МИНОБРНАУКИ РОСИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания №6.2
Тема:
Поиск образца в тексте
Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Васильев Б.А.
Группа: ИКБО-20-23

Москва 2024

СОДЕРЖАНИЕ

ЦЕЛЬ РАБОТЫ

Освоить приёмы реализации алгоритмов поиска образца в тексте.

ХОД РАБОТЫ

Формулировка задачи

Разработайте приложения в соответствии с заданиями в индивидуальном варианте (п.2). В отчёте в разделе «Математическая модель решения (описание алгоритма)» разобрать алгоритм поиска на примере. Подсчитать количество сравнений для успешного поиска первого вхождения образца в текст и безуспешного поиска. Определить функцию (или несколько функций) для реализации алгоритма поиска. Определить предусловие и постусловие. Сформировать таблицу тестов с указанием успешного и неуспешного поиска, используя большие и небольшие по объему текст и образец, провести на её основе этап тестирования. Оценить практическую сложность алгоритма в зависимости от длины текста и длины образца и отобразить результаты в таблицу (для отчета).

Вариант	Задачи варианта
1	1. Линейный поиск первого вхождения подстроки в строку. 2. Используя алгоритм Бойера-Мура-Хорспула, найти последнее вхождение подстроки в строку.

Рисунок 1 – Индивидуальный вариант задачи

Описание подхода к решению

Прямой поиск: описание алгоритма и пример

Алгоритм **прямого поиска** (или наивного поиска) строк заключается в последовательном сравнении каждого символа образца с соответствующим символом подстроки в тексте. Если все символы совпадают, алгоритм завершает поиск. В противном случае, образец сдвигается на одну позицию вправо, и процедура повторяется. Если образец не найден в тексте, алгоритм завершает работу.

Рассмотрим пример на основании следующего текста и образца.

Пример 1: успешный поиск

Текст (haystack): `ABCDABCDABEE`
Образец (needle): `ABCD`

1. Алгоритм начинает сравнивать образец с первой позиции текста:

- Сравниваем `A` с `A` — совпадение.
- Сравниваем `B` с `B` — совпадение.
- Сравниваем `C` с `C` — совпадение.
- Сравниваем `D` с `D` — совпадение.

Так как все символы образца совпали с подстрокой текста, алгоритм завершает поиск, и первое вхождение образца найдено на позиции 1.

Количество сравнений: 4.

Пример 2: безуспешный поиск

Теперь рассмотрим пример безуспешного поиска, где в тексте нет вхождений образца.

Текст (haystack): `ABCDABCDABEE`
Образец (needle): `XYZ`

1. Алгоритм начинает сравнение с первой позиции текста:
   - Сравниваем `A` с `X` — не совпадает.
2. Сдвигаем образец на одну позицию вправо:
   - Сравниваем `B` с `X` — не совпадает.
3. Сдвигаем образец на одну позицию вправо:
   - Сравниваем `C` с `X` — не совпадает.

Этот процесс повторяется до тех пор, пока не будут проверены все возможные сдвиги. Так как в тексте не нашлось совпадений с образцом, поиск завершается неудачей.

Количество сравнений: 12 (каждый символ текста сравнивается с первым символом образца).

Пример 3: худший случай

Теперь рассмотрим худший случай, когда первые символы образца совпадают с текстом, но последний символ не совпадает.

Текст (haystack): `AAAAAAAAAAAA`
Образец (needle): `AAAB`

1. Алгоритм начинает с первой позиции текста:
   - Сравниваем `A` с `A` — совпадение.
   - Сравниваем `A` с `A` — совпадение.
   - Сравниваем `A` с `A` — совпадение.
   - Сравниваем `A` с `B` — не совпадает.

Образец сдвигается на одну позицию вправо, и весь процесс повторяется до конца текста. Этот случай является худшим, так как алгоритм вынужден каждый раз проверять почти все символы образца, прежде чем обнаружить несовпадение.

Количество сравнений: 48 (для каждого сдвига проверяются все символы образца)

Прямой поиск может быть эффективным для небольших строк, однако его производительность в худших случаях оставляет желать лучшего, так как количество сравнений линейно зависит от длины текста и образца.

Худший случай:

В худшем случае каждый символ образца почти полностью совпадает с подстрокой текста, кроме последнего символа. Тогда алгоритм проверяет все символы образца для каждого сдвига по тексту.

