

本节大纲

九章算法

- 背包动态规划
- 区间型动态规划

LintCode 125: Backpack II

九章算法

- 题意:
- 给定 N 个物品，重量分别为正整数 A_0, A_1, \dots, A_{N-1} ，价值分别为正整数 V_0, V_1, \dots, 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 $5+4=9$ ）

背包动态规划：如何下手

九章算法

- 给定 N 个物品，重量分别为 A_0, A_1, \dots, A_{N-1} ，价值分别为 V_0, V_1, \dots, V_{N-1}
- 一个背包最大承重是 M
- 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

0	1	2	...	M-2	M-1	M
\$0	✗	\$3	\$5	✗	\$10	\$8

动态规划组成部分一：确定状态



- 和前一题类似，需要知道 N 个物品
 - 是否能拼出重量 W ($W=0, 1, \dots, M$)
 - 对于每个重量 W ，最大总价值是多少
- 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

选择一：如果前 $N-1$ 个物品能拼出 W ，最大总价值是 V ，前 N 个物品也能拼出 W 并且总价值是 V

选择二：如果前 $N-1$ 个物品能拼出 $W - A_{N-1}$ ，最大总价值是 V ，则再加上最后一个物品（重量 A_{N-1} ，价值 V_{N-1} ），能拼出 W ，总价值是 $V + V_{N-1}$

0	1	2	...	M-2	M-1	M
\$0	✗	\$3	\$5	✗	\$10	\$8
						✗

动态规划组成部分一：确定状态



- 例子：
- 3个物品，重量为2, 3, 5，价值为10, 20, 50
- 前2个物品可以拼出重量5 (2+3)，最大价值是30 (10+20)，前3个物品也可以拼出重量5，价值30
- 前2个物品可以拼出重量0，价值0，加上最后一个物品，可以拼出重量5，价值50，50>30，所以所有物品拼出重量5时，最大总价值是50



子问题



- 要求前 N 个物品能不能拼出重量 $0, 1, \dots, M$ ，以及拼出重量 W 能获得的最大价值
- 需要知道前 $N-1$ 个物品能不能拼出重量 $0, 1, \dots, M$ ，以及拼出重量 W 能获得的最大价值
- 子问题
- 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)

动态规划组成部分三：转移方程

九章算法

- 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)

$$f[i][w] = \max\{f[i-1][w], f[i-1][w-A_{i-1}] + V_{i-1} \mid w \geq A_{i-1} \text{ 且 } f[i-1][w-A_{i-1}] \neq -1\}$$

用前 i 个物品拼出重量 w 时最大总价值

用前 $i-1$ 个物品拼出重量 w 时最大总价值

用前 $i-1$ 个物品拼出重量 $w-A_{i-1}$ 时最大总价值，加上第 i 个物品

动态规划组成部分三：初始条件和边界情况

九章算法

- 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)
- $f[i][w] = \max\{f[i-1][w], f[i-1][w-A_{i-1}] + V_{i-1} \mid w \geq A_{i-1} \text{ 且 } f[i-1][w-A_{i-1}] \neq -1\}$
- 初始条件：
 - $f[0][0] = 0$: 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
- 边界情况：
 - $f[i-1][w-A_{i-1}]$ 只能在 $w \geq A_{i-1}$ ，并且 $f[i-1][w-A_{i-1}] \neq -1$ 时使用

动态规划组成部分四：计算顺序

九章算法

- 初始化 $f[0][0], f[0][1], \dots, f[0][M]$
- 计算前 1 个物品拼出各种重量的最大价值： $f[1][0], f[1][1], \dots, f[1][M]$
- ...
- 计算前 N 个物品拼出各种重量的最大价值： $f[N][0], f[N][2], \dots, f[N][M]$
- 答案： $\max_{0 \leq j \leq M} \{f[N][j] \mid f[N][j] \neq -1\}$
- 时间复杂度（计算步数）： $O(MN)$ ，空间复杂度（数组大小）：优化后可以达到 $O(M)$

LintCode 440: Backpack III

九章算法

- 题意:
- 给定N种物品, 重量分别为正整数 A_0, A_1, \dots, A_{N-1} , 价值分别为正整数 V_0, V_1, \dots, 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 $5*3=15$)

