Федеральное государственное автономное образовательное учреждение высшего образования
«Санкт-Петербургский национальный исследовательский
университет информационных технологий, механики и оптики»

Образовательный центр Инфохимии

Дисциплина: Алгоритмы и структуры данных / Algorithms and Data Structures

Курсовая работа по теме «Алгоритмы поиска подстрок»

Выполнила: Конаныхина Антонина
Группа: O4150
Вариант: 10
Преподаватель: Гунько Татьяна Александровна

Санкт-Петербург, 2025г

Оглавление

Цель

Изучить алгоритмы поиска подстрок: алгоритм Рабина-Карпа и алгоритм Кнута-Морриса-Пратта. Среди многих других программ алгоритмы сопоставления строк ищут определенные шаблоны в последовательностях ДНК, поисковые системы в Интернете также используют их для поиска веб-страниц, соответствующих запросам.

Задача

1) Взять любую главу из книги или поэмы.
2) Реализовать алгоритм поиска подстроки «в лоб» из главы 32.1 «Алгоритмы. Построение и анализ» Кормена и алгоритм Рабина-Карпа для поиска данного шаблона в текстеЮ запустить на выбранном тексте.
3) Проделать то же самое с алгоритмом Кнута-Морриса-Пратта.
4) Проанализировать и сравнить полученные результаты.

   Формат входных данных: подстрока (шаблон) P и текст T.
   Задача: найти все вхождения строки P в строку T как подстроки.
   Формат выходных данных: в первой строке вывести число вхождений строки P в строку T. Во второй строке вывести в порядке возрастания номера символов в строке T, с которых начинаются вхождения строки P.

Выполнение

Алгоритмы поиска подстроки в строке – класс алгоритмов над строками, которые позволяют найти паттерн (pattern/P) в тексте (text/T).

В данной работе будут рассматриваться три алгоритма:

1. Наивный алгоритм поиска подстроки;
2. Алгоритм Рабина-Карпа;
3. Алгоритм Кнута-Морриса-Пратта;

Полное время работы каждого из этих алгоритмов равно сумме времени предварительной обработки входного текста и времени сравнения шаблона(паттерна) с исходным текстом. Все рассматриваемые алгоритмы пользуются только одним поисковым шаблоном (алгоритм Рабина-Карпа можно модифицировать для поиска нескольких паттернов), имеют прямой порядок сравнения и хороши для работы по поиску небольших паттернов.

Алгоритм	**Время предварительной обработки**	**Время сравнения**
Наивный	0	O((n – m)*m)
Рабина-Карпа	Θ (m)	O((n – m)*m)
Кнута-Морриса-Пратта	Θ (m)	Θ (n)

Таблица 1. Времена предварительной обработки и сравнения алгоритмов. N - длина исходного текста, m - длина искомой подстроки.

Наивный алгоритм

Данный алгоритм является алгоритмом «в лоб» и выполняет поиск всех допустимых сдвигов с помощью одного цикла, в котором проверяется условие соответствия шаблона с рассматриваемой подстрокой. Происходит «скольжение» шаблона по массиву, в процессе которого отмечается, для каких сдвигов все символы шаблона равны соответствующим символам текста.

Худшее время выполнения: $O(n^2)$. Если p достаточно мало по сравнению с n, то асимптотика будет близкой к $O(n)$, что позволяет эффективно использовать данный алгоритм на практике в случаях, когда паттерн много меньше текста.

Псевдокод наивного алгоритма:

```
vector<int> naiveStringMatcher(t : string, p : string):
   int n = t.length
   int m = p.length
   vector<int> ans
   for i = 0 to n - m
      if t[i .. i + m - 1] == p
         ans.push_back(i)
   return ans
```

Реализация наивного алгоритма поиска подстрок

Реализуем наивный алгоритм на языке C++:

```cpp
vector<int> naive_string_matcher(const string& text, const string& pattern)
{
    vector<int> matches;
    int n = text.size();
    int m = pattern.size();

    if (m == 0 || n < m) return matches;

    for (int i = 0; i <= n - m; ++i)
      {
        int j;
        for (j = 0; j < m; ++j)
{
            if (text[i + j] != pattern[j])
                break;
        }
        if (j == m)
            matches.push_back(i);
    }
```

```
    return matches;
}
```

Алгоритм Рабина-Карпа

Алгоритм Рабина-Карпа решает задачу поиска подстроки в тексте, используя хэширование для эффективного сравнения шаблона с подстроками. Основная идея заключается в предварительном вычислении хэша для шаблона и последовательном вычислении хэшей для всех возможных подстрок текста. При совпадении хэшей выполняется дополнительная проверка символов для исключения ложных совпадений (коллизий). Для ускорения процесса применяется «скользящий хэш», который позволяет быстро пересчитывать хэш для следующей подстроки на основе предыдущего значения.

Худшее время выполнения: O(m*n), где n — длина текста, m — длина шаблона. Однако при использовании эффективной хэш-функции и оптимизации коллизий средняя сложность стремится к O(n+m).

