动态规划(下)

九章算法强化班 第6章

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://zhuanlan.zhihu.com/jiuzhang
官网: http://www.jiuzhang.com

Overview

I. 区间类DP
   I. Stone Game
   II. Burst Ballons
   III. Scramble String
II. 匹配类动规
   I. Longest Common Subsequence
   II. Edit Distance
   III. K Edit Distance
   IV. Distinct Subquence
   V. Interleaving String
III. 背包类DP
   I. BackPackI
   II. BackPackII
   III. K SUM
   IV. Minimum Adjustment Cost

区间类Dp

特点：

1. 求一段区间的解max/min/count
2. 转移方程通过区间更新
3. 从大到小的更新

Stone Game

http://www.lintcode.com/en/problem/stone-game/
http://www.jiuzhang.com/solutions/stone-game/

[3,4,5,6]

贪心反例：

[6,4,4,6]

Stone-Game

- 死胡同：容易想到的一个思路从小往大，枚举第一次合并是在哪？
- 记忆化搜索的思路，从大到小，先考虑最后的0-n-1 合并的总花费
- State:
  - dp[i][j] 表示把第i到第j个石子合并到一起的最小花费
- Function:
  - 预处理sum[i,j] 表示i到j所有石子价值和
  - dp[i][j] = min(dp[i][k]+dp[k+1][j]+sum[i,j]) 对于所有k属于{i,j}
- Intialize:
  - for each i
    - dp[i][i] = 0
- Answer:
  - dp[0][n-1]

Burst Ballons

http://www.lintcode.com/en/problem/burst-balloons/

http://www.jiuzhang.com/solutions/burst-ballons/

Burst Ballons

- 死胡同：容易想到的一个思路从小往大，枚举第一次在哪吹爆气球？
- 记忆化搜索的思路，从大到小，先考虑最后的0-n-1 合并的总价值

- State:
  - dp[i][j] 表示把第i到第j个气球打爆的最大价值
- Function:
  - 对于所有k属于{i,j}, 表示第k号气球最后打爆。
    - midValue = arr[i-1] * arr[k] * arr[j+1];
    - dp[i][j] = max(dp[i][k-1]+dp[k+1][j]+midvalue)
- Intialize:
  - for each i
    - dp[i][i] = arr[i-1] * arr[i] * arr[i+1];
- Answer:
  - dp[0][n-1]

Scramble String

http://www.lintcode.com/en/problem/scramble-string/
http://www.jiuzhang.com/solutions/scramble-string/
[abcd, dcab]

Scramble String

- 看 f[great][rgreat] 这个参考例子
- f[gr|eat][rgreat] =
  - f[gr][rg] && f[eat][eat]
  - f[gr][at] && f[eat][rgr]

Scramble String

- State:
  - dp[x][y][k] 表示是从s1串x开始，s2串y开始，他们后面k个字符组成的substr是Scramble String
- Function:
  - 对于所有i属于{1,k}
  - s11 = s1.substring(0, i); s12 = s1.substring(i, s1.length());
  - s21 = s2.substring(0, i); s22 = s2.substring(i, s2.length());
  - s23 = s2.substring(0, s2.length() - i); s24 = s2.substring(s2.length() - i, s2.length());
  - for i = x -> x+k
    - dp[x][y][k] = (dp[x][y][i] && dp[x+i][y+i][k-i]) || dp[x][y+k-i][i] && dp[x+i][y][k-i])
- Intialize:
  - dp[i][j][1] = s1[i]==s[j].
- Answer:
  - dp[0][0][len]

区间DP

- coin in a line III
- stone game
- scramble string

- 这种题目共性就是区间最后求[0,n-1] 这样一个区间
- 逆向思维分析 从大到小就能迎刃而解

- 逆向=》分治类似

匹配类动态规划

匹配类动态规划

- state: f[i][j]代表了第一个sequence的前i个数字/字符，配上第二个sequence的前j个...
- function: f[i][j] = 研究第i个和第j个的匹配关系
- initialize: f[i][0] 和 f[0][i]
- answer: f[n][m] min/max/数目/存在关系
- n = s1.length()
- m = s2.length()

