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: И. П. Попов
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 Бойера-Мура.

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

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

Формат ввода
Искомый образец задаётся на первой строке входного файла.

В случае, если в задании требуется найти несколько образцов, они задаются по одному на строку вплоть до пустой строки. Затем следует текст, состоящий из слов или чисел, в котором нужно найти заданные образцы.

Никаких ограничений на длину строк, равно как и на количество слов или чисел в них, не накладывается.

Формат вывода
В выходной файл нужно вывести информацию о всех вхождениях искомых образцов в обрабатываемый текст: по одному вхождению на строку.

Для заданий, в которых требуется найти только один образец, следует вывести два числа через запятую: номер строки и номер слова в строке, с которого начинается найденный образец. В заданиях с большим количеством образцов, на каждое вхождение нужно вывести три числа через запятую: номер строки; номер слова в строке, с которого начинается найденный образец; порядковый номер образца.

Нумерация начинается с единицы. Номер строки в тексте должен отсчитываться от его реального начала (то есть, без учёта строк, занятых образцами).

Порядок следования вхождений образцов несущественен.

1 Описание

Интернет-ресурс [2] дает следующее описание алгоритму Бойера-Мура:

Алгоритм основан на трёх идеях.

1. **Сканирование слева направо, сравнение справа налево**
Совмещается начало текста (строки) и шаблона, проверка начинается с последнего символа шаблона. Если символы совпадают, производится сравнение предпоследнего символа шаблона и т. д. Если все символы шаблона совпали с наложенными символами строки, значит, подстрока найдена, и выполняется поиск следующего вхождения подстроки.

Если же какой-то символ шаблона не совпадает с соответствующим символом строки, шаблон сдвигается на несколько символов вправо, и проверка снова начинается с последнего символа.

Эти «несколько», вычисляются по двум эвристикам.

2. **Эвристика стоп-символа**
(Замечание: эвристика стоп-символа присутствует в большинстве описаний алгоритма Бойера — Мура, включая оригинальную статью Бойера и Мура, но не является необходимой для достижения оценки $O(n+m)O(n+m);,.)$

3. **Эвристика совпавшего суффикса**
Неформально, если при чтении шаблона справа налево совпал суффикс S, а символ b, стоящий перед S в шаблоне (то есть шаблон имеет вид PbS), не совпал, то эвристика совпавшего суффикса сдвигает шаблон на наименьшее число позиций вправо так, чтобы строка S совпала с шаблоном, а символ, предшествующий в шаблоне данному совпадению S, отличался бы от b (если такой символ вообще есть). Формально, для данного шаблона s[0..m-1] считается целочисленный массив suffshift[0..m], в котором suffshift[i] равно минимальному числу j>0, такому что s[i-j] != s[i-1] (если i>0 и i-j >= 0) и s[i-j+k] = s[i-1+k] для любого k>0, для которого выполняется 0 <= i-j+k<m и 0 <= i-1+k<m (для пояснения смотрите примеры ниже). Затем, если при чтении шаблона s справа налево совпало k-1 символов s[m-1],s[m-2], ... ,s[m-k+1], а символ s[m-k] не совпал, то шаблон сдвигается на suffshift[m-k] символов вправо.

2 Исходный код

Основная логика работы прогарммы состоит в следующем: двигаясь по входному тексту слева направо(начиная с символа, по индексу равному размеру входного паттерна), идет его сравнение с паттерном справаа налево.

При выявлении несоответствия, происходит сдвиг по входному тексту на параметр $offset$. Данный параметр вычисляется с помощью двух эвристик, описанных выше. Применяется та эвристика, которая разрешит нам больший сдвиг.

Если паттерн пройден полностью, значит зафиксировано вхождение подстроки. И программа печатает номер строки и номер индекса в этой строке, где было зафиксировано вхождение.