背包动态规划: 如何下手

九章算法

- 和上一题唯一的的不同是: 每种物品都有无穷多个
- 一个背包最大承重是M
- Backpack II的转移方程
- 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)

$$f[i][w] = \max\{f[i-1][w], f[i-1][w-A_{i-1}] + V_{i-1}\}$$

用前i个物品拼出重量w时最大总价值

用前i-1个物品拼出重量w时最大总价值

用前i-1个物品拼出重量 $w-A_{i-1}$ 时最大总价值, 加上第i个物品

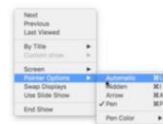
动态规划组成部分二: 转移方程

九章算法

- 因为 A_i 有无穷多个, 所以可以用0个 A_i , 1个 A_i , 2个 A_i , ...

- Backpack III的转移方程

$O(n^2)$



- 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)

$$f[i][w] = \max_{k \geq 0} \{f[i-1][w-kA_{i-1}] + kV_{i-1}\}$$

用前i种物品拼出重量w时最大总价值

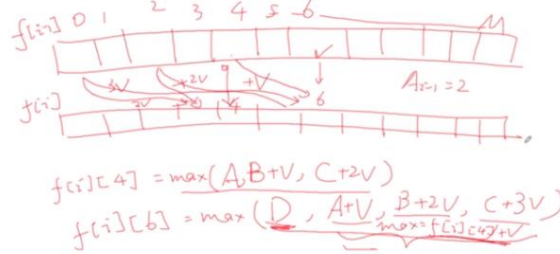
用前i-1种物品拼出重量 $w-kA_{i-1}$ 时最大总价值, 加上k个第i种物品

Backpack III优化

九章算法

- 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)

$$f[i][w] = \max\{f[i-1][w], f[i-1][w-A_{i-1}] + V_{i-1}, f[i-1][w-2A_{i-1}] + 2V_{i-1}, \dots\}$$



Backpack III优化

九章算法

- 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)

$$f[i][w] = \max\{f[i-1][w], f[i-1][w-A_{i-1}] + V_{i-1}, f[i-1][w-2A_{i-1}] + 2V_{i-1}, \dots\}$$

用前 i 种物品拼出重量 w 时最大总价值

用前 $i-1$ 种物品拼出重量 w 时最大总价值

用前 $i-1$ 种物品拼出重量 $w-A_{i-1}$ 时最大总价值，加上第 i 个物品



$$f[i][w] = \max\{f[i-1][w], f[i][w-A_{i-1}] + V_{i-1}\}$$

用前 i 种物品拼出重量 w 时最大总价值

用前 $i-1$ 种物品拼出重量 w 时最大总价值

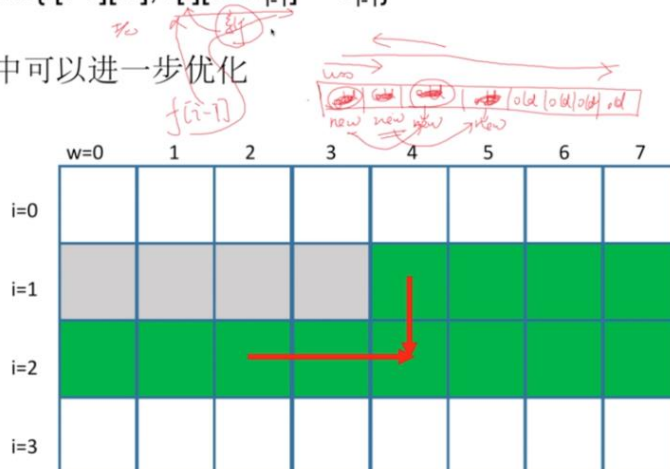
用前 i 种物品拼出重量 $w-A_{i-1}$ 时最大总价值，加上第 i 个物品

