猜数字游戏问题探究

原创 赵云龙 LOA算法学习笔记 2021-01-26 23:15

本文以算法课中学习的分析求解问题的思想，对两种形式猜数字游戏进行分析与解答。本文的关键字为：分治、极小极大化、动态规划。

01 问题1

1.1 问题描述

首先我们先看第一种形式的猜数字游戏。猜数字游戏的规则如下：

每轮游戏，我都会从 1 到 n 随机选择一个数字。请你猜选出的是哪个数字。如果你猜错了，我会告诉你，你猜测的数字比我选出的数字是大了还是小了。

你可以通过调用一个预先定义好的接口 int guess(int num) 来获取猜测结果，返回值一共有 3 种可能的情况（-1，1 或 0）：

- -1：我选出的数字比你猜的数字小 pick < num
- 1：我选出的数字比你猜的数字大 pick > num
- 0：我选出的数字和你猜的数字一样。恭喜！你猜对了！pick == num

1.2 问题结构分析

本问题可以利用分治思想进行快速求解，是一个很显然的二分查找问题。每次都猜位于中间的数字，若猜中游戏结束。若没有猜中，如果猜测的数字比所选出的数字大，则去右半部分继续寻找，否则去左半部分寻找，问题分解为规模一半的子问题。

时间复杂度 $T(n) = T(\frac{n}{2}) + O(1)$，时间复杂度为 $O(logn)$。

1.3 代码实现

```
int guessNumber(int n) {
        int start = 1;
        int end = n;
        int middle = (start+end)/2;
        bool res = guess(middle);
        while(res!=0){
            cout<<middle<<","<<res<<endl;
            if(res==1){
                end = middle;
                middle = (start+end)/2;
                res =  guess(middle);
            }
            else{
                start = middle+1;
                middle = (start+end)/2;
                res =  guess(middle);
            }
        }
        return middle;
```

```
20      }
```

02 问题2

2.1 问题描述

接下来我们看另一种形式的猜数字游戏。

猜数字游戏的规则如下：

我从 1 到 n 之间选择一个数字，你来猜我选了哪个数字。每次你猜错了，我都会告诉你，我选的数字比你的大了或者小了。然而，当你猜了数字 x 并且猜错了的时候，你需要支付金额为 x 的现金。直到你猜到我选的数字，你才算赢得了这个游戏。

给定 n ≥ 1，计算你至少需要拥有多少现金才能确保你能赢得这个游戏。

2.2 问题结构分析

本题的目标是找到一个最小的现金数目，能够保证无论出现什么情况都能赢得该游戏。也就是说，本题所要选择的策略能够最小化你所面临的最差情况下的损失。这也是极小极大化的思想（MiniMax theorem）。

极小极大化思想是一种广泛应用于博弈论、人工智能等领域的决策规则。John von Neumann（冯诺依曼） 1928年就已经在思考两人零和博弈中各方该如何决策的问题中提出。在处理损失时，极小极大化策略中，博弈参与人站在自己最差情况下，考虑最大化自己的收益（或者最小化自己的损失）。极小极大化方法十分适用于两人零和博弈问题的求解，例如五子棋等棋类游戏的求解。在卜老师的课上关于线性规划的对偶问题的部分，也介绍过极小极大化思想，详细内容可以回看该部分内容。

对于该问题的求解，我们首先从最简单的情况开始看。

当**n=1**时，是最平凡的情况，此时直接猜中被选中的目标数字，不需要付钱。

当**n=2**时，有两种可能的决策，选择1或者选择2，此时显然选择1，若猜中则获胜，若未猜中，则支付1元，下次必定猜中，因此总共只需要1元即可保证能够赢得这个游戏。

当**n=3**时，问题变得有些复杂，我们无法直接看出结果。一种显然的想法是我们能不能把问题分解为规模更小的情况。依次选择1到3，所剩下的两个数字就能够以n=1或n=2的情况进行求解，我们只需要比较选1,2,3后最差情况（每次都没猜中）中最小的。

现在考虑一般情况，首先我们明确最差情况也就是每次我们都未猜中目标数字，我们想要得到在每次都猜错的条件下，能够选择的最小的损失。考虑将问题进行分解，随机选择对范围在 （ i ， j ） 的从i到j的数字 k ，问题分解为对位于k左边的对区间 （ i ， $k-1$ ） 和位于k右边的区间 （ $k+1$ ， j ） 求解。因为我们要求最差情况所要支付的现金，所以要取两个区间所要支付现金数更大的，加上在当前步猜 k 未猜中所要支付的现金数 k ，为最差情况下的现金数。对于 k 取 i 到 j 的所有最坏的情况（最大损失）求最小，就是我们想要求取的结果。

$$\text{cost}(i,j) = k + \max(\text{cost}(i,k-1), \text{cost}(k+1,j)), k = i \dots j$$

