Лабораторная работа № 7 по курсу дискретного анализа: Динамическое программирование

Выполнил студент группы 08-208 МАИ *Куликов Алексей*.

Условие

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 зададния.

В качестве конкретного задания предлагается решить следующую задачу (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.

Метод решения

Исходная задача эквивалентна задаче по нахождению всех уникальных подпоследовательностей, являющихся палиндромами т.к. каждой уникальной подпоследовательности соответствует свой набор индексов, элементы по которым как бы удаляются, для получения палиндрома.

Будем подсчитывать количество подпоследовательностей-палиндромов следующим образом:

- если `S[i] != S[j]`, то `C[i][j] = C[i][j-1] + C[i+1][j] - C[i+1][j-1]`,
- иначе, если `S[i] == S[j]`, то `C[i][j] = C[i][j-1] + C[i+1][j] + 1`,

где S[i] – символ строки S по индексу i, $C[i][j]$ – количество подпоследовательностей-палиндромов, содуржащихся в $S[i...j]$.

Это так потому что, если символы на конце и начале подстроки не совпали, палиндромы могут существовать с участием с 1-го символа текущей подстроки или же с участием последнего символа. Поэтому сложим количество палиндромов для подстроки без первого символа `C[i+1][j]`, с количеством палиндромов в подстроке без последнего `C[i][j-1]`. Но, таким образом, палиндромы, содержащиеся в середине подстроки (подстрка без первого и без последнего символа) будут учтены дважды. Поэтому вычтем их количество `C[i+1][j-1]` из полученного результата.

Для случая с совпавшими символами, ситуация такая же, но т.к. символы по краям равны, то добавятся новые палиндромы вида `A_A`, где `A = S[i] = S[j]`, а `_` – каждый из палиндромов из `S[i+1, j-1]`(т.е. из серединки). Таких палиндромов ровно `C[i+1, j-1]` штук. Кроме того добавится еще один палиндром `AA`.

Описание программы

Программа состоит из единственного исходного файла. В нем записано рекурсивное решение согласно алгоритму, описанному выше, с применением мемоизации.

Дневник отладки

1. 07.04 Не угадал с типом. Оказалось что искомых подпоследовательностей может быть очень много.
   РЕШЕНИЕ: Заменил тип переменных с количеством подпалиндромов с `int` на `long long`.

Тест производительности

Рис. 1: Зависимость времени работы алгоритма от длины строки)

