Algorithms

Study Notes

Zhijie Xia

Study Notes

github.com/zhijie-os
zhijiexia.website

Contents

Chapter 1

Dynamic Programming

1.1 Palindromic Subsequence

1.1.1 Longest Palindromic Subseqence

Problem Statment:

Given a sequence, find the length of its Longest Palindromic Subsequence(LPS). In a palindromic subsequence, elements read the same backward and forward.

A subsequence is a sequence that can be derived from another sequence by deleting some or no elements without changing the order of the remaining elements.

Examples:

Input: "abdbca"
Output: 5
Explanation: LPS is "abdba"

Input: "cddpd"
Output: 3
Explanation: LPS is "ddd"

Basic Solution:

A basic brute-force solution could be to try all the subsequences of the given sequence. We can start processing from the beginning and the end of the sequence. So at any step, we have two options:

1. If str[begin]==str[end], increment counter by two. Subproblem: LPS in str[begin+1][end-1].
2. If str[begin]!=str[end], nothing. Subproblem: max LPS in str[begin+1][end] and LPS in str[begin][end-1].

Bottom Up Idea:

n = string.length ⇒ dp[n][n] and the subproblem is $dp[i][j]$: the LPS in substring str[i:j].
The solution to dp[i][j]:

1. str[i] == str[j] ⇒ dp[i+1][j-1]+2
2. str[i] != str[j] ⇒ max(dp[i+1][j],dp[i][j-1])

The solution to the LPS original problem is dp[0][n] which is the top right corner. Also, in 2×2 block, top right corner is based on the surrounding three.
⇒ populate the table in the following order:

1. From left to right.
2. From bottom to top.

Bottom Up Code:

```
class LPS{
    public:
        int findLPSLength(const string &st){
            vector<vector<int>> dp(st.length(), vector<int>(st.length(), 0));

            // every sequence with one element is a palindrome of length 1
            for (int i = 0; i < st.length(); i++) {
                dp[i][i] = 1;
            }

            // rows from bottom to up
            for (int startIndex = st.length() - 1; startIndex >= 0; startIndex--) {

                // column from left to right
                for (int endIndex = startIndex + 1; endIndex < st.length(); endIndex++) {

                    // case 1: elements at the beginning and the end are the same
                    if (st[startIndex] == st[endIndex]) {

                        dp[startIndex][endIndex] = 2 + dp[startIndex + 1][endIndex - 1];
                    }
                    else { // case 2: skip one element either from the beginning or the end

                        dp[startIndex][endIndex] =
                            max(dp[startIndex + 1][endIndex], dp[startIndex][endIndex - 1]);
                    }
                }
            }

            return dp[0][st.length() - 1];
        }
    }
```

1.1.2 Longest Palindromic Substring:

Problem Statment

Given a string, find the length of its Longest Palindromic Substring (LPS). In a palindromic string, elements read the same backward and forward.

Examples:

Input: "abdbca"
Output: 3
Explanation: LPS is "bdb"

Input: "cddpd"
Output: 3
Explanation: LPS is "dpd"