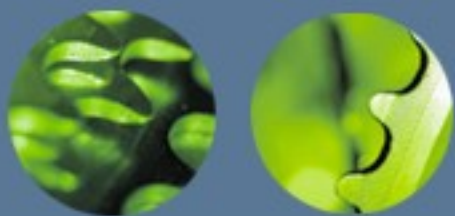




正难则反 —— 浅谈逆向思维在解题中的应用

绍兴市第一中学
唐文斌

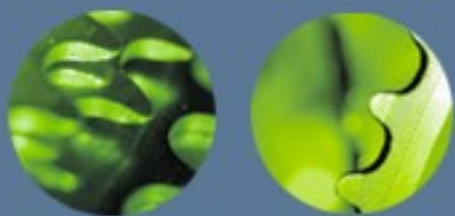


引入

有一排路灯，一共八盏，均关闭。要求打开其中三盏，没有任何两盏相邻，有多少种不同的方式。

如果直接考虑三盏打开的灯，需要讨论！

不妨来考虑没有被打开的那些灯。



引入

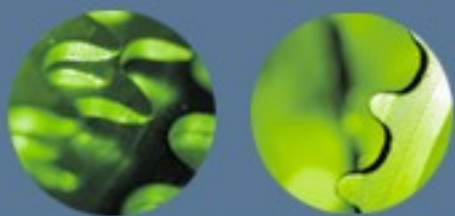
要开 3 盏灯，则有 5 盏是关闭的



两盏相邻的关闭的灯之间只能插入一盏开着的灯

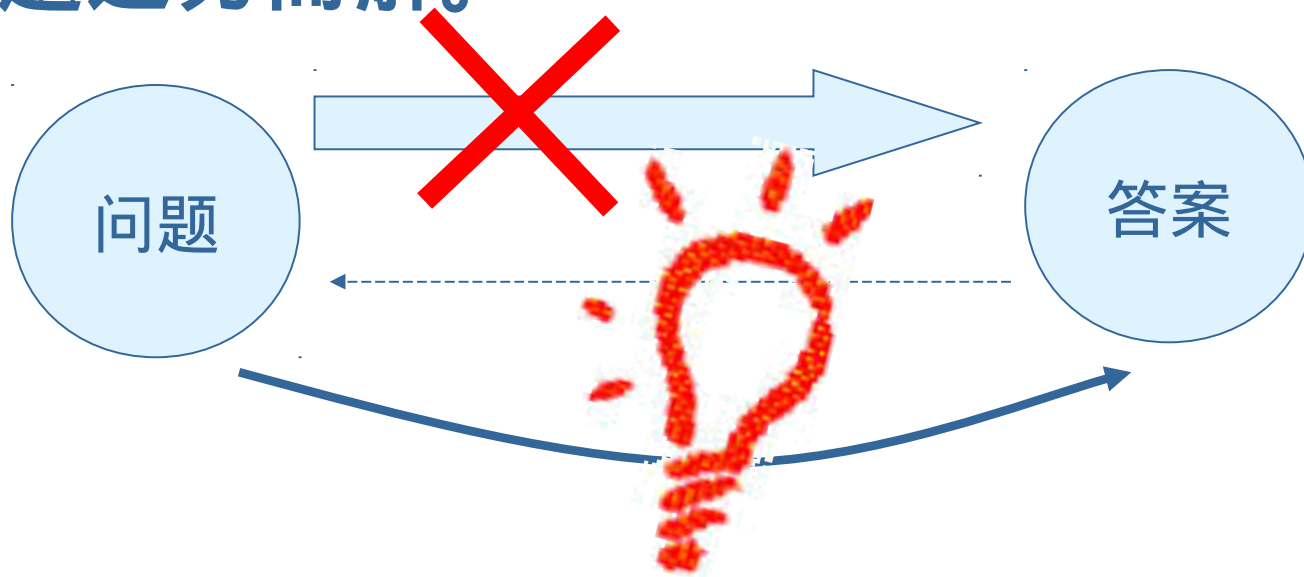
等价于在六个可选位置中选三个插入开着的灯
所以答案为 $C(6,3)$

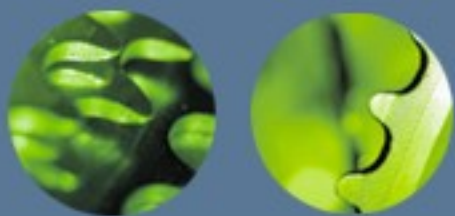
逆向思维



引入

逆向思维是一种思考问题的方式，它有悖于通常人们的习惯，而正是这一特点，使得许多靠正常思维不能或是难于解决的问题迎刃而解。





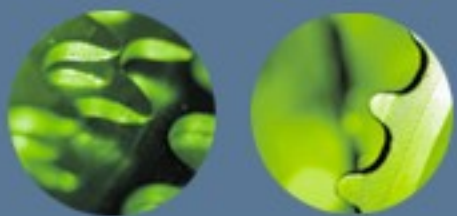
例一、 Dinner Is Ready

[题目描述]

M 根骨头分给 n 个孩子，第 i 个孩子有两个参数 Min_i 和 Max_i ，表示第 i 个孩子至少要得到 Min_i 根骨头，至多得到 Max_i 根骨头。

给出 $n(0 < n \leq 8)$ ， $M(0 < M)$ 以及 Min_i 和 Max_i
($0 \leq Min_i \leq Max_i \leq M$)

计算有多少种分配方案（骨头不能浪费，必须都分给孩子们）



例一、Dinner Is Ready

实例：

$N=3$ $M=7$

3 组可行方案

$\text{Min}_1=1$, $\text{Max}_1=2$

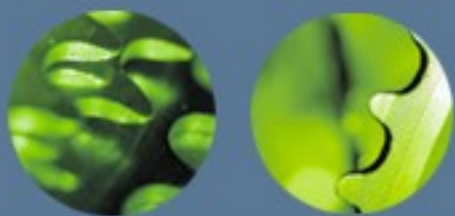
