Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу «Дискретный анализ»

Студент: А. С. Бычков
Преподаватель: Н. К. Макаров
Группа: М8О-201Б
Дата:
Оценка:
Подпись:

Москва, 2024

Лабораторная работа №4

Задача: Необходимо реализовать один из стандартных алгоритмов поиска образцов для указанного алфавита.

Вариант алгоритма: Поиск одного образца при помощи алгоритма Апостолико-Джанкарло.

Вариант алфавита: Числа в диапазоне от 0 до $2^{32} - 1$.

Запрещается реализовывать алгоритмы на алфавитах меньшей размерности, чем указано в задании.

1 Описание

Требуется написать алгоритм Апостолико-Джанкарло. Согласно [1] Апостолико и Джанкарло предложили вариант алгоритма Бойера-Мура, который допускает замечательно простое доказательство линейной оценки наихудшего времени счета. В этом варианте никакой символ из $Text$ не участвует в сравнениях после его первого совпадения с каким-нибудь симовлом из $Pattern$. Отсюда немедленно следует, что число сравнений не превзойдёт $2 * |Text|$. Каждое сравнение даёт либо совпадение, либо несовпадение; последних может быть только m, т.к. при каждом несовпадении происходит ненулевой сдвиг $Pattern'a$, а совпадений - не больше m, т.к. никакой символ из $Text$ не сравнивается после совпадения с символом из P. Так же, остальная работа, совершаемая поверх обычного алгоритма Бойера-Мура выполняется за линейное относительно $|Text|$ время.

По сути, всё отличие алгоритма Апостолико-Джанкарло от алгоритма Бойера-Мура заключается в том, что у нас добавляется ещё один массив M длины $|Text|$. В котором каждое $M[i] = k$ говорит о том, что строка $Text[i - k + 1...i]$ совпадает «как минимум» с суффиксом $Pattern$ длины k. Т.е. на самом деле, совпадение может быть длины больше, чем k. После этого, вместо простого сравнения до первого несовпадения, как это происходит в алгоритме Бойера-Мура мы используем наш массив M для того, логически заключить, что из текущей позиции может совпасть определенное число символов. Этот массив не вычисляется заранее, как массивы L, l, N для алгоритма Бойера-Мура, этот массив вычисляется динамически во время поиска шаблона.

2 Исходный код

Первым делом я реализовал алгоритм Бойера-Мура. Для его реализации требуется препроцессинг вспомогательных массивов:

- Изначально нам нужен массив N, где элемент $N[i]$ говорит о длине максимального суффикса строки $Pattern[1...i]$ совпадающего с суффиксом всей строки $Pattern$.
- Не сложно заметить, что N похож на $Z-$, которая говорит, что $Z[i]$ - максимальная длина префикса строки $Pattern[i...n]$, совпадающего с суффиксом всей строки $Pattern$. Поэтому, напишем реализацию линейного нахождения $Z-$ и после вычисли $Z-$ для перевёрнутого $Pattern$. Получим массив, перевернув который получим искомый массив N
- Теперь можно легко получить массив L, который говорит о том, что $L[i] = j$ - максимальный индекс j меньший n, такой что какой-то суффикс строки $Pattern[1...j]$ равен строке $Pattern[i...n]$. Причём, есть две реализации этого массива: сильная и слабая. Выше описание слабой версии, я же использовал сильную. Сильная версия накладывает ещё одно ограничение: $Pattern[i-1]$ отлично от символа перед суффиксом строки $Pattern[1...j]$. Именно это условие позволяет легко вычислить данный массив через массив N. $L[i]$ - наибольший индекс j меньший n, такой что $N[j] = |Pattern[i..n]| = n - i + 1$.
- Теперь мы можем вычислить и массив l, суть которого заключается в следующем: $l[i]$ - длина наибольшего суффикса строки $Pattern[i...n]$, совпадающего с перффиксом строки $Pattern$. Через массив N так же можно легко вычислить этот массив: $l[i]$ - максимальное $j <= |P[i..n]|$, для которого $N[j] = j$ (это с учётом, что индексация с 1, иначе - $N[j] = j + 1$, так же всё выше тоже для индексации с единицы).
- Кроме этого, нужно ещё написать словарь R, в котором для каждой буквы из $Pattern$ будут храниться индексы её вхождений, начиная с последнего.
- После этого, остаётся лишь сравнивать символы. Если встретили несовпадение - считаем максимум из Правила Плохого Символа и Правила Хорошего Суффикса и делаем сдвиг на указанную величину.