- Допустим, длина текста — `n`, а длина образца — `m`.
- На каждом шаге происходит до `m` сравнений, и таких шагов может быть до `n - m + 1` (всего возможных сдвигов).
- Таким образом, количество сравнений в худшем случае будет около $m \times (n - m + 1)$, что асимптотически равно $O(n \times m)$.

Средний случай:

В среднем случае предполагается, что совпадения случаются реже, и алгоритм быстрее находит несовпадения на ранних стадиях сравнения.

- В среднем ожидаемое количество сравнений будет меньше, так как чаще всего образец не совпадает с подстрокой на ранних позициях.
- Поэтому сложность алгоритма в среднем случае приближается к $O(n)$, так как большинство сравнений происходит до первого несовпадения.

Алгоритм Бойера-Мура-Хорспула: описание и пример

Алгоритм **Бойера-Мура-Хорспула** является упрощенной версией алгоритма Бойера-Мура. Он использует идею сдвига образца по тексту на основе анализа символов, не входящих в подстроку. Этот алгоритм эффективнее на практике по сравнению с наивным прямым поиском, особенно при работе с большими строками.

Основная идея заключается в том, что при несовпадении символа образца и текста на определенной позиции происходит сдвиг образца вправо на несколько символов, минуя некоторые позиции, где несовпадение гарантировано. Сдвиг определяется таблицей смещений, которая рассчитывается для каждого символа образца.

Описание алгоритма

1. Алгоритм начинает сравнение символов образца и текста справа налево.
2. Если найдено несовпадение, используется таблица смещений для сдвига образца вправо. Если символ, на котором произошло несовпадение, не входит в образец, то образец сдвигается настолько, что можно пропустить несколько символов текста.
3. Этот процесс повторяется до тех пор, пока образец не будет найден или не завершится весь текст.

Пример 1: успешный поиск

Текст (haystack): `ABCDABCDABEE`
Образец (needle): `ABCD`

1. Строим таблицу смещений для символов образца. Если символ встречается в образце, сдвиг равен расстоянию до последнего вхождения этого символа. Если символа нет в образце, сдвиг равен длине образца:
   - `A` — 3,
   - `B` — 2,
   - `C` — 1,
   - `D` — 4 (длина образца).
2. Начинаем сравнение с позиции 4 в тексте:
   - Сравниваем `D` с `D` — совпадение.
   - Сравниваем `C` с `C` — совпадение.
   - Сравниваем `B` с `B` — совпадение.
   - Сравниваем `A` с `A` — совпадение.

Образец совпал с подстрокой текста на первой позиции. Поиск завершен.

Количество сравнений: 4.

Пример 2: безуспешный поиск

Текст (haystack): `ABCDABCDABEE`
Образец (needle): `XYZ`

1. Строим таблицу смещений:
   - `X` — 3,
   - `Y` — 2,

    - `z` — 1.
2. Начинаем сравнение с позиции 3 в тексте:
    - Сравниваем `c` с `z` — не совпадает.
    - Так как `c` нет в образце, смещаем образец вправо на 3 позиции.
3. Повторяем процесс для всех возможных сдвигов в тексте, но совпадений не будет найдено.

Алгоритм завершит работу, когда все возможные сдвиги будут проверены.

Количество сравнений: значительно меньше, чем при прямом поиске, так как сдвиги позволяют пропускать части текста. В этом случае 4 сравнения.

Сложность алгоритма

Средний случай

Алгоритм Бойера-Мура-Хорспула работает быстрее на практике, так как в среднем использует меньше сравнений благодаря сдвигам. Средняя сложность приближается к O(n /m), так как сдвиги позволяют быстро пропускать символы текста.

Худший случай

В худшем случае, когда образец почти полностью совпадает с текстом, количество сравнений приближается к O(n×m).

Код программы

Реализуем код приложения на языке программирования C++ (рис. 2-6)

```cpp
#include <iostream>
#include <fstream>
#include <chrono>
#include <string>
#include <string.h>

using namespace std;

int NOT_FOUND_INDEX = -1;
const int ALPHABET_SIZE = 256;
int COMPARISONS = 0;

// Search for the FIRST occurence of needle in haystack
int linear_search(const string& haystack, const string& needle)
{
    int i = 0;
    int j = 0;
    int lh = haystack.length();
    int ln = needle.length();

    if (lh < ln) // Needle is longer than Haystack
    {
        return NOT_FOUND_INDEX; // Unsuccessful search
    }

    // Iterating over characters in Haystack
    for (int i = 0; i <= lh - ln; i++)
    {
        int j = 0;
        while (j < ln && haystack[i + j] == needle[j])
        {
            COMPARISONS++;
            j++;
        }
        COMPARISONS++;

        // Succesful search
        if (j == ln) return i;
    }
    return NOT_FOUND_INDEX; // Unsuccessful search
}
```