动态规划组成部分四：计算顺序

九章算法

- $f[i][w] = \max\{f[i-1][w], f[i][w-A_{i-1}] + V_{i-1}\}$

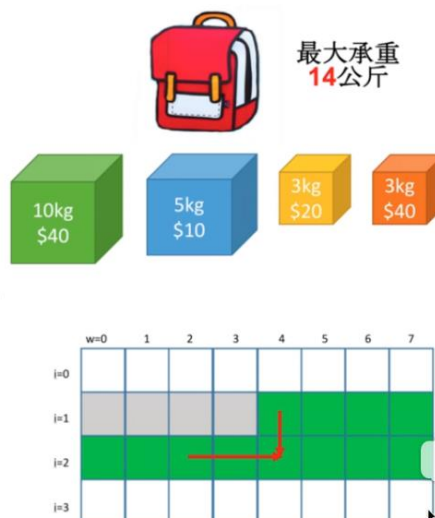
- 实际编程中可以进一步优化



小结

九章算法

- Backpack 可行性背包
 - 题面：要求不超过Target时能拼出的最大重量
 - 记录 $f[i][w]$ =前i个物品能不能拼出重量w
- Backpack V, Backpack VI, 计数型背包
 - 题面：要求有多少种方式拼出重量Target
 - 记录 $f[i][w]$ =前i个物品有多少种方式拼出重量w
- Backpack II, Backpack III, 最值型背包
 - 题面：要求能拼出的最大价值
 - 记录 $f[i][w]$ =前i个/种物品拼出重量w能得到的最大价值
- 关键点
 - 最后一步
 - 最后一个背包内的物品是哪个
 - 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



区间型动态规划


九章算法

- 给定一个序列/字符串，进行一些操作
- 最后一步会将序列/字符串去头/去尾
- 剩下的会是一个区间 $[i, j]$
- 状态自然定义为 $f[i][j]$ ，表示面对子序列 $[i, \dots, j]$ 时的最优性质

区间型动态规划

区间型动态规划

九章算法

- 给定一个序列/字符串，进行一些操作
- 最后一步会将序列/字符串去头/去尾
- 剩下的会是一个区间 $[i, j]$ 
- 状态自然定义为 $f[i][j]$ ，表示面对子序列 $[i, \dots, j]$ 时的最优性质



LintCode 667 Longest Palindromic Subsequence 九章算法

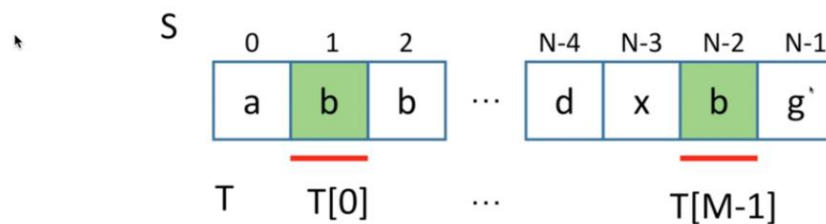
- 题意：
- 给定一个字符串S，长度是N
- 找到它最长的回文子序列的长度

- 例子：
- 输入：
 - “bbbab”
- 输出：
 - 4 (“bbbb”)

动态规划组成部分一：确定状态



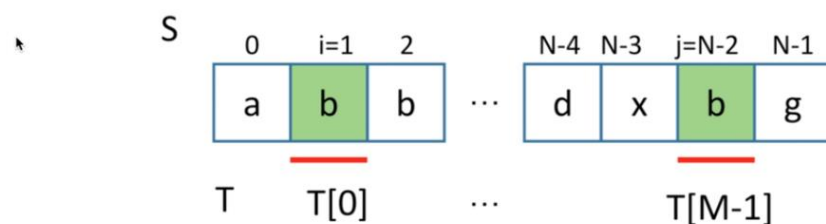
- 最优策略产生最长的回文子串T，长度是M
- 情况1：回文串长度是1，即一个字母
- 情况2：回文串长度大于1，那么必定有 $T[0]=T[M-1]$



动态规划组成部分一：确定状态

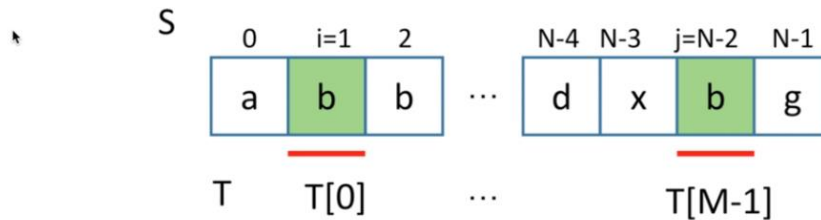


- 设 $T[0]$ 是 $S[i]$, $T[M-1]$ 是 $S[j]$
- T剩下的部分 $T[1..M-2]$ 仍然是一个回文串，而且是 $S[i+1..j-1]$ 的最长回文子串