После того, как реализован Бойер-Мур остаётся изменить лишь цикл, в котором происходят сравнения. В этом цикле для каждого положения $pattern_ptr$ и $text_ptr$ будем сравнивать величины $N[pattern_ptr]$ и $M[text_ptr]$ и будем решать, можно ли пропустить какие-то сравнения или нет. На этом заканчиваются различия между Бойером-Муром и Апостолико-Джанкарло.

```cpp
#include <string>
#include <vector>
#include <unordered_map>
#include <algorithm>
#include <iostream>
#include <sstream>

template <typename T>
std::ostream& operator<<(std::ostream& os, const std::vector<T>& v) {
    for (const T& elem : v) {
        os << elem << ' ';
    }
    os << '\n';
    return os;
}

std::vector<ssize_t> naive_(const std::vector<uint32_t>& text, const std::vector<
    uint32_t>& pattern) {
    std::vector<ssize_t> ans;
    ssize_t n = text.size();
    ssize_t m = pattern.size();

    if (m > n) return ans;

    for (ssize_t i = 0; i <= n - m; ++i) {
        bool exist = true;
        for (ssize_t j = 0; j < m; ++j) {
            if (text[i + j] != pattern[j]) {
                exist = false;
                break;
            }
        }
        if (exist) {
            ans.push_back(i);
        }
    }

    return ans;
}


class ApostolicoGiancarlo {
public:
    const std::vector<uint32_t> text;
    const std::vector<uint32_t> pattern;
    const size_t t_sz;
    const size_t p_sz;

    ApostolicoGiancarlo(const std::vector<uint32_t>& text, const std::vector<uint32_t>&
```

```
         pattern)
49       : text(text), pattern(pattern), t_sz(text.size()), p_sz(pattern.size()), N(p_sz
             , 0), l(p_sz, 0), L(p_sz, -1), M(t_sz, 0) {
50           n_func();
51           strong_L_func();
52           strong_l_func();
53           R_func();
54           find_pattern();
55       }
56
57   ApostolicoGiancarlo(const std::vector<uint32_t>&& text, const std::vector<uint32_t
         >&& pattern)
58       : text(text), pattern(pattern), t_sz(text.size()), p_sz(pattern.size()), N(p_sz
             , 0), l(p_sz, 0), L(p_sz, -1), M(t_sz, 0) {
59           n_func();
60           strong_L_func();
61           strong_l_func();
62           R_func();
63           find_pattern();
64       }
65
66   std::vector<ssize_t> get_answer() { return ans; }
67
68 private:
69   std::vector<ssize_t> N;
70   std::vector<ssize_t> l;
71   std::vector<ssize_t> L;
72   std::vector<ssize_t> M;
73   std::unordered_map<char, std::vector<ssize_t>> R;
74   std::vector<ssize_t> ans;
75
76
77   void z_func(const std::vector<uint32_t>& s) {
78       ssize_t l = 0, r = 0;
79
80       N[0] = p_sz;
81
82       for (ssize_t i = 1; i < p_sz; ++i) {
83           if (i < r) N[i] = std::min(r - i, N[i - l]);
84
85           while (i + N[i] < p_sz && s[N[i]] == s[i + N[i]]) ++N[i];
86
87           if (i + N[i] > r) {
88               l = i;
89               r = i + N[i];
90           }
91       }
92
93       #ifdef DEBUG
```

```
94          std::cout << "Z-function for reversed pattern:\n" << N << "\n\n\n" << std::endl
                ;
95          #endif
96      }
97
98      void n_func() {
99          std::vector<uint32_t> reversed_pattern = pattern;
100         std::reverse(reversed_pattern.begin(), reversed_pattern.end());
101         z_func(reversed_pattern);
102         std::reverse(N.begin(), N.end());
103
104         #ifdef DEBUG
105         std::cout << "N-function:\n" << N << "\n\n\n" << std::endl;
106         #endif
107     }
108
109     void strong_L_func() {
110         for (ssize_t j = 0; j < p_sz - 1; ++j) {
111             if (N[j]) {
112                 ssize_t i = p_sz - N[j];
113                 L[i] = j;
114             }
115         }
116
117         #ifdef DEBUG
118         std::cout << "Strong L-function:\n" << L << "\n\n\n" << std::endl;
119         #endif
120     }
121
122     void strong_l_func() {
123         ssize_t prev = 0;
124         for (ssize_t j = 0; j < p_sz; ++j) {
125             ssize_t i = p_sz - j - 1;
126             if (N[j] == j + 1) {
127                 prev = j + 1;
128             }
129             l[i] = prev;
130         }
131
132         #ifdef DEBUG
133         std::cout << "Strong l-function:\n" << l << "\n\n\n" << std::endl;
134         #endif
135     }
136
137     void R_func() {
138         for (ssize_t i = p_sz - 1; i >= 0; --i) {
139             R[pattern[i]].push_back(i);
140         }
141
```

```
142         #ifdef DEBUG
143         std::cout << "R-function: \n";
144         for (const auto& kv : R) {
145             std::cout << "Char: " << kv.first << ", indexes: " << kv.second << std::
                endl;
146         }
147         std::cout << "\n\n\n";
148         #endif
149     }
150
151     ssize_t get_bad_char_shift(ssize_t text_ptr, ssize_t pattern_ptr) {
152         /*
153
154                   ,
155               text[text_ptr]   ,   char_ind < pattern_ptr
156         */
157
158         ssize_t char_ind = -1;
159         ssize_t n = R[text[text_ptr]].size();
160
161         for (ssize_t i = 0; i < n; ++i) { //    ,
162             if (R[text[text_ptr]][i] < pattern_ptr) {
163                 char_ind = R[text[text_ptr]][i];
164                 break;
165             }
166         }
167
168         #ifdef DEBUG
169         std::cout << "    \n";
170         std::cout << "  T[" << text_ptr << "] = \'" << text[text_ptr] << "\'  P[" <<
