1.LIS 返回最长递归子序列具体元素

1.算法1

- 递归关系：$L(k) = \begin{cases} 1 & \text{if } k = 1 \\ \max\{1, \max_{1 \le i \le k-1} \{L(i) + 1 \mid s[k] > s[i]\}\} & \text{if } k > 1 \end{cases}$
- $O(n)$个子问题
  - 计算$L(k)$的时间复杂度：$O(k)$
- 时间复杂度：$O(n^2)$

存在当s[k]>s[i]时，找到最大的L(i)满足此条件，更新L(k) = L(i) +s[k].

否则取len = 1，即 L(k) = s[k]

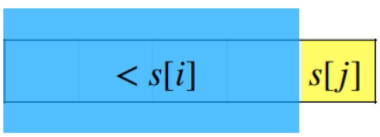
时间复杂度不变

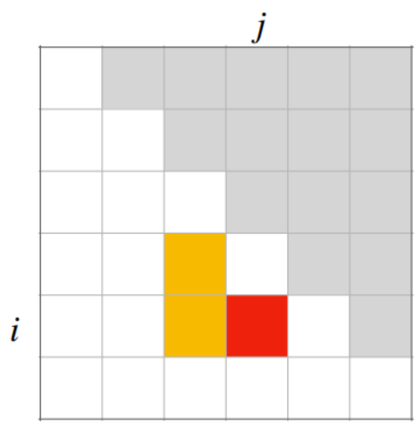
空间复杂度：L(1)-L(n),共记录n个数组

每个数组与n成常数关系,即O(n)
所以时间复杂度 O(n^2)

2.算法2

- 考虑后缀：令$L(i,j)$表示$s[j..n]$中每个元素都大于$s[i]$的LIS的长度
- 考虑前缀：令$L(i,j)$表示$s[1..j]$中*********************的LIS的长度
  - 令$L(i,j)$表示$s[1..j]$中每个元素都小于$s[i]$的LIS的长度
- 令$s[n+1] = \infty$，原问题即求解$L(n+1,n)$
- 基本情况：如果$j = 0$，那么$L(i,j) = 0$
- 归纳步骤
  - 如果$s[i] \le s[j]$，$L(i,j) = L(i,j-1)$
  - 否则，$L(i,j) = \max\{L(i,j-1), 1 + L(j,j-1)\}$
- $O(n^2)$个子问题：时空复杂度均为$O(n^2)$





$$\text{递归关系：} L(i,j) = \begin{cases} 0 & \text{if } j = 0 \\ L(i,j-1) & \text{if } s[i] \le s[j] \\ \max \begin{cases} L(i,j-1) \\ 1 + L(j,j-1) \end{cases} & \text{otherwise} \end{cases}$$

改写

if j = 0,L(i,j) = [] (即空数组)

if s[i] < s[j] L(i,j) = L(i,j-1)

otherwise L(i,j) = max(L(i,j-1) , s[j] + L(j,j-1)) 这里max表示取两个数组长的

时间复杂度不变

空间复杂度:

(n+1)*(n+1)的数组,每个元素为一个O(n)大小的数组

所以 O(n^3)

算法3

- 令$L(k)$表示$s[1..n]$中以$s[k]$结尾的LIS，原问题即为求解 $\max\limits_{1\le k\le n} L(k)$
- 递归关系：$L(k) = \begin{cases} 1 & \text{if } k = 1 \\ \max\{1, \max_{1\le i\le k-1} \{L(i)+1 \mid s[k] > s[i]\}\} & \text{if } k > 1 \end{cases}$
- 虽然只有$O(n)$个子问题，但时间复杂度是$O(n^2)$
  - 计算$L(k)$需要$O(n)$时间：遍历i来找最大的满足条件的$L(i)$：$s[i] > s[k]$
  - 能否更快？比如$O(\log n)$
    - $\Rightarrow O(n \log n)$

- 考虑计算$L[6]$
  - 长度为2：[3,6]和[1,3]
  - 长度为1：[8]、[3]和[1]

