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

Дисциплина: Анализ алгоритмов

Лабораторная работа № 7

Поиск подстроки в строке

Гибадулин О.Н.
Студент группы ИУ7-52

2019 г.

Содержание

Введение

Поиск подстроки в длинном куске текста — важный элемент текстовых редакторов. В программах, предназначенных для редактирования текста, часто необходимо найти все фрагменты текста, совпадающие с заданным образцом. Обычно. текст - это редактируемый документ, а образец - искомое слово, введённое пользователем. Эффективные алгоритмы решения этой задачи могут сокращать время реакции текстовых редакторов на действия пользователя.

В данной работе требуется рассмотреть алгоритм Кнута-Морриса-Пратта и алгоритм Бойера-Мура, предназначенные для поиска подстроки в строке.

Цель работы: изучение алгоритмов поиска подстроки в строке.

Задачи работы:

1. разработка и реализация алгоритмов;
2. исследование работы алгоритма при различных параметрах;
3. описание и обоснование полученных результатов.

1 Аналитический раздел

В данном разделе будет описаны алгоритм Кнута-Морриса-Пратта и алгоритм Бойера-Мура, предназначенные для поиска подстроки в строке.

1.1 Постановка задачи

Даны образец (строка) S и строка T. Требуется определить индекс, начиная с которого образец S содержится в строке T. Если S не содержится в T — вернуть индекс, который не может быть интерпретирован как позиция в строке (например, отрицательное число). При необходимости отслеживать каждое вхождение образца в текст.

1.2 Алгоритм Кнута-Морриса-Пратта

Алгоритм Кнута-Морриса-Пратта находит все вхождения образца в строку. В данном алгоритме ключевым элементом является префикс функция, еще говорят, что это функция построения конечного автомата. В данном алгоритме, перед тем как начнет выполняться основная часть, должна выполниться префикс функция или функция для нахождения перехода по несовпадению. При поиске подсктроки с помощью данной функции происходит перемещение по строке[1].

Алгоритм Кнута-Морриса-Пратта основан на принципе конечного автомата. 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. Успеш- ное 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.

1.3 Алгоритм Бойера-Мура

Алгоритм поиска строки Бойера — Мура считается наиболее быстрым среди алгоритмов общего назначения, предназначенных для поиска подстроки в строке. Преимущество этого алгоритма в том, что ценой некоторого количества предварительных вычислений над шаблоном сравнения с исходным текстом происходит не во всех позициях — часть проверок пропускаются как заведомо не дающие результата[1].

Алгоритм сравнивает символы шаблона справа налево, начиная с самого правого, один за другим с символами исходной строки. Если символы совпадают, производится сравнение предпоследнего символа шаблона и так до конца шаблона. 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.

1.4 Вывод

В данном разделе были описаны алгоритм Кнута-Морриса-Пратта и алгоритм Бойера-Мура, предназначенные для поиска подстроки в строке.

2 Конструкторский раздел

В данном разделе в соответствии с описанием алгоритмов, приведенными в аналитической части работы, будут рассмотрена схемы алгоритма Кнута-Морриса-Пратта и алгоритма Бойера-Мура.

2.1 Разработка Алгоритма

В данном пункте представлены схемы алгоритма Кнута-Морриса-Пратта и алгоритма Бойера-Мура.