                pattern_ptr << "] = \'" << pattern[pattern_ptr] << "\'\n";
171         std::cout << "R = [" << R[text[text_ptr]] << "]\n";
172         if (char_ind == -1) std::cout << " \'" << text[text_ptr] << "\',    " <<
                pattern_ptr << "   " << '\n';
173         else std::cout << " \'" << text[text_ptr] << "\',    " << pattern_ptr << "    "
                << char_ind << '\n';
174         std::cout << "    : " << (char_ind == -1 ? pattern_ptr + 1 : pattern_ptr -
                char_ind) << "\n\n\n" << std::endl;
175         #endif
176
177         return char_ind == -1 ? pattern_ptr + 1 : pattern_ptr - char_ind;
178     }
179
180     ssize_t get_good_suffix_shift(ssize_t pattern_ptr) {
181
182         /*
183
184                   :
185                 strong_L_arr[pattern_ptr]  strong_l_arr[pattern_ptr]
```

```
186                 strong_L_arr[pattern_ptr + 1]  strong_l_arr[pattern_ptr + 1]
187               (+1), .. pattern_ptr -    ,
188                   .
189         */
190
191         ssize_t good_suffix_shift = -1;
192
193         if (L[pattern_ptr + 1] != -1) {
194             good_suffix_shift = p_sz - 1 - L[pattern_ptr + 1];
195
196         } else if (L[pattern_ptr + 1] == -1) {
197             good_suffix_shift = p_sz - l[pattern_ptr + 1];
198         }
199
200         #ifdef DEBUG
201         std::cout << "   " << '\n';
202         if (L[pattern_ptr + 1] != -1) {
203             std::cout << " `L[pattern_ptr + 1] != -1`" << '\n';
204             std::cout << ",     (  P),   pattern_ptr`\n";
205         } else if (L[pattern_ptr + 1] == -1) {
206             std::cout << " `L[pattern_ptr + 1] == -1`" << '\n';
207             std::cout << ",     (  P),   pattern_ptr`\n";
208             std::cout << "   ,     P       T,  pattern   \n";
209         } else {
210             std::cout << " !!!     ...\n";
211         }
212         std::cout << "     = " << good_suffix_shift << "\n\n\n" << std::endl;
213         #endif
214         return good_suffix_shift;
215     }
216
217     void find_pattern() {
218         if (p_sz > t_sz) return;
219
220         /*
221               ,
222                   0123456789
223             T: abobaameba
224             P: bbb
225             shift = 6
226         */
227         ssize_t shift = p_sz - 1;
228
229         while (shift < t_sz) {
230             ssize_t pattern_ptr = p_sz - 1;
231             ssize_t text_ptr = shift;
232
233             // while (pattern_ptr >= 0 && text[text_ptr] == pattern[pattern_ptr]) {
234             // --pattern_ptr;
```

```
235             // --text_ptr;
236             // }
237
238             bool pattern_finded = false;
239
240             while (true) {
241
242                 #ifdef DEBUG
243                 std::cout << "M1 = " << M << '\n';
244                 std::cout << "Text PTR = " << text_ptr << " Pattern PTR = " <<
                        pattern_ptr << std::endl;
245                 #endif
246
247                 //  1)
248                 if (M[text_ptr] == 0) {
249
250                     #ifdef DEBUG
251                     std::cout << "  1" << std::endl;
252                     #endif
253
254                     //  1.1)   .
255                     if (text[text_ptr] == pattern[pattern_ptr] && pattern_ptr == 0) {
256                         #ifdef DEBUG
257                         std::cout << "  1.1" << std::endl;
258                         #endif
259
260                         M[shift] = p_sz;
261                         ans.push_back(text_ptr);
262                         shift += p_sz - l[1];
263                         pattern_finded = true;
264                         break;
265
266                     //  1.2)  .
267                     } else if (text[text_ptr] == pattern[pattern_ptr] && pattern_ptr >
                            0) {
268                         #ifdef DEBUG
269                         std::cout << "  1.2" << std::endl;
270                         #endif
271
272                         --text_ptr;
273                         --pattern_ptr;
274
275                     //  1.3)  .
276                     } else {
277                         #ifdef DEBUG
278                         std::cout << "  1.3" << std::endl;
279                         #endif
280
281                         M[shift] = shift - text_ptr;
```

```
282                     break;
283                 }
284
285             //  2)
286             } else if (M[text_ptr] < N[pattern_ptr]) {
287                 #ifdef DEBUG
288                 std::cout << "  2" << std::endl;
289                 #endif
290
291                 //   .
292                 //  , pattern_ptr -= M[text_ptr]  , ..
293                 // text_ptr   .
294                 pattern_ptr -= M[text_ptr];
295                 text_ptr -= M[text_ptr];
296
297             //  3)
298             } else if (M[text_ptr] >= N[pattern_ptr] && N[pattern_ptr] ==
                    pattern_ptr + 1) {
299                 #ifdef DEBUG
300                 std::cout << "  3" << std::endl;
301                 #endif
302
303
304                 M[shift] = shift - text_ptr;
305                 ans.push_back(shift - p_sz + 1);
306                 shift += p_sz - l[1];
307                 pattern_finded = true;
308                 break;
309
310             //  4)
311             //   ,   N[pattern_ptr] = 0
