

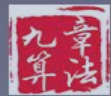
Dynamic Programming II

课程尚未开始, 请大家耐心等待



关注微信
ninechapter

获得最新面试
题、面经、题解



Outline

复习上一节课的内容
单序列动态规划(下)
双序列动态规划



Recursive VS DP

递归是一种程序的实现方式:函数的自我调用

```
Function(x) {  
    ...  
    Funciton(x-1);  
    ...  
}
```

动态规划是一种解决问题的思想:大规模问题的结果,是由小规模问题的结果运算得来的。

动态规划可以用递归来实现(Memorization Search)



什么情况下可能是动态规划？

满足下面三个条件之一

1. Maximum/Minimum
2. Yes/No
3. Count(*)

则“极有可能”是使用动态规划求解



什么情况下可能不是动态规划？

如果题目需要求出所有“具体”的方案而非方案“个数”

<http://www.lintcode.com/problem/palindrome-partitioning/>

输入数据是一个“集合”而不是“序列”

<http://www.lintcode.com/problem/longest-consecutive-sequence/>



动态规划的4点要素

1. 状态 State

灵感, 创造力, 存储小规模问题的结果

2. 方程 Function

状态之间的联系, 怎么通过小的状态, 来算大的状态

3. 初始化 Intialization

最极限的小状态是什么, 起点

4. 答案 Answer

最大的那个状态是什么, 终点



面试最常见的DP类型

1. Matrix DP (15%)
2. Sequence (40%)
3. Two Sequences DP (40%)
- *4. Others (5%)



1. Matrix DP

state: $f[x][y]$ 表示我从起点走到坐标 x,y

function: 研究走到 x,y 这个点之前的一步

intialize: 起点

answer: 终点



2. Sequence Dp

state: $f[i]$ 表示前 i 个位置/数字/字母,第 i 个...

function: $f[i] = f[j]$... j 是 i 之前的一个位置

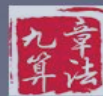
intialize: $f[0]$..

answer: $f[n-1]$..



Palindrome Partitioning II

<http://www.lintcode.com/problem/palindrome-partitioning-ii/>
<http://www.jiuzhang.com/solutions/palindrome-partitioning-ii/>



Palindrome Partitioning ii

state: $f[i]$ "前 i "个字符组成的子字符串需要最少几次cut(最少能被分割为多少个字符串-1)

function: $f[i] = \text{MIN}\{f[j]+1\}$, $j < i$ && $j+1 \sim i$ 这一段是一个回文串

intialize: $f[i] = i - 1$ ($f[0] = -1$)

answer: $f[s.length()]$



Word Break

<http://www.lintcode.com/problem/word-break/>
<http://www.jiuzhang.com/solutions/word-break/>



Word Break

state: $f[i]$ 表示“前 i ”个字符能否被完美切分

function: $f[i] = \text{OR}\{f[j]\}$, $j < i$, $j+1 \sim i$ 是一个词典中的单词

intialize: $f[0] = \text{true}$

answer: $f[s.length()]$

注意:切分位置的枚举->单词长度枚举

$O(NL)$, N : 字符串长度, L : 最长的单词的长度



10 minutes break



3. Two Sequences Dp

state: $f[i][j]$ 代表了第一个sequence的前*i*个数字/字符 配上第二个sequence的前*j*个...

function: $f[i][j]$ = 研究第*i*个和第*j*个的匹配关系

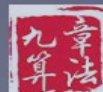
intialize: $f[i][0]$ 和 $f[0][i]$

answer: $f[s1.length()][s2.length()]$



Longest Common Subsequence

<http://www.lintcode.com/problem/longest-common-subsequence/>
<http://www.jiuzhang.com/solutions/longest-common-subsequence/>



