**Федеральное государственное бюджетное образовательное учреждение   
высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

**Анализ алгоритмов**

**Лабораторная работа №8**

**Отчёт на тему:**

# «Задача поиска подстроки в строке»

Выполнила:

Янова Даниэлла

ИУ7-53

**Москва, 2018**

**Введение**

**Цель работы:** изучение алгоритмов поиска подстроки в строке, в частности, алгоритма Кнута-Морриса-Пратта и алгоритма Бойера-Мура.

1. **Аналитическая часть**

В данном разделе приведено описание алгоритмов поиска подстроки.

* 1. **Задача поиска подстроки**

Пусть дана некоторая строка T (текст) и подстрока W (слово). Задача поиска подстроки сводится к поиску вхождения этой подстроки в указанной строке. Строго задача формулируется следующим образом: пусть задан массив T из N элементов и массив W из M элементов, 0<M≤N. Если алгоритм поиска подстроки обнаруживает вхождение W в T, то возвращается индекс, указывающий на первое совпадение подстроки со строкой.

**Простейшим алгоритмом является примитивный алгоритм. Данный алгоритм выглядит следующим образом:**

1. I = 1, J = 1
2. Сравнение T[I] с W[J]
3. Совпадение: J = J + 1, I = I +1
4. Несовпадение: J = 1, I = I + 1
5. Если J = M, то подстрока найдена
6. Если I + M > N, то подстрока отсутствует

В худшем случае алгоритм делает MN сравнений, в среднем – 2N сравнений.

* 1. **Алгоритм Кнута-Морриса-Пратта**

Алгоритм Кнута-Морриса-Пратта позволяет улучить показатель количества сравнений: данный алгоритм требует только N сравнений в худшем случае.

Идея алгоритма в том, что при каждом несовпадении T[I] и W[J] мы сдвигаемся не на единицу, а на J, так как меньшие сдвиги не приведут к полному совпадению. К сожалению, этот алгоритм поиска дает выигрыш только тогда, когда несовпадению предшествовало некоторое число совпадений, иначе алгоритм работает как примитивный. Так как совпадения встречаются реже, чем несовпадения, выигрыш в большинстве случаев незначителен.

* 1. **Алгоритм Бойера-Мура**

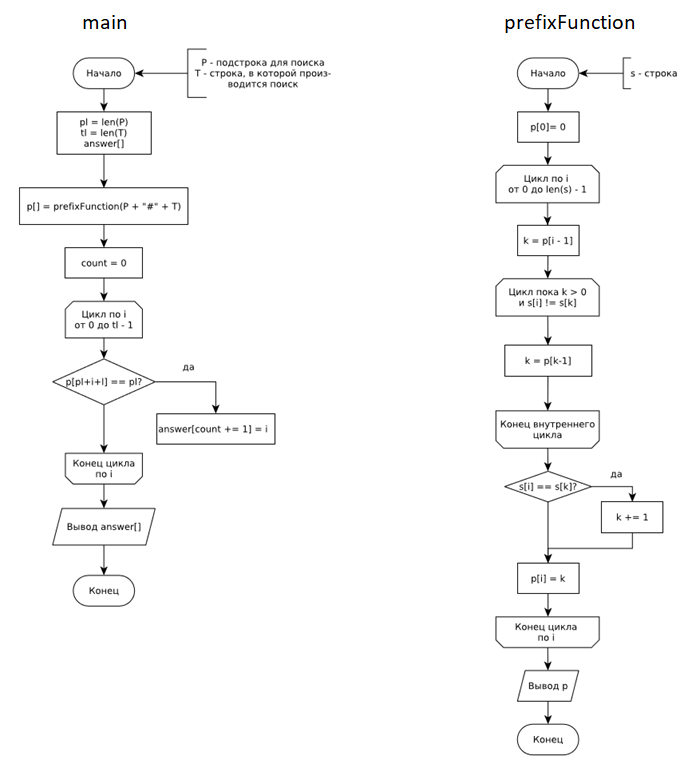
Алгоритм Бойера-Мура из трёх алгоритмов считается наиболее быстрым. В среднем он делает сравнений меньше, чем N. Данный алгоритм считается стандартным для поиска на странице браузера или в текстовых редакторах.

Особенностью алгоритма является то, что сравнение подстроки выполняется справа налево, а не с первого символа. Для оптимизации сдвигов производится предобработка подстроки, вычисляются значения сдвига для каждой из букв.

