《数据结构与算法》大作业题目说明

1. 背景知识

众所周知，有一种国际知名的棋类游戏叫做**黑白棋**，黑白双方通过相互翻转对方棋子来获取优势，并最终以棋盘上棋子的数量来判断胜负。简述黑白棋的规则和特点：黑白棋一般使用 8×8 的棋盘，棋子放置在格子中而不是交叉点。初始状态时，如图 1 所示，棋盘正中有黑白交替四个棋子。由黑方先行。当放下的棋子在横、竖、斜八个方向内有一个自己的棋子时，则被夹在中间的其他棋子全部翻转会成为自己的棋子。每次落子都必须要有翻转，反之则不能落子，由对方继续落子。游戏将持续到所有格子都被填满或游戏一方的棋子全部被翻转，因此双方落子的机会总共不大于 8×8-4=62 次。黑棋（先手）有着较大的优势。

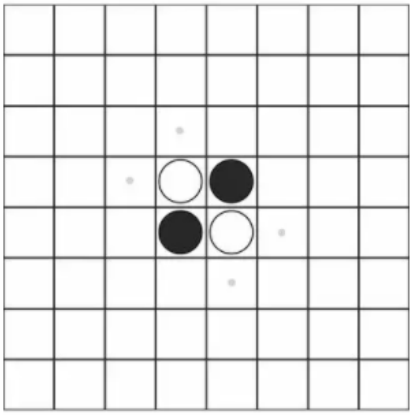

图 1 黑白棋的初始状态

黑白棋的游戏规则简单，上手很容易，棋盘的变化也远没有围棋那么复杂。早在 1997 年 8 月，使用着不如当今手机高效的计算机，执行着非机器学习的代码，人工智能就已经将人类冠军击败，并从此把人类棋手远远甩在后面，是最早被人工智能攻克的人类游戏之一。所以，十分适合拿来作为我们大作业的课题。

但是，我们要做的并不是普通的黑白棋。众所周知，黑白棋中，边沿位置和角落位置因为能够有效地防止被对方棋子包夹，占有着极大优势，是游戏过程中必须要争取位置。那么，能不能设计一个没有边沿和没有角落的黑白棋棋盘？当然是可以的。如图 2 所示，将棋盘卷曲成球面，让每一条边两端连接成一个循环的圆，就可以消除边沿和角落。

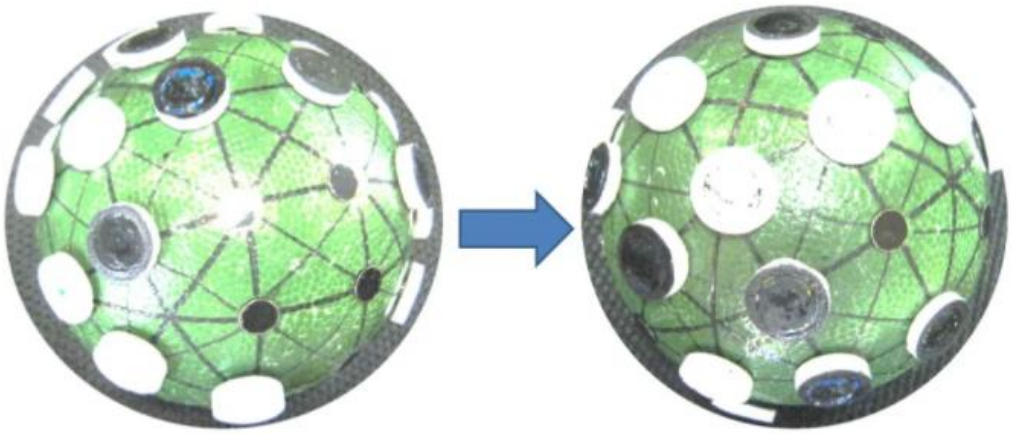

图 2　球面黑白棋

图 3 为将图 2 所示球面黑白棋的 2×2×2 版本进行抽象化之后的立体图，通过线条颜色区分 x,y,z 轴 3 个不同方向的线条，不同方向的两条线之间会产生 2 个交叉点，每条线都会与不同方向的 4 条线交叉并总共产生 8 个交叉点，总计会有 24 个交叉点。