Псевдокод алгоритма Рабина-Карпа:

```
vector<int> rabinKarp (s : string, w : string):
   vector<int> answer
   int n = s.length
   int m = w.length
   int hashS = hash(s[0..m - 1])
   int hashW = hash(w[0..m - 1])
   for i = 0 to n - m
        if hashS == hashW
             answer.add(i)
        hashS = (p * hashS - p^m * hash(s[i]) + hash(s[i + m])) mod r // r — некоторое большое число, p — некоторое просто число
   return answer
```

Реализация алгоритма Рабина-Карпа

Реализуем алгоритм Рабина-Карпа на языке C++:

```
vector<int> rabin_karp_matcher(const string& text, const string& pattern)
{
    vector<int> matches;
    int n = text.size();
    int m = pattern.size();
    if (m == 0 || n < m) return matches;

    const int base = 256;
    const int q = 1e9 + 7;
```

```
    long long h = 1;
    for (int i = 0; i < m - 1; i++)
        h = (h * base) % q;

    long long hash_pat = 0, hash_text = 0;
    for (int i = 0; i < m; i++) {
        hash_pat = (hash_pat * base + static_cast<unsigned char>(pattern[i])) % q;
        hash_text = (hash_text * base + static_cast<unsigned char>(text[i])) % q;
    }

    for (int i = 0; i <= n - m; i++)
    {
        if (hash_pat == hash_text)
        {
            bool match = true;
            for (int j = 0; j < m; j++)
            {
                if (static_cast<unsigned char>(text[i + j]) != static_cast<unsigned
char>(pattern[j]))
                {
                    match = false;
                    break;
                }
            }
            if (match)
                matches.push_back(i);
        }

        if (i < n - m) {
            hash_text = (base * (hash_text - static_cast<unsigned char>(text[i]) * h)
+ static_cast<unsigned char>(text[i + m]));
            hash_text %= q;
            if (hash_text < 0)
                hash_text += q;
        }
    }

    return matches;
}
```

Алгоритм Кнута-Морриса-Пратта

Данный алгоритм использует префикс-функцию, в основе которой лежит вычисление функции $\pi(s)$ – функции, отображающей строку в число, которое является длиной **наибольшего** суффикса строки *s*, который совпадает с ее префиксом той же длины, но при этом не совпадает со всей строкой *s*.

Пример вычисления функций $\pi(s)$ для строки $P(\text{"ababa"})$:

Префикс	$\pi(s)$
a	0
ab	0
aba	1
abab	2
ababa	3

$P(\text{"ababa"}) = [0, 0, 1, 2, 3]$

Префикс-функция – это массив чисел π для всех префиксов строки (количество префиксов строки совпадает с ее длиной, то есть размер массива равен длине строки).

Сам же алгоритм использует вычисленную префикс-функцию следующим образом: искомая подстрока склеивается с исходной строкой с добавлением разделяющего символа, который гарантированно не встречается ни в строке, ни в подстроке. Для построенной склеенной строки считается префикс-функция. Так как мы берём все подстроки полученной склейки, при наличии шаблона в исходной строке рано или поздно будет найдено это вхождение.

Пример:

Исходная строка: *cabcc*

Искомая строка: *abc*

Склейка: *abc#cabcc*

Полученная префикс функция: $P(\text{"abc\#cabcc"}) = [\pi(\text{a}),\ \pi(\text{ab}),\ \pi(\text{abc}),\ \pi(\text{abc\#}),\ \pi(\text{abc\#c}),\ \pi(\text{abc\#ca}),\ \pi(\text{abc\#cab}),\ \pi(\text{abc\#cabc}),\ \pi(\text{abc\#cabcc})] = [0, 0, 0, 0, 1, 2, 3, 0]$

Пусть длина искомой подстроки равна *k*. Найден такой префикс склеенной строки $s[0 \dots i]$, для которого $\pi(s[0 \dots i])$= k. То есть найден суффикс длины *k*, совпавший с префиксом длины *k*. Префикс длины *k* и есть искомая подстрока.

Суффикс длины *k* гарантированно полностью лежит в исходной строке, так как это гарантирует добавленный служебный символ #. Благодаря нему префиксы и суффиксы склеенной строки не пересекаются (за исключением самой строки).

Псевдокод алгоритма Рабина-Карпа:

```
int[] kmp(string P, string T):
   int pl = P.length
   int tl = T.length
   int[] answer
   int[] p = prefixFunction(P + "#" + T)
   int count = 0
   for i = 0 .. tl - 1
      if p[pl + i + 1] == pl
         answer[count++] = i - pl
   return answer
```

Реализация алгоритма Кнута-Морриса-Пратта

Реализуем алгоритм Кнута-Морриса-Пратта на языке C++:

```cpp
vector<int> prefix_function(string s)
{
    int n = s.size();
    vector<int> pi(n, 0);
    int k = 0;

    for (int i = 1; i < n; i++)
    {
        while (k > 0 && s[i] != s[k])
            k = pi[k - 1];
        if (s[i] == s[k])
            k++;
        pi[i] = k;
    }
    return pi;
}

vector<int> KMP_matcher(string& text, string& pattern)
{
    vector<int> matches;
    int n = text.size();
    int m = pattern.size();

    if (m == 0 || n < m) return matches;

    vector<int> prefix = prefix_function(pattern);

    int j = 0;
    for (int i = 0; i < n; ++i)
    {
        while (j > 0 && text[i] != pattern[j])
            j = prefix[j-1];

        if (text[i] == pattern[j])
            ++j;

        if (j == m) {
            matches.push_back(i - m + 1);
            j = prefix[j-1];
        }
    }

    return matches;
}
```

Тестирование алгоритмов

Тестировать алгоритмы будем на следующем наборе тестов с последующим промежуточным анализом полученных результатов:

Набор тестов 1: проверка поиска числовых значений.

