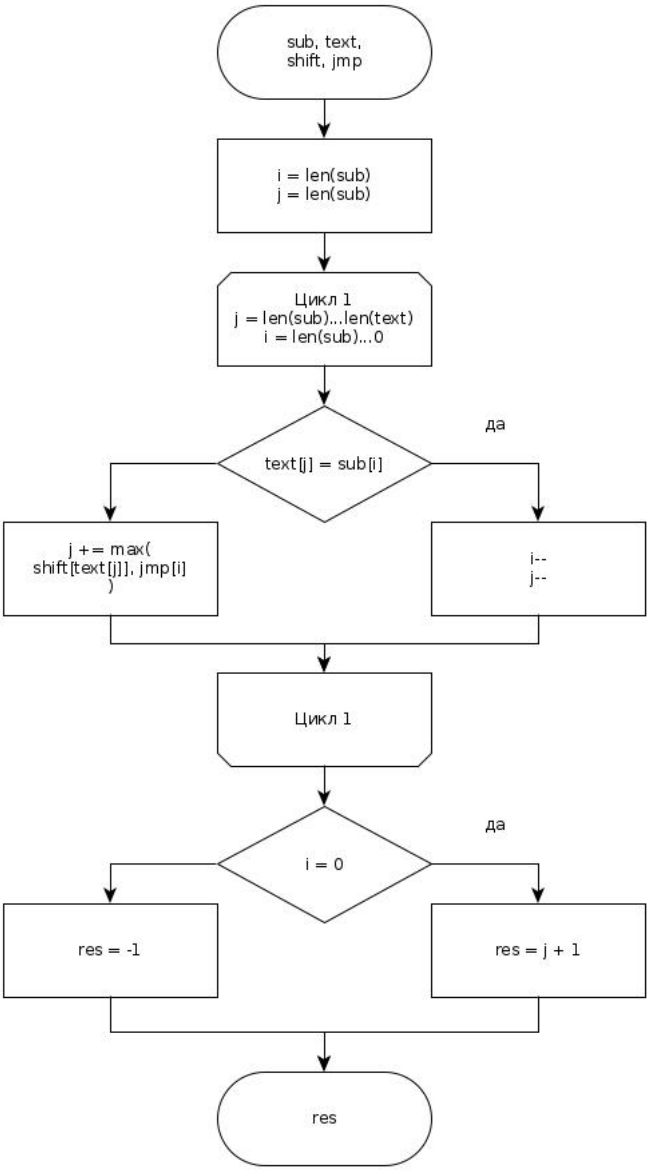

Рис. 3: Схема нахождения подстроки в строке в алгоритме Бойера-Мура

На рисунке 4 приведена схема нахождения массива сдвигов в алгоритме Бойера-Мура.

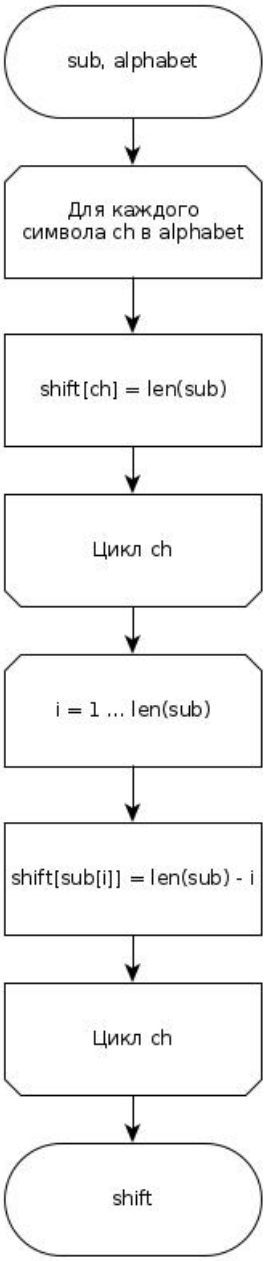

Рис. 4: Схема нахождения массива сдвигов в алгоритме Бойера-Мура

На рисунке 5 приведена схема нахождения массива прыжков в алгоритме Бойера-Мура.