对其时间复杂度分析，该方法的复杂度为 $O(n!)$ ，是指数级别的。而我们也可以注意到该问题有许多的重复子问题，我们可以记忆化存储这些子问题的结果，避免重复计算。动态规划的完整递推表达式与代码实现如下所示：

2.3 递推表达式

设在区间 (i,j) 中猜数字，所需的保证胜利最少的现金为 $OPT(i,j)$ ，有

$$OPT(i,j) = \begin{cases} 0 & i \geq j \\ \min_{i \leq k \leq j}[k + \max(OPT(i,k-1), OPT(k+1,j))] & otherwise \end{cases}$$

2.4 代码实现

```
    bool isvist[300][300] = {false};
    int dp[300][300];
    int calculate(int low, int high){
        if(isvist[low][high]){
            return dp[low][high];
        }
        if(low>=high){
            isvist[low][high] = true;
            dp[low][high] = 0;
            return 0;
        }
        int minmoney = INT_MAX;
        for(int i=low;i<=high;i++){
            int money = i+max(calculate(i+1, high), calculate(low,i-1));
            minmoney = min(minmoney, money);
        }
        isvist[low][high] = true;
        dp[low][high] = minmoney;
        return minmoney;
    }
    int getMoneyAmount(int n) {
        calculate(1,n);
        return dp[1][n];
    }
```

递归调用会有较大的时间开销，为进一步提高速度，我们舍弃递归调用，改用迭代技术来实现递归表达式。由于子问题求解时互相依赖，我们需要先求解规模最小的子问题，然后逐步用小的子问题组合成大的子问题的解，直到迭代到最原始问题的解。代码实现如下所示：

```
int getMoneyAmount(int n) {
        int dp[300][300] = {0};
        for(int l=2;l<=n;l++){
            for(int i=1; i<=n-l+1; i++){
                int j=i+l-1;
                int minmoney = INT_MAX;
                for(int k=i; k<=j; k++){
                    int money = k + max(dp[i][k-1], dp[k+1][j]);
                    minmoney = min(minmoney, money);
                }
                dp[i][j] = minmoney;
            }
        }
        return dp[1][n];
```

```
15      }
```