Рисунок 2 – код программы (часть 1)

```
43  void fill_char_shift_table(const string& needle, int char_shift_table[])
44  {
45      int ln = needle.length();
46
47      for (int i = 0; i < ALPHABET_SIZE; i++)
48      {
49          char_shift_table[i] = ln;
50      }
51
52      for (int i = 0; i < needle.length() - 1; i++)
53      {
54          char_shift_table[static_cast<unsigned char>(needle[i])] = ln - 1 - i;
55      }
56
57  }
58
59  // Search for the LAST occurence of needle in haystack
60  int bmh_search(const string& haystack, const string& needle)
61  {
62      int lh = haystack.length();
63      int ln = needle.length();
64      int last_occurence_index = NOT_FOUND_INDEX;
65
66      if (lh < ln) // Needle is longer than Haystack
67      {
68          return NOT_FOUND_INDEX; // Unsuccessful search
69      }
70
71      int char_shift_table[ALPHABET_SIZE];
72      fill_char_shift_table(needle, char_shift_table);
73      int i = 0;
74
75      while (i <= lh - ln)
76      {
77          int j = ln - 1;
78
79          while (j >= 0 && haystack[i+j] == needle[j])
80          {
81              COMPARISONS++;
82              j--;
83          }
```

Рисунок 3 – код программы (часть 2)

```
84          COMPARISONS++;
85
86          if (j < 0) last_occurence_index = i;
87          // Determine shift
88          // Mismatch at the last character of the needle
89          if (j == ln - 1)
90          {
91              // Shift based on the mismatched character from haystack
92              i += char_shift_table[static_cast<unsigned char>(haystack[i+j])];
93          }
94          // Mismatch left of the last character of the needle
95          else
96          {
97              // Shift based on the last character of the needle
98              i+= char_shift_table[static_cast<unsigned char>(needle[ln - 1])];
99          }
100     }
101     return last_occurence_index;
102
103 }
104
105 int main()
106 {
107     string filename; // Where we search
108
109
110     // Get name of Haystack file
111     cout << "Enter name of textfile (haystack) with .txt: ";
112     getline(cin, filename);
113
114     // Opening file
115     ifstream file(filename);
116     if (!file.is_open()) // If failed to open
117     {
118         cout << "Couldn't open file: " << filename << endl;
119         return 1;
120     }
```

Рисунок 4 – код программы (часть 3)

```
122     // Reading the whole file into Haystack string
123     string haystack;
124     string line;
125     while (getline(file, line))
126     {
127         haystack += line + "\n";
128     }
129
130     // Closing file
131     file.close();
132
133
134     // Get Needle from user
135     string needle; // What we search
136     cout << "Enter substring to search for (needle): ";
137     getline(cin, needle);
138
139     // Propmting user to choose search option
140     int search_option;
141     cout << "\t---Search options---" << endl;
142     cout << "1 - Linear search to find the first match" << endl;
143     cout << "2 - Boyer-Moore-Horspool search to find the last match" << endl;
144     cout << "Which search to use: ";
145     cin >> search_option;
146
147     // Executing chosen search
148     int found_index;
149     auto start_time = chrono::high_resolution_clock::now();
150     switch(search_option)
151     {
152         case 1:
153             found_index = linear_search(haystack, needle);
154             break;
155
156         case 2:
157             found_index = bmh_search(haystack, needle);
158             break;
159     }
160     auto end_time = chrono::high_resolution_clock::now();
161     auto time_delta = chrono::duration_cast<chrono::microseconds>(end_time - start_time);
```

Рисунок 5 – код программы (часть 5)

```
162     cout << "Search completed in " << time_delta.count() << " microseconds" << endl;
163     cout << "Total number of comparisons made " << COMPARISONS << endl;
164     cout << "Needle: \"" << needle << "\"";
165     if (found_index == NOT_FOUND_INDEX)
166     {
167         cout << " was not found in textfile" << endl;
168         return 1;
169     }
170
171     cout <<  " was found at index: " << found_index << endl;
172     return 0;
173 }
```

Рисунок 6 – код программы (часть 6)

Результаты тестирования

Выполним тестирование программы (рис. 7-15). Обозначим каждый из запусков четырёхзначным двоичным кодом. 0-й разряд означает успех/неуспех поиска, 1-й разряд означает длину текста (0 – короткий, 1 – длинный), 2-й разряд означает длину образца (0 – короткий, 1 – длинный), 3-й разряд означает выбранный алгоритм поиска (0 – прямой поиск, 1 – алгоритм Бойера-Мура-Хорспула). В итоге будет проведено 16 тестовых запусков. Текстовые файлы заполнены случайными символами ASCII с 32ого по 125ый, последней строкой вставлена специальная последовательность.