解题技巧画矩阵，填写矩阵

Longest Common Subsequence

http://www.lintcode.com/problem/longest-common-subsequence/

http://www.jiuzhang.com/solutions/longest-common-subsequence/

求Max

Longest Common Subsequence

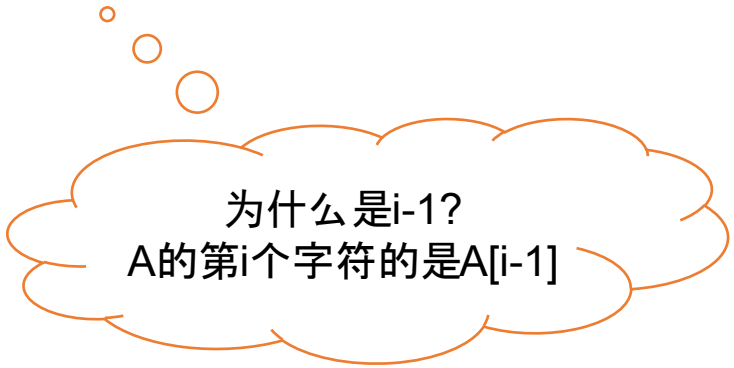
•

	null	A	B	C	D
null	0	0	0	0	0
E	0				
A	0				
C	0				
B	0				

	null	A	B	C	D
null	0	0	0	0	0
E	0	0	0	0	0
A	0	1	1	1	1
C	0	1	1	1	1
B	0	1	2	2	2

Longest Common Subsequence

- state: f[i][j]表示前i个字符配上前j个字符的LCS的长度
- function: f[i][j] = MAX(f[i-1][j], f[i][j-1], f[i-1][j-1] + 1) // A[i - 1] == B[j - 1]
- = MAX(f[i-1][j], f[i][j-1]) // A[i - 1] != B[j - 1]
- intialize: f[i][0] = 0 f[0][j] = 0
- answer: f[n][m]



- Related Question:
- http://www.lintcode.com/problem/longest-common-substring/

Edit Distance

http://www.lintcode.com/problem/edit-distance/

http://www.jiuzhang.com/solutions/edit-distance/

求Min

Longest Common Subsequence

•

	null	m	a	r	t
null	0	1	2	3	4
k	1				
a	2				
r	3				
m	4				
a	5				

	null	m	a	r	t
null	0	1	2	3	4
k	1	1	2	3	4
a	2	2	1	2	3
r	3	3	2	1	2
m	4	3	3	2	2
a	5	4	3	3	3

Edit Distance

- state: f[i][j]表示A的前i个字符最少要用几次编辑可以变成B的前j个字符
- function: f[i][j] = MIN(f[i-1][j]+1, f[i][j-1]+1, f[i-1][j-1]) // A[i - 1] == B[j - 1]
- = MIN(f[i-1][j]+1, f[i][j-1]+1, f[i-1][j-1]+1) // A[i - 1] != B[j - 1]
- initialize: f[i][0] = i, f[0][j] = j
- answer: f[n][m]

K Edit Distance

http://www.lintcode.com/en/problem/k-edit-distance/

http://www.jiuzhang.com/solutions/k-edit-distance/

Distinct Subsequence

http://www.lintcode.com/problem/distinct-subsequences/

http://www.jiuzhang.com/solutions/distinct-subsequences/

求方案总数

Longest Common Subsequence

-

	null	r	a	b	b	i	t
null	1	0	0	0	0	0	0
r	1						
a	1						
b	1						
b	1						
b	1						
t	1						

	null	r	a	b	b	i	t
null	1	0	0	0	0	0	0
r	1	1	0	0	0	0	0
a	1	1	1	1	0	0	0
b	1	1	1	2	1	0	0
b	1	1	1	3	3	0	0
b	1	1	1	3	3	3	0
t	1	1	1	3	3	3	3

Distinct Subsequence

- state: f[i][j] 表示 S的前i个字符中选取T的前j个字符，有多少种方案
- function: f[i][j] = f[i - 1][j] + f[i - 1][j - 1] // S[i-1] == T[j-1]
- = f[i - 1][j] // S[i-1] != T[j-1]
- initialize: f[i][0] = 1, f[0][j] = 0 (j > 0)
- answer: f[n][m] (n = sizeof(S), m = sizeof(T))

Interleaving String

http://www.lintcode.com/problem/interleaving-string/

http://www.jiuzhang.com/solutions/interleaving-string/

求是否可行

Interleaving String

- state: f[i][j]表示s1的前i个字符和s2的前j个字符能否交替组成s3的前i+j个字符
- function: f[i][j] = (f[i-1][j] && (s1[i-1]==s3[i+j-1]) ||
- (f[i][j-1] && (s2[j-1]==s3[i+j-1])
- initialize: f[i][0] = (s1[0..i-1] == s3[0..i-1])
- f[0][j] = (s2[0..j-1] == s3[0..j-1])
- answer: f[n][m], n = sizeof(s1), m = sizeof(s2)

背包类Dp

特点：

1. 用值作为DP维度
2. Dp过程就是填写矩阵
3. 可以滚动数组优化

BackPack

http://www.lintcode.com/en/problem/backpack/
http://www.jiuzhang.com/solutions/backpack/