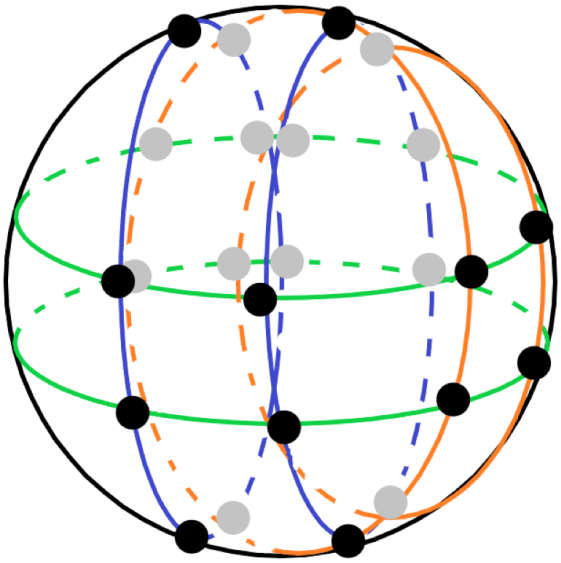

图 3　球面黑白棋的简化立体图

图 4 为对图 3 进一步平面展开后的平面图，线条颜色与图 3 中相对应，分别代表 3 个不同方向的线。

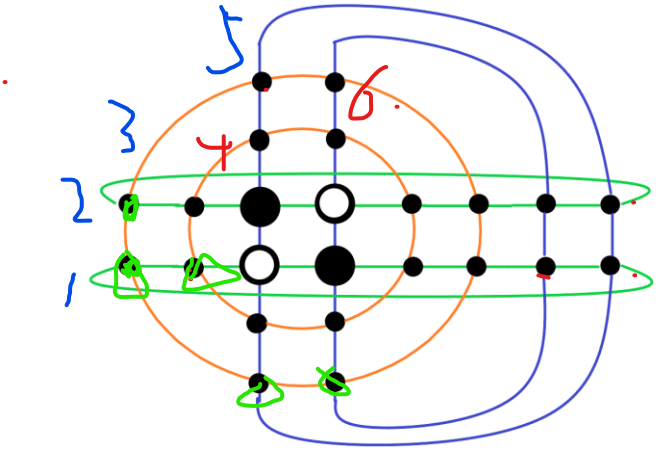

图 4　球面黑白棋的平面展开示意图

详细规则如下，与普通黑白棋不同的地方将加粗表示：

1. 为了方便观察计算，**球面黑白棋的棋子放在线的交叉点上；**
2. 初始状态同样有黑白交错的 4 个棋子预先放在棋盘上，**位置固定为图 4 中所示的中间 4 个交叉点位置；**
3. 黑棋先行；
4. 落子之后同一条线上如果有其他己方棋子，则两颗己方棋子之间包夹的所有对方棋子均变换颜色；
5. 球面黑白棋中判断是否包夹时，**只考虑已有线条而不考虑无线条的斜方向，并且该规则可以顺着圆环线条的两方向同时适用，**从而实现图 5 所示翻转通吃效果；

