Анализ Алгоритмов

Лабораторная работа №7

По теме “*Методы поиска подстроки в строке*”

Студент: Юмаев Артур

Группа: ИУ7-55

Оглавление

[Введение 3](#_Toc27602978)

[1. Аналитическая часть 4](#_Toc27602979)

[1.1 Обзор алгоритма Кнута-Мориса-Прата 4](#_Toc27602980)

[1.2 Обзор алгоритма Бойера-Мура 5](#_Toc27602981)

[Введение 5](#_Toc27602982)

[1.2.1 Сканирование слева направо, сравнение справа налево 6](#_Toc27602983)

[1.2.2 Эвристика стоп-символа 6](#_Toc27602984)

[1.2.3 Эвристика совпавшего суффикса 6](#_Toc27602985)

[Вывод 7](#_Toc27602986)

[2. Конструкторская часть 8](#_Toc27602987)

[Вывод 12](#_Toc27602988)

[3. Технологическая часть 13](#_Toc27602989)

[Вывод 14](#_Toc27602990)

[4. Исследовательская часть 15](#_Toc27602991)

[Вывод 16](#_Toc27602992)

[Заключение 17](#_Toc27602993)

[Литература 18](#_Toc27602994)

# Введение

Поиск подстроки в строке — одна из простейших задач поиска информации. Применяется в виде встроенной функции в текстовых редакторах, СУБД, поисковых машинах, языках программирования и т. п. В задачах поиска традиционно принято обозначать шаблон поиска как needle (с англ. — «иголка»), а строку, в которой ведётся поиск — как haystack (с англ. — «стог сена»). Обычно через Σ обозначается алфавит, на котором проводится поиск.

## Цели и задачи

Целью данной работы является изучение способов поиска подстроки в строке с помощью алгоритмов Кнута-Мориса-Прата и Бойера-Мура. Для достижения поставленной цели требуется решить следующие задачи.

1. Изучить применение алгоритмов Кнута-Мориса-Прата и Бойера-Мура к задаче поиска подстроки в строке.
2. Математически описать решение задачи поиска подстроки в строке на основании алгоритмов Кнута-Мориса-Прата и Бойера-Мура.
3. Реализовать алгоритмы Кнута-Мориса-Прата и Бойера-Мура.
4. Сравнить алгоритмы Кнута-Мориса-Прата и Бойера-Мура по скорости работы.
5. Сделать выводы о применимости алгоритмов Кнута-Мориса-Прата и Бойера-Мура к решению задачи поиска подстроки в строке.

# 1. Аналитическая часть

В данном разделе будeт дано полное описание алгоритма Кнута-Морриса-Прата и алгоритма Бойера-Мура и их математические описания.

## 1.1 Обзор алгоритма Кнута-Мориса-Прата

В задачах поиска информации одной из важнейших задач является поиск точно заданной подстроки в строке. Примитивный алгоритм поиска подстроки в строке основан на переборе всех подстрок, длина которых равна длине шаблона поиска, и посимвольном сравнении таких подстрок с шаблоном поиска.

Алгоритм Кнута-Морриса-Пратта является одним из первых алгоритмов с линейной оценкой в худшем случае.

Обозначим через:

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

где означает длину слова *word*.

Префикс-функция строки – это длина наибольшего префикса строки , который не совпадает с этой строкой и одновременно является ее суффиксом. Проще говоря, это длина наиболее длинного начала строки, являющегося также и ее концом. Для строки *S* удобно представлять префикс функцию в виде вектора длиной . Можно рассматривать префикс-функцию длины , положив . Пример приведен на таблице 1.

Таблица 1

Пример префикс функции для строки «abcdabca»

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | a | b | c | d | a | b | c | a |
|  | 0 | 0 | 0 | 0 | 1 | 2 | 3 | 1 |

Наиболее полный алгоритм вычисления префикс-функции на псевдокоде показан на листинге 1.

