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: В. С. Епанешников
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №7

Вариант №6

Задача: При помощи метода динамического программирования разработать алгоритм решения задачи, определяемой своим вариантом; оценить время выполнения алгоритма и объем затрачиваемой оперативной памяти. Перед выполнением задания необходимо обосновать применимость метода динамического программирования.

Разработать программу на языке C или C++, реализующую построенный алгоритм. Формат входных и выходных данных описан в варианте задания:

Задана строка S состоящаяя 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, отличающиееся только порядком вычеркивания символов, считаются одинаковыми.

Формат входных данных: Задана одна строка S ($|S| \leqslant 100$).

Формат результата: Необходимо вывести одно число - ответ на задачу. Гарантируется, что он $\leqslant 2^{63} - 1$.

1 Описание

Для того, чтобы решить задачу, создаётся 1 матрица matr для того, чтобы хранить число палиндромов-подпоследовательной для каждой промежуточной подстроки с i-ого до j-го индекса.

Если в позициях i и j стоят разные символы, то необходимо использовать выражение $A[i, j] := A[i + 1, j] + A[i, j - 1] - A[i + 1, j - 1]$

Если одинаковые - $A[i, j] := 1 + A[i + 1, j] + A[i, j - 1]$.

В самом деле, если символы различны, то один из них обязательно должен быть вычеркнут. Надо не забыть, что вычеркивания, когда были вычеркнуты оба символа (i-ый и j-ый) были посчитаны дважды, их нужно вычесть. Если же символы совпадают, то кроме уже описанного можно взять любой палиндром, получаемый из подстроки с (i+1)-го по (j-1)-ый символ, и к нему спереди и сзади прибавить этот одинаковый символ. После этого осталось сделать начальное заполнение $A[i, i] = 1$ и посчитать значения $A[i, j]$ в порядке увеличения разности (j - 1).

2 Исходный код

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

// bottom-up DP
unsigned long long psubsequences_count(const std::string& s) {
    int size = s.size();
    std::vector<std::vector<unsigned long long> > matr;
    matr.resize(size);

    for (int i = 0; i < size; ++i) {
        matr[i].resize(size, 0);
    }

    for (int i = 0; i < size; ++i) {
        matr[i][i] = 1;
    }

    for (int l = 2; l <= size; ++l) {
        for (int i = 0; i <= size - l; ++i) {
            int k = l + i - 1;
            if (s[i] == s[k]) {
                matr[i][k] = matr[i][k - 1] + matr[i + 1][k] + 1;
            }
            else {
                matr[i][k] = matr[i][k - 1] + matr[i + 1][k] - matr[i + 1][k - 1];
            }
        }
    }

    return matr[0][size - 1];
}

// top-down DP
unsigned long long psubsequences_count2(const std::string& s, int i, int j) {
    if (i == j) {
        return 1;
    }
    if (i > j) {
        return 0;
    }

    if (s[i] == s[j]) {
        return 1 + psubsequences_count2(s, i + 1, j) + psubsequences_count2(s, i, j -
            1);
    }

```

```
47      return psubsequences_count2(s, i + 1, j) + psubsequences_count2(s, i, j - 1)
48          - psubsequences_count2(s, i + 1, j - 1);
49  }
50
51  int main() {
52      std::string s;
53      std::cin >> s;
54
55      // bottom-up DP
56      std::cout << psubsequences_count(s) << "\n";
57
58      // top-down DP with recursion
59      //std::cout << psubsequences_count2(s, 0, s.size() - 1) << "\n";
60
61      return 0;
62  }
```

3 Консоль

```
MacBook:exercise_07 vladislove$ g++ main.cpp
MacBook:exercise_07 vladislove$ ./a.out
BAOBAB
22
MacBook:exercise_07 vladislove$ ./a.out
CCCCCCCC
255
```

4 Тест производительности

Для теста производительности использовалась наивная реализация решения задачи. Тест проводился при S = 100.

```
MacBook:exercise_07 vladislove$ ./da7
naive alg: 11000060 microseconds
dp: 1150 microseconds
```

Как видно, даже при небольших S время работы наивной реализации значительно больше.

5 Выводы

Выполнив седьмую лабораторную работу по курсу «Дискретный анализ», я вспомнил принципы динамического программирования и научился решать различные типы задач. Кроме того, увидел насколько эффективен алгоритм с применением динамического программирования по сравнению с наивным решением задачи.

Список литературы

[1] *Динамическое программирование*

`https://acm.khpnets.info/wiki/ (дата обращения 21.04.2021)`