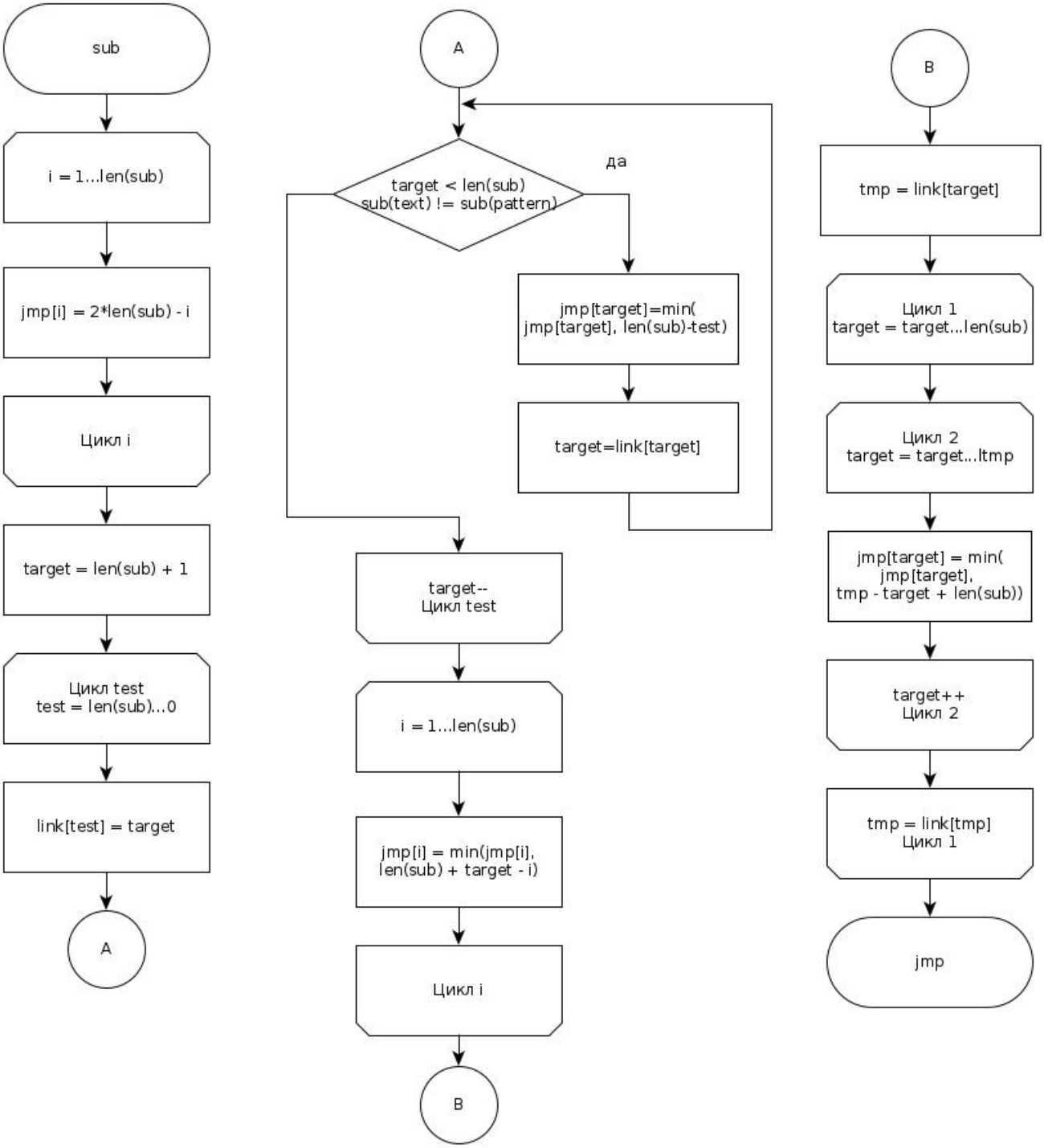

Рис. 5: Схема нахождения массива прыжков в алгоритме Бойера-Мура

2.3 Пример работы алгоритмов

Рассмотрим работу алгоритмов на примере. Подстроку примем равной "test а строку - "zestesctestpppp".