Листинг 1. Псевдокод функции поиска префикса

|  |  |
| --- | --- |
|  | **алг** префикс\_функция (**арг** строка )  // Предположим, что  **Если** то    **Иначе**  **Если** , то  .  **Иначе**  Положить и перейти к шагу 3. |

Рассмотрим алгоритм Кнута-Морриса-Пратта. Пусть – образец, – строка, в которой ведется поиск. Рассмотрим сравнение строк на позиции *i*, то есть образец сопоставляется с частью строки . Предположим, первое несовпадение произошло между символами и , где . Обозначим . При сдвиге можно ожидать, что префикс *S* сойдется с каким-либо суффиксом строки *P*. Поскольку длина наиболее длинного префикса, являющегося одновременно суффиксом, есть префикс-функция от строки *S* для индекса *j*, приходим к следующему алгоритму.

1. Построить префикс-функцию образца *S*, обозначим ее *F*.
2. Положить .
3. Сравнить символы . Если символы равны, увеличить *k* на 1. Если при этом *k* стало равно длине образца, то вхождение образца *S* в строку *T* найдено, индекс вхождения равен . Алгоритм завершается. Пока , присвоим и переходим в начало шага 3.
4. Пока , увеличиваем на 1 и переходим к шагу 3.

## 1.2 Обзор алгоритма Бойера-Мура

### Введение

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

Алфавит обозначим буквой . Пусть . Предположим, что в процессе сравнения возникает несовпадение между символом шаблона и символом исходного текста при проверке на позиции *j*. Тогда:

|  |  |  |
| --- | --- | --- |
|  | , | (2) |

и

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

тогда символом шаблона не совпало (оставшийся “хвост” подстроки). В целом алгоритм можно описать с помощью трех основных идей.

### 1.2.1 Сканирование слева направо, сравнение справа налево

Совмещается начало текста (строки) и шаблона, проверка начинается с последнего символа шаблона. Если символы совпадают, производится сравнение предпоследнего символа шаблона и т. д. Если все символы шаблона совпали с наложенными символами строки, значит, подстрока найдена, и выполняется поиск следующего вхождения подстроки. Если же какой-то символ шаблона не совпадает с соответствующим символом строки, шаблон сдвигается на несколько символов вправо, и проверка снова начинается с последнего символа.

### 1.2.2 Эвристика стоп-символа

Эвристика стоп-символа присутствует в большинстве описаний алгоритма Бойера — Мура, включая оригинальную статью Бойера и Мура, но не является необходимой для достижения оценки времени работы. Предположим, что мы производим поиск слова «колокол». Первая же буква не совпала — «к» (назовём эту букву стоп-символом). Тогда можно сдвинуть шаблон вправо до последней его буквы «к», что показано в таблице 2.

Таблица 2

Поиск слова “колокол” в строке “к”

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Строка | \* | \* | \* | \* | \* | \* | к | \* | \* | \* | \* | \* | \* |
| Шаблон | к | о | л | о | к | о | л |  |  |  |  |  |  |
| След. шаг |  |  | к | о | л | о | к | о | л |  |  |  |  |

Если стоп-символ «к» оказался за другой буквой «к», эвристика стоп-символа не работает. В таких ситуациях может быть полезна третья идея алгоритма Бойера — Мура — эвристика совпавшего суффикса.

### 1.2.3 Эвристика совпавшего суффикса

Если при чтении шаблона справа налево совпал суффикс *S*, а символ *b*, стоящий перед *S* в шаблоне (т. е. шаблон имеет вид *PbS*), не совпал, то эвристика совпавшего суффикса сдвигает шаблон на наименьшее число позиций вправо так, чтобы строка *S* совпала с шаблоном, а символ, предшествующий в шаблоне данному совпадению *S*, отличался бы от *b* (если такой символ вообще есть). Для данного шаблона считается целочисленный массив , в котором равно минимальному числу , такому, что и для любого , для которого выполняется и .

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Строка | \* | \* | \* | \* | \* | \* | р | к | а | \* | \* | \* | \* | \* |  |
| Шаблон | с | к | а | л | к | а | л | к | а |  |  |  |  |  |  |
| След. шаг |  |  |  |  |  |  | с | к | а | л | к | а | л | к | а |

В данном случае совпал суффикс «ка», и шаблон сдвигается вправо до ближайшего «ка», перед которым нет буквы «л».

## Вывод

В данном разделе было приведено общее и математическое описание муравьиного алгоритма, приведены примеры.

# 2. Конструкторская часть

В данном разделе будут приведены схемы алгоритма Кнута-Морриса-Пратта и Бойера Мура на рисунках 1 - 4.

