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: П. А. Мохляков
Преподаватель: Н. С. Капралов
Группа: М8О-308Б

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №7

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

Вариант: 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 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

Динамическое программирование — это метод решение задачи путем разбиения ее на более малые, простые подзадачи. В нашем случае нужно найти количество вариантов вычеркиваний приводящих нашу строку к акронимому.

Давайте попробуем выделить меньшие подзадачи. Учитывая независимость от порядка вычеркиваний можно выделить три подзадачи: количество вариантов без первого символа, количество вариантов без последнего символа и количество вариантов без первого и последнего символов. Давайте рассмотрим вариант задачи где первый и последний символ отличаются.

В таком случае количество вариантов вычеркиваний равно: количеству вариантов без последнего символа плюс количество вариантов без последнего символа минус количество вариантов без последнего символа и количество вариантов без первого и последнего символов. Мы производим вычитание так как два других слогаемых содержат в себе его, и из за это мы считали бы его 2 раза.

$$Vars(str(i,j)) = Vars(str(i+1,j)) + Vars(str(i,j-1)) - Vars(str(i+1,j-1))$$

Теперь рассмотрим варианты где первый пи последний символ равны. В этом случае все схоже за исключением пары оссобеностей. Во первых если мы вычтем всю середину, то получим еще один полиндром. Во вторых если мы к каждому полиндрому середины прибавим по одинаковому символу слева и справа, то тоже получаем полиндром, то есть мы должны прибавить еще количество вариантов без первого и последнего символов. Слагаемые сокращаются и мы получаем: количеству вариантов без последнего символа плюс количество вариантов без последнего символа плюс один.

$$Vars(str(i,j)) = Vars(str(i+1,j)) + Vars(str(i,j-1)) + 1$$

2 Исходный код

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

uint64_t Variants(std::string &str,uint64_t presum[100][100], int i, int j){
    if(j < i){
        return 0;
    } else {
        if(str[i] == str[j]){
            if(presum[i][j] == 0){
                presum[i][j] = Variants(str,presum,i + 1,j) + Variants(str,presum,i,
                    j - 1) + 1;
            }
            return presum[i][j];
        } else {
            if(presum[i][j] == 0){
                presum[i][j] = Variants(str,presum,i + 1,j) + Variants(str,presum,i,
                    j - 1) - Variants(str,presum,i + 1,j - 1);
            }
            return presum[i][j];
        }
    }
}

int main(){
    uint64_t presum[100][100];
    std::memset(presum,0,sizeof(uint64_t)*10000);
    std::string str;
    std::cin >> str;
    std::cout << Variants(str,presum,0,str.size() - 1) << std::endl;
    return 0;
}
```

3 Консоль

```
pavel@DESKTOP-SVKRTNN ~/work/MAI/2_course/DA/LB7 cat test
BAOBAB
pavel@DESKTOP-SVKRTNN ~/work/MAI/2_course/DA/LB7 g++ main.cpp -o solution
pavel@DESKTOP-SVKRTNN ~/work/MAI/2_course/DA/LB7 cat test | ./solution
22
```

4 Тест производительности

```
String size: 5
Dynamic: 7.22e-07
Naive: 5.591e-06
String size: 10
Dynamic: 2.314e-06
Naive: 8.1853e-05
String size: 25
Dynamic: 8.847e-06
Naive: 3.16346
String size: 30
Dynamic: 1.1181e-05
Naive: 124.613
```

Далее наивный алгоритм работает слишком долго, поэтому дальше мы рассмотри только динамический алгоритм.

```
String size: 50
Dynamic: 2.9234e-05
String size: 101
Dynamic: 0.000116328
String size: 1000
Dynamic: 0.0200347
String size: 10000
Dynamic: 1.68691
```

Как видно наивный алгоритм прогирывает динамическому. Это не удивительно, ведь сложность наивного алгоритма выше, чем у динамического.

5 Выводы

Выполнив лабораторную работу по курсу «Дискретный анализ», я познакомился с методом динамического программирования. С помощью этого метода решается большинство задач оптимизации. Тем не менее решение некоторых задач может занять очень много времени, поэтому, если это возможно, вместо этого метода используют метод жадных алгоритмов.

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.))