PROJECT3 说明文档

518021910789 刘书畅

PART1

1. 设总金额为13，硬币={1,2,5,10}

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	2	2	3	3	4	4	5	5	6	6	7	7
5	1	1	2	2	3	4	5	6	7	8	10	11	13	14
10	1	1	2	2	3	4	5	6	7	8	11	12	15	16

2. 时间复杂度

N种硬币，总金额M

填满表格需要时间复杂度O(NM)

填一个格子需要时间O(1)

所以总的时间复杂度为

$$O(NM)$$

3. 优化空间复杂度

注意到每一行的格子的值只和上一行格子有关，因此只要保存两行即可，空间复杂度为O(M)

在我的方案中，开了两个O(M)的空间，space0[M]和space1[M],行数i%2==0是，将数据存储在space0中，并利用space1中的数据进行计算。i%2==1时恰好相反。

4. 解题思路+代码实现思路

数组coins[N]记录所含的硬币种类数，sapce(i,j)表示使用硬币coins[0]到coins[i-1]的硬币组合出数额j的组合种数。

状态转移函数如下。当数额j大于硬币coins[i]的面额时，组合出数额i的组合数为不使用该硬币时的组合数（也就是space(i,j-coins[i-1])），加上使用该硬币时(少用一个该硬币)的的组合数。

$$space[i][j] = \begin{cases} space[i-1][j] & j - coin[i-1] < 0 \\ space[i-1][j] + space[i][j-coin[i-1]] & j - coin[i-1] >= 0 \end{cases}$$

最终的答案就是space(N,M)

优化空间复杂度后不需要使用M*N的表格只需要两个space[M],同（3. 优化空间复杂度）的描述，交替使用两个数组存储。

关键代码如下:

```
for(int i=1;i<coinNum;i++)
{
    for(int a=0;a<=amount;a++)
    {
        // if i%2==0 use space0
        if(i%2==0)
        {
            if(a-coins[i-1]>=0)
                space0[a]=space1[a]+space0[a-coins[i-1]];
            else
                space0[a]=space1[a];
        } else{
            if(a-coins[i-1]>=0)
                space1[a]=space0[a]+space1[a-coins[i-1]];
            else
                space1[a]=space0[a];
        }
    }
}
```

PART2

1. 分析case3

case3中是第二个人不能顺利吃鸡。

其余每个人顺利吃鸡过程举例:

吃鸡者\决斗轮数	1	2	3	4	5
1	4-3 ->4	1-2 ->1	1-4 ->1	1-5 ->1	1-6 ->1
3	5-4 ->5	5-6 -> 5	3-2 ->3	3-1 ->3	3-5 ->3
4	5-6 ->5	3-2 ->3	3-1->3	3-5 ->3	3-4 ->4
5	3-2 ->3	3-1->3	3-4 ->4	5-4 ->5	5-6 ->5
6	5-4 ->5	3-2 ->3	3-1 ->3	3-5 ->3	3-6 ->6

2. 时间空间复杂度

时间复杂度:

$$O(n^3)$$

空间复杂度:

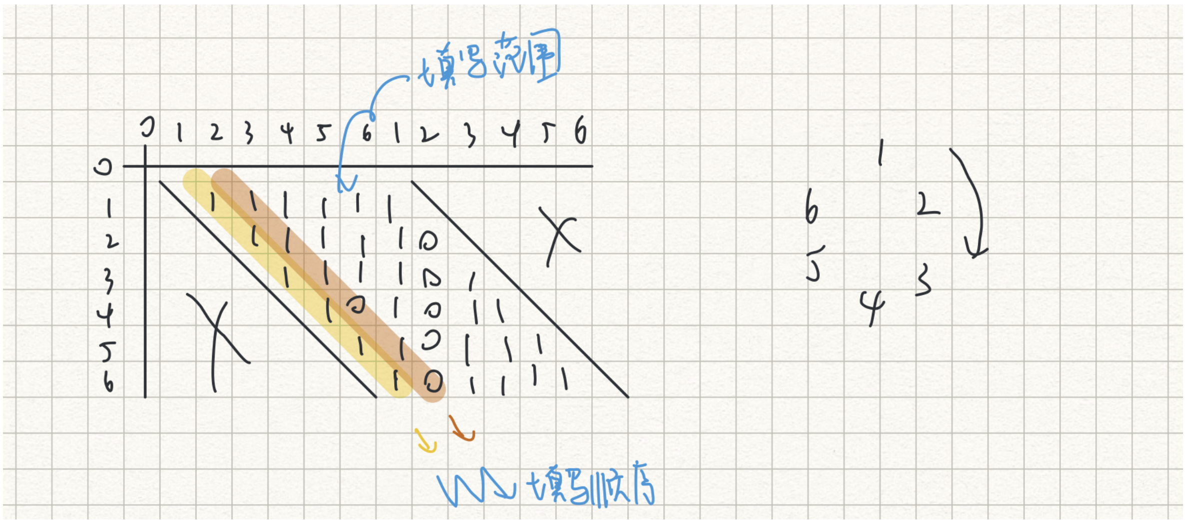
$$O(n^2)$$

暂时认为时间复杂度已经为最优。空间复杂度也为最优。因为此处填写每斜列数据时，所依赖的数据不仅是前一列，可能会使用前面所有列的数据，并且存储conquer也需要这么空间。用动态规划解此题需要填写n*n个数据,填写每个数据平均要用O(0.5n)的复杂度。

