深 圳 大 学 实 验 报 告

课程名称 算法设计与分析

项目名称 动态规划--金罐游戏问题

学　　院 计算机与软件学院

专　　业 软件工程

指导教师 卢亚辉

报 告 人 郑彦薇 学号 2020151022

实验时间 2022/5/9~2022/5/17

提交时间 2022/5/14

教务处制

一、实验目的与要求

（1）掌握动态规划算法设计思想。
（2）掌握金罐游戏问题的动态规划解法。

二、实验内容与方法

1. 问题描述

金罐游戏中有两个玩家，A 和 B，所有的金罐排成一排，每个罐子里都有一些金币，玩家可以看到每个金罐中有多少硬币。A 和 B 两个玩家交替轮流打开金罐，但是必须从一排的某一端开始挑选，玩家可以从一排罐的任一端挑选一个罐打开。 获胜者是最后拥有更多硬币的玩家。 我们是 A 玩家，问如何才能使 A 收集的硬币数量最大。

假设 B 也是按照“最佳”策略玩，并且 A 开始游戏。

2. 解决方法

给出该问题的动态规划方程，用蛮力法测试算法的正确性。给出随机生成的金罐数和金币数，统计算法实际运行时间，分析算法运行效率，并与理论效率进行对比。

三、实验步骤与过程

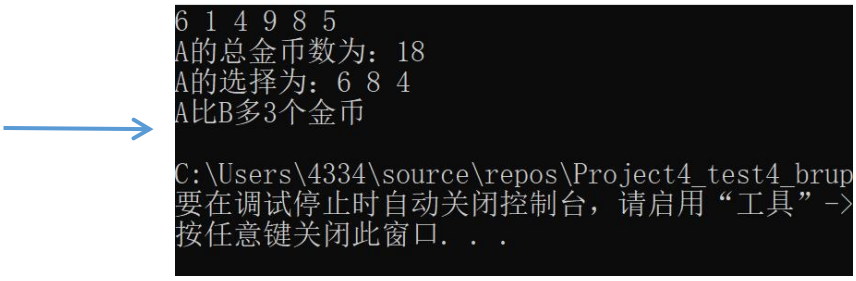
（一）用蛮力法解决问题

1. **思路：**根据规则，A 和 B 每次只能选择一排金罐中的第一个（记为 start）或最后一个（记为 end），故在 A 或 B 选择过后，剩下的金罐数一定是连续的。当 start 小于 end 时，当前玩家可以选择当前金罐的 start 或 end，然后轮到另一位玩家在剩下的金罐数中同样进行 start 或 end 的选择。两位玩家都视为“最聪明的玩家”，该过程可以使用暴力递归实现，其中 start=end 是暴力递归的结束条件。
2. **伪代码：**

```
int getScore(vector<int>& nums, int size, int start, int end, int cnt, int count)
{
    if (cnt == A) {
        if (start == end) {
            choice_A[count] = nums[start];
            return nums[start];
        }
        int temp1 = nums[start] + getScore(nums, size, start + 1, end, B, count + 1);
        //copy choice_A to tmp;
        int temp2 = nums[end] + getScore(nums, size, start, end - 1, B, count + 1);
        if (temp1 > temp2){
            //copy tmp to choice_A;
            choice_A[count] = nums[start];
            return temp1;
        }
        choice_A[count] = nums[end];
        return temp2;
    }
    if (start == end)
        return 0;
    int p1 = getScore(nums, size, start + 1, end, A, count);
    //copy choice_A to tmp;
    int p2 = getScore(nums, size, start, end - 1, A, count);
    if (p1 < p2){
        //copy tmp to choice_A;
        return p1;
    }
    return p2;
}
```

3. 正确性验证：

金罐中的金币个数	A	B
6，1，4，9，8，5	6	
1，4，9，8，5		5
1，4，9，8	8	
1，4，9		9
1，4	4	
1		1
total	18 coins	15 coins



