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

Дисциплина: анализ алгоритмов

Лабораторная работа №7

Студент: Власова Екатерина, ИУ7-54

2019 г.

Содержание

Введение

Поиск подстроки в длинном куске текста — важный элемент текстовых редакторов. В программах обработки текстов обычно имеется функция проверки синтаксиса, которая не только обнаруживает неправильно написанные слова, но и предлагает варианты их правильного написания. Один из подходов к проверке состоит в составлении отсортированного списка слов документа. Затем этот список сравнивается со словами, записанными в системном словаре и словаре пользователя; слова, отсутствующие в словарях, помечаются как возможно неверно написанные. Процесс выделения возможных правильных написаний неверно набранного слова может использовать приблизительное совпадение с образцом. При обсуждении приблизительных совпадений строк речь идет о поиске подстроки в данном отрезке текста. Ту же самую технику можно использовать и для поиска приблизительных совпадений в словаре.

Алгоритмы поиска подстроки в строке применяются в текстовых редакторах, СУБД, поисковых машинах, языках программирования и т. п. Задачи поиска слова в тексте используются в криптографии, различных разделах физики, сжатии данных, распознавании речи и других сферах человеческой деятельности.

1 Аналитическая часть

Задача состоит в нахождении первого вхождения некоторой подстроки в длинном тексте. Поиск последующих вхождений основан на том же подходе. Совпадать должна вся строка целиком. Согласно [1], стандартный алгоритм начинает со сравнения первого символа текста с первым символом подстроки. Если они совпадают, то происходит переход ко второму символу текста и подстроки. При совпадении сравниваются следующие символы. Так продолжается до тех пор, пока не окажется, что подстрока целиком совпала с отрезком текста, или пока не встретятся несовпадающие символы. В первом случае задача решена, во втором мы сдвигаем указатель текущего положения в тексте на один символ и заново начинаем сравнение с подстрокой.

Пример:

Текст: there they are

Проход 1: they

Текст : there they are

Проход 2: they

Текст: there they are

Проход 3: they

Текст: there they are

Проход 4: they

Текст: there they are

Проход 5: they

Текст: there they are

Проход 6: they

Текст: there they are

Проход 7: they

Пример 1. Работа стандартного алгоритма поиска подстроки в строке

Поиск образца they в тексте there they are. При первом проходе три первых символа подстроки совпадают с символами текста. Однако только седьмой проход дает полное совпадение. (Совпадение находится после 13 сравнений символов.)

1.1 **Описание алгоритмов**

Описание рассматриваемых алгоритмов поиска шаблона в тексте.

1.1.1 Алгоритм Кнута-Морриса-Пратта

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. Сравнение с образцом нас в следующий узел автомата, а в случае несовпадения мы попадаем в предыдущий узел, отвечающий образцу.

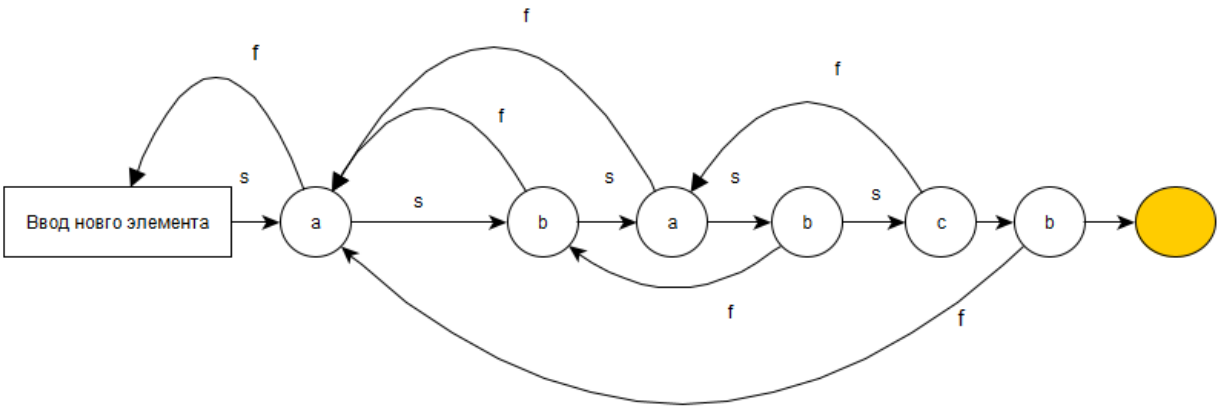

Рисунок 1.1.1 Полный автомат Кнута-Морриса-Пратта для подстроки ababcb

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. Рассмотрим, как задаются переходы по несовпадению. Заметим, что

при совпадении ничего особенного делать не надо: происходит переход к следующему узлу. Напротив, переходы по несовпадению определяются тем, как искомая подстрока соотносится сама с собой.