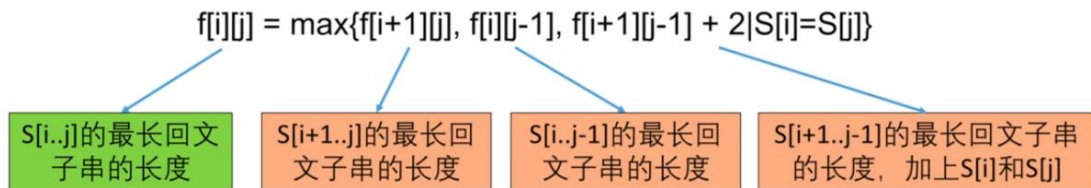
子问题

- 要求 $S[i..j]$ 的最长回文子串
- 如果 $S[i]=S[j]$, 需要知道 $S[i+1..j-1]$ 的最长回文子串
- 否则答案是 $S[i+1..j]$ 的最长回文子串或 $S[i..j-1]$ 的最长回文子串
- 子问题
- 状态：设 $f[i][j]$ 为 $S[i..j]$ 的最长回文子串的长度



动态规划组成部分二：转移方程

- 设 $f[i][j]$ 为 $S[i..j]$ 的最长回文子串的长度



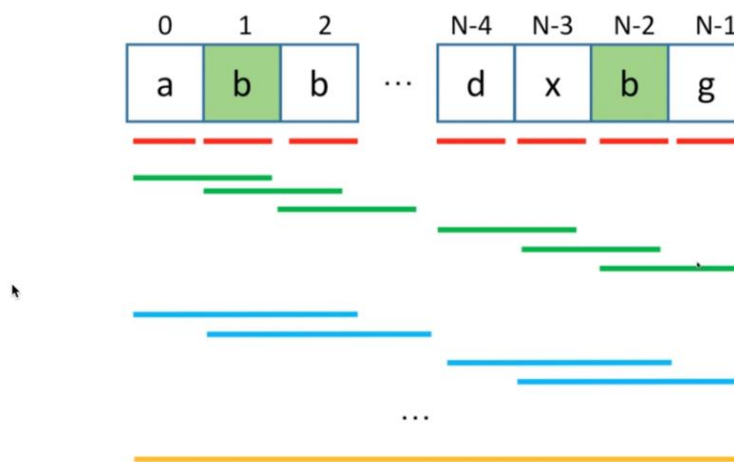
动态规划组成部分三：初始条件和边界情况

- 设 $f[i][j]$ 为 $S[i..j]$ 的最长回文子串的长度
- $f[i][j] = \max\{f[i+1][j], f[i][j-1], f[i+1][j-1] + 2[S[i]=S[j]]\}$
- 初始条件
 - $f[0][0] = f[1][1] = \dots = f[N-1][N-1] = 1$
 - 一个字母也是一个长度为1的回文串
 - 如果 $S[i] == S[i+1]$, $f[i][i+1] = 2$
 - 如果 $S[i] != S[i+1]$, $f[i][i+1] = 1$

动态规划组成部分四：计算顺序

- 设 $f[i][j]$ 为 $S[i..j]$ 的最长回文子串的长度
- $f[i][j] = \max\{f[i+1][j], f[i][j-1], f[i+1][j-1] + 2|S[i]=S[j]|\}$
- 不能按照 i 的顺序去算
- 区间动态规划：按照长度 $j-i$ 从小到大的顺序去算

动态规划组成部分四：计算顺序



动态规划组成部分四：计算顺序

- 长度1 : $f[0][0], f[1][1], f[2][2], \dots, f[N-1][N-1]$
- 长度2 : $f[0][1], \dots, f[N-2][N-1]$
- ...
- 长度N : $f[0][N-1]$
- 答案是 $f[0][N-1]$
- 时间复杂度 $O(N^2)$, 空间复杂度 $O(N^2)$



