Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе №7

по курсу «Анализ Алгоритмов»

на тему: «Алгоритмы поиска»

Студент ИУ7-51Б (Группа) _____ (Подпись, дата) Савинова М. Г. (Фамилия И. О.)

Преподаватель _____ (Подпись, дата) Волкова Л. Л. (Фамилия И. О.)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Задача поиска в строке заключается в следующем: дана длинная строка («текст») $w = a_1 \dots a_n$, и короткая искомая строка («шаблон») $x = b_1 \dots b_m$. Требуется найти все вхождения x в w в качестве подстроки, то есть, все смещения s, для которых подстрока $ws = a_{s+1} \dots a_{s+m}$ совпадает с $b_1 \dots b_m$ [1].

Целью данной лабораторной работы является исследование лучших и худших случаев работы алгоритмов поиска подстроки в строке: Кнута – Морриса – Пратта и его модификации.

Для достижения поставленной цели необходимо решить следующие задачи:

1) описать используемые алгоритмы поиска;

2) выбрать средства программной реализации;

3) реализовать данные алгоритмы поиска;

4) проанализировать алгоритмы по количеству сравнений.

1 Аналитический раздел

В данном разделе будут рассмотрены алгоритм Кнута – Морриса – Пратта и его модификация.

Для этого введем следующие обозначения:

— подстрока x называется *префиксом* строки w, если есть такая подстрока z, что $w = xz$;

— подстрока x называется *суффиксом* строки w, если есть такая подстрока z, что $w = zx$ [2].

1.1 Алгоритм Кнута – Морриса – Пратта

Рассматриваемый алгоритм основывается на том, что после частичного совпадения начальной части подстроки с соответствующими символами строки фактически известна пройденная часть строки и можно, вычислить некоторые сведения, с помощью которых затем быстро продвинуться по строке.

Основным отличием алгоритма Кнута – Морриса – Пратта от алгоритма прямого поиска заключается в том, что сдвиг подстроки выполняется не на один символ на каждом шаге алгоритма, а на некоторое переменное количество символов. Следовательно, перед тем как осуществлять очередной сдвиг, необходимо определить величину сдвига [2].

Для определения этого сдвига используют *префиксную функцию*. Это функция, которая для всякого префикса $b_1 \dots b_i$ строки $x = b_1 \dots b_m$ выдает длину наибольшего суффикса подстроки $b_1 \dots b_i$, который одновременно будет и префиксом x [1].

1.2 Модификация алгоритма Кнута – Морриса – Пратта

Модификация заключается в введении эвристики «плохого» символа по аналогии с алгоритмом Бойера – Мура с адаптацией к обходу подстроки слева направо.

Эвристика «плохого» символа определяет сдвиг наибольшего возможного количества позиций, если символ строки не совпадает с символом подстроки в заданной позиции [1].

Таким образом, в модифицированном алгоритме мы используем как информацию о суффиксах, так и о длине сдвига.

Вывод

В данном разделе были рассмотрены алгоритм Кнута – Морриса – Пратта и его модификация.

2 Конструкторский раздел

В данном разделе будут представлены псевдокоды алгоритмов Кнута – Морриса – Пратта и его модификации, а также алгоритмов для формирования массивов сдвигов и суффиксов.

2.1 Требования к программному обеспечению

К программному обеспечению предъявлен ряд требований:

1) наличие интерфейса для выбора действия;

2) возможность ввода строки и подстроки;

3) возможность выполнения операции поиска подстроки в строке.

2.2 Разработка алгоритмов

В качестве инструмента для создания псевдокода использован пакет `algorithm`.

В случае нахождения подстроки в строке просиходит возврат **shift**, то есть смещение относительно начала строки, иначе — **-1**.

Определим следующие операторы и функции:

— оператор $\leftarrow$ обозначает присваивание значение переменной;

— оператор $[i]$ обозначает получение элемента из массива с индексом i;

— функция max возвращает максимальное значение;

— функция *ord* возвращает целое число, представляющее символ Юникода;

— функция *length* возвращает длину массива, строки.

