Государственное образовательное учреждение высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Лабораторная работа №7

по курсу «Анализ алгоритмов»

Поиск подстроки в строке

Выполнил: Тимонин А.С., гр. ИУ7-52Б

Преподаватели: Волкова Л.Л., Строганов Ю.В.

2019 г.

Оглавление

Введение

Часто перед разработчиками стоит задача нахождения количества вхождений подстроки в строку. Это могут быть разработчики поисковых систем, а также разработчики программного обеспечения такого как Word, Pages. Алгоритм поиска подстроки в строке используется во всех программах, где нужно найти какое-нибудь слово в тексте. Если текст в котором мы ищем нужную нам подстроку невелик, то классический перебор может подойти. А вот когда нам нужно найти вхождение слова в тексте размером в миллион символов, то классический перебор станет очень трудозатратным, поэтому нам нужен более эффективный способ рещения данной задачи. И этим эффективным рещением являются алгоритмы Кнута-Морриса-Пратта и Байера-Мура.

Цель работы: изучение более эффективного способа нахождения поиска подстроки в строке, чем классическое сравнение каждого символа строки с подстрокой.

Задачи лабораторной работы:

1. Изучение и описание алгоритмов.
2. Разработка и реализация алгоритмов.
3. Тестирование полученного программного обеспечения.

1. Аналитический раздел

В данном разделе будуд описаны алгоритмы Кнута-Морриса-Пратта и Байера-Мура.

1.1 Алгоритм Кнута-Морриса-Пратта

Алгорритм 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, данные несовпадения записываются в массив fail. После нахождения массива fail выполняется основная часть алгоритма, в которой, как в конченом автомате происходит переход из одного состояния в другое: если сравнение успешно, то переход к следующему состоянию автомата, иначе выбранный символ используется повторно.

1.2 Алгоритм Байера-Мура

В алгоритме Байера-Мура сравнение с образцом осуществляется справа налево, в отличие от прямого перебора, где сравнение производится слева направо. В алгоритме, в случае несовпадения последних символов, мы сдвигаемся на столько символов, чтобы билжайщий символ с конца в подстроке совпал с символом в тексте. В данном алгоритме таким образом производится намного меньше бесполезных сравнений, нежели чем в исхожном простом алгоритме.

1.3 Вывод

В данном разделе были описаны алгоритмы Кнута-Морриса-Пратта и Байера-Мура.

2. Конструкторский раздел

В данном разделе будет приведена блок-схема алгоритма Кнута-Морриса-Пратта и Байера-Мура, приведена сложность алгоритмов, основанная на учебнике Дж. Макконнелла[1].

2.1 Разработка алгоритмов

В данном разделе приведены блок-схемы на рисунках. 2.1-2.4 алгоритмов поиска подстроки в строке.

