Project 3:动态规划

蓝浩宁518021910270

问题一

解题思路

原思路：

这是一个完全背包问题，我们可以构建一个$M \times N$的矩阵dp，用来存放中间结果，其中$dp[i][j]$表示当硬币的面值最大为$coins[i]$，总金额为j时的方案数，我们可以得到状态转移方程

$$dp(i,j)=\begin{cases}0 & \text{if } i=0\&\&j\neq 0\\ 1 & \text{if } j=0\\ \sum_{0}^{j/coins[i]} dp[i-1][j-kcoins[i]] & \text{if } i\geq 3\end{cases}$$

最后取出$dp[0][amount]$，即为答案。

改进思路：

因为利用状态转移方程计算dp时只需要用到上一层的数据，而且最终结果不要求回溯，所以可以只用一个$2 \times N$的矩阵记录数据即可，空间复杂度可以优化至$O(N)$。

代码实现思路

```
int func1(int amount, vector<int> &coins)
{
    构造并初始化dp矩阵(初值均为0)；
    dp矩阵第一行赋值为0；
    dp矩阵第一列赋值为0；

    从第二行开始，遍历每行
        从第一列开始，遍历每个元素
            利用状态转移方程，求出dp，放在dp第二行；
        end
        复制第二行dp至第一行
    end

    return dp[0][amount];
}
```

问题解答

(1) 设总金额为13，硬币$=\{1,2,5,10\}$，画出动态规划的表格并填满结果。

硬币\金额	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) 这个动态规划的时间复杂度是多少?

第一列赋值 $O(N)$，第一行赋值 $O(M)$，接下来填表三重循环 $O(NM^2)$。

所以整体时间复杂度 $O(NM^2)$。

(3) 请将这个动态规划的空间复杂度优化到 $O(M)$，并指出你是如何优化的。

因为只需要上一行的数据就可以算出下一行的数据，所以可以只用一个$M \times 2$的矩阵就可以存储所需信息,每次计算结束后覆盖上次信息即可，详见代码。

问题二

解题思路

这是一个剑客决斗问题，我们可以构建一个$N \times N$的矩阵dp，用来存放中间结果，其中$dp[i][j]$为$true$时表示i和j能相遇，为$false$表示不能相遇。我们可以得到状态转移方程

$$dp(i,j) = \begin{cases} ture & \text{if } j = (i+1)\%n \\ true & \text{if } (dp(i,k)\&\&dp(k,j))\&\&(conquer(i,k)||conquer(j,k)\&\&(i<k<j) \\ false & \text{other} \end{cases}$$

若选手能和自己相遇，则他能吃鸡。

代码实现思路

```
int func2(int amount, vector<vector<int>> &conquer)
{

    构造并初始化dp矩阵(初值为false);
    选手一定能遇见周围的人，赋值ture;

    设i为两个选手的间距，从1开始到amount遍历
        从第一个选手A开始
            若相隔i的选手B此时不能相遇
                利用状态转移方程检查是否能相遇（遍历中间选手），若能相遇赋值true;
            end
        end
    end

    用sum计算所有能遇到自己的选手，即dp[j][j]为true。
    return sum;
}
```