Входные данные:

- 17
- 84
- 51
- 421343454

1 тест:

```
Enter the substring you are looking for: 17
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 0
1
90
Process returned 0 (0x0)   execution time : 3.018 s
Press any key to continue.
```

```
Enter the substring you are looking for: 17
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 1
1
90
Process returned 0 (0x0)   execution time : 4.755 s
Press any key to continue.
```

```
Enter the substring you are looking for: 17
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 2
1
90
Process returned 0 (0x0)   execution time : 3.246 s
Press any key to continue.
```

2 тест:

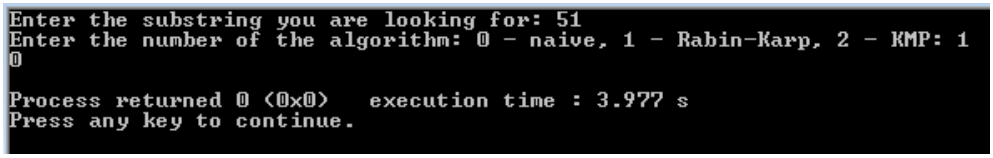
```
Enter the substring you are looking for: 84
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 0
1
95
Process returned 0 (0x0)   execution time : 3.807 s
Press any key to continue.
```

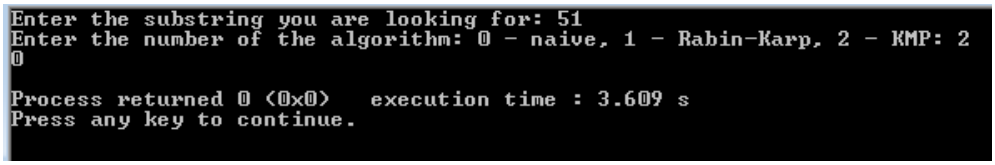
```
Enter the substring you are looking for: 84
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 1
1
95
Process returned 0 (0x0)   execution time : 4.283 s
Press any key to continue.
```

```
Enter the substring you are looking for: 84
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 2
1
95
Process returned 0 (0x0)   execution time : 4.803 s
Press any key to continue.
```

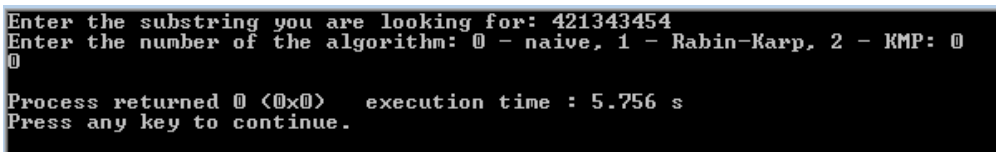
3 тест:

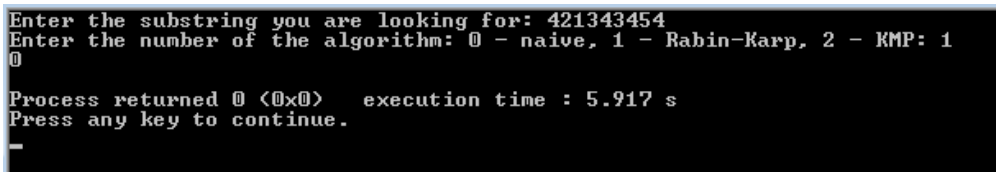
```
Enter the substring you are looking for: 51
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 0
0

Process returned 0 (0x0)   execution time : 3.274 s
Press any key to continue.
```

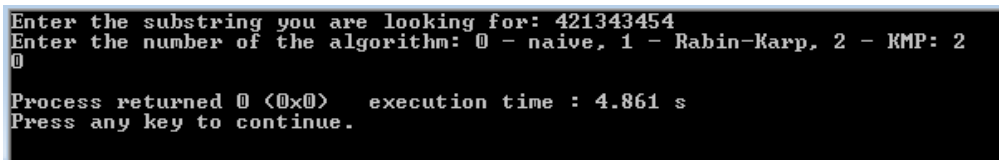




4 тест:







Выводу по первому набору тестов:

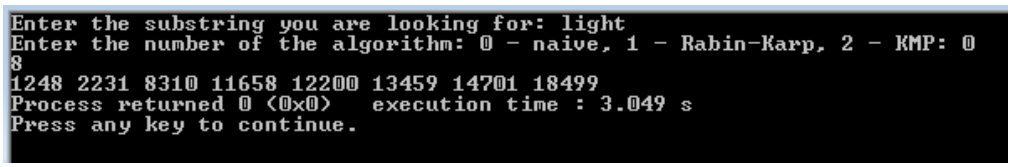
Оценить время выполнения на маленьких паттернах не представляется возможным (для операционной системы Windows, для Linux можно посмотреть в тиках), они все работают достаточно быстро даже на поиск больших подстрок ($1.7*10^{-15}$ мс).

Набор тестов 2: проверка поиска символьных значений.

Входные данные:

- light
- he
- I failed them
- Adolin
- Lord

1 тест:



```
Enter the substring you are looking for: light
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 1
8
1248 2231 8310 11658 12200 13459 14701 18499
Process returned 0 (0x0)   execution time : 4.826 s
Press any key to continue.
```

```
Enter the substring you are looking for: light
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 2
8
1248 2231 8310 11658 12200 13459 14701 18499
Process returned 0 (0x0)   execution time : 3.917 s
Press any key to continue.
```

