Отчет по python-task: поиск подстроки в строке

Постановка задачи

Разработать консольную утилиту, которая реализует 5 разных алгоритмов поиска подстроки в строке. Провести анализ и тестирование алгоритмов а также построить графики их работы.

План тестирования

1. Подготовка к тестированию
2. Генерация входных данных
3. Подготовка к измерению времени
4. Измерение
5. Обработка результатов
6. Написание отчета

Теоретическая оценка сложности алгоритмов

- $|\Sigma| = \sigma$— размер алфавита
- $|text| = t$ — длина текста, в котором ищем
- $|pattern| = p$ — длина паттерна (подстрока, которую ищем)

Название алгоритма	Среднее время	Худшее время	Препроцессинг	Дополнительная память
Наивный алгоритм (Brute Force)	$O(p(t-p))$	$O(t^2)$	Нет	$O(1)$

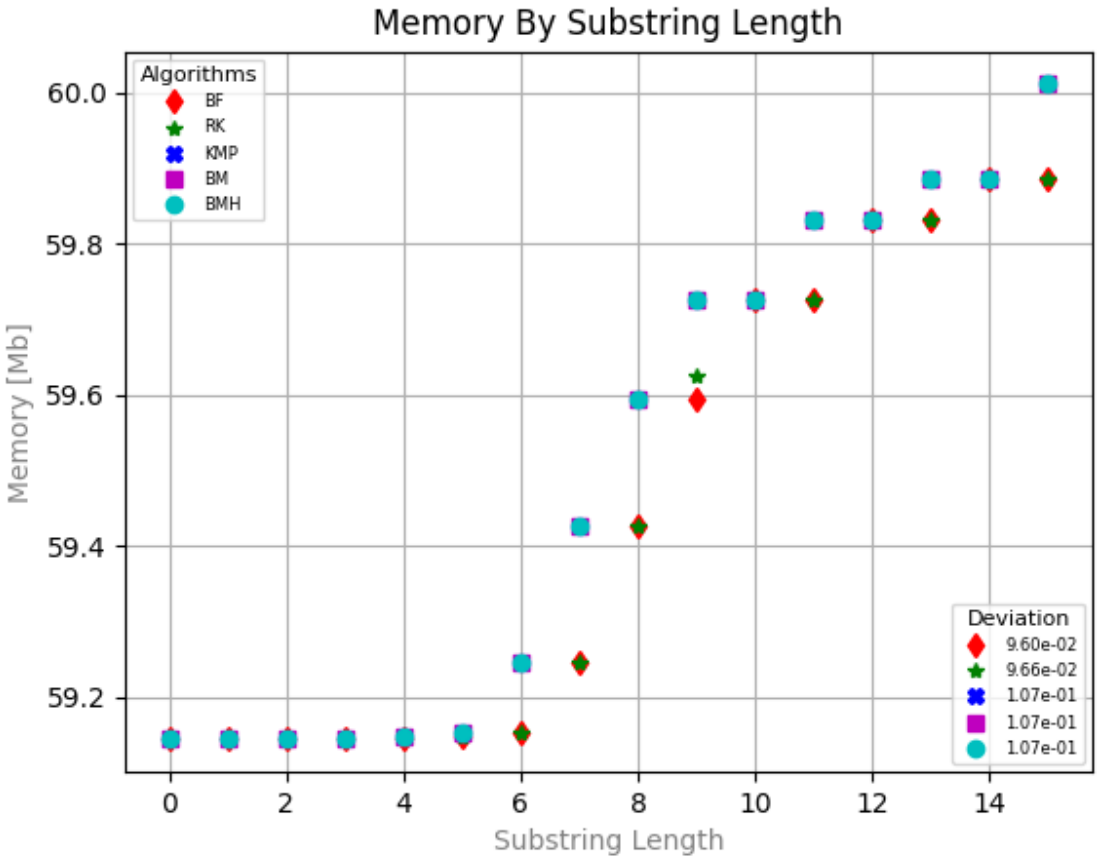
Название алгоритма	Среднее время	Худшее время	Препроцессинг	Дополнительная память
Алгоритм Кнута-Мориса-Пратта (Knuth-Morris-Pratt)	$O(p+t)$	$O(p+t)$	$O(p)$	$O(p)$
Алгоритм Рабина-Карпа (Karp-Rabin)	$O(p+t)$	$O(pt)$	$O(p)$	$O(1)$
Алгоритм Бойера-Мура (Boyer-Moore)	$O(t)$	$O(pt)$	$O(p+\sigma)$	$O(p+\sigma)$
Алгоритм Бойера-Мура-Хорспула (Boyer-Moore-Horspul)	$O(t)$	$O(pt)$	$O(p+\sigma)$	$O(p+\sigma)$

