王争的算法训练营

习题课：爬楼梯问题

王争的算法训练营

动态规划

王争的算法训练营

重中之重，大厂笔试、面试必考项、学习难但面试不难、掌握模型举一反三

DP专题：

- 专题：适用问题
- 专题：解题步骤
- 专题：最值、可行、计数三种类型　一些特殊小类别：树形DP、区间DP、数位DP
- 专题：空间优化

经典模型：

- 背包问题（0-1、完全、多重、二维费用、分组、有依赖的）
- 路径问题
- 打家劫舍&股票买卖
- 爬楼梯问题
- 匹配问题（LCS、编辑距离）
- 其他（LIS）

王争的算法训练营

动态规划

王争的算法训练营

配套习题（24）：

背包：

416. 分割等和子集
494. 目标和
322. 零钱兑换
518. 零钱兑换 II

路径问题

64. 最小路径和
剑指 Offer 47. 礼物的最大价值
120. 三角形最小路径和
62. 不同路径
63. 不同路径 II

打家劫舍 & 买卖股票：

198. 打家劫舍
213. 打家劫舍 II
337. 打家劫舍 III (树形DP)
714. 买卖股票的最佳时机含手续
309. 最佳买卖股票时机含冷冻期

爬楼梯问题

70. 爬楼梯
322. 零钱兑换
518. 零钱兑换 II
剑指 Offer 14- I. 剪绳子
剑指 Offer 46. 把数字翻译成字符串
139. 单词拆分

匹配问题

1143. 最长公共子序列
72. 编辑距离

其他

437. 路径总和 III (树形DP)
300. 最长递增子序列

王争的算法训练营

习题课：动态规划

爬楼梯问题

70. 爬楼梯

322. 零钱兑换

518. 零钱兑换 II

剑指 Offer 14- I. 剪绳子

剑指 Offer 46. 把数字翻译成字符串

139. 单词拆分

王争的算法训练营

70. 爬楼梯

假设你正在爬楼梯。需要 *n* 阶你才能到达楼顶。

每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢?

注意：给定 *n* 是一个正整数。

示例 1：

```
输入： 2
输出： 2
解释： 有两种方法可以爬到楼顶。
1.  1 阶 + 1 阶
2.  2 阶
```

之间，讲过递归的解法，现在，构建多阶段决策模型，通过DP解法

int dp[n+1] dp[i]表示走完i个台阶有多少种走法。

到达i这个状态，那上一步只有可能是走1、2个台阶，也就是从状态i-1,i-2转化过来。dp[i]值也由 dp[i-1]、dp[i-2]推导出来。

dp[i] = dp[i-1] + dp[i-2]

示例 2：

```
输入： 3
输出： 3
解释： 有三种方法可以爬到楼顶。
1.  1 阶 + 1 阶 + 1 阶
2.  1 阶 + 2 阶
3.  2 阶 + 1 阶
```

70. 爬楼梯

```
class Solution {
    public int climbStairs(int n) {
      if (n <= 2) return n;

      int[] dp = new int[n+1];
      dp[1] = 1;
      dp[2] = 2;
      for (int i = 3; i <= n; ++i) {
        dp[i] = dp[i-1] + dp[i-2];
      }
      return dp[n];
    }
}
```

```
class Solution {
    public int climbStairs(int n) {
      int[] dp = new int[n+1];
      dp[0] = 1;
      for (int i = 1; i <= n; ++i) {
        if (i-1>=0) dp[i] += dp[i-1];
        if (i-2>=0) dp[i] += dp[i-2];
      }
      return dp[n];
    }
}
```

之间，讲过递归的解法。现在，构建多阶段决策模型，通过DP解法

int dp[n+1] dp[i]表示走完i个台阶有多少种走法。

到达i这个状态，那上一步只有可能是走1、2个台阶，也就是从状态i-1，i-2转化过来。dp[i]值也由 dp[i-1]、dp[i-2]推导出来。

dp[i] = dp[i-1] + dp[i-2]

动态规划

322. 零钱兑换

给你一个整数数组 `coins` ，表示不同面额的硬币；以及一个整数 `amount` ，表示总金额。

计算并返回可以凑成总金额所需的 **最少的硬币个数** 。如果没有任何一种硬币组合能组成总金额，返回 `-1` 。

你可以认为每种硬币的数量是无限的。

每个阶段从1、2、5种选择一个硬币

int dp[amount+1] dp[i]表示凑够i元最少需要多少硬币。

到达i这个状态，那上一步只有可能是选了1、2、5，也就是从状态i-1, i-2、i-5转化过来。dp[i]值也由 dp[i-1]、dp[i-2]、dp[i-5]推导出来。

