Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа № 4 по курсу дискретного анализа: Поиск образцов

Студент: А. О. Тояков
Преподаватель: Н. А. Зацепин
Группа: М8О-307Б-18
Дата:
Оценка:
Подпись:

Москва
2020

Условие

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

2. Вариант алгоритма: Поиск одного образца при помощи алгоритма Бойера-Мура. Вариант алфавита: Числа в диапазоне от 0 до $2^{32} - 1$

Метод решения

1. Считываю данные в формате: P на первой строке и T на всех остальных, где P - шаблон, T -текст. С помощью функций ParsePatt и ParseText преобразую образец и текст в вектор чисел и структуру данных TWords соответственно.

2. Затем выполняю алгоритм Бойера-Мура (функция Boyer-Moore), который использует в себе правила плохого символа (функция BadChar) и правило хорошего суффикса (функция GoodSuff).

3. Функция алгоритма выводит номера строк и номер слова, начиная с которого нашёлся образец в тексте.

4. Если совпадений нет функция ничего не выведет и просто пройдёт по всему тексту.

Описание программы

В данной работе создана структура TWords, в которой хранится три параметра: номер строки (str), номер слова (wrd) и само значение (val).

1. Функция AddWord используется при парсинге текста в функции ParseText. Она проходит по всему тексту посимвольно и вводит поочереди числа в структуру TWords, если видит пробел, то увеличивает количество слов и пушит слово, в вектор, если видит переход на новую строку то увеличивает количество строк и обнуляет количество слов.

2. Функция ParsePatt работает аналогичым образом, только значения шаблона заносятся в вектор чисел, а не вектор структур TWords.

3. Функция BoyerMoore работает следующим образом: с помощью правила плохого суффикса вычисляется мап-таблица R, в которой каждому элементу из алфавита x шаблона ставится в соответствие самый правый номер индекса, под которым он находится в образце. Затем с помощью обратной Z-функции строится вектор L(i) для правила хорошего суффикса, где для каждого i указано максимальное правое положение такого же суффикса в образце. После этого мы прикладываем шаблон к тексту и начинаем с конца шаблона сравнивать элементы. Если есть полное совпадение, то в вектор пар result заносится номер строки и номер слова, где появилось совпадение и делается сдвиг на длину шаблона. Если же на каком

то этапе выявилось несовпадние, то для этого коэффициента вычисляется сдвиг по правилу хорошего суффикса и плохого символа и берётся максимальный из них. В конце просто осуществляется вывод результата из вектора пар result.

Исходный код

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

using namespace std;

struct TWords {
    long long wrd;
    long long str;
    long long val;
};

TWords AddWord(long long wrd, long long str, long long val) {
        TWords tmp;
        tmp.wrd = wrd;
        tmp.str = str;
        tmp.val = val;
        return tmp;
}

void ParseText(vector<TWords> &text) {
    string s;
    long long str_num = 1;
    long long wrd = 1;
    long long symbol;
    bool num_read = false;
    int temp = 0;
    symbol = getchar();
    while (true) {
        if ((symbol == ' ') || (symbol == '\n')) {
            if (num_read) {
                text.push_back(AddWord(wrd, str_num, temp));
                num_read = false;
                temp = 0;
                wrd++;
            }
```

```
            if (symbol == '\n') {
                str_num++;
                wrd = 1;
            }
        } else {
            if (!num_read) {
                num_read = true;
            }
            temp = temp * 10 + (symbol - '0');
        }
        symbol = getchar();
        if (symbol == EOF) {
            break;
        }
    }
    if (num_read) {
        text.push_back(AddWord(wrd, str_num, temp));
    }
}