Алгоритм Кнута-Морриса-Пратта

Для строки "test"конечный автомат будет выглядеть так, как показано на рисунке 6.

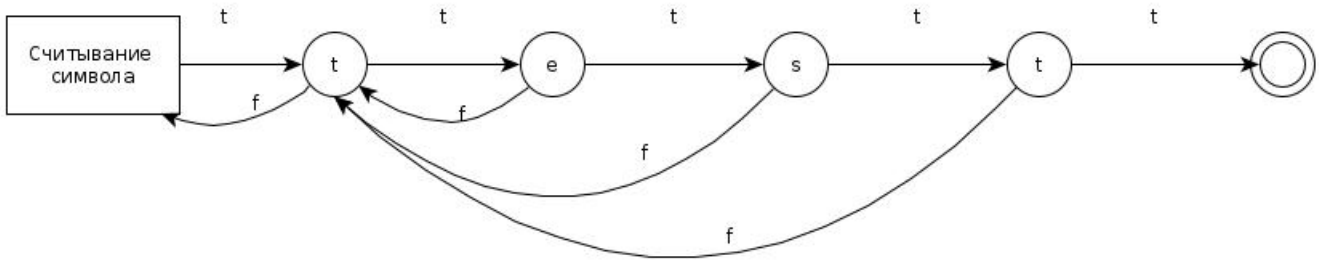

Рис. 6: Конечный автомат в алгоритме Кнута-Морриса-Пратта для строки "test"

Содержание массива fail: 0, 1, 1, 1
Пошаговая работа алгоритма показана в таблице 1.

Таблица 1. Пошаговая работа алгоритма Кнута-Морриса-Пратта.

z	e	s	t	e	s	c	t	e	s	t	p	p	p	p
t	e	s	t											
	t	e	s	t										
		t	e	s	t									
			t	e	s	t								
						t	e	s	t					
							t	e	s	t				

Алгоритм Бойера-Мура

Рассмотрим сначала составление массивов shift и jump.
Нахождение массива сдвигов.

После первого цикла все элементы (в данном случае их 256, так как за алфавит была принята таблица ASCII) были инициализированы значением 4 (длина подстроки). После второго цикла свое значение изменят следующие элементы массива: shift['e'] = 0, shift['s'] = 1, shift['t'] = 2.
Нахождение массива прыжков.

После первого цикла его значения будут такими: 7, 6, 5, 4.
После второго цикла: 7, 6, 5, 1.
Массив link после второго цикла: 0, 3, 3, 4.
Массив jump после третьего цикла: 6, 5, 4, 1.
Массив jump после четвертого цикла: 6, 5, 4, 1.
Теперь рассмотрим работу непосредственно поиска подстроки в строке. Пошаговый поиск показан в таблице 2:

Таблица 2. Пошаговая работа алгоритма Бойера-Мура.

z	e	s	t	e	s	c	t	e	s	t	p	p	p	p
t	e	s	t											
			t	e	s	t								
							t	e	s	t				

2.4 Вывод

В данном разделе были рассмотрены схемы алгоритмов, а также разобрана работа каждого из них на примере. На примере с подстрокой "test а строкой - "zestesctestpppp"алгоритму Кнута-Морриса-Пратта потребовалось 12 сравнений, а алгоритму Бойера-Мура - 9, что подтверждает то, что по количеству сравнений алгоритм Бойера-Мура более эффективный.

