九章算法强化班第六讲
动态规划(下) - 双序列与背包

主讲人 侯卫东

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

- 双序列型DP
  - Longest Common Subsequence
  - Interleaving String
  - Edit Distance
  - K Edit Distance

- 背包型DP
  - BackPack I
  - BackPack II
  - BackPack III
  - K SUM

双序列型动态规划

两个序列/字符串的最优值/方案数/可行性

可以使用滚动数组优化空间

Longest Common Subsequence

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

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

LintCode 77

九章算法

- 给定两个字符串A，B
- 一个字符串的子序列是这个字符串去掉某些字符（可能0个）之后剩下的字符串
- 找到两个字符串的最长公共子序列的长度
- 例子：
- 输入：A="jiuzhang", B="lijiang"
- 输出：5（最长公共子序列是jiang ）

分析：

- 确定状态：
  - 最后一步：观察A[m-1]和B[n-1]这两个字符是否作为一个对子在最优策略中
  - 状态：设f[i][j]为A前i个字符A[0..i-1]和B前j个字符[0..j-1]的最长公共子序列的长度
- 转移方程：f[i][j] = max{f[i-1][j], f[i][j-1], f[i-1][j-1]+1|A[i-1]=B[j-1]}
- 初始条件和边界情况：
  - f[0][j] = 0, j=0..n
  - f[i][0] = 0, i=0..m
- 计算顺序：
  - f[0][0..n]
  - …
  - f[m][0..n]
- 答案是f[m][n]
- 时间复杂度：O(MN)，空间复杂度：O(MN)，可以用滚动数组优化空间至O(N)

编程

Edit Distance

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

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

LintCode 119

- 给定两个字符串A，B
- 要求把A变成B，每次可以进行下面一种操作：
  - 增加一个字符
  - 去掉一个字符
  - 替换一个字符
- 最少需要多少次操作，即*最小编辑距离*

- 例子：
- 输入：A=“mart”, B=“karma”
- 输出：3 (m换成k，t换成m，加上a)

分析：

- 确定状态：
  - 最后一步：观察B[n-1]是如何产生的（插入，删除，替换，和A[m-1]匹配）
  - 状态：设f[i][j]为A前i个字符A[0..i-1]和B前j个字符B[0..j-1]的最小编辑距离
- 转移方程：f[i][j]=min{f[i][j-1]+1, f[i-1][j-1]+1, f[i-1][j]+1, f[i-1][j-1]|A[i-1]=B[j-1]}
- 初始条件和边界情况：
  - f[0][j] = j, j=0..n
  - f[i][0] = i, i=0..m
- 计算顺序：
  - f[0][0..n]
  - …
  - f[m][0..n]
- 答案是f[m][n]
- 时间复杂度：O(MN)，空间复杂度：O(MN)，可以用滚动数组优化空间至O(N)

Interleaving String

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

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

LintCode 29

九章算法

- 给定三个字符串A, B, X
- 判断X是否是由A, B交错在一起形成
  - 即A是X的子序列，去掉A后，剩下的字符组成B

- 例子：
- 输入：A=“aabcc” B=“dbbac”, X=“aadbbcbcac”
- 输出：True（ X=“aadbbcbcac” ）

分析

- 最后一步：假设X是由A和B交错形成的，那么X的最后一个字符X[m+n-1]要么是A[m-1]，要么是B[n-1]
- 状态：f[i][j]=X前i+j个字符是否由A前i个字符和B前j个字符交错形成
- 转移方程：f[i][j] = (f[i-1][j] AND X[i+j-1]==A[i-1])OR(f[i][j-1] AND X[i+j-1]==B[j-1])
- 初始条件：f[0][0]=True
- 计算顺序：
  - f[0][0], f[0][1], …, f[0][n]
  - …
  - f[m][0], f[m][1], …, f[m][n]
- 时间复杂度（计算步数）O(MN)，空间复杂度（数组大小）O(MN)，可以用滚动数组优化空间至O(N)