Backpack

- State:
  - f[i][S] “前i”个物品，取出一些能否组成和为S
  - [2, 3, 5, 7], 11

	体积0	体积1	体积2	体积3	体积4	体积5	体积6	体积7	体积8	体积9	体积10	体积11
前0个物品	1	0	0	0	0	0	0	0	0	0	0	0
前1个物品	1	0	1	0	0	0	0	0	0	0	0	0
前2个物品	1	0	1	1	0	1	0	0	0	0	0	0
前3个物品	1	0	1	1	0	1	0	1	1	0	1	0

Backpack

- State:
  - f[i][S] “前i”个物品，取出一些能否组成和为S
- Function:
  - a[i-1] 是第i个物品下标是i-1
  - f[i][S] = f[i-1][S - a[i-1]] or f[i-1][S]
- Intialize:
  - f[i][0] = true; f[0][1..target] = false
- Answer:
  - 检查所有的f[n][j]
- O(n*S) ， 滚动数组优化

BackPack 马甲题型

把一个[1,24,5,6]数组尽量平分。

Backpack II

http://www.lintcode.com/en/problem/backpack-ii/
http://www.jiuzhang.com/solutions/backpack-ii/

贪心反例:
背包容量 = 9
A=[4,5,7]
V=[3,4,6]

Backpack II

- 状态 State
  - f[i][j] 表示前i个物品当中选一些物品组成容量为j的最大价值
- 方程 Function
  - f[i][j] = max(f[i-1][j], f[i-1][j-A[i-1]] + V[i-1]);
- 初始化 Intialization
  - f[0][0]=0;
- 答案 Answer
  - f[n][s]
- O(n*s)

Backpack

- State:
  - f[i][S] “前i”个物品，取出一些能否组成和为S
  - size [2, 3, 5, 7] and value [1, 5, 2, 4]

	体积0	体积1	体积2	体积3	体积4	体积5	体积6	体积7	体积8	体积9	体积10	体积11
前0个物品	0	0	0	0	0	0	0	0	0	0	0	0
前1个物品	0	0	1	0	0	0	0	0	0	0		
前2个物品	0	0	1	5								
前3个物品	0											

BackPack IV 个数无限

http://www.lintcode.com/en/problem/backpack-iv/

http://www.jiuzhang.com/solutions/backpack-iv/

Backpack

- State:
  - f[i][S] “前i”个物品，取出一些能否组成和为S

	体积0	体积1	体积2	体积3	体积4	体积5	体积6	体积7	体积8	体积9	体积10	体积11
前0个物品	0	0	0	0	0	0	0	0	0	0	0	0
前1个物品	0											
前2个物品	0											
前3个物品	0											

Backpack

- State:
  - f[i][S] “前i”个物品，取出一些能否组成和为S
- Function:
  - a[i-1] 是第i个物品下标是i-1
  - k 是第i个物品选取的次数
  - f[i][S] = f[i-1][S - k*a[i-1]] or f[i-1][S]
- Intialize:
  - f[i][0] = true; f[0][1..target] = false
- Answer:
  - 答案是f[n][S]

K Sum

http://www.lintcode.com/en/problem/k-sum/
http://www.jiuzhang.com/solutions/k-sum/

K Sum

- n个数，取k个数，组成和为target
- State:
  - f[i][j][t]前i个数取j个数出来能否和为t
- Function:
  - f[i][j][t] = f[i - 1][j - 1][t - a[i-1]] + f[i - 1][j][t]
- Intialization
  - f[i][0][0] = 1
- Answer
  - f[n][k][target]

Minimum Adjustment Cost

http://www.lintcode.com/en/problem/minimum-adjustment-cost/

最小调整代价

- State:
  - f[i][v] 前i个数，第i个数调整为v，满足相邻两数<=target，所需要的最小代价
- Function:
  - f[i][v] = min(f[i-1][v'] + |A[i]-v|, |v-v'| <= target)
- Answer:
  - f[n][a[n]-target~a[n]+target]

- O(n * A * T)

Summary

- 区间类DP问题
  - 从大到小去思考
  - 主要是通过记忆化来直观理解DP的思路
- 双序列类DP问题
  - 二维数组
  - 画出矩阵的表格，填写矩阵
- 背包DP问题
  - 用值作为DP维度
  - DP过程就是填写矩阵
  - 可以滚动数组优化

今日重点题型

- Edit Distance
  - 双序列常考题
- Stone-Game
  - 区间类DP的入门题
- Backpack II
  - 有价值的背包题目才有价值

Thank You