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: М. А. Инютин
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б-19
Дата:
Оценка:
Подпись:

Москва, 2020

Лабораторная работа №4

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

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

Вариант алгоритма: Поиск одного образца при помощи алгоритма Кнута-Морриса-Пратта.

Вариант алфавита: Слова не более 16 знаков латинского алфавита (регистронезависимые)

1 Описание

Требуется реализовать алгоритм Кнута-Морриса-Пратта для поиска подстроки в строке. Учитывая, что алфавит состоит из регистронезависимых слов не более 16 знаков, нужно уметь правильно представлять переводы строки, пробелы и табуляции.

Согласно [1], алгоритм Кнута-Морриса-Пратта прикладывает образец к тексту и начинает делать сравнение с левого конца. В случае полного совпадения, было найдено вхождение, сдвигаем образец на один символ вправо. Если же есть несовпадения, то мы делаем сдвиг по особому правилу, в отличии от алгоритма наивного поиска, который всегда сдвигает на один символ.

Для каждой позиции i определим $sp_i(P)$ как длину наибольшего собственного суффикса $P[1..i]$, который совпадает с префиксом P, при чём символы в позициях $i+1$ и $sp_i(P)+1$ различны.

Для каждой позиции i, определим $Z_i(P)$ как длину префикса строки $P[i..|P|]$, который совпадает с префиксом P. Причём $Z_0(P)$ принятно считать равным 0. Набор таких значений называется Z-функцией строки P. Z-функция является известным алгоритмом и может быть вычислена за линейное время от длины строки, как описано в [3].

Значения $Z_j(P)$ соответствует такому $sp_i(P)$, что $i = j + Z_j(P) - 1$. Таким образом вычисление всех $sp_i(P)$ имеет сложность $O(n)$, где n — длина образца.

Будем делать сдвиг, используя вычисленное в каждой позиции значение $sp_i(P)$. Если при сравнении было найдено несовпадение в позиции $i+1$, то мы можем сделать сдвиг на $i - sp_i(P)$, не теряя вхождений.

Алгоритм Кнута-Морисса-Пратта сравнивает каждый символ не более двух раз, то есть совершает не более $2 * m$ сравнений символов, где m — длина текста. Учитывая сложность препроцессинга, асимптотика алгоритма $O(n + m)$.

2 Исходный код

Выполнение работы разобьём на следующие шаги:

1. Реализация вспомогательных структур и функций.
2. Реализация вычисления Z-функция строки.
3. Реализация алгоритма Кнута-Морриса-Пратта с использованием Z-функции.
4. Реализация правильного ввода данных, поиска подстроки в строке.
5. Тесты производительности, сравнение с наивным поиском и классическим алгоритмом Кнута-Морриса-Пратта.

Входной файл содержит слова латинского алфавита. Для вывода нужно знать номер строки, в которой находится слово, и номер слова в строке. Для этого создадим вспомогательную структуру $TWord$. $ZFunction$ возвращает Z-функцию строки. StrongPrefixFunction — функция вычисления sp_i. $KMPStrong$ — реализация алгоритма Кнута-Морриса-Пратта.

```
1  #ifndef SEARCH_HPP
2  #define SEARCH_HPP
3
4  #include <algorithm>
5  #include <vector>
6
7  const unsigned short MAX_WORD_SIZE = 16;
8
9  struct TWord {
10   char Word[MAX_WORD_SIZE];
11   unsigned int StringId, WordId;
12 };
13
14 std::vector<unsigned int> ZFunction(const std::vector<TWord> & s);
15 std::vector<unsigned int> StrongPrefixFunction(const std::vector<TWord> & s);
16 std::vector<unsigned int> KMPStrong(const std::vector<TWord> & pattern, const std::
       vector<TWord> & text);
17
18 #endif /* SEARCH_HPP */
```

