课程尚未开始
请大家耐心等待

关注微信公共账号，获得最新面试题信息及解答

Facebook: http://www.facebook.com/ninechapter
Weibo: http://www.weibo.com/ninechapter
Renren: http://page.renren.com/601712402

Dynamic Programming

九章算法IT求职面试培训课程 第5章

www.ninechapter.com

Number Triangle

http://www.lintcode.com/zh-cn/problem/number-triangle/

http://www.ninechapter.com/solutions/triangle/

Triangle

```
// dfs(x,y) 从x,y走到最下面一层的最短距离
// answer: dfs(0,0)
int dfs(int x, int y) {
	if (x == n) {
		return 0;
	}
	if (hash[x][y] != -1) {
		return hash[x][y];
	}
	hash[x][y] = Min(dfs(x + 1, y), dfs(x + 1, y + 1)) + a[x][y];
	return hash[x][y];
}
```

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)

如何想到使用DP

1. One of the following three :
    a) Maximum/Minimum
    b) Yes/No
    c) Count all possbile solutions

2. Can not sort / swap
    https://oj.leetcode.com/problems/longest-consecutive-sequence/

面试最常见的四种类型

1. Matrix DP (10%)
2. Sequence (40%)
3. Two Sequences DP (40%)
4. Backpack (10%)

1. Matrix DP

state: f[x][y] 表示我从起点走到坐标x,y……

function: 研究最后一步怎么走

intialize: 起点

answer: 终点

Unique Paths

http://www.lintcode.com/zh-cn/problem/unique-paths/
http://www.ninechapter.com/solutions/unique-paths/

Unique Paths

state: f[x][y]从起点到x,y的路径数

function: (研究倒数第一步)

    f[x][y] = f[x - 1][y] + f[x][y - 1]

intialize: f[0][0] = 1

                // f[0][i] = 1, f[i][0] = 1

answer: f[n-1][m-1]

Unique Paths II

http://www.lintcode.com/zh-cn/problem/unique-paths-ii/
http://www.ninechapter.com/solutions/unique-paths-ii/

Minimum Path Sum

http://www.lintcode.com/zh-cn/problem/minimum-path-sum/

http://www.ninechapter.com/solutions/minimum-path-sum/

Minimum Path Sum

state: f[x][y]从起点走到x,y的最短路径

function: f[x][y] = min(f[x-1][y], f[x][y-1]) + cost[x][y]

intialize: f[0][0] = cost[0][0]

// f[i][0] = sum(0,0 -> i,0)

// f[0][i] = sum(0,0 -> 0,i)

answer: f[n-1][m-1]

5 minutes break

2. Sequence Dp

state: f[i]表示“**前i**”个位置/数字/字母,(以第i个为)...

function: f[i] = f[j] … j 是i之前的一个位置

intialize: f[0]..

answer: f[n-1]..

Climbing Stairs

http://www.lintcode.com/zh-cn/problem/climbing-stairs/
http://www.ninechapter.com/solutions/climbing-stairs/

Climbing Stairs

state: f[i]表示前i个位置，跳到第i个位置的方案总数

function: f[i] = f[i-1] + f[i-2]

intialize: f[0] = 1

answer: f[n]

Jump Game

http://www.lintcode.com/zh-cn/problem/jump-game/

http://www.ninechapter.com/solutions/jump-game/

Jump game

state: f[i]代表我能否跳到第i个位置
function: f[i] = OR(f[j], j < i && j能够跳到i)
initialize: f[0] = true;
answer: f[n-1]

Jump Game II

http://www.lintcode.com/zh-cn/problem/jump-game-ii/
http://www.ninechapter.com/solutions/jump-game-ii/

Jump game II

state: f[i]代表我跳到这个位置最少需要几步

function: f[i] = MIN(f[j]+1, j < i && j能够跳到i)

initialize: f[0] = 0;

answer: f[n-1]

Palindrome Partitioning II

http://www.lintcode.com/zh-cn/problem/palindrome-partitioning-ii/

http://www.ninechapter.com/solutions/palindrome-partitioning-ii/

Palindrome Partitioning ii

state: f[i]”前i”个字符组成的字符串需要最少几次cut

function: f[i] = MIN(f[j]+1, j < i && j+1 ~ i这一段是一个回文串)

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

answer: f[s.length()]

Word Segmentation

http://www.lintcode.com/zh-cn/problem/word-segmentation/
http://www.ninechapter.com/solutions/word-break/

Word Segmentation

state: f[i]表示前i个字符能否被完美切分

function: f[i] = OR(f[j], j < i, j+1 ~ i是一个词典中的单词)

intialize: f[0] = 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

Longest Increasing Subsequence

http://www.lintcode.com/zh-cn/problem/longest-increasing-subsequence/
http://www.ninechapter.com/solutions/longest-increasing-subsequence/

Longest Increasing Subsequence

state:

错误的方法: f[i]表示前i个数字中最长的LIS的长度

正确的方法: f[i]表示前i个数字中**以第i个结尾的LIS的长度**

function: f[i] = MAX(f[j]+1, j < i && a[j] <= a[i])

intialize: f[0..n-1] = 1

answer: max(f[0..n-1])

LIS 贪心反例

1 1000 2 3 4

10 11 12 1 2 3 4 13

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/en/problem/longest-common-subsequence/
http://www.ninechapter.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: f[i][j] = f[i-1][j-1] + 1 // a[i] == b[j]

= 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()]

Longest Common Substring

http://www.lintcode.com/en/problem/longest-common-substring/
http://www.ninechapter.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: MAX(f[0..a.length()][0..b.length()])

Edit Distance

http://www.lintcode.com/en/problem/edit-distance/
http://www.ninechapter.com/solutions/edit-distance/

Edit Distance

state: f[i][j]a的前i个字符配上b的前j个字符最少要用几次编辑使得他们相等

function:

f[i][j] = MIN(f[i-1][j-1], f[i-1][j]+1, f[i][j-1]+1) // a[i] == b[j]

= MIN(f[i-1][j], f[i][j-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()]

其他题目

Distinct Subsequence

Interleaving String

4. Backpack

题目：给n个正整数，一个数target，问能否从n个数中取出***若干***个数，他们的和为target。

state: f[i][S] “前i”个数字，取出一些能否组成和为S

function: f[i][S] = f[i-1][S - a[i]] or f[i-1][S]

intialize: f[0][0] = true; f[0][1..SUM] = false

answer: f[n][target]

k Sum

n个数，取k个数，组成和为target

state: f[i][j][t]前个数取j个数出来能否和为t

function: f[i][j][t] = f[i - 1][j - 1][t - a[i]] or

f[i - 1][j][t]

1. 问是否可行 (DP)
2. 问方案总数 (DP)
3. 问所有方案 (递归/搜索)

最小调整代价

n个数，可以对每个数字进行调整，使得相邻的两个数的差都<=target，调整的费用为

Sigma(|A[i]-B[i]|)

A[i]原来的序列 B[i]是调整后的序列

A[i] < 200, target < 200

让代价最小

B[woB[]B[]B[B

最小调整代价

state: f[i][v] 前i个数，第i个数调整为v，满足相邻两数<=target，所需要的最小代价

function: f[i][v] = min(f[i-1][v’] + |A[i]-v|, |v-v’| <= target)

answer: f[n][…]

O(n * A * T)

Conclusion

4 key points of DP:

1. State
2. Function
3. Initialize
4. Answer

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)