K Edit Distance

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

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

LintCode 623

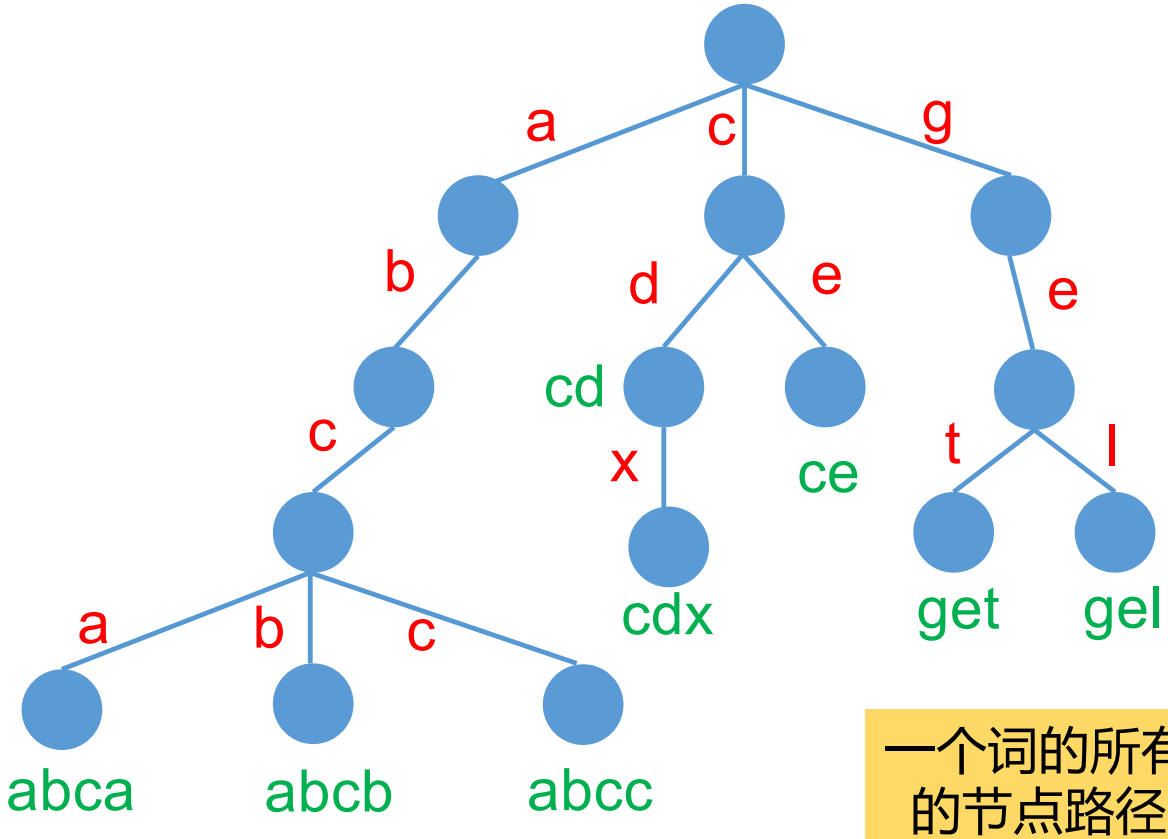
- 给定N个字符串，以及目标字符串T
- 问哪些字符串和T的编辑距离不大于K
- 一次编辑包括插入一个字符或删除一个字符或修改一个字符

- 例子：
- 输入：
  - A = ["abc", "abd", "abcd", "adc"]
  - Target = "ac”
  - K = 1
- 输出： ["abc", "adc"]

分析

- 如果依次用Edit Distance得到结果，存在重复计算
  - 如果给定的字符串是"abca", "abcb", "abcc"
  - 三个字符串的前3个字符都一样
  - "abca"前0~3个字符和Target前0~n个字符的最小编辑距离
  - "abcb"前0~3个字符和Target前0~n个字符的最小编辑距离
  - "abcc"前0~3个字符和Target前0~n个字符的最小编辑距离
- 如何避免重复计算
- 如果几个字符串共享一段前缀，他们对应的f[i][j]可以共享，即只计算一次
- 数据结构Trie：字母树

字母树Trie



一个词的所有前缀就是从根到这个词的节点路径上形成的所有的字符串

分析

- 状态：$f[s_P][j]$为前缀s_P(即节点P对应的字符串)和T前j个字符的最小编辑距离
- 设P的父亲是Q
- 转移方程：$f[s_P][j] = \min\{f[s_P][j-1]+1, f[s_Q][j-1]+1, f[s_Q][j]+1, f[s_Q][j-1] | s_P[last]=T[j-1]$

- 初始条件：一个空串和一个长度为L的串的最小编辑距离是L
  - $f[s_{root}][j] = f[""][j] = j$ (j = 0, 1, 2, …, n)
  - $f[s_p][0] = length(s_p)$