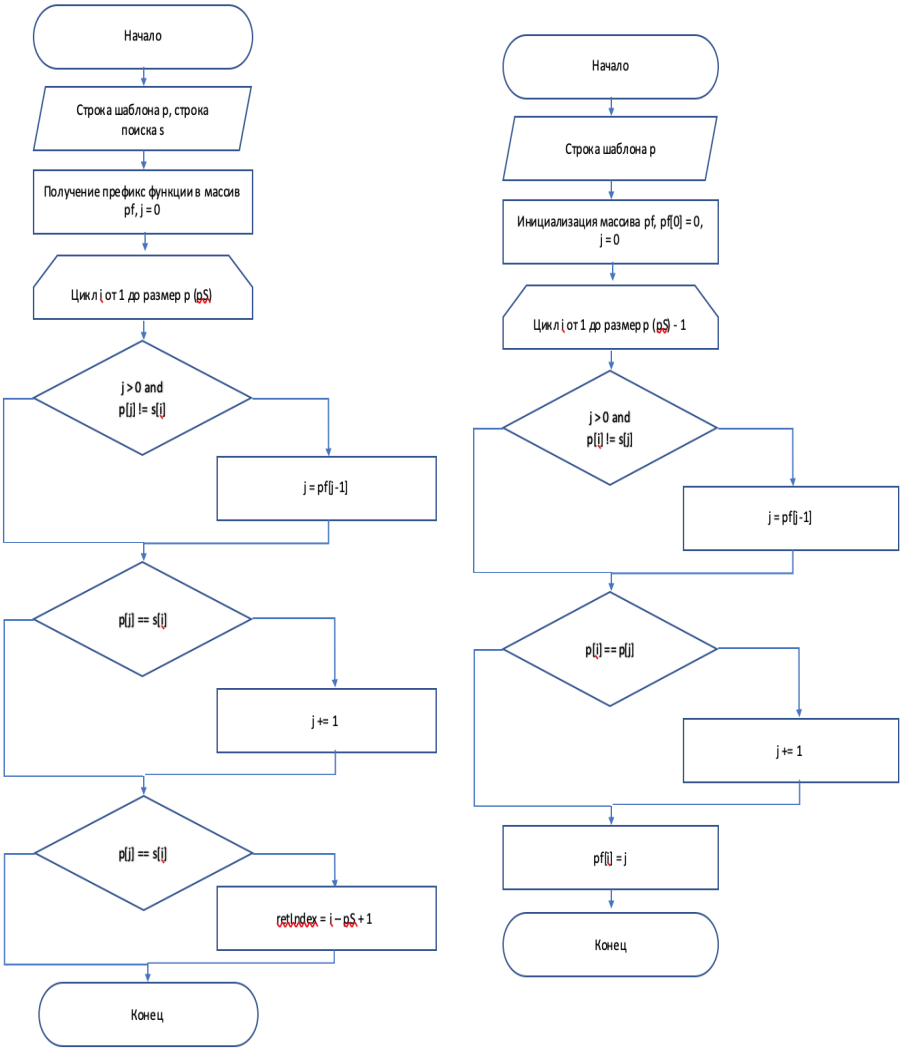

Рисунок 2.2.1. – схема алгоритма Кнута-Морриса-Пратта (слева), схема получения префикс функции (справа)