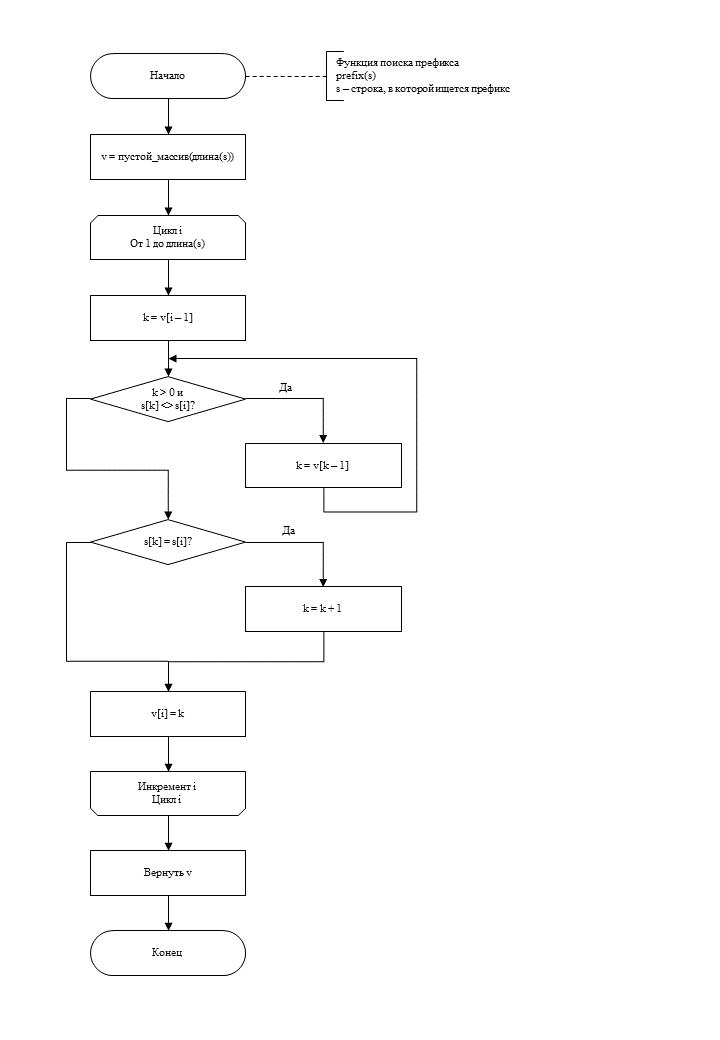


Рисунок 1. Функция поиска префикса в алгоритма Кнута-Морриса-Пратта

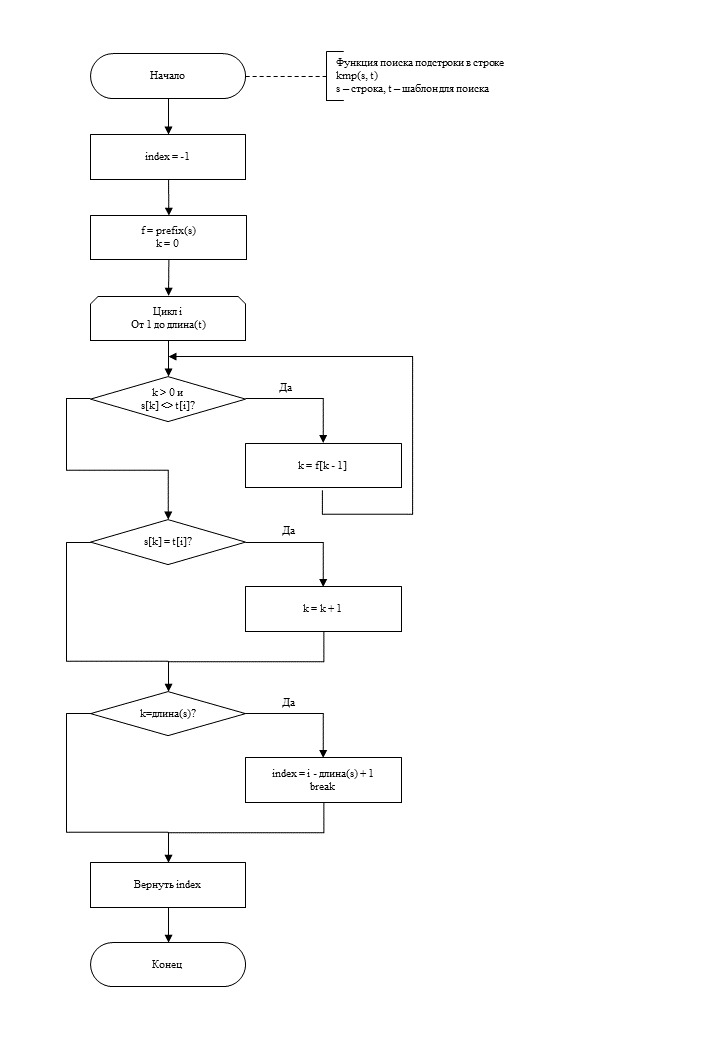


Рисунок 2. Алгоритм Кнута-Морриса-Пратта. Основная часть

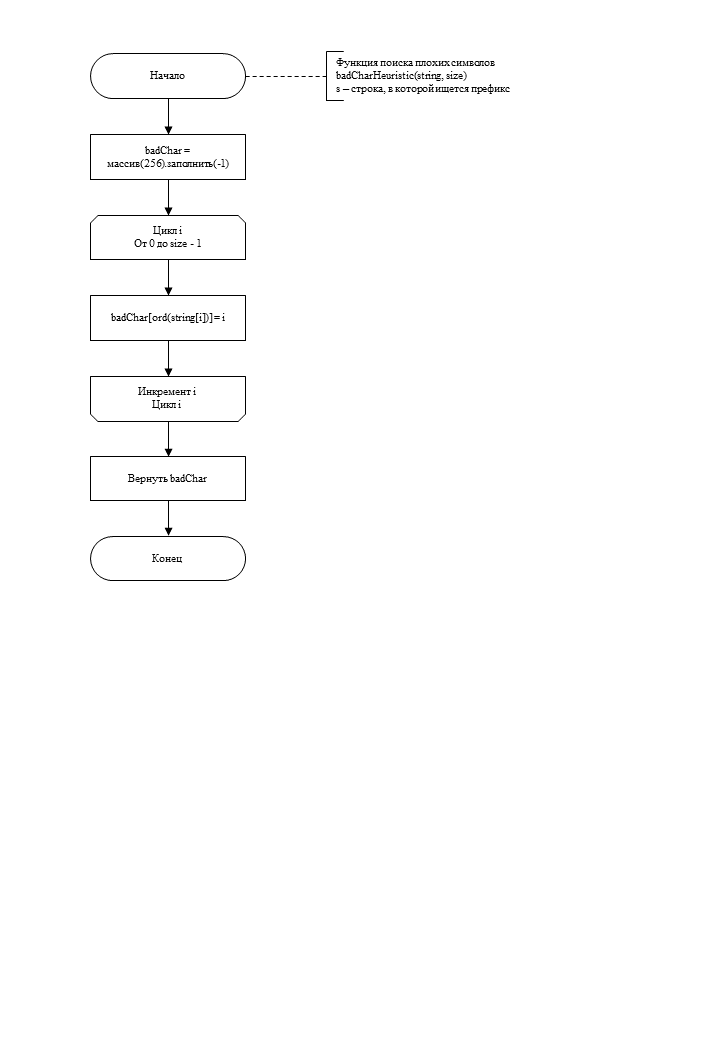


Рисунок 3. Функция поиска плохих символов алгоритма Бойера-Мура

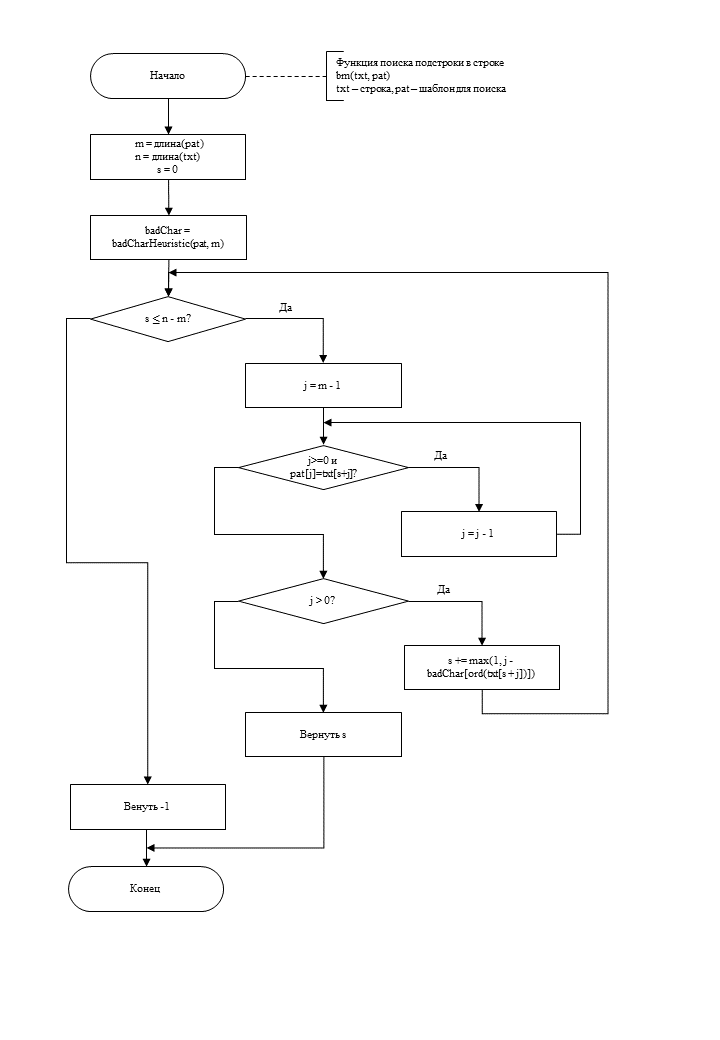


Рисунок 4. Алгоритма Бойера-Мура. Основная часть

## Вывод

В данном разделе были приведены схемы алгоритмов поиска подстроки в строке Кнута-Морриса-Пратта и Бойера Мура.

# 3. Технологическая часть

В качестве языка программирования был выбран Python [3], так как имеется большой опыт работы с ним. На листингах 2-3 будут приведены алгоритмы Кнута-Мориса-Прата и Бойера-Мура поиска подстроки в строке. На листинге 4 представлен метод тестирования данных алгоритмов.

Листинг 2. Алгоритм Кнута-Мориса-Прата

|  |  |
| --- | --- |