1 1 2

$\text{Min}_2=2$, $\text{Max}_2=4$

2 3 2

$\text{Min}_3=3$, $\text{Max}_3=6$

4 3 3



例一、Dinner Is Ready

该题模型即求如下方程组的整数解的个数：

则方程组变为 $X_1 + X_2 + \dots + X_n = M$

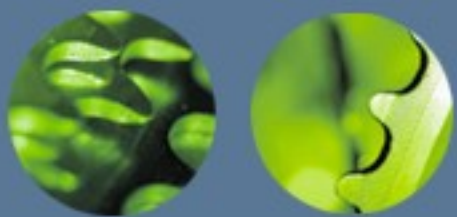
$$\left\{ \begin{array}{l} Y_1 + \text{Min}_1 \leq X_1 \leq \text{Max}_1 \\ 0 \leq Y_1 \leq \text{Max}_1 - \text{Min}_1 \\ \dots \\ \text{Min}_n \leq X_n \leq \text{Max}_n \\ 0 \leq Y_n \leq \text{Max}_n - \text{Min}_n \end{array} \right. \quad \left\{ \begin{array}{l} X_1 + X_2 + \dots + X_n = M \\ X_i \geq 0 (1 \leq i \leq n) \end{array} \right.$$

对于方程组的简单形式

下界我们可以通过换元得到简单形式 $C(M-n+1, n-1)$

但是上界仍然无法解决！

设 $Y_i = X_i + \text{Min}_i$



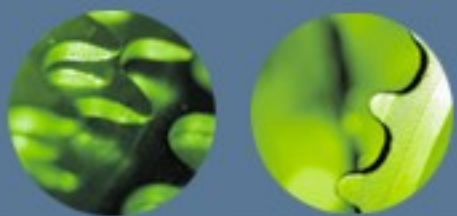
例一、Dinner Is Ready

设 S 为全集，表示满足 $X_i \geq \text{Min}_i$ 的整数解集。

设 S_i 为 S 中满足约束条件 $X_i \leq \text{Max}_i$ 的整数解集， $\overline{S_i}$ 为 S_i 在 S 中的补集，即满足 $X_i > \text{Max}_i$ ($X_i \geq \text{Max}_i + 1$)