Дана цепочка T и образец P. Требуется найти все позиции, начиная с которых P входит в T.

Построим строку S = P # T, где # — любой символ, не входящий в алфавит P и T. Посчитаем на ней значение префикс-функции p. Благодаря разделительному символу #, выполняется $\forall i$: $p[i] \leqslant |P|$. Заметим, что по определению префикс-функции при $i>|P|$ и $p[i]=|P|$ подстроки длины P, начинающиеся с позиций 0 и $i-|P|+1$, совпадают. Соберем все такие позиции $i-|P|+1$ строки S, вычтем из каждой позиции $|P|+1$, это и будет ответ. То есть, если в какой-то позиции i выполняется условие $p[i]=|P|$, то в этой позиции начинается очередное вхождение образца в цепочку.

Итак, алгоритм Кнута-Морриса-Пратта состоит из:

1.Префикс-функции

Префикс-функция для i-го символа подстроки возвращает значение, равное максимальной длине совпадающих префикса и суффикса подстроки в образе, которая заканчивается i-м символом. Это значение будем хранить в P[i].

2. Поиска подстроки в строке.

Пример алгоритма Кнута-Морриса-Пратта:

В подстроке abcabeabcabcabd найти все вхождения подстроки abcabd.

1. ***Шаг 1: Префиксная функция, формирование массива P.***

1.1

Subtext: $a\ b\ c\ a\ b\ d$

Массив P: . . . . . .

1.2 Для первого символа любого образа P[0] всегда равно 0:

Subtext: $a\ b\ c\ a\ b\ d$

Массив P: 0 . . . . .

1.3 Рассматриваем Subtext [0] и Subtext [1]. Суффикс b и префикс a длины 1 не совпадают, следовательно P[1] = 0

Subtext: $a\ b\ c\ a\ b\ d$

Массив P: 0 0 . . . . .

1.4 Переходим на следующий элемент subtext[2]. Суффикс и префикс длины 1(a и c) не совпадают, суффикс и префикс длины 2(ab и bc) не совпадают, следовательно, в P[2] записываем 0.

Subtext: $a\ b\ c\ a\ b\ d$

Массив P: 0 0 0 . . .

1.5 Следующий элемент subtext[3]. Префикс и суфикс длины 1 совпадают(subtext[0] и subtext[3]). Префиксы и суфиксы других длин не совпадают, записываем в P[3] длину совпавших префикса и суфикса.

Subtext: $a\ b\ c\ a\ b\ d$

Массив P: 0 0 0 1 . .

1.6 Следующий элемент subtext[4] - b. Префикс и суфикс длины 1 не совпадают, длины 3 также не совпадают, а вот длины 2(ab и ab) совпадают. Следовательно, в P[4] записываем 2.

Subtext: $a\ b\ c\ a\ b\ d$

Массив P: 0 0 0 1 2 .

1.7 Рассматриваем последний элемент подстроки – subtext[5]. Ранее символ d не встречался в подстроке, и ни один префикс с суфиксом любой длины не совпадают. Поэтому записываем 0 в P[5].

Subtext: $a\ b\ c\ a\ b\ d$

Массив P: 0 0 0 1 2 0

2. Поиск подстроки в строке

Text: a b c a b e a b c a b c a b d

Subtext: a b c a b d

Массив P: 0 0 0 1 2 0

2.1 Сравниваем символы подстроки subtext и строки text. Сравнение происходит до элемента text[5], где c != e. Берём содержимое массива P[5-1] = 2. Это значит, что суффикс уже совпал с символами в строке: два символа в начале(префикс) равны двум символам в строке. Таким образом, мы можем «сдвинуть» подстроку.

