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: М. Ю. Курносов
Преподаватель: А. А. Кухтичев
Группа: М8О-306Б
Дата:
Оценка:
Подпись:

Москва, 2025

Лабораторная работа №5

Задача: Найти в заранее известном тексте поступающие на вход образцы с использованием суффиксного массива.

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.

Примеры:

Входные данные:
abcdabc
abcd
bcd
bc

Результат работы:
1: 1
2: 2
3: 2, 6

1 Описание

Разработанная программа предназначена для решения задачи поиска всех вхождений заданного паттерна в тексте с использованием суффиксного массива. В основе решения лежит алгоритм построения суффиксного массива с помощью цифровой сортировки со сложностью O(n log n) и алгоритм бинарного поиска по суффиксному массиву со сложностью O(m log n) для каждого паттерна.

Основные этапы работы программы: 1. Чтение входных данных - текст считывается из стандартного ввода, паттерны передаются как аргументы командной строки 2. Построение суффиксного массива - с использованием алгоритма цифровой сортировки с классами эквивалентности 3. Поиск паттернов - бинарный поиск по суффиксному массиву для нахождения границ вхождений 4. Вывод результатов - позиции всех найденных вхождений паттерна в тексте

Программа демонстрирует практическое применение суффиксных массивов для решения задачи поиска подстрок, обеспечивая высокую производительность при работе с большими объемами текстовых данных.

2 Исходный код

fin.hpp:

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


using namespace std;

vector<int> BuildArr(const string &str) {
    int n = str.size();
    vector<int> suff_arr(n);

    int max_sim = -1;
    for (int i = 0; i < n; ++i) {
        if (str[i] > max_sim) {
            max_sim = str[i];
        }
    }

    vector<int> count(max(max_sim + 1, n), 0);
    for (int i = 0; i < n; ++i) {
        ++count[str[i]];
    }
    for (int i = 1; i < count.size(); ++i) {
        count[i] += count[i - 1];
    }

    for (int i = n - 1; i >= 0; --i) {
        suff_arr[--count[str[i]]] = i;
    }

    vector<int> class_eq(n);
    class_eq[suff_arr[0]] = 0;
    int class_eq_count = 1;
    for (int i = 1; i < n; ++i) {
        if (str[suff_arr[i]] != str[suff_arr[i - 1]]) {
            ++class_eq_count;
        }
        class_eq[suff_arr[i]] = class_eq_count - 1;
    }

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

```
47                tmp_suff_arr[i] += n;
48            }
49        }
50
51        count.assign(class_eq_count, 0); //0000000000000
52
53        for (int i = 0; i < n; ++i) {
54            ++count[class_eq[tmp_suff_arr[i]]];
55        }
56        for (int i = 1; i < class_eq_count; ++i) {
57            count[i] += count[i - 1];
58        }
59        for (int i = n - 1; i >= 0; --i) {
60            suff_arr[--count[class_eq[tmp_suff_arr[i]]]] = tmp_suff_arr[i];
61        }
62
63        tmp_eq_class[suff_arr[0]] = 0;
64        class_eq_count = 1;
65        for (int i = 1; i < n; ++i) {
66            pair<int, int> now = {class_eq[suff_arr[i]], class_eq[(suff_arr[i] + len) %
                  n]};
67            pair<int, int> prev = {class_eq[suff_arr[i - 1]], class_eq[(suff_arr[i - 1]
                  + len) % n]};
68            if (now != prev) {
69                ++class_eq_count;
70            }
71            tmp_eq_class[suff_arr[i]] = class_eq_count - 1;
72        }
73        class_eq = tmp_eq_class;
74    }
75
76    vector<int> res(n - 1);
77    for (int i = 1; i < n; ++i) {
78        res[i - 1] = suff_arr[i];
79    }
80
81    return res;
82 }
83
84 vector<int> Search(const string &text, const string &patt, const vector<int> &
       suffixArray) {
85    vector<int> res;
86
87    if (patt == "") {
88        return res;
89    }
90
91    int n = text.size(), m = patt.size();
92    int L = 0, R = n;
```

```
93
94      while (L < R) {
95          int M = (L + R) / 2;
96          if (text.compare(suffixArray[M], m, patt) >= 0) {
97              R = M;
98          }
99          else {
100             L = M + 1;
101         }
102     }
103     int first_inner = L;
104
105     R = n;
106     while (L < R) {
107         int M = (L + R) / 2;
108         if (text.compare(suffixArray[M], m, patt) > 0) {
109             R = M;
110         }
111         else {
112             L = M + 1;
113         }
114     }
115     int last_inner = L;
116
117     if (first_inner == last_inner) {
118         return res;
119     }
120
121     res.resize(last_inner - first_inner);
122     for (int i = first_inner, j = 0; i < last_inner; ++i) {
123         res[j++] = suffixArray[i];
124     }
125     sort(res.begin(), res.end());
126
127     return res;
128 }
129
130 int main() {
131     ios::sync_with_stdio(false);
132     cin.tie(nullptr);
133
134     string text;
135     getline(cin, text);
136
137     vector<int> suff_arr = BuildArr(text + "$");
138
139     string patt;
140     int numb = 0;
141
```

```
142     while (getline(cin, patt)) {
143         ++numb;
144         vector<int> entries = Search(text, patt, suff_arr);
145 
146         if (!entries.empty()) {
147             cout << numb << ": ";
148 
149             for (size_t i = 0; i < entries.size(); ++i) {
150                 cout << entries[i] + 1;
151                 if (i < entries.size() - 1) {
152                     cout << ", ";
153                 }
154             }
155 
156             cout << '\n';
157         }
158     }
159 
160     return 0;
161 }
```