Остановлюсь подробнее на подсчете сдвигов.
Код выисления **эвристики плохого символа**:

```
class TBadLetterRule {
public:
  TBadLetterRule(const vector<uint32_t>& pattern) :
    patt_size(pattern.size()) {
    for (int i = patt_size - 1; i >= 0; --i) {
      rule[pattern[i]].push_back(i);
    }
  }

  int Use(uint32_t letter, size_t idx_patt) const {
    auto it = rule.find(letter);
    if (it == rule.end()) {
      return patt_size - idx_patt;
    }
    const vector<uint32_t>& idxs = it->second;
    for (uint32_t i : idxs) {
      if (i > idx_patt){
        return 1;
      }
      else if (i < idx_patt) {
        return idx_patt - i;
      }
    }
    return patt_size;
  }

private:
  unordered_map<uint32_t, vector<uint32_t>> rule;
```

```
  size_t patt_size;
};
```

Код выисления **эвристики хорошего суффикса**

```
class TGoodSuffix {
public:
  TGoodSuffix(const vector<uint32_t>& pattern) : patt_size(pattern.size())
  {
    rule = LFunction(pattern, gp_size);
  }

  int Use(size_t idx_patt) const {
    if (idx_patt >= patt_size) {
      return 1;
    }
    if (rule[idx_patt] == UNDEFINED) {
      return patt_size - gp_size;
    }
    return patt_size - rule[idx_patt] - 1;
  }

  size_t Getgp_size() const {
    return gp_size;
  }
private:
  vector<size_t> rule;
  size_t patt_size;
  size_t gp_size = 0;
};
```

Особого внимания заслуживает способ вычисления L-функции:

```
vector<size_t> ZFunction(const vector<uint32_t>& pattern) {
  size_t n = pattern.size();
  vector<size_t> res(n, 0);
  size_t l = 0;
  size_t r = 0;
  for (size_t i = 1; i < n; ++i) {
    if (i <= r) {
      res[i] = min(r - i + 1, res[i - l]);
    }
    while (i + res[i] < n && pattern[res[i]] == pattern[i + res[i]]) {
      ++res[i];
    }
    if (i + res[i] - 1 > r) {
      l = i;
      r = i + res[i] - 1;
    }
  }
  return res;
```

```
}

vector<size_t> NFunction(vector<uint32_t> pattern) {
  reverse(pattern.begin(), pattern.end());
  size_t n = pattern.size();
  vector<size_t> z_func = ZFunction(move(pattern));
  vector<size_t> res(n);
  for (size_t i = 0; i < n; ++i) {
    res[i] = z_func[n - i - 1];
  }
  return res;
}

const size_t UNDEFINED = -1;

vector<size_t> LFunction(const vector<uint32_t>& pattern, size_t& gp_size) {
  gp_size = 0;
  size_t n = pattern.size();
  vector<size_t> res(n, UNDEFINED);
  vector<size_t> n_func = NFunction(move(pattern));
  for (size_t i = 0; i < n; ++i) {
    size_t j = n - n_func[i];
    if (j != n) {
      res[j] = i;
      if (i == n - j - 1) {
        gp_size = i + 1;
      }
    }
  }
  return res;
}
```

3 Консоль

tmp:

```
11 45 11 45 90
0011 45 011 0045 11 45 90    11
45 11 45 90
```

console output:

```
1,3
1,8
```

4 Тест производительности

Тест представлял из себя сравнение реализованного мной алгорима Бойера-Мура с бинарным поиском, реализованным в STL посредством функции upper$_b$*ound*.

В результате работы *benchmark.cpp* видны следующие результаты:

```
root@Lunidep:~/DA/da_lab4# ./wrapper.sh
[info][Tue May 17 11:35:24 MSK 2022] Stage #1. Compiling...
g++ -std=c++17 -pedantic -Wall -Wextra -Wno-unused-variable da_lab4.cpp -o
da_lab4
g++ -std=c++17 -pedantic -Wall -Wextra -Wno-unused-variable benchmark.cpp -o
benchmark
[info][Tue May 17 11:35:25 MSK 2022] Compiling OK
[info][Tue May 17 11:35:25 MSK 2022] Stage #2. Benchmark generating...
[info][Tue May 17 11:35:26 MSK 2022] Benchmark generating OK
[info][Tue May 17 11:35:26 MSK 2022] Stage #3. Benchmark results:
BM_search: 181 ms
find_search: 619 ms
[info][Tue May 17 11:35:28 MSK 2022] Benchmark OK
```

Из примера видно, релизованный мной алгорим превосходит своего STL-соперника, потому что он обладает линейной сложностью, а бинарный поиск - операция за O(log(n)).

5 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», мною были изучены различные алгоритмы поиска подстроки в строке и реализован один из них - алгоритм Бойера-Мура.

Реализация данного алгоритма очень удобно легла на ООП-парадигму языка C++, в частности методы посчета сдвиго в данной лабораторной были реализованы с её помощью.

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 Бойера—Мура. URL: https://ru.wikipedia.org/wiki/Алгоритм_