312             //     text[text_ptr] != pattern[pattern_ptr]    (  )
313             //   pattern_ptr = p_sz - 1,      , .. N[p_sz - 1] = p_sz
314             } else if (M[text_ptr] > N[pattern_ptr] && N[pattern_ptr] < pattern_ptr
                    + 1) {
315                 // ,   ,     .
316                 #ifdef DEBUG
317                 std::cout << "  4" << std::endl;
318                 #endif
319
320                 M[shift] = shift - text_ptr;
321                 text_ptr -= N[pattern_ptr];
322                 pattern_ptr -= N[pattern_ptr];
323                 break;
324
325             //  5)
326             } else if (M[text_ptr] == N[pattern_ptr] && 0 < N[pattern_ptr] && N[
                    pattern_ptr] < pattern_ptr + 1) {
327                 #ifdef DEBUG
```

```
328                     std::cout << "  5" << std::endl;
329                     #endif
330
331                     //   .
332                     //  , pattern_ptr -= M[text_ptr]  , ..
333                     // text_ptr   .
334                     pattern_ptr -= M[text_ptr];
335                     text_ptr -= M[text_ptr];
336
337                 // -   .
338                 } else {
339                     std::cout << " ..." << std::endl;
340                 }
341             }
342
343
344             if (pattern_finded) continue;
345
346             #ifdef DEBUG
347             std::cout << "M2 = " << M << '\n';
348             std::cout << "Text PTR = " << text_ptr << " Pattern PTR = " << pattern_ptr
                    << std::endl;
349             #endif
350
351
352             // ,
353             if (pattern_ptr == p_sz - 1) {
354                 shift += 1;
355                 continue;
356             }
357
358
359             ssize_t bad_char_shift = get_bad_char_shift(text_ptr, pattern_ptr);
360             ssize_t good_suffix_shift = get_good_suffix_shift(pattern_ptr);
361             shift += std::max(bad_char_shift, good_suffix_shift);
362
363             #ifdef DEBUG
364             std::cout << "max(, ) = " << std::max(bad_char_shift, good_suffix_shift) <<
                     '\n';
365             std::cout << " shift = " << shift << "\n\n\n" << std::endl;
366             #endif
367
368         }
369     }
370 };
371
372
373 int main() {
374
```

```
375     #ifndef BENCHMARK
376
377     std::vector<uint32_t> text, pattern;
378     std::vector<size_t> lines_length;
379
380     size_t current_line_length = 0;
381     uint32_t word;
382
383
384     std::string line;
385     std::getline(std::cin, line);
386     std::stringstream ss(line);
387     while (ss >> word) {
388         pattern.push_back(word);
389     }
390
391     while (std::getline(std::cin, line)) {
392         std::stringstream ss(line);
393         while (ss >> word) {
394             text.push_back(word);
395             ++current_line_length;
396         }
397         lines_length.push_back(current_line_length);
398         current_line_length = 0;
399     }
400
401     ApostolicoGiancarlo ag(std::move(text), std::move(pattern));
402
403     std::vector<ssize_t> ans = ag.get_answer();
404     for (auto& el : ans) ++el; //    1-
405
406     size_t words_before_current_line_exclude_current_line = 0;
407     size_t words_before_current_line_include_current_line = 0;
408     size_t current_ans_ind = 0;
409
410     for (size_t i = 0; i < lines_length.size(); ++i) {
411         words_before_current_line_include_current_line += lines_length[i];
412         while (current_ans_ind < ans.size() && ans[current_ans_ind] <=
                words_before_current_line_include_current_line) {
413             std::cout << i + 1 << ", " << ans[current_ans_ind++] -
                    words_before_current_line_exclude_current_line << '\n';
414         }
415         words_before_current_line_exclude_current_line =
                words_before_current_line_include_current_line;
416     }
417
418     #else
419     std::srand(time(NULL));
420
```

```
421     for (int i = 0; i < 1000; ++i) {
422 
423         int n = std::rand() % 50 + 5;
424         int m = std::rand() % 5 + 1;
425         std::string text;
426         std::string pattern;
427         for (int i = 0; i < n; ++i) text.push_back('a' + (std::rand() % 5));
428         for (int i = 0; i < m; ++i) pattern.push_back('a' + (std::rand() % 5));
429 
430         std::vector<ssize_t> naive = naive_(text, pattern);
431 
432         ApostolicoGiancarlo ag(text, pattern);
433 
434         std::vector<ssize_t> apostolico = ag.get_answer();
435 
436         if (naive != apostolico) {
437             std::cout << "ERROR on " << i + 1 << "!!!!" << std::endl;
438             std::cout << "Text = " << text << '\n' << "Pattern = " << pattern << std::
                endl;
439             std::cout << "Naive: " << naive << '\n' << "Apostolico-Giancarlo: " <<
                apostolico << std::endl;
440             break;
441         } else {
442             std::cout << "Ok on " << i + 1 << std::endl;
443             std::cout << "Len = " << apostolico.size() << std::endl;
444             // std::cout << "Text = " << text << '\n' << "Pattern = " << pattern << std
                ::endl;
445             // std::cout << "Naive: " << naive << '\n' << "Boyer: " << boyer << std::
                endl;
446         }
447     }
448     #endif
449 
450     return 0;
451 }
```