2.5 时间复杂度分析

OPT表格中共有 $O(n^2)$ 个元素，每个元素求解时需要在n个决策项中进行比较，因此该问题总的时间复杂度为 $O(n^3)$ 。

2.6 算法运行过程示例

下图展示了在迭代过程中，动态规划数组进行更新的过程。该例为 $n = 5$ 时的更新情况。

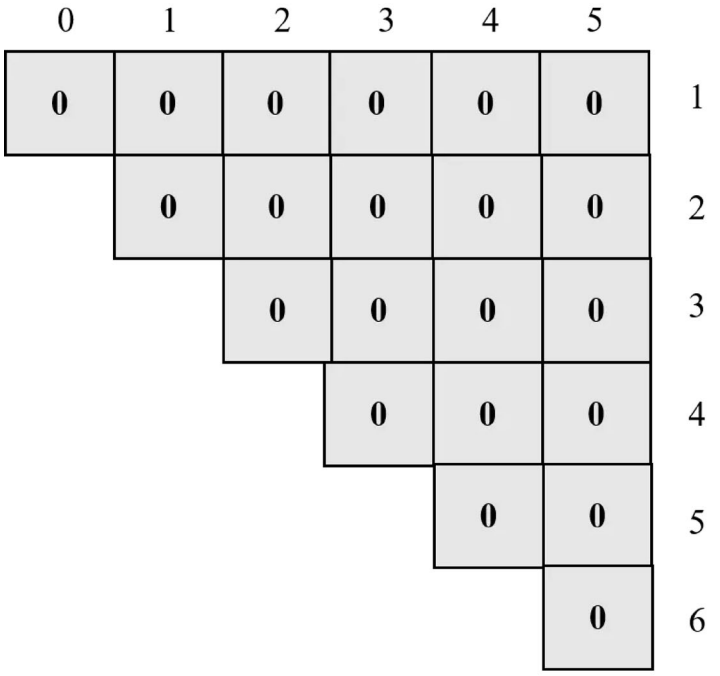


图1 初始动态数组

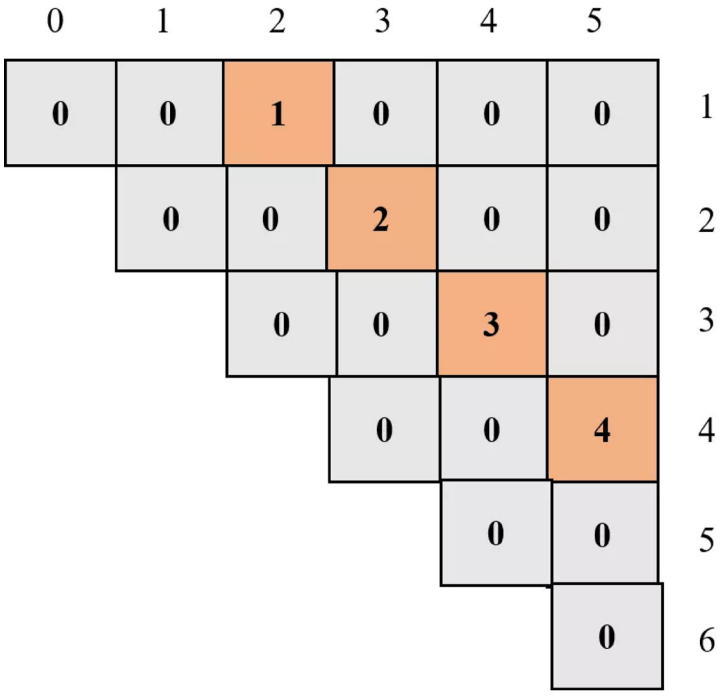


图2 第一次迭代更新

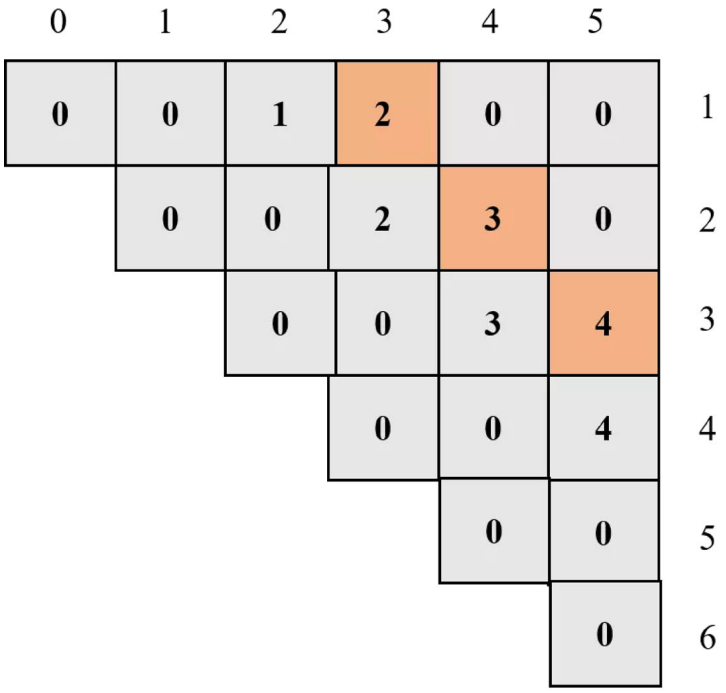


图3 第二次迭代更新

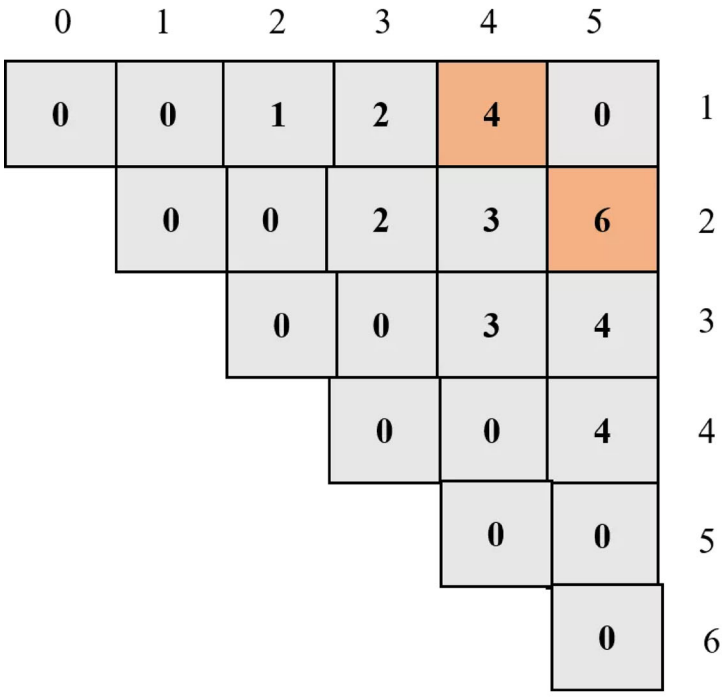


图4 第三次迭代更新

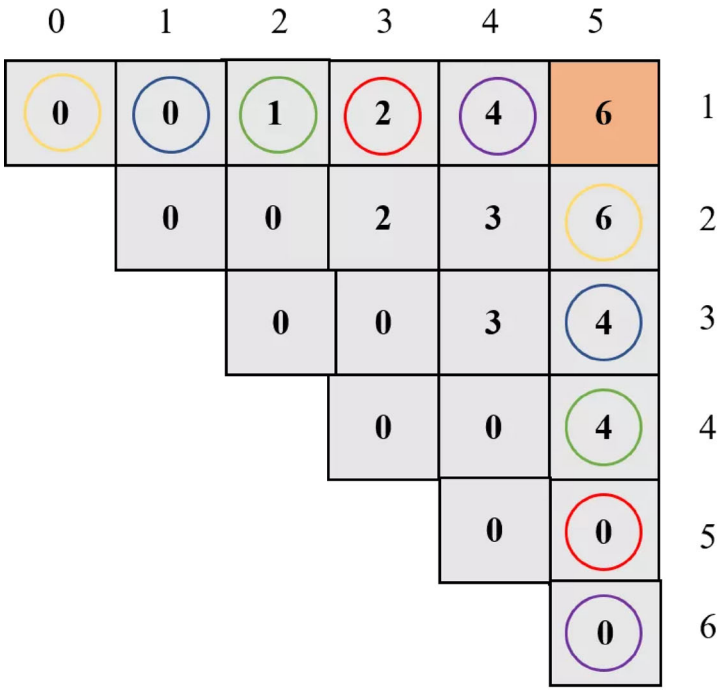


图5 第四次迭代更新

以第四次迭代更新为例，给出动态规划数组更新的具体过程。此时，动态规划数组中只剩 $OPT(1,5)$ 需要更新，更新方式为：

$$OPT[1,5]=\min\begin{cases} max(OPT[1,0],OPT[2,5])+1 & (max(0,6)+1=7)\\ max(OPT[1,1],OPT[3,5])+2 & (max(0,4)+2=6)\\ max(OPT[1,2],OPT[4,5])+3 & (max(1,4)+3=7)\\ max(OPT[1,3],OPT[5,5])+4 & (max(2,0)+4=6)\\ max(OPT[1,4],OPT[6,5])+5 & (max(4,0)+5=9) \end{cases}$$

03 总结

总结一下我们分析与求解问题的思路

1. 对于问题一，我们发现其问题形式十分适合二分查找。
2. 对于问题二，首先我们通过阅读题目，发现能够以极大极小化的思想对问题进行分析与建模。
3. 对问题进行分析后，我们发现了问题的可分性与重复子问题，因此可以使用动态规划的技术进行求解。

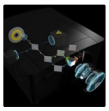
在问题二中，我们借助极大极小化的思想，有了分析问题思路的工具。在分析出问题结构后，我们可以发现问题二最终的递归表达式与与卜老师课上讲解矩阵序列的链式相乘问题的形式是类似的，此部分的思路与卜老师课上讲解矩阵序列的链式相乘问题的思路也是一致的。通过该题我们也可以认识到，课堂上很多的思想对我们在遇到新问题分析求解时很有帮助，很多问题的结构也和我们上课时学到的例子的结构相似，在遇到新的问题时，可以灵活应用卜老师课上的所讲到的算法思想进行分析。

本文以猜数字游戏为例，对分析求解过程进行了阐述，如有谬误，还请指正。

LOA算法学习笔记

光纤是如何发明的?

光电汇OESHOW



以习近平同志为核心的党中央关心学校思想政治工作纪实

中国文明网