2 тест:

```
Enter the substring you are looking for: he
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 0
392
36 143 195 237 249 257 275 331 344 363 370 438 462 537 540 612 671 711 725 738 7
50 755 820 834 855 883 921 937 962 971 992 998 1012 1065 1245 1271 1319 1342 135
5 1369 1469 1510 1581 1607 1776 1784 1799 1807 1881 1889 1936 2082 2103 2114 217
2 2199 2269 2283 2294 2330 2361 2425 2439 2465 2487 2541 2656 2673 2726 2759 280
1 2843 2874 2945 2973 2978 2996 3063 3190 3247 3328 3355 3415 3432 3525 3602 369
3 3720 3727 3745 3769 3795 3805 3940 3953 3970 4016 4026 4046 4131 4149 4283 432
8 4338 4386 4452 4569 4606 4746 4751 4763 4846 4882 4901 4918 4954 4964 4970 498
6 5011 5017 5057 5081 5141 5223 5343 5371 5454 5512 5555 5611 5654 5701 5726 573
7 5749 5760 5764 5788 5814 5840 5867 5901 5949 6012 6357 6362 6415 6482 6534 661
7 6640 6661 6711 6819 6980 7023 7058 7104 7146 7233 7253 7265 7355 7371 7519 761
5 7653 7723 7728 7740 7843 7887 7934 7944 7986 8029 8087 8116 8136 8163 8174 818
8 8229 8270 8391 8399 8407 8466 8477 8480 8488 8608 8633 8649 8660 8671 8720 880
4 8892 8918 8932 8937 8954 8972 9034 9068 9074 9101 9192 9292 9399 9457 9502 951
9 9553 9654 9710 9727 9731 9746 9805 9842 9941 9947 9976 10038 10059 10080 10120
 10140 10148 10169 10189 10243 10277 10307 10340 10387 10393 10453 10477 10521 1
0537 10569 10622 10677 10789 10927 10936 10955 11078 11084 11125 11232 11251 112
93 11318 11349 11374 11456 11474 11499 11525 11601 11684 11708 11735 11756 11768
 11790 11795 11837 11856 11913 11967 11997 12008 12096 12151 12261 12358 12366 1
2382 12508 12524 12556 12760 12838 12896 12994 13022 13076 13269 13289 13344 134
28 13516 13544 13648 13760 13890 13895 14043 14055 14091 14120 14684 14834 15011
 15158 15200 15298 15321 15352 15365 15402 15428 15444 15466 15495 15537 15591 1
5619 15651 15658 15672 15755 15776 15791 15856 15883 15953 16019 16045 16110 162
30 16276 16285 16318 16399 16496 16534 16546 16614 16744 16790 16861 16894 17051
 17058 17070 17158 17182 17238 17257 17282 17642 17654 17741 17821 17844 17877 1
7894 17910 17949 17965 18052 18060 18098 18113 18143 18163 18210 18241 18263 183
16 18336 18355 18420 18521 18541 18554 18574 18675 18698 18716 18731 18754 18767
 18823 18887
Process returned 0 (0x0)   execution time : 3.977 s
Press any key to continue.
```

```
Enter the substring you are looking for: he
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 1
392
36 143 195 237 249 257 275 331 344 363 370 438 462 537 540 612 671 711 725 738 7
50 755 820 834 855 883 921 937 962 971 992 998 1012 1065 1245 1271 1319 1342 135
5 1369 1469 1510 1581 1607 1776 1784 1799 1807 1881 1889 1936 2082 2103 2114 217
2 2199 2269 2283 2294 2330 2361 2425 2439 2465 2487 2541 2656 2673 2726 2759 280
1 2843 2874 2945 2973 2978 2996 3063 3190 3247 3328 3355 3415 3432 3525 3602 369
3 3720 3727 3745 3769 3795 3805 3940 3953 3970 4016 4026 4046 4131 4149 4283 432
8 4338 4386 4452 4569 4606 4746 4751 4763 4846 4882 4901 4918 4954 4964 4970 498
6 5011 5017 5057 5081 5141 5223 5343 5371 5454 5512 5555 5611 5654 5701 5726 573
7 5749 5760 5764 5788 5814 5840 5867 5901 5949 6012 6357 6362 6415 6482 6534 661
7 6640 6661 6711 6819 6980 7023 7058 7104 7146 7233 7253 7265 7355 7371 7519 761
5 7653 7723 7728 7740 7843 7887 7934 7944 7986 8029 8087 8116 8136 8163 8174 818
8 8229 8270 8391 8399 8407 8466 8477 8480 8488 8608 8633 8649 8660 8671 8720 880
4 8892 8918 8932 8937 8954 8972 9034 9068 9074 9101 9192 9292 9399 9457 9502 951
9 9553 9654 9710 9727 9731 9746 9805 9842 9941 9947 9976 10038 10059 10080 10120
 10140 10148 10169 10189 10243 10277 10307 10340 10387 10393 10453 10477 10521 1
0537 10569 10622 10677 10789 10927 10936 10955 11078 11084 11125 11232 11251 112
93 11318 11349 11374 11456 11474 11499 11525 11601 11684 11708 11735 11756 11768
 11790 11795 11837 11856 11913 11967 11997 12008 12096 12151 12261 12358 12366 1
2382 12508 12524 12556 12760 12838 12896 12994 13022 13076 13269 13289 13344 134
28 13516 13544 13648 13760 13890 13895 14043 14055 14091 14120 14684 14834 15011
 15158 15200 15298 15321 15352 15365 15402 15428 15444 15466 15495 15537 15591 1
5619 15651 15658 15672 15755 15776 15791 15856 15883 15953 16019 16045 16110 162
30 16276 16285 16318 16399 16496 16534 16546 16614 16744 16790 16861 16894 17051
 17058 17070 17158 17182 17238 17257 17282 17642 17654 17741 17821 17844 17877 1
7894 17910 17949 17965 18052 18060 18098 18113 18143 18163 18210 18241 18263 183
16 18336 18355 18420 18521 18541 18554 18574 18675 18698 18716 18731 18754 18767
 18823 18887
Process returned 0 (0x0)   execution time : 2.501 s
Press any key to continue.
```

```
Enter the substring you are looking for: he
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 2
392
36 143 195 237 249 257 275 331 344 363 370 438 462 537 540 612 671 711 725 738 7
50 755 820 834 855 883 921 937 962 971 992 998 1012 1065 1245 1271 1319 1342 135
5 1369 1469 1510 1581 1607 1776 1784 1799 1807 1881 1889 1936 2082 2103 2114 217
2 2199 2269 2283 2294 2330 2361 2425 2439 2465 2487 2541 2656 2673 2726 2759 280
1 2843 2874 2945 2973 2978 2996 3063 3190 3247 3328 3355 3415 3432 3525 3602 369
3 3720 3727 3745 3769 3795 3805 3940 3953 3970 4016 4026 4046 4131 4149 4283 432
8 4338 4386 4452 4569 4606 4746 4751 4763 4846 4882 4901 4918 4954 4964 4970 498
6 5011 5017 5057 5081 5141 5223 5343 5371 5454 5512 5555 5611 5654 5701 5726 573
7 5749 5760 5764 5788 5814 5840 5867 5901 5949 6012 6357 6362 6415 6482 6534 661
7 6640 6661 6711 6819 6980 7023 7058 7104 7146 7233 7253 7265 7355 7371 7519 761
5 7653 7723 7728 7740 7843 7887 7934 7944 7986 8029 8087 8116 8136 8163 8174 818
8 8229 8270 8391 8399 8407 8466 8477 8480 8488 8608 8633 8649 8660 8671 8720 880
4 8892 8918 8932 8937 8954 8972 9034 9068 9074 9101 9192 9292 9399 9457 9502 951
9 9553 9654 9710 9727 9731 9746 9805 9842 9941 9947 9976 10038 10059 10080 10120
 10140 10148 10169 10189 10243 10277 10307 10340 10387 10393 10453 10477 10521 1
0537 10569 10622 10677 10789 10927 10936 10955 11078 11084 11125 11232 11251 112
93 11318 11349 11374 11456 11474 11499 11525 11601 11684 11708 11735 11756 11768
 11790 11795 11837 11856 11913 11967 11997 12008 12096 12151 12261 12358 12366 1
2382 12508 12524 12556 12760 12838 12896 12994 13022 13076 13269 13289 13344 134
28 13516 13544 13648 13760 13890 13895 14043 14055 14091 14120 14684 14834 15011
 15158 15200 15298 15321 15352 15365 15402 15428 15444 15466 15495 15537 15591 1
5619 15651 15658 15672 15755 15776 15791 15856 15883 15953 16019 16045 16110 162
30 16276 16285 16318 16399 16496 16534 16546 16614 16744 16790 16861 16894 17051
 17058 17070 17158 17182 17238 17257 17282 17642 17654 17741 17821 17844 17877 1
7894 17910 17949 17965 18052 18060 18098 18113 18143 18163 18210 18241 18263 183
16 18336 18355 18420 18521 18541 18554 18574 18675 18698 18716 18731 18754 18767
 18823 18887
Process returned 0 (0x0)   execution time : 2.731 s
Press any key to continue.
```

