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

Outline

从递归到动划 - Triangle
什么样的题适合使用动态规划？
当我们谈论动态规划的时候，我们在谈论什么？
面试中常见动态规划的分类
矩阵上的动态规划
序列上的动态规划

Triangle

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

DFS

```
// traverse
void dfs(int x, int y, int sum) {
    if (x == n) {
        if (sum < best) {
            best = sum;
        }
        return;
    }

    dfs(x + 1, y, sum + a[x][y]);
    dfs(x + 1, y + 1, sum + a[x][y]);
}

dfs(0, 0);
```

```
// Divide & Conquer
// 从x,y出发走到最底层所能找到的最小路径和
int dfs(int x, int y) {
    if (x == n) {
        return 0;
    }
    return min(dfs(x + 1, y), dfs(x + 1, y + 1)) + a[x][y];
}

dfs(0, 0);
```

DFS Optimization

```
// Divide & Conquer (optimization)
// 从x,y出发走到最底层所能找到的最小路径和
int dfs(int x, int y) {
    if (x == n) {
        return 0;
    }
    // = -1 表示还没有被计算出来
    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];
}
```

记忆化搜索

本质上:动态规划

动态规划就是解决了重复计算的搜索

动态规划的实现方式:

1. 记忆化搜索
2. 循环

自底向上的动态规划

```
A[][]

// 状态定义
f[i][j] 表示从i,j出发走到最后一层的最小路径长度

// 初始化，终点先有值
for (int i = 0; i < n; i++) {
   f[n-1][i] = A[n-1][i];
}

// 循环递推求解
for (int i = n - 2; i >= 0; i--) {
   for (int j = 0; j <= i; j++) {
       f[i][j] = Math.min(f[i + 1][j], f[i + 1][j + 1]) + A[i][j];
   }
}

// 求结果：起点
f[0][0]
```

自顶向下的动态规划

```
// 自顶向下的动态规划
状态定义:
f[i][j] 表示从0,0出发，到达i,j的最短路径是什么

// 初始化
f[0][0] = A[0][0]

// 递推求解
for (int i = 1; i < n; i++) {
    for (int j = 1; j <= i; j++) {
        // f[i][j] = min(f[i - 1][j], f[i - 1][j - 1]) + A[i][j];
        f[i][j] = oo;
        if (i - 1, j存在) {
            f[i][j] = min(f[i][j], f[i - 1][j]);
        }
        if (i - 1, j - 1存在) {
          . ..
        }
        f[i][j] += A[i][j];
    }
}

// 答案
min(f[n-1][0], f[n-1][1], f[n-1][2] ... )
```

如何想到使用DP

1. One of the following three
   a) Maximum/Minimum
   b) Yes/No
   c) Count(*)

2. Can not sort / swap
   http://www.lintcode.com/en/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

面试最常见的四种类型

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

5 minutes break

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

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]) + A[x][y]

intialize: f[0][0] = A[0][0]

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

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

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

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/

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]代表我能否从起点跳到第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] = 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