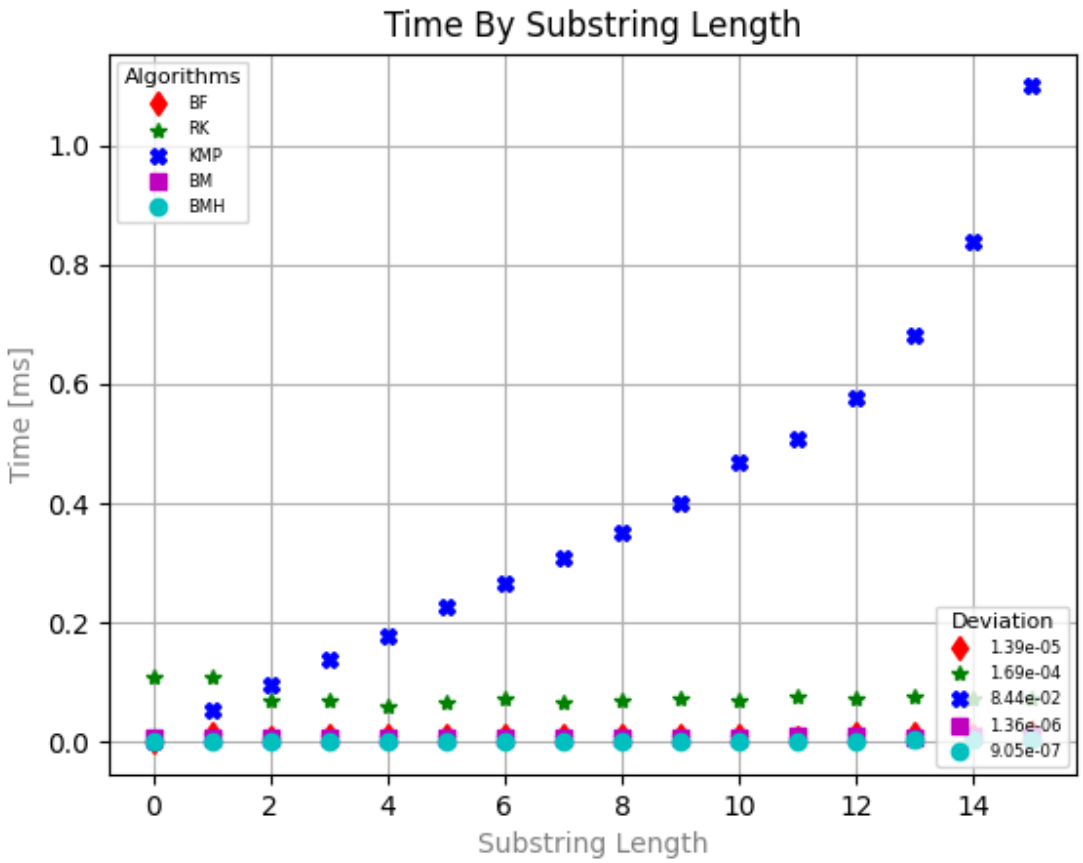
Возможные случаи входных данных для тестирования

1. **Случайный набор данных** - подстрока находится в случайных местах текста. Будем считать, что где-то в середине.
2. **Лучший случай** - подстрока находится в начале текста
3. **Худший случай** - подстрока находится в конце текста