正确。

4. 算法运行效率分析

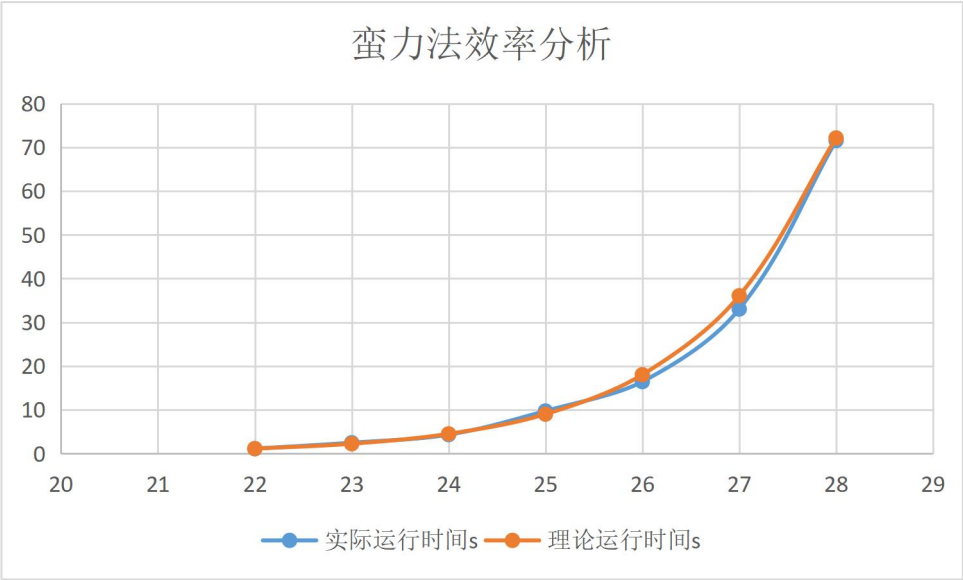
随机生成金罐的个数（N）以及每个金罐的金币数，统计算法实际运行时间，得到算法运行时间与规模关系如下所示：

规模 n	22	23	24	25	26	27	28
时间 (s)	1.127	2.48	4.33	9.73	16.44	33.03	71.53

根据蛮力递归算法的时间复杂度 $O(2^n)$，取第一个规模的时间为理论运行时间，不同规模的理论运行时间如下所示：

规模（N）	实际运行时间（s）	理论运行时间（s）
22	1.1269	1.1269
23	2.4787	2.2538
24	4.3262	4.5076
25	9.73	9.0152
26	16.4378	18.0304
27	33.0316	36.0608
28	71.5334	72.1216
…	…	…
34	>1h	4615.7824s>1h

得到算法实际效率与理论效率对比如下：



可以看到两条曲线几乎重合，只在 26~27 区间内实际运行时间略低于理论运行时间。这可能是随机生成的金币数值较小，在递归进行数据运算时所需时间较短所引发的误差。

（二）动态规划解决问题

1. 解决该问题的动态规划方程

根据上述蛮力递归思路，给出动态规划方程：

当前进行选择的玩家为 A，下一位玩家为 B，根据当前选择加上后手玩家的选择结果（因为后手玩家也是“最聪明”的玩家），决定选择当前剩余金罐中的第一个或最后一个。

$$DP\_total(nums,start,end,A,dp)$$
$$= MAX(nums[start] + DP\_total(nums,start+1,end,B,dp),$$
$$nums[end] + DP\_total(nums,start,end-1,B,dp))$$

$$DP\_total(nums,start,end,B,dp)$$
$$= MIN(nums[start] + DP\_total(nums,start+1,end,A,dp),$$
$$nums[end] + DP\_total(nums,start,end-1,B,dp))$$

2. 伪代码

求解 A 获得的金币总数：

```
DP_total(int* nums,int start,int end,int cnt,int** dp)
{
    if cnt:=A
        if dp[start][end]>=0
            return dp[start][end];//dp表记录
        if start:=end
            return nums[start];
        p1 = nums[start] + DP_total(nums,start+1,end,B,dp);
        p2 = nums[end] + DP_total(nums,start,end-1,B,dp);
        if p1>p2:
            {dp[start][end] = p1;return p1;}
        dp[start][end] = p2;return p2;
    if start:=end
        return 0;
    sum = sum(start,end);//数组从start到end的和
    q1 = DP_total(nums,start+1,end,A,dp);
    q2 = DP_total(nums,start,end-1,A,dp);
    if q1<q2:
        {dp[start][end] = sum-q1;return q1;}
    dp[start][end] = sum-q2;return q2;
}
```

回溯 dp 表求解 A 的选择：

```
i=0,j=N-1,a=0,b=0;
choice_AB[2][N];
cnt = A;
for k→0 to N-1
    if dp[i+1][j] < dp[i][j-1]
        if cnt == A
            {choice_AB[0][a++] = nums[i];cnt = B;}
        else
            {choice_AB[0][b++] = nums[i];cnt = A;}
        i++;
    else
        if cnt == A
            {choice_AB[0][a++] = nums[j];cnt = B;}
        else
            {choice_AB[0][b++] = nums[j];cnt = A;}
        j--;
```

3. 用蛮力法对小规模验证算法正确性

给出两个小规模验证：

```
金罐总数为：8
蛮力递归的结果：
A获得的金币总数→1524
A的选择→440 395 405 284

动态规划的结果：
A获得的金币总数→1524
A的选择→440 395 405 284
```

```
金罐总数为：20
蛮力递归的结果：
A获得的金币总数→55779
A的选择→9426 5647 3240 107 8737 5869 4814 3276 9287 5376

动态规划的结果：
A获得的金币总数→55779
A的选择→9426 5647 3240 107 8737 5869 4814 3276 9287 5376
```

