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: Л. Я. Вельтман
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б

Дата:
Оценка:
Подпись:

Москва, 2019

Лабораторная работа №7

1 Описание

На вход подается строка str. Передаем ее в функцию Palindrome. Там объявляем двумерный массив plnd, он будет хранить количество палиндромов, которое можно получить из подстрок исходной строки str удалением букв. Заполняем plnd[i][i] = 1, так как слово из одного символа является палиндромом. Перебираем длины подстрок len и позиции их начала i.Для каждой такой подстроки str[i]...str[j] вычисляем значение plnd[i][j] – количество палиндромов, которое можно получить из нее удалением символов. Существует два случая, когда начало и конец подстроки равны и не равны между собой. В первом случае общее количество палиндромов равно числу палиндромов в строке, включающей самый левый символ и не включающий самый правый символ + число панлиндромов в строке, включающей самый правый символ и не включающий самый левый символ + палиндром, состоящий только из самого левого и самого правого символа + палиндромы, находящиеся между самым крайним левым и самым крайним правым символом * 2, так как с каждым панлидромом такой строки можно тоже построить палиндромы. Второй случай: общее количество палиндромов равно числу палиндромов в строке, включающей самый левый символ и не включающий самый правый символ + число палиндромов в строке, включающей самый правый символ и не включающий самый левый символ + палиндромы, находящиеся между самым крайним левым и самым крайним правым символом. Поскольку подстроки str[i]...str[j] перебираются в порядке возрастания их длин, то значения plnd на всех подотрезках меньшей длины уже вычислены.

2 Исходный код

```
#include <iostream>
#include <string>

const int MAX = 100;
uint64_t Palindrome(const std::string&);

int main(int argc, const char * argv[]) {
    std::string str;
    std::cin >> str;
    std::cout << Palindrome(str) << std::endl;
    return 0;
}

uint64_t Palindrome(const std::string& str) {
    int sizeStr = (int) str.size();
    if (!sizeStr) return 0;

    uint64_t plnd[MAX][MAX];
    for (int i = 0; i < sizeStr; ++i) {
        for (int j = 0; j < sizeStr; ++j) {
            if (i == j) {
                plnd[i][j] = 1;
            }
            else {
                plnd[i][j] = 0;
            }
        }
    }

    for (int len = 1; len < sizeStr; ++len) {
        for (int i = 0, j = len; i < sizeStr - len; ++i, ++j) {
            if (str[i] == str[j]) {
                plnd[i][j] = plnd[i][j-1] + plnd[i+1][j] + 1;
            }
            else {
                plnd[i][j] = plnd[i][j-1] + plnd[i+1][j] - plnd[i+1][j-1];
            }
        }
    }

    return plnd[0][sizeStr - 1];
}
```

3 Консоль

```
MacBook-Pro-Lina:da7 linuxoid$ make
g++ -std=c++11 -Wall -Werror -Wno-sign-compare -Wno-unused-result -O3 -o da7
main.cpp
MacBook-Pro-Lina:da7 linuxoid$ ./da7
ABAODDUEEDAAASBAB
720
MacBook-Pro-Lina:da7 linuxoid$ ./da7
URURUR
26
MacBook-Pro-Lina:da7 linuxoid$ ./da7
IOIOADIDAOI
161
MacBook-Pro-Lina:da7 linuxoid$ ./da7
POTOP
13
MacBook-Pro-Lina:da7 linuxoid$ ./da7
TRYTRYTRYTRYTRY
1023
```

4 Тест производительности

```
MacBook-Pro-Lina:da7 linuxoid$ ./da7
uiuiuiu
53
DP time: 0.000039 sec
MacBook-Pro-Lina:da7 linuxoid$ ./da7
ERYERYERYERYERYERY
4095
DP time: 0.000053 sec
MacBook-Pro-Lina:da7 linuxoid$ ./da7
QUOOOOOOQUOQ
485
DP time: 0.000068 sec
```

Решение, основанное на динамическом программировании показало хороший результат. Поиск 4096 способов вычеркиваний выполнился всего лишь за несколько миллисекунд. Человек такой результат получил бы очень нескоро.

5 Выводы

Динамическое программирование – это способ решения сложных задач путем разбиения их на более простые подзадачи. Он применим к задачам с оптимальнои подструктурои, выглядящим как набор перекрывающихся подзадач, сложность которых чуть меньше исходнои: в этом случае время вычислении можно значительно сократить. Как правило, чтобы решить поставленную задачу, требуется решить отдельные части задачи (подзадачи), после чего объединить решения подзадач в одно общее решение. Часто многие из этих подзадач одинаковы. Подход динамического программирования состоит в том, чтобы решить каждую под- задачу только один раз, сократив тем самым количество вычислении. Это особенно полезно в случаях, когда число повторяющихся подзадач экспоненциально велико. Этот метод зачастую позволяет построить ускоренную и улучшенную версию алгоритма. Всюду, где имеются перекрывающиеся подзадачи, и где эти самые подзадачи относительно легко выделить, динамическое программирование находит здесь свое применение. Оно может оказать неоценимую поддержку быстродеиствию программы. Таким образом, выполнив данную лабораторную работу, я научилась решать данную задачу при помощи динамического программирования.

Список литературы

[1] 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.))