

Dynamic Programming II

现在是课程答疑时间



扫描二维码关注微信/微博
获取最新面试题及权威解答

微信: [ninechapter](#)

微博: <http://www.weibo.com/ninechapter>

官网: www.jiuzhang.com

- 复习上一节课的内容
- 单序列动态规划
- 双序列动态规划

什么情况下使用动态规划？

满足下面三个条件之一：

- 求最大值最小值
- 判断是否可行
- 统计方案个数

则 **极有可能** 是使用动态规划求解

什么情况下**不使用**动态规划？

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/>

则 **极不可能** 使用动态规划求解

状态 State

灵感, 创造力, 存储小规模问题的结果

方程 Function

状态之间的联系, 怎么通过小的状态, 来算大的状态

初始化 Initialization

最极限的小状态是什么, 起点

答案 Answer

最大的那个状态是什么, 终点

坐标型动态规划 15%

序列型动态规划 30%

双序列动态规划 30%

划分型动态规划 10%

背包型动态规划 10%

区间型动态规划 5%

state:

$f[x]$ 表示我从起点走到坐标 x

$f[x][y]$ 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

单序列动态规划

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

initialize: $f[0]..$

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

Palindrome Partitioning II

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

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

Palindrome Partitioning II



九章算法

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

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

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

独孤九剑 之 破鞭式

如果不是跟坐标相关的动态规划
一般有N个数/字符, 就开N+1个位置的数组
第0个位置单独留出来作初始化

Word Break

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

<http://www.jiuzhang.com/solutions/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

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

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

注意: 切分位置的枚举 \rightarrow 单词长度枚举 $O(NL^2)$

- N : 字符串长度
- L : 最长的单词的长度

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

initialize: $f[i][0]$ 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



九章算法

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

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

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

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

Related Question:

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

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

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

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

Distinct Subsequence

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

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

Distinct Subsequence

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: $f[i][j] = f[i-1][j] + f[i-1][j-1]$ // $S[i-1] == T[j-1]$
 $= f[i-1][j] (S[i-1] != T[j-1])$

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 s_1 的前 i 个字符和 s_2 的前 j 个字符能否交替组成 s_3 的前 $i+j$ 个字符

function: $f[i][j] = (f[i-1][j] \ \&\& \ (s_1[i-1] == s_3[i+j-1])) \ ||$
 $(f[i][j-1] \ \&\& \ (s_2[j-1] == s_3[i+j-1]))$

initialize: $f[i][0] = (s_1[0..i-1] == s_3[0..i-1])$

$f[0][j] = (s_2[0..j-1] == s_3[0..j-1])$

answer: $f[n][m]$, $n = \text{sizeof}(s_1)$, $m = \text{sizeof}(s_2)$

什么情况下可能使用/不用动态规划？

- 最大值最小值/是否可行/方案总数
- 求所有方案/集合而不是序列

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

- 坐标, 单序列, 双序列

一些奇技淫巧

- 初始化第0行和第0列
- n 个数开 $n+1$ 个位置的数组

背包类:

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

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

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

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

区间类:

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

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

划分类：

<http://www.lintcode.com/problem/best-time-to-buy-and-sell-stock-iv/>

<http://www.lintcode.com/problem/maximum-subarray-iii/>