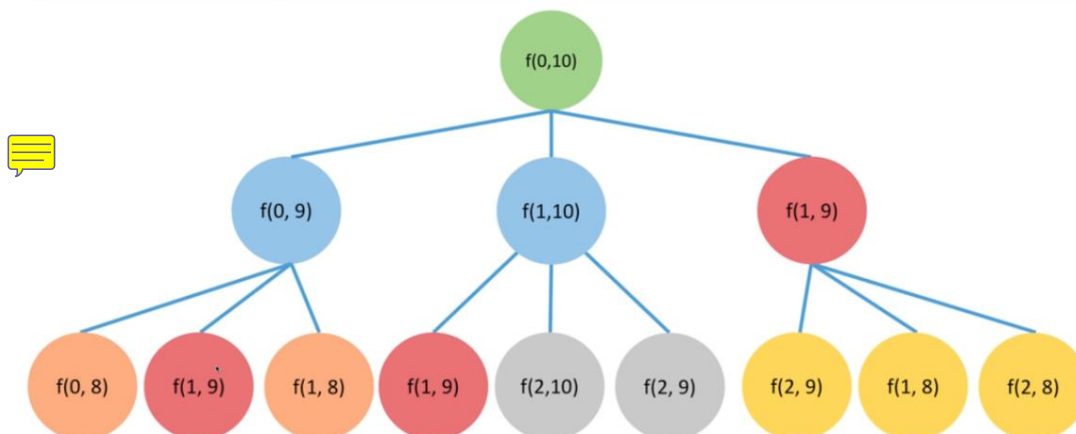
记忆化搜索方法

九章算法

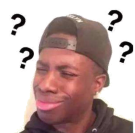
- 动态规划编程的另一个选择
- $f[i][j] = \max\{f[i+1][j], f[i][j-1], f[i+1][j-1] + 2|S[i]=S[j]\}$
- 计算 $f(0, N-1)$
- 递归计算 $f(1, N-1), f(0, N-2), f(1, N-2)$
- **记忆化**: 计算 $f(i, j)$ 结束后, 将结果保存在数组 $f[i][j]$ 里, 下次如果需要再次计算 $f(i, j)$, 直接返回 $f[i][j]$

记忆化搜索方法

九章算法



- 递推方法自下而上： $f[0], f[1], \dots, f[N]$
- 记忆化方法自上而下： $f(N), f(N-1), \dots$
- 记忆化搜索编写一般比较简单
- 递推方法在某些条件下可以做空间优化，记忆化搜索则必须存储所有 f 值



LintCode 396 Coins In A Line III

- 题意：
- 给定一个序列 $a[0], a[1], \dots, a[N-1]$
- 两个玩家Alice和Bob轮流取数
- 每个人每次只能取第一个数或最后一个数
- 双方都用最优策略，使得自己的数字和尽量比对手大
- 问先手是否必胜
 - 如果数字和一样，也算先手胜
- 例子：
- 输入： $[1, 5, 233, 7]$
- 输出：**True**（先手取走1，无论后手取哪个，先手都能取走233）

博弈

- 这题是一道博弈题，目标是让自己拿到的数字之和不比对手小
- 设己方数字和是 A ，对手数字和是 B ，即目标是 $A \geq B$
- 等价于 $A - B \geq 0$
- 也就是说，如果Alice和Bob都存着自己的数字和与对手的数字和之差，分别记为 $S_A = A - B$ ， $S_B = B - A$
- 则Alice的目标是最大化 S_A ，Bob的目标是最大化 S_B

动态规划组成部分一：确定状态

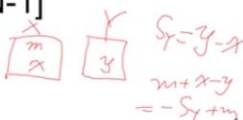
九章算法

- 如果Alice第一步取走 $a[0]$, Bob面对 $a[1..N-1]$
- Bob的最大数字差是 S_Y
- Alice的数字差是 $a[0]-S_Y$
- 如果Alice第一步取走 $a[N-1]$, Bob面对 $a[0..N-2]$
- Bob的最大数字差是 S'_Y
- Alice的数字差是 $a[N-1]-S'_Y$
- Alice选择较大的数字差

博弈子问题

九章算法

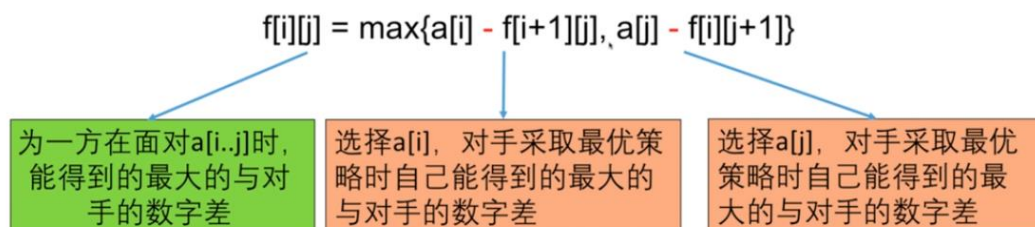
- 当Bob面对 $a[1..N-1]$, 他这时是先手
- 他的目标同样是最大化先手（自己）和后手（Alice）的数字差
- 但是此时的数字少了一个： $a[1..N-1]$
- 子问题
- 状态：设 $f[i][j]$ 为一方先手在面对 $a[i..j]$ 这些数字时，能得到的最大的与对手的数字差

