Dynamic Programming Question Collections

·Maximum Subarray(最大子段和) (1.4.1)

Given an integer array `nums`, find the contiguous subarray (containing at least one number) which has the largest sum and return *its sum*. Leet code 53

- f(i): The maximum subarray of nums[0:i] containing nums[i]
- f(0) = nums[0]
- f(i) = max(f(i-1),0) + nums[i]

my code

·Longest Common Subsequence(最长公共子序列) (1.4.2)

Given two strings `text1` and `text2`, return *the length of their longest* ***common subsequence***. If there is no **common subsequence**, return `0`.

A **subsequence** of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

- For example, `"ace"` is a subsequence of `"abcde"`.

A **common subsequence** of two strings is a subsequence that is common to both strings. leet code 1143

Recursive solution:

- LCS(text1, text2): return the longest common subsequence of text1 and text2
- if text1.length()==0 or text2.length() == 0: return 0
- if text1.length()==1 and text2.length() == 1: return text1 == text 2
- if text1[n_1] == text1[n_2]: LCS(text1,text2) = LCS(text1[1:n_1-1], text2[1:n_2-1]) + 1
- if text1[n_1] != text1[n_2]: LCS(text1,text2) = MAX(LCS(text1, text2[1:n_2-1]) , LCS(text1[1:n_1-1], text2))

my code

T(n,m) = MAX(T(n-1,m-1)+ O(1), T(n-1,m)+T(n,m-1)+O(1))

Iterative solution:

- f(i,j): the longest common subsequence of text1[1:i] and text2[1:j]; 1<=i<= n,1<=j<=m
- f(i,0) = f(0,j)=0
- f(i,j): if (text1[i]==text2[j]) f(i,j) = 1+ f(i-1,j-1); else f(i,j) = MAX(f(i,j-1), f(i-1,j)), i,j >= 1

my code

·Memoization(记忆化搜索) (1.4.4)

- **Word Break** [leetcode 139]

  Given a string `s` and a dictionary of strings `wordDict`, return `true` if `s` can be segmented into a space-separated sequence of one or more dictionary words.

  Note that the same word in the dictionary may be reused multiple times in the segmentation.

  Recursive version

  WB(s) = for (int i=0; i< s.length(); i++) any s[0:i] in wordDict and WB(s[i+1:])

  Iterative version

  f(i): whether the s[0:i-1] can be constructed with words in wordDict

  f(0)= 0

  f(i) = OR(f(j) and check(s[j:i-1])) for some j in {1,..., i-1}, where check means checking whether the string is in wordDict

  my solution

  Time complexity: $O(n^2)$

- **Longest common subsequence of m strings**

  Given m strings, and find their longest common subsequence

  the idea is similar to the LCS of two strings, think about how we solved the longeszt common string:

  1. Use a 2-D array storing the the status, where $A[i][j]$ represents the longest common subsequence of string $s_1[1...i]$ and $s_2[1...j]$
  2. For m strings we can use a m-D array $A[k_1][k_2]...[k_m]$ where:
     - $A[k_1][k_2]...[k_m] = \emptyset$, if any exist any k_i = 0 (i.e. empty string)
     - $A[k_1][k_2]...[k_m] = 1 + A[k_1 - 1][k_2 - 1]...[k_m - 1]$, if $s_1[k_1] == s_2[k_2] == ... == s_m[k_m]$
     - $A[k_1][k_2]...[k_m] = \max\{A[k_1]...[k_i - 1]...[k_m]\}$

- **String editing:**

  Given two string s_1, s_2 of size m, n respectively, there are three supported unit operations to edit the string:

  - Insert: insert one char in any position of the string
  - Delete: delete one char of any position in the string
  - Edit: edit one char in the string to any other char.

  df(i,j): the minimum steps to edit $s_1[1...i]$ to $s_2[1...j]$

  - df(i,j) = m, if n == 0 (just delete all chars)
  - df(i,j) = n, if m == 0 (just insert all correspond strings)

- d(i,j) = d(i-1,j-1), if $s_1[n] == s_2[m]$
  - Modify by insert: d(i,j) = d(i, j-1) + 1 (insert char $s_2[m]$ at the end)
  - Modify by delete: d(i.j) = d(i-1,j) + 1 (delete char $s_1[n]$ at the end)
  - Modify by editing: d(i,j) = d(i-1, j-1) + 1 (edit the $s_1[n]$ to $s_2[m]$ directly)

·最优矩阵链乘 (1.4.5)

·最优三角剖分 (1.4.6)

·背包问题（背包九讲）(1.4.7)

·滚动数组优化空间 (1.4.8)

·状压dp (1.4.9)

·区间dp (1.4.10)

·一般dp题（会自己想状态和转移方程）(1.4.11)

数位ar子集中sosdp (2.1.1)

矩阵加速dp (2.1.2)

四边形优化区间dp长链剖分优化 (2.1.3)

关链剖分优上下界优化 (2.1.4)

1D/1Ddp优化（单调队列优化、单调栈优化、分治优化、斜率优化）插头dp (2.1.5)

LIS转LCS (2.1.6)