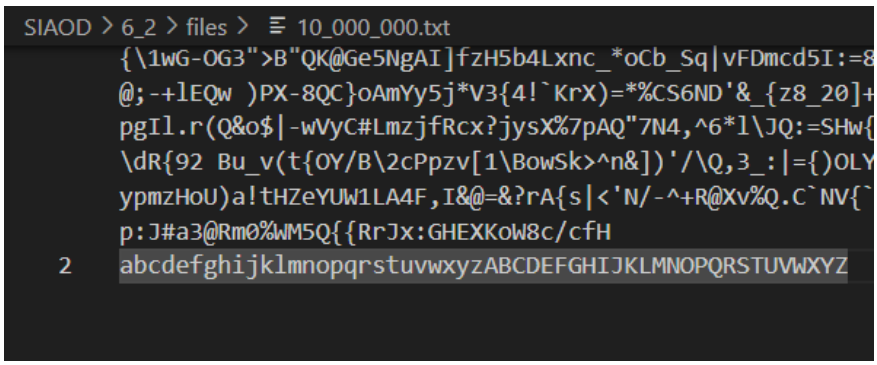

Рисунок 7 – Пример текстового файла

```
D:\Boris\Documents\UNI\SIAOD\6_2>g++ main.cpp -o main && .\main
Enter name of textfile (haystack) with .txt: 1_000.txt
Enter substring to search for (needle): XYZ
        ---Search options---
1 - Linear search to find the first match
2 - Boyer-Moore-Horspool search to find the last match
Which search to use: 1
Search completed in 5 microseconds
Total number of comparisons made 1070
in a file of 1054 characters
Needle: "XYZ" was found at index: 1050

D:\Boris\Documents\UNI\SIAOD\6_2>g++ main.cpp -o main && .\main
Enter name of textfile (haystack) with .txt: 1_000.txt
Enter substring to search for (needle): XY~
        ---Search options---
1 - Linear search to find the first match
2 - Boyer-Moore-Horspool search to find the last match
Which search to use: 1
Search completed in 21 microseconds
Total number of comparisons made 1070
in a file of 1054 characters
Needle: "XY~" was not found in textfile
```

Рисунок 8 – Тестирование запусков 0000 и 0001

```
D:\Boris\Documents\UNI\SIAOD\6_2>g++ main.cpp -o main && .\main
Enter name of textfile (haystack) with .txt: 10_000_000.txt
Enter substring to search for (needle): XYZ
        ---Search options---
1 - Linear search to find the first match
2 - Boyer-Moore-Horspool search to find the last match
Which search to use: 1
Search completed in 33396 microseconds
Total number of comparisons made 10107067
in a file of 10000054 characters
Needle: "XYZ" was found at index: 10000050

D:\Boris\Documents\UNI\SIAOD\6_2>g++ main.cpp -o main && .\main
Enter name of textfile (haystack) with .txt: 10_000_000.txt
Enter substring to search for (needle): XY~
        ---Search options---
1 - Linear search to find the first match
2 - Boyer-Moore-Horspool search to find the last match
Which search to use: 1
Search completed in 40584 microseconds
Total number of comparisons made 10107067
in a file of 10000054 characters
Needle: "XY~" was not found in textfile
```

Рисунок 9 – Тестирование запусков 0010 и 0011

```
D:\Boris\Documents\UNI\SIAOD\6_2>g++ main.cpp -o main && .\main
Enter name of textfile (haystack) with .txt: 1_000.txt
Enter substring to search for (needle): abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
        ---Search options---
1 - Linear search to find the first match
2 - Boyer-Moore-Horspool search to find the last match
Which search to use: 1
Search completed in 8 microseconds
Total number of comparisons made 1065
in a file of 1054 characters
Needle: "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ" was found at index: 1001

D:\Boris\Documents\UNI\SIAOD\6_2>g++ main.cpp -o main && .\main
Enter name of textfile (haystack) with .txt: 1_000.txt
Enter substring to search for (needle): abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXY~
        ---Search options---
1 - Linear search to find the first match
2 - Boyer-Moore-Horspool search to find the last match
Which search to use: 1
Search completed in 9 microseconds
Total number of comparisons made 1065
in a file of 1054 characters
Needle: "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXY~" was not found in textfile
```

