# 【分治算法+记忆搜索】【动态规划的优化】

## 题目描述：

312. 戳气球

难度 困难 777

有 `n` 个气球，编号为 `0` 到 `n - 1`，每个气球上都标有一个数字，这些数字存在数组 `nums` 中。

现在要求你戳破所有的气球。戳破第 `i` 个气球，你可以获得 `nums[i - 1] * nums[i] * nums[i + 1]` 枚硬币。 这里的 `i - 1` 和 `i + 1` 代表和 `i` 相邻的两个气球的序号。如果 `i - 1` 或 `i + 1` 超出了数组的边界，那么就当它是一个数字为 `1` 的气球。

求所能获得硬币的最大数量。

**示例 1：**

```
输入：nums = [3,1,5,8]
输出：167
解释：
nums = [3,1,5,8] --> [3,5,8] --> [3,8] --> [8] --> []
coins =  3*1*5    +   3*5*8   +  1*3*8  + 1*8*1 = 167
```

**示例 2：**

```
输入：nums = [1,5]
输出：10
```

# 问题分析：

1. 困难体不愧是困难题。主要是难想。
2. 这道题如果从分治的角度来看：必须采取自顶向下的思维方式：从最终结果倒推。
3. 假如下标为**k**的气球是最后一个射爆的气球，那么可以以**k**为分界线分为两个部分，（**i，k)**部分和（**k,j**）部分；两部分的和并即为（**i，j)**部分的最大值加上（**j,k)**部分的最大值，再加上**nums[k]*num[i] *num[j]**

# 分治+记忆化搜索

```
class Solution {
private:
    vector<vector<int>>memo;
    int divided(vector<int>&nums,int l,int r){
        if(r==l||r==l+1)return 0;
        if(memo[l][r]!=-1)return memo[l][r];
        for(int i=l+1;i<r;i++){
            int res=0;
            res+=nums[l]*nums[r]*nums[i];
            if(memo[l][i]!=-1)res+=memo[l][i];
            else res+=divided(nums,l,i);
            if(memo[i][r]!=-1)res+=memo[i][r];
            else res+=divided(nums,i,r);
            memo[l][r]=max(res,memo[l][r]);
        }
        return memo[l][r];
    }
public:
    int maxCoins(vector<int>& nums) {
        //自顶向下的分治算法+记忆化搜索
        //向nums的受位置和末位置插入1
        nums.push_back(1);
        nums.insert(nums.begin(),1);
        int n=nums.size();
        memo.resize(n,vector<int>(n,-1));
        return divided(nums,0,n-1);
    }
};
```

# 动态规划

```
class Solution {
public:
    int maxCoins(vector<int>& nums) {
        //动态规划
        nums.push_back(1);
        nums.insert(nums.begin(),1);
```

```
        int n=nums.size();
        vector<vector<int>>dp(n,vector<int>(n,-1));
        //初始化
        for(int i=0;i<n;i++)dp[i][i]=0;
        for(int i=0;i<n-1;i++)dp[i][i+1]=0;
        for(int dif=2;dif<n;dif++){
            for(int i=0;i<n-dif;i++){
                int j=i+dif;
                for(int k=i+1;k<j;k++){
                    dp[i][j]=max(dp[i][j],dp[i][k]+dp[k]
[j]+nums[i]*nums[k]*nums[j]);
                }
            }
        }
        return dp[0][n-1];

    }
};
```