$|S_i|$ 无法计算，但是， $|\overline{S_i}|$ 可解！！！！

那么，我们是否可以通过可解的 $|\overline{S_i}|$ 从而得到无法计算的 $|S_i|$ 呢？



例一、Dinner Is Ready

于是，我们就得到了：

$$\begin{aligned} \text{Answer} &= |S_1 \cap S_2 \cap S_3 \dots \cap S_n| \\ &= |S| - (|\overline{S_1}| + |\overline{S_2}| + \dots + |\overline{S_n}|) \\ &\quad + (|\overline{S_1}| \cap |\overline{S_2}| + |\overline{S_1}| \cap |\overline{S_3}| + \dots + |\overline{S_{n-1}}| \cap |\overline{S_n}|) - \dots \\ &\quad + (-1)^n * |\overline{S_1} \cap \overline{S_1} \cap \dots \cap \overline{S_n}| \end{aligned}$$

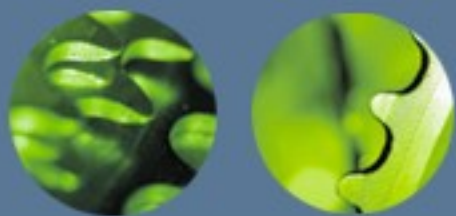


例一、Dinner Is Ready

至此，问题已经被解决。

在原集合 S_i 的模 $|S_i|$ 不可解的情况下，我们通过可解的 $|S_i|$ 得到了一个基于容斥原理的算法。

时间复杂度为 $O(2^n \cdot (n+M))$



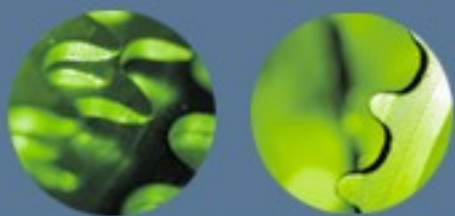
例二、 Greedy Path

[题目描述]

给定一个有向图 $G=(V,E)$

对于 $e \in E$, 有参数 C_e 和 T_e , 分别表示该条边的费用与时间。

求一个回路, 使得回路中费用总和与时间总和的比值最大。

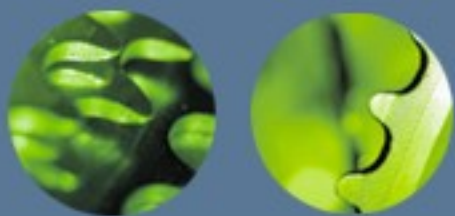


例二、 Greedy Path

题目是求一条回路，但不是边权和最大或者最小，不能直接使用经典算法，似乎无从下手。

我们的目标是找一条回路 $C=(V',E')$ ，使得 $F(C)$ 最大：

$$F(C) = \frac{\sum_{e \in E'} C_e}{\sum_{e \in E'} T_e}$$

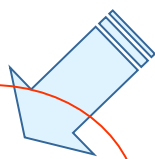


例二、 Greedy Path

设 S 为 G 中所有回路组成的集合。

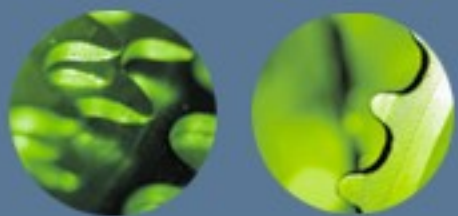
假定 $C^*=(V^*,E^*) \in S$ 就是我们要求的最优回路

$$F(C^*) = \frac{\sum_{e \in E^*} C_e}{\sum_{e \in E^*} T_e}$$


$$\sum_{e \in E^*} C_e - F(C^*) \times \sum_{e \in E^*} T_e = 0$$

定义函数 $O(t)$:

$$O(t) = \max_{C=(V',E') \in S} \sum_{e \in E'} C_e - t^* \sum_{e \in E'} T_e$$



例二、 Greedy Path

我们做一个猜想：

如果有 $o(t^*)=0$ ，也就是存在 $C^* = (V^*, E^*) \in S$ 满足：

$$t^* = \frac{\sum_{e \in E^*} C_e}{\sum_{e \in E^*} T_e}$$

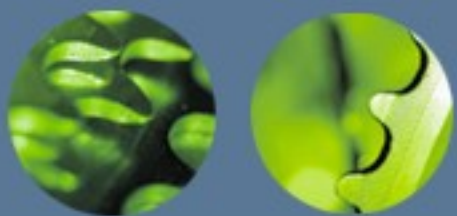
我们认为 C^* 就是最优回路！

证明：如果存在另一条回路 $C_1 = (V_1, E_1) \in S$ 更优

则

$$t_1 = \frac{\sum_{e \in E_1} C_e}{\sum_{e \in E_1} T_e} > t^* \Leftrightarrow 0 < \sum_{e \in E_1} C_e - t^* (\sum_{e \in E_1} T_e) \leq o(t^*)$$

与 $o(t^*)=0$ 矛盾



例二、 Greedy Path

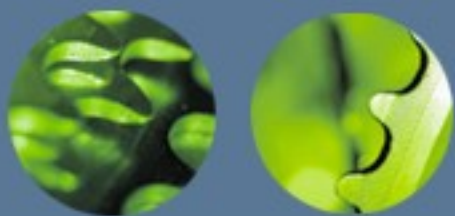
更进一步，可以发现：

如果 t^* 是最优答案，则存在：

$$\begin{cases} o(t) = 0 \Leftrightarrow t = t^* \\ o(t) < 0 \Leftrightarrow t > t^* \\ o(t) > 0 \Leftrightarrow t < t^* \end{cases}$$

根据该性质，便得到了算法：

我们只要从一个包含 t^* 的区间 (t_l, t_h) 开始，不断地二分，计算 $o((t_l + t_h)/2)$ ，得到新的上下界，直到达到精度要求。时间复杂度为 $O(\lg K * N^3)$

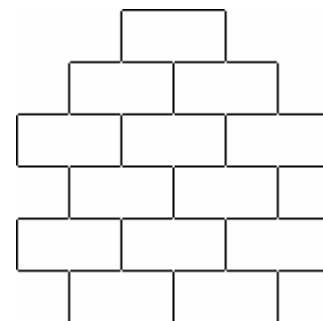


例三、 Building Towers

[题目描述]

用 N 块积木来搭建 H 层的塔 (不一定要用完所有的积木)

要求相邻的两层积木数量相差为 1
最底层积木数量为 M

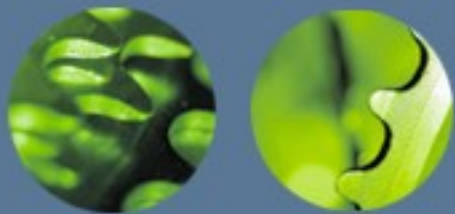


($N \leq 32767$, $H \leq 60$, $M \leq 10$)

($H=6$, $M=2$, $N=13$)

任务 1 : 可以搭建成多少种不同的塔 ?

任务 2 : 用 H 个整数从底到顶表示一个塔的结构 , 问字典序第 K 小的塔是什么 ?