Longest Common Subsequence

state: $f[i][j]$ 表示前 i 个字符配上前 j 个字符的 LCS 的长度

function:

$$\begin{aligned} f[i][j] &= \text{MAX}(f[i-1][j], f[i][j-1], f[i-1][j-1] + 1) \quad // \quad a[i] == b[j] \\ &= \text{MAX}(f[i-1][j], f[i][j-1]) \quad // \quad a[i] \neq b[j] \end{aligned}$$

intialize: $f[i][0] = 0$

$$f[0][j] = 0$$

answer: $f[a.length()][b.length()]$



Longest Common Substring

<http://www.lintcode.com/problem/longest-common-substring/>

<http://www.jiuzhang.com/solutions/longest-common-substring/>



Longest Common Substring

state: $f[i][j]$ 表示前 i 个字符配上前 j 个字符的 LCS 的长度
(一定以第 i 个和第 j 个结尾的 LCS)

function: $f[i][j] = f[i-1][j-1] + 1$ // $a[i] == b[j]$
 $= 0$ // $a[i] != b[j]$

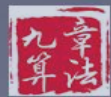
intialize: $f[i][0] = 0$
 $f[0][j] = 0$

answer: $\text{MAX}(f[0..a.\text{length()}][0..b.\text{length()}])$



Edit Distance

<http://www.lintcode.com/problem/edit-distance/>
<http://www.jiuzhang.com/solutions/edit-distance/>



Edit Distance

state: $f[i][j]$ a的前i个字符最少要用几次编辑可以变成b的前j个字符

function:

$f[i][j] = \text{MIN}(f[i-1][j]+1, f[i][j-1]+1, f[i-1][j-1])$ // $a[i] == b[j]$
 $= \text{MIN}(f[i-1][j]+1, f[i][j-1]+1, f[i-1][j-1]+1)$ // $a[i] != b[j]$

intialize: $f[i][0] = i, f[0][j] = j$

answer: $f[a.length()][b.length()]$



Distinct Subsequences

<http://www.lintcode.com/problem/distinct-subsequences/>
<http://www.jiuzhang.com/solutions/distinct-subsequences/>



Distinct Subsequences

state: $f[i][j]$ 表示 S 的前 i 个字符中选取 T 的前 j 个字符, 有多少种方案

function:

$$\begin{aligned} f[i][j] &= f[i-1][j] + f[i-1][j-1] \quad (S[i-1] == T[j-1]) \\ &= f[i-1][j] \quad (S[i-1] != T[j-1]) \end{aligned}$$

initialize: $f[i][0] = 1, f[0][j] = 0 \ (j > 0)$

answer: $f[n][m]$ ($n = \text{sizeof}(S), m = \text{sizeof}(T)$)



Interleaving String

<http://www.lintcode.com/problem/interleaving-string/>
<http://www.jiuzhang.com/solutions/interleaving-string/>



Interleaving String

state: $f[i][j]$ 表示 $s1$ 的前 i 个字符和 $s2$ 的前 j 个字符能否交替组成 $s3$ 的前 $i+j$ 个字符

function: $f[i][j] = (f[i-1][j] \ \&\& \ (s1[i-1]==s3[i+j-1])) \ ||$
 $f[i][j-1] \ \&\& \ (s2[j-1]==s3[i+j-1])$

initialize: $f[i][0] = s1[0..i-1] = s3[0..i-1]$
 $f[0][j] = s2[0..j-1] = s3[0..j-1]$

answer: $f[n][m]$
 $n = \text{sizeof}(s1), m = \text{sizeof}(s2)$



动态规划总结

什么情况下可能使用/不用动态规划？

解决动态规划问题的四点要素

状态, 方程, 初始化, 答案

三种面试常见的动态规划类别及状态特点

矩阵, 单序列, 双序列

一些奇技淫巧

初始化第一行和第一列

n 个数开 $n+1$ 个位置的数组



其他DP类别的练习 - 背包类

<http://www.lintcode.com/en/problem/backpack/>

<http://www.lintcode.com/en/problem/backpack-ii/>

<http://www.lintcode.com/en/problem/minimum-adjustment-cost/>

<http://www.lintcode.com/en/problem/k-sum/>



其他DP类别的练习 - 区间类

<http://www.lintcode.com/en/problem/coins-in-a-line-iii/>

<http://www.lintcode.com/en/problem/scramble-string/>

