МГТУ им. Баумана

Лабораторная работа №7

По курсу: "Анализ алгоритмов"

Поиск подстроки в строке

Работу выполнил: Мокеев Даниил, ИУ7-54

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

Москва, 2019

Оглавление

Введение

Муравьиный алгоритм — один из эффективных полиномиальных алгоритмов для нахождения приближённых решений задачи коммивояжёра, а также решения аналогичных задач поиска маршрутов на графах.

Целью данной лабораторной работы является изучение муравьиных алгоритмов и приобретение навыков параметризации методов на примере муравьиного алгоритма, примененного к задаче коммивояжера.

Задачи данной лабораторной работы:

- рассмотренть муравьиный алгоритм и алгоритм полного перебора в задаче коммивояжера;
- реализовать эти алгоритмы;
- сравнить время работы этих алгоритмов.

1 Аналитическая часть

В данной части будут рассмотрены существующие на данный момент алгоритмические решения проблемы поиска подстроки в строке.

1.1 Общие сведения об алгоритмах поиска подстроки

Поиск подстроки в строке — одна из простейших задач поиска информации. Сферы применения алгоритмов поиска включают в себя:

- Текстовые редакторы;
- СУБД;
- компиляторы;
- программы определения проверки плагиата;
- поисковые системы;
- биоинформатика.

На сегодняшний день существует огромное разнообразие алгоритмов поиска подстроки. Программисту приходится выбирать подходящий в зависимости от таких факторов: длина строки, в которой происходит поиск, необходимость оптимизации, размер алфавита, возможность проиндексировать текст, требуется ли одновременный поиск нескольких строк. В данной лабораторной работе будут рассмотремы два алгоритма сравнения с образцом, алгоритм Кнута-Морриса-Пратта и алгоритм Бойера-Мура.

1.1.1 Стандартный алгоритм

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[2].

1.1.2 Алгоритм 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. В этом алгоритме кроме таблицы суффиксов применяется таблица стоп-символов. Она заполняется для каждого сивола в алфавите. Для каждого встречающегося в подстроке символа таблица заполняется по принципу максимальной позиции символа в строке, за исключением последнего символа. При определении сдвига при очередном несовпадении строк, выбирается максимальное значение из таблицы суффиксов и стоп-символов[2].

Таблица 2. Пошаговая работа алгоритма Бойера-Мура.

a	b	a	b	a	c	a	b	a	a
a	b	a	a						
		a	b	a	a				
						a	b	a	a

1.1.3 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. Успешное сравнение переводит нас в следующий узел автомата, а в случае несовпадения мы попадаем в предыдущий узел, отвечающий образцу. В программной реализации этого алгоритма применяется масив сдвигов, который создается для каждой подстроки, которая ищется в тексте. Для каждого символа из подстроки рассчитывается значение, равное максимальной длине совпадающего префикса и суффикса отсительно конкретного элемента подстроки. Создание этого массива позволяет при несовпадении строки сдвигать ее на расстояние, большее, чем 1 (в отличие от стандартного алгоритма).

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

a	b	a	b	a	c	a	b	a	a
a	b	a	a						
		a	b	a	a				
				a	b	a	a		
					a	b	a	a	
						a	b	a	a

Вывод

В данном разделе были рассмотрены алгоритмы для решения задачи поиска подстроки в строке.

2 Конструкторская часть

В данном разделе будут рассмотрены основные требования к программе и схемы алгоритмов.

2.1 Требования к программе

Требования к вводу:

- Подаются не пустые подстрока и строка;
- длина подстроки меньше, чем длина строки.

Требования к программе:

- Программа должна возвращать первое индекс первого вхождения подстроки.

.

Входные данные - на вход подается строка и подстрока;
Выходные данные - программа возвращает индекс первого вхождения подстроки в строку, если вхождения не было возвращается -1.

2.2 Схемы алгоритмов

В данном разделе будут приведены схемы алгоритмов для решения задачи поиска подстроки в строке: Бойера-Мура(Рис.2.1) и Кнута-Морриса-Пратта (Рис. 2.2)

Вывод

В данном разделе были рассмотрены требования к программе и схемы алгоритмов.