Методы класса «ApostolicoGiancarlo».

main.cpp	
ApostolicoGiancarlo(const std::vector<uint32_t>& text, const std::vector<uint32_t>& pattern)	Конструктор от текста и шаблона
ApostolicoGiancarlo(const std::vector<uint32_t>&& text, const std::vector<uint32_t>&& pattern)	Конструктор от текста и шаблона, но и текст и шаблон - rvalue ссылки
std::vector<ssize$_t$ > $get_answer()$	Функция, которая возвращает массив индексов - ответов

void z_func(const std::vector<uint32_t>& s)	Вычисление Z-массива для перевернутой строки
void n_func()	Вычисление N-массива
void strong$_{L_func()}$	Вычисление сильного L-массива
void strong_l_func()	Вычисление сильного l-массива
void R$_{func}$()	Вычисление R-словаря
ssize_t get_bad_char_shift(ssize_t text_ptr, ssize_t pattern_ptr)	Определение сдвига по Правилу Плохого Символа
ssize_t get_good_suffix_shift(ssize_t pattern_ptr)	Определение сдвига по Правилу Хорошего Суффикса
void find_pattern()	Функция, которая ищет шаблон в тексте
std::ostream& operator«(std::ostream& os, const std::vector<T>& v)	Шаблонная функция для вывода вектора