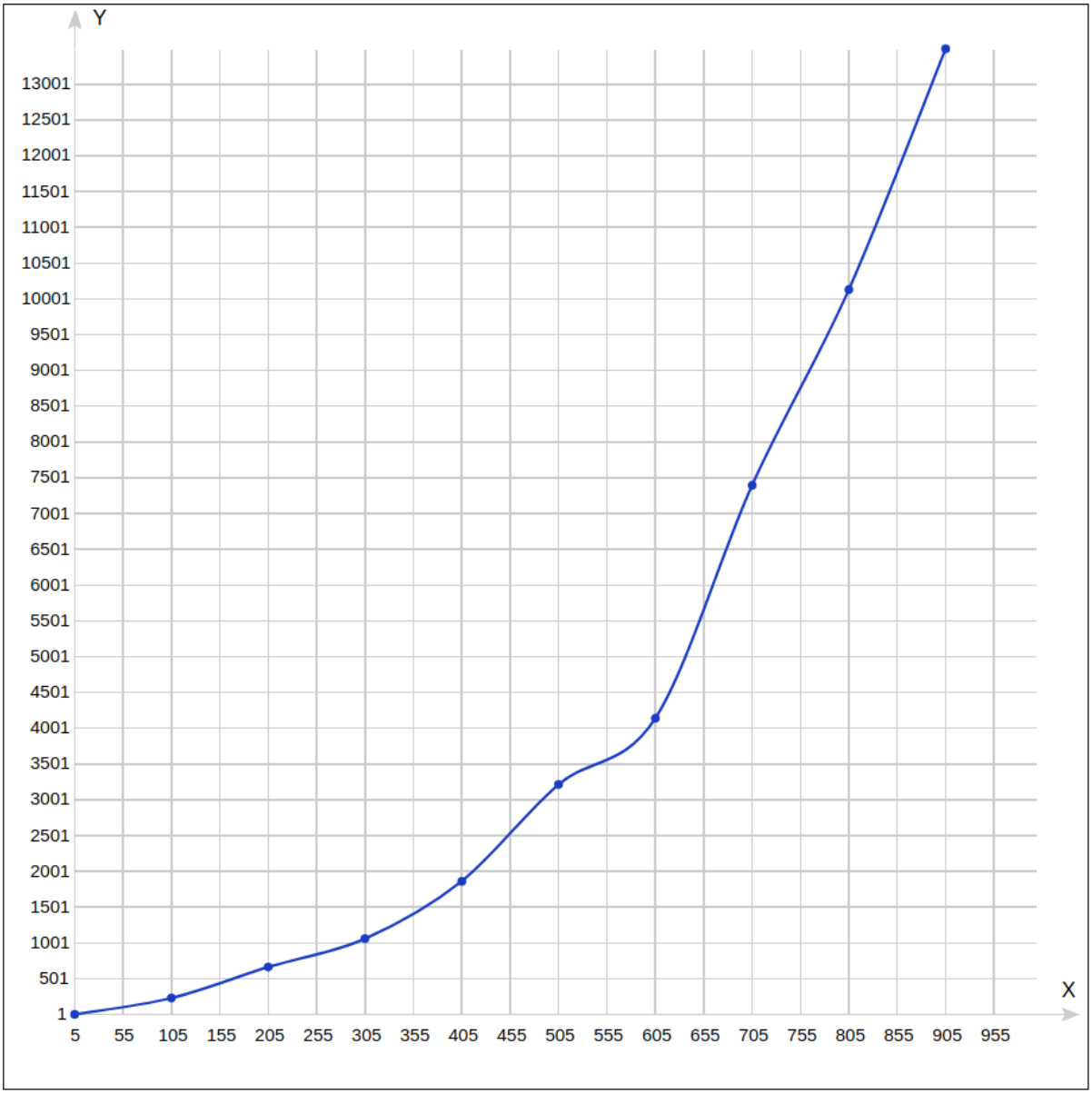
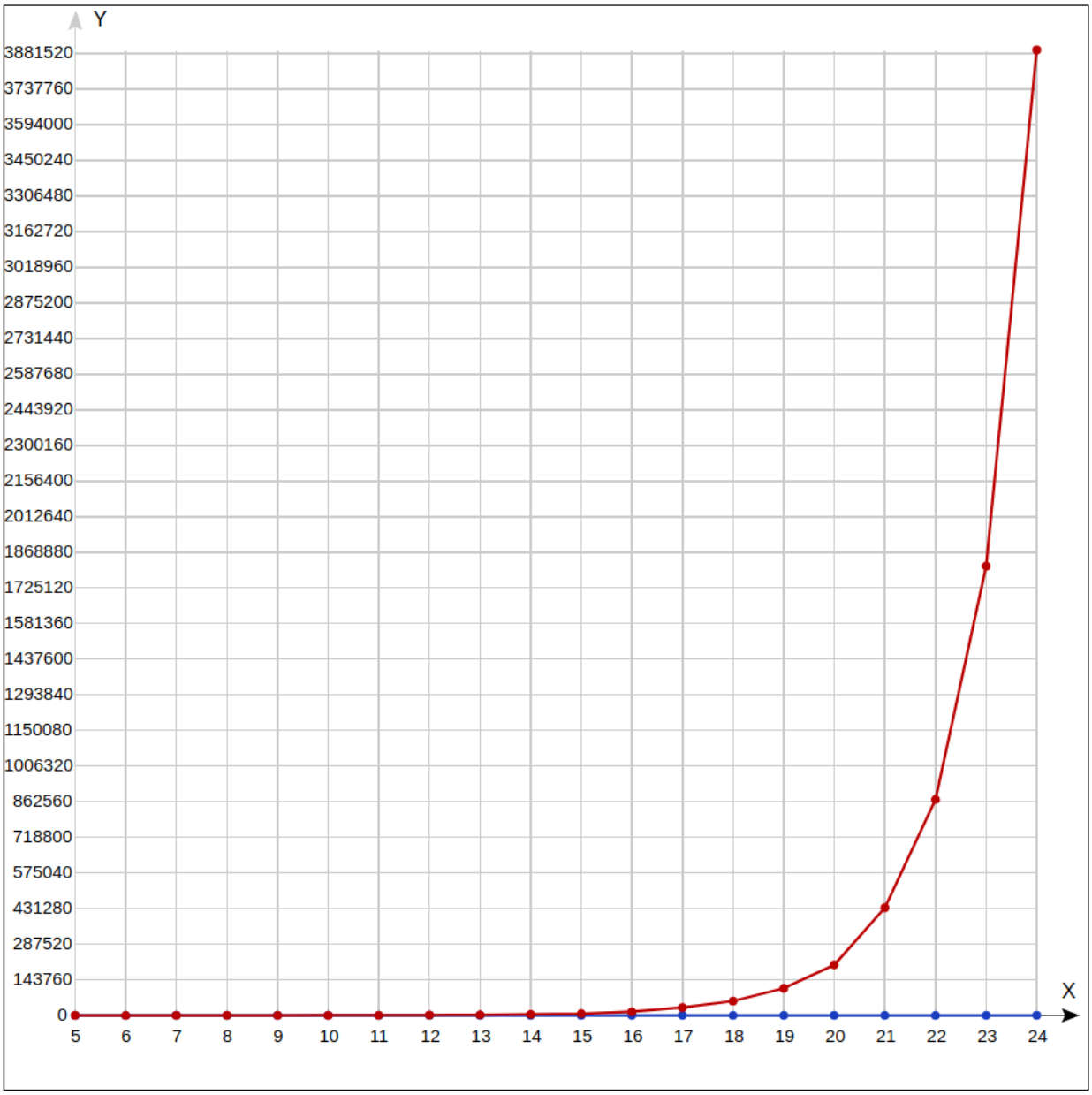


Рис. 2: Сравнение решения перебором и с помощью динамисеского программирования)



Из графиков можно видеть, что решение перебором неприемлимо, т.к. уже для строки длины 24 время выполнения приближается к 4 секундам т.к. зависит экспоненциально от количества данных. Поэтому для решения этой задачи и испоьзуется динамическое программирование.

Выводы

Динамическое программирование может быть полезным в задачах, структура которых предполагает деление на схожие подзадачи меньшего размера (возможно пересекающиеся), решив которые, можно построить решение исходной задачи.

В решении данной задачи самым трудным (но не очень) было дойти до этой идеи, сама реализация трудностей не вызвала.