3. 解题思路+代码实现思路

一个人能否成功吃鸡取决于他能否存活过n-1轮决斗见到自己。举例分析，比如在一次六人决斗中1可以和4相见，说明23或56可以被干掉，所以1能获胜等价于1能否与自己见面。

此题是环形图，不便分析，所以将环形拆成链。以case3为例子分析：



space(i,j)为1表示按顺时针方向i能与j相见，若space(i,i+amount)为1，说明自己可以和自己相见，说明此人可以吃鸡。上图标出了表格的生成和生长方式。

$$space[i][j]=\begin{cases}1 & \exists k\in(i,j)s.t.\,space[i][k]=1space[k][j]=1\text{且}conquer[i][k]\text{或}conquer[j][k]\\0 & \text{不存在上述}k\end{cases}$$

同样以case3为例，为什么1可以和3见面呢？ 因为1可以和2见面，2可以和3见面，并且1可以打败2。所以状态转移方程如上，i可以和j见面的条件是，在i和j 之间存在一个人k ,k和i，k和j都能见面，并且i或j能打败k。(当然在代码完成过程中，k可能需要进行取模运算来真正表示这个人) 关键代码如下：

```
for(int column=row+1;column<=row+n;column++)
{
    int row2 = column > amount ? column - amount : column;
    int column2 = column > amount ? row+n-amount : row+n;

    if(space[row][column]
    &&space[row2][column2]
&&(conquer[row-1][column%amount-1]||(!conquer[row2-1][(column2)%amount-1])))
    {
        flag=true;
        space[row][row+n]=1;
        break;
    }
}
```

最后计算能成功吃鸡的人数即为：

$$sum(space[i][i+amount])i\in[1,amount]$$

PART3

1. 状态转移方程

$$space[i][j] = \begin{cases} \sum space[i+damage[i]][k]/count[k] & i!=0 \\ \sum \sum_{m=1}^{damage[i]} space[i+m][k]/count[k]+ & j-coin[i-1]>=0 \end{cases}$$

2. 矩阵

以part3-case3 在hp=2 时的方程组为特例。令space(2,0),space(2,1),space(2,4) 为x0,x1,x4

$$\begin{cases} x_0+(-\frac{1}{3})x_1+0*x_4=0.07 \\ (-\frac{1}{2})x_0+x_1+0*x_4=0.1366667 \\ 0*x_0+0*x_1+x_4=0.1366667 \end{cases}$$

增广矩阵为

$$\begin{bmatrix} 1 & -\frac{1}{3} & 0 & 0.07 \\ -\frac{1}{2} & 1 & 0 & 0.1366667 \\ 0 & 0 & 1 & 0.1366667 \end{bmatrix}$$

方程组的未知数：到达各个无陷阱点的期望概率

系数：主元素系数为1，与主元素节点不相连的节点的系数为0，与主元素相连的节点k的系数为k的度数的倒数的相反数，终点节点的系数永远是0

常数项为与主元素相连的有陷阱点k的期望概率除以k的度数，也就是从k到达主元素节点的概率

3. 时间空间复杂度

时间复杂度为：

设一共有n个路口，t个路口没有陷阱。计算每一行时填有陷阱的格子需要

$$O((n-t)*n)$$

填没有陷阱的格子需要用高斯消元法解方程组，因为抵达终点后不用再往外走，所以可以不考虑终点，时间复杂度为

$$O((t-1)^3)$$

总共有hp行所以时间复杂度为

$$O(hp*((n-t)*n)+hp*t^3)=O(hp*(t-1)^3)$$

空间复杂度为

$$O(n*hp)$$

时间复杂度的优化：

1. 高斯消元法求解过程中有一步操作就是，寻找该列最大的数最为主元素，此题中不需要这步。（因为题目的特征保证了）（已实现）
2. 考虑到有些无陷阱点不会相连，则可把一个大的有t-1个方程的方程组拆成若干个小的方程组，所以时间复杂度可以优化为

$$O((k-1)^3)(k\text{为最大的无陷阱联通集所包含的节点的个数})$$

因为次数为三次方，所以降低底数可以极大降低复杂度。

4. 解题+代码思路

创建一个hp*n的矩阵space(i,j)表示还有i滴血时走到节点j时的期望概率。表格的生长方式为：从i=hp行开始填，填到i=1行（不用计算i=0行）。填各个行时，先填有陷阱点，再列方程填无陷阱点。关键代码为：

```
for(int row=hp-1;row>0;row--)
    {
        fill_trap(row,hp,n,noTrap,trap,count,myEdges,damage,space);
        fill_noTrap(row,hp,n,noTrap,trap,count,myEdges,damage,space);
    }
```

由于填i=hp行时,有陷阱点的概率为0，起点有初始概率1，所以起点为主元素的方程的常系数要再加上1。

由于抵达终点后就不会再走出来，所以从终点来的概率永远为零，终点系数永远为0。