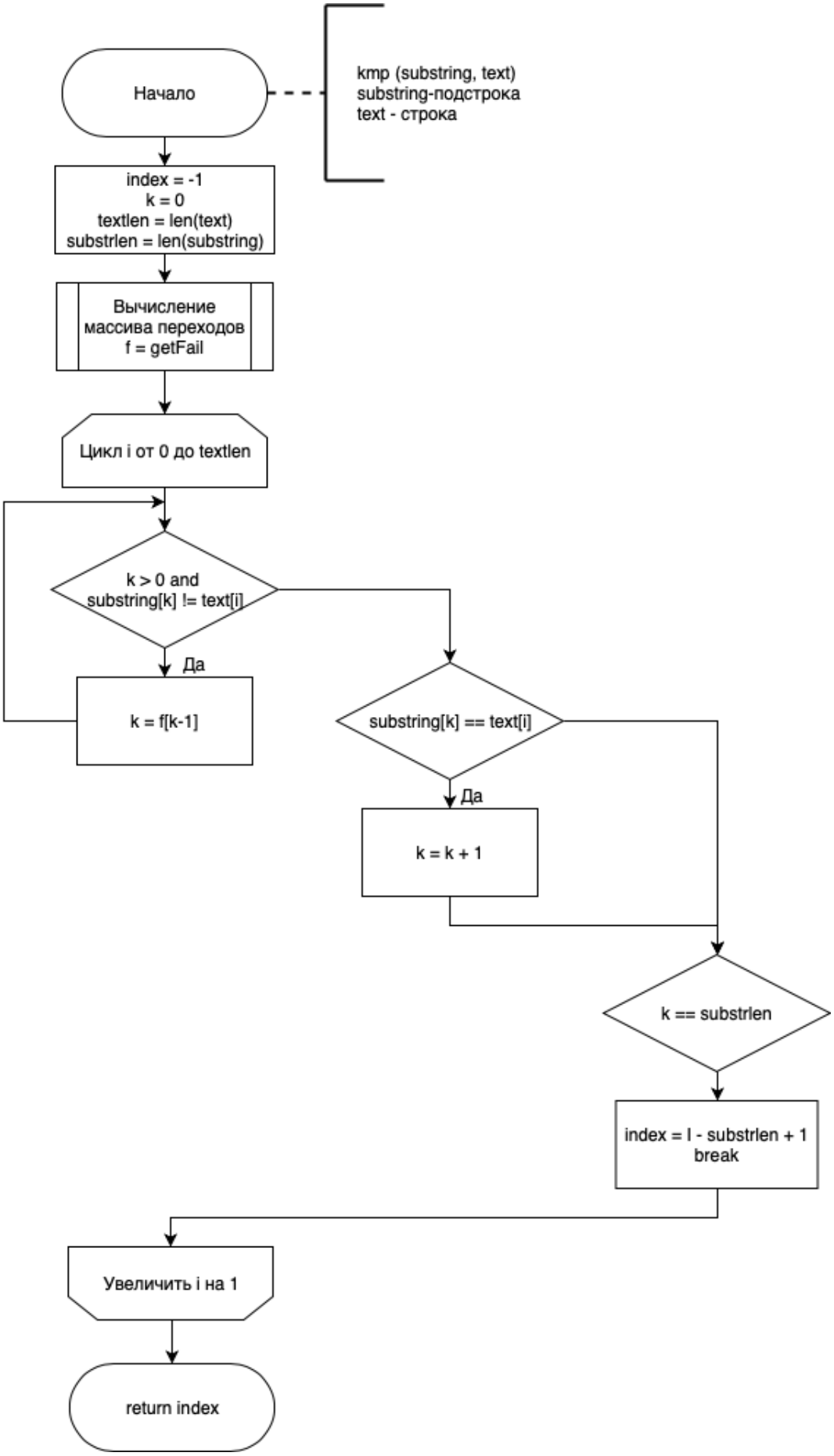

Рис. 2.1: Схема алгоритма Кнута-Морриса-Пратта

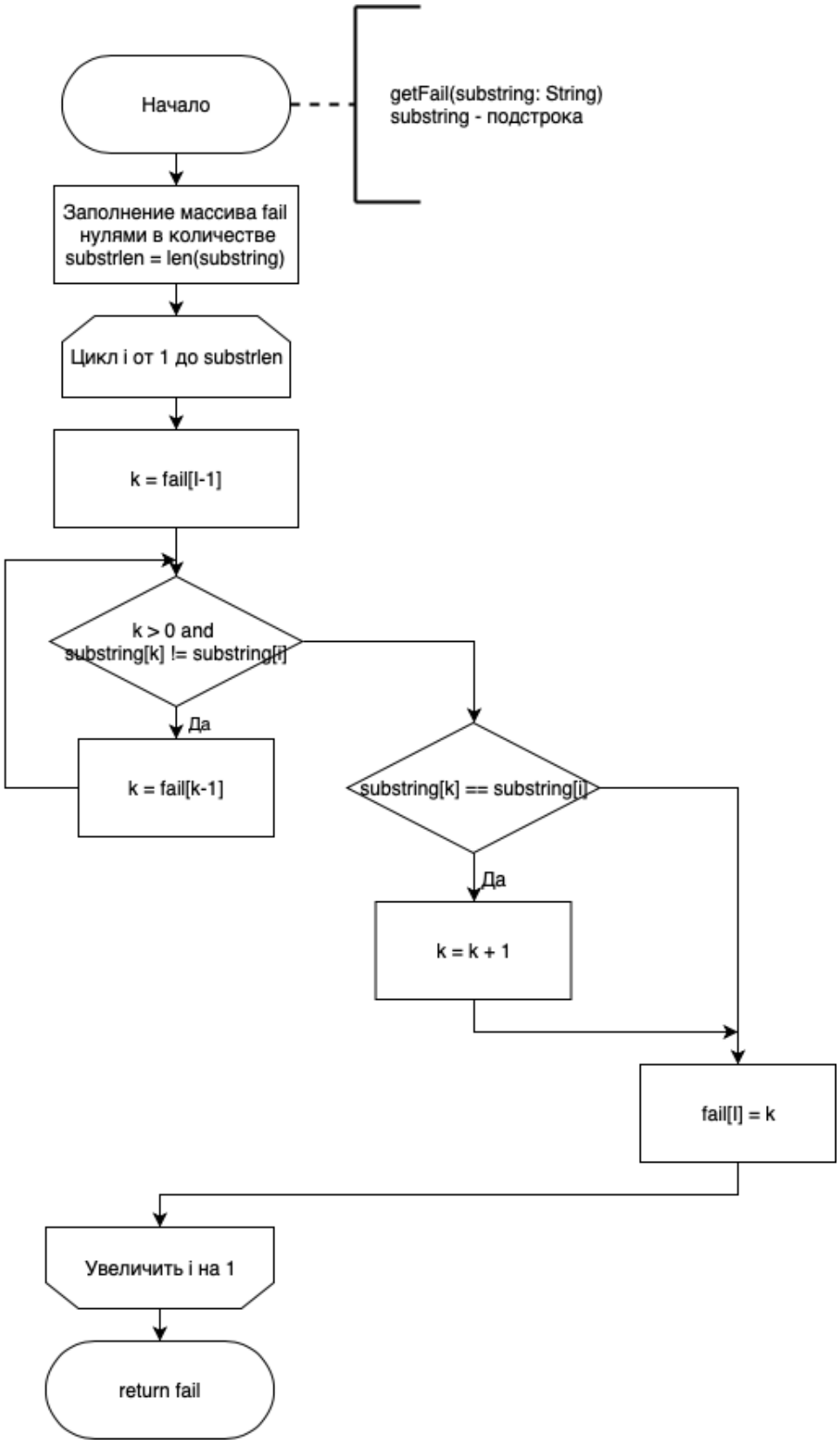

Рис. 2.2: Схема нахождения массива f в алгоритме Кнута-Морриса-Пратта