Вспомогательные операторы для типа $TWord$ и реализация функций.

```cpp
#include "search.hpp"

bool operator == (const TWord & lhs, const TWord & rhs) {
  for (unsigned short i = 0; i < MAX_WORD_SIZE; ++i) {
    if (lhs.Word[i] != rhs.Word[i]) {
      return false;
    }
  }
  return true;
}

bool operator != (const TWord & lhs, const TWord & rhs) {
  return !(lhs == rhs);
}

std::vector<unsigned int> ZFunction(const std::vector<TWord> & s) {
  unsigned int n = s.size();
  std::vector<unsigned int> z(n);
  unsigned int l = 0, r = 0;
  for (unsigned int i = 1; i < n; ++i) {
    if (i <= r) {
      z[i] = std::min(z[i - l], r - i);
    }
    while (i + z[i] < n and s[i + z[i]] == s[z[i]]) {
      ++z[i];
    }
    if (i + z[i] > r) {
      l = i;
      r = i + z[i];
    }
  }
  return z;
}

std::vector<unsigned int> StrongPrefixFunction(const std::vector<TWord> & s) {
  std::vector<unsigned int> z = ZFunction(s);
  unsigned int n = s.size();
  std::vector<unsigned int> sp(n);
  for (unsigned int i = n - 1; i > 0; --i) {
    sp[i + z[i] - 1] = z[i];
  }
  return sp;
}

std::vector<unsigned int> KMPStrong(const std::vector<TWord> & pattern, const std::
    vector<TWord> & text) {
  std::vector<unsigned int> p = StrongPrefixFunction(pattern);
  unsigned int m = pattern.size();
```

```
48    unsigned int n = text.size();
49    unsigned int i = 0;
50    std::vector<unsigned int> ans;
51    if (m > n) {
52      return ans;
53    }
54    while (i < n - m + 1) {
55      unsigned int j = 0;
56      while (j < m and pattern[j] == text[i + j]) {
57        ++j;
58      }
59      if (j == m) {
60        ans.push_back(i);
61      } else {
62        if (j > 0 and j > p[j - 1]) {
63          i = i + j - p[j - 1] - 1;
64        }
65      }
66      ++i;
67    }
68    return ans;
69  }
```

Функция *Clear* нужна для очистки слова. Будем считывать входные данные посимвольно и постепенно увеличить счётчики номера строки и слова. Если мы впервые встретили перевод строки, то следует поменять местами текст и образец. Так все последующие слова будут сохранены в текст.

```
1   #include <cstdio>
2   #include "search.hpp"
3
4   void Clear(char arr[MAX_WORD_SIZE]) {
5     for (unsigned short i = 0; i < MAX_WORD_SIZE; ++i) {
6       arr[i] = 0;
7     }
8   }
9
10  int main() {
11    bool flagPatternText = 1;
12    std::vector<TWord> pattern;
13    std::vector<TWord> text;
14    TWord cur;
15    Clear(cur.Word);
16    char c = getchar();
17    unsigned short j = 0;
18    while (c > 0) {
19      if (c == '\n') {
20        if (j > 0) {
21          text.push_back(cur);
22          Clear(cur.Word);
```

```
23        j = 0;
24      }
25      ++cur.StringId;
26      if (flagPatternText) {
27        std::swap(pattern, text);
28        flagPatternText = 0;
29        cur.StringId = 1;
30      }
31      cur.WordId = 1;
32    } else if (c == '\t' or c == ' ') {
33      if (j > 0) {
34        text.push_back(cur);
35        Clear(cur.Word);
36        j = 0;
37        ++cur.WordId;
38      }
39    } else {
40      if ('A' <= c and c <= 'Z') {
41        c = c + 'a' - 'A';
42      }
43      cur.Word[j] = c;
44      ++j;
45    }
46      c = getchar();
47  }
48  if (j > 0) {
49    text.push_back(cur);
50  }
51  std::vector<unsigned int> ans = KMPStrong(pattern, text);
52  for (const unsigned int & id : ans) {
53    printf("%d, %d\n", text[id].StringId, text[id].WordId);
54  }
55 }
```