3 тест:

```
Enter the substring you are looking for: I failed them
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 0
1
9931
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 5.433 s
Press any key to continue.
```

```
Enter the substring you are looking for: I failed them
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 1
1
9931
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 6.194 s
Press any key to continue.
```

```
Enter the substring you are looking for: I failed them
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 2
1
9931
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 9.412 s
Press any key to continue.
```

4 тест:

```
Enter the substring you are looking for: Adolin
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 0
0

Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 5.381 s
Press any key to continue.
```

```
Enter the substring you are looking for: Adolin
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 1
0

Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 5.844 s
Press any key to continue.
```

```
Enter the substring you are looking for: Adolin
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 2
0

Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 4.309 s
Press any key to continue.
```

5 тест:

```
Enter the substring you are looking for: Lord
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 0
3
1979 3545 12626
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 2.973 s
Press any key to continue.
```

```
Enter the substring you are looking for: Lord
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 1
3
1979 3545 12626
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 3.940 s
Press any key to continue.
```

```
Enter the substring you are looking for: Lord
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 2
3
1979 3545 12626
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 6.058 s
Press any key to continue.
```

Выводу по второму набору тестов:

Изначально алгоритм Рабина-Карпа не всегда верно выводил ответ. Вероятно, на этапе вычисления хэша происходило переполнение в промежуточных вычислениях для формулы hash_text = (base * (hash_text - static_cast<unsigned char>(text[i]) * h) + static_cast<unsigned char>(text[i + m])), поэтому тип был изменён на long long, ошибки в некоторых случаях поиска больше не было. Так же, как и с числовыми значениями, промахов и ошибочных выводов не было обнаружено. Время выполнения всё так же мало, разницы как таковой между алгоритмами не выявлено в этом плане.

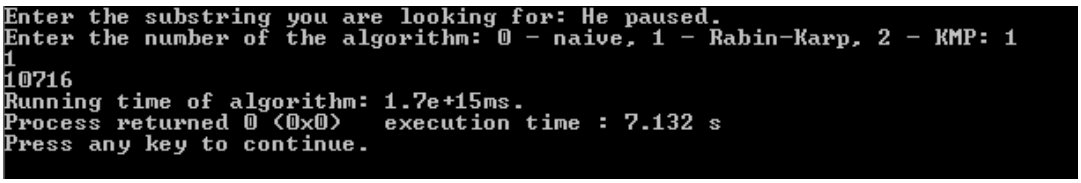
Набор тестов 3: проверка поиска подстроки из нескольких слов/предложения.

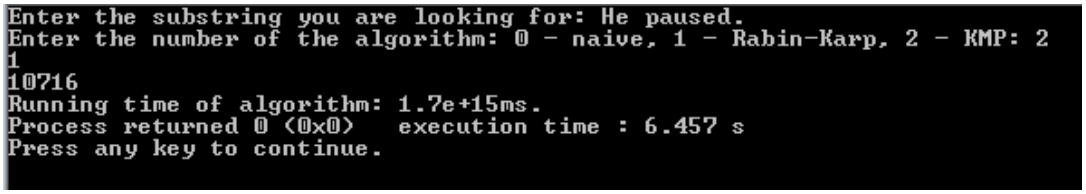
Входные данные:

- He paused.
- She was a fairy.
- He was cold and wet, but he felt a tiny, warm candle flame of determination come alight inside him.
- “Lording,” Gaz called.