|  | # Алгоритм поиска префикса  **def** prefix(s):  v = [0] \* len(s)  **for** i in **range**(1, len(s)):  k = v[i - 1]  **while** k > 0 **and** s[k] != s[i]:  k = v[k - 1]  **if** s[k] == s[i]:  k = k + 1  v[i] = k  **return** v  # Алгоритм Кнута-Мориса-Прата  **def** kmp(s, t):  index = -1  f = prefix(s)  k = 0  **for** i in **range**(len(t)):  **while** k > 0 **and** s[k] != t[i]:  k = f[k - 1]  if s[k] == t[i]:  k = k + 1  if k == **len**(s):  index = i - **len**(s) + 1  break  **return** index |

Листинг 3. Алгоритма Бойера-Мура

|  |  |
| --- | --- |
|  | **def** badCharHeuristic(string, size):  badChar = [-1] \* 256  **for** i in **range**(size):  badChar[ord(string[i])] = i  **return** badChar  **def** search(txt, pat):  m = **len**(pat)  n = **len**(txt)  badChar = badCharHeuristic(pat, m)  s = 0  **while** s <= n - m:  j = m - 1  **while** j >= 0 **and** pat[j] == txt[s + j]:  j -= 1  **if** j < 0:  return s # Return only first entry  **else**:  s += max(1, j - badChar[ord(txt[s + j])])  **return** -1 |

Листинг 4. Функция тестирования методов поиска построки в строке

|  |  |
| --- | --- |
|  | **def** test():  tests = [  ["000110", "01", 2],  ["abcdef", "de", 3],  ["acgtagtcgtc", "gtcg", 5],  ["atgcatcg", "gta", -1]]  **for** t **in** tests:  status = **None**  haystack = t[0]  needle = t[1]  ground\_true = t[2]  response = **searching\_algorithm**(needle, haystack)  **if** response == ground\_true:  status = "OK"  **else**:  status = "ERROR"  **print**("Tested haystack '{0}' and needle '{1}',  ground true: {2}, found: {3}. {4}".format(  haystack, needle, ground\_true, response, status)) |

## Вывод

В данном разделе были приведены листинги алгоритмов Кнута-Мориса-Прата и Бойера на языке программирования Python, а также была приведена функция тестирования этих алгоритмов.