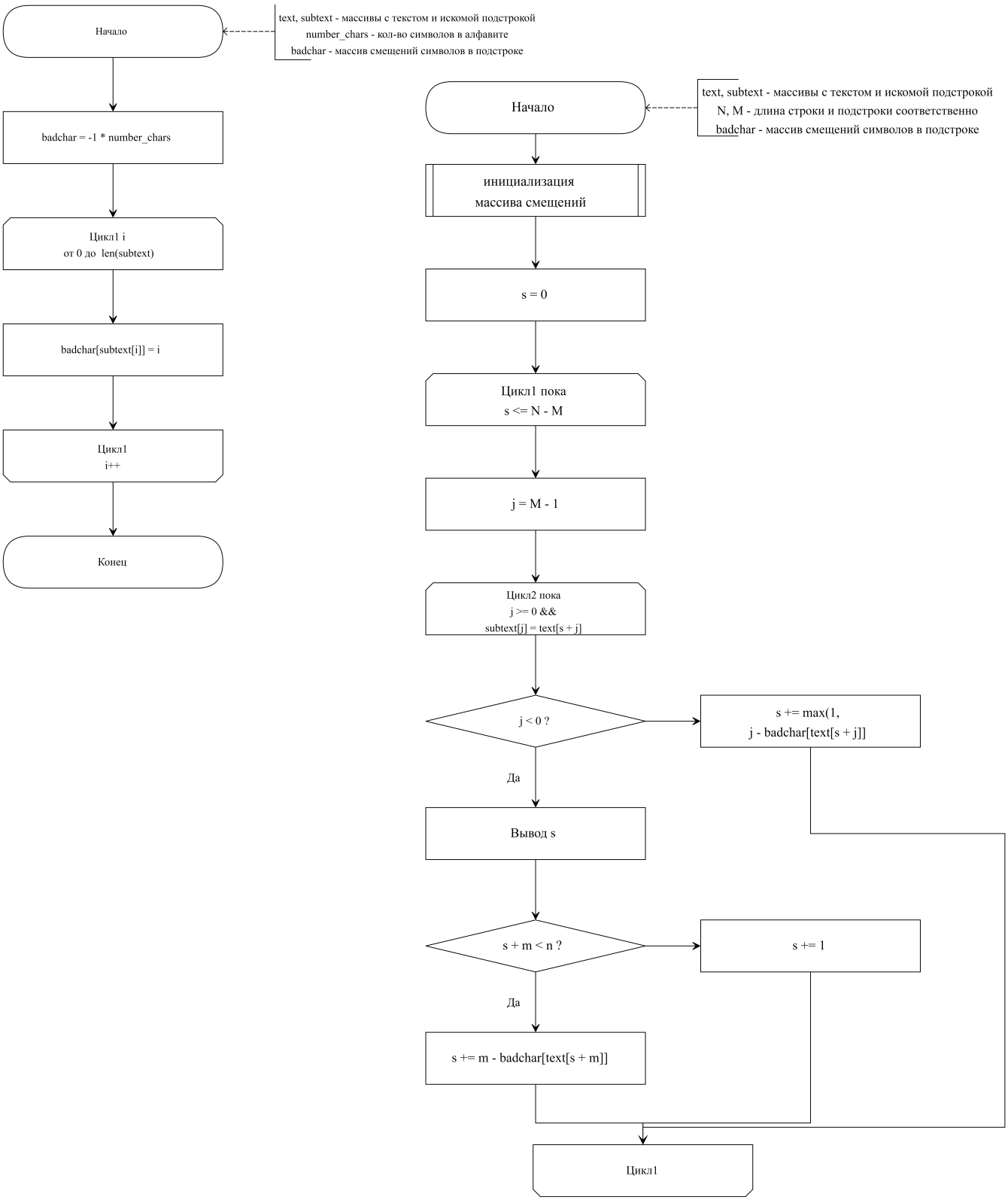

Рис. 2.1: Схема алгоритма Бойера-Мура

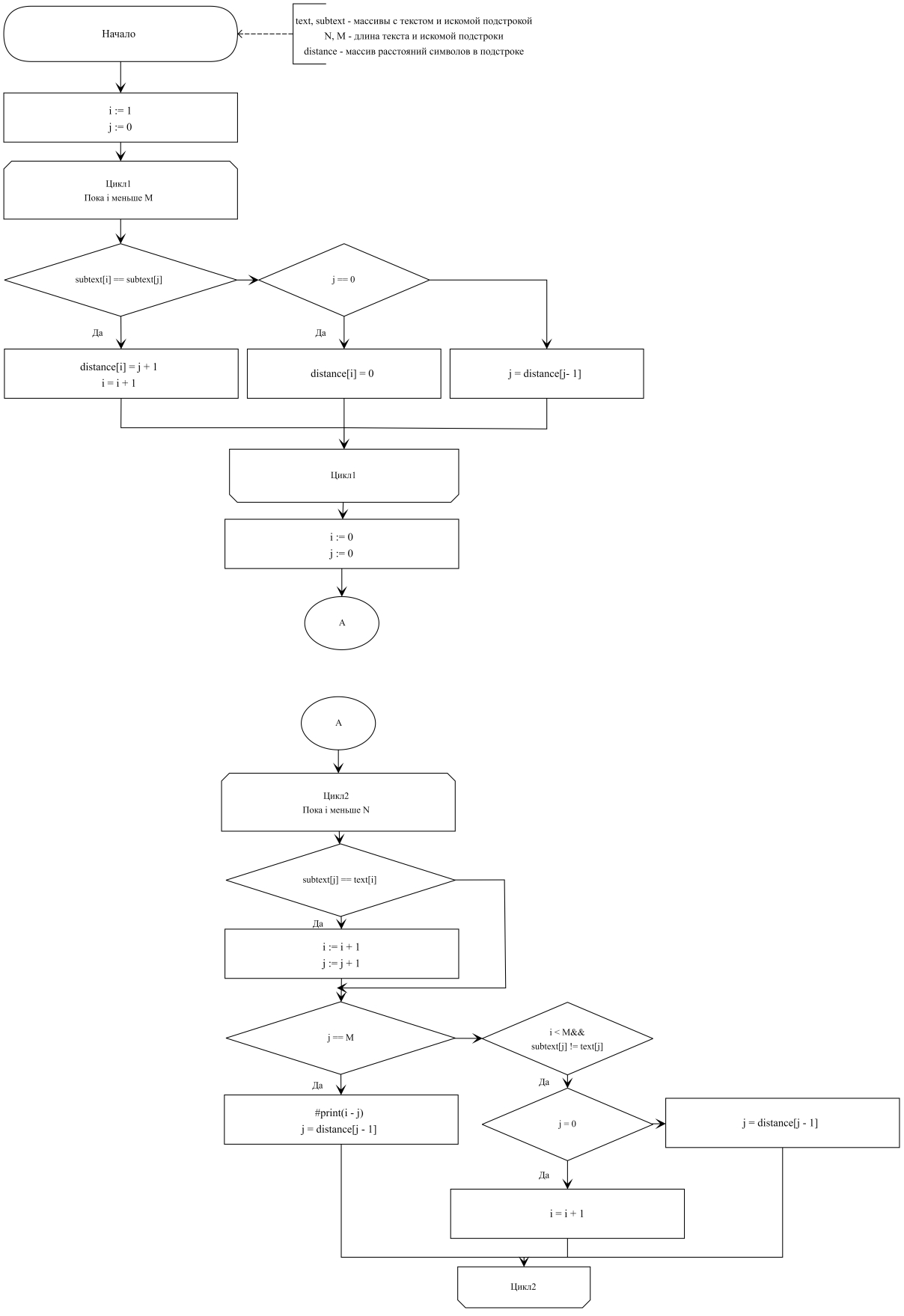

Рис. 2.2: Схема алгоритма Кнута-Морриса-Пратта

3 Технологическая часть

3.1 Выбор ЯП

В качестве языка программирования был выбран golang. Время работы алгоритмов было замерено с помощью time.

3.2 Описание структуры ПО

В данном разделе будет рассмотрена структура ПО (Рис. 3.1)

