9C 动态规划视频： b站
Dynamic programming

Jan 7
动态规划特点：
- 计数
- 求最大最小值
- 求存在性

Coin change
动态规划组成部分一： 确定状态
- 最后一步 （最优策略种使用的最后一枚硬币ak）
- 化成子问题 （最少的硬币拼出更小的面值27-ak）

动态规划组成部分二：转移方程
- f[x] = min{f[x-2] + 1, f[x - 5] + 1, f[x - 7] + 1}

动态规划组成部分三：初始条件和边界情况
- f[0] = 0， 若不能拼出Y，f[Y] = 正无穷

动态规划组成部分四：计算顺序
- 一般是从小到大
    - 看 LC 518 coin change2

January 8
LC 746 min cost climbing stairs
F[i] 表示到i位置要cost最少是多少
F [i] = min{f[i - 1] + cost[i - 1], f[i - 2] + cost[i-2]}
Follow up maybe： how to use O(1) space

January 15:
LC 62 unique path（m * n 格子）
计数型动态规划
- 确定状态
    - 最后一步： 只能从上面或者从左边那个格子过来 [m-1,n-2] / [m-2, n-1]
    - 化成子问题：f[m-1][n-1] = f[m-1][n-2] + f[m-2][n-1]
- 转移方程：
    - f[i][j] = f[i-1][j] + f[i][j-1]

- 初始条件和边界情况：
  - f[0][0] = 1　if i== 0 or j == 0 ⇒ f[i][j] = 1
  - O(mn)

LC 55 jump game

- **存在型动态规划**

常见类型

1. 坐标型动态规划 (20%)
2. 序列型动态规划（20%）
3. 划分型动态规划
4. 区间型动态规划
5. 背包型动态规划
6. 最长序列型动态规划
7. 博弈型动态规划
8. 综合类型

LC 63 unique path ||

- O(n^2) ⇒ O(M*N)

LC256 paint house(**序列型动态规划**， dp数组初始化的长度跟坐标型不一样，是因为初始化的状态意义不同)

- 用状态记录油漆房子N-2的颜色
- 转移方程： 油漆 前 i 栋房子并且房子i-1分别是 红色， 蓝色， 绿色的最小花费是 f[i][0], f[i][1], f[i][2].
  - **f [i][0] = min (f[i-1][1] + cost[i-1][0] ,f[i-1][2] + cost[i-1][0])**
- 序列型动态规划， 前多少， 前0个就是空的，初始条件就是：
  - f[0][0]= f[0][1] = f[0][2] = 0　不油漆任何房子
  - 结果是 f[N][0], f[N][1], f[N][2]
  - **这里序列型和坐标型的区别可以仔细思考一下，在初始化那里。**

LC91 decode ways (划分型动态规划，也用前 i个。。。。)

- **F[0] = 1;**
- **String convert to char array**
- **Similar**
  - 62. Unique Paths
  - 70. Climbing Stairs

- 509. Fibonacci Number

Todo： 之后讲坐标型动态规划，视频P3：45:31分钟

January 26 Sunday:
Nothing

January 27 Monday:

最简单的动态规划类型
给定一个序列或网格

需要找到序列中某些/个子序列或网格中的某条路径
- 某种性质最大/最小
- 计数
- 存在性

动态规划方程 f [i] 中的下标i表示以ai 为结尾的满足条件的子序列的性质， f[i][j] 中的下标i，j表示以格子(i,j)为结尾的满足条件的路径的性质
- 最大值/最小值
- 个数
- 是否存在

January 28 Tuesday:

LC longest continuous increasing subsequence
- 使用滚动数组压缩空间到常数项

LC 64 minimum path sum:
- 子问题
  - 状态
- 转移方程
  - f[i][j] = min{f[i-1][j], f[i][j-1]} + A[i][j] 从格子(0,0)走到格子(i,j)的最小数字和为f[i][j]
- 初始条件，边界值
- 计算顺序
- 滚动数组实现空间优化 rolling array

LC 361 bomb enemy:

- 首先假设能在任何位置放炸弹 up[i][j] = (up[i-1][j] || up[i-1][j] + 1 || 0)
- **注意这里的grid[i][j] == 'W'； 不能使用双引号， 不知道为啥**
- **Check later**