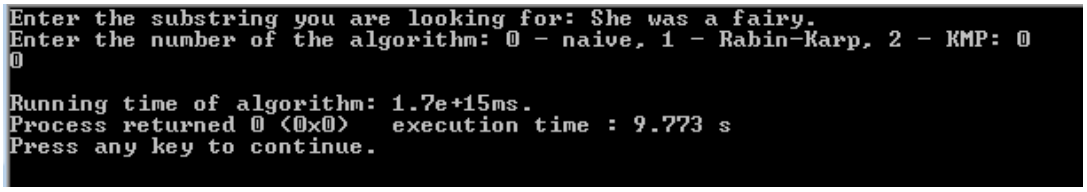
1 тест:

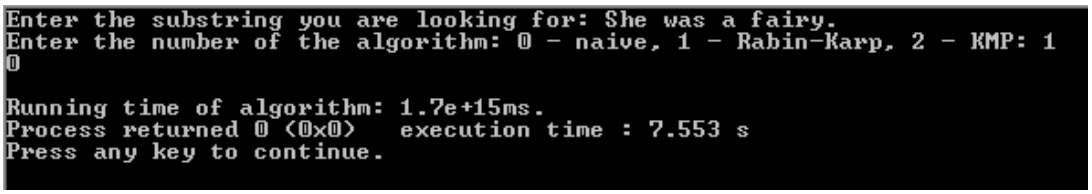
```
Enter the substring you are looking for: He paused.
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 0
1
10716
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 7.542 s
Press any key to continue.
```

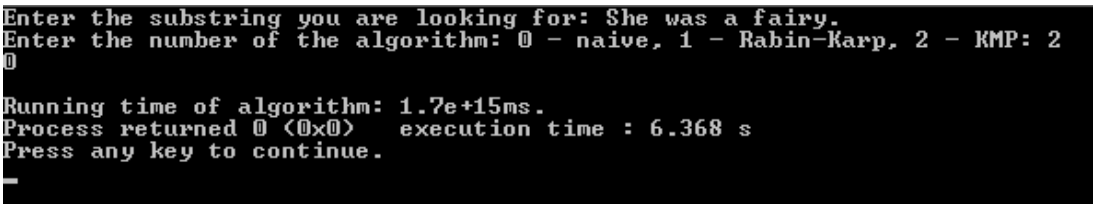




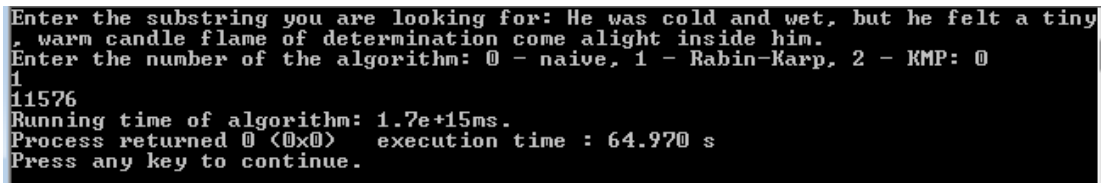
2 тест:

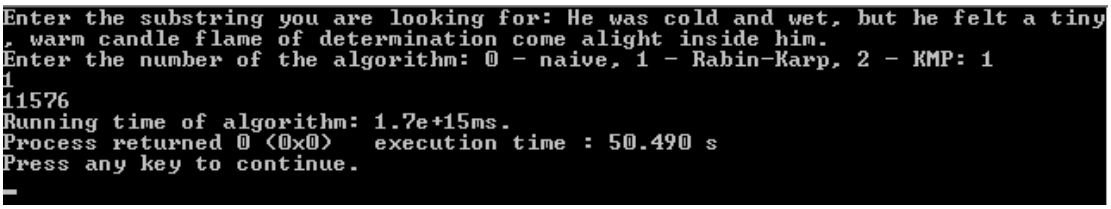


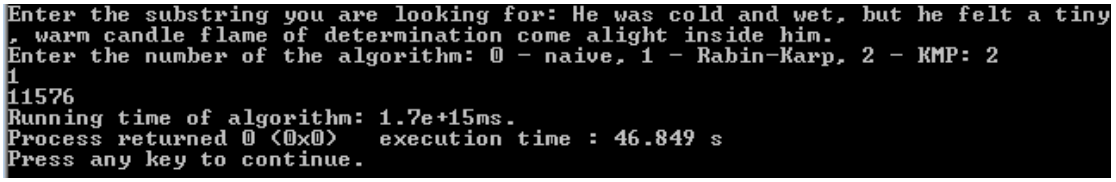




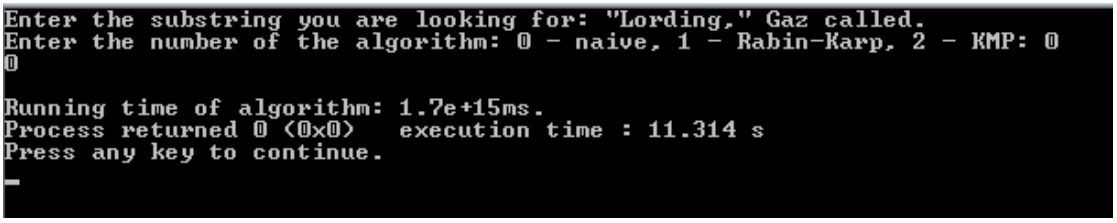
3 тест:

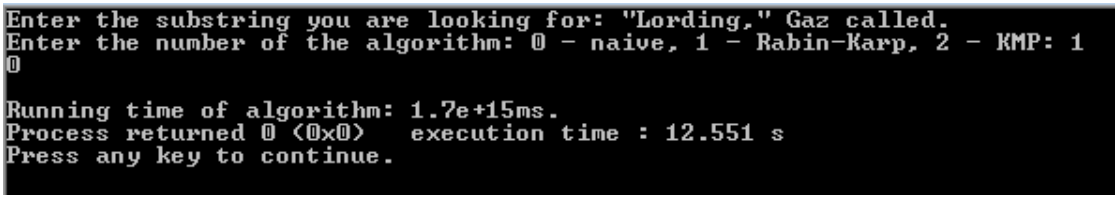


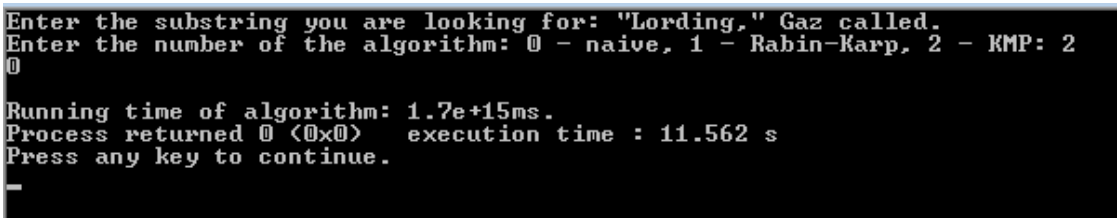




4 тест:







Выводу по третьему набору тестов:

Все данные алгоритмы, очевидно, без модификаций в их первоначальном виде чувствительны к регистру и кодировке. Поэтому последний 4 тест не проходит, хотя данная подстрока имеется в исходном тексте. Есть несколько «видов» символа кавычек, и стандартный терминал ОС Windows поддерживает только один, который не матчится с тем, что есть в исходном тексте. С пробелами и другими знаками препинания проблем не обнаружено. Так же для КМП алгоритма всегда необходим какой-нибудь служебный символ, не использующийся в исходном алфавите (в стандартной реализации берут # или служебные символы, не использующиеся на письме), это тоже необходимо учитывать при поиске подстрок, если текст не является, например, художественным, а парсится какой-нибудь исходный код программы как обычный текст.

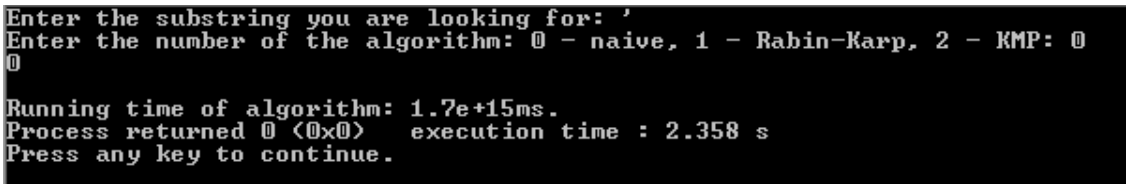
Алгоритмы также чувствительны к пробелам и табам, переносам строк и другим служебным символам, поэтому обычно используют предобработку входного текста.

Набор тестов 4: проверка поиска пунктуационных символов.

Входные данные:

- ‘
- ;
- ?..
- -

1 тест:



```
Enter the substring you are looking for: '
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 1
0

Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 2.617 s
Press any key to continue.
```

```
Enter the substring you are looking for: '
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 2
0

Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 3.694 s
Press any key to continue.
```

2 тест:

```
Enter the substring you are looking for: ;
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 0
3
1268 1578 3187
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 4.129 s
Press any key to continue.
```

```
Enter the substring you are looking for: ;
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 1
3
1268 1578 3187
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 3.389 s
Press any key to continue.
```

```
Enter the substring you are looking for: ;
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 2
3
1268 1578 3187
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 3.119 s
Press any key to continue.
```

3 тест:

```
Enter the substring you are looking for: ?..
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 0
0

Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 4.982 s
Press any key to continue.
```

```
Enter the substring you are looking for: ?..
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 1
0

Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 6.544 s
Press any key to continue.
```

```
Enter the substring you are looking for: ?..
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 2
0

Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 6.586 s
Press any key to continue.
```

4 тест:

```
Enter the substring you are looking for: -
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 0
6
109 374 2958 4777 16346 16361
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 3.452 s
Press any key to continue.
```

```
Enter the substring you are looking for: -
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 1
6
109 374 2958 4777 16346 16361
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 3.060 s
Press any key to continue.
```

```
Enter the substring you are looking for: -
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 2
6
109 374 2958 4777 16346 16361
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 2.043 s
Press any key to continue.
```

Выводу по четвёртому набору тестов:

Как и в случае с предыдущим набором тестов, на этом наборе возникла та же проблема – различия в видах символов. Для пользователя разницы между коротким тире - и длинным – нет, но для алгоритма, очевидно, это разные символы. Поэтому несмотря на б**о**льшее количество тире в тексте, алгоритм считает только конкретные, указанные на вход, и выводит меньшее количество, чем есть на самом деле.