$$S_x = -S_y + m$$

$$S_y = y - z$$
$$m + x - y = -S_y + m$$

动态规划组成部分二：转移方程

九章算法

- 设 $f[i][j]$ 为一方在面对 $a[i..j]$ 这些数字时，能得到的最大的与对手的数字差



动态规划组成部分三：初始条件与边界情况

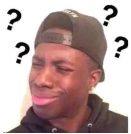


- 设 $f[i][j]$ 为一方在面对 $a[i..j]$ 这些数字时，能得到的最大的与对手的数字差
- $f[i][j] = \max\{a[i] - f[i+1][j], a[j] - f[i][j+1]\}$
- 只有一个数字 $a[i]$ 时，己方得 $a[i]$ 分，对手0分，数字差为 $a[i]$
– $f[i][i] = a[i] \ (i=0, 1, \dots, N-1)$

动态规划组成部分四：计算顺序



- 长度1： $f[0][0], f[1][1], f[2][2], \dots, f[N-1][N-1]$
- 长度2： $f[0][1], \dots, f[N-2][N-1]$
- ...
- 长度N： $f[0][N-1]$
- 如果 $f[0][N-1] \geq 0$ ，先手Alice必赢，否则必输
- 时间复杂度 $O(N^2)$ ，空间复杂度 $O(N^2)$

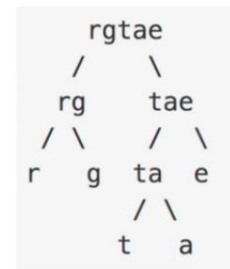
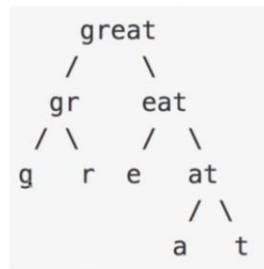


LintCode 430 Scramble String

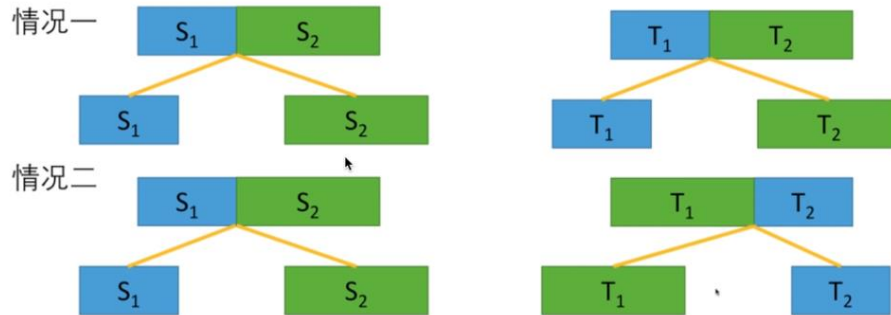


- 题意：
- 给定一个字符串S，按照树结构每次二分成左右两个部分，直至单个字符
- 在树上某些节点交换左右儿子，可以形成新的字符串
- 判断一个字符串T是否由S经过这样的变换而成

- 例子：
- 输入：S="great" T="rgtae"
- 输出：True



- 显然， T 如果长度和 S 不一样，那么肯定不能由 S 变换而来
- 如果 T 是 S 变换而来的，并且我们知道 S 最上层二分被分成 $S=S_1S_2$ ，那么一定有：
 - T 也有两部分 $T=T_1T_2$ ， T_1 是 S_1 变换而来的， T_2 是 S_2 变换而来的
 - T 也有两部分 $T=T_1T_2$ ， T_1 是 S_2 变换而来的， T_2 是 S_1 变换而来的



子问题

- 要求 T 是否由 S 变换而来
- 需要知道 T_1 是否由 S_1 变换而来的， T_2 是否由 S_2 变换而来
- 需要知道 T_1 是否由 S_2 变换而来的， T_2 是否由 S_1 变换而来
- S_1, S_2, T_1, T_2 长度更短
- 子问题
- 状态： $f[i][j][k][h]$ 表示 $T[k..h]$ 是否由 $S[i..j]$ 变换而来

