T1 buc

题意简化：

此题拆完包装后，就是经典倒水问题。

有3个水杯，它们的容积分别是N，Q，P（N,Q,P<=1000），其中容积为N的水杯装满水。

可以从一个水杯向另外一个水杯倒水。直到这个水杯装满或是没有水可到了。

求当容积为Q的水杯内没有水时，容积为N的水杯内的水的多少的所有可能值。

## 解题思路

因为题目求的是所有可能值，所以很容易就可以想到要用搜索。

仔细思考后可以发现：每次可以有6种倒法（为了说明方便，将容积为N的水杯简称为N，容积为Q的简称Q，容积为P的简称P）

1、N倒Q   2、N倒P   3、Q倒N   4、Q倒P   5、P倒N   6、P倒Q

但这样直接搜永远都搜不完，因为它会重复许多相同的动作。

所以我们要添加一个剪枝，若搜到的状态以前搜到过，就退出。

这样，程序实现起来就很简单了。就是进行深搜。

记a[i,j]表示，Q有i单位的水，P有j单位的水这个状态有没有被搜到过。

记ans[i]表示，i这个数是不是可能值。最后进行一遍循环就可找出答案。

T2 length

算法分析要求最长的一条坡道，想到将从每个点出发的最长坡道计算出来。最后取个max即可。

那么如何计算从第i行第j列出发的最长坡道呢？

我们发现，求解该问题具有明显的子问题性质：

于是我们得到了如下递归函数：

int dfs(int i,int j) //求从[i][j]出发，最长的滑道长度

{

int ret=0; //天然的递归边界

if(i-1>0 && a[i-1][j]<a[i][j])

ret=max(ret,dfs(i-1,j)); //如果左边点比它小，找到子问题一

if(j-1>0 && a[i][j-1]<a[i][j])

ret=max(ret,dfs(i,j-1));//如果上边点比它小，找到子问题二

if(i+1<=r && a[i+1][j]<a[i][j])

ret=max(ret,dfs(i+1,j));//如果右边点比它小，找到子问题三

if(j+1<=c && a[i][j+1]<a[i][j])

ret=max(ret,dfs(i,j+1));//如果下边点比它小，找到子问题四

return ret+1;

}

仔细分析问题易得：一样会有大量的状态被重复计算！

因此考虑记忆化搜索。

将计算过的答案记录在b[i][j]中，初始化b数组为-1，即如果b[i][j]=-1表示dfs(i,j)还未被计算过，否则表示dfs(i,j)已经被计算过，可以直接return b[i][j]。

为何本题可以用-1标记而不用打vis数组呢？

因为对于坡道长度而言，-1显然是一个不可能取到的值。

于是我们有了下面程序段：

int dfs(int i,int j) //求从[i][j]出发，最长的滑道长度

{

if(b[i][j]>0) return b[i][j];//记忆化

int ret=0;

if(i-1>0 && a[i-1][j]<a[i][j]) ret=max(ret,dfs(i-1,j)); //如果左边点比它小，找到子问题一

if(j-1>0 && a[i][j-1]<a[i][j]) ret=max(ret,dfs(i,j-1));//如果上边点比它小，找到子问题二

if(i+1<=r && a[i+1][j]<a[i][j]) ret=max(ret,dfs(i+1,j));//如果右边点比它小，找到子问题三

if(j+1<=c && a[i][j+1]<a[i][j]) ret=max(ret,dfs(i,j+1));//如果下边点比它小，找到子问题四

return b[i][j]=ret+1;//记忆化

}

时间复杂度分析？

复杂度=状态数\*决策数\*转移费用

每个点出发的最长坡度只会被计算一次，每次计算可以看作是O(1)的。

因此总的时间复杂度为O(r\*c)。

T3 stone

分析：要求n个石子归并，我们根据dp的思想划分成子问题，先求出每两个合并的最小代价，然后每三个的最小代价，依次知道n个。

定义状态dp [ i ] [ j ]为从第i个石子到第j个石子的合并最小代价。

那么**dp [ i ] [ j ] = min(dp [ i ] [ k ] + dp [ k+1 ] [ j ])**

那么我们就可以从小到大依次枚举让石子合并，直到所有的石子都合并。

这个问题可以用到平行四边形优化，用一个s【i】【j】=k 表示区间 i---j 从k点分开才是最优的，这样的话我们就可以优化掉一层复杂度，变为O（n^2）.

T4 multiplication

设f[i][j]表示前i个数字加入j个乘号最大可以得到的最大值是多少，a[i][j]表示数字串里第i个到第j个字符拼接成的数

则：f[i][0] = a[1][i], f[i][j] = max{f[l][j - 1] \* a[l + 1][j]} 其中 j >= 1 1 <= l < i

答案就是f[n][k]

P. S. 高精度走起！