1

思路

解题思路

当填x元硬币所在行时，前0至$x-1$列即为前一行的同列数据，之后列的数据即为前一行同列数据加上该列$-(x-1)$列的数据，状态转移方程为

$$dp[i][j]=\begin{cases} dp[i-1][j] & i>0, j<x_j \\ dp[i-1][j]+dp[i][j-x_j] & i>0, j\geq x_j \\ 1 & i=0, j=0 \\ 0 & i=0, j>0 \end{cases}$$

代码思路

代码思路与解题思路基本相同，只是不用一个$M\times N$的数组，而是用一个$M\times 2$的数组，方法是(3)问的解答。

```
int zeroOrOne = 0;
int prev;
int now = zeroOrOne;
//begin dp
for (int k : coins) {
    int coin = k;
    prev = zeroOrOne;
    now = (zeroOrOne = !zeroOrOne);

    for (int i = 0; i < k; ++i) {
        dp[now][i] = dp[prev][i];
    }
    dp[now][coin] = 1 + dp[prev][coin];

    for (int j = coin+1; j < amount+1; ++j) {
        dp[now][j] = dp[now][j-coin]+ dp[prev][j];
    }
}
```

zeroOrOne用来区分当前填写的是第零行还是第一行，在循环中，第一个for语句遍历前0至$x-1$，第二个for语句遍历其余的。

(1)

	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

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

(2)

$O(M \times N)$

(3)

经观察，对于(1)中的表格，从第二行开始，每一行的数据填充只需要本行和前一行的数据，所以申请一个$2 \times M$的数组，当有第三行数据进入时，覆盖掉第一行的数据即可。

2

思路

解题思路

用一个 $N \times N$的布尔dp矩阵表示两个人能否有可能相邻，这样最终能吃鸡的人数即为$dp[i][i]$为1的数量。初始时N个人围一个圈，$dp[i][j] = 1$当且仅当$j - i = 0$或$i = n - 1, j = 0$,这样其余的$dp[i][j]$为1当且仅当有一个中间人k使得$dp[i][k] = 1, dp[k][j] = 1$,并且$conquer[i][k] = 1$或$conquer[j][k] = 1$,其状态转移方程(除去初始时为1的)为

$$dp[i][j] = \begin{cases}1 & 存在k,使得dp[i][k] \quad dp[k][i] \quad (conquer[i][k] || conquer[j][k]) 0 & 不满足上述条件\end{cases}$$

代码思路

```
//loop the procedure
for (int l = 2; l <= amount; ++l) {  //loop amount-1 times
    for (int i = 0; i < amount; ++i) { //row i
        bool flag = false;
        int target = (i+l) % amount;
        for (int j = 1; j < l ; ++j) {
            int k = (i+j) % amount;
            if(dp[i][k] && dp[k][target] && (conquer[i][k] || conquer[target][k])){
                flag = true;
                break;
            }
        }
        if(flag)
            dp[i][target] = 1;
    }
}
```

n-1次循环，步长从2增加到n，比如初始时dp[0][1]=1,第一次循环考察dp[0][2],第二次循环考察dp[0][3]，最后循环结束后总结dp[i][i]=1的数量，即为可能吃鸡的人数。

(1)

1不可能吃鸡

0吃鸡：

- 3 -> 2
- 0 -> 1,3,4,5

2吃鸡:

- 4 -> 3
- 0 -> 5
- 2 -> 1,0,4

3吃鸡：

- 0 -> 5,4
- 2 -> 0,1
- 3 -> 2

4吃鸡

- 2 -> 0,1
- 3 -> 2
- 4 -> 3,5

5吃鸡

- 4 -> 3
- 2 -> 4,1,0
- 5 -> 2

(2)

令$amount = N$,则时间复杂度为$O(N^3)$,空间复杂度为$O(N^2)$。
我认为不能再优化了。对于这个$N \times N$的数组，遍历填满就需$O(N^2)$的时间复杂度了。而填每一格时，要再根据步长遍历$1\ N-1$个格子，故最终时间复杂度只能为$O(N^3)$

3

思路

解题思路

从血量为hp开始，依次考虑血量为$hp-1, hp-2, \cdots, 2, 1$的情况，每层中的有陷阱结点计算思路为问(1)所示，无陷阱结点的计算思路即为高斯消元法。

代码思路

```
vector<int> eachEdge[n+1];
for (int i = 0; i < edges.size(); i+=2) {
    int edge1 = edges[i];
    int edge2 = edges[i+1];
    if(edge2 != n)
        eachEdge[edge1].push_back(edge2);
    if(edge1 != n)
        eachEdge[edge2].push_back(edge1);
}

for(int i = 0; i < n+1; i++){
    sort(eachEdge[i].begin(),eachEdge[i].end());
}
```

为每个路口创建相邻可来自路口数组，考虑到不可能从路口n回到别的路口，故不不计入路口n

```
bool toEnd(int node,vector<int>& endNote)
{
    for (int i : endNote) {
        if(node == i)
            return true;
    }
    return false;
}
```