动态规划组成部分一：确定状态

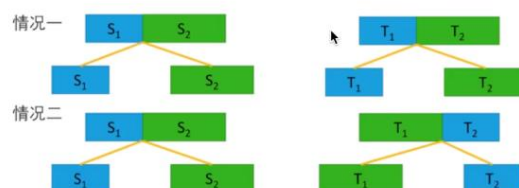
- 这里所有串都是S和T的子串，且长度一样
- 所以每个串都可以用(起始位置, 长度)表示
- 例如：
 - S_1 长度是5，在S中位置7开始
 - T_1 长度是5，在T中位置0开始
 - 可以用 $f[7][0][5] = \text{True/False}$ 表示 S_1 能否通过变换成为 T_1
- 状态：设 $f[i][j][k]$ 表示 S_1 能否通过变换成为 T_1
 - S_1 为S从字符i开始的长度为k的子串
 - T_1 为T从字符j开始的长度为k的子串

动态规划组成部分二：转移方程

九章算法

- 设 $f[i][j][k]$ 表示 S_1 能否通过变换成为 T_1
 - S_1 为S从字符i开始的长度为k的子串
 - T_1 为T从字符j开始的长度为k的子串

$$f[i][j][k] = \text{OR}_{1 \leq w \leq k-1} \{f[i][j][w] \text{ AND } f[i+w][j+w][k-w]\} \\ \text{OR} \\ \text{OR}_{1 \leq w \leq k-1} \{f[i][j+k-w][w] \text{ AND } f[i+w][j][k-w]\}$$



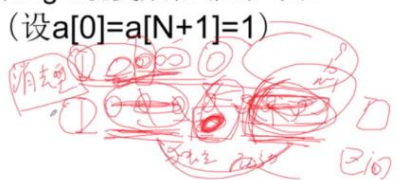
动态规划组成部分三：初始条件和边界情况

九章算法

- 设 $f[i][j][k]$ 表示 S_1 能否通过变换成为 T_1
 - S_1 为S从字符i开始的长度为k的子串
 - T_1 为T从字符j开始的长度为k的子串
- 如果 $S[i] = T[j]$, $f[i][j][1] = \text{True}$, 否则 $f[i][j][1] = \text{False}$

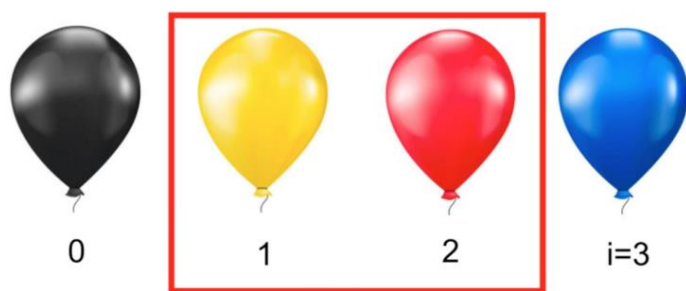
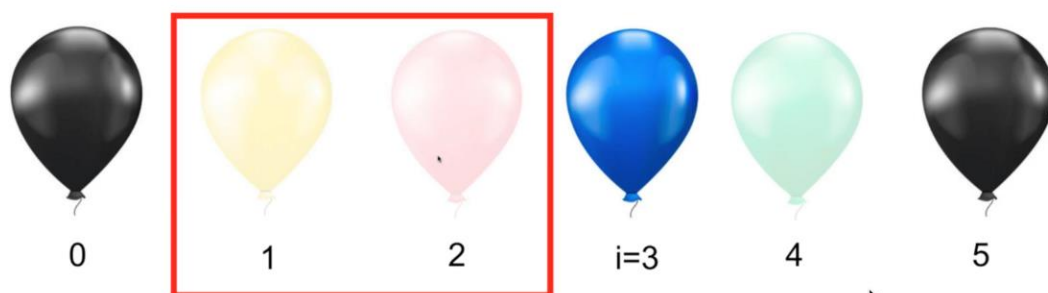
- 设 $f[i][j][k]$ 表示 S_i 能否通过变换成为 T_j
 - S_i 为 S 从字符 i 开始的长度为 k 的子串
 - T_j 为 T 从字符 j 开始的长度为 k 的子串
- 按照 k 从小到大的顺序进行计算
 - $f[i][j][1]$, $0 \leq i < N$, $0 \leq j < N$
 - $f[i][j][2]$, $0 \leq i < N-1$, $0 \leq j < N-1$
 - ...
 - $f[0][0][N]$
- 答案是 $f[0][0][N]$
- 时间复杂度 $O(N^4)$, 空间复杂度 $O(N^3)$