正确。

4. 算法运行效率分析

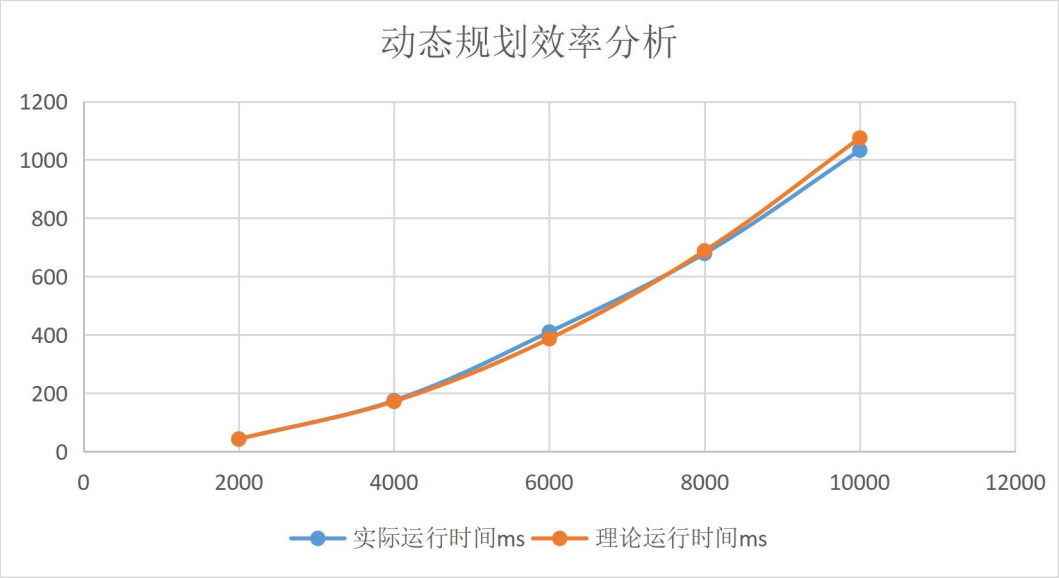
随机生成金罐的个数（N）以及每个金罐的金币数，统计算法实际运行时间，得到算法运行时间与规模关系如下所示：

规模(n)	2000	4000	6000	8000	10000
时间(ms)	43	181	441	711	1108

根据动态规划算法的时间复杂度 $O(n^2)$，取第一个规模的时间为理论运行时间，不同规模的理论运行时间如下所示：

规模（n）	实际运行时间（ms）	理论运行时间（ms）
2000	43	43
4000	181	172
6000	441	387
8000	711	688
10000	1108	1075
...	...	...
26000	6588	7267
>26000	？栈溢出	？

得到算法实际效率与理论效率对比如下：



可以看到以 ms 为单位计时时，两曲线拟合程度较高。

四、实验结论或体会

该问题主要是通过递归思路对问题进行求解，利用动态规划的填表原理进行算法的优化，将时间复杂度从 $O(2^n)$缩短至 $O(n^2)$。总体上运行时间随规模的增大而增大。在动态规划中，还利用回溯思想和求解过程中所填的 DP 表获得 A 的选择路径。

五、思考

在使用蛮力法解决该问题时，对 A 的记录比较复杂，会使得蛮力递归的实际运行时间很高。**尝试一种放弃对 A 的选择的记录，只求 A 获得金币数总和的策略：**

1) 设置两个值，存放 A 当前选择后的总和。A 选择第一个或最后一个，轮到 B 选，B 会给 A 留下一个**更差**的局面，因此可以得到如下递归公式：

$$scoreStart = nums[start] + \min(getScore(nums, start+2, end), getScore(nums, start+1, end-1))$$

$$scoreEnd = nums[end] + \min(getScore(nums, start, end-2), getScore(nums, start+1, end-1))$$

2) 可以看到，在计算两种情况 A 会获得的金币总数时，$getScore(nums, start+1, end-1)$ 重复计算。因此可以将这一部分提取出来，进一步降低运行时间。

3) 根据上述思路，得到伪代码如下：

```
int getScore_new(vector<int>& nums, int start, int end)
    gap = end - start;
    if gap == 0
        return nums[start];
    else if gap == 1
        scoreStart = nums[start];
        scoreEnd = nums[end];
    else if gap >= 2
        num = getScore_new(nums, start + 1, end - 1);
        //将原先重复计算的值提取出来
        scoreStart = nums[start] + min(getScore_new(nums, start + 2, end), num);
        scoreEnd = nums[end] + min(getScore_new(nums, start, end - 2), num);
    return max(scoreStart, scoreEnd);
```

4) 运行时间对比

规模N	优化前运行时间s	优化后运行时间s
22	1.1269	0.007
23	2.4787	0.016
24	4.3262	0.023
25	9.73	0.045
26	16.4378	0.064

可以看到在经过上述优化后，运行时间大大减少，**缺点是放弃了对 A 的选择的记录。**

指导教师批阅意见： 成绩评定： 指导教师签字： 年　　月　　日
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。