Project 3 动态规划

--窦嘉伟 518021911160

**1 简介**

动态规划(dynamic programming)是[运筹学](https://baike.baidu.com/item/%E8%BF%90%E7%AD%B9%E5%AD%A6/1559" \t "https://baike.baidu.com/item/%E5%8A%A8%E6%80%81%E8%A7%84%E5%88%92/_blank)的一个分支，是求解决策过程最优化的数学方法。动态规划一般可分为线性动规，区域动规，树形动规，背包动规四类。

**2 设计思路(解题思路)**

**Part1**

先填上题述表格 N为硬币个数，M为总金额

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 2 | 2 | 3 | 3 |
| 5 | 1 | 1 | 2 | 2 | 3 | 4 |

**时间复杂度：**

首先分析一下算法用矩阵D表示，D[n][m]表示用前n种纸币凑成总金额m的组合数，令C[n]表示第n种硬币面值,有一下几种情况：

1. C[n]>m ： D[n][m]=D[n-1][m]
2. C[n]=m : D[n][m]=D[n-1][m]+1
3. C[n]<m : D[n][m]=D[n-1][m]+D[n][m-C[n]]

前两种情况很好理解，第三情况分为不用第n枚硬币和用第n枚硬币。用的话取用前n种硬币凑成m-C[n]金额的组合数加上一枚n硬币即可。

对每一个元素循环，每次循环的时间复杂度是常数级，最终时间复杂度为O（MN）

**优化空间至O（M）：**

由于每一次D[n][m]的迭代要不就是从正上一层寻找要不就是从同层已迭代的元素寻找。因此可以用一个D[M]数组来存组合数，每一次迭代有一下情况:

1. C[n]>m ： D[m]=D[m]
2. C[n]=m : D[m]=D[m]+1
3. C[n]<m : D[m]=D[m]+D[m-C[n]]

这样只需要O（M）空间即可

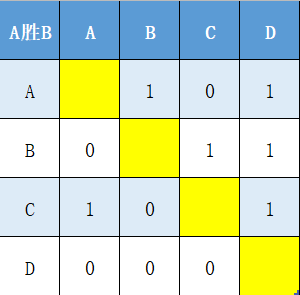
**Part 2:**

将吃鸡问题换一种说法，如果A和B能进入最终决赛圈，并且A赢，则A有可能获胜。这样我们对于A就需要取寻找这样一个B。也就是说，赛场上存在这样一个B，在其他人都被淘汰后，能与A相遇。我们可以说，B被淘汰后，A就能自己和自己决斗，赢下比赛。

这样问题转化为，在何种情况下A能和A决斗。可以理解为存在一个B，A与B能决斗，A‘与B能决斗并且A或A’能赢下B，那么A和A‘能决斗。

在看A与B能决斗的条件，就是赛场上存在一个C，A与C能决斗，B与C能决斗并且A或B能赢下C。

那可以用一个NXN的矩阵DP来存下决斗信息。DP[i][j]表示i与j能决斗。由一个简单的例子ABCDA四乘四来看，初始化矩阵如下：



我们不妨规定每个人只能和自己右边的人决斗！

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | B | C | D |
| A | 0 | 1 | 0 | 0 |
| B | 0 | 0 | 1 | 0 |
| C | 0 | 0 | 0 | 1 |
| D | 1 | 0 | 0 | 0 |

用i表示两者之间间隔人数，i从0到n-1。当i=0时，矩阵如上，相邻两者可以决斗。那么当i增加时，两者之间顺时针间隔一人P。两人可以决斗的条件就是P可以和两者都决斗（满足）且两者之一能干掉P，这样AC BD CA都置为1。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | B | C | D |
| A | 0 | 1 | 1 | 0 |
| B | 0 | 0 | 1 | 1 |
| C | 1 | 0 | 0 | 1 |
| D | 1 | 0 | 0 | 0 |

接下来i继续增加为2，那么AD之间有B，C,顺时针来看，AB，BD都可以决斗，且A能干掉B，因此AD置为1.同理，BA，CB，DC都为1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | B | C | D |
| A | 0 | 1 | 1 | 1 |
| B | 1 | 0 | 1 | 1 |
| C | 1 | 1 | 0 | 1 |
| D | 1 | 0 | 1 | 0 |