$s[1..n]$	8	3	6	1	3	5	4	7
$L(k)$	1	1	2	1	2	3	3	4

对于长度为k的IS，只需记住末尾元素最小的那个

$O(\log n)$时间计算$L(k)$

- 令$L(k)$表示$s[1..n]$中长度为k且末尾元素最小的递增子序列，且$L(k).last$表示该序列中最后那个元素
- 引理：$L(1).last < L(2).last < \cdots < L(k).last$
  - 假设$x \geq y$，而$y > z$，所以$x > z$
  - 那么灰色元素构成一个长度为k且末尾元素最小的递增子序列，矛盾

$L(k-1)$	…	…	x	
$L(k)$	…	…	z	y

- 归纳假设：对长度小于n的序列，可以计算其所有的$L(k)$，并有序存储
- 基本情况：长度为1的序列，那么$L[1] \leftarrow s[1]$
- 考虑如何基于归纳假设求解$s[1..n]$的所有的$L(k)$
  - 在$L(k).last$构成的有序数组中查找插入位置k'，使得$s[n]$加入后仍然有序
  - 如果$k' = k + 1$，那么$L(k+1) \leftarrow L(k) + 1$且$L(k+1).last \leftarrow s[n]$
  - 否则$L(k').last \leftarrow s[n]$，但$L(k')$的值不变
  - 时间复杂度：$O(\log n)$

升序 →

L(1).last	L(2).last	…	L(k).last

用一个数组 d[i]， 表示长度为 iii 的最长上升子序列的末尾元素的最小值

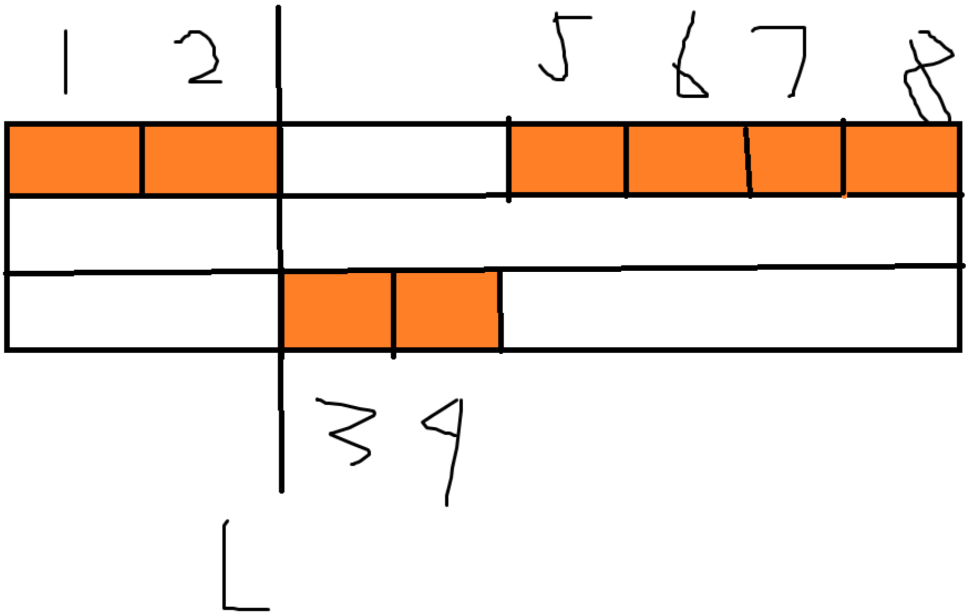
所以最后返回的就是d

时间复杂度不变

空间复杂度:只有数组d所以为O(n)

2.

因为时3 * n,给定的骨牌是1 * 2,所以考虑行数3,每两列必定有一个骨牌是横着摆的,且一定在第一列或者第三列



后一个方块可以和前一个方块在同一列,如1,2;

也可以不在同一列,如2,3;

每个交错放置的(不在同一列)的方块会使得一定存在交界线L,可以将大问题划分为两个子问题.

先假设第一个方块在第一列,然后枚举后面每一个方块是否在同一列
剩余的方块会成为一个2 * n的子问题,而2 * n 的方块划分是一个斐波那契数列.

所有的划分构成最后的答案.

3.

因为要最大化,所以只考虑一列中出现2个石块的情况

1. 一列中可以出现 T1(1,3); T2(1,4); T3(2,4); 存在石块的情况
2. 根据F(n-1)计算F(n),当F(n-1) 为T1是,F(n)为T3;当F(n-1) 为T3是,F(n)为T1. 当F(n-1)为T2时,F(n)无解.

所以最后的答案一定是T1 T3交替出现.