Text: a b c a b e a b c a b c a b d

Subtext: a b c a b d

Массив P: 0 0 0 1 2 0

2.2 Продолжая сравнение, находим несоответствие text[5] != subtext[2]. Обращаемся к предыдущему символу в масстве P, т.е. P[1] = 0.

Text: a b c a b e a b c a b c a b d

Subtext: a b c a b d

Массив P: 0 0 0 1 2 0

2.3 subtext[0] != text[5].

Text: a b c a b e a b c a b c a b d

Subtext: a b c a b d

Массив P: 0 0 0 1 2 0

2.3 Продолжаем сравнение. Text[6] != subtext[5].

Text: a b c a b e a b c a b c a b d

Subtext: a b c a b d

Массив P: 0 0 0 1 2 0

2.4 Символы строки и подстроки совпадают.

Text: a b c a b e a b c a b c a b d

Subtext: a b c a b d

Массив P: 0 0 0 1 2 0

1.1.2Алгоритм Бойера-Мура

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. Сначала строится таблица смещений для каждого символа, потом происходит сравнение. Строка и подстрока совмещаются по началу, сравнение ведётся по последнему символу. Если символы не совпали, то подстрока смещается вправо на число позиций, взятое из таблицы смещений по символу из исходной строки. И тогда опять сравнение последних символов. Таблица смещений строится так, чтобы пропускать максимальное число незначащих символов, которые заведомо не совпадут.

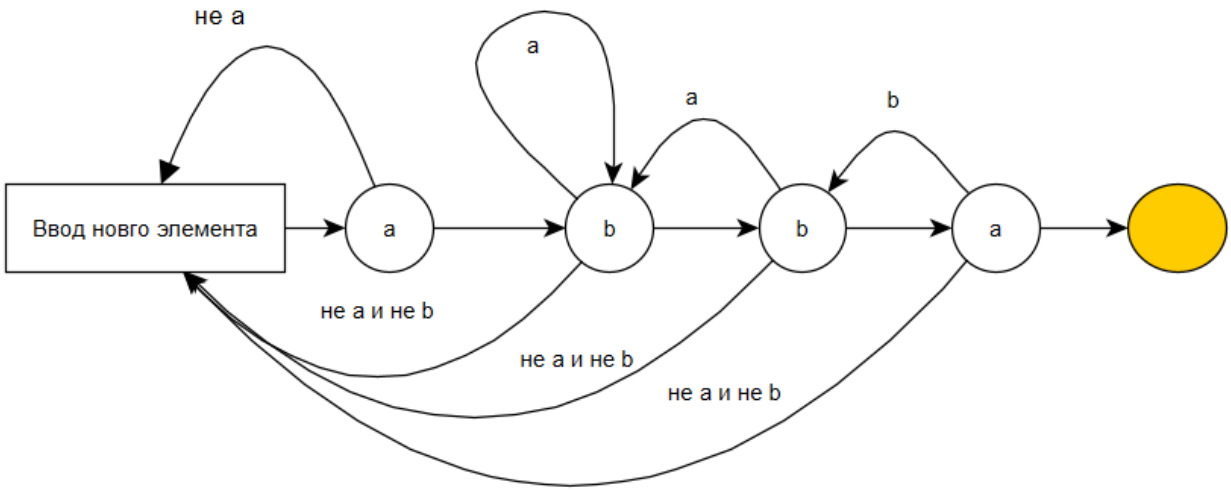

Рисунок 1.1.2 Конечный автомат алгоритма Бойера-Мура

Пример алгоритма Бойера-Мура:

Найдём подстроку «данные» в строке «данныхнетданные».

1. ***Формирование таблицы смещений***

В таблице смещений указывается последняя позиция в подстроке каждого из символов алфавита. Для всех символов, не вошедших в подстроку, заносим длину самой подстроки.