由于在上述数组中不计入路口n，所以可能实际路口数少一，该函数用来判断是否与路口n有连接

```
for(int i = hp; i > 0; i--) {
    int rowOfGaussArray = 0; //?
    for (int j = 1; j <= n; ++j) {
        int nowDamage = damage[j - 1];
        if (nowDamage != 0) {
            if (i + nowDamage <= hp)
                for (int k = 0; k < eachEdge[j].size(); ++k) {
                    if(dp[i + nowDamage][eachEdge[j][k]]!=0)
                        dp[i][j] += dp[i + nowDamage][eachEdge[j][k]] / (double) (toEnd(eachEdge[j][k], &: eachEdge[n])?eachEdge[eachEdge[j][k]].size()+1:eachEdge[eachEdge[j][k]].size());
                }
        }
    }
```

首先计算出有陷阱结点的概率。

```
for (int j = 1; j <= n ; ++j) {
    int nowDamage = damage[j - 1];
    if (nowDamage == 0){
        rowOfGaussArray++;
        gaussArray[rowOfGaussArray][rowOfGaussArray] = 1;
        for (int k = 0; k < eachEdge[j].size(); ++k) { //找遍j的支路
            if (damage[eachEdge[j][k]-1] != 0) { //
                if(dp[i][eachEdge[j][k]]!=0) {
                    gaussArray[rowOfGaussArray][numOfEmptyTrap + 1] +=
                            dp[i][eachEdge[j][k]] /
                            (double) (toEnd(eachEdge[j][k],  &: eachEdge[n]) ? eachEdge[eachEdge[j][k]].size() + 1
                                                                      : eachEdge[eachEdge[j][k]].size());
                }
            } else {
                for (int l = 1; l < n + 1; ++l) {
                    if (emptyTrap[l - 1] == eachEdge[j][k])
                        gaussArray[rowOfGaussArray][l] = -(double) 1 / (double) (toEnd(eachEdge[j][k], &: eachEdge[n])?eachEdge[eachEdge[j][k]].size()+1:eachEdge[eachEdge[j][k]].size());
                }
            }
        }

    }
}
```

再由无陷阱结点构建出高斯消元法矩阵

```
if(i == hp) {

    gaussArray[1][numOfEmptyTrap + 1] = 1;
}

double temp; //用于记录消元时的因数
for (int q = 1; q <= numOfEmptyTrap; q++) {
    int r = q;
    for (int j = q + 1; j <= numOfEmptyTrap; j++)
        if (fabs(gaussArray[j][q]) > fabs(gaussArray[r][q]))
            r = j;
    if (r != q)
        for (int j = q; j <= numOfEmptyTrap + 1; j++)
            swap( &: gaussArray[q][j],  &: gaussArray[r][j]);//与最大主元所在行交换
    for (int j = q + 1; j <= numOfEmptyTrap; j++) {//消元
        temp = gaussArray[j][q] / gaussArray[q][q];
        for (int k = q; k <= numOfEmptyTrap + 1; k++)
            gaussArray[j][k] -= gaussArray[q][k] * temp;
    }
}
for (int q = numOfEmptyTrap; q >= 1; q--) {//回代求解
    for (int j = q + 1; j <= numOfEmptyTrap; j++)
        gaussArray[q][n + 1] -= gaussArray[q][j] * gaussArray[j][numOfEmptyTrap + 1];
    gaussArray[q][numOfEmptyTrap + 1] /= gaussArray[q][q];
}
//将计算结果代入dp中
for (int q = 0; q < numOfEmptyTrap; ++q) {
    dp[i][emptyTrap[q]] = gaussArray[q+1][numOfEmptyTrap+1];
}
```

高斯消元法用的是列主元消去法，将结果回代入dp矩阵中。
但很遗憾的是，并没有通过part3的所有case。

(1)

令任意路口l处岔路数为E_l(不包括n),设与路口i有通路的路口共m个，则

$$hp[i][j] = \sum_{q=1}^{m} hp[i + damage[j]][k_q] \times \frac{1}{E_{k_q}}$$

(2)

未知数是各个无陷阱结点的概率，对于第i行，第i个未知数的系数为1,其余结点j的系数为$-\frac{1}{E_j}$，常数项为由有陷阱结点计算得来的概率。例如对于某个具有三个无陷阱结点的图，其增广矩阵为

$$\begin{bmatrix} 1 & -\frac{1}{E_{x_2}} & -\frac{1}{E_{x_3}} & f(x_1) \\ -\frac{1}{E_{x_1}} & 1 & -\frac{1}{E_{x_3}} & f(x_2) \\ -\frac{1}{E_{x_1}} & -\frac{1}{E_{x_2}} & 1 & f(x_3) \end{bmatrix}$$

(3)

设路口数为N，血量为M，无陷阱数为E，时间复杂度为$O(M \times N + M \times E^3)$
空间复杂度为$O(M \times N + E^2)$