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. А. Кабанов
Преподаватель: С. А. Михайлова
Группа: М8О-301Б-22

Дата:
Оценка:
Подпись:

Москва, 2024

Лабораторная работа №7

Задача: Палиндромы.

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.

Формат ввода: Строка S ($|S| \leq 100$).

Формат вывода: Число, меньшее $2^{63} - 1$

1 Описание

Исходную строку обозначим S, количество способов вычеркиваний подстроки $S[i, j]$ - $d(S[i, j])$.

Начнем решать задачу с простых подпоследовательностей. Однобуквенная подпоследовательность является палиндромом, поэтому $dp(S[i, i]) = 1$.

Для двухбуквенной последовательности $S[i, i + 1]$ может существовать два варианта ответа:

1. $d(S[i, i + 1]) = 2$, когда $S[i] \neq S[i + 1]$;
2. $d(S[i, i + 1]) = 3$, когда $S[i] = S[i + 1]$.

Это база ДП. Мы можем распространить этот принцип на подстроки длины больше 2:

1. $d(S[i, j]) = d(S[i + 1, j]) + d(S[i, j - 1]) + 1$, если $S[i] = S[j]$;
2. $d(S[i, j]) = d(S[i + 1, j]) + d(S[i, j - 1]) - d(S[i + 1, j - 1])$, если $S[i] \neq S[j]$.

Заполним матрицу D этими значениями. Получится матрица с нулями ниже главной диагонали и ненулевыми значениями сверху (поэтому мы можем распространить принцип на более сложные случаи). Ответ находится в клетке $D[0][|S| - 1]$.

2 Исходный код

Реализуем алгоритм, описанный в предыдущем пункте, считав строку, обработав простой случай и распространив вычисления на другие (более сложные) случаи.

```
#include <bits/stdc++.h>

using namespace std;

int main() {
  string s;
  cin >> s;
  size_t s_sz = s.size();
  uint64_t D[s_sz][s_sz];
  for (size_t i = 0; i < s_sz; ++i) {
    for (size_t j = 0; j < s_sz; ++j) {
      if (i == j) { // one char string is a palindrome
        D[i][j] = 1;
      } else if (i > j) {
        D[i][j] = 0;
      }
    }
  }

  for (size_t len = 2; len <= s_sz; ++len) {
    for (size_t i = 0; i <= s_sz - len; ++i) {
      size_t j = i + len - 1;
      if (s[i] == s[j]) {
        D[i][j] = D[i + 1][j] + D[i][j - 1] + 1;
      } else {
        D[i][j] = D[i + 1][j] + D[i][j - 1] - D[i + 1][j - 1];
      }
    }
  }

  cout << D[0][s_sz - 1] << endl;
}
```

3 Консоль

```
[anton@home lab7-dp]$ make
g++ -O2 -lm -fno-stack-limit -std=c++20 -x c++ solution.cpp -o executable
[anton@home lab7-dp]$ ./executable
BAOBAB
22
```

4 Тест производительности

Тест производительности представляет из себя следующее: сравнение написанного мной алгоритма за $O(n^2)$ и наивного алгоритма за $O(2^n)$.

```
[anton@home lab7-dp]$ make
g++ main.cpp -o main
g++ naive.cpp -o naive
[anton@home lab7-dp]$ ./main < tests/test0.txt
[DP] Done in 2013 microseconds.
[anton@home lab7-dp]$ ./naive < tests/test0.txt
[NAIVE] Done in 3497 microseconds.
[anton@home lab7-dp]$ ./main < tests/test1.txt
[DP] Done in 2822 microseconds.
[anton@home lab7-dp]$ ./naive < tests/test1.txt
[NAIVE] Done in 2409338 microseconds.
```

Видно, что ДП-алгоритм выигрывает у наивного при увеличении длины строки в несколько тысяч раз, несмотря на большее потребление памяти.

5 Выводы

Выполнив седьмую лабораторную работу по курсу «Дискретный анализ», я познакомился с динамическим программированием как способом решения различных задач. Он позволяет ускорять решения при помощи разбиения задачи на более мелкие независимые. При этом необходимо создать оптимальные подзадачи.

Процесс разработки алгоритмов динамического программирования состоит из следующих шагов:

1. описание структуры оптимального решения;
2. рекурсивное определение значения оптимального решения;
3. вычисление значения с помощью метода восходящего анализа;
4. составление решения на основе оптимального решения.

Разработанный алгоритм был идейно похож на алгоритм решения задачи о наибольшей подпоследовательности-палиндроме, поэтому было интересно применить идею к решению лабораторной работы.

Список литературы

[1] *Динамическое программирование - Викиконспекты*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Динамическое_программирование`
(дата обращения: 23.10.2024).

[2] *Задача о наибольшей подпоследовательности-палиндроме - Викиконспекты*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Задача_о_наибольшей_подпоследовате`
(дата обращения: 23.10.2024).