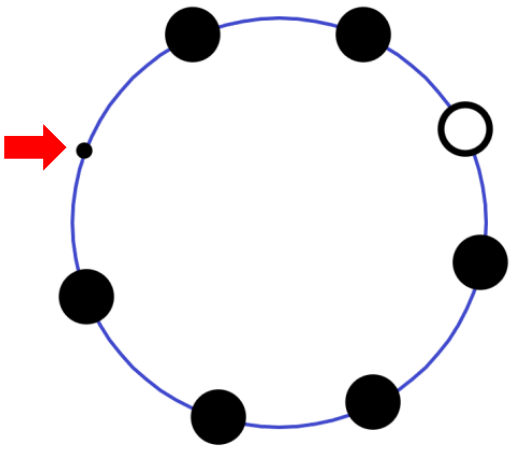

图 5　图中箭头位置放下白子便能够将所有黑子都翻转

6. 每次落子都必须要有翻转，反之则不能落子，由对方继续行动；

7. 所有交叉点都落子，或是一方所有棋子都被翻转时，游戏结束；

8. 游戏结束时，棋子更多的一方获胜。

大作业的课题：实现上述球面黑白棋的逻辑以及球面黑白棋的人工智能。

2. 题目设定

此处将详细说明大作业中的各个部分的提示和细节要求。

2.1 使用合理的数据结构表现球面黑白棋的落子与棋盘的状态

如上文背景中所述，需要保存的数据主要是 24 个交叉点的落子情况，以及 24 个交叉点之间的 6 条连线的逻辑情况。因此，候选方案有多种，下面提示两种方案：

（1） 使用 7 组顺序结构（数组）。其中，数组 1 长度 24 类型为字符型（或其他数据类型），用来保存 24 个交叉点的落子情况；另外 6 个数组长度为 8 类型为数组 1 数据类型的指针类型，指向数组 1 中的对应元素，用来记录各个交叉点的共线情况。

（2） 使用 6 组链表结构来串联 24 个节点。考虑到每个节点只会存在于两条线之上，所以节点的链接域是可以确定的。如下所示，其中可以给 6 条连线标上编号，并在节点中记录线编号与对应的指针域，防止进行遍历时产生混乱。

```
class OthelloNode {
    int status;
    int Lng_code, Lat_code;
    OthelloNode* Lng_pre, Lng_next;
    OthelloNode* Lat_pre, Lat_next;
}
```

其中记录落子位置状况的24个数据构成了黑白棋的棋盘状态，用status进行表示。落子位置之间的连线固定不变且只用来处理翻转逻辑，并不需要记录为棋盘状态。

2.2 设计算法实现落子后的翻转效果

简而言之，遍历。候选方案同样有多种，下面提示两种思路：

（1） 以落子点作为起点，遍历落子点所在的两条连线，查找是否有同色包夹的情况：如果有，则执行翻转处理；否则，则拒绝落子。

（2） 以落子颜色作为对象，遍历所有 6 条联系，查找落子颜色是否出现同色包夹的情况：如果有，则执行翻转处理；否则，则拒绝落子。

落子的函数可以设计为 status move(pos, color, status),表示在 status 状态的棋盘的 pos 位置放置 color 颜色的棋子，经过翻转处理后返回新的棋盘状态。

2.3 设计球面黑白棋的人工智能

棋类的游戏过程本质上便是回溯树的构建、剪枝以及探索过程。图 6 所示黑白棋开局的部分回溯树，如俗话所说的“走一步看十步”，棋类游戏的过程便是通过“预判对方”“预判对方的预判”“预判对方预判己方预判的预判”“……”来执行回溯树的搜索过程，最终采取己方收益最大化的行动。