2. **Конструкторская часть**

В данном разделе представлены схемы алгоритмов.

* 1. **Схемы алгоритмов**



**Рис.** 1 – схема алгоритма Кнута-Морриса-Пратта

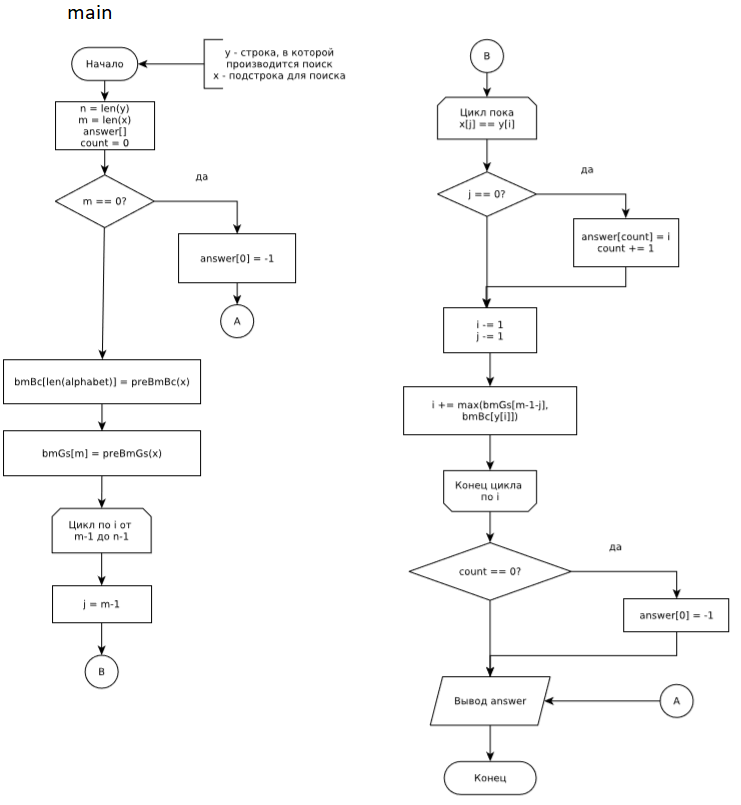


Рис.2-1 Схема алгоритма Бойера-Мура ч.1

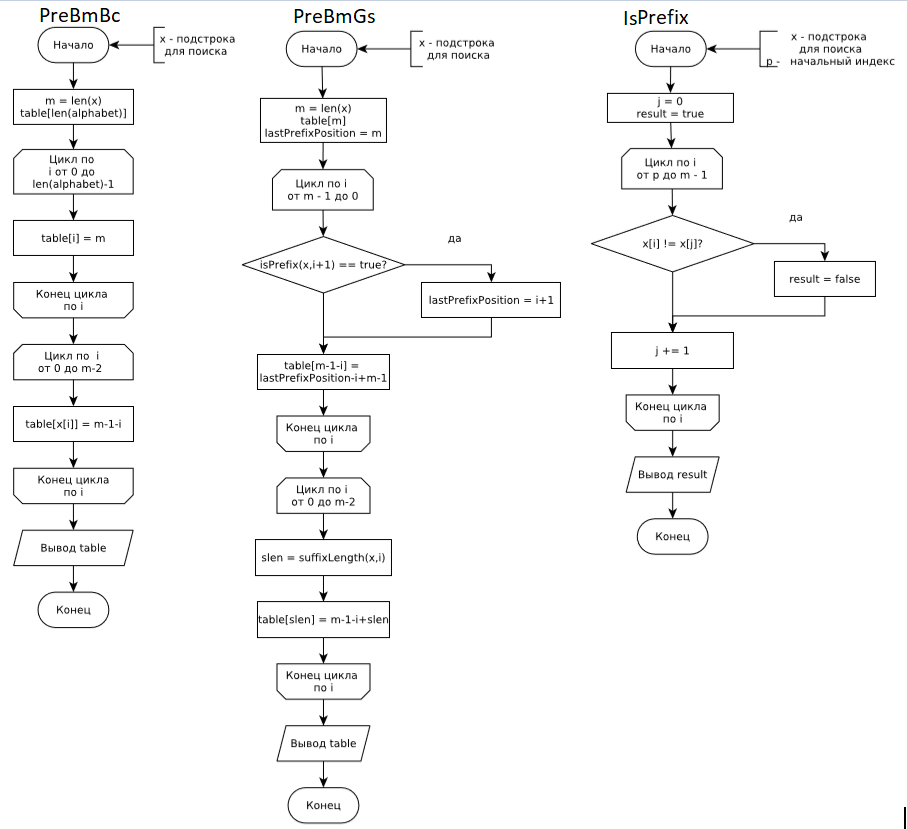


Рис. 2-2 – Схема Бойера-Мура ч.2

**2.2 Вывод**

В данном разделе в соответствии с теоретическим описанием алгоритмов, представленным в аналитическом разделе, были построены схемы указанных алгоритмов. Данные схемы позволяют реализовать представленные ими алгоритмы на любом языке программирования.

**3. Технологическая часть**

В данном разделе представлена реализация алгоритмов, указан язык программирования, а также необходимые модули.

**3.1 Средства реализации**

Лабораторная работа была выполнена на языке Python. Использован компьютер с операционной системой Ubuntu 18.10, процессором Intel Core i5 1,7 ГГц и 6 ГБ DDR3 1600 МГц оперативной памяти.

**3.2 Листинг кода**

**Листинг 1: Алгоритм Кнута-Морриса-Пратта**

1.def KMPSearch(pattern, text):

2. M = len(pattern)

3. N = len(text)

4. longestPrefixSuffix = [0]\*M

5. j = 0

6. computeLPSArray(pattern, M, longestPrefixSuffix)

7. i = 0

8. while i < N:

9. if pattern[j] == text[i]:

10. i += 1

11. j += 1

12. if j == M:

13. print("Found␣pattern␣at␣index␣" + str(i - j))

14. j = longestPrefixSuffix[j-1]

15. elif i < N and pattern[j] != text[i]:

16. if j != 0:

17. j = longestPrefixSuffix[j-1]

18. else:

19. i += 1

20.

21. def computeLPSArray(pattern, M, longestPrefixSuffix):

22. len = 0

23. longestPrefixSuffix[0]

24. i = 1

25. while i < M:

26. if pattern[i]== pattern[len]:

27. len += 1

28. longestPrefixSuffix[i] = len

29. i += 1

30. else:

31. if (len != 0):

32. len = longestPrefixSuffix[len-1]

33. else:

34. longestPrefixSuffix[i] = 0

35. i += 1

**Листинг 2: Алгоритм Бойера-Мура**

1. def badCharHeuristic(string, size):

2. badChar = [-1] \* NO\_OF\_CHARS

3. for i in range(size):

4. badChar[ord(string[i])] = i;

5. return badChar

6.

7. def BMSearch(text, pattern):

8. m = len(pattern)

9. n = len(text)

10. badChar = badCharHeuristic(pattern, m)

11. s = 0

12. while(s <= (n - m)):

13. j = m - 1

14. while (j >= 0 and pattern[j] == text[s+j]):

15. j -= 1

16. if (j < 0):

17 print("Pattern␣occur␣at␣shift␣=␣{}".format(s))

18. s += (m-badChar[ord(text[s+m])] if s+m<n else 1)

19. else:

20. s += max(1, j-badChar[ord(text[s+j])])

**3.4 Вывод**

На основе схемы алгоритма, представленного в конструкторском разделе, в соответствии с указанными требованиями к реализации с использованием средств языка Python было разработано программное обеспечение, содержащее реализацию указанного алгоритма.

1. **Экспериментальная часть**

В данном разделе будут произведены примеры работы алгоритмов.

**4.1 Пример работы**

Ввод: строка: "THIS IS A TEST TEXTподстрока: "TEST".

Вывод:

1)Алгоритм Кнута-Морриса-Пратта: найдена подстрока со сдвигом 10; 2)Алгоритм Бойера-Мура: найдена подстрока со сдвигом 10.

Ввод: строка:"AABAACAADAABAABAподстрока "AABA".

Вывод:

1)Алгоритм Кнута-Морриса-Пратта: найдена подстрока со сдвигом 0,9,12. 2)Алгоритм Бойера-Мура: найдена подстрока со сдвигом 0,9,12.

**4.2 Вывод**

На основе приведённых выше примеров работы ПО можно сделать вывод, что алгоритмы реализованы верно.

**Заключение**

В процессе выполнения лабораторной работы были изучены два алгоритма поиска подстрок. Во время разработки программного обеспечения в соответствии с поставленными требованиями были получены практические навыки реализации указанных алгоритмов на языке Python.