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. О. Знай
Преподаватель: С. А. Михайлова
Группа: М8О-201Б-21

Дата: 15.05.2023
Оценка:
Подпись:

Москва, 2023

Лабораторная работа №4

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

Вариант алгоритма: Поиск одного образца при помощи алгоритма Бойера-Мура.

Вариант алфавита: Числа в диапазоне от 0 до 232 - 1.

1 Описание

Алгоритм Бойера-Мура используется для поиска подстроки в строке и обладает следующими особенностями: считывание символов справа налево, правило плохого символа, правило хорошего суффикса.

2 Описание программы

Правило плохого символа: в процессе сравнения символов паттерна и текста в случае несовпадения какого-то символа мы можем обратиться к структуре, которая хранит в себе крайние правые позиции символовов P. Таким образом, если мы найдем в структуре этот несовпавший символ, то сдвинем P так, чтобы найденный символ оказался в нужной позиции. Если же его нет в P, то сдвигаем за этот символ.

Сильное правило хорошего суффикса: в процессе сравнения паттерна и текста до несовпадения символов у нас уже могли быть успешные проверки. Тогда в случае несовпадения надо найти в P такую крайнюю правую подстроку, которая бы совпадала с успешным проверенным суффиксом P и символ слева от нее различался с символом слева от суффикса, и сделать сдвиг, подставив эту подстроку вместо суффикса. Если же такой подстроки нет, то попробуем найти такой суффикс у проверенного суффикса, который был бы равен префиксу P, и сделаем соответствующий сдвиг. В случае вхождения паттерна в текст сдвиг определяется путем вычесления максимального суффикса подстроки P[1..n-1], который равен префиксу всего паттерна. Его надо вычесть из длины паттерна.

Конечный сдвиг определяется путем выбора максимального из двух описанных выше эвристик.

Алгоритм Бойера-Мура работает за линейную сложность, но только в случае совместного использования этих двух эвристик и некоторых улучшений: Z-функция с построением за линейное время и из нее вытекающие N, L' и l функции, которые соответственно тоже строятся за линейное время.

[1]