Подстрока: д а н н ы е

Таблица: 5 4 2 2 1 6

2. ***Поиск подстроки в строке***

text***:*** д а н н ы х н е т д а н н ы е

subtext: д а н н ы е

Таблица: 5 4 2 2 1 6

2.1 Начинаем сравнение с конца подстроки, т. е. с subtext[5]. Subtext[5] = е и text[5] = х не совпадают, более того, символа «х» вообще не присутствует в подстроке. Тогда смещаемся на длину подстроки:

text***:*** д а н н ы х н е т д а н н ы е

subtext: д а н н ы е

Таблица: 5 4 2 2 1 6

2.2 Опять начинаем сравнение с конца подстроки. Subtext[5] != text[11], но text[11] = н содержится в подстроке. Сдвигаем подстроку на Таблица[н] = 2.

text*:* д а н н ы х н е т д а н н ы е

subtext: д а н н ы е

Таблица: 5 4 2 2 1 6

2.3 Сравниваем с конца подстроки. Subtext[5] != text[13], но text[13] = ы, этот символ присутствует в подстроке. Используя таблицу смещений, сдвигаем подстроку на 1:

text*:* д а н н ы х н е т д а н н ы е

subtext: д а н н ы е

Таблица: 5 4 2 2 1 6

2.4 Совпадение найдено.

text*:* д а н н ы х н е т д а н н ы е

subtext: д а н н ы е

Таблица: 5 4 2 2 1 6

1.2 Задание на выполнение лабораторной работы

- Описать алгоритмы Кнута-Морриса-Пратта и Бойера-Мура;
- Реализовать алгоритмы Кнута-Морриса-Пратта и Бойера-Мура;

1.3 Вывод по аналитической части

Были описаны алгоритмы Кнута-Морриса-Пратта и Бойера-Мура, поставлены задачи на выполнение лабораторной работы.

2 Конструкторская часть

В данном разделе представлены блок-схемы рассматриваемых алгоритмов.