Рисунок 10 – Тестирование запусков 0100 и 0101

```
D:\Boris\Documents\UNI\SIAOD\6_2>g++ main.cpp -o main && .\main
Enter name of textfile (haystack) with .txt: 10_000_000.txt
Enter substring to search for (needle): abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
        ---Search options---
1 - Linear search to find the first match
2 - Boyer-Moore-Horspool search to find the last match
Which search to use: 1
Search completed in 39996 microseconds
Total number of comparisons made 10106859
in a file of 10000054 characters
Needle: "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ" was found at index: 10000001

D:\Boris\Documents\UNI\SIAOD\6_2>g++ main.cpp -o main && .\main
Enter name of textfile (haystack) with .txt: 10_000_000.txt
Enter substring to search for (needle): abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXY~
        ---Search options---
1 - Linear search to find the first match
2 - Boyer-Moore-Horspool search to find the last match
Which search to use: 1
Search completed in 32234 microseconds
Total number of comparisons made 10106859
in a file of 10000054 characters
Needle: "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXY~" was not found in textfile
```

Рисунок 11 – Тестирование запусков 0110 и 0111

```
D:\Boris\Documents\UNI\SIAOD\6_2>g++ main.cpp -o main && .\main
Enter name of textfile (haystack) with .txt: 1_000.txt
Enter substring to search for (needle): XYZ
        ---Search options---
1 - Linear search to find the first match
2 - Boyer-Moore-Horspool search to find the last match
Which search to use: 2
Search completed in 9 microseconds
Total number of comparisons made 363
in a file of 1054 characters
Needle: "XYZ" was found at index: 1050

D:\Boris\Documents\UNI\SIAOD\6_2>g++ main.cpp -o main && .\main
Enter name of textfile (haystack) with .txt: 1_000.txt
Enter substring to search for (needle): XY~
        ---Search options---
1 - Linear search to find the first match
2 - Boyer-Moore-Horspool search to find the last match
Which search to use: 2
Search completed in 7 microseconds
Total number of comparisons made 355
in a file of 1054 characters
Needle: "XY~" was not found in textfile
```

Рисунок 12 – Тестирование запусков 1000 и 1001

```
D:\Boris\Documents\UNI\SIAOD\6_2>g++ main.cpp -o main && .\main
Enter name of textfile (haystack) with .txt: 10_000_000.txt
Enter substring to search for (needle): XYZ
        ---Search options---
1 - Linear search to find the first match
2 - Boyer-Moore-Horspool search to find the last match
Which search to use: 2
Search completed in 26747 microseconds
Total number of comparisons made 3405672
in a file of 10000054 characters
Needle: "XYZ" was found at index: 10000050

D:\Boris\Documents\UNI\SIAOD\6_2>g++ main.cpp -o main && .\main
Enter name of textfile (haystack) with .txt: 10_000_000.txt
Enter substring to search for (needle): XY~
        ---Search options---
1 - Linear search to find the first match
2 - Boyer-Moore-Horspool search to find the last match
Which search to use: 2
Search completed in 34500 microseconds
Total number of comparisons made 3369135
in a file of 10000054 characters
Needle: "XY~" was not found in textfile
```

Рисунок 13 – Тестирование запусков 1010 и 1011

```
D:\Boris\Documents\UNI\SIAOD\6_2>g++ main.cpp -o main && .\main
Enter name of textfile (haystack) with .txt: 1_000.txt
Enter substring to search for (needle): abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
        ---Search options---
1 - Linear search to find the first match
2 - Boyer-Moore-Horspool search to find the last match
Which search to use: 2
Search completed in 15 microseconds
Total number of comparisons made 77
in a file of 1054 characters
Needle: "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ" was found at index: 1001

D:\Boris\Documents\UNI\SIAOD\6_2>g++ main.cpp -o main && .\main
Enter name of textfile (haystack) with .txt: 1_000.txt
Enter substring to search for (needle): abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXY~
        ---Search options---
1 - Linear search to find the first match
2 - Boyer-Moore-Horspool search to find the last match
Which search to use: 2
Search completed in 3 microseconds
Total number of comparisons made 25
in a file of 1054 characters
Needle: "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXY~" was not found in textfile
```

