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: А. Н. Марков
Преподаватель: Н. А. Зацепин
Группа: М8О-308Б
Дата:
Оценка:
Подпись:

Москва, 2020

Условие

1. Общая постановка задачи

   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:

2. Вариант задания

   Вариант 6. 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, отличающиееся только порядком вычеркивания символов, считаются одинаковыми.

3. Формат входных данных

   Задана одна строка S ($|S| \leq 100$).

4. Формат результата

   Необходимо вывести одно число – ответ на задачу. Гарантируется, что он $\leq 2^{63}-1$.

Метод решения

Динамическое программирование позволяет решать задачи, комбинируя решения вспомогательных задач. Таким образом, для решения задачи нужно решить подзадачи, и из результатов подзадач сформировать результат задачи. Подзадачи могут быть одинаковы. В алгоритме динамического программирования каждая подзадача решается только один раз, после чего ответ сохраняется в таблице. Это позволяет избежать одних и тех же повторных вычислений каждый раз, когда встречается данная, уже ранее решенная, подзадача.

Моя задача эквивалентна задаче поиска количества уникальных палиндромных подпоследовательностей, поскольку каждой уникальной палиндромной подпоследовательности можно поставить в соответствие набор индексов символов, вычеркивающихся из строки. Такая задача идеально подходит для решения алгоритмом динамического программирования, поскольку она прекрасно разбивается на подзадачи, решения которых перекрываются.

Введем двумерный массив D: $D[i][j]$ - количество палиндромных подпоследовательной в строке, начинающейся с символа, имеющего индекс i, и заканчивающаяся в символе, имеющем индекс j.

- Если $i = j$, то строка состоит из одного символа, а значит является палиндромом.
- Если $S[i] \neq S[j]$, то количество палиндромов в строке $S[i..j]$ равно количеству палиндромов в строке $S[i+1..j]$ плюс количество палиндромов в $S[i..j-1]$ минус количество палиндромов в $S[i+1..j-1]$. Вычитание здесь необходимо, поскольку при сложении количеств палиндромов в $S[i+1..j]$ и $S[i..j-1]$ дважды будут учитываться палиндромы из строки $S[i+1..j-1]$.
- Если $S[i] = S[j]$, то ситуация такая же как и в случае $S[i] \neq S[j]$, но нужно учитывать, что строка, состоящая только из символов $S[i]$ и $S[j]$, тоже является палиндромом, а также добавятся палиндромы вида $A\$A$, где $A = S[i] = S[j]$, а \$ - палиндром из $S[i+1..j-1]$, таких палиндромов будет столько же, сколько и палиндромов в $S[i+1..j-1]$.

Таким образом, можно сформировать рекуррентное соотношение:

$$D[i][j] = \begin{cases} 1, i = j \\ 0, i > j \\ 1 + D[i+1][j] + D[i][j-1], S[i] = S[j] \\ D[i+1][j] + D[i][j-1] - D[i+1][j-1], S[i] \neq S[j] \end{cases}$$

Описание программы

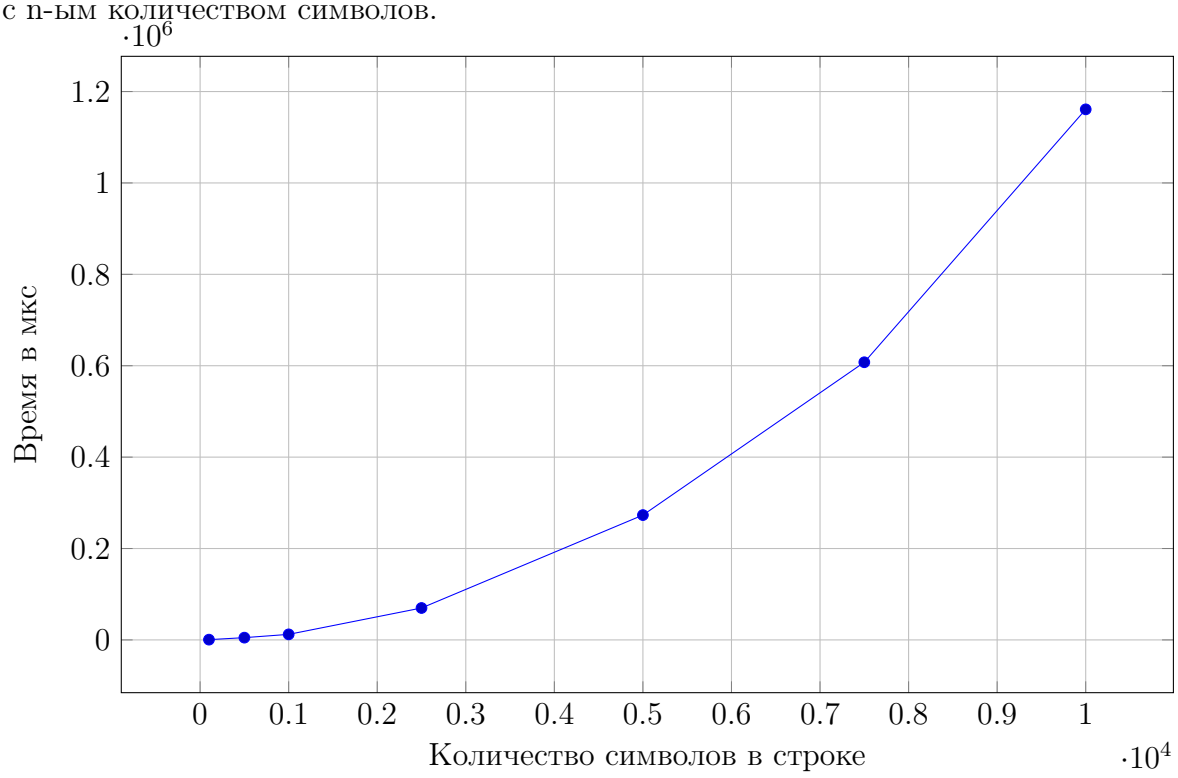
Программа состоит из одного файла main.cpp, в котором находится восходящая реализация алгоритма и функция main.

Дневник отладки

Программа зашла на чекер с первой попытки.

Тест производительности

Тесты создавались с помощью небольших программы generator.py. Создавались строки с n-ым количеством символов.



Выводы

Динамическое программирование полезно, когда задача, разбивается на более простые подзадачи, которые могут перекрываться. Решив такие подзадачи, можно сформировать ответ на первоначальную задачу.

В данной лабораторной работе было сложнее понять, как решать поставленную задачу, нежели реализовать ее программно.