

# 动态规划

# Dynamic Programming

课程版本 4.0      主讲 令狐冲



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>

知乎: <http://zhuanglan.zhihu.com/jiuzhang>

官网: <http://www.jiuzhang.com>

- 通过一道经典题理解动态规划
  - 递归与动规的联系与区别
  - 记忆化搜索
- 什么时候使用动态规划
  - 适用动态规划的三个条件
  - 不适用动态规划三个条件
- 动规四要素
  - vs 递归三要素
- 面试中常见动态规划的分类
- 坐标(矩阵)动态规划
- 接龙型动态规划

# Triangle

- <http://www.lintcode.com/problem/triangle/>
- <http://www.jiuzhang.com/solutions/triangle/>
- 解决方法:
- DFS: Traverse
- DFS: Divide Conquer
- Divide Conquer + Memorization
- Traditional Dynamic Programming

- 时间复杂度？
- A -  $O(n^2)$
- B -  $O(2^n)$
- C -  $O(n!)$
- D - I don't know

```
1 void traverse(int x, int y, int sum) {  
2     if (x == n) {  
3         // found a whole path from top to bottom  
4         if (sum < best) {  
5             best = sum;  
6         }  
7         return;  
8     }  
9  
10    traverse(x + 1, y, sum + A[x][y]);  
11    traverse(x + 1, y + 1, sum + A[x][y]);  
12 }  
13  
14 best = MAXINT;  
15 traverse(0, 0, 0);  
16 // best is the answer
```

# DFS: Divide Conquer

- 时间复杂度？
- A -  $O(n^2)$
- B -  $O(2^n)$
- C -  $O(n!)$
- D - I don't know

```
1 // return minimum path from (x, y) to bottom
2 int divideConquer(int x, int y) {
3     if (x == n) {
4         return 0;
5     }
6     return A[x][y] + Math.min(
7         divideConquer(x + 1, y),
8         divideConquer(x + 1, y + 1)
9     );
10 }
11
12 divideConquer(0, 0);
```

# DFS: Divide Conquer + Memorization

- 时间复杂度？
- A -  $O(n^2)$
- B -  $O(2^n)$
- C -  $O(n!)$
- D - I don't know

```
1 // return minimum path from (x, y) to bottom
2 int divideConquer(int x, int y) {
3     // row index from 0 to n-1.
4     if (x == n) {
5         return 0;
6     }
7
8     // if we already got the minimum path from (x, y) to bottom.
9     // just return it
10    if (hash[x][y] != Integer.MAX_VALUE) {
11        return hash[x][y]
12    }
13
14    // set before return
15    hash[x][y] = A[x][y] + Math.min(divideConquer(x + 1, y),
16                                     divideConquer(x + 1, y + 1));
17
18    return hash[x][y];
19 }
20
21 initialize: hash[*][*] = Integer.MAX_VALUE;
22 answer: divideConquer(0, 0);
```

# 记忆化搜索的本质：动态规划

动态规划为什么会快？

动态规划与分治的区别？

重复计算！

# 多重循环 vs 记忆化搜索

优点: 正规, 大多数面试官可以接受, 存在空间优化可能性。

缺点: 思考有难度。

优点: 容易从搜索算法直接转化过来。有的时候可以节省更多的时间。

缺点: 递归。



- 时间复杂度？

`A[][]`

- 空间复杂度？

// 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]`

- 时间复杂度？
- 空间复杂度？

```
1 // 初始化，起点
2 f[0][0] = A[0][0];
3
4 // 初始化三角形的左边和右边
5 for (int i = 1; i < n; i++) {
6     f[i][0] = f[i - 1][0] + A[i][0];
7     f[i][i] = f[i - 1][i - 1] + A[i][i];
8 }
9
10 // top down
11 for (int i = 1; i < n; i++) {
12     for (int j = 1; j < i; j++) {
13         f[i][j] = Math.min(f[i - 1][j], f[i - 1][j - 1]) + A[i][j];
14     }
15 }
16
17 Math.min(f[n - 1][0], f[n - 1][1], f[n - 1][2] ...);
```

# 自底向上 vs 自顶向下

两种方法没有太大优劣区别

思维模式一个正向，一个逆向

为了方便教学，后面我们统一采用 **自顶向下** 的方式

## 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/>
- **暴力**算法的复杂度已经是**多项式**级别
  - 动态规划擅长与优化指数级别复杂度( $2^n, n!$ )到多项式级别复杂度( $n^2, n^3$ )
  - 不擅长优化 $n^3$ 到 $n^2$
- 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

## 递归三要素:

- 定义(状态)
  - 接受什么参数
  - 做了什么事
  - 返回什么值
- 拆解(方程)
  - 如何将参数变小
- 出口(初始化)
  - 什么时候可以直接 return

## 面试中常见的动态规划类型

- 坐标型动态规划 10%
- 接龙型动态规划 20%
- 划分型动态规划 20% (算法强化班)
- 双序列动态规划 20% (算法强化班)
- 背包型动态规划 20% (算法强化班)
- 区间型动态规划 10% (算法强化班)