序列 + 位操作型动态规划

& 与， | 或， ^异或， ！非

LC 338 counting bits
首先，计算机里关于bit的处理，比如>>，&， |等的时间复杂度都是O（1）
设f[i]表示i的二进制表示中有多少个1。
注意： 和位操作的动态规划一般用值作状态
f[i] = f[i>>1] +(i mod 2) mod这里表示最后一位是不是1

Todo january 28/ 下午2:38######下次该看 p4了

February 3 晚上9:45
序列型动态规划
动态规划方程中f[i]中的下标i 表示前i个元素a[0], a[1]... a[i-1]
初始化中， f[0]表示空序列的性质

- 指以a[0]结尾的子序列的性质

LC265 paint house ||
优化 f[i][j] = min (k!=j) {f[i-1][k]} + cost[i-1][j]
这个式子会多次求重复最小值，可优化，找全局最小值和次小值
假设最小值是f[i-1][a]， 次小值f[i-1][b]，对于j = 1，2，3，。。。a-1，a+1….K,
f[i][j] = f[i-1][a] + cost [i-1][j]
f[i][a] = f[i-1][b] + cost[i-1][a]

LC 384 house robber
LC 213 house robber ||
LC best time to buy and sell stock
LC 123 best time to buy and sell stock ||| ======没看太懂

- dp组成部分一
  - 确定状态， 最优策略中最后一次卖发生在第j天
  - 枚举最后一次买发生在第几天，但是不知道之前有没有买卖过，需要记录买卖过多少次

LC 188. Best Time to Buy and Sell Stock IV(不会)
LC 300. Longest Increasing Subsequence

- **注意 O(nlogn)的方法！！！**

LC 354 russian doll envelopes

- **Java, arrays.sort comparator**

Feb4 12:05 看到了 p4- 03- DP on sequence 1:48

Feb 10 看 04- Dynamic P on partitioning, game theory and backpack

划分型的入门题：
LC 279 perfect squares

LC132 palindrome partitioning || (hard)
When you see partitioning , dp most likely
设dp[i] 为S 前i个字符S[0...i-1]最少可以被划分成几个回文串

Upper bound and lower bound
回文串种类：

- 奇数个
- 偶数个

生成回文串

遇到string 先换成array toCharArray
Char[] s = ss.toCharArray();

LintCode 437 copy books
划分型
Assume dp[k][i] 为k个抄写员最少需要多少时间完成前i本书

04 - DP 44分钟 博弈型DP开始
LintCode 394 coins in a line
博弈的DP从第一步开始分析

Feb 11 tuesday

Backpack 1:07:24 in 04
LintCode 92 backpack
boolean
背包问题： dp数组大小跟总承重有关 ==== 1:17:35
背包问题的空间优化很常见
状态 f[i][w] 表示能否前i个物品拼出重量w （T/F）， 背包问题要把总承重放入状态

LintCode backpack V 563 **p5- 04- dynamic P 91:57 空间优化讲解！！！只开一个数组**
计数
进一步优化空间 f[i][w] = f[i-1][w] + f[i-1][w- Ai-1]
!!

Lintcode 564 backpack VI combination sum
https://leetcode.com/problems/combination-sum/discuss/16502/A-general-approach-to-backtracking-questions-in-Java-(Subsets-Permutations-Combination-Sum-Palindrome-Partitioning)

Dp 第五讲 侯卫东 p6- 05 DP
背包问题
区间型dp

Lintcode 125 backpack ||

Feb 12 wednesday:
LintCode 125

- 求前N个物品能不能拼出重量0，1，2。。。M， 以及拼出重量W能获得的最大价值。
- 需要知道前N-1个物品能不能拼出0，1，2.。。M以及拼出W时获得的最大价值。
- 子问题
- 状态 设f[i][w] 前i个物品拼出重量w时的最大价值（用-1表示不能拼出w）

F[i][w] = max{f[i-1][w], f[i-1][w- Ai-1] + Vi-1 | w>= Ai-1 and f[i-1][w-Ai-1] != -1}