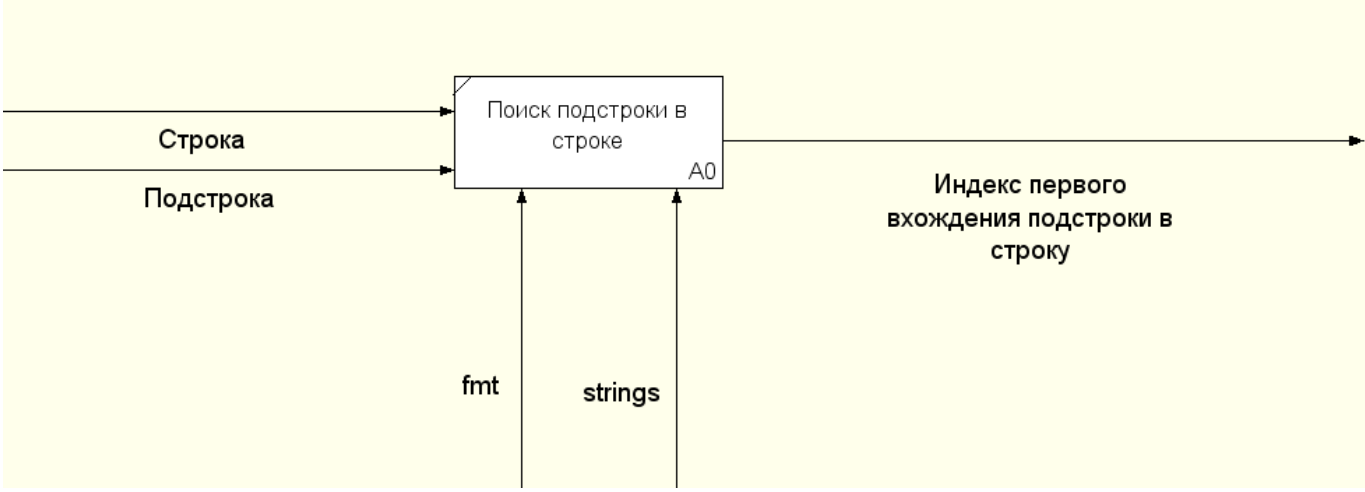

Рис. 3.1: Функциональная схема поиска подстроки в строке (IDEF0 диаграмма 1 уровня)

3.3 Сведения о модулях программы

Программа состоит из:

- main.go- главный файл программы, в котором располагается точка входа в программу.
- estim.go - файл содержащий функции замера времени.

3.4 Примеры работы

В данном разделе приведен пример работы программы (Рис. 3.2)