void ParsePatt(vector <long long> &pattern) {
    long long symb;
    long long temp = 0;
    bool num_read = false;
    symb = getchar();
    while (true) {
        if (symb == ' ') {
            if (num_read) {
                pattern.push_back(temp);
                num_read = false;
                temp = 0;
            }
        } else {
            if (!num_read) {
                num_read = true;
            }
            temp = temp * 10 + (symb - '0');
        }
        symb = getchar();
        if (symb == '\n') {
            break;
        }
```

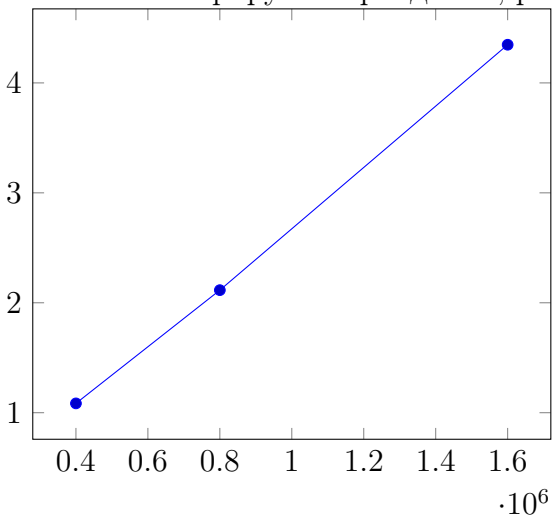
```
    }
    if (num_read) {
        pattern.push_back(temp);
    }
}

void ZFunction(vector <long long> &patt, vector <long long> &z) {
    long long right = 0, left = 0;
    long long size = patt.size();
    for (long long j = 1; j < size; ++j) {
        if (j <= right)
            z[j] = min(right - j + 1, z[j - left]);
        while ((j + z[j] < size) && (patt[size - 1 - z[j]] == patt[size - 1 - (j + z[j])
            z[j]++;
        }
        if ((j + z[j] - 1) > right) {
            left = j;
            right = j + z[j] - 1;
        }
    }
}

void GoodSuff(vector <long long> &patt, vector <long long> &suff) {
    vector <long long> z(patt.size(), 0);
    ZFunction(patt, z);
    for (long long j = patt.size() - 1; j > 0; j--) {
        suff[patt.size() - z[j]] = j;
    }
    for (long long j = 1, r = 0; j < patt.size(); j++) {
        if (j + z[j] == patt.size()) {
            for (; r <= j; r++) {
                if (suff[r] == patt.size()) {
                    suff[r] = j;
                }
            }
        }
    }
}

void BadChar(vector <long long> &patt, map <long long, long long> &R) {
    for (long long i = patt.size() - 1; i >= 0; i--) {
        if (R.count(patt[i]) == 0) {
```

```
            R.insert(make_pair(patt[i], i));
        }
    }
}

void BoyerMoore(vector <long long> &patt, vector <TWords> &text) {
    vector <pair<long, long>> result;
    map <long long, long long> R;
    BadChar(patt, R);
    vector <long long> suff(patt.size() + 1, patt.size());
    GoodSuff(patt, suff);
    long long bound = 0;
    long long j;
    for (long long i = 0; i <= (long long)(text.size() - patt.size()); ) {
        for (j = patt.size() - 1; (j >= bound) && (patt[j] == text[i + j].val); j--) {}
        //compare pattern and text until find unmatching
        if (j < bound) {
            result.push_back(make_pair(text[i].str, text[i].wrd));
            bound = patt.size() - suff[0];
            i += suff[0];
        } else {
            long long bad_suffix;
            if (R.count(text[i + j].val) == 1) {
                bad_suffix = R[text[i + j].val];
            } else {
                bad_suffix = -1;
            }
            i += max(suff[j + 1], j - bad_suffix); //use the rule of a good suffix or th
            bound = 0;
        }
    }
    for (long long i = 0; i < result.size(); i++) {
        cout << result[i].first << ", " << result[i].second << '\n';
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    vector <long long> patt;
    vector <TWords> text;
    ParsePatt(patt);
```

```
    ParseText(text);
    BoyerMoore(patt, text);
    return 0;
}
```

Тест производительности

В данных тестах используются строки только из 50 элементов, сгенерированных рандомно. Числа генерируются рандомно, размер числа: от 0 до 16 символов.



Пояснения к графику: Ось y - время в секундах. Ось x - количество строк.

1. За 1.0848 обрабатывается 400000 слов.
2. За 2.11492 обрабатывается 800000 слов.
3. За 4.34717 обрабатывается 1600000 слов.

Выводы

В ходе данной работы я познакомился с таким алгоритмом: поиск подстроки в строке с помощью алгоритма Бойера-Мура. Также алгоритм построения z функции делится на тривиальную и нетривиальную реализацию. В нетривиальной реализации используются числа left и right, которые характеризуют позиции начала и конца максимального Z-блока. Эта реализация позволяет строить Z-функцию за O(n). Сам алгоритм Бойера-Мура работает за линейное время в худшем случае и позволяет обычно сравнивать меньше чем (n + m) символов.