2.1 Разработка алгоритмов

2.1.1 Алгоритм Кнута-Морриса-Пратта

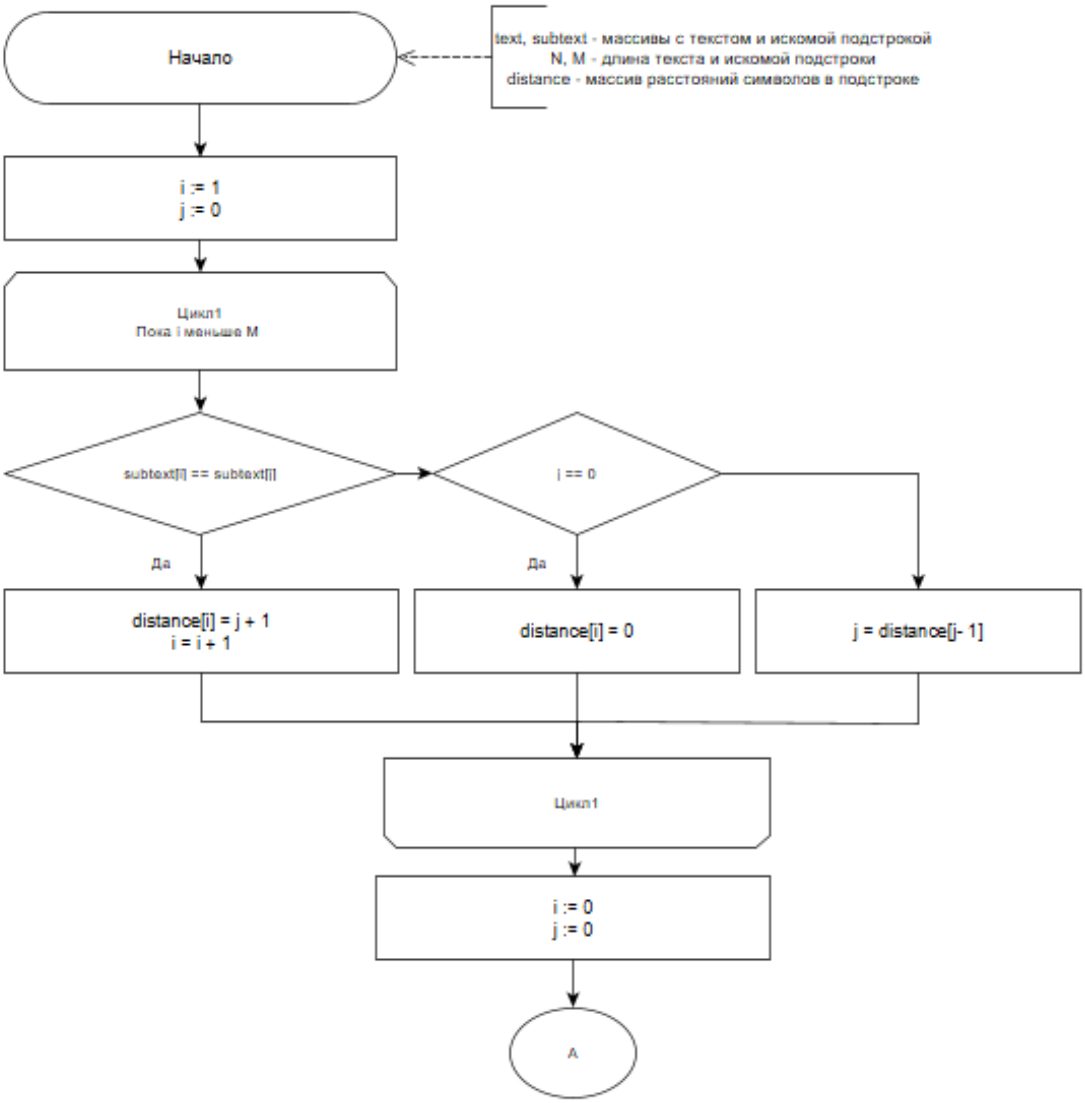

Рисунок 2.1.1.1 Алгоритм Кнута-Морриса-Пратта (Начало)