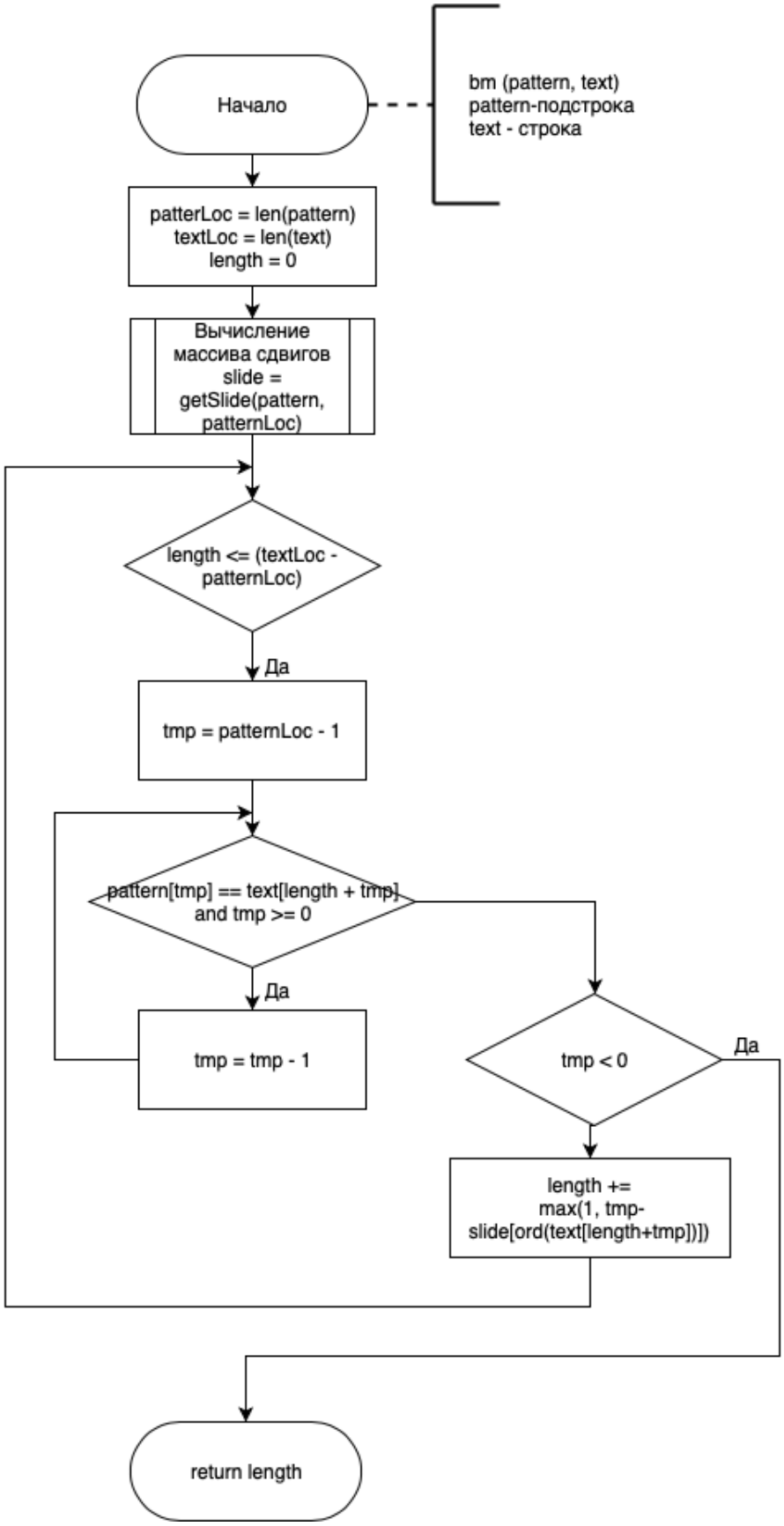

Рис. 2.3: Схема алгоритма Байера-Мура

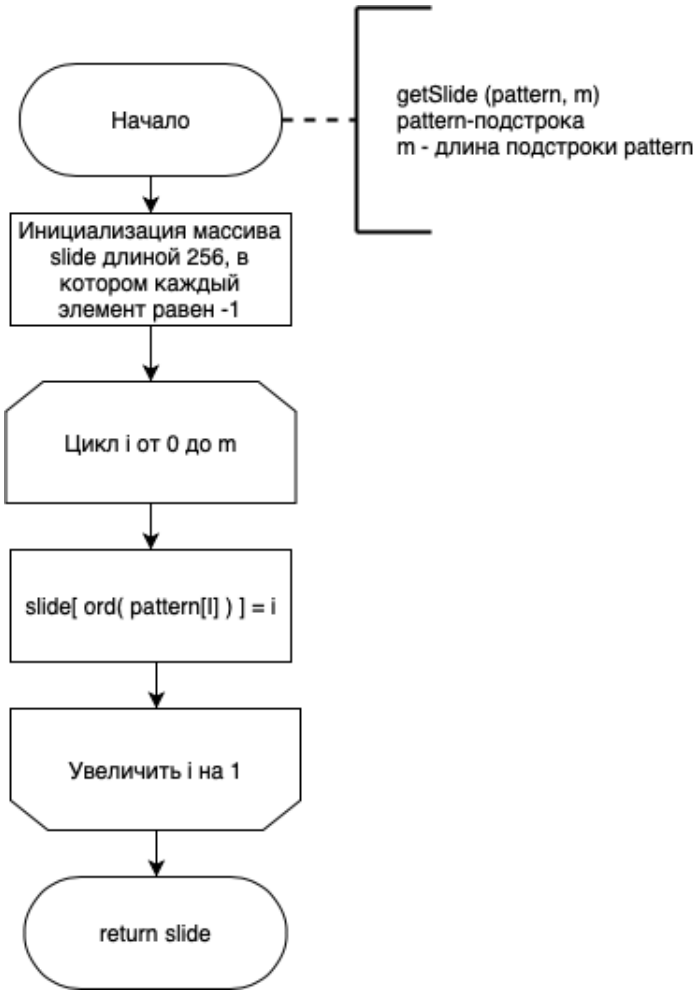

Рис. 2.4: Схема нахождения массива slide в алгоритме Байера-Мура

2.2 Сложность алгоритмов

Алгоритм Кнута-Морриса-Пратта выполняет всего 25 + 2T - 3 сравнений символов, что составляет величину порядка O(T + S), где T – длина строки, S - длина подстроки.