- 计算顺序
  - 初始化$f[s_{root}][0]$~$f[s_{root}][n]$
  - 按照字母树深度优先搜索顺序计算每个$f[s_P][0]$~$f[s_P][n]$
  - 答案是满足$f[s_P][n]<=K$且s_P在给定列表中的所有单词
  - 用DFS实现
  - 时间复杂度（计算步数）O(前缀个数*N)
  - 空间复杂度（数组大小）O(前缀个数 + 最长单词长度*N)

编程

课间休息五分钟

背包型DP

BackPack

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

LintCode 92

九章算法

- 给定N个物品，重量分别为正整数$A_0, A_1, \ldots, A_{N-1}$
- 一个背包最大承重是正整数M
- 最多能带走多重的物品

- 例子：
- 输入：4个物品，重量为2, 3, 5, 7. 背包最大承重是11
- 输出：10（三个物品：2, 3, 5）

分析：

- 每个装物品的方案的总重量都是0到M，如果对于每个总重量，我们能知道有没有方案能做到，就可以解决
- **背包问题中，数组大小和总承重有关**
- 最后一步：最后一个物品（重量A_{N-1}）是否进入背包
- 状态：设f[i][w] = 能否用**前i个物品**拼出重量w (TRUE / FALSE)
- 常见误区：**错误** 设f[i]表示前i个物品能拼出的最大重量（不超过M）
  - 反例：A=[3 9 5 2], M=10
  - 错误原因：最优策略中，前N-1个物品拼出的**不一定是**不超过M的最大重量

分析：

- 转移方程：f[i][w] = f[i-1][w] OR f[i-1][w-A_{i-1}]

- 初始条件和边界情况：
  - f[0][0] = TRUE: 0个物品可以拼出重量0
  - f[0][1..M] = FALSE: 0个物品不能拼出大于0的重量
  - f[i-1][w-A_{i-1}]只能在w≥A_{i-1}时使用

- 计算顺序：
  - f[0][0..m]
  - …
  - f[n][0..m]
- 答案是最大的j使得f[n][j]=True
- 时间复杂度：O(MN)，空间复杂度：O(MN)，可以用滚动数组优化空间至O(N)

BackPack 马甲题型

把一个[1,24,5,6]数组尽量平分。

Backpack II

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

LintCode 125

- 给定N个物品，重量分别为正整数$A_0, A_1, \ldots, A_{N-1}$，价值分别为正整数$V_0, V_1, \ldots, V_{N-1}$
- 一个背包最大承重是正整数M
- 最多能带走多大价值的物品

- 例子：
- 输入：4个物品，重量为2, 3, 5, 7，价值为1, 5, 2, 4. 背包最大承重是11
- 输出：9（物品一+物品三，重量3+7=10，价值5+4=9）

分析：

- 每个装物品的方案的总重量都是0到M，如果对于每个总重量，我们能知道对应的最大价值是多少，就能知道答案
- **背包问题中，数组大小和总承重有关**
- 最后一步：最后一个物品（重量A_{N-1}, 价值V_{N-1}）是否进入背包
- 状态：设f[i][w] = 用**前i个物品**拼出重量w时最大总价值 (-1表示不能拼出w)

分析：

- 转移方程：$f[i][w] = \max\{f[i-1][w], f[i-1][w-A_{i-1}] + V_{i-1} \mid w \geq A_{i-1}$ 且 $f[i-1][w-A_{i-1}] \neq -1\}$
- 初始条件和边界情况：
  - f[0][0] = 0: 0个物品可以拼出重量0，最大总价值是0
  - f[0][1..M] = -1: 0个物品不能拼出大于0的重量
  - $f[i-1][w-A_{i-1}]$只能在**$w \geq A_{i-1}$，并且$f[i-1][w-A_{i-1}] \neq -1$**时使用
- 计算顺序：
  - f[0][0..m]
  - …
  - f[n][0..m]
- 答案是$\max_{0<=j<=M}\{f[N][j] \mid f[N][j] \neq -1\}$

编程

Backpack III

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

LintCode 440

九章算法

- 给定N种物品，重量分别为正整数$A_0, A_1, \ldots, A_{N-1}$，价值分别为正整数$V_0, V_1, \ldots, V_{N-1}$
- **每种物品都有无穷多个**
- 一个背包最大承重是正整数M
- 最多能带走多大价值的物品