# Take a break

5 minutes



## 坐标型动态规划

---

- 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
- initialize: 起点
- answer: 终点

# Minimum Path Sum

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

<http://www.jiuzhang.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:  $f[x][y] = \min(f[x-1][y], f[x][y-1]) + A[x][y]$
- initialize:  $f[i][0] = \text{sum}(0, 0 \sim i, 0)$
- $f[0][i] = \text{sum}(0, 0 \sim 0, i)$
- answer:  $f[n-1][m-1]$

# 独孤九剑 —— 破气式

初始化一个二维的动态规划时  
就去初始化**第0行**和**第0列**

# Unique Path

<http://www.lintcode.com/problem/unique-paths/>

<http://www.jiuzhang.com/solutions/unique-paths/>

# Unique Path

- state:  $f[x][y]$  从起点到  $x, y$  的路径数
- function: (研究倒数第一步)  $f[x][y] = f[x - 1][y] + f[x][y - 1]$
- initialize:  $f[0][i] = 1$
- $f[i][0] = 1$
- answer:  $f[n-1][m-1]$
  
- Related Question: Unique Path II

# Climbing Stairs

<http://www.lintcode.com/problem/climbing-stairs/>

<http://www.jiuzhang.com/solutions/climbing-stairs/>

# Climbing Stairs

---

- state:  $f[i]$  表示跳到第  $i$  个位置的方案总数
- function:  $f[i] = f[i-1] + f[i-2]$
- initialize:  $f[0] = 1$
- answer:  $f[n]$  // index from  $0 \sim n$



跳跃游戏 I && II

<http://www.lintcode.com/en/problem/jump-game/>

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

这个题最优的方法是使用“贪心法”，动态规划复杂度较高

我们已与LintCode协商，将数据范围调低，让你也可以通过动态规划算法拿到Accepted。

# 接龙型动态规划

属于“坐标型”动态规划的一种

# Longest Increasing Subsequence

<http://www.lintcode.com/problem/longest-increasing-subsequence/>

<http://www.jiuzhang.com/solutions/longest-increasing-subsequence/>

# Longest Increasing Subsequence

- 将 $n$ 个数看做 $n$ 个木桩, 目的是从某个木桩出发, 从前向后, 从低往高, 看做多能踩多少个木桩。
- state:  $f[i]$  表示(从任意某个木桩)跳到第 $i$ 个木桩, 最多踩过多少根木桩
- function:  $f[i] = \max\{f[j] + 1\}$ ,  $j$ 必须满足  $j < i \ \&\& \ \text{nums}[j] < \text{nums}[i]$
- initialize:  $f[0..n-1] = 1$
- answer:  $\max\{f[0..n-1]\}$

# Perfect Squares

<http://www.lintcode.com/en/problem/perfect-squares/>

<http://www.jiuzhang.com/solutions/perfect-squares/>

# Largest Divisible Subset

<http://www.lintcode.com/en/problem/largest-divisible-subset/>

<http://www.jiuzhang.com/solutions/largest-divisible-subset/>

# 动态规划只能记录一种最优的方案

动规无法记录所有的最优方案  
为什么？

俄罗斯套娃

<http://www.lintcode.com/en/problem/russian-doll-envelopes/>

青蛙过河

<http://www.lintcode.com/en/problem/frog-jump/>



## 动态规划 总结

---

- 动态规划的实质
  - 记忆化搜索
  - 避免重复的中间结果计算
- 动态规划与递归的关系
  - 动规四要素 vs 递归三要素
- 什么时候使用动态规划
  - 最优, 可行, 方案数
- 什么时候不用动态规划
  - 所有方案而不是方案数
  - 集合而非序列
  - 暴力算法已经是多项式级别复杂度
    - 动态规划擅长优化指数级别( $2^n$ )到多项式级别( $n^2$ )

## 动态规划 总结

---

- 动态规划四要素
  - 状态, 方程, 初始化, 答案
- 动态规划经典题
  - Longest Increasing Subsequence (LIS)

# 点题时间

<http://www.jiuzhang.com/qa/1034/>

# 老师领进门，修行靠自身

刷多少题够？一个题要刷几遍？  
代码风格和编程速度哪个重要？

<http://www.jiuzhang.com/qa/3>

# 多给予你周围的人帮助

帮助别人也是在提高自己

欢迎来九章办讲座，分享你的求职经验

联系 [info@jiuzhang.com](mailto:info@jiuzhang.com)

# 还想继续上课？

有很多基础知识需要补——Java入门与基础算法  
九章算法班的内容还比较吃力——在第26期可半价重听  
九章算法班的内容可以消化——九章算法强化班  
算法知识已经学够了——系统设计班  
通过项目实战提高硬实力——Android / Big Data 项目实战班  
老学员上其他课程均直接享受团购价优惠

# 推荐奖励 Referral

对方可以团购价基础上获得10\$的优惠  
你也可以同时获得10\$的优惠券

祝大家找到好工作！  
有缘再见！