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: О. А. Артамонов
Преподаватель: Н. С. Капралов
Группа: М8О-208Б-19
Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №1
Задача: Необходимо реализовать один из стандартных алгоритмов поиска образцов для указанного алфавита.
Вариант алгоритма: Поиск одного образца при помощи алгоритма Бойера-Мура.
Вариант ключа: Числа в диапазоне от 0 до 2^{32} - 1.

1 Описание

Основная идея в поиске подстроки в строке с помощью алгоритма Бойера-Мура заключается в том, что паттерн прикладывают слева-направо, но символы сравнивают справо-налево. При несовпадении какого-то символа проверяются 2 правила: правило плохого символа (ППС) и правило хорошего суффикса (ПХС). Каждое из этих правил гарантирует, что сдвиг на какое-то x кол-во символов не пропустит вхождение паттерна в текст, т.е. мы должны выбрать максимальный сдвиг среди ППС, ПХС и 1. Это позволяет нам пропустить лишние сравнения символов без потерь вхождения. Для работы алгоритма необходимо предварительно посчитать функции Z, N, l, L для паттерна и подготовить массив/map, в котором хранится(хранятся) индекс(ы), по которым символ входит в паттерн в порядке убывания.
Правило плохого символа в случае несовпадения символа в паттерне и тексте пытается найти другое вхождение такого символа текста в паттерне и возвращает разницу между текущей позицией паттерна и позицией нужного символа.
Правило хорошего суффикса в случае совпадения каких-то символов до несовпадения пытается найти такое же вхождение суффикса паттерна в этом же паттерне и возвращает разницу между текущей позицией паттерна и позицией символа, после которого начинается этот суффикс.
Сложность алгоритма Бойера-Мура O(n+m), где m - длина паттерна, а n - длина текста.

2 Исходный код

main.cpp:

```
#include <algorithm>
#include <iostream>
#include <map>
#include <sstream>
#include <string>
#include <vector>

using namespace std;

vector<uint32_t> BuildZFunc(vector<uint32_t>& pattern) {
    uint32_t n = pattern.size();
    vector<uint32_t> Z(n);
    uint32_t l = 0;
    uint32_t r = 0;

    for (uint32_t i = 1; i < n; i++) {
        if (i <= r) {
            Z[i] = min(r - i + 1, Z[i - l]);
        }
        while (i + Z[i] < n && pattern[Z[i]] == pattern[i + Z[i]]) {
            Z[i]++;
        }
        if (i + Z[i] - 1 > r) {
            l = i;
            r = i + Z[i] - 1;
        }
    }

    return Z;
}

vector<uint32_t> BuildNFunc(vector<uint32_t>& pattern) {
    reverse(pattern.begin(), pattern.end());
    vector<uint32_t> Z = BuildZFunc(pattern);
    vector<uint32_t> N(Z.rbegin(), Z.rend());
    reverse(pattern.begin(), pattern.end());

    return N;
}

void BoyerMoore(vector<uint32_t>& pattern, vector<uint32_t>& text,
                vector<pair<uint32_t, uint32_t>>& counters, map<uint32_t,
                vector<uint32_t>>& occurrences) {
    vector<uint32_t> N = BuildNFunc(pattern);
    vector<uint32_t> l(pattern.size() + 1);
    vector<uint32_t> L(pattern.size() + 1);
    uint32_t j;

    for (int i = 0; i < pattern.size() - 1; i++) {

        if (N[i] != 0) {
            j = pattern.size() - N[i];
            l[j] = i;
        }
```

```
    if (N[i] == i + 1) {
        L[(pattern.size() - 1) - i] = i + 1;
    } else {
        L[(pattern.size() - 1) - i] = L[(pattern.size() - 1) - i + 1];
    }
}

int offset = pattern.size() - 1;

while (offset < text.size()) {
    int i = pattern.size() - 1;
    int j = offset;

    while (i >= 0 && pattern[i] == text[j]) {
        i--;
        j--;
    }

    if (i == -1) {
        int index = offset - pattern.size() + 1;

        cout << counters[index].first << ", " << counters[index].second << "\n";

        if (pattern.size() > 2) {
            offset += pattern.size() - L[1];
        } else {
            offset++;
        }

    } else {
        int goodSuffix = 1;
        int badChar = 1;

        if (i < pattern.size() - 1) {
            if (l[i + 1] > 0) {
                goodSuffix = pattern.size() - l[i + 1] - 1;
            } else {
                goodSuffix = pattern.size() - L[i + 1];
            }
        }

        for (uint32_t& rightIndex : occurrences[text[j]]) {
            // cout << "Index: " << rightIndex << endl;
            if (rightIndex < i && rightIndex != 0) {
                badChar = i - rightIndex;
                break;
            }
        }

        // ДЛЯ ППППС
        // if (occurrences[text[j]] < i && occurrences[text[j]] != 0) {
        //  badChar = i - occurrences[text[j]];
        //  // cout << "i: " << i << " j: " << j << " occurrences[text[j]]: " << occurrences[text[j]];
        // }
        // // cout << endl;
        // КОНЕЦ
```

```
            // cout << "suff = " << goodSuffix << " badchar = " << badChar << endl;

            offset += max(goodSuffix, badChar);
        }
    }
}

int main() {
    cin.tie(0);
    cout.tie(0);
    ios::sync_with_stdio(false);

    vector<uint32_t> pattern;
    vector<uint32_t> text;
    vector<pair<uint32_t, uint32_t>> counters;
    map<uint32_t, vector<uint32_t>> occurrences;
    // ДЛЯ ППППС
    // map<uint32_t, uint32_t> occurrences;

    // РЕАЛИЗАЦИЯ ЧЕРЕЗ СТРИМЫ
    // uint32_t number;
    // string line;

    // getline(cin, line);
    // istringstream ss(line);
    // while (ss >> number) {
    //  pattern.push_back(number);
    // }

    // for (int i = pattern.size() - 1; i >= 0; i--) {
    //  occurrences[pattern[i]].push_back(i);
    //  // cout << "to " << pattern[i] << " added index " << i << endl;
    // }

    // if (pattern.size() == 0) {
    //  return 0;
    // }

    // int row = 1;
    // int word = 1;
    // while (getline(cin, line)) {
    //  istringstream ss(line);
    //  while (ss >> number) {
    //      text.push_back(number);
    //      counters.push_back({row, word});
    //      word++;
    //  }
    //  word = 1;
    //  row++;
    // }
    // КОНЕЦ

    string temp;
    char s;
    // uint32_t number;
    // uint32_t counter = 0;
```

```
while ((s = getchar()) != EOF) {
    if (s == ' ') {
        if (!temp.empty()) {
            pattern.push_back(static_cast<uint32_t>(stoul(temp)));
            temp.clear();
            // occurrences[number] = counter;
            // counter++;
        }
    } else if (s == '\n') {
        if (!temp.empty()) {
            pattern.push_back(static_cast<uint32_t>(stoul(temp)));
            // occurrences[number] = counter;
            // counter++;
            temp.clear();
        }
        break;
    } else {
        temp.push_back(s);
    }
}

if (pattern.size() == 0) {
    return 0;
}

uint32_t row = 1;
uint32_t word = 1;
uint32_t space = 0;

while ((s = getchar()) != EOF) {
    if (s == ' ') {
        if (!temp.empty()) {
            text.push_back(static_cast<uint32_t>(stoul(temp)));
            temp.clear();
            counters.push_back({row, word});
        }
        if (!space) {
            space = 1;
            word++;
        }
    } else if (s == '\n') {
        if (!temp.empty()) {
            text.push_back(static_cast<uint32_t>(stoul(temp)));
            temp.clear();
            counters.push_back({row, word});
        }
        space = 0;
        word = 1;
        row++;
    // } else if (s < 0) {
    //  if (!temp.empty()) {
    //      text.push_back(stoul(temp));
    //      temp.clear();
    //      counters.push_back({row, word});
    //  }
    //  break;
```

```
        } else {
            space = 0;
            temp.push_back(s);
        }
    }

    if (text.size() < pattern.size()) {
        return 0;
    }

    for (long i = pattern.size() - 1; i >= 0; i--) {
        occurrences[pattern[i]].push_back(i);
        // cout << "to " << pattern[i] << " added index " << i << endl;
    }

    BoyerMoore(pattern, text, counters, occurrences);

    return 0;
}
```