# 4. Исследовательская часть

Замеры времени проводились на 64-битной операционной системе Windows 10 и на x64 процессоре Inter Core i7 с 4 гб оперативной памяти для случайно сгенерированных слов размером от 10,000 до 1,000,000 с шагом 10,000. Случайным образом в слове *x* выбирался срез длиной 1,000 символом *y,* далее *x* и *y* подаются на вход алгоритмам поиска подстроки в строке. Алгоритмы тестировались на больших длинах строк, так как на строках в диапозона от 1 до 1,000 среднее время работы алгоритмов было 0,00017 секунд. Эксперимент с замерами времени работы алгоритмов в этом диапозон длин строк может оказаться некорректным, так как из-за быстрой работы на замеры времени могут влиять системные прерывания и другие процессы, усложнив поиск типа зависимости скорости работы от длины строк. На момент замера времени работало в среднем 76 активных процессов. Результаты эксперимента являются воспроизводимыми. Скорость работы обоих алгоритмов линейно зависит от количества символов в строке, в которой ведется поиск подстроки.

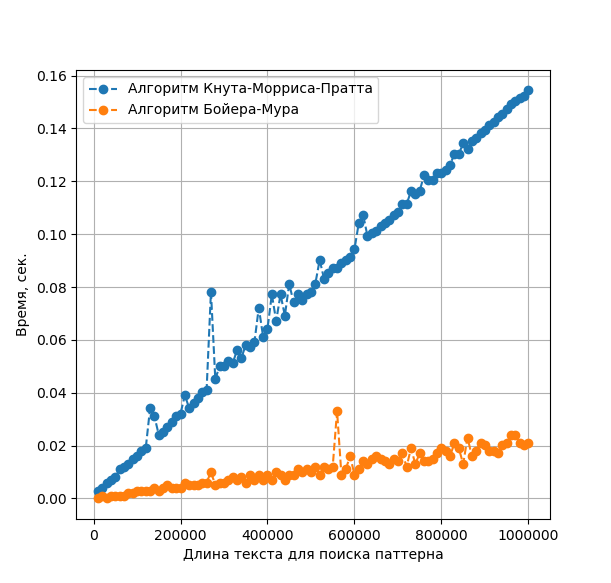


Рисунок 5. Сравнение времени работы в секундах алгоритмов Кнута-Морриса-Пратта и алгоритма Бойера-Мура

Тестирование на корректность работы проводилось в среде разработки PyCharm 2019.2.1 с помощью функции, представленной на листинге 4. На рисунках 1 – 2 представлены резульаты тестирования методов поиска подстроки в строке. Ground true в данном случае означает индекс первого вхождения подстроки *needle* в строку *haystack*.