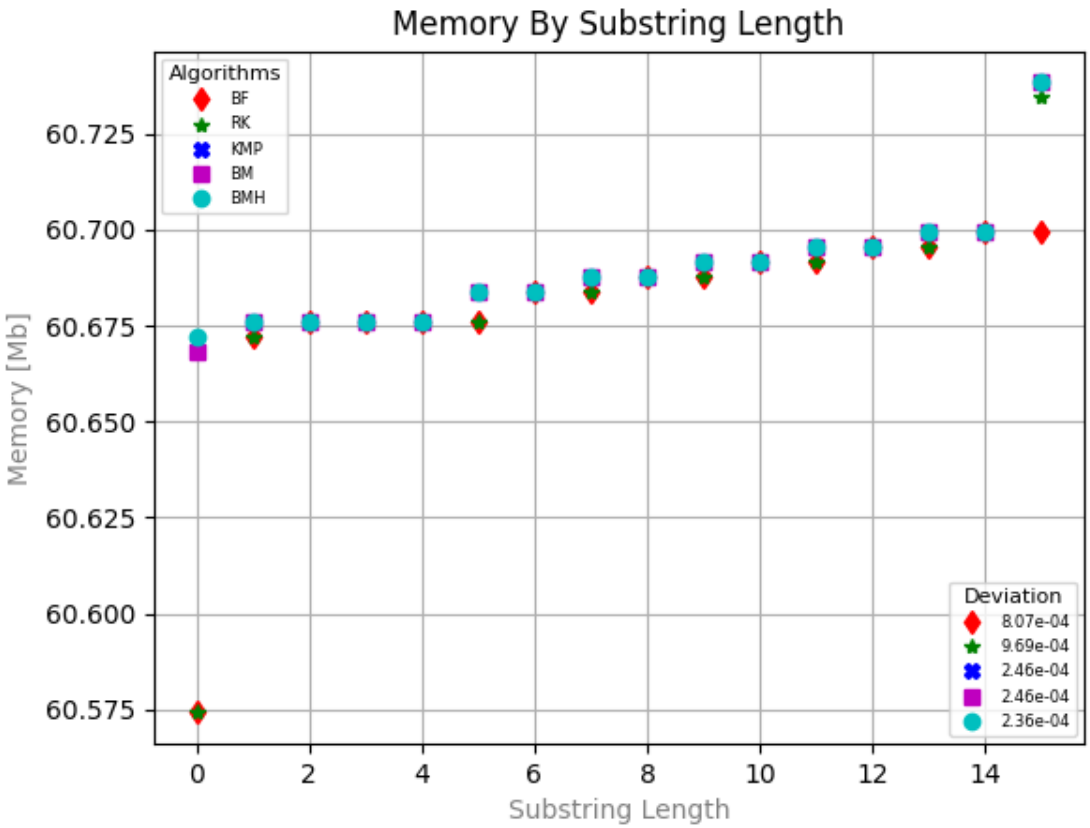
Результаты тестирования

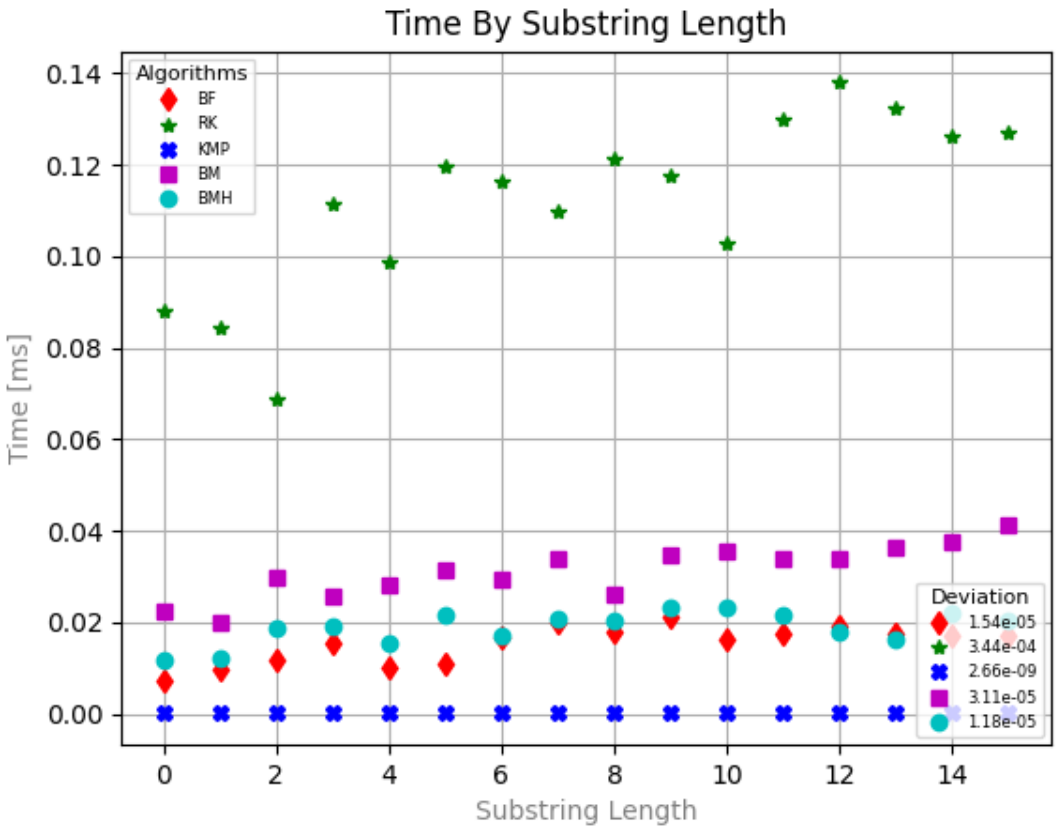
На худшем наборе данных


Memory By Substring Length
Algorithms
BF
RK
KMP
BM
BMH
Deviation
9.60e-02
9.66e-02
1.07e-01
1.07e-01
1.07e-01
Memory [Mb]
Substring Length
60.0
59.8
59.6
59.4
59.2
0
2
4
6
8
10
12
14