LintCode 168 Burst Balloons

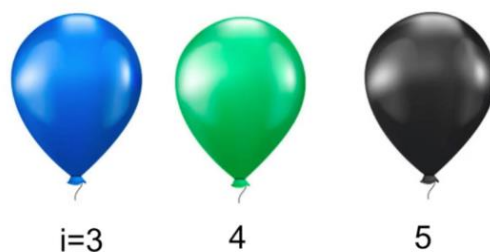
- 题意：
 - 给定 N 个气球，每个气球上都标有一个数字： a_1, a_2, \dots, a_N
 - 要求扎破所有气球，扎破第 i 个气球可以获得 $a[\text{left}] * a[i] * a[\text{right}]$ 枚金币
 - left 和 right 是与 i 相邻的下标
 - 扎破气球 i 以后， left 和 right 就变成相邻的气球
 - 求最多获得的金币数（设 $a[0]=a[N+1]=1$ ）
- 
- 例子：
 - 输入： $[3, 1, 5, 8]$
 - 输出：167
 - $[3, 1, 5, 8] \rightarrow [3, 5, 8] \rightarrow [3, 8] \rightarrow [8] \rightarrow []$
 - 金币 $3*1*5 + 3*5*8 + 1*3*8 + 1*8*1 = 167$

动态规划组成部分一：确定状态

- 所有 N 个气球都被扎破
- 最后一步：一定有最后一个被扎破的气球，编号是 i
- 扎破 i 时，左边是气球 0 ，右边是气球 $N+1$ ，获得金币 $1 * a_i * 1 = a_i$
- 此时气球 $1 \sim i-1$ 以及 $i+1 \sim N$ 已经被扎破，并且已经获得对应金币



扎破1~ $i-1$ 号气球最多获得的金币数

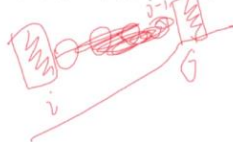


扎破 $i+1$ ~ N 号气球最多获得的金币数

子问题

九章算法

- 要求扎破1~N号气球，最多获得的金币数
- 需要知道扎破1~i-1号气球，最多获得的金币数和扎破i+1~N号气球，最多获得的金币数
- 子问题
- 状态：设 $f[i][j]$ 为扎破i+1~j-1号气球，最多获得的金币数



动态规划组成部分二：转移方程

九章算法

- 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

$$f[i][j] = \max_{i < k < j} \{f[i][k] + f[k][j] + a[i] * a[k] * a[j]\}$$

扎破i+1~j-1号气球
最多获得的金币数

扎破i+1~k-1号气球
最多获得的金币数

扎破k+1~j-1号气球
最多获得的金币数

最后扎破k号气球
获得的金币数



动态规划组成部分三：初始条件和边界情况

九章算法

- 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
- $f[i][j] = \max_{i < k < j} \{f[i][k] + f[k][j] + a[i] * a[k] * a[j]\}$
- 初始条件： $f[0][1] = f[1][2] = \dots = f[N][N+1] = 0$
- 当没有气球要扎破时，最多获得0枚金币

- 设 $f[i][j]$ 为扎破 $i+1 \sim j-1$ 号气球，最多获得的金币数
 - i 和 j 不能扎破
- $f[i][j] = \max_{i < k < j} \{f[i][k] + f[k][j] + a[i] * a[k] * a[j]\}$
- 区间动态规划：按照长度 $j-i$ 从小到大的顺序去算
 - $f[0][1], f[1][2], f[2][3], \dots, f[N][N+1]$
 - $f[0][2], f[1][3], f[2][4], \dots, f[N-1][N+1]$
 - ...
 - $f[0][N+1]$
- 时间复杂度 $O(N^3)$ ，空间复杂度 $O(N^2)$

总结

- 背包型动态规划
 - 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
- 区间型动态规划
 - 状态用区间左右端点： $f[i][j]$
 - 记忆化搜索