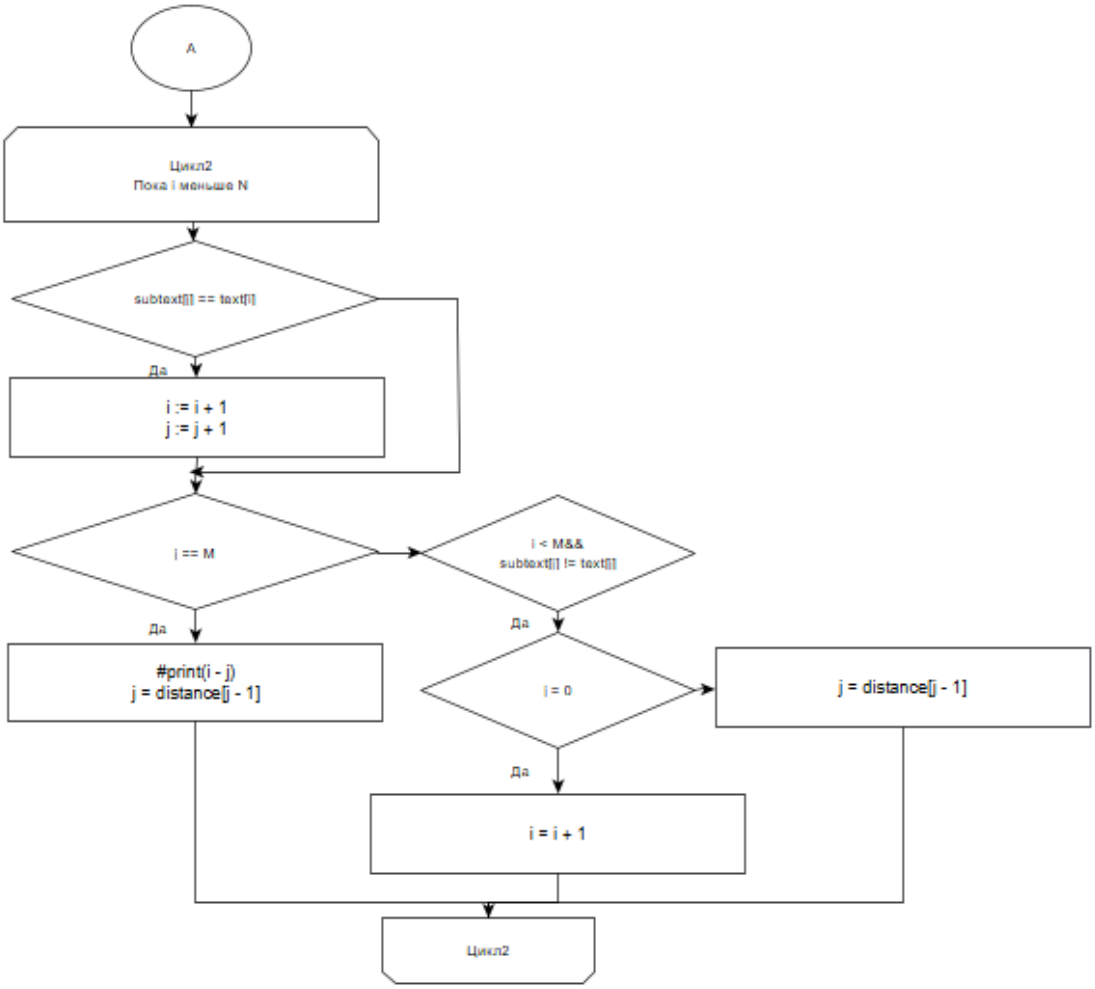

Рисунок 2.1.1.2 Алгоритм Кнуса-Морриса-Пратта (Окончание)