3 Консоль

```
>g++ main.cpp -std=c++20
>cat sample_input.txt
11 45 11 45 90
0011 45 011 0045 11 45 90    11
45 11 45 90
11 0045 0011 45 90%
>./a.out <sample_input.txt
1,3
1,8
3,1
```

4 Тест производительности

Тест производительности представляет из себя следующее: поиск строки в тексте. Алфавит как и в задании - числа в диапазоне от 0 до $2^{32} - 1$. Тестирование производится на тексте длиной 1 миллион символов. Длина паттерна варьируется, я ее задаю запуском *python* скрипта, указывая левую и правую границы длины.

```
>g++ benchmark.cpp
>python3 test-gen.py 3 6
>./a.out
Apostolico-Giancarlo 32ms
std::search 16ms
>./a.out
Apostolico-Giancarlo 32ms
std::search 16ms
>diff benchmark_out_ag.txt benchmark_out_find.txt
>python3 test-gen.py 1000 2000
>./a.out
Apostolico-Giancarlo 20ms
std::search 16ms
>diff benchmark_out_ag.txt benchmark_out_find.txt
>./a.out
Apostolico-Giancarlo 21ms
std::search 17ms
>./a.out
Apostolico-Giancarlo 19ms
std::search 16ms
>diff benchmark_out_ag.txt benchmark_out_find.txt
>python3 test-gen.py 100 200
>./a.out
Apostolico-Giancarlo 23ms
std::search 16ms
>diff benchmark_out_ag.txt benchmark_out_find.txt
>python3 test-gen.py 300 600
>./a.out
Apostolico-Giancarlo 21ms
std::search 16ms
>./a.out
Apostolico-Giancarlo 21ms
std::search 17ms
>./a.out
```

```
Apostolico-Giancarlo 15ms
std::search 16ms
```

Но, если урезать мощность алфавита для 3-х символов:

```
>g++ benchmark.cpp
>python3 test-gen.py 1000 2000
>./a.out
Apostolico-Giancarlo 16ms
std::search 28ms
>python3 test-gen.py 3 6
>./a.out
Apostolico-Giancarlo 55ms
std::search 40ms
>python3 test-gen.py 100 200
>./a.out
Apostolico-Giancarlo 27ms
std::search 28ms
```

Как видно, на случайных данных данный алгоритм может быть хуже наивного. Моё предположение, почему так заключается в том, что из-за M-массива, к которому мы постоянно обращаемся нарушается локальность кэша, что делает долгим выполнение некоторых операций. Однако, когда происходит много совпадений: маленький алфавит или же реальный текст, в котором может быть много слов похожих на образец, алгоритм довольно эффективен.

5 Выводы

Эта лабораторная работа далась мне тяжелее всего. Всё из-за того, что очень мало информации по таким алгоритмам. Если по Красно-Чёрным деревьям информации уйма, то по алгоритму Бойера-Мура её сильно меньше, а по Апостолико-Джанкарло по сути один источник - Гастфилд. Поэтому, приходилось долго вчитываться и понимать, что имеет в виду автор.

Благодаря данной лабораторной работе я больше узнал про поиск в строках, и прям отлично понял Z-функцию, которая очень полезна из-за её простоты. Алгоритм Бойера-Мура так же крайне понятен, однако, писать его куда тяжелее, чем Z-функцию. Однако, с Апостолико-Джанкарло чуть хуже из-за его скудного описания в Гастфилде, без единого примера. А так же не понятен момент, зачем вообще нужен Апостолико-Джанкарло, если единственное его преимущество над Бойером-Муром - простота доказательства линейности? Из-за этой простоты мы жертвует $O(|Text|)$ дополнительной памяти, в то время, как Бойер-Мур использует всего $O(|Pattern|)$ памяти! При том, есть та же Z-функция, в которой смысловой нагрузки - 3 строки, её линейность - очевидна, и использует она те же $O(|Text| + |Pattern|)$ памяти, что и Апостолико-Джанкарло.

Список литературы

[1] Гастфилд Дэн *Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология* — СПб.: «Невский Диалект», 2003. Перевод с английского: И. В. Романовский. — 654 с. (ISBN 5-7940-0103-8 (рус.))

[2] *Алгоритм Бойера-Мура - ИТМО.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Бойера-Мура` (дата обращения: 26.05.2024).

[3] *Алгоритм Бойера-Мура - Википедия.*
URL: `https://ru.wikipedia.org/wiki/Алгоритм_Бойера_-_Мура` (дата обращения: 26.05.2024).