3 Технологическая часть

В данной части будут рассмотрены средства для реализации алгоритмов.

3.1 Требования к программному обеспечению

Входные данные - текст, подстрока.
Выходные данные - индекс первого вхождения.
На рисунке 7 дана функциональная схема решения задачи поиска подстроки в строке в нотации IDEF0.

Рис. 7: Функциональная схема решения задачи поиска подстроки в строке

3.2 Средства реализации

В качестве языка программирования был выбран c++ [6] из-за большого числа библиотек, в качестве фреймворка для разработки был выбран QT [8] по тем же причинам, среда разработки - QTCreator [7].

3.3 Листинг кода

На листинге 1 представлена функция нахождения подстроки в строке по алгоритму Кнута-Морриса-Пратта.

Листинг 1: Функция нахождения подстроки в строке по алгоритму Кнута-Морриса-Пратта

```
int k_m_p(std::string text, std::string sub, std::vector<int> fail)
{
    int subloc = 1, textloc = 1;
    int res = -1;
    while ((subloc <= (int)sub.size()) && (textloc <= (int)text.size()))
    {
        if((subloc == 0) || (text[textloc - 1] == sub[subloc - 1]))
        {
            textloc++;
            subloc++;
        }
        else
            subloc = fail[subloc - 1];
    }
    if(subloc > sub.size())
        res = textloc - sub.size() - 1;
    return res;
}
```

На листинге 2 представлена функция нахождения массива fail в алгоритме Кнута-Морриса-Пратта.

Листинг 2: Функция нахождения массива fail в алгоритме Кнута-Морриса-Пратта

```
std::vector<int> create_state_machine(std::string s)
{
    std::vector<int> fail;
    for(unsigned int i = 0; i < s.size() + 1;i++)
        fail.push_back(0);
    for(unsigned int i = 1; i < s.size();i++)
    {
        int tmp = fail[i - 1];
        while((tmp > 0) && (s[tmp] != s[i - 1]))
            tmp = fail[tmp];
        fail[i] = tmp + 1;
    }
    return fail;
}
```

На листинге 3 представлена функция нахождения подстроки в строке по алгоритму Бойера-Мура.

Листинг 3: Функция нахождения подстроки в строке по алгоритму Бойера-Мура

```
int b_m(std::string text, std::string sub , std::vector<int> shift, std::vector<int> jump
    )
{
    int res = -1;
    int textLoc = sub.size() - 1;
    int subLoc = sub.size() - 1;
    while ((textLoc <= text.size()) && (subLoc >= 0))
    {
        if (text[ textLoc ] == sub[ subLoc ])
        {
            textLoc--;
            subLoc--;
        }
        else
        {
            textLoc = textLoc + std::max(shift[(int)text[textLoc]], jump[subLoc]);
             subLoc = sub.size() - 1;
        }

    }
    if (subLoc == -1)
        res = textLoc;
    return res;
}
```

На листинге 4 представлена функция нахождения массива shift в алгоритме Бойера-Мура.

Листинг 4: Функция нахождения массива shift в алгоритме Бойера-Мура

```
std::vector<int> create_arr_shift(std::string sub)
{
    std::vector<int> shift;
    for(int i = 0; i < ALPH_SIZE; i++)
        shift.push_back(sub.size() - 1);
    for(int i = 0; i <  sub.size(); i++)
    {
        shift[(int) sub[i]] = i - 1;
    }
    return shift;
}
```

На листинге 5 представлена функция нахождения массива jump в алгоритме Бойера-Мура.