fin.cpp	
vector<int> BuildArr(const string &str)	Построение суффиксного массива
vector<int> Search(const string &text, const string &patt, const vector<int> &suffixArray)	Поиск паттернов
int main()	Основная программа где осуществляется ввод/вывод результатов

3 Консоль

```
me@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025.2/5lab$ make
g++ -std=c++11 -Wextra -Wno-sign-compare  -o program fin.cpp

me@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025.2/5lab$ ./program
abacaba
ba
caba
1: 2,6
2: 4
me@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025.2/5lab$
```

4 Тест производительности

Тестирование заключается в проведении комплексной оценки эффективности реализованного алгоритма построения суффиксного массива и поиска подстрок, проверке соответствия теоретической сложности практическим результатам.

В тестах исследуются: время построения суффиксного массива для текстов различной длины, Время поиска паттернов разной длины в текстах различного объема, Зависимость времени выполнения от размера входных данных

Сценарии тестирования:

Малые объемы данных: тексты длиной 100-1,000 символов Средние объемы данных: тексты длиной 5,000-10,000 символов Большие объемы данных: тексты длиной 50,000-100,000 символов Влияние длины паттерна: поиск паттернов длиной от 5 до 100 символов

Ожидаемые результаты:

Подтверждение теоретической сложности O(n log n) для построения суффиксного массива Подтверждение сложности O(m log n) для операции поиска Демонстрация линейной зависимости использования памяти от размера текста Выявление практических ограничений алгоритма

```
nixx@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025.2/5lab$ ./benchmark
=== SMALL BENCHMARKS ===
Construction for text size 100: 128 μs
Construction for text size 500: 1016 μs
Construction for text size 1000: 2444 μs
Search 100 patterns (size 10) in text size 1000: 48 μs
Estimated memory for text size 1000: 4 KB

=== MEDIUM BENCHMARKS ===
Construction for text size 5000: 13544 μs
Construction for text size 10000: 27823 μs
Search 50 patterns (size 20) in text size 10000: 33 μs
Estimated memory for text size 10000: 48 KB

=== LARGE BENCHMARKS ===
Construction for text size 50000: 124320 μs
Construction for text size 100000: 311545 μs
Search 20 patterns (size 30) in text size 100000: 39 μs
Estimated memory for text size 100000: 488 KB
```

```
=== PATTERN LENGTH IMPACT ===
Pattern length 5: 95 $\mu sfor100searches$
Pattern length 10: 125 $\mu sfor100searches$
Pattern length 20: 191 $\mu sfor100searches$
Pattern length 50: 382 $\mu sfor100searches$
Pattern length 100: 829 $\mu sfor100searches$
nixx@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025.2/5lab$
```

5 Выводы

В ходе выполнения работы была успешно разработана и реализована программа для эффективного поиска всех вхождений заданных паттернов в тексте с использованием суффиксного массива.

Основные достижения:

Эффективность алгоритма: Реализованный алгоритм построения суффиксного массива с помощью цифровой сортировки демонстрирует сложность O(n log n), что является оптимальным для данной задачи. Последующий поиск паттернов выполняется за O(m log n) для каждого паттерна.

Практическая применимость: Программа успешно решает поставленную задачу поиска подстрок, обрабатывая как одиночные паттерны, так и множественные запросы.

Алгоритм корректно обрабатывает граничные случаи, включая пустые паттерны, отсутствие вхождений и специальные символы. Добавление терминального символа '$' гарантирует корректное построение суффиксного массива. Программа использует O(n) дополнительной памяти, что делает её применимой для работы с текстами значительного объема. Реализованная система тестирования позволяет автоматически проверять корректность работы программы на различных наборах данных.

Список литературы

[1] *Гасфилд Дэн. Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология / Пер. с англ. И.В.Романовского. — СПб.: Невский Диалект; БХВ Петербург, 2003. — 654 с.: ил.*

[2] Фундаментальные алгоритмы на C++ Роберт Седжвик Г7Г/ПИ/ торгово-иадатмьскнй дом 1Ж DiaSoft Москва • Санкт-Петербург •2002 (дата обращения: 10.10.2025).

[3] 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.))