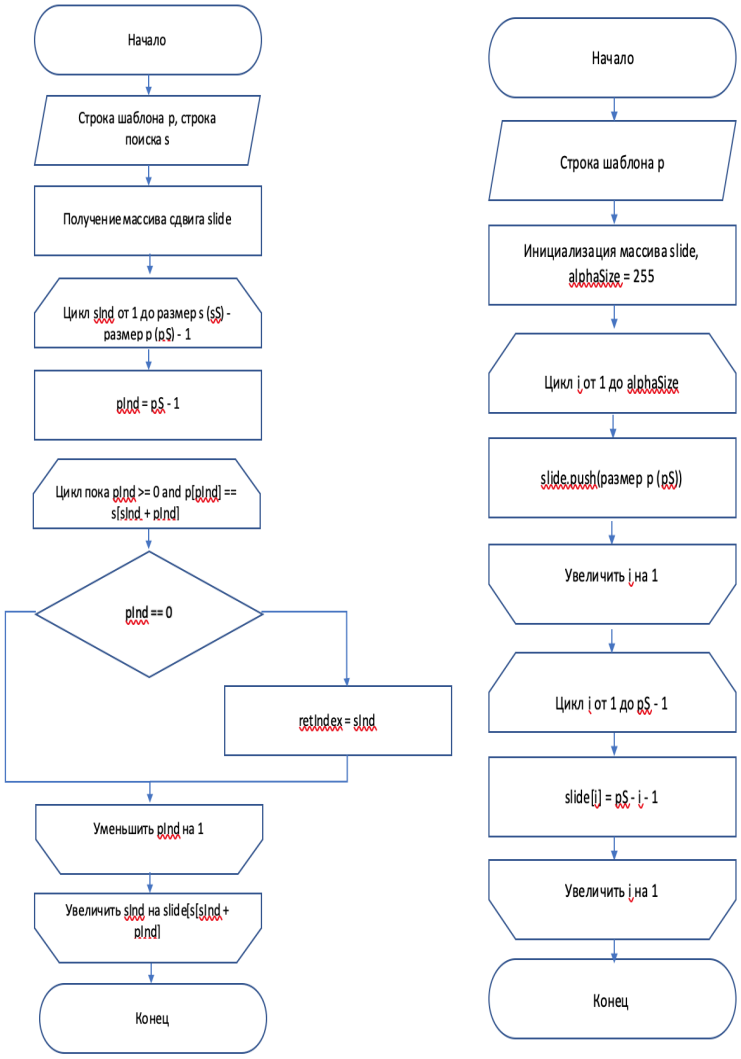

Рисунок 2.2.2. – схема алгоритма Бойера-Мура (слева), схема получения массива сдвига (справа)

2.2 Вывод

В данном разделе были рассмотрены схемы алгоритма Кнута-Морриса-Пратта и алгоритма Бойера-Мура.

=

3 Технологический раздел

В данном разделе будут рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач, а также представлен листинг кода программы.

3.1 Требования к программному обеспечению

Программное обеспечение должно реализовывать алгоритм Кнута-Морриса-Пратта и алгоритм Бойера-Мура, предназначенные для поиска подстроки в строке. Пользователь должен иметь возможность ввода подстроки и строки, которые которые будут учавствоавть в работе алгоритма.

3.2 Средства реализации

Для реализации поставленной задачи был использован язык программирования C++[2]. Проет был выполнен в среде XCode[3].

3.3 Листинг кода

На основе схем алгоритмов, представленных в конструкторском разделе, в соответствии с указанными требованиями к реализации с использованием средств языка C++ было разработано программное обеспечение, содержащее реализацию выбранного алгоритма. В данном пункте приведён листиннг этой реализации.

Листинг 1. Код реализации алгоритма Кнута-Морриса-Пратта

```
void getPrefixFunction(vector <int> &pf, const
    string &pattern) {
if (pattern.empty()) {
return;
}

pf[0] = 0;

for (int i = 1, j = 0; i < pattern.size(); ++i) {
if (j > 0 && pattern[i] != pattern[j]) {
j = pf[j - 1];
}

if (pattern[i] == pattern[j]) {
++j;
}
```

```
pf[i] = j;
}
}

vector <int> KMP(const string &s, const string &
    pattern) {
if (s.empty() || pattern.empty()) {
return vector <int>();
}

vector <int> mathesIndexes;
vector <int> pf(pattern.size());

getPrefixFunction(pf, pattern);

for (int i = 0, j = 0; i < s.size(); ++i) {
if (j > 0 && pattern[j] != s[i]) {
j = pf[j - 1];
}

if (pattern[j] == s[i]) {
++j;
}

if (j == pattern.size()) {
mathesIndexes.push_back(i - pattern.size() + 1);
}
}

return mathesIndexes;
}
```

Листинг 2. Код реализации алгоритма Бойера-Мура

```
void getSlide(vector <int> &slide, const string
    &pattern) {
if (pattern.empty()) {
return;
}

const int alphabetSize = 255;

for (int i = 0; i < alphabetSize; ++i) {
slide.push_back(pattern.size());
}

for (int i = 0; i < pattern.size() - 1; ++i) {
slide[pattern[i]] = pattern.size() - i - 1;
}
}
```

```
vector <int> BM(const string &s, const string &
    pattern) {
if (s.empty() || pattern.empty()) {
return vector <int>();
}

vector <int> mathesIndexes;
vector <int> slide;

getSlide(slide, pattern);

int sLen = s.size();
int pLen = pattern.size();
int sInd = 0;

while (sInd < sLen - (pLen - 1)) {
int pInd = pLen - 1;

for (;pInd >= 0 && pattern[pInd] == s[sInd + pInd
    ]; --pInd) {
if (pInd == 0) {
mathesIndexes.push_back(sInd);
}
}

sInd += slide[s[sInd + pInd]];
}

return mathesIndexes;
}
```

3.4 Вывод

В данном разделе были рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки, а также был представлен листинг реализации алгоритмов поиска подстроки в строке.