2.1.2 Алгоритм Бойера-Мура

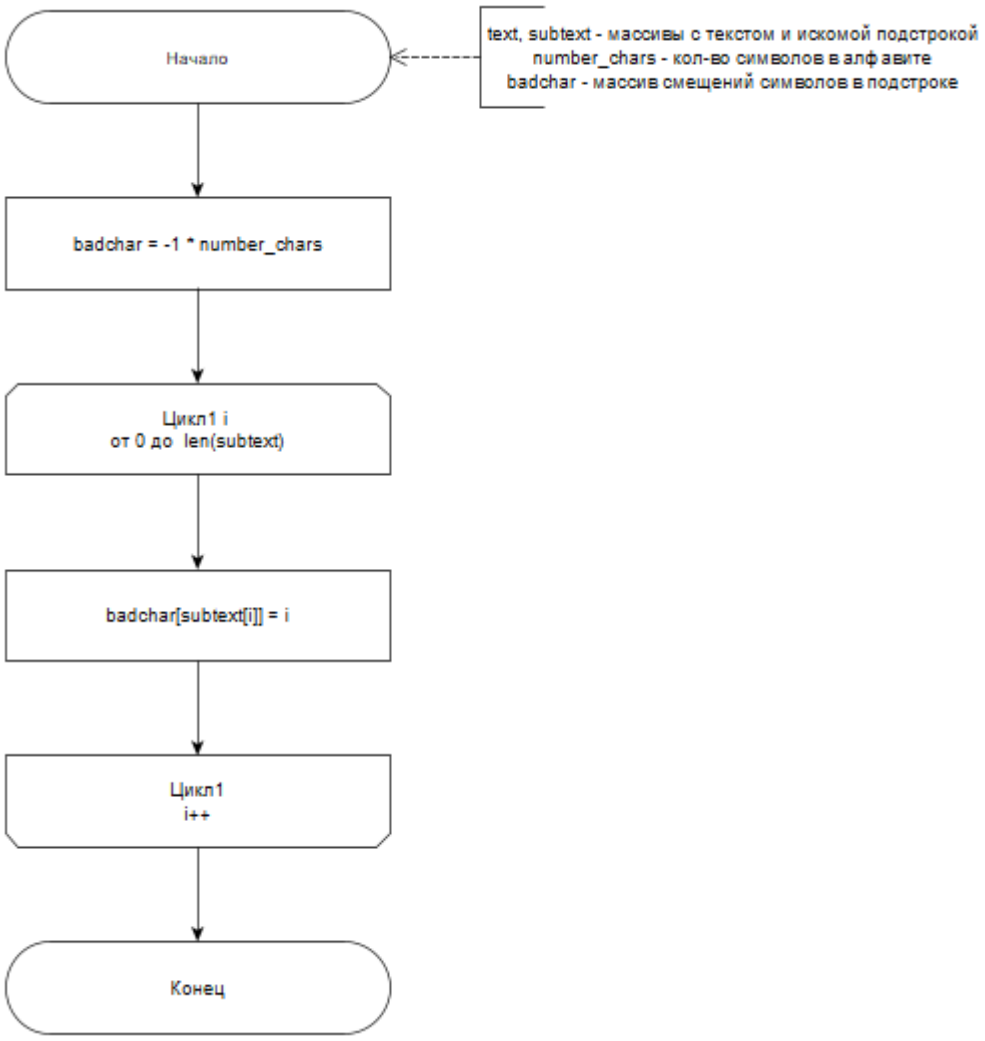

Рисунок 2.1.2.1 Алгоритм Бойера-Мура, инициализация массива

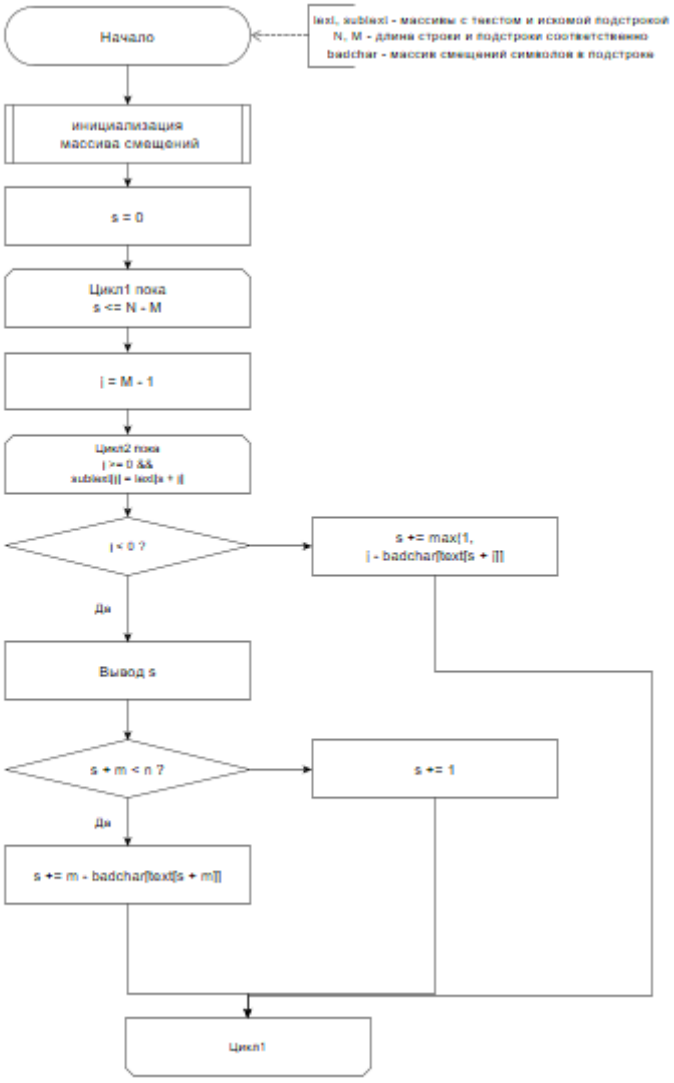

Рисунок 2.1.2.2 Алгоритм Бойера-Мура, поиск

2.2 Выводы по конструкторскому разделу

Представлены блок-схемы реализуемых алгоритмов.

3 Технологическая часть

В данном разделе сделан выбор используемого языка программирования, предоставлен код алгоритмов.

3.1 **Требования к программному обеспечению**

Программа должна поддерживать пользовательский режим, в котором пользователь может с клавиатуры задать строку и подстроку, вхождение которой необходимо найти. Программа должна корректно отрабатывать при пустом вводе.

3.2 **Средства реализации**

В качестве языка программирования в данной работе был выбран Forth – конкатентативный язык программирования, в котором программы записываются последовательностью лексем. Этот язык был выбран с целью ознакомления с особенностями его работы. Его стандарт приведён в [6].

3.3 Листинг кода

3.3.1 Алгоритм Кнута-Морриса-Прата

Листинг 0

```
: KMP
  0 1
  BEGIN
  DUP len2 <
  WHILE
  DUP subtext SWAP chars + c@
  >R
  OVER subtext SWAP chars + c@
  R>
  = IF
  SWAP 1 + OVER OVER
  SWAP distance [!]
  SWAP 1 +
  ELSE
    OVER 0=
    IF
      DUP 0 SWAP distance [!]
      1 +
    ELSE
      SWAP 1 - distance swap cells + @
  SWAP
    THEN
  THEN
  REPEAT
  DROP DROP
  0 0
  BEGIN
  DUP len1 <
  WHILE
  OVER subtext SWAP chars + c@
  OVER text SWAP chars + c@
  = IF
  1 + SWAP
  1 + SWAP
  THEN
   OVER len2 =
    IF
    OVER OVER SWAP - .
    SWAP 1 - distance SWAP cells + @ SWAP
    ELSE
    DUP len1 < >R
    OVER subtext SWAP chars + c@
    OVER text SWAP chars + c@
    <>
    R>
    AND IF
    OVER 0 =
    IF
```

```
   1 +
   ELSE
  SWAP 1 - distance swap cells + @ SWAP
  THEN
  THEN
  THEN
  REPEAT
;
```