dp[i] = min(dp[i-1] , dp[i-2], dp[i-5])+1

示例 1：

```
输入：coins = [1, 2, 5], amount = 11
输出：3
解释：11 = 5 + 5 + 1
```

示例 2：

```
输入：coins = [2], amount = 3
输出：-1
```

322. 零钱兑换

```
class Solution {
    public int coinChange(int[] coins, int amount) {
        int k = coins.length;
        int[] dp = new int[amount + 1];
        for (int i = 0;i <= amount; ++i) {
            dp[i] = Integer.MAX_VALUE;
        }
        dp[0] = 0;
        for (int i = 1; i <= amount; ++i) {
            for (int j = 0; j < k; ++j) {
                if (i-coins[j]>=0 &&
                    dp[i-coins[j]] != Integer.MAX_VALUE &&
                    dp[i-coins[j]]+1 < dp[i]) {
                    dp[i] = dp[i-coins[j]] + 1;
                }
            }
        }
        if (dp[amount] == Integer.MAX_VALUE) return -1;
        return dp[amount];
    }
}
```

每个阶段从1、2、5种选择一个硬币

int dp[amount+1] dp[i]表示凑够i元最少需要多少硬币。

到达i这个状态，那上一步只有可能是选了1、2、5，也就是从状态i-1, i-2、i-5转化过来。dp[i]值也由 dp[i-1]、dp[i-2]、dp[i-5]推导出来。

dp[i] = min(dp[i-1] , dp[i-2], dp[i-5])+1

动态规划

518. 零钱兑换 II

给你一个整数数组 `coins` 表示不同面额的硬币，另给一个整数 `amount` 表示总金额。

请你计算并返回可以凑成总金额的硬币组合数。如果任何硬币组合都无法凑出总金额，返回 `0` 。

假设每一种面额的硬币有无限个。

题目数据保证结果符合 32 位带符号整数。

每个阶段从1、2、5种选择一个硬币

int dp[amount+1] dp[i]表示凑够i元有多少种方法。

到达i这个状态，那上一步只有可能是选了1、2、5，也就是从状态i-1, i-2、i-5转化过来。dp[i]值也由 dp[i-1]、dp[i-2]、dp[i-5]推导出来。

dp[i] = dp[i-1] + dp[i-2] + dp[i-5]

示例 1：

输入： amount = 5, coins = [1, 2, 5]
输出： 4
解释： 有四种方式可以凑成总金额：
5=5
5=2+2+1
5=2+1+1+1
5=1+1+1+1+1

王争的算

习题课：动态规划

剑指 Offer 14- I. 剪绳子

给你一根长度为 `n` 的绳子，请把绳子剪成整数长度的 `m` 段（m、n都是整数，n>1并且m>1），每段绳子的长度记为 `k[0],k[1]...k[m-1]` 。请问 `k[0]*k[1]*...*k[m-1]` 可能的最大乘积是多少？例如，当绳子的长度是8时，我们把它剪成长度分别为2、3、3的三段，此时得到的最大乘积是18。

示例 1：

```
输入：2
输出：1
解释：2 = 1 + 1，1 × 1 = 1
```

示例 2:

```
输入：10
输出：36
解释：10 = 3 + 3 + 4，3 × 3 × 4 = 36
```

提示：

- `2 <= n <= 58`

每个阶段从绳子中割出1、2、…、k长度的一段。

int dp[n+1] dp[i]表示长度为i的绳子切割之后的最大乘积。

到达i这个状态，那上一步只有可能是割了1、2、…、i长度一段，也就是从状态i-1, i-2、i-3、…、0转化过来。dp[i]值也由 dp[i-1]、dp[i-2]、dp[i-3]…dp[0]推导出来。

dp[i] = max（1*dp[i-1]，2*dp[i-2]， 3*dp[i-3]， … ， i * dp[0]）

习题课：动态规划

[剑指 Offer 14- I. 剪绳子]

```
class Solution {
    // 递归转非递归的写法，类比上台阶
    public int cuttingRope(int n) {
        if (n == 1) return 1;
        if (n == 2) return 1;
        if (n == 3) return 2;
        // dp[i]表示长度为i的最大乘积
        int[] dp = new int[n+1];
        dp[0] = 1;
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= i; j++) {
                if (dp[i] < j*dp[i-j]) {
                    dp[i] = j*dp[i-j];
                }
            }
        }
        return dp[n];
    }
}
```

每个阶段从绳子中割出1、2、...、k长度的一段。

int dp[n+1] dp[i]表示长度为i的绳子切割之后的最大乘积。

到达i这个状态，那上一步只有可能是割了1、2、...、i长度一段，也就是从状态i-1, i-2、i-3、...、0转化过来。dp[i]值也由 dp[i-1]、dp[i-2]、dp[i-3]...dp[0]推导出来。

dp[i] = max（1*dp[i-1] ，2*dp[i-2] ， 3*dp[i-3] ， ... ， i * dp[0]）

习题课：动态规划

剑指 Offer 46. 把数字翻译成字符串

给定一个数字，我们按照如下规则把它翻译为字符串：0 翻译成 “a”，1 翻译成 “b”，……，11 翻译成 “l”，……，25 翻译成 “z”。一个数字可能有多个翻译。请编程实现一个函数，用来计算一个数字有多少种不同的翻译方法。

