让面试官眼前一亮的算法
——记忆化搜索

主讲人 令狐冲
课程版本 v7.0

什么是记忆化搜索

在函数返回前，记录函数的返回结果

在下一次以**同样参数**访问函数时直接返回记录下的结果

记忆化搜索函数的三个特点

函数有返回值

函数返回结果之和输入参数相关，和其他全局状态无关

参数列表中传入哈希表或者其他用于记录计算结果的数据结构

记忆化搜索 vs 动态规划

记忆化搜索是动态规划的**一种实现方式**

动态规划的另外一种实现方式是多重循环（下节课）

所以记忆化搜索**就是**动态规划

动态规划的核心思想：由大化小

动态规划的算法思想：大规模问题的依赖于小规模问题的计算结果

类似思想算法的还有：递归，分治法

独孤九剑 —— 破箭式

三种适用动态规划的场景

三种不适用动态规划的场景

三种适用DP的场景

求最优值

求方案数

求可行性

三种适用动规的场景

- 求最值
  - dp[] 的值的类型是最优值的类型
  - dp[大问题] = max{dp[小问题1], dp[小问题2], ...}
  - dp[大问题] = min{dp[小问题1], dp[小问题2], ...}
- 求方案数
  - dp[] 的值的类型是方案数（整数）
  - dp[大问题] = ∑(dp[小问题1], dp[小问题2], ...)
  - ∑=sum
- 求可行性
  - dp[] 的值是 true / false
  - dp[大问题] = dp[小问题1] or dp[小问题2] or ...
  - 代码通常用 for 小问题 if dp[小问题] == true then break 的形式实现

三种不适用DP的场景

求所有的具体方案

输入数据是无序的

暴力算法时间复杂度已经是多项式级别

三种不适用 DP 的场景

- 求出所有的具体方案
  - http://www.lintcode.com/problem/palindrome-partitioning/
  - 只求出一个具体方案还是可以用 DP 来做的（下节课）
  - 该判断标准成功率 99%
- 输入数据是无序的
  - http://www.lintcode.com/problem/longest-consecutive-sequence/
  - 背包类动态规划不适用此判断条件
  - 除去背包问题后，该判断标准成功率 60-70%，有一些题可以先排序之后按序处理
- 暴力算法的复杂度已经是多项式级别
  - http://www.lintcode.com/problem/largest-rectangle-in-histogram/
  - 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
  - 该判断标准成功率 80%
- 则 **极不可能** 使用动态规划求解

Wildcard Matching

http://www.lintcode.com/problem/wildcard-matching/

http://www.jiuzhang.com/solution/wildcard-matching/

类别：匹配型动态规划

适用场景：求可行性

Follow up:
Regular Expression Matching

http://www.lintcode.com/problem/regular-expression-matching/

http://www.jiuzhang.com/solution/regular-expression-matching/

面试是一定不会让你做完整版的 Regular Expression 的

所以一定是阉割版的

面试评分标准

Strong Hire: 两个都答出来，且写出来，Bug Free or Bug 很少

Hire / Weak Hire: 两个都答出来，写完第一个，第二个能基本在第一个的基础上改完，允许有一些提示和少量 Bug

No Hire: 没写完，或者需要很多提示

Strong No: 第一个都没写完

休息 5 分钟

Take a break

Word Pattern II

http://www.lintcode.com/problem/word-pattern-ii/

http://www.jiuzhang.com/solutions/word-pattern-ii/

这个题是否可以记忆化?

Word Break

右边的代码正确性没有问题

但是存在一个问题导致其无法通过测试

这个问题是什么?

```python
    def is_possible(self, s, index, max_length, dict, memo):
        if index in memo:
            return memo[index]

        if index == len(s):
            return True

        memo[index] = False
        for i in range(index, len(s)):
            if i + 1 - index > max_length:
                break
            word = s[index: i + 1]
            if word not in dict:
                continue
            if self.is_possible(s, i + 1, max_length, dict, memo):
                memo[index] = True
                break

        return memo[index]

    def get_max_length(self, dict):
        max_length = 0
        for word in dict:
            max_length = max(max_length, len(word))
        return max_length
```

记忆化搜索的缺陷

递归深度太深，导致 StackOverflow

Word Break II

http://www.lintcode.com/problem/word-break-ii/

http://www.jiuzhang.com/solution/word-break-ii/

不适用场景：求出所有具体方案而非方案总数

但是可以使用动态规划进行优化

优化方案1

用 Word Break 这个题的思路

使用 is_possible[i] 代表从 i 开始的后缀是否能够被 break

在 DFS 找所有方案的时候，通过 is_possible 可以进行**可行性剪枝**

优化方案 2

直接使用 memo[i] 记录从位置 i 开始的后缀
能够被 break 出来的所有方案

极端情况

以上两种方法在极端情况下是否能有优化效果呢?

s = “aaaaaaaaaa...”

dict = {“a”, “aa”, “aaa”, ...}

Word Break II 的面试评分标准

Strong Hire: DFS+DP优化

Hire / Weak Hire: DFS 能写完，且 Bug free or Bug 不多，不需要提示 or 需要少量提示

No Hire: DFS 写不完，或者需要很多提示

Strong No: 啥都想不出

* Palindrome Partitioning

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

http://www.jiuzhang.com/solutions/palindrome-partitioning/

一个类似 Word Break II 的题

但是使用记忆化搜索优化效果甚微

Word Break III

https://www.lintcode.com/problem/word-break-iii

https://www.jiuzhang.com/solution/word-break-iii

类别：前缀型/划分型动态规划

适用场景：求方案总数