1 Introduction

1.1 数字三角形

“拉”法：$dp[i,j] = \max\{dp[i-1,j-1], dp[i-1,j]\} + value[i,j]$
“推”法：

$$dp[i,j]+value[i+1,j] \to dp[i+1,j] dp[i,j]+value[i+1,j+1] \to dp[i+1,j+1]$$

1.1.1 DP 与图论的关系

迷宫问题，求解 S-T 的最短路。

考虑搜索的方案。

TODO FIX CODE

搜索算法有什么特点？第一次到达某个点，就是最优的解。

尝试改进搜索的算法。

拓扑排序。以图论的思想考虑这个问题，会发现 DP 方程与图上最短路问题有些类似。每一个点看做具有一个点权，最大化得到的点权值（以状态转移过程方向连边）。与拓扑排序相近。

如果这张图不能被拓扑排序，则说明状态设计错了（如果不能理解，参见 CLRS 上动态规划一章状态转移图是拓扑图的解释）。

单源最短路问题（伪代码采用 py 的语法高亮）：

```
global ans = infinity
procedure Search(pos, temp):
    if pos == T:
        ans = max(ans, temp)
    else:
        for(pos, next_pos, cost) in graph:
            Search(next_pos, temp + cost)
```

事实上就是把所有的状态画到一张纸上，然后按照状态转移的方向连成一张图，发现是一个拓扑图。然后需要按照拓扑图上的方向进行转移，即是一个动态规划的思路。

拓扑排序：所有在到达某个点前到达的点被排在这个点的前面。充要条件：图是一个有向无环图（DAG—Directed Acyclic Graph）。树上 DFS 时产生的 DFS 序就是一个拓扑序。时间复杂度 $O(V+E)$.

迷宫问题显然不能拓扑排序。但是仍然可以使用 BFS 等方法完成。能否使用类似的方法优化这个问题？

就是所有的决策都是由内圈扩展到外圈的过程，就可以使用 DP。当边权都为正的时候，就能由内圈扩展到外圈（明显地，边权不可能变小，即不可能出现经过某些额外的节点可以使得边权更小），按照到 S 的距离使用堆排序，Dijkstra 算法（就是一个贪心的思想，如果不能理解，参见 CLRS 图算法最短路径部分）。

以上的问题统一称为线性规划问题（包括最短路问题），请参见 CLRS，我不会。

遇到负边权怎么办？Dijkstra 算法不能使用了。可能需要重新设计一下状态：

1. Bellman-Ford 算法 dp[i,x] 表示最多经过 x 个点，到达点 i 所需要的最短路径长度。则有以下状态转移方程：

$$dp[i,x] = \min\{dp[i-1],x,min_p\{dp[i-1,p]+cost(p,x)\}\}$$

2. Floyd 算法

1.2 状态设计

核心是最优子结构，或者可以称为“封闭子结构”，到达一个状态的方法，和之后的决策是已经无关的。

距离，TSP 问题（旅行商问题，可以参考 CLRS）。

1.2.1 矩形嵌套问题

设计状态发现，只需要考虑最后一个矩形是 i 时，最多已经排了多少个矩形：

```
function Search(i):
    ans = 1
    for j = 1 to n:
        if CanInlay(j,i):
```

1.2.2 计数问题

骨牌覆盖问题：有一个 2 行 n 列的长方形网格，要求用 n 个 1*2 的骨牌铺满，求覆盖方案。

是否存在一种交错的方案？可以证明不可能出现交错的情况。

考虑以下做法：

假设第 i 个位置是纵向骨牌，那么有以下递推式：$f(n) = f(0)f(n-1) + f(1)f(n-2) + \cdots + f(n-1)f(0)$

这个做法是错误的。问题如下：

1. 没有纵向的骨牌
2. 枚举纵向骨牌出现在何处，有些方案（出现两个纵向骨牌）会被计算两次
   - 考虑最早枚举的纵向骨牌出现的位置
   - 假设第 i 个位置是第一个纵向骨牌，他的左边全部都是横向骨牌
   - 分两种情况讨论：
     - 偶数：$f(n) = f(n-1) + f(n-3) + f(n-5) + \cdots + f(1)$
     - 奇数：$f(n) = f(n-1) + f(n-3) + f(n-5) + \cdots + f(0)$

1.2.3 凸多边形划分问题

解法 1 ：考虑 v_1v_n 这条边所在的三角形的位置，则有：

$$f(n) = f(2)f(n-1) + f(3)f(n-2) + \cdots + f(n-1)f(2)$$

每次把多边形划分成两个凸多边形。解法 2 ：考虑 v_1v_k 这条对角线，假设连了 v_1v_k 这条对角线，则会发现一边是 $n-k$ 的多边形，另一侧是一个 $n-k+2$ 的多边形。

$$f(3)f(n-1) + f(4) + f(n-2) + \cdots + f(n-1)f(3)$$

每一个点出发的对角线都有上述的效果，所以枚举每一个点出发的每一条对角线，共计有 n 个点：

$$n[f(3)f(n-1) + f(4) + f(n-2) + \cdots + f(n-1)f(3)]$$

每一条对角线的效果会被计算两次（明显地，枚举每一个点，v_1v_k 这条对角线会在 v_1 这个点被枚举一次，在 v_k 这个点又被枚举一次）：

$$f(n) = n[f(3)f(n-1) + f(4) + f(n-2) + \cdots + f(n-1)f(3)]/[2(n-3)]$$
$$=> f(n+1) = f(n) * (4n-6)/n$$

这是一种计数问题中非常常见的处理方法，我们发现每种情况被算了多次，那么就除以这种情况被算的次数即可（类似排列组合的公式的推导）。

以上技巧称为 “Double-counting”，以上递推公式就是 Catelan 数的定义。

常见应用还有括号序列，给定一个长度，问在这个长度下有多少个合法的括号序列。

出栈顺序、乘法方案数等等。