Листинг 5: Функция нахождения массива jump в алгоритме Бойера-Мура

```
std::vector<int> create_arr_jump(std::string sub)
{
    std::vector<int> jump, link;
    for(int i = 0; i < sub.size(); i++)
    {
        jump.push_back(2*sub.size() - i - 1);
        link.push_back(0);
    }
    link.push_back(0);
    int test = sub.size() - 1;
    int target = sub.size();
    while(test > 0)
    {
        link[test] = target;
        while ((target < sub.size()) && (sub[test] != sub[target]))
        {
            jump[target] = std::min(jump[target], (int)sub.size() - test - 1);
            target = link[target];
        }
        test--;
        target--;
    }
    for(int i = 0; i < target; i++)
        jump[i] = std::min(jump[i], (int)sub.size() + target - i - 1);
    int tmp = link[target];
    while(target < sub.size())
    {
        while(target < tmp)
        {
            jump[target] = std::min(jump[target], tmp - target + (int)sub.size() + 1);
            target++;
        }
        tmp = link[tmp];
    }
    return jump;
}
```

На листингах 6 и 7 представлены "оберточные"функции обоих алгоритмов, нужные для того, чтобы не разбивать алгоритм на 2 и 3 вызова различных функций соответственно.

Листинг 6: "Оберточная"функция алгоритма Кнута-Морриса-Пратта

```
int k_m_p(std::string text, std::string sub)
{
    return k_m_p(text, sub, create_state_machine(sub));
}
```

Листинг 7: "Оберточная"функция алгоритма Бойера-Мура

```
int b_m(std::string text, std::string sub)
{
    return b_m(text, sub, create_arr_shift(sub), create_arr_jump(sub));
}
```

3.4 Тестирование на правильность работы

Было проведено тестирование обоих алгоритмов на правильность работы для 9 различных пар строк text и substring. Результаты тестирования приведены в таблице 3. Поля таблицы 3: substring - подстрока, text - строк, К-М-П - результат работы алгоритма Кнута-Морриса-Пратта, Б-М - результат работы алгоритма Бойера-Мура.

Таблица 3. Результаты тестирования алгоритмов.

substring	text	К-М-П	Б-М	ожидаемый результат
aab	aaaaaaaab	6	6	6
test	testestesttes	0	0	0
37	37373737	0	0	0
cc	aba	-1	-1	-1
they	therezтheyzare	6	6	6
ser	sergey	0	0	0
gey	sergey	3	3	3
asdb	asdasdasdasd	-1	-1	-1

Во всех рассматриваемых в таблице 3 случаях результаты работы алгоритмов совпали с ожидаемыми, что подтвердило их правильность.

3.5 Вывод

В данном разделе были представлены реализации алгоритмов Кнута-Морриса-Пратта и Бойера-Мура, и подтверждена их правильность (реализаций).

Заключение

В ходе данной лабораторной работы были изучены алгоритмы Кнута-Морриса-Пратта и Бойера-Мура для поиска подстроки в строке, рассмотрена работа каждого из них на примере, и представлена реализация обоих алгоритмов на языке c++. Правильность реализации была проверена с помощью тестирования на заготовленных данных.

Список литературы

[1] Белоусов А.И., Ткачев С.Б(2006). Дискретная математика, 4-е издание.

[2] Морозова В.Д, Введение в анализ(1996).

[3] Солдатова Г.П., Татаринов А.А., Болдырихин Н.В., Основные алгоритмы поиска подстроки в строке.

[4] Чернушко М. М. Применение машины Тьюринга для реализации алгоритмов шифрования [Текст] // Технические науки: теория и практика: материалы II Междунар. науч. конф. (г. Чита, январь 2014 г.). — Чита: Издательство Молодой ученый, 2014. — С. 19-22. — URL https://moluch.ru/conf/tech/archive/88/4317/ (дата обращения: 11.12.2019).

[5] Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход.-М.:Техносфера, 2009.

[6] Документация по языку C++ [Электронный ресурс] - режим доступа https://devdocs.io/cpp/

[7] Документация по среде QTCreator [Электронный ресурс] - режим доступа http://doc.crossplatform.ru/qtcreator/1.2.1/

[8] Документация по фреймворку QT [Электронный ресурс] - https://doc.qt.io/