Рисунок 14 – Тестирование запусков 1100 и 1101

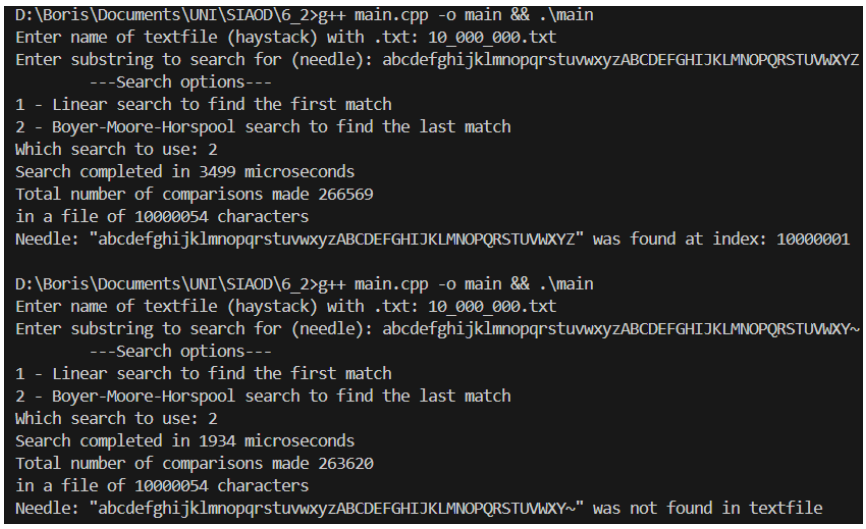

Рисунок 15 – Тестирование запусков 1110 и 1111

Тестирование показало, что программа работает корректно. Результаты тестирования занесём в таблицы 1, 2, 3, 4 для успешного и неуспешного прямого поиска, и для успешного и неуспешного поиска алгоритмом Бойера-Мура-Хорспула соответственно.

Таблица 1 — тестирование успешного прямого поиска

Длина текста / Длина образца	3 символа	52 символа
1_000 символов	5 мкс / 1070 сравнений	8 мкс / 1065 сравнений
10_000_000 символов	33396 мкс / 10107067 сравнений	39996 мкс / 10106859 сравнений

Таблица 2 — тестирование неуспешного прямого поиска

Длина текста / Длина образца	3 символа	52 символа
1_000 символов	21 мкс / 1070 сравнений	9 мкс / 1065 сравнений
10_000_000 символов	40584 мкс / 10107067 сравнений	32234 мкс / 10106859 сравнений

Таблица 3 — тестирование успешного поиска Бойера-Мура-Хорспула

Длина текста / Длина образца	3 символа	52 символа
1_000 символов	9 мкс / 363 сравнений	15 мкс / 77 сравнений
10_000_000 символов	26747 мкс / 3405672 сравнений	3499 мкс / 266569 сравнений

Таблица 4 — тестирование неуспешного поиска Бойера-Мура-Хорспула

Длина текста / Длина образца	3 символа	52 символа
1_000 символов	7 мкс / 355 сравнений	3 мкс / 25 сравнений
10_000_000 символов	34500 мкс / 3369135 сравнений	1934 мкс / 263620 сравнений

Получили что для текстового файла длины N, заполненного произвольными символами, и образца длины M, сложность алгоритма прямого поиска равна O(N). Сложность алгоритма Бойера-Мура-Хорспула же будет равна O(N/M). В худшем случае оба алгоритма имеют сложность O(N*M).

ВЫВОД

В результате выполнения работы были освоены навыки по реализации алгоритмов поиска в тексте, в частности были изучены алгоритм прямого поиска и алгоритм Бойера-Мура-Хорспула. Последний является чаще более эффективным, особенно при работе с длинными текстами и образцами.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Рысин, М. Л. Введение в структуры и алгоритмы обработки данных : учебное пособие / М. Л. Рысин, М. В. Сартаков, М. Б. Туманова. — Москва : РТУ МИРЭА, 2022 — Часть 2 : Поиск в тексте. Нелинейные структуры данных. Кодирование информации. Алгоритмические стратегии — 2022. — 111 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/310826 (дата обращения: 28.09.2024).

2. Документация по языку C++ [Электронный ресурс]. URL: https://docs.microsoft.com/ruru/cpp/cpp/ (дата обращения 28.09.2024).