LintCode 440 backpack |||　lock in LintCode 这道题在 p6-05的49分钟左右讲解

f[i][w] = 前 i种物品拼出重量w时最大总价值 （-1表示不能拼出）优化空间，查看找重复步骤
f[i][w] = max{f[i-1][w] , f[i-1][w - Ai-1] + Vi-1， f[i-1][w- 2Ai-1] + 2Vi-1…..}
== f[i][w] = max{f[i-1][w]，f[i][w-Ai-1] + Vi-1}

背包总结(在50分钟左右)
Backpack 1 可行性背包
Backpack V VI 计数型
Backpack II III 最值型
1 最后一步

- 最后一个背包内的物品时哪个
- 最后一个物品有没有进背包

数组大小和最大承重target有关
空间优化

Feb 13 Thursday:
区间型dp
Onsite的话面试前先上个厕所，清醒一下

区间： 给定一个序列/字符串，最后一步会将序列去头/去尾
剩下的会是一个区间 [i, j]
状态自然定义尾 f [i][j]

LC 516 longest palindromic subsequence (subsequence 可以不是连续的)
初始条件：按长度来的
按照长度从 j - i 从小到大长度!!!!!!!!!! 区间dp的特点
F[0][0] = f[1][1] = f[2][2] = 1

记忆化搜索
相当于递归， memorized- recursion， 自上而下
但是dp可以滚动数组优化空间，但是记忆化搜索不行

Lintcode 396 coins in a line |||　lintcode锁着

LC 87 scramble string 05- dp on knapsack and intervals (3) (hard)

Todo 8:23 in 这个video

Feb 14
LC 312 burst balloons
消去型，可以相当成区间,用最后一步去考虑，最后一个被扎破的气球
子问题： 扎破 1～i-1号，保留0和i号， 和 i+1～N，保留i和N+1
枚举最后一个被扎破的气球
设f[i][j]为扎破 i+1 ～ j - 1 号气球最多获得的金币数， i和j不能扎破
F[0][1] = f[1][2] = ….. 0

背包dp
物品重量，价值
状态用物品个数和当前重量
单个物品，无限多物品

区间型dp

- 状态用区间左右端点 （先用最大， 然后在loop里面break超过范围的情况）
- 记忆化搜索

开始06- dp on double sequence
双序列型dp 考虑尾巴， 要么砍掉A的尾巴，要么B的尾巴，要么都砍掉
初始条件， 空和别人的关系！ 或者空和null

LC1143 Longest Common Subsequence
之前我自己做出来， 是用 2-d的方法做的
f[0][j] = 0

如果要记录序列，可以把每次的决策记下来，然后打印
=======看到了0:24:58 分钟视频

Feb 16 sunday
06- dp with double sequence 从24:58 开始

LC 97 interleaving string
f[i][j] |= f[i-1][j] 就是这个等于 or， 用在boolean

LC 72 edit distance **滚动数组看一下！！！ 可以先正常2维的做一下，在用now和old来改**
Delete, add, replace, same(nothing to do)

前一行是old，这一行是now

LC 115 distinct subsequence 可以之后想一想
f[i][j] 为B的前j个字符[0，j-1] 在A前i个字符[0, i-1] 中出现的次数
f[i][1] = 1

LC 10 regular expression matching

f[i][j] = 1::: f[i-1][j-1] if (B[j-1] == '.' or A[i-1] == B[j-1])
2::: f[i][j-2] OR (f[i-1][j] AND (B[j-2] == '.' OR B[j-2] = A[i-1])) if B[j-1] = '*'

数组的初始值， 还是赋值一下，有好处， 比如false 等

LC 44 Wildcard matching
注意boolean的时候， 不能是 =， 而是 |= 或等于 这几道题很相似

LC 474 ones and zeros
f[i][j][k] 前i个 01 串中最多有多少被j个0和k个1组成

设 Si中有ai个0， bi个1
f[i][j][k] = max{f[i-1][j][k], f[i-1][j-a(i-1)[k-b(j-1)] **+ 1** | j >= a(i-1) AND k >= b(j-1)}
滚动数组的改法
i ⇒ now， i-1 ⇒ old