4 Экспериментальный раздел

В данном разделе будет проведен анализ корректности и сравнительный анализ работы реализованных алгоритмов при различных параметрах.

4.1 Сравнительное исследование

Для анализа корректности работы алгоритма Кнута-Морриса-Пратта (табл. 4.1.1) и алгоритма Бойера-Мура (табл. 4.1.2) программа запускалась с различными входными параметрами.

Таблица 4.1.1. Примеры работы алгоритма Кнута-Морриса-Пратта

Строка	Подстрока	Вхождения	Количество сравнений
-	-	-	0
-	a	-	0
a	-	-	0
a	a	0	1
ab	a	0	2
a	ab	-	1
ererer	er	0 2 4	6
asfaasfasfasfsaf	asf	0 4 7 10	16
erererererasfer	asf	8	13

Таблица 4.1.2. Примеры работы алгоритма Бойера-Мура

Строка	Подстрока	Вхождения	Количество сравнений
-	-	-	0
-	a	-	0
a	-	-	0
a	a	0	1
ab	a	0	2
a	ab	-	0
ererer	er	0 2 4	6
asfaasfasfasfsaf	asf	0 4 7 10	15
erererererasfer	asf	8	6

Из данной таблицы можно сделать вывод о том, что алгоритмы работают корректно, а также алгоритм Бойера-Мура работает с меньшим количеством сравнений (для строки "erererererasfer"и подстроки "asf"на 50%) по сравнению с алгоритмом Кнута-Морриса-Пратта.

4.2 Вывод

В данном разделе был проведен анализ корректности и сравнительный анализ работы реализованных алгоритмов при различных параметрах, из которых можно сделать вывод, что алгоритмы работают корректно, а также алгоритм Бойера-Мура работает с меньшим количеством сравнений (для

строки "erererererasfer"и подстроки "asf"на 50%) по сравнению с алгоритмом Кнута-Морриса-Пратта.

Заключение

В ходе выполнения данной лабораторной работы были изучены и реализованы алгоритм Кнута-Морриса-Пратта и алгоритм Бойера-Мура, предназначенные для поиска подстроки в строке. Было проведено исследование работы алгоритма при различных параметрах, показавшее корректность работы алгоритмов.

В аналитическом разделе было дано описание алгоритма Кнута-Морриса-Пратта и алгоритма Бойера-Мура. В конструкторском разделе были разработаны алгоритмы, описанные в аналитическом разделе. В технологическом разделе были рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач, а также представлен листинг кода программы. В экспериментальном разеде было проведено сравнительно исследование, из которого следует, что алгоритм Бойера-Мура работает с меньшим количеством сравнений (для строки "erererererasfer"и подстроки "asf"на 50%) по сравнению с алгоритмом Кнута-Морриса-Пратта и исследование корректности работы алгоритмов.

Список литературы

[1] Дж. Макконел. Анализ алгоритмов, активный обучающий подход. Глава 5 - Алгоритмы сравнения с образцом. ISBN 5-94836-005-9

[2] ISO/IEC JTC1 SC22 WG21 N 3690 «Programming Languages — C++» [Электронный ресурс]. – Режим доступа: https://devdocs.io/cpp/, свободный. (Дата обращения: 29.09.2019 г.)

[3] Apple «Apple Developer Documentation» [Электронный ресурс]. – Режим доступа: https://developer.apple.com/documentation/, свободный. (Дата обращения: 29.09.2019 г.)