В листингах 1–2 рассмотрены псевдокоды алгоритмов формирования дополнительных массивов суффиксов и сдвигов соответственно.

В листингах 3–4 рассмотрены псевдокоды алгоритма Кнута – Морриса – Пратта и его модификации.

Листинг 1 Псевдокод алгоритма формирования массива суффиксов

На входе: строка $w = a_1 \dots a_n$, ссылка на массив π

```
1:  function MakePiList(w)
2:      j ← 0
3:      i ← 1
4:      while i < n do
5:          if a_j = a_i then
6:              π[i] ← j + 1
7:              j ← j + 1
8:              i ← i + 1
9:          else
10:             if j = 0 then
11:                 π[i] ← 0
12:                 i ← i + 1
13:             else
14:                 j ← π[j − 1]
15:             end if
16:         end if
17:     end while
18: end function
```

Листинг 2 Псевдокод алгоритма формирования массива сдвигов

На входе: строка $w = a_1 \dots a_n$, ссылка на массив $badChars$

```
1:  function MakeBadChars(w, badChars)
2:      j ← 0
3:      i ← 1
4:      for i = (0, n) do
5:          badChars[ord(w_i)] ← i
6:      end for
7:  end function
```

Листинг 3 Псевдокод алгоритма Кнута – Морриса – Пратта

На входе: строка $w = a_1 \dots a_n$, подстрока $x = b_1 \dots b_m$, массив суффиксов π

```
1:  function KMP(w, x, π)
2:      i ← 0
3:      j ← 0
4:      n ← length(w)
5:      m ← length(x)
6:      while i < n do
7:          if a_i = b_j then
8:              i ← i + 1
9:              j ← j + 1
10:             if j = m then return i − n
11:             end if
12:         else
13:             if j = 0 then
14:                 i ← i + 1
15:             else
16:                 j ← π[j − 1]
17:             end if
18:         end if
19:     end while
         return −1
20: end function
```

Листинг 4 Псевдокод модифицированного алгоритма Кнута – Морриса – Пратта

На входе: строка $w = a_1 \dots a_n$, подстрока $x = b_1 \dots b_m$,
массив суффиксов π, массив сдвигов $badChars$

```
1:  function КМРОРТ(w, x, π, badChars)
2:      n ← length(w)
3:      m ← length(x)
4:      shift ← 0
5:      while shift ≤ n − m do
6:          j ← m − 1
7:          while j ≥ 0 and x_j = w_{shift+j} do
8:              j ← m − 1
9:          end while
10:         if j < 0 then return shift
11:         else
12:             shift ← shift + max(pi[j], badChars[ord(x_{shift+j})])
13:         end if
14:     end while
         return −1
15: end function
```

Вывод

В данном разделе были описаны псевдокоды для алгоритмов формирования массивов суффиксов и сдвигов, а также для алгоритма Кнута – Морриса – Пратта и его модификации.

3 Технологический раздел

В данном разделе будут описаны средства реализации программного обеспечения, а также представлены листинги и функциональные тесты.

3.1 Средства реализации

Для реализации данной лабораторной работы был выбран язык `C++`, так как в нем есть стандартная библиотека `ctime`, которая позволяет производить замеры процессорного времени выполнения программы [3].

3.2 Сведения о модулях программы

Данная программа разбита на следующие модули:

— `main.cpp` — файл, содержащий точку входа в программу;

— `algs.cpp` — файл, содержащий реализации алгоритмов поиска.

3.3 Реализация алгоритмов

На листингах 3.1 — 3.4 представлены реализации разрабатываемых алгоритмов.