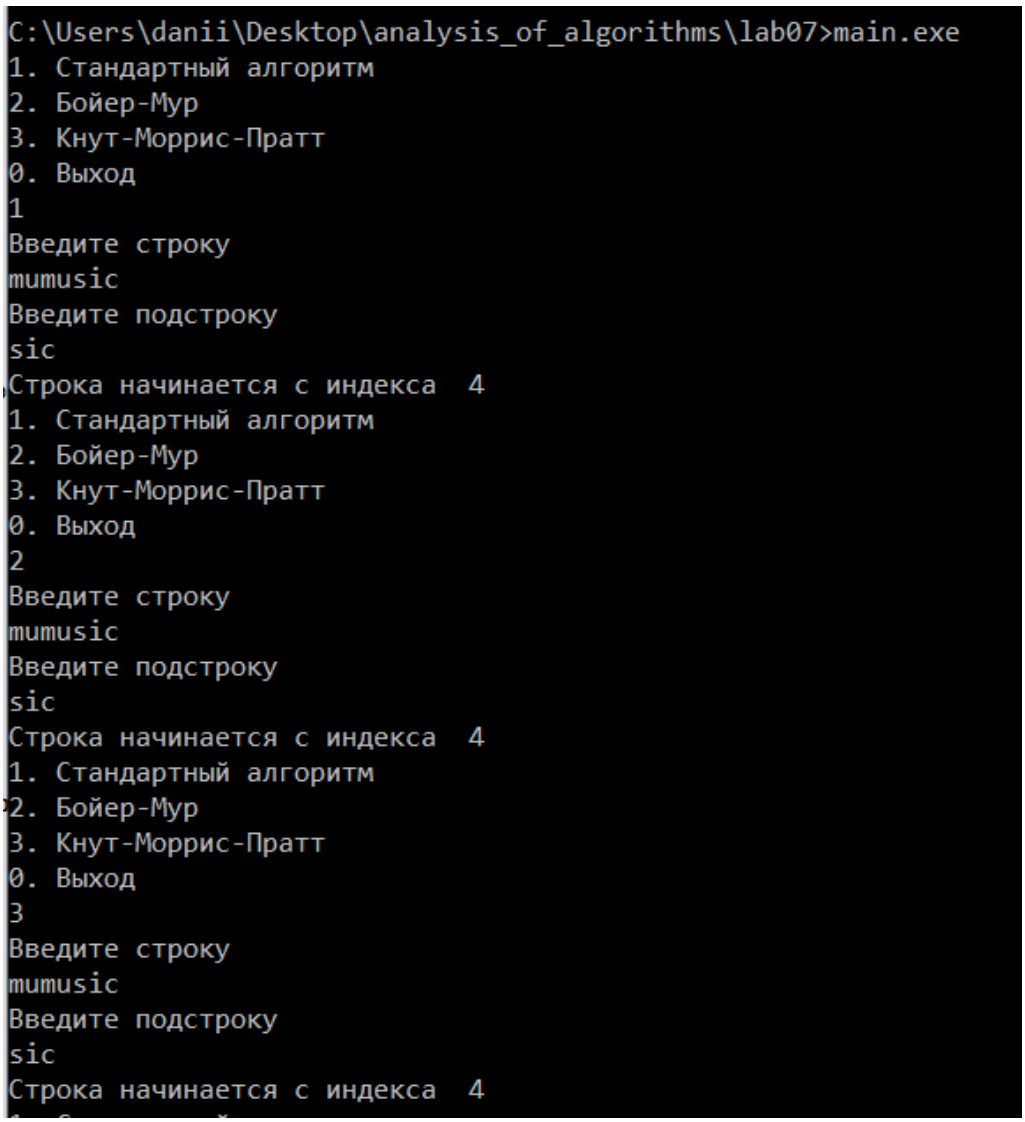

Рис. 3.2: Пример работы программы

3.5 Листинг кода алгоритмов

В данном разделе будут приведены листинги кода стандартного алгоритма поиска подстроки в строке (Листинг 3.1), алгоритма Бойера-Мура (Листинг 3.2) и алгоритма Кнута-Морриса-Пратта (Листинг 3.3)

Листинг 3.1: Стандартный алгоритм поиска подстроки в строке

```
1
2 func std(str, sub string) int{
3   if len(sub) > len(str) || len(sub)== 0 || len(str) == 0{
4     return -1
5   }
6
7   flag := true
8   for i:=0;i<len(str);i++{
9     flag = true
```

```
10      if str[i] == sub[0]{
11        for j:=0;j<len(sub);j++{
12          if str[i+j] != sub[j]{
13            flag = false
14          }
15          if flag{
16            return i
17          }
18        }
19      }
20    }
21    return -1
22  }
```

Листинг 3.2: Алгоритм Бойера-Мура

```
1  func get_table(substring string) map[rune]int {
2    length := utf8.RuneCountInString(substring)
3    runes := []rune(substring)
4
5    table := make(map[rune]int)
6
7    for i := 0; i < length; i++ {
8      j := runes[i]
9      table[j] = length - i - 1
10   }
11   return table
12 }
13
14 func BM(str, sub string) int{
15   if len(sub) > len(str) || len(sub)== 0 || len(str) == 0{
16     return -1
17   }
18   table := get_table(sub)
19   strrunes := []rune(str)
20   subrunes := []rune(sub)
21   str_l := utf8.RuneCountInString(str)
22   sub_l := utf8.RuneCountInString(sub)
23   i:=sub_l-1
24   j,k:=i,i
25   for (j>=0 && i<=str_l-1){
26     j = sub_l-1
27     k = i
28     for(j>=0 && (table[strrunes[k]] == table[subrunes[j]])){
```

```
29        k--
30        j--
31      }
32      if _,ok := table[strrunes[i]]; !ok{
33        i+= sub_l
34      }else{
35        i+=table[strrunes[i]]
36      }
37    }
38    if k>= str_l - sub_l{
39      return -1
40    } else {
41      return k+1
42    }
43  }
```

Листинг 3.3: Алгоритм Кнута-Морриса-Пратта

```
1  func get_preph(str string) []int{
2    res := make([]int, len(str))
3    for i:= 1; i<len(str);i++{
4      j := res[i-1]
5      for (j>0 && str[i] != str[j]){
6        j = res[j-1]
7      }
8      if (str[i] == str[j]){
9        j++
10     }
11     res[i] = j
12   }
13   return res
14 }
15
16 func KMP(str, sub string) int{
17   if len(sub) > len(str) || len(sub)== 0 || len(str) == 0{
18     return -1
19   }
20
21   tmp := []string{sub, str}
22   str = strings.Join(tmp, "@")
23   pr := get_preph(str)
24   len_sub := len(sub)
25   for i:=len_sub+1; i<len(str);i++{
26     if pr[i] == len_sub{
```

```
27          return i-2*len_sub
28        }
29      }
30      return -1
31  }
```

Вывод

В данном разделе были рассмотрены основные сведения о модулях программы и листинг кода алгоритмов.

Заключение

Были реализованы и изучены основные существующие алгоритмы поиска подстроки в строке - стандартный тривиальный алгоритм, алгоритм Бойера-Мура и алгоритм Кнута-Морриса-Пратта.

Литература

[1] Окулов С. М. Алгоритмы обработки строк. — М.: Бином, 2013. — 255 с.

[2] Дж. Макконнелл. Анализ лгоритмов. Активный обучающий подход

[3] Основные сведения о модульных тестах [Электронный ресурс], - режим доступа: https://docs.microsoft.com/ru-ru/visualstudio/test/unit-test-basics?view=vs-2019