Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №7 по курсу «Дискретный анализ»

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

Москва, 2025

Лабораторная работа №7

Задача: Задана строка S состоящая из n прописных букв латинского алфавита. Вычеркиванием из этой строки некоторых символов можно получить другую строку, которая будет являться палиндромом. Требуется найти количество способов вычеркивания из данного слова некоторого (возможно, пустого) набора таких символов, что полученная в результате строка будет являться палиндромом. Способы, отличающиеся только порядком вычеркивания символов, считаются одинаковыми.

Формат ввода: Задана одна строка S $|S| \leq 100$
Формат вывода: Необходимо вывести одно число – ответ на задачу. Гарантируется, что он $\leq 2^{63} - 1$

Примеры:

Входные данные:
BAOBAB

Результат работы:
22

1 Описание

Для эффективного решения задачи используется метод динамического программирования с заполнением двумерной таблицы. Основная идея: Определение состояния: table[i][j] — количество палиндромных подстрок в подстроке s[i..j].

Базовые случаи:

Любой отдельный символ является палиндромом: table[i][i] = 1 для всех i.

Рекуррентные соотношения:

Если крайние символы s[i] и s[j] совпадают, то:

table[i][j] = table[i+1][j] + table[i][j-1] + 1 Здесь мы учитываем все палиндромы из левой и правой частей, а также добавляем 1 для новой подстроки, образованной крайними символами.

Если крайние символы не совпадают, то:

table[i][j] = table[i+1][j] + table[i][j-1] - table[i+1][j-1] В этом случае мы вычитаем пересечение (подстроку s[i+1..j-1]), которая была учтена дважды.

2 Исходный код

main.cpp:

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

using namespace std;

int main() {

    string s;
    cin >> s;
    size_t n = s.size();

    vector<vector<int64_t> > poli_count(n);
    for (size_t i = 0; i < n; i++) {
        vector<int64_t> row(n, 0);
        poli_count[i] = row;
        poli_count[i][i] = 1;
    }

    for (size_t l = 1; l <= n; l++) {
        int i = 0, j;
        while(i + l < n) {
            j = i + l;
            if(s[i] == s[j]) {
                poli_count[i][j] = poli_count[i + 1][j] + poli_count[i][j - 1] + 1;
            }
            else {
                poli_count[i][j] = poli_count[i + 1][j] + poli_count[i][j - 1] -
                    poli_count[i + 1][j - 1];
            }
            i++;

        }
    }

    cout << poli_count[0][n - 1] << endl;

    return 0;
}
```

main.cpp	
int main()	Основная функция работы программы

3 Консоль

```
nixx@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025.2/7lab$ g++ -o p main.cpp
nixx@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025.2/7lab$ ./p
baobab
22
nixx@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025.2/7lab$
```

4 Тест производительности

Тестирование заключается в проведении нескольких тестовых запусков программы с разными тестовыми данными с увеличением длины входных строк и последующим сравнением результатов и времени работы с наивным алгоритмом.

Объёмы тестовых данных: Строки по 10, 20, 50, 100, 200, 500 символов.

Ожидаемые результаты: Незначительное ускорение (3-5x) для малых строк (10-20 символов) Существенное ускорение (50-100x) для средних строк (50-100 символов) Экспоненциальный рост ускорения для больших строк (200+ символов)

```
nixx@DESKTOP:/mnt/c/Users/Max Kar/Desktop/МАИ/ДА/2025.2/7lab/bench$ ./benchmark

=== ТЕСТ ПРОИЗВОДИТЕЛЬНОСТИ ===
Размер | Naive (мкс) | DP1 (мкс) | Результат
-------|-------------|-----------|----------
10 |          187 |13         |103
20 |       292079 |44         |6729
50 |       144941 |124        |8070476822
100 |155526374478 |565        |4713264851383175735
200 |          N/A | 1413       |6678461532314472470
500 |          N/A | 7714       |2144192560048506586
```

5 Выводы

В ходе выполнения работы была исследована задача подсчета палиндромных подстрок в строке и реализованы два алгоритма для ее решения: наивный алгоритм и алгоритм на основе динамического программирования.

Основные достижения:

Эффективность алгоритмов: Алгоритм на основе динамического программирования показал значительное превосходство над наивным подходом. При увеличении длины входной строки разница в производительности становится экспоненциально больше.

Асимптотическая сложность: Теоретические оценки сложности алгоритмов подтвердились практическими измерениями:

Наивный алгоритм: $O(n^3)$

Алгоритм ДП: $O(n^2)$

Практическая применимость: Для коротких строк (до 20 символов) оба алгоритма демонстрируют приемлемое время выполнения, однако для строк длиной более 50 символов наивный алгоритм становится практически неприменимым из-за резкого роста времени выполнения.

Масштабируемость: Алгоритм динамического программирования обладает хорошей масштабируемостью и может обрабатывать строки длиной до 1000 символов за приемлемое время, в то время как наивный алгоритм уже для строк длиной 200 символов требует неоправданно больших вычислительных ресурсов.

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.))