Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №5 по курсу

«Дискретный анализ»

Студент: Ибрагимов Р. Р.
Группа: М8О-303Б-22
Вариант: 2
Преподаватель: Макаров Н.К
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Постановка задачи
Найти в заранее известном тексте поступающие на вход образцы с использованием суффиксного массива.
Примечание: можно делать как через суффиксное дерево, так и через сортировку.

Формат ввода
Текст располагается на первой строке, затем, до конца файла, следуют строки с образцами.

Формат вывода
Для каждого образца, найденного в тексте, нужно распечатать строчку, начинающуюся с последовательного номера этого образца и двоеточия, за которым, через запятую, нужно перечислить номера позиций, где встречается образец в порядке возрастания.

Метод решения
Для построения суффиксного массива буду использовать алгоритм его построения без использования суффиксного дерева — алгоритм цифровой сортировки циклических строк.
Далее буду осуществлять поиск всех вхождений паттерна в текст следующим образом: с помощью бинарного поиска для текущего паттерна найду первое и последнее вхождения в суффиксный массив, затем в ответ надо будет взять все суффиксы, которые располагаются между первым и последним вхождениями. Таким образом, найдутся все вхождения паттерна. В конце, отсортируем все вхождения по возрастанию.
Сложность построения суффиксного массива — O(nlogn), поиск в нем — O(mlogn), где n — длина текста, m — длина паттерна.

Исходный код
```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

using namespace std;

vector<int> buildSuffixArray(const string &s) {
    int n = s.size();
    vector<int> suffix_array(n);

    int max_symbol = -1;
    for (int i = 0; i < n; ++i) {
        if (s[i] > max_symbol) {
            max_symbol = s[i];
        }
    }
```

```
    vector<int> cnt(max(max_symbol + 1, n), 0);
    for (int i = 0; i < n; ++i) {
        ++cnt[s[i]];
    }
    for (int i = 1; i < cnt.size(); ++i) {
        cnt[i] += cnt[i - 1];
    }

    for (int i = n - 1; i >= 0; --i) {
        suffix_array[--cnt[s[i]]] = i;
    }

    vector<int> eq_class(n);
    eq_class[suffix_array[0]] = 0;
    int class_value = 1;
    for (int i = 1; i < n; ++i) {
        if (s[suffix_array[i]] != s[suffix_array[i - 1]]) {
            ++class_value;
        }
        eq_class[suffix_array[i]] = class_value - 1;
    }

    vector<int> tmp_suffix_array(n), tmp_eq_class(n);
    for (int len = 1; len < n; len *= 2) {
        for (int i = 0; i < n; ++i) {
            tmp_suffix_array[i] = suffix_array[i] - len;
            if (tmp_suffix_array[i] < 0) {
                tmp_suffix_array[i] += n;
            }
        }

        cnt.assign(class_value, 0);

        for (int i = 0; i < n; ++i) {
            ++cnt[eq_class[tmp_suffix_array[i]]];
        }
        for (int i = 1; i < class_value; ++i) {
            cnt[i] += cnt[i - 1];
        }
        for (int i = n - 1; i >= 0; --i) {
            suffix_array[--cnt[eq_class[tmp_suffix_array[i]]]] =
tmp_suffix_array[i];
        }

        tmp_eq_class[suffix_array[0]] = 0;
        class_value = 1;
        for (int i = 1; i < n; ++i) {
            pair<int, int> curr = {eq_class[suffix_array[i]],
eq_class[(suffix_array[i] + len) % n]};
            pair<int, int> prev = {eq_class[suffix_array[i - 1]],
eq_class[(suffix_array[i - 1] + len) % n]};
```

```
            if (curr != prev) {
                ++class_value;
            }
            tmp_eq_class[suffix_array[i]] = class_value - 1;
        }
        eq_class = tmp_eq_class;
    }

    vector<int> result(n - 1);
    for (int i = 1; i < n; ++i) {
        result[i - 1] = suffix_array[i];
    }

    return result;
}

vector<int> searchPattern(const string &text, const string &pattern,
const vector<int> &suffixArray) {
    vector<int> result;

    if (pattern == "") {
        return result;
    }

    int n = text.size(), m = pattern.size();
    int L = 0, R = n;

    while (L < R) {
        int M = (L + R) / 2;
        if (text.compare(suffixArray[M], m, pattern) >= 0) {
            R = M;
        }
        else {
            L = M + 1;
        }
    }
    int first_entry = L;

    R = n;
    while (L < R) {
        int M = (L + R) / 2;
        if (text.compare(suffixArray[M], m, pattern) > 0) {
            R = M;
        }
        else {
            L = M + 1;
        }
    }
    int last_entry = L;

    if (first_entry == last_entry) {
```

```
        return result;
    }

    result.resize(last_entry - first_entry);
    for (int i = first_entry, j = 0; i < last_entry; ++i) {
        result[j++] = suffixArray[i];
    }
    sort(result.begin(), result.end());

    return result;
}

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

    string text;
    getline(cin, text);

    vector<int> suffix_array = buildSuffixArray(text + "$");

    string pattern;
    int pattern_number = 0;

    while (getline(cin, pattern)) {
        ++pattern_number;
        vector<int> entries = searchPattern(text, pattern, suffix_array);

        if (!entries.empty()) {
            cout << pattern_number << ": ";

            for (size_t i = 0; i < entries.size(); ++i) {
                cout << entries[i] + 1;
                if (i < entries.size() - 1) {
                    cout << ", ";
                }
            }

            cout << '\n';
        }
    }

    return 0;
}
```

Тесты

Номер теста	Ввод	Вывод
1	abcdabc abcd bcd	1: 1 2: 2 3: 2, 6

	bc	
2	abacaba aba tttt a aw caba qu	1: 1, 5 3: 1, 3, 5, 7 6: 4
3	capdup tiop qwry bn zcja	

Выводы

В ходе выполнения данной лабораторной работы я познакомился с суффиксными массивами, узнал как их строить и искать в них. Алгоритм построения мне показался очень занятным.