例三、 Building Towers

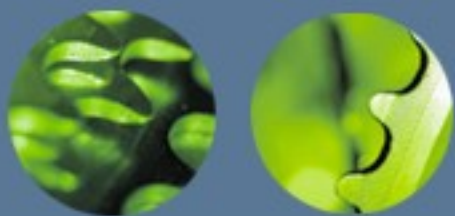
动态规划算法

用 $F[i,j,k]$ 表示当前层有 i 块砖，还剩下 j 层，可用的砖块数量为 k 的方案总数

动态规划方程：

$$F[i,j,k] = F[i+1,j+1,k-i] + F[i-1,j+1,k-i]$$

边界条件： $F[M,H,N]=1$ ，其他为 0



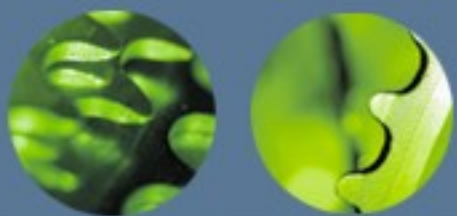
例三、 Building Towers

容易看出，这个动态规划方程一共涉及了 $N*M*K$ 个状态，最大约为 $60*70*2400 = 10M$ 。

如果每个状态用一个 double 来保存信息，则至少需要 80M 的内存

不论是时间复杂度，还是空间复杂度，都让人难以接受。

失败



例三、 Building Towers

自底向上的动态规划算法

自顶向下的搜索算法



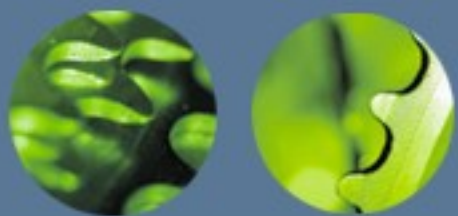
记忆化因子



逆向动态规划

将规划的顺序作一个反转，有什么好处呢？



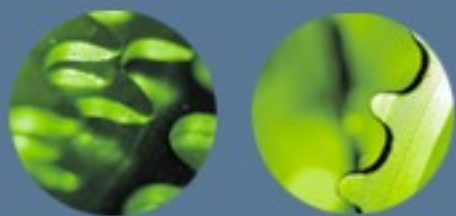


例三、 Building Towers

逆向动态规划的好处在于：

1) 自顶向下的看问题，有“一览众山小”的开阔视野，可以顺利地规划如何搜，先搜什么后搜什么，以及如何剪枝等作合理的布局。

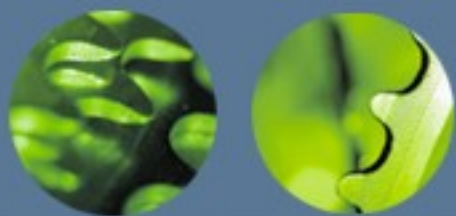
2) 在类似于本题的计数问题中，可以采用部分记忆化的方法，即只记录那些比较容易被多次搜索到的状态。这样一来，减少了存储的状态量，加快了查询的速度。



例三、 Building Towers

让我们来看看逆向动态规划的精彩表现：

N	H	M	正向动态规划 所需要处理的 状态量	逆向动态规划 中存储的状态 量	正逆向动态 规划状态数 比值
1000	40	10	1200000	3055	392.80
1500	50	10	2625000	5371	488.74
1200	60	10	2880000	49875	57.74
1500	60	10	3600000	38018	94.69

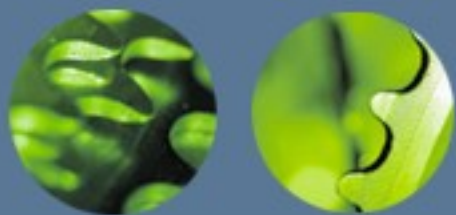


总结

例一 补集转化思想

例二 参变量法

例三 逆向动态规划



总结

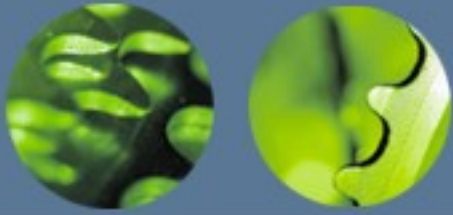
打破思维定势

以退为进

避其锋芒

攻其软肋

反弹琵琶成新曲



Thank You!