**Part1**

解题思路：

如文档所说，这是一个完全背包问题，动态规划中的中间变量是是从 1 开始直到总金额中每一个金额被硬币组合出的组合数。为此，构建一个 M×|N|的矩阵，用来存放中间结果。根据下面写的递归方程填满表格，最后取出dp[coins.size()][amount]就是答案了

代码实现思路：

首先对dp数组进行初始化，dp[0][i] = 0; dp[i][0] = 1

之后根据递归方程填表

最后取dp[coins.size()][amount]就是答案

问题回答

1.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 硬币/金额 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 4 |
| 5 | 1 | 1 | 2 | 2 | 3 | 4 | 5 | 6 |
| 10 | 1 | 1 | 2 | 2 | 3 | 4 | 5 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 硬币/金额 | 8 | 9 | 10 | 11 | 12 | 13 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 5 | 5 | 6 | 6 | 7 | 7 |
| 5 | 7 | 8 | 10 | 11 | 13 | 14 |
| 10 | 7 | 8 | 11 | 12 | 15 | 16 |

2.时间复杂度：O（MN）；

3.观察发现，

dp[i][j - coins[i - 1]] + dp[i - 1][j]; if (j - coins[i - 1] >= 0)

dp [i][j] =

dp[i][j] = dp[i - 1][j]; if (j - coins[i - 1] < 0)

当填dp[i][j]时，只需要第i-1行和第i行的值。故只需要保存两行即可。将空间复杂度优化到O（M）；

**Part2**

参考文档：https://www.cnblogs.com/shenben/p/5776665.html

解题思路：

构建一个amount\*amount的数组，其中meet[i][j]表示第i个人是否可以与第j个人存在pk机会，如果一个人可以与自己pk，那么他就能吃鸡

代码实现思路：

首先申请meet数组，并赋初始值，meet[i][(i+1)%amount] = 1，即每个人能和旁边的人决斗。

接着利用三层循环。第一层循环是两名决斗者之间的偏移量，第二层循环遍历每一个决斗者，有了这两层循环就能找到被决斗者，第三层循环寻找两名决斗者AB之间的其他人，判断AB是否能通过干掉中间人的方式来遇到，然后根据是否相遇填好meet表。

最后遍历meet表，找到能遇到自己的人数，即为能吃鸡的人数。

问题回答：

1. 不能吃鸡的人：第二个人，即下标序号为1的人

0号吃鸡： ① 4-5 4胜

② 1-2 2胜

③ 2-3 3胜

④ 3-4 4胜

⑤ 0-4 0胜，吃鸡

2号吃鸡： ① 4-5 4胜

② 3-4 4胜

③ 0-4 0胜

④ 0-1 0胜

⑤ 0-2 2胜，吃鸡

3号吃鸡： ① 4-5 4胜

② 4-0 0胜

③ 0-1 0胜

④ 0-2 2胜

⑤ 2-3 3胜

4号吃鸡： ① 0-5 0胜

② 0-1 0胜

③ 0-2 2胜

④ 2-3 3胜

⑤ 3-4 4胜，吃鸡

5号吃鸡： ① 2-3 2胜

② 3-4 3胜

③ 0-1 0胜

④ 0-2 2胜

⑤ 2-5 5胜，吃鸡

1. 时间复杂度：O(n3)

空间复杂度：O(n2)

在当前算法下，时间复杂度和空间复杂度都已经达到最佳了。在这个算法下，一个循环算决斗两人偏移量，一个循环选当前决斗的发起角色，一个循环找决斗的两个人之间的路径，缺一不可，故时间复杂度最优也是O（n3）了。

空间复杂度：由于填当前meet表的时候不确定当前还剩下谁，只能保留全部的meet表，不能覆盖，所以空间复杂度不能小于O（n2）

**Part3**

参考资料：

<https://www.luogu.com.cn/problem/P3232>

http://leungyukshing.cn/archives/%E9%AB%98%E6%96%AF%E6%B6%88%E5%85%83%E6%B3%95%E4%B9%8B%E8%AE%B2%E8%A7%A3%E4%B8%8E%E4%BB%A3%E7%A0%81%E5%AE%9E%E7%8E%B0.html

解题思路：

以 hp\*n 建立动态规划的矩阵是一个不 错的想法，把 f(i,j)设置为剩下 i 点血时到达 j 点的期望次数。然后写出非特殊点位的转移方程。根据转移方程，在一层中可以求出有伤害点的f，无伤害点的f可以列方程组用高斯消元解决。求出所有f后，把n号点的所有f相加就是最后结果。

代码实现思路：

首先把edges的关系转化成二维数组relation的关系矩阵表示。同时初始化各个点的度degree，以及把有伤害的点和无伤害的点分开。

使用for循环，对每一层f进行遍历（即有相同血量的点）

先对有伤害的点进行处理，直接用状态方程解出来f

再处理没伤害的点，根据状态转移方程，找出系数矩阵A，常数向量b,高斯消元。

最后把n号点对应所有血量的f相加，即最后概率

1. 非特殊点位转移方程：

特殊点：f[hp][1] = 1 ，当i+damage[j]>hp时，f[i+damage[j]][k] = 0;

1. 方程组的未知数是f[i][j],i为当前的血量，j为伤害为0的节点的标号

系数其他的与当前节点相连的、伤害为0的点的度的负倒数

常数项是所有已知伤害点的f/degree

增广矩阵即方程组系数矩阵和常数列向量矩阵合成的矩阵

1. 高斯消元的时间复杂度是O（n3）,针对每一层hp，都要进行一次高斯消元，所以总体时间复杂度是O(hp\*n3)

空间复杂度：需要申请（hp）+1\*n大小的f数组，故空间复杂度是O（n\*hp）

时间复杂度方面，我个人认为如果继续使用高斯消元法，时间复杂度应该已经达到最大值。查阅资料，发现如果使用牛顿迭代法解方程，解方程的时间复杂度可以达到O（n2），这应该是一个新的优化思路。