2022.11.7

每日一题816 模糊坐标

这个题注意题意说能还原为一个二维坐标！！也就是能且只能加一个逗号。那就是二分的思想。

首先枚举能加逗号的位置，将字符串分成两半，然后再对两边进行处理加逗号能有几种合法情况，最后两边组合即可。

```
class Solution {
public:
    vector<string> getPos(string s) {
        vector<string> pos;
        if (s[0] != '0' || s == "0") pos.push_back(s);
        for (int p = 1; p < s.size(); ++p) {
            if ((p != 1 && s[0] == '0') || s.back() == '0') continue;
            pos.push_back(s.substr(0, p) + "." + s.substr(p));
        }
        return pos;
    }
    vector<string> ambiguousCoordinates(string s) {
        int n = s.size() - 2;
        vector<string> res;
        s = s.substr(1, s.size() - 2);
        for (int l = 1; l < n; ++l) {
            vector<string> lt = getPos(s.substr(0, l));
            if (lt.empty()) continue;
            vector<string> rt = getPos(s.substr(l));
            if (rt.empty()) continue;
            for (auto& i : lt) {
                for (auto& j : rt) {
                    res.push_back("(" + i + ", " + j + ")");
                }
            }
        }
        return res;
    }
};
```

741 摘樱桃

第一次做线性三维DP，emmm有被虐到qwq

这道题最大的问题在于，去一次又要回来，两次加起来要求能摘樱桃的最大值。如果沿袭以往的二维DP的话也就是把两个过程分开独立求解。但是这么做最大的问题在于**其破坏了最优子结构的性质**，即回来的时候的最大值跟去的时候怎么走的是强烈相关的。

为此我们确定，两边要同时考虑，也要涉及到相互的影响。那么我们就不得不同时考虑两边的情况。做出第一步变形：

有两个人从 (0,0) 出发，向下或向右走到 (N-1,N-1) 时，摘到的樱桃个数之和的最大值。

整理下状态和谁相关：首先肯定和步数有关，另外又和两个人分别的位置有关，因此设计出三维DP的数组：

$DP[i][j][k]$ 表示两个人都走了 i 步以后，第一个人在第 j 行而第二个人在第 k 行时能拿到的最多樱桃数。为什么不把每个人的列数设计进去呢，因为列数可以由走的步数减去所在行数来确定，不是一个自由度。

现在考虑如何转移：对于 $DP[i][j][k]$ 而言，每个人有两种移动方向，因此有四种转移形式：

$$DP[i][j][k] = max \begin{cases} DP[i-1][j][k] + cost & \text{两个人均向右走} \\ DP[i-1][j-1][k] + cost & \text{第一个人向下走，第二个人向右走} \\ DP[i-1][j][k-1] + cost & \text{第一个人向右走，第二个人向下走} \\ DP[i-1][j-1][k-1] + cost & \text{两个人均向下走} \end{cases}$$

其中的 $cost$ 指两个人按照当前走法走之后能新摘到的樱桃数。

最后的答案即为 $max\{dp[2n-1][n-1][n-1], 0\}$ 因为要处理无法到达的情况。代码如下：

```
class Solution {
public:
    int cherryPickup(vector<vector<int>>& grid) {
        int dp[102][52][52];
        int n = grid.size();
        for(int i = 0; i <= 100; i++)
            for(int j = 0; j <= 50; j++)
                for(int k = 0; k <= 50; k++)
                    dp[i][j][k] = INT_MIN;   //初始化

        dp[0][0][0] = grid[0][0];
        for(int i = 1; i <= 2 * n - 2; ++i)
            for(int x1 = max(i-n+1,0); x1 <= min(n-1,i); ++x1) {
                int y1 = i - x1;
                if(grid[x1][y1] == -1) continue;

                for(int x2 = x1; x2 <= min(n-1,i); ++x2) {
                    int y2 = i - x2;
                    if(grid[x2][y2] == -1) continue;

                    int res = dp[i-1][x1][x2];
                    if(x1) res = max(res,dp[i-1][x1-1][x2]);
                    if(x2) res = max(res,dp[i-1][x1][x2-1]);
                    if(x1 && x2) res = max(res,dp[i-1][x1-1][x2-1]);
                    //对应四种情况加和

                    res += grid[x1][y1];
                    if(x1 != x2) res += grid[x2][y2];  //如果两个人现在走到了相同位置则只需加一次
                    dp[i][x1][x2] = res;
                }
            }
        return max(dp[2*n-2][n-1][n-1],0);
    }
};
```

有些细节还需要说明一下：

- 关于 $x1$ 遍历时的范围选择问题

```
for(int x1 = max(i-n+1,0); x1 <= min(n-1,i); ++x1)  // i为当前走的步数
```

i 的取值是 $[0, 2n-2]$，当 $i \leq n-1$ 时第一个人所在的行数可以任取，从0到任意位置均可开始。但是一旦 $i > n-1$ 时不可能任取，因为其行数加列数必须等于步数的限制，导致当其列数为最大 $(n-1)$ 时有最小行数 $i-(n-1)$，综上所述其开始的位置可以写作 $max(i-n+1, 0)$。结束位置同理，当$i < n-1$ 时无论如何最远边界是 i，可写作 $min(n-1, i)$。

- 关于 $x2$ 遍历时的范围选择问题

```
for(int x2 = x1; x2 <= min(n-1,i); ++x2)
```

结束位置好理解，为什么起始位置跟上面不一样呢？起始一样也是可以的，只不过算了两次没必要。设当前第一个人的位置为 (x_1, y_1) 而第二个人位置为 (x_2, y_2) 会发现，光比较所在的行数，两个位置始终是一上一下的情况(在同一行除外)，那么到底谁在上谁在下呢起始并不重要，我们关心的是这个状态下的最大樱桃数。故可以规定 $x1$ 一定在 $x2$ 的上面或者在同一行，这样就降低了一半的时间复杂度同时还遍历到了全部情况，细节满满。

- 关于初始化的值为什么是 INT_MIN

```
for(int i = 0; i <= 100; i++)
    for(int j = 0; j <= 50; j++)
        for(int k = 0; k <= 50; k++)
            dp[i][j][k] = INT_MIN;    //初始化
```

这个其实是为了便于最后确定不存在通路的。观察代码可以发现当 $grid[i][j] = -1$ 时 DP 数组直接跳过维持初值。那么由这些等于 -1 的地方转移过来的 DP 数组会怎么样呢，有这么一步操作：

```
res += grid[x1][y1];
if(x1 != x2) res += grid[x2][y2];  //如果两个人现在走到了相同位置则只需加一次
dp[i][x1][x2] = res;
```

也就是说一定会转移的时候加上0或者1或者2，如果认为 $dp[i][j][k] < 0$ 时为不存在到达该点的通路时，那么初始值一定要足够小，使得遍历完每一次加完 $grid$ 的值到达 $dp[2n-1][n-1][n-1]$ 时一定是个负值。为此初值给一个较大的负数就很合理了。或者任意给定一个负值，每次加 $grid$ 数组值之前判断该种情况能否实现，如果不能实现直接标记为一个负值也是可以的。两种写法任取其一 即可。