Листинг 3.1 – Реализация алгоритма Кнута – Морриса – Пратта поиска строки в подстроке

```
int KMP(const string& text, const string& pattern) {

    int m = pattern.length();
    int n = text.length();

    vector<int> pi = makePiList(pattern);

    int i = 0, j = 0;

    ++comps.first;
    while (i < n) {

        ++comps.first;
        if (text[i] == pattern[j]) {

            i++, j++;
```

```
17
18              ++comps.first;
19              if (j == m)
20                  return i - m;
21
22          } else {
23              ++comps.first;
24
25              if (j == 0) {
26                  i++;
27              }
28              else
29                  j = pi[j - 1];
30          }
31          ++comps.first;
32      }
33
34      return -1;
35  }
```

Листинг 3.2 – Реализация модифицированного алгоритма Кнута – Морриса – Пратта поиска строки в подстроке

```
1  int KMP_optimized(const string& text, const string& pattern) {
2
3      int m = pattern.length();
4      int n = text.length();
5
6      vector<int> pi = makePiList(pattern);
7      unordered_map<char, int> badChars =
           makeBadCharsList(pattern);
8
9      int shift = 0;
10
11     ++comps.second;
12     while (shift <= n - m) {
13
14         int j = m - 1;
15
16         ++comps.second;
17         while (j >= 0 && pattern[j] == text[shift + j])
18             j--;
19
```

```
20          ++comps.second;
21          if (j < 0)
22              return shift;
23          else {
24              int k = m;
25
26              ++comps.second;
27              if (badChars.contains(text[shift + j]))
28                  k = badChars[text[shift + j]];
29
30              shift += max(pi[j], k);
31          }
32
33          ++comps.second;
34      }
35
36      return -1;
37  }
```

Листинг 3.3 – Реализация алгоритма формирования массива суффиксов

```
1   vector<int> makePiList(const string& pattern) {
2
3       int n = pattern.length();
4       vector pi(n, 0);
5
6       int j = 0, i = 1;
7
8       while (i < n) {
9
10          if (pattern[j] == pattern[i]) {
11              pi[i] = j + 1;
12              j++, i++;
13          } else {
14              if (j)
15                  j = pi[j - 1];
16              else {
17                  pi[j] = 0;
18                  i++;
19              }
20          }
21      }
22
```

```
23      return pi;
24  }
```

Листинг 3.4 – Реализация алгоритма формирования массива сдвигов

```
1  unordered_map<char, int> makeBadCharsList(const string& pattern)
       {
2      int n = pattern.length();
3      unordered_map<char, int> badChars;
4
5      for (int i = 0; i < n; ++i)
6          badChars[pattern[i]] = n - i - 1;
7      return badChars;
8  }
```

3.4 Тестирование

В таблице 3.1 приведены модульные тесты для разработанных алгоритмов поиска. Все тесты успешно пройдены.

Таблица 3.1 – Модульные тесты

Строка-текст	**Строка-образец**	**Ожидаемый р-т**	**Фактический р-т**	
			КМП	**Модиф. КМП**
abcdefgabcdefg	bcd	1	1	1
abcdefghi	xyz	-1	-1	-1
abababababab	abab	0	0	0
abcdefg	abcd	0	0	0
xyabc	abc	2	2	2

Вывод

В данной части работы были представлены листинги реализованных алгоритмов и тесты, успешно пройденные программой.

4 Исследовательский раздел

В данном разделе будут приведены примеры работы программ, постановка исследования и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени, представлены далее.

— Процессор: AMD Ryzen 5 5500U – 2.10 ГГц;

— Оперативная память: 16 ГБайт;

— Операционная система: Windows 10 Pro 64-разрядная система версии 22H2.

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.2 Демонстрация работы программы

На рисунке 4.1 представлена демонстрация работы разработанного ПО.

```
                Меню
1. Выполнить поиска подстроки в строке
    а) алгоритм Кнута-Морриса-Пратта;
    б) модифицированный КМП.
2. Подсчитать количество сравнений при выполнении поиска подстроки.
0. Выход.

Выберите опцию (0-2): 1

Введите исходную строку: abracadabrad
Введите исходную подстроку: abra
Смещение относительно начала (КМП): 0
Смещение относительно начала (КМП оптимизированный): 0

                Меню
1. Выполнить поиска подстроки в строке
    а) алгоритм Кнута-Морриса-Пратта;
    б) модифицированный КМП.
2. Подсчитать количество сравнений при выполнении поиска подстроки.
0. Выход.

Выберите опцию (0-2): 1

Введите исходную строку: abracadabrad
Введите исходную подстроку: rad
Смещение относительно начала (КМП): 9
Смещение относительно начала (КМП оптимизированный): 9
```

Рисунок 4.1 – Демонстрация работы программы

4.3 Число сравнений

В качестве длины исходной строки были выбраны следующие значения: $256, 512, 1024, 2048, 4096$. Так же для получения более точного результата, каждый замер производился 100 раз.