Код

```
#include <string.h>
#include <map>
#include <vector>
#include <iostream>
#include <algorithm>

void readPattern (std::vector<unsigned int>& pattern) {
  std::string number;
  char c = ' ';

  while (true) {
    c = getchar();

    if (c == ' ') {
      if(number.size() != 0){
        pattern.push_back(std::stoul(number));
        number.clear();
      }
    }
    else if (c == '\n'){
      if(number.size() != 0){
        pattern.push_back(std::stoul(number));
        number.clear();
      }
      break;
    }
    else if (c >= '0' && c <= '9') {
      number.push_back(c);
    }
  }
}

void readText (std::vector<unsigned int>& text, std::vector<std::pair<unsigned int,
    unsigned int> >& positionsInText) {
  std::string number;
  unsigned int rowNumber = 1;
  unsigned int wordNumber = 0;
  char c = ' ';

  while (true) {
    c = getchar();

    if (c == ' ') {
      if(number.size() != 0){
        wordNumber++;
        positionsInText.emplace_back(rowNumber, wordNumber);
        text.push_back(std::stoul(number));
        number.clear();
```

```
48        }
49      }
50      else if (c == '\n'){
51        if(number.size() != 0){
52          wordNumber++;
53          positionsInText.emplace_back(rowNumber, wordNumber);
54          text.push_back(std::stoul(number));
55          number.clear();
56        }
57        wordNumber = 0;
58        rowNumber++;
59      }
60      else if (c == EOF){
61        if(number.size() != 0){
62          wordNumber++;
63          positionsInText.emplace_back(rowNumber, wordNumber);
64          text.push_back(std::stoul(number));
65        }
66        break;
67      }
68      else if (c >= '0' && c <= '9') {
69        number.push_back(c);
70      }
71    }
72  }
73
74  void badChar(std::map<unsigned int, unsigned int>& badCharMap ,std::vector<unsigned
        int>& pattern){
75    for (int i = 0; i < pattern.size(); i++){
76      badCharMap[pattern[i]] = i;
77    }
78  }
79
80  long long badCharShift(std::map<unsigned int, unsigned int>& badCharMap, unsigned int
        number, long long position){
81    if (badCharMap.find(number) == badCharMap.end()){
82      return position + 1;
83    }
84    return position - badCharMap[number];
85  }
86
87  std::vector<int> zFunction(std::vector<unsigned int>& pattern){
88    std::vector<int> z(pattern.size());
89
90    for (int i = 1, l = 0, r = 0; i < pattern.size(); i++){
91      if (i <= r){
92        z[i] = std::min(r-i+1, z[i-l]);
93      }
94      while (i + z[i] < pattern.size() && pattern[i + z[i]] == pattern[z[i]]){
```

```
95        ++z[i];
96      }
97      if (i + z[i] - 1 > r){
98        l = i;
99        r = i + z[i] - 1;
100     }
101   }
102
103   return z;
104 }
105
106 std::vector<int> nFunction(std::vector<unsigned int>& pattern){
107   std::vector<unsigned int> reversedPattern = pattern;
108   std::reverse(reversedPattern.begin(), reversedPattern.end());
109   std::vector<int> n(pattern.size());
110   std::vector<int> z = zFunction(reversedPattern);
111   std::reverse(z.begin(), z.end());
112
113   n = z;
114
115   return n;
116 }
117
118 std::vector<int> LFunction(std::vector<unsigned int>& pattern){
119   std::vector<int> L(pattern.size());
120   std::vector<int> n = nFunction(pattern);
121
122   for (int i = 0; i < pattern.size() - 1; i++){
123     int j = pattern.size() - n[i];
124     if (j < pattern.size()) {
125       L[j] = i;
126     }
127   }
128   return L;
129 }
130
131 std::vector<int> lFunction(std::vector<unsigned int>& pattern){
132   std::vector<int> l(pattern.size() + 1);
133   std::vector<int> n = nFunction(pattern);
134
135   for (int i = pattern.size() - 1; i >= 0; i--){
136     int j = pattern.size() - i;
137
138     if (n[j - 1] == j) {
139       l[i] = j;
140     }
141     else {
142       l[i] = l[i + 1];
143     }
```

```
144   }
145   return l;
146 }
147
148 long long goodSufShift(long long position, std::vector<int>& L, std::vector<int>& l,
        size_t size){
149   if (position == size - 1){
150     return 1;
151   }
152   if (L[position + 1] != 0) {
153     return size - L[position + 1] - 1;
154   }
155   return size - l[position + 1];
156 }
157
158 long long successShift(size_t size, std::vector<int>& l){
159   return size - l[1];
160 }
161
162 int main(int argc, char const *argv[])
163 {
164   std::ios_base::sync_with_stdio(false);
165   std::cin.tie(nullptr);
166
167   std::vector<unsigned int> pattern;
168   readPattern(pattern);
169   std::vector<unsigned int> text;
170   std::vector<std::pair<unsigned int, unsigned int> > positionsInText;
171   readText(text, positionsInText);
172   std::map<unsigned int, unsigned int> badCharMap;
173   badChar(badCharMap, pattern);
174   std::vector<int> L = LFunction(pattern);
175   std::vector<int> l = lFunction(pattern);
176
177   long long curPosInText = pattern.size() - 1;
178
179   while (curPosInText < text.size()) {
180     long long textIterator = curPosInText;
181     long long patternIterator = pattern.size() - 1;
182
183     while (patternIterator >= 0 && pattern[patternIterator] == text[textIterator]){
184       patternIterator--;
185       textIterator--;
186     }
187     if (patternIterator == -1) {
188       std::cout << positionsInText[curPosInText - pattern.size() + 1].first << ", " <<
            positionsInText[curPosInText - pattern.size() + 1].second << '\n';
189       curPosInText += successShift(pattern.size(), l);
190     }
```

```
191     if (patternIterator != -1){
192       curPosInText += std::max(badCharShift(badCharMap, text[textIterator],
              patternIterator), goodSufShift(patternIterator, L, l, pattern.size()));
193     }
194   }
195
196   return 0;
197 }
```

3 Консоль

```
artemznaj@MacBook-Air-Artem lab4 % ./lab4
11 45 11 45 90
0011 45 011 0045 11 45 90    11
45 11 45 90
1,3
1,8
artemznaj@MacBook-Air-Artem lab4 % ./lab4
098789    00007788 88 877 88
098767787 9876 7887 678 098776  09876 98
998878 7878 78 787 9 987 080 0 867 0 78
artemznaj@MacBook-Air-Artem lab4 % ./lab4
00067 78 0090
00000 00066 0067   78 00090   09878 067 0079
0067    78 90
1,3
2,1
```

4 Тест производительности

Я написал алгоритм наивного поиска подстроки в строке. Сгенерировал два теста: в 1 есть вхождения паттерна в текст, во 2 - нет.

Результаты работы двух алгоритмов на этих тестах:

Тест 1

artemznaj@MacBook-Air-Artem lab4 Time: 0.21362 seconds

artemznaj@MacBook-Air-Artem lab4 Time: 0.492431 seconds

Тест 2

artemznaj@MacBook-Air-Artem lab4 Time: 0.0831217 seconds

artemznaj@MacBook-Air-Artem lab4 Time: 0.210024 seconds

По результатам тестов видно, что реализованный мной алгоритм Бойера-Мура работает значительно быстрее, чем наивный алгоритм поиска.

Сложность:

- БМ - O(m + n)
- Наивный - O(m*n)

5 Выводы

Алгоритм Бойера-Мура поиска подстроки в строке — очень эффективный алгоритм поиска. Благодаря двум эвристикам (правила плохого символа и хорошего суффикса) в случае несовпадения алгоритм делает большие сдвиги. Благодаря улучшениям: линейным z, n, L и l функциям достигается линейная сложность даже в наихудшем варианте.

Список литературы

[1] String Algorithms and Algrorithms in Computational Biology - Gusfield