3 Консоль

```
[eartqk@eartqk-pc BM]$ ./solution
11 45 11 45 90
0011 45 011 0045 11 45 90    11
45 11 45 90
1, 3
1, 8
[eartqk@eartqk-pc BM]$ ./solution
1 1 1
2 1 1 5 1 1 1 1 6
0
1 1
1 2
1, 5
1, 6
3, 1
[eartqk@eartqk-pc BM]$
```

4 Тест производительности

Сравнивать производительность будем с наивной реализацией поиска подстроки в строке. benchmark_naive - программа с наивной реализацией поиска, solution - алгоритм Бойера-Мура.
Сравнение производится с помощью утилиты time. Всего в тексте содержится 15000 строк, в каждой по 10000 случайных чисел.

```
[eartqk@eartqk-pc BM]$ time ./benchmark_naive < input.txt > /dev/null

real    1m56,700s
user    1m38,983s
sys     0m14,043s
[eartqk@eartqk-pc BM]$ time ./solution < input.txt > /dev/null

real    0m48,870s
user    0m33,772s
sys     0m12,601s
[eartqk@eartqk-pc BM]$
```

Можно смотреть на реальное время выполнения программ, у наивного поиска заняло 116 секунд, против 48 секунд у Бойера-Мура.
Алгоритм Бойера-Мура оказался быстрее более, чем в 2 раза.
Результат не удивляет, ведь сложность у БМ O($m + n$), а у наивного O(mn), где m - длина паттерна, а n - текста.

5 Выводы

Благодаря лабораторной работе я познакомился с алгоритмами поиска подстроки в строке и реализовал алгоритм Бойера-Мура.
Данный алгоритм довольно универсальный и часто используется, например, в браузерах или текстовых редакторах для поиска. Из преимуществ можно отметить его эффективность: при использовании расширенных правил плохого символа и хорошего суффикса можно пропустить большое количество ненужных сравнений.
Из недостатков как и КМП можно отметить то, что поиск осуществляется по только одному паттерну, поэтому в случае необходимости найти несколько разных паттерном лучше воспользоваться, например, алгоритмом Ахо-Корасика, ну или каждый раз запускать БМ с новым паттерном.

Список литературы

[1] Дэн Гасфилд. *Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология* — Невский Диалект, БХВ-Петербург, 2003. 656 с. (ISBN 5-7940-0103-8, 5-94157-321-9, 0-521-58519-8)

[2] *Алгоритм Бойера-Мура — Викиконспекты ИТМО*
URL:
`https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Бойера-Мура`
(дата обращения 28.03.2021).

[3] *MAXimal :: algo :: Z-функция строки и её вычисление*
URL: `https://e-maxx.ru/algo/z_function` (дата обращения 27.03.2021).