Определим следующие случаи:

— **лучший**: когда подстрока находится в начале строки;

— **худший**: когда подстрока находится в конце строки или отсутствует в целом.

На рисунке 4.2 изображены результаты исследования для лучшего случая.

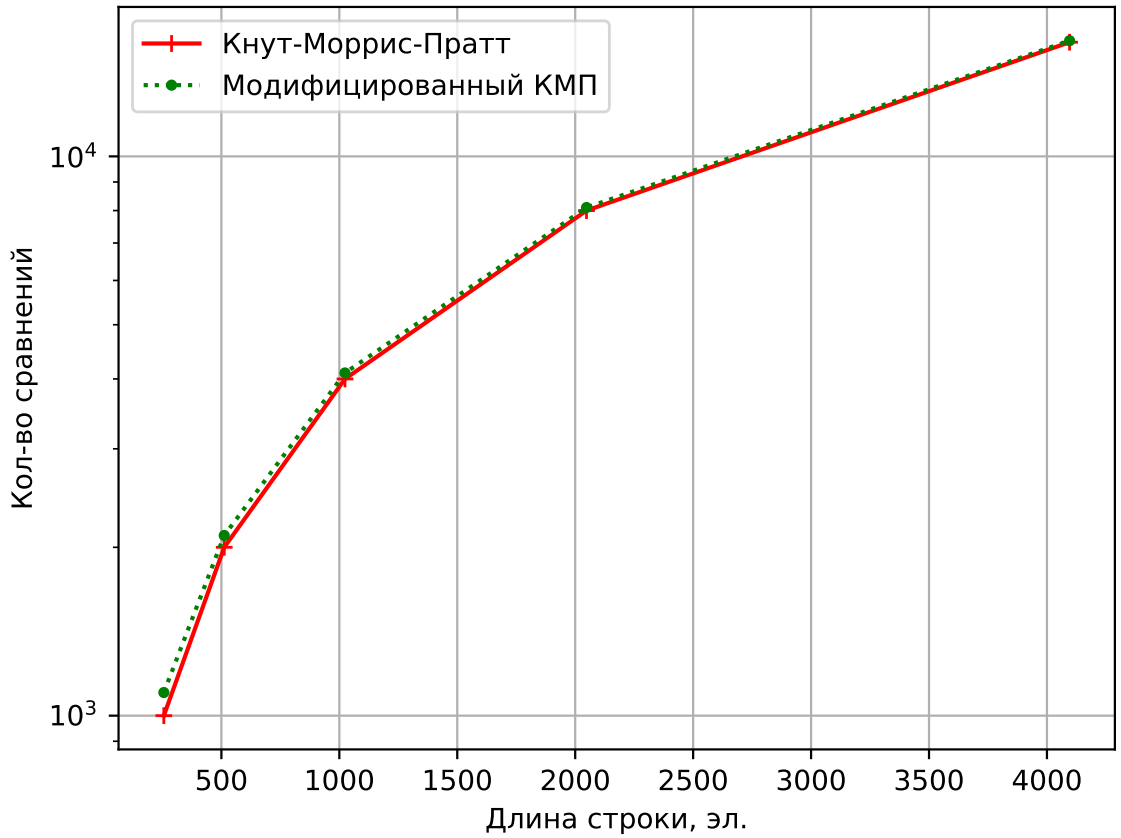

Рисунок 4.2 – Сравнение количества сравнений при работе алгоритмов для лучшего случая

На рисунке 4.3 изображены результаты исследования для худшего случая, когда искомая подстрока находится в конце строки.

