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. А. Почечура
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2022

Лабораторная работа №4

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

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

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

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

1 Описание

Требуется написать реализацию алгоритма KMP. Идея его заключается в том, что в начале нужно посчитать префикс функцию для каждого элемента паттерна. Префикс функция показывает, какого наибольшего размера суффикс текущей строки совпадает с её префиксом. При прикладывании паттерна к тексту, мы смотрим, какова длина совпавшей части, после чего сдвигаем паттерн на значение, равное значению префикс функции последнего совпавшего элемента.

2 Исходный код

Для начала введём паттерн в массив стрингов, параллельно приводя все буквы к нижнему регистру. В векторе $numbersOfFirstAlphaOnPos$ значение на конкретной позиции показывает, сколько слов, совпадающих с первым, встречалось до этого в паттерне. Затем посчитаем сильную префикс функцию (она отличается от обычной тем, что после префиксов одинакового размера всегда следует разная буква) с помощью z - функции для каждой позиции (z - функция показывает, какой длины префикс строки совпадает с последующими символами относительно текущей позиции, включая символ на текущей позиции). Из z - функции получается сильная префикс функция путём переносов значений по массиву на число, равное значению z - функции в конкретной позиции. Затем создадим очередь $posOfFirstAlpha$, в которой будут содердаться все позиции элементов в тексте, совпадающих с первым элементом паттерна и совпадающих с тем положением паттерна, на которое он передвинут в данный момент относительно текста. Далее построчно с помощью потоков (а затем по словам) вводим сам текст и сравниеваем его с паттерном. При совпадении слова с элементом паттерна сдвигаем позицию, на которой мы стоим в паттерне, и вводим следующее слово. Если слово не совпало, смотрим, какое значение префикс функции у предыдущего элемента, и на это значение относительно начала сдвигаем наш паттерн. Сдвигаем его до тех пор, пока не сдвинем на начало паттерна, либо элементы не совпадут. Если мы сверили все элементы паттерна, то выводим значение первого элемента очереди и сдвигаем наш паттерн на значение, равное значению префикс функции последнего элемента паттерна.

```
1  #include <iostream>
2  #include <string>
3  #include <vector>
4  #include <queue>
5  #include <sstream>
6
7  using namespace std;
8
9  int main(){
10     ios::sync_with_stdio(false);
11     cin.tie(nullptr); cout.tie(nullptr);
12     vector<string> pattern;
13     vector<long long> numbersOfFirstAlphaOnPos;
14     vector<string> text;
15     long long numbersOfFirstAlpha = 0;
16     string s;
17     getline(cin, s);
18     istringstream patternStream(s);
19     while(patternStream >> s){
20         transform(s.begin(), s.end(), s.begin(),
21             [](char c) { if(c < 'a'){
```

```
                        return char (c + ' ');
                    }
                    return c; });
        pattern.push_back(s);
        if(s.size() == pattern[0].size() && s == pattern[0]){
            numbersOfFirstAlpha++;
        }
        numbersOfFirstAlphaOnPos.push_back(numbersOfFirstAlpha);
    }
    int patterSizeOfFirstAlpha = pattern[0].size();
    long long p_size = pattern.size();
    vector<long long> spfunc(p_size);
    spfunc[0] = 0;
    long long k;
    long long l;
    long long r;
    for(long long i = p_size - 1; i > 0 ; i--){
        l = 0;
        r = i;
        k = 0;
        while(r < p_size && pattern[l].size() == pattern[r].size() && pattern[l] ==
            pattern[r]) {
            l++;
            r++;
            k++;
        }
        r--;
        spfunc[r] = k;
    }
    long long pos = 0;
    long long alpha = 0;
    long long line = 0;
    queue<pair<long long, long long>> posOfFirstAlpha;
    string st;
    istringstream textStream;
    while(getline(cin, st)){
        line++;
        textStream.clear();
        textStream.str(st);
        alpha = 0;
        while (textStream >> s) {
            transform(s.begin(), s.end(), s.begin(),
            [](char c) { if(c < 'a'){
                        return char (c + ' ');
                    }
                    return c; });
            alpha++;
            if(s.size() == patterSizeOfFirstAlpha && s == pattern[0]){
                posOfFirstAlpha.push({line, alpha});
```

```
70                  if(posOfFirstAlpha.size() > numbersOfFirstAlpha){
71                      posOfFirstAlpha.pop();
72                  }
73              }
74              if(s.size() == pattern[pos].size() && s == pattern[pos]){
75                  pos++;
76              }
77              else {
78                  if(pos != 0){
79                      pos--;
80                  }
81                  pos = spfunc[pos];
82                  if(s.size() == pattern[pos].size() && pattern[pos] == s){
83                      pos++;
84                  } else {
85                      while(pos != 0 && pattern[pos] != s){
86                          pos--;
87                          pos = spfunc[pos];
88                      }
89                      if(s.size() == pattern[pos].size() && pattern[pos] == s){
90                          pos++;
91                      }
92                  }
93                  while(posOfFirstAlpha.size() > numbersOfFirstAlphaOnPos[pos]){
94                      posOfFirstAlpha.pop();
95                  }
96              }
97              if(pos == p_size){
98                  cout << posOfFirstAlpha.front().first << ", " << posOfFirstAlpha.front()
                        .second << '\n';
99                  pos = spfunc[pos - 1];
100                 while(posOfFirstAlpha.size() > numbersOfFirstAlphaOnPos[pos]){
101                     posOfFirstAlpha.pop();
102                 }
103             }
104         }
105     }
106 }
```

3 Консоль

```
root@DESKTOP-5HM2HTK:~# cat <test
cat dog cat dog bird
CAT dog CaT Dog Cat DOG bird CAT
dog cat dog bird
root@DESKTOP-5HM2HTK:~# g++ lab4.cpp
```

```
root@DESKTOP-5HM2HTK:~# ./a.out <test
1,3
1,8
root@DESKTOP-5HM2HTK:~#
```

4 Тест производительности

```
root@DESKTOP-5HM2HTK:~# g++ -pedantic -Wall benchmark.cpp
root@DESKTOP-5HM2HTK:~# ./a.out <tests/1.t
Count of lines 100000
KMP time: 7371us
Simple search time: 21939us
root@DESKTOP-5HM2HTK:~#
```

Как видно, KMP алгоритм работает в три раза быстрее на данных тестах, чем наивный алгоритм. И это очевидно, потому что наивный алгоритм работает за O(n^2), а алгоритм Кнута-Морриса-Пратта за O(n), плюс требуется время на построение префикс функции.

5 Выводы

Выполнив четвёртую лабораторную работу по курсу «Дискретный анализ», я освоил алгоритм Кнута-Морриса-Прата, который может мне помочь в дальнейшим при решении практичесих задач. Также попрактиковался в работе с потоками и оптимизации программы.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))

[2] *Алгоритм Кнута — Морриса — Пратта — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Алгоритм_Кнута_-_Морриса_-_Пратта` (дата обращения: 14.05.2022).