这个视频完了。 该看p10

Feb 17 monday
最后一个视频
辅助数据结构/算法 with dp 字母树， 哈希表，二分查找等

Lintcode 91 minimum adjustment cost
最小修改代价： 最值型dp。 从最后一步入手
首先证明修改后的元素是1～100的

最后一步： 将A 改成B， A[n-1]改成X，这一步代价是 |A[n-1] - X|， 同时，确保 |X - B[n-2]| <= target
记录下来：序列加状态
设状态f[i][j]将A的前i个元素改成B的最小代价，确保前i个元素中任意两个相邻的元素的差不超过target，并且A[i-1]改为j
如果A[i-1]为j， A[i-2]必须改为在 j-target～j+target之间且在1～100之间

这里可能因为 范围是1～100， 所以n没有0的取值?????????

滚动数组
In java , if A is an ArrayList function : are A.get, A.size(), Integer.MAX_VALUE

f[i][j] = min (j-target<=k<=j+target, 1<=k<=100){f[i-1][k] + |j-A[i-1]|}

Init:
A[0] could be any one
A[0][j] = |j-A[0]|(j = 1,2,3...100)

LintCode 89 K sum （背包问题）
Dp组成部分一： 确定状态
最后一步： 最后一个数是A(n-1) 是否选入这K个数
1. A(n-1)不选入：需要在前n-1个数中选K个数，使的它们的和是target
2. A(n-1)选入： 需要在前n-1个数中选k-1个数，使它们的和是Target - A(n-1)

需要知道还有几个数可选，以及它们的和需要是多少： 序列加状态
状态： f[i][k][s] 表示多少种方法可以在前i个数中选出k个，使得它们的和是s

Dp组成部分二： 转移方程
f[i][k][s] = f[i-1][k][s] + f[i-1][k-1][s - A(i-1)] | s>= A(i-1)

Dp组成部分三： init and 边界条件
f[0][0][0] = 1 什么都不干
f[0][0][s] = 0 also f[0][i][j] = 0 except 000

Dp组成部分四：计算顺序

滚动数组

Int now ,old = 0
Old = now
Now = 1-now
i⇒ now i-1 ⇒ old

LC 300. Longest Increasing Subsequence
Follow up 二分优化查找， 44分钟左右
将视频截图ppt存在了ipad里面

Lintcode 623 K edit distance

字符串共享前缀，避免重复
数据结构**Trie**： 字母树
生成Trie

List<String> 不支持加加加加加， 所以用arrayList

序列 + 哈希表 todo！ 看到视频的1:31:21
LC 403 frog jump

Feb 18 Tuesday
LC403 frog jump
Dp组成部分1: 确定状态

- 最后一步：如果可以跳到最后一个石头a(n-1)， 考虑最后跳的一步L
- 青蛙一定是从某块石头a(i) = a(n-1) - L 跳来的
- 所以考虑能否跳到ai
- 还是知道怎么跳到ai的，倒数第二跳只能是L-1， L， L+1

子问题

- 要求是否能最后一跳L跳到最后一个石头a(n-1)
- 需要知道最后一跳L-1，L或者L+1跳到石头ai = a(n-1) - L
- 设 f[i][j]表示是否能最后一跳长度j跳到石头ai， i 是石头index， j是跳的长度
- 坐标加状态

Dp组成部分2: 转移方程

- 设 f[i][j]表示是否能最后一跳长度j跳到石头ai， i 是石头index， j是跳的长度
- 设上一块石头是ak = ai - j， 可以通过一个哈希表（ak ⇒ k） 快速找到k
- f[i][j] = f[k][j-1] OR f[k][j] OR f[k][j+1] ak = ai - j

DP组成部分3: 初始条件和边界 情况

- 第一跳距离是1，一直向右，最多N-1步，所以一步的最大跳跃是N-1
- f[1][1] = true f[1][2] = …= f[1][N-1] = false

优化用哈希表
因为有大量false，所以用hashmap来存true的情况
HashMap use **put**
如果有重复石头？

LC 221 maximum square hard
以正方形的右下角为突破口

LC 85. Maximal Rectangle