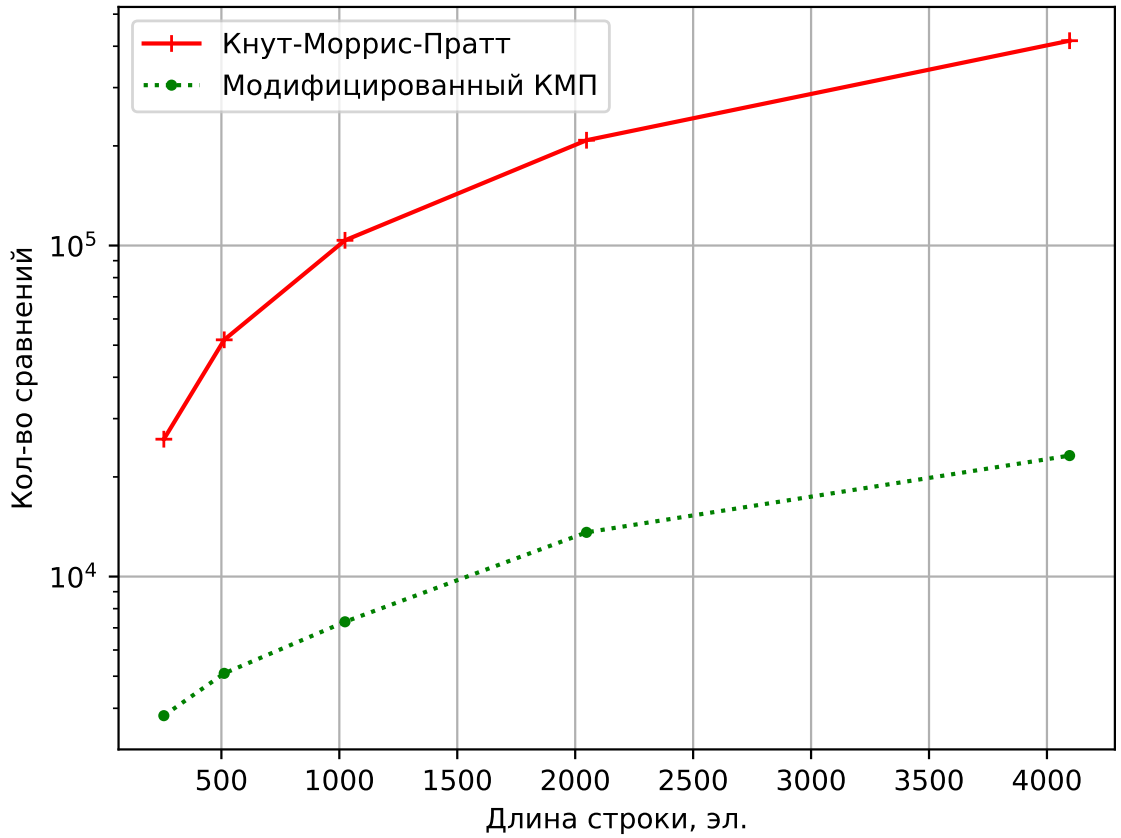

Рисунок 4.3 – Сравнение количества сравнений при работе алгоритмов для лучшего случая (подстрока в конце строки)

На рисунке 4.4 изображены результаты исследования для худшего случая, когда искомой подстроки нет.