В алгоритме Бойера-Мура число присваиваний равно O(A + P), где A - длина строки, P - длина подстроки[1].

2.3 Вывод

В данном разделе были рассмотрены схемы алгоритмов поиска подстроки в строке, а также приведена сложность для каждого алгоритма.

3. Технологический раздел

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

3.1 Требования к программному обеспечению

Программное обеспечение должно реализовывать алгоритмы Кнута-Морриса-Пратта и Бойера-Мура, предназначенные для поиска подстроки в строке.

3.2 Средства реализации

Для выполнения поставленной задачи был использован язык программирования Python. Среда для разработки XCode.

Версия компилятора Python3.7

3.3 Листинг кода

На основе схемы, приведенной в конструкторском разделе, в соответствии с указанными требованиями к реализации было разработано программное обеспечение, содержащее реализации выбранных алгоритмов. В данном пункте приведены листинги 4.1=4.2 реализации алгоритмов.

Листинг 3.1: Алгорит Кнута-Морриса-Пратта (КМП)

```
def getFail(substring):
  fail = [0]*len(substring)
  for i in range(1,len(substring)):
    k = fail[i-1]
      while k > 0 and substring[k] != substring[i]:
        k = fail[k-1]
    if substring[k] == substring[i]:
      k = k + 1
    fail[i] = k
return fail

def kmp(substring, text):
  index = -1
  f = getFail(substring)
  k = 0
  for i in range(len(text)):
    while k > 0 and substring[k] != text[i]:
```

```
18        k = f[k-1]
19      if substring[k] == text[i]:
20        k = k + 1
21      if k == len(substring):
22        index = i - len(substring) + 1
23        break
24    return index
```

Листинг 3.2: Алгорит Бойера-Мура (БМ)

```
1  def getSlide(pattern, m):
2    slide = 256*(-1)
3    for i in range(m):
4      slide[ ord(pattern[i]) ] = i;
5    return slide
6
7  def bm(pattern, text):
8    patternLoc = len(pattern)
9    textLoc = len(text)
10
11   slide = getSlide(pattern, patternLoc)
12   length = 0
13
14   while(length <= (textLoc - patternLoc)):
15     tmp = patternLoc - 1
16       while (tmp >= 0 and pattern[tmp] == text[length + tmp]):
17         tmp -= 1
18       if (tmp < 0):
19         return length
20       else:
21         length += max(1, tmp-slide[ord(text[length + tmp])])
```

3.4 Вывод

В данном разделе были рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки, а также были представлены листинги кода реализации алгоритма Кнута-Морриса-Пратта и Бойера-Мура.

4. Экспериментальный раздел

В экспериментальном разделе будут представлены примеры работы разработанного программного обеспечения.

4.1 Пример работы

В данном подразделе приведены таблицы 4.1-4.2 с примером работы алгоритмов для поиска подстроки в строке.

Таблица 4.1: Пример работы алгоритма Кнута-Морриса-Пратта

Строка	**Подстрока**	**Вывод**
$'abababcbab'$	$'ababcb'$	2
$'abcabacac'$	$'a'$	0 3 5 7
$'abababacac'$	$'abc'$	-1
$'kfhdlfogklhd'$	$'hd'$	2 10

Таблица 4.2: Пример работы алгоритма Бойера-Мура

Строка	**Подстрока**	**Вывод**
$'abcababbcab'$	$'abb'$	5
$'abcababbcab'$	$'ab'$	0 3 5 9
$'abb'$	$'abcababbcab'$	Нет
$'ergwefd'$	$'ff'$	Нет

4.2 Вывод

В данном разделе были приведены примеры работы разработанного программного обеспечения.

Заключение

В ходе выполнения данной лабораторной работы были изучены два алгоритма для поиск подстроки в строке: Кнута-Морриса-Пратта и Бойера-Мура. Во время разработки программного обеспечения были получены практические навыки реализации указанных алгоритмов на языке Python.

Оба алгоритма эффективнее, чем алгоритм обычного перебора и сравнения символов.

Литература

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