接下来i增加至3，那么对A，A‘有AB能决斗且A能干掉B。

同理BB。CC。都置为1，最终只有DD为0，因此有三人可能获胜。

**时间复杂度分析：**

在此算法中，首先对i进行外层循环n-1次，对内部决斗组合循环n次，每次内循环还需要寻找中间人k，最坏情况想每个中间人都要比较，最终时间复杂度为O（N3）

**空间复杂度分析：**

算法用一个二维数组存储决斗信息，空间复杂度O（N2）。

**空间复杂度优化：**

个人感觉，该算法的空间复杂度不可再继续优化，对于dp[i][j],每次都需要寻找k令dp[i][k]和dp[k][j]都为真当i递增时总是需要遍历全表。

**案例分析：**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **A** | **B** | **C** | **D** | **E** | **F** |
| **A** | **0** | **1** | **0** | **1** | **1** | **1** |
| **B** | **0** | **0** | **0** | **1** | **1** | **0** |
| **C** | **1** | **1** | **0** | **0** | **1** | **0** |
| **D** | **0** | **0** | **1** | **0** | **0** | **0** |
| **E** | **0** | **0** | **0** | **1** | **0** | **1** |
| **F** | **0** | **1** | **1** | **1** | **0** | **0** |

最终dp表如下

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F |
| A | 1 | 1 | 1 | 1 | 1 | 1 |
| B | 0 | 1 | 1 | 1 | 1 | 1 |
| C | 0 | 0 | 0 | 1 | 0 | 0 |
| D | 0 | 0 | 1 | 1 | 1 | 0 |
| E | 1 | 0 | 1 | 1 | 1 | 1 |
| F | 1 | 0 | 1 | 1 | 1 | 1 |

最终只有C不能获胜,下面给出吃鸡过程A-B-C-D-E-F-A：

A: BC CD DE EF AE

B: EF EA DA CA BC

D: EF AB CA DE DC

E: EF AB AC CD ED

F: AB AC CD DE FD

**Part3**

设C[n]数组存储n个路口的分叉数。假设走到一个路口X，有两种情况：

第一种，X有陷阱，伤害为x，剩余血量为HP，那么之前一定来自一个路口T，到达T的剩余血量为HP+x且TX相连。

第二种，X没有伤害，那么之前一定来自一个路口T，到达T的剩余血量也为HP

有状态方程(E[i] 表示第i个路口的出度)

f[i][j]=

对于某个特定的hp，某个无陷阱节点构成的方程，其未知数是与该节点相连的所有其他同hp节点的期望到达次数，系数是未知数节点所对应的出度分之一，常数项0为与他不相连的节点。

Part3-case3中hp为2时，有dp矩阵如下

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 |
| 1 |  |  |  |  |  |
| 2 | X1 | X2 | 0.28 | 0.2 | X5 |
| 3 | 0.32 | 0.36 | 0.8 | 0 | 0.2 |
| 4 | 1.2 | 0.6 | 0 | 0 | 0 |

有如上图未知数，有如下方程组

算法中，最外层循环对hp，内层循环（对每一个位置数循环+对每一个常数循环），每层内层循环执行高斯消元，高斯消元时间复杂度为O（n3），算法总时间复杂度为O（n5）；

算法定义dp举证和argument增广矩阵，空间复杂度为O（hpN）

该算法的时间复杂度似乎集中在高斯消元法部分,若能减少未知数的个数或者减少高斯消元的次数应该会对算法有一步优化。

基于上一问的例子发现，对于终点路口（x5），该路口未知数不包含在其他方程中，因此应该可以剔去该未知数，使得方程数量减少1。

再者，当终点路口相连的其他路口damage都不为0时，好像可以算到hp=2就可以停止，因此加一步判断或许可以减少一次高斯消元。

**3 代码分析**

在上述解题思路中已经涉及部分代码思路，这部分内容对代码功能详细说明。

**Part1**

在第一部分代码中，用数组来存储状态

int dp[amount];

然后动态更新数组值，最终返回数组最后一个元素，即题目所求值

return dp[amount-1];

**Part2**

在第二部分，用n来存人数并用dp数组来存状态，该处dp[i][j]表示i和j能进行决斗.初始化dp时将dp[m][(m+1)%n]置为1，表示相邻两人可以决斗。

for(int i=0;i<n;++i){

if(dp[i][i]==1) count++;

}

最后统计能和自己决斗的人数，即最终可能获胜的人数

**Part3**

在第三部分,用函数Gauss（）执行高斯消元操作。

int m=edges.size()/2;//边数

double dp[hp][n];//dp[i][j]表示剩余i+1点hp到达j+1点的期望次数

vector<vector<int> > conj;//记录相邻点

var;//未知数列表

vector<int> con;//常数项列表

参数列表如上，这里将终点也列为未知数项。

在循环中，对每一个未知数，分别求其对应的未知数项和常数项，得到增广矩阵的一行，存入argument矩阵中。

将argument矩阵传给Gauss函数，经过最终步骤得到期望次数。