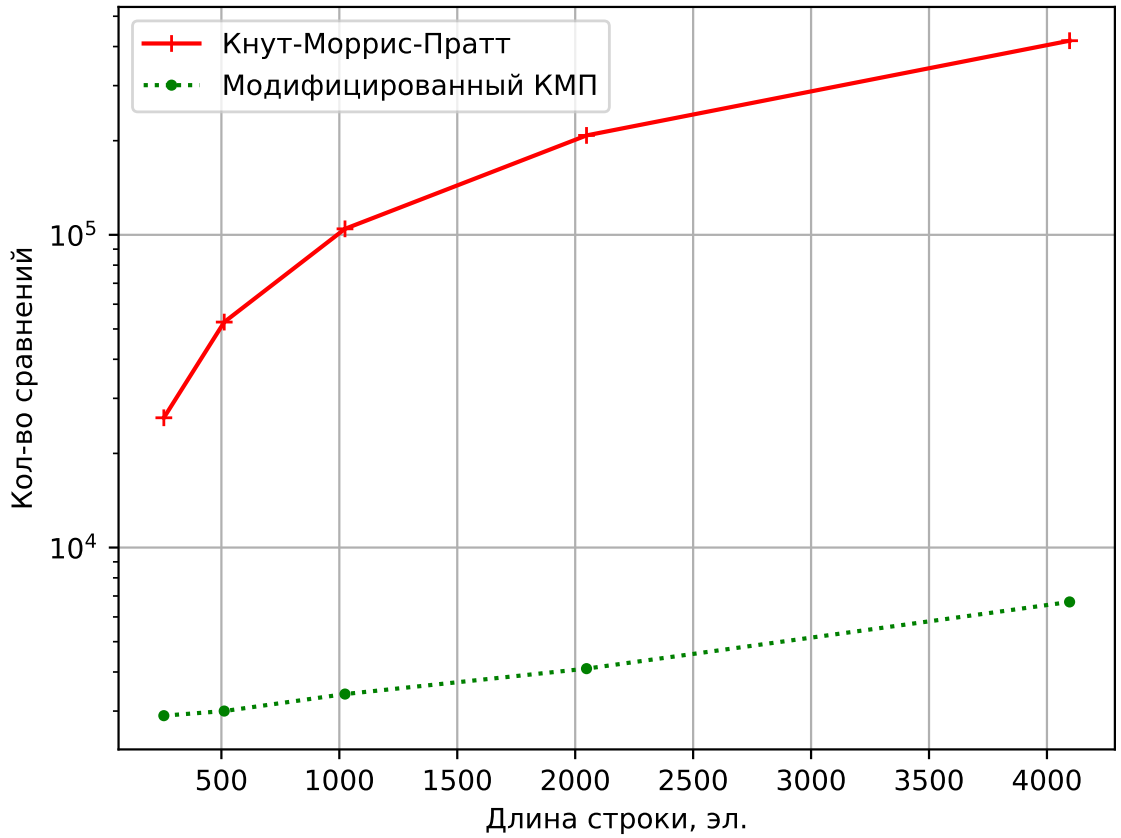

Рисунок 4.4 – Сравнение количества сравнений при работе алгоритмов для лучшего случая (подстроки нет)

Вывод

Наименьшее число сравнений получается в случае нахождения исходной подстроки в начале строки (для КМП и его модификации).

Количество сравнений в случае нахождения подстроки в начале исходной строки для алгоритма КМП и его модификации одинаково.

Самым худшим случаем является нахождение исходной подстроки в самом конце исходной строки, поскольку в таком случае алгоритм выполняет максимальное количество сравнений.

ЗАКЛЮЧЕНИЕ

Цель данной лабораторной работы была достигнута: исследованы лучшие и худшие случаи работы алгоритмов поиска.

Для достижения поставленной цели были решены следующие задачи:

1) описаны используемые алгоритмы поиска;
2) выбраны средства программной реализации;
3) реализованы алгоритмы поиска Кнута – Морриса – Пратта и его модификация;
4) проанализированы алгоритмы по количеству сравнений и сделаны следующие выводы:
   - наименьшее число сравнений получается в случае нахождения исходной подстроки в начале строки (для КМП и его модификации);
   - количество сравнений в случае нахождения подстроки в начале исходной строки для алгоритма КМП и его модификации одинаково;
   - самым худшим случаем является нахождение исходной подстроки в самом конце исходной строки, поскольку в таком случае алгоритм выполняет максимальное количество сравнений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Охотин. А. С.* Математические основы алгоритмов. — 2022. — URL: https://users.math-cs.spbu.ru/~okhotin/teaching/algorithms1_2022 ; Лекция 6.
2. *Ваныкина. Г. В.* Структуры и алгоритмы компьютерной обработки данных. — 2012. — URL: https://intuit.ru/studies/courses/648/504/info ; Лекция 40.
3. C++ language. — [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/language (дата обращения: 21.12.2023).