示例 1：

```
输入：12258
输出：5
解释：12258有5种不同的翻译，分别是"bccfi", "bwfi", "bczi", "mcfi"和"mzi"
```

提示：

- $0 <= num < 2^{31}$

每个阶段从1个或者和2个数字翻译

int dp[n+1] （n表示数字个数）dp[i]表示长度为i的数字序列有多少种翻译方法。

到达i这个状态，那上一步只有可能是选了1个或2个数字翻译，也就是从状态i-1，i-2转化过来。dp[i]值也由 dp[i-1]、dp[i-2]推导出来。

dp[i] = dp[i-1] + dp[i-2] (前提是2个数字不超过25)

```java
class Solution {
    public int translateNum(int num) {
        if (num <= 9) return 1;
        // 把十进制数转化成数字数组
        List<Integer> digitlist = new ArrayList<>();
        while (num != 0) {
            digitlist.add(num%10);
            num /= 10;
        }
        int n = digitlist.size();
        int[] digits = new int[n];
        for (int i = 0; i < n; ++i) {
            digits[i] = digitlist.get(n-i-1);
        }

        int[] dp = new int[n+1];
        dp[0] = 1;
        // dp[i]表示digits[0~i-1](长度为i)转化为字母有多少种方法
        // dp[i] = dp[i-1] + dp[i-2](digits[i-2, i-1]可翻译)
        // dp[i] = dp[i-1] (digits[i-2, i-1]不可翻译)
        for (int i = 1; i <= n; ++i) { // 爬楼梯
            dp[i] = dp[i-1];
            if (i-2>=0 && isValid(digits[i-2], digits[i-1])) {
                dp[i] += dp[i-2];
            }
        }
        return dp[n];
    }

    private boolean isValid(int a, int b) {
        if (a == 1) return true;
        if (a == 2 && b >= 0 && b <= 5) return true;
        return false;
    }
}
```

每个阶段从1个或者和2个数字翻译

int dp[n+1] （n表示数字个数）dp[i]表示长度为i的数字序列有多少种翻译方法。

到达i这个状态，那上一步只有可能是选了1个或2个数字翻译，也就是从状态i-1, i-2转化过来。dp[i]值也由 dp[i-1]、dp[i-2]推导出来。

dp[i] = dp[i-1] + dp[i-2] (前提是2个数字不超过25)

习题课：动态规划

139. 单词拆分

给定一个**非空**字符串 *s* 和一个包含**非空**单词的列表 *wordDict*，判定 *s* 是否可以被空格拆分为一个或多个在字典中出现的单词。

说明：

- 拆分时可以重复使用字典中的单词。
- 你可以假设字典中没有重复的单词。

示例 1：

```
输入: s = "leetcode", wordDict = ["leet", "code"]
输出: true
解释: 返回 true 因为 "leetcode" 可以被拆分成 "leet code"。
```

示例 2：

```
输入: s = "applepenapple", wordDict = ["apple", "pen"]
输出: true
解释: 返回 true 因为 "applepenapple" 可以被拆分成 "apple pen apple"。
     注意你可以重复使用字典中的单词。
```

示例 3：

```
输入: s = "catsandog", wordDict = ["cats", "dog", "sand", "and", "cat"]
输出: false
```

每个阶段从单词列表中选择一个可以匹配的单词

boolean dp[n+1] dp[i]表示s[0~i-1]，也就是长度为i的字符串被拆分

假设wordlist中word1、word2、word3能跟s[0~i-1]后缀匹配，那说明到达i这个状态，只有可能从i-len1，i-len2, i-len3这三个状态过来

dp[i] = dp[i-len1] || dp[i-len2] || dp[i-len3]

习题课：动态规划

王争的算法训练营

139. 单词拆分

```
class Solution {
    public boolean wordBreak(String s, List<String> wordDict) {
        int n = s.length();
       // dp[i]表示长度为i的字符串是可拆分的
        boolean[] dp = new boolean[n+1];
        dp[0] = true;
        for (int i = 1; i <= n; i++) { //i表示长度
            for (String word : wordDict) { //走法
                int len = word.length();
                int startp = i-len;
                if (startp>=0 && s.startsWith(word, startp) && dp[i-len]) {
                    dp[i] = true;
                    break;
                }
            }
        }
        return dp[n];
    }
}
```

每个阶段从单词列表中选择一个可以匹配的单词

boolean dp[n+1] dp[i]表示s[0~i-1]，也就是长度为i的字符串被拆分

假设wordlist中word1、word2、word3能跟s[0~i-1]后缀匹配，那说明到达i这个状态，只有可能从i-len1，i-len2, i-len3这三个状态过来

dp[i] = dp[i-len1] || dp[i-len2] || dp[i-len3]

王争的算法训练营

提问环节

王争的算法训练营

关注微信公众号“**小争哥**”，

后台回复“**PDF**”获取独家算法资料