3.3.2 Алгоритм Бойера-Мурра

Листинг 1

```
: BM1
  CR ." Boyer–Moore: "
  256 0 ?DO
  -1 I slide [!]
  LOOP
  len2 0 ?DO
  I subtext I chars + c@ slide [!]
  LOOP

  len2 len1 0
  BEGIN
  >R
  OVER OVER
  SWAP -
  R>
  DUP
  ROT
  <=
  WHILE
  len2 1 -
    BEGIN
    DUP 0 >=
     >R
    DUP subtext
    SWAP chars + c@
    >R
    OVER OVER +
   text SWAP chars + c@
    R>
    =
    R>
    AND
    WHILE
    1 -
    REPEAT
  DUP 0 <
   IF
  OVER .
```

```
  >R
  >R
  OVER R>
  DUP ROT
  +
  >R OVER
    R> >
  R> SWAP
  IF
    >R >R OVER
    R>
    DUP
    >R +
    text SWAP chars + c@
    slide SWAP cells + @
    >R
   OVER
   R> -
    R>
    +
    R>
    ELSE
    SWAP 1 + SWAP
    THEN

  ELSE
  OVER OVER +
  text SWAP chars + c@
  slide SWAP cells + @
  OVER SWAP -
  1
  MAX
  ROT +
  SWAP
  THEN
  DROP
  REPEAT
;
```

3.4 Выводы по технологическому разделу

Был выбран язык программирования Forth, средство реализации языка Gforth. Реализованы функции нахождения подстроки в строке. Описаны требования к ПО.

4 Экспериментальная часть

В данном разделе предоставлены примеры работы программы.

4.1 Примеры работы

Подстрока	Строка	Ожидаемый результат	Алгоритм Кнута-Морриса-Прата	Алгоритм Бойера-Мура
ab	abcd	0	0	0
bc	abcd	1	1	1
aa	aaaa	0 1 2	0 1 2	0 1 2
j	abdjd	3	3	3

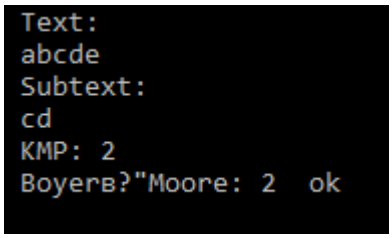

Рисунок 4.1 Пример работы программы

4.2 Вывод

В данном разделе представлены примеры работы алгоритмов поиска подстроки в строке.

Заключение

В результате выполнения данной лабораторной работы были реализованы алгоритмы поиска подстроки в строке: алгоритм Кнту-Морриса-Прата и алгоритм Бойеса-Мура.

Список использованной литературы

1. Дж. Макконелл Анализ алгоритмов, 2009.
2. Т. Кормен. Алгоритмы. Построение и анализ, 2013.
3. Гасфилд Д., Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология, 2003.
4. Kurtz, St. Fundamental Algorithms For A Declarative Pattern Matching System, 1995.
5. Ахметов И. Поиск подстрок с помощью конечных автоматов, 2008.
6. Скиена С. Алгоритмы. Руководство по разработке. 2-е изд.: Пер. с англ. — СПб.: БХВ-Петербург. 2011. — 720 с.: ил.
7. Стандарт Forth, [Электронный ресурс], URL: https://forth-standard.org/
8. Л. Броуди, Начальный курс программирования на языке ФОРТ, 1990.
9. Stephen Pelc, Programming Forth Stephen Pelc, 2005.
10. Leo Brodie, Thinking Forth, 1984.
11. J. L. Bezemer, And so forth, 2004.
12. С. Н. Баранов, Н. Р. Ноздрунов, Язык Форт и его реализации, 1988.
13. Язык Форт. [Электронный ресурс] URL: https://www.forth.org.ru/
14. Gforth Manual.[Электронный ресурс] URL: http://www.complang.tuwien.ac.at/forth/gforth/Docs-html/