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: Т. Б. Даутов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-22
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 $2^{32} - 1$.

1 Описание

В этой лабораторной работе требуется реализовать поиск всех вхождений образца (паттерна) в текст. Алгоритм посика - алгоритм Апостолико-Джанкарло.

Рассматриваемый алгоритм, по своей сути, является вариантом реализации алгоритма Бойера-Мура, включающим несколько дополнений.

Для каждой позиции текста мы можем запомнить наидлиннейшую подстроку, заканчивающуюся в этой позиции, к оторая является суффиксом образца. Тогда, используя эту информацию, мы сможем быстро “перепрыгивать” большие куски текста, в процессе его сравнения с образцом. Стоит обратить внимание, что эти значения будут посчитаны только для тех позиций, где образец прикладывался к тексту.

Важно, что алгоритм имеет простую линейную оценку времени в отличие от оригинального алгоритма Бойера-Мура. При этом он точно воспроизводит сдвиги алгоритма Бойера-Мура (согласно правилам плохого знака и хорошего суффикса), поэтому предлагаемый метод сохраняет все преимущества быстрых сдвигов метода Бойера-Мура и допускает простой анализ времени счета с линейной оценкой для наихудшего случая.

2 Исходный код

Сначала были реализованы функции для препроцессинга образца: функция вычисления массива сдвигов для расширенного правила плохого символа и массива сдвигов по правилу хорошего суффикса (в т.ч. сильного).

Были объявлены псевдонимы типов согласно специфике варианта задания (алфавит числовой). Наконец, был реализован алгоритм поиска Апостолико-Джанкарло. Вхождения накапливаются в вектор *occurances*.

В конце содержимое вектора вхождений печатается на стандартный вывод.

main.cpp	
std::map<uint32_t, std::vector<size_t» calcRPos(const pattern_t& p)	Функция подсчёта вхождений знаков в строку
std::vector<size_t> calcZFunction(const pattern_t& p)	Функция вычлисления Z-функции
std::vector<size_t> calcLL(const pattern_t& p, const std::vector<size_t>& nf)	Функция вычисления сдвигов по сильному правилу хорошего суффикса

```
  while (k <= text.size()) {
      i = n - 1;
      h = k - 1;
      while (true) {
          if(M[h] == 0) {
              if(pattern[i] == text[h].first) {
                  if(i == 0) {
                      occurances.push_back(text[k - n].second);
                      M[k - 1] = n;
                      if(n > 2) {
                          k += n - sgs[1];
                      }
                      else {
                          k++;
                      }
                      break;
                  }
                  else {
                      i--;
                      h--;
                  }
              }
              else {
                  M[k - 1] = k - h -1;
                  k += std::max(gs[i + 1], getBCShift(i, text[h].first, rpos));
                  break;
              }
          }
          else if(M[h] < nf[i]) {
              i -= M[h];
              h -= M[h];
          }
          else if(M[h] >= nf[i] && nf[i] >= i) {
              occurances.push_back(text[k - n].second);
              M[k - 1] = k - 1 - h;
              if(n > 2) {
                  k += n - sgs[1];
              }
```

```
39                  else {
40                      k++;
41                  }
42                  break;
43              }
44              else if(M[h] > nf[i] && nf[i] < i) {
45                  M[k - 1] = k - h - 1;
46                  h -= nf[i];
47                  i -= nf[i];
48                  k += std::max(gs[i + 1], getBCShift(i, text[h].first, rpos));
49                  break;
50              }
51              else if(M[h] == nf[i] && nf[i] < i) {
52                  i -= M[h];
53                  h -= M[h];
54              }
55          }
56      }
```

3 Консоль

```
timur@workstation$ g++ -pedantic -Wall -std=c++2a main.cpp -o lab4
timur@workstation$ cat input.txt
11 45 11 45 90
0011 45 011 0045 11 45 90    11
45 11 45 90
timur@workstation$ ./lab4 <input.txt
1,3
1,8
```

4 Тест производительности

Тест производительности был проведён следующим образом: сначала из открытого источника был взят большой текст из 8971 слов, затем был выбран случайный образец из текста. Сравнивались реализованный мной алгоритм Апостолико-Джанкарло и наивный алгоритм поиска.

```
timur@workstation$ g++ -std=c++2a benchmark.cpp -o bench
timur@workstation$ ./bench <bench.txt
[naive] = 4.901 ms
[ag] = 0.572 ms
```

Таким образом, алгоритм Апостолико-Джанкарло ожидаемо превосходит по производительности наивный алгоритм.

5 Выводы

В ходе выполнения лабораторной работы 4 мной был изучен и реализован на языке C++ один из вариантов алгоритма Бойера-Мура поиска образца в тексте - алгоритм Апостолико-Джанкарло.

Задача поиска образца в тексте, очевидно, является одной из самых важных и имеет множество практических прикладных применений, так что люди придумали множество алгоритмов, отличных от наивного, дающих существенно большую производительность.

Список литературы

[1] Гасфилд Д. *Строки, деревья и последовательности в алгоритмах* — Издательский дом «Невский Диалект», 2003. Перевод с английского: И. В. Романовский. — 654 с. (ISBN 5-7940-0103-8)

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