问题解答

(1) 请分析./lab3/test/test2.txt中case3（amount=6）究竟是哪个倒霉蛋不可能吃鸡，并写出其余每个人吃鸡的一种随机决斗过程。

A胜B	A	B	C	D	E	F
A		1	0	1	1	1
B	0		0	1	1	0
C	1	1		0	1	0
D	0	0	1		0	0
E	0	0	0	1		1
F	0	1	1	1	0	

倒霉蛋：B。

A吃鸡					
对决	C-D	A-B	A-D	A-E	A-F
淘汰	C	B	D	E	F
剩余	A-B-D-E-F-A	A-D-E-F-A	A-E-F-A	A-F	A

C吃鸡					
对决	D-E	A-F	C-E	A-C	B-C
淘汰	D	F	E	A	B
剩余	A-B-C-E-F-A	A-B-C-E-A	A-B-C-A	B-C	C

D吃鸡					
对决	D-E	A-F	C-E	A-C	B-C
淘汰	D	F	E	A	B
剩余	A-B-C-E-F-A	A-B-C-E-A	A-B-C-A	B-C	C

E吃鸡					
对决	B-C	A-C	C-D	F-D	E-F
淘汰	B	A	C	D	F
剩余	E-F-A-C-D-E	E-F-C-D-E	E-F-D-E	E-F	E

F吃鸡					
对决	B-C	A-C	D-E	C-E	C-F
淘汰	B	A	D	E	C
剩余	F-A-C-D-E-F	F-C-D-E-F	F-C-E-F	C-F	F

(2) 这个程序的时间复杂度和空间复杂度是多少？能否继续优化？你可以写下你的优化思路或者在自己的代码中实现它，或是说明你的程序时间复杂度已经达到最佳了。

时间复杂度：$O(n^3)$

空间复杂度：$O(n^2)$

时间复杂度最佳了：假设我们已知n个选手中有那些人能吃鸡，现新增加一人，我们仍需要$O(n^2)$的时间复杂度去计算他能否吃鸡，所以最佳时间复杂度是$O(n^3)$。

问题三

解题思路

我们可以构建一个$hp \times N$的矩阵f，用来存放中间结果，其中$f(i,j)$设置为到达j点时剩下i点血的概率与进入次数的乘积（算是期望吧）。设$degree[i]$为第i个点的度，$edge[i][j] == 1$表示i和j之间有通路。我们可以得到状态转移方程

$$f(i,j) = \begin{cases} \sum_1^{n-1} f[i+damage[j]][k] * edge[i][j]/degree[k] + 1 & \text{if } i = hp, j = 1 \\ \sum_1^{n-1} f[i+damage[j]][k] * edge[i][j]/degree[k] & \text{other} \end{cases}$$

（超出矩阵部分默认为0）

其中对于damage大于0的节点，可以直接计算出来；而对于damage等于0的节点，可以建立方程组，用高斯消元解出。

因为只可能到达最后一个点一次，所以最后计算每个hp到达最后一个点的概率之和即可得出答案。

代码实现思路

```
//M是矩阵，n是未知数个数
double func3(int n, int hp, vector<int> &damage, vector<int> &edges)
{
    初始化f矩阵，均赋值为0；

    从第hp层开始向下遍历
        若是有陷阱节点
            直接利用状态转移方程求值；
        end
        若是无陷阱节点
            利用状态转移方程构造方程组：
                由无陷阱节点部分得到系数；
                由有陷阱节点部分得到常数项；//此处注意hp层的初始节点的常数项特殊处理
            高斯消元法求值并赋值到f；
        end
    end

    用count累加所有hp层能到达最后一个节点概率；
    return count;
}
```

问题解答

(1) 写出这个算法的状态转移方程。

设$degree[i]$为第i个点的度，$edge[i][j] == 1$表示i和j之间有通路

$$f(i,j) = \begin{cases} \sum_1^{n-1} f[i+damage[j]][k] * edge[i][j]/degree[k] + 1 & \text{if } i = hp, j = 1 \\ \sum_1^{n-1} f[i+damage[j]][k] * edge[i][j]/degree[k] & \text{other} \end{cases}$$

(2) 对于某个特定的hp，所有无陷阱节点构成的方程组，这个方程组的未知数是什么？系数是什么？常数项又是什么？写出它的增广矩阵。PS:如果你觉得一般情况很抽象的话，可以以$part3 - case3$在$hp = 2$时的方程组为特例考虑。

未知数是此hp状态下无陷阱节点的概率，系数是其余有通路节点的度的倒数之和，常数项是有陷阱节点概率的倒数之和。

(3) 这个算法的时间复杂度和空间复杂度是多少？如果按照该文档截至目前的思路，算法的时间复杂度还有提升的空间。上一小问给了你什么启发？如果你能回答出来，你将会获得更高一点的分数；如果你能在你的程序中实现，你将获得满分。Hint:你需要从高斯消元法的时间复杂度去考虑这个问题。

时间复杂度：$O(n^3 hp)$

空间复杂度：$O(nhp)$

优化方向：在所有无陷阱节点构成的方程组中，未知数及其系数都是相同的，所以在高斯消元的时候我们可以记录消元时的比例系数，仅需计算一次，可以减少时间复杂度至$O(n^3 + n^2hp)$。

实现见代码。