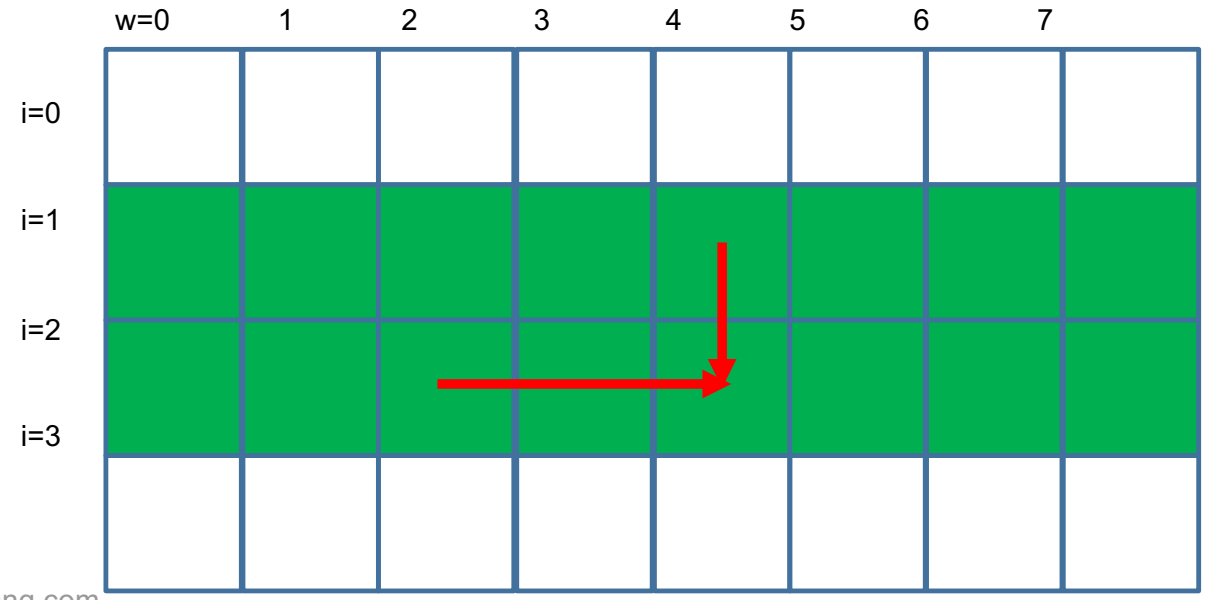
- 例子：
- 输入：4个物品，重量为2, 3, 5, 7，价值为1, 5, 2, 4. 背包最大承重是10
- 输出：15（3个物品一，重量3*3=9，价值5*3=15）

分析：

- 和Backpack II唯一的不同是：每种物品都有**无穷多个**
- 状态：设f[i][w] = 用前i种物品拼出重量w时最大总价值 (-1表示不能拼出w)
- 转移方程：$f[i][w] = \max_{k>=0}\{f[i-1][w-kA_{i-1}] + kV_{i-1}\}$
- 可以转化为$f[i][w] = \max\{f[i-1][w], f[i][w-A_{i-1}] + V_{i-1}\}$

优化

- f[i][w] = max{f[i-1][w], f[i][w-A_{i-1}] + V_{i-1}}
- 实际编程中可以进一步优化



编程

K Sum

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

LintCode 89

九章算法

- 给定数组A，包含n个互不相等的正整数
- 问有多少种方式从中找出K个数，使得它们的和是Target

- 例子：
- 输入：A=[1, 2, 3, 4], K=2, Target = 5
- 输出：2 (1 + 4 = 5, 2 + 3 = 5)

分析

- 最后一步：最后一个数A_{n-1}是否选入这K个数
- 状态：f[i][k][s]表示有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
- 转移方程：f[i][k][s] = f[i-1][k][s] + f[i-1][k-1][s-A_{i-1}]|k>=1 AND s>=A_{i-1}
- 初始条件：
  - f[0][0][0] = 1
  - f[0][0][s] = 0, s = 1, 2, …, Target
- 计算顺序：
  - f[0][0~K][0~Target]
  - …
  - f[N][0~K][0~Target]
- 时间复杂度O(N*K*Target)，空间复杂度O(N*K*Target)，可以用滚动数组优化至O(K*Target)

总结

- 双序列型DP问题
  - 二维数组
  - 最后一步考虑两个序列最后的元素
  - 可以滚动数组优化

- 背包型DP问题
  - 用背包承重作为DP维度之一
  - 最后一步考虑最后一个物品是否进背包
  - 可以滚动数组优化

今日重点题型

- Longest Common Subsequence
  - 双序列常考题
- Edit Distance
  - 双序列常考题
- Backpack II
  - 有价值的背包题目才有价值

Thank You