Набор тестов 5:

Входные данные:

- st side of the lumberyard. A low thu
- m just a ma
- al, these chasms. Th

1 тест:

```
Enter the substring you are looking for: st side of the lumberyard. A low thu
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 0
1
3958
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 25.125 s
Press any key to continue.
```

```
Enter the substring you are looking for: st side of the lumberyard. A low thu
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 1
1
3958
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 21.135 s
Press any key to continue.
```

```
Enter the substring you are looking for: st side of the lumberyard. A low thu
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 2
1
3958
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 19.282 s
Press any key to continue.
```

2 тест:

```
Enter the substring you are looking for: m just a ma
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 0
0

Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 10.461 s
Press any key to continue.
```

```
Enter the substring you are looking for: m just a ma
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 1
0

Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 4.951 s
Press any key to continue.
```

```
Enter the substring you are looking for: m just a man
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 2
0

Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 6.382 s
Press any key to continue.
```

3 тест:

```
Enter the substring you are looking for: al, these chasms. Th
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 0
1
5076
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 11.370 s
Press any key to continue.
```

```
Enter the substring you are looking for: al, these chasms. Th
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 1
1
5076
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 11.185 s
Press any key to continue.
```

```
Enter the substring you are looking for: al, these chasms. Th
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 2
1
5076
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 7.148 s
Press any key to continue.
```

Тест с поиском большого объёма данных

Увеличим объём искомой строки (будем считывать её из другого файла).

1 тест: фрагмент, равный трети исходного текста

```
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 0
1
193
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 3.219 s
Press any key to continue.
```

```
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 1
1
193
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 5.849 s
Press any key to continue.
```

```
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 2
1
193
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 2.532 s
Press any key to continue.
```

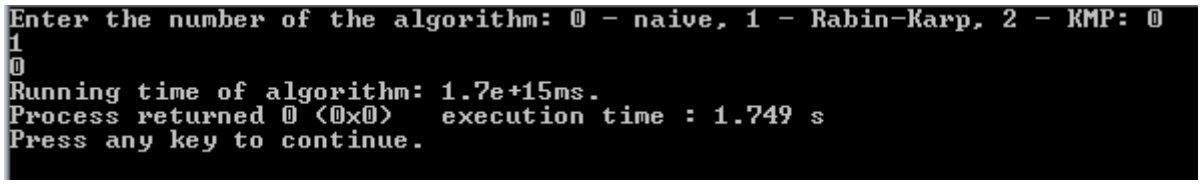
2 тест: фрагмент, равный половине исходного текста

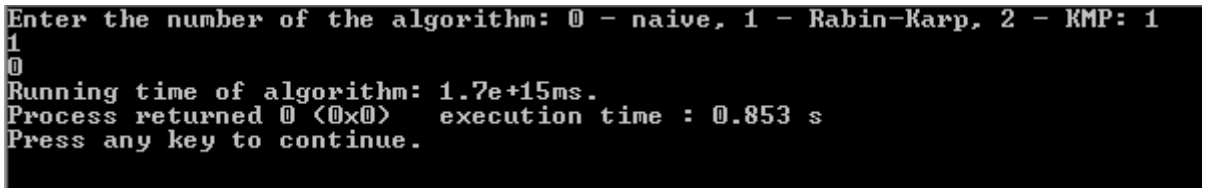
```
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 0
1
78
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 2.530 s
Press any key to continue.
```

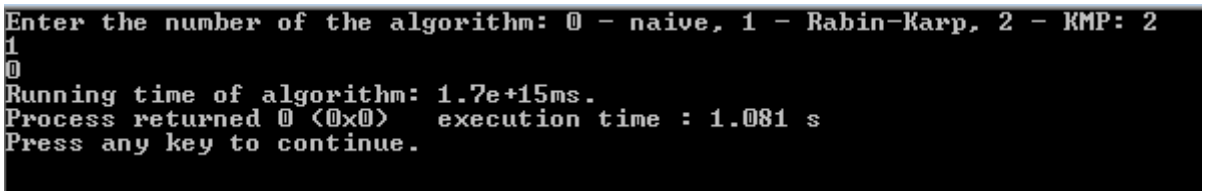
```
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 1
1
78
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 1.194 s
Press any key to continue.
```

```
Enter the number of the algorithm: 0 - naive, 1 - Rabin-Karp, 2 - KMP: 2
1
78
Running time of algorithm: 1.7e+15ms.
Process returned 0 (0x0)   execution time : 1.863 s
Press any key to continue.
```

3 тест: фрагмент, равный исходному тексту







Вывод по тестам с большими данными:

Даже в случае с полным совпадением шаблона с текстом алгоритмы все отрабатывают за считанные наносекунды (в среднем 2000000 нс). Вероятно, нужны ещё большие объёмы и шаблона, и исходного текста, чтобы увидеть значительную разницу. В любом случае, данные алгоритмы не использовались бы, например, в сетевых поисках, если бы не справлялись с большими объёмами данных, так что можно предположить, что разница проявится только на объёмах в сотни раз больше данного отрывка текста.

Вывод

Все три рассмотренных алгоритма работают за короткое время и удобны для поиска single-шаблонов в исходных текстах. Несмотря на небольшую разницу во временах работы, на практике особо не проявляющуюся, наивный алгоритм в данном случае нисколько не уступает более продвинутым алгоритмам Рабина-Карпа и Кнута-Морриса-Пратта. Для первых двух рассмотренных алгоритмов важно, на каких (обработанных или нет) данных будет искаться подстрока, так как в выигрышных случаях уменьшается время работы этих алгоритмов. Тем не менее, как показала практика, все три алгоритма достаточно эффективны и пригодны для и малых, и больших задач поиска.

Заключение

В результате выполнения курсовой работы были рассмотрены, изучены, реализованы, протестированы и проанализированы три алгоритма поиска подстроки в строке. Были возобновлён опыт работы с синтаксисом языка C++, в том числе работы со структурами данных, циклами и библиотеками.

Исходный код:

https://github.com/tchn11/ITMO-labs-master-prog/tree/main/Algorithms%20and%20Data%20Structures/course%20work