3 Консоль

```
engineerxl@engineerxl:~/Study/DA/lab4$ cat sample.txt
cat dog cat dog bird
CAT dog CaT Dog Cat DOG bird CAT
dog cat dog bird
engineerxl@engineerxl:~/Study/DA/lab4$ make
g++ -O2 -g -pedantic -std=c++17 -Wall -Werror -c search.cpp
g++ -O2 -g -pedantic -std=c++17 -Wall -Werror -c main.cpp
g++ -O2 -g -pedantic -std=c++17 -Wall -Werror search.o main.o -o solution
engineerxl@engineerxl:~/Study/DA/lab4$ ./solution <sample.txt
1,3
1,8
```

4 Тест производительности

Реализованный алгоритм Кнута-Морриса-Пратта с использованием Z-функции сравнивается с наивным алгоритмом поиска и классическим алгоритмом Кнута-Морриса-Пратта через префикс-функцию.

Тесты состоят из 10^3, 10^4, 10^5 и $2 * 10^6$ слов.

Длины образцов 10, 25, 50 и 100 соответственно.

```
engineerxl@engineerxl:~/Study/DA/lab4/benchmark$ make
g++ -O2 -pedantic -std=c++17 -Wall -Werror -c search.cpp
g++ -O2 -pedantic -std=c++17 -Wall -Werror -c main.cpp
g++ -O2 -pedantic -std=c++17 -Wall -Werror search.o main.o -o benchmark
engineerxl@engineerxl:~/Study/DA/lab4/benchmark$ ./benchmark <test1e3.txt
Naive: 0.804 ms
KMP: 0.049 ms
Z-KMP: 0.042 ms
engineerxl@engineerxl:~/Study/DA/lab4/benchmark$ ./benchmark <test1e4.txt
Naive: 5.063 ms
KMP: 0.248 ms
Z-KMP: 0.220 ms
engineerxl@engineerxl:~/Study/DA/lab4/benchmark$ ./benchmark <test1e5.txt
Naive: 25.243 ms
KMP: 2.820 ms
Z-KMP: 2.752 ms
engineerxl@engineerxl:~/Study/DA/lab4/benchmark$ ./benchmark <test2e6.txt
Naive: 860.019 ms
KMP: 119.228 ms
Z-KMP: 101.489 ms
```

Видно, что наивный алгоритм поиска почти на порядок проигрывает алгоритмам Кнута-Морриса-Пратта. Классический алгоритм допускает лишние сравнения на этапе поиска образца в тексте, а алгоритм с применением Z-функции нет. Обработка лишних сравнений требует больше времени, чем препроцессинг для Z-функции, поэтому классический алгоритм Кнута-Морисса-Пратта отстаёт по времени.

5 Выводы

Во время выполнения лабораторной работы я вспомнил основные понятия, связанные со строками, изучил алгоритм Кнута-Морриса-Пратта и его варианты с препроцессингом через Z-функцию и префикс-функцию строки.

Задачи поиска часто встречаются в жизни, будь то поиск забытых вещей или нужной книги в библиотеке. Конкретно алгоритмы поиска подстроки используются в текстовых редакторах и браузере.

Я дополнительно ознакомился с другими алгоритмами поиска образцов в тексте. Для чего нужно так много алгоритмов? Каждый алгоритм хорош в отдельном случае. Например алгоритм Кнута-Морриса-Пратта хорошо ищет один образец в большом тексте, но не может эффективно найти несколько образцов в не самом большом тексте, а алгоритм Ахо-Корасик наоборот больше подходит для поиска множества образцов.

Список литературы

[1] Гасфилд Дэн. *Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология.* — Издательский дом «БХВ-Петербург». Перевод с английского: И. В. Романовский. — 654 с. (ISBN 5-7490-0103-8 ("БХВ-Петербург"))

[2] *MAXimal :: algo :: Префикс-функция. Алгоритм Кнута-Морриса-Пратта*
URL: `https://e-maxx.ru/algo/prefix_function` (дата обращения: 03.12.2020).

[3] *MAXimal :: algo :: Z-функция строки и её вычисление*
URL: `https://e-maxx.ru/algo/z_function` (дата обращения: 03.12.2020).