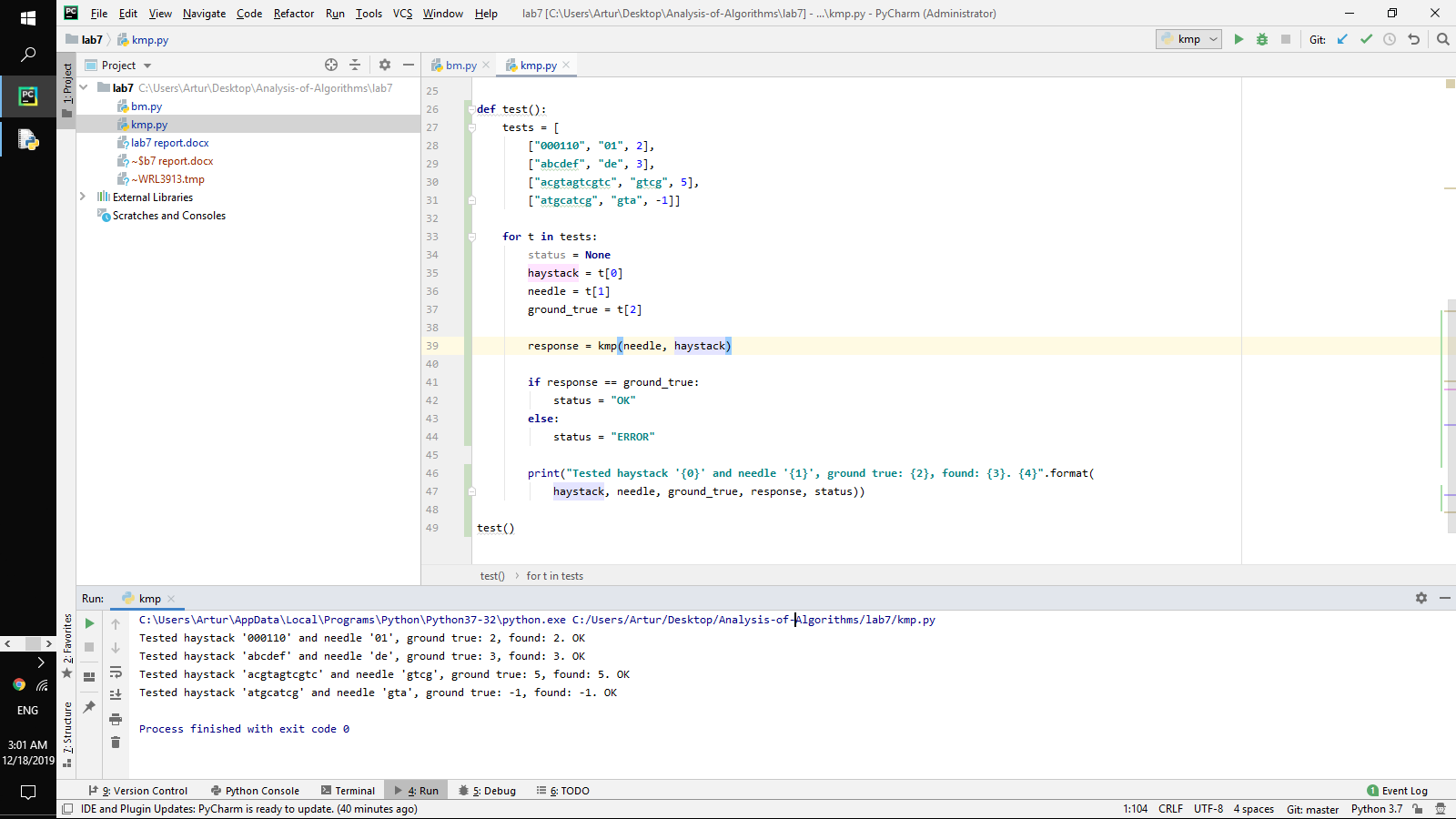


Рисунок 6. Тестирование алгоритма Бойера-Мура

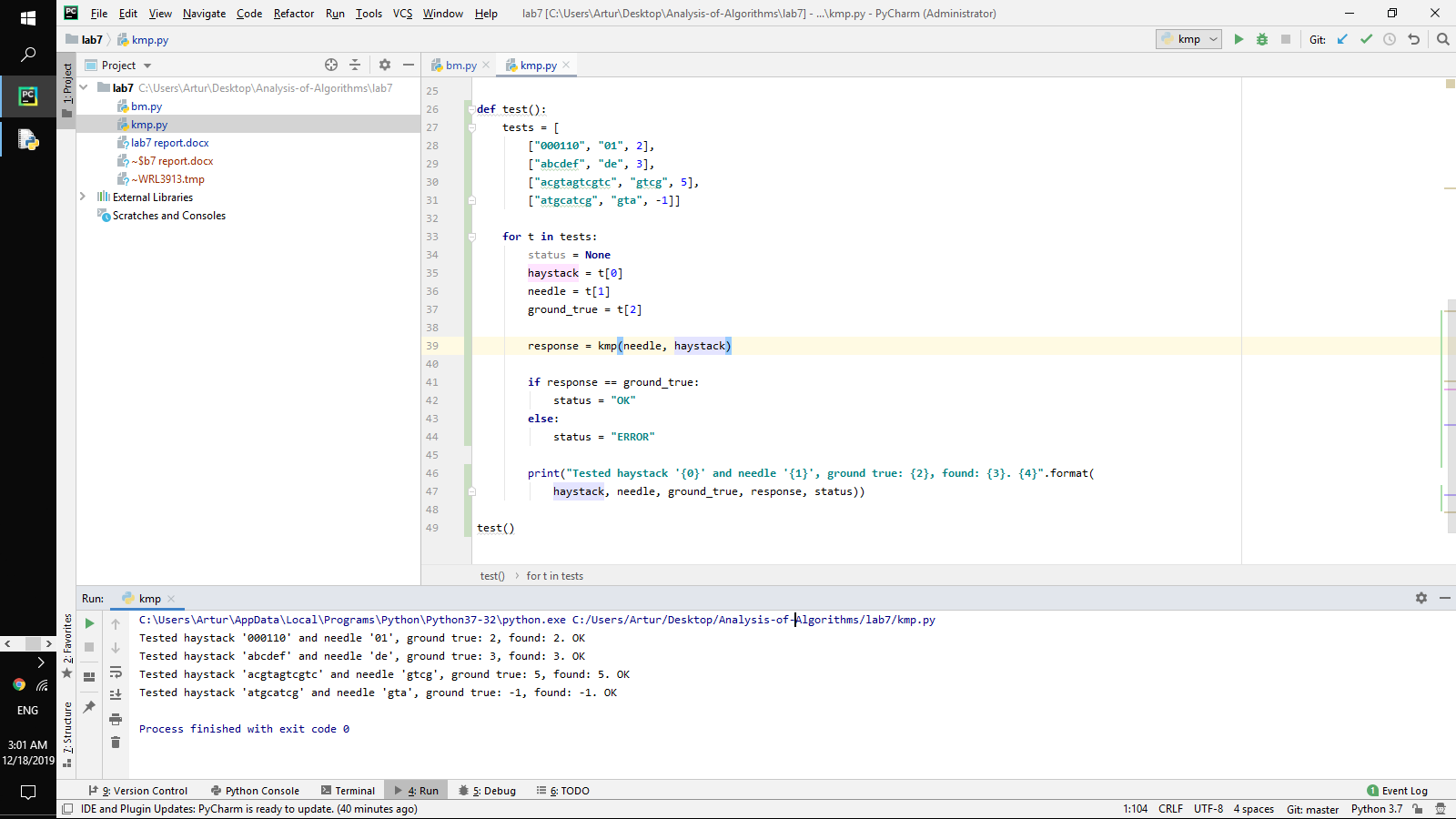


Рисунок 7. Тестирование алгоритма Кнута-Мориса-Пратта

## Вывод

В данном разделе было проведено экспериментальное сравнение алгоритмов Кнута-Мориса-Пратта и Бойера-Мура поиска построки в строке по времени работы, была найдена зависимость алгоритмов от длин строк и проведено тестирование на корректность работы. Опыт выявил, что алгоритм Бойера-Мура работает в среднем в 6 раз быстрее алгоритма Кнута-Мориса-Пратта.

# Заключение

В результате выполнения данной работы были рассмотрены способы решения задачи поиска подстроки в строке с помощью алгоритмов Кнута-Мориса-Прата и Бойера-Мура. Опыт показал, что рекомендуется использовать алгоритм Бойера-Мура, так как он показал большую производительность по времени в сравнении с алгоритмом Кнута-Мориса-Прата. Было изучено применение алгоритмов Кнута-Мориса-Прата и Бойера-Мура к задаче поиска подстроки в строке. Математически описано решение задачи поиска подстроки в строке на основании алгоритмов Кнута-Мориса-Прата и Бойера-Мура. Реализованы алгоритмы Кнута-Мориса-Прата и Бойера-Мура. Сравнены алгоритмы Кнута-Мориса-Прата и Бойера-Мура по скорости работы. Сделаны выводы о применимости алгоритмов Кнута-Мориса-Прата и Бойера-Мура к решению задачи поиска подстроки в строке.

# Литература

[1] A. Colorni, M. Dorigo et V. Maniezzo, Distributed Optimization by Ant Colonies, actes de la première conférence européenne sur la vie artificielle, Paris, France, Elsevier Publishing, 134—142, 1991.

[2] M. Dorigo, Optimization, Learning and Natural Algorithms, PhD thesis, Politecnico di Milano, Italie, 1992.