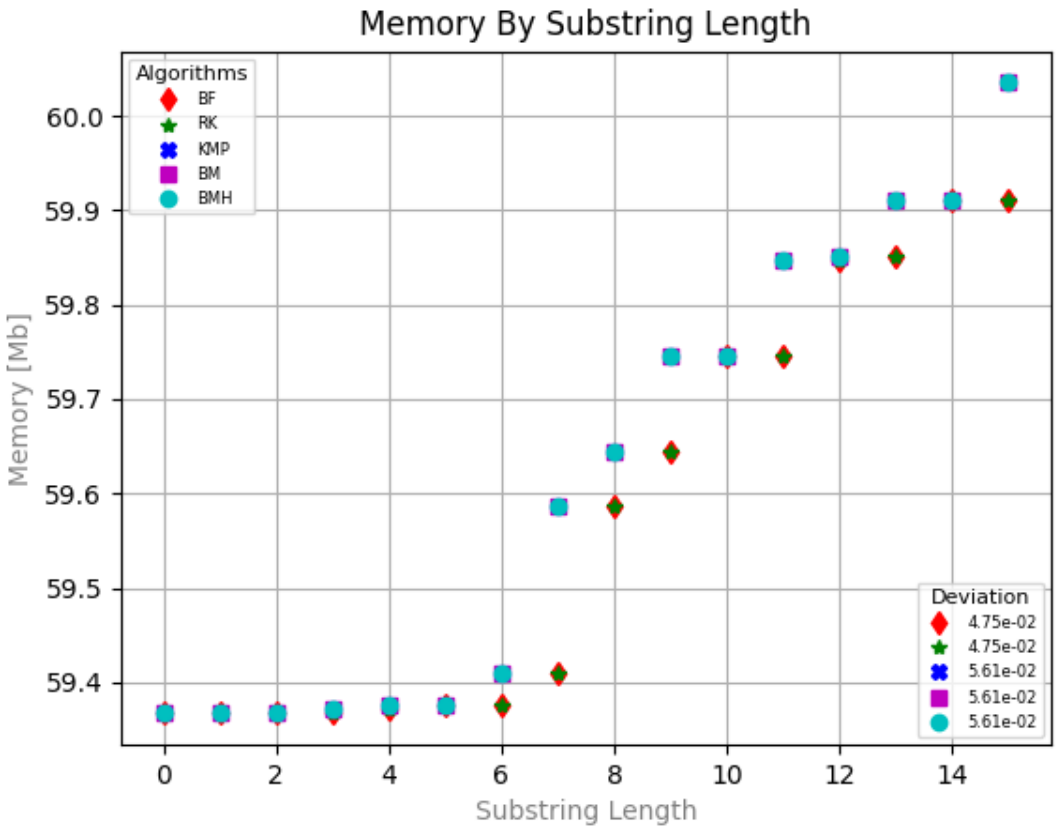


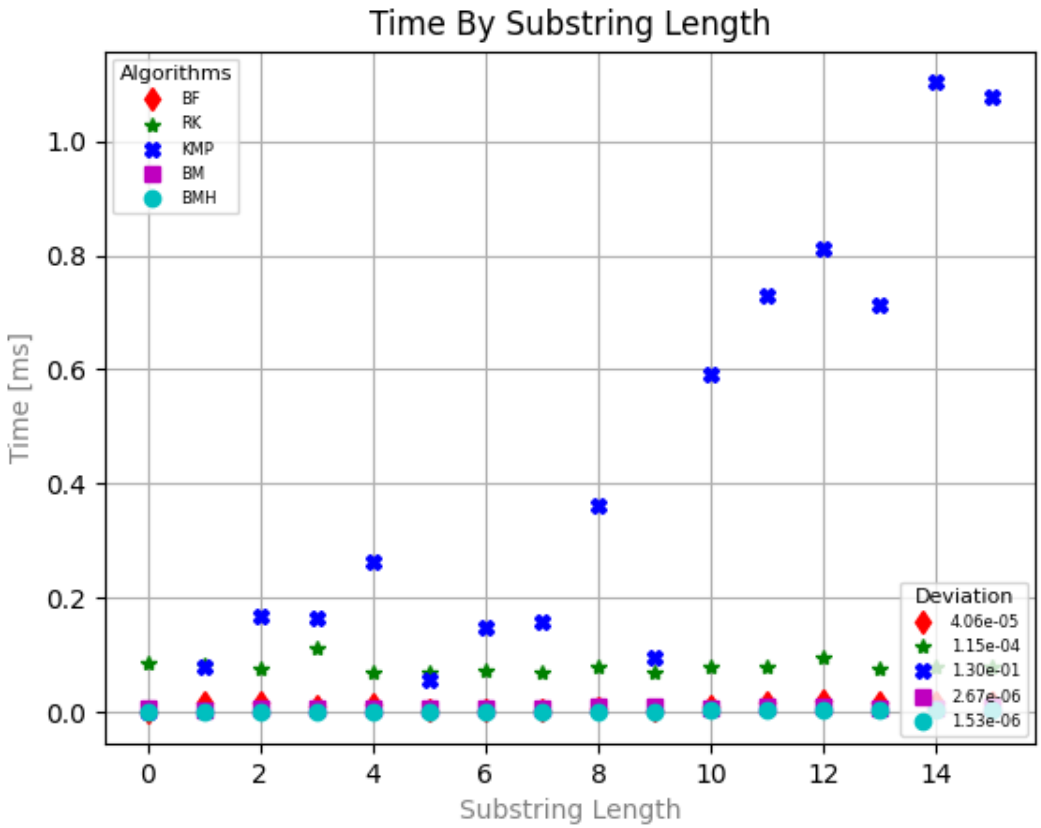
На лучшем наборе данных:


Memory By Substring Length
Algorithms
BF
RK
KMP
BM
BMH
Deviation
8.07e-04
9.69e-04
2.46e-04
2.46e-04
2.36e-04
Memory [Mb]
60.725
60.700
60.675
60.650
60.625
60.600
60.575
0
2
4
6
8
10
12
14
Substring Length



На случайном наборе данных:


Memory By Substring Length
Algorithms
BF
RK
KMP
BM
BMH
Deviation
4.75e-02
4.75e-02
5.61e-02
5.61e-02
5.61e-02
Memory [Mb]
60.0
59.9
59.8
59.7
59.6
59.5
59.4
0
2
4
6
8
10
12
14
Substring Length



Каждое деление по оси x - 1000 символов.

Квадратичное отклонение от среднего считалось по формуле:

$$D = \frac{1}{N} \sum_{x \in arr} (x - M)^2$$

Где:

- arr - массив полученных данных
- M - среднее массива
- N - количество элементов массива

Выводы

Исследуя графики можно сказать, что алгоритм Кнута-Морриса-Пратта самый медленный, а алгоритм Бойера-Мура-Хорспула самый быстрый. Имеет смысл

смотреть со второго прогона (с цифры 1), так как в первом прогоне строка слишком короткая и может встретиться в тексте раньше. Если говорить про использование памяти, все алгоритмы используют её примерно одинаково.

Щербакова Полина, Головачев Георгий, ФТ-203