其中，单次的预判可以简单描述为下述过程：

a) 模拟己方落子；

b) 模拟对方落子；

c) 重复 a)和 b)两个步骤 n 次；

d) 计算当次模拟的收益；

e) 变化模拟的落子状况，重复 a),b),c)和 d)四个步骤；

f) 在所有的模拟情况中，选出收益最大的一条路径，执行己方的落子。

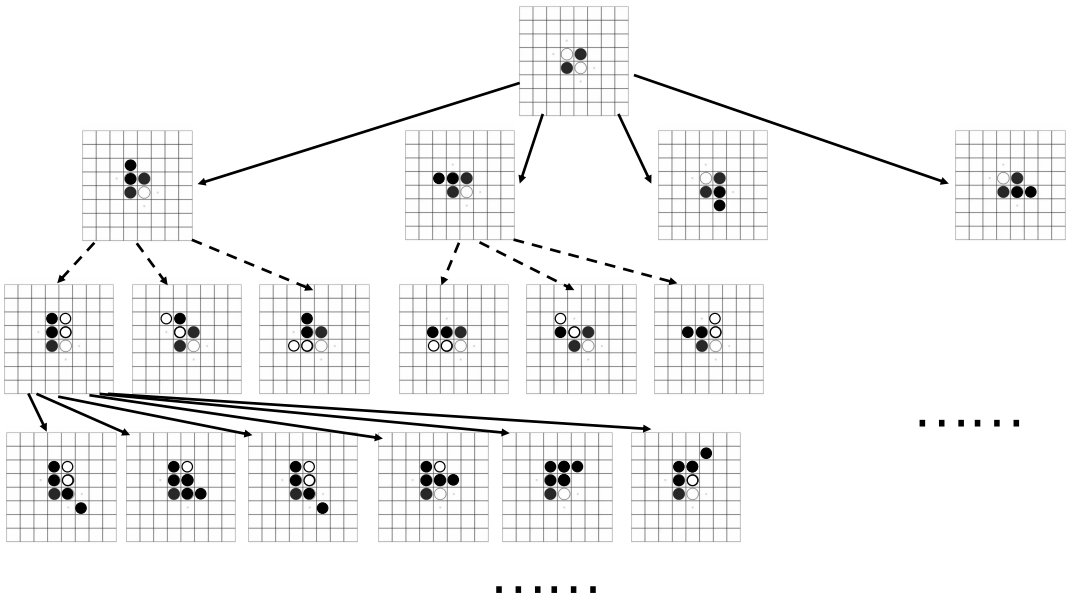

图 6 平面黑白棋的回溯树（黑线代表黑子落子，虚线代表白子落子）

人类智能在上述过程中，能够通过经验和学习对 a)和 b)步骤中模拟的落子进行筛选（即对回溯树进行剪枝）来加快回溯树的探索过程以及回溯树的探索深度。近年来的人工智能才刚能通过深度学习方法实现这种剪枝操作。但是，黑白棋之所以在如此早期便被人工智能攻克，主要是因为其棋盘的变化相对简单，人工智能甚至能够通过多线程的暴力计算穷举所有情况，几乎实现“走一步看全局”的效果，从而实现了对人类棋手的压倒性优势。

球面黑白棋与平面黑白棋同理，因此球面黑白棋的人工智能实现方式有多种，下面提示三种思路：

（1） 使用递归方法探索回溯树。即，模拟上文中所述预判过程，通过递归函数来构建回溯树，模拟落子过程来得到 n 步之后的收益，并根据模拟所得的所有收益结果 results 综合判断落

子位置。下述伪代码仅供参考。

```
void forecast(step_number, color, &results, status){
    // step_number表示当前模拟进行的步数
    // color表示当前落子方颜色
    // results记载所有模拟的最终收益结果
    // status为当前的棋盘状态
    if (step_number > max_step_number){
        // 步数大于预定的最大步数
        // 计算当前局面的收益情况并保存
        results.append(compute_result(status))
    } else {
        // 反之，开始进行模拟
        for (pos: available_pos(color, status)){
            // available_pos找出己方可落子位置，并逐个模拟
            status = move(pos, color, status);
            // 落子，同时变化棋盘状态
            for(opp_pos: available_pos(opp_color, status)){
                // 模拟对方行动
                status = move(opp_pos, pp_color, status);
                // 进行下一步预测
                forecast(step_number+1, color, &results, status);
            }
        }
    }
}
```

（2） 球面黑白棋中落子位置为 20 个交叉点，每次落子的翻转判断只有两线，较之平面黑白棋，棋盘的变化进一步减少，完全能够在游戏之前构建整个回溯树，在游戏过程中探索回溯树进行遍历，基于探索所得收益来判断落子位置。回溯树并非二叉树，构建方法可以参考数据结构课程中所讲述的内容，下述代码为树的“孩子兄弟表示法”，仅供参考。而回溯树的探索过程同样可以使用上述递归代码，因为可以省略重复的模拟落子的处理操作，探索速度能够极大加快，因此可以加大探索的深度。

```
class OthelloTreeNode{
    OthelloStatus status; //记录棋盘状态
    OthelloTreeNode* next;
    // next记录孩子结点，即对方落子后的棋盘状态
    OthelloTreeNode* other;
    // other记录兄弟结点，即己方当前落子的其他可能棋盘状态
    ...
}
OthelloTreeNode OthelloTree(init_status);
//记录初始状态，即只有2黑2白的4个落子
//next指向黑子先行落子的后一种棋盘局面情况
//other 指向 null
```

（3） 在（2）思路构建的全局回溯树基础上，可以进一步在游戏之前就计算每一步落子所有能够得到的最终收益。如下图 7 所示，每一步的收益可以通过该步落子所能到达的最终状态（即叶子结点），从下到上对各个中间状态来进行评估和计算。于是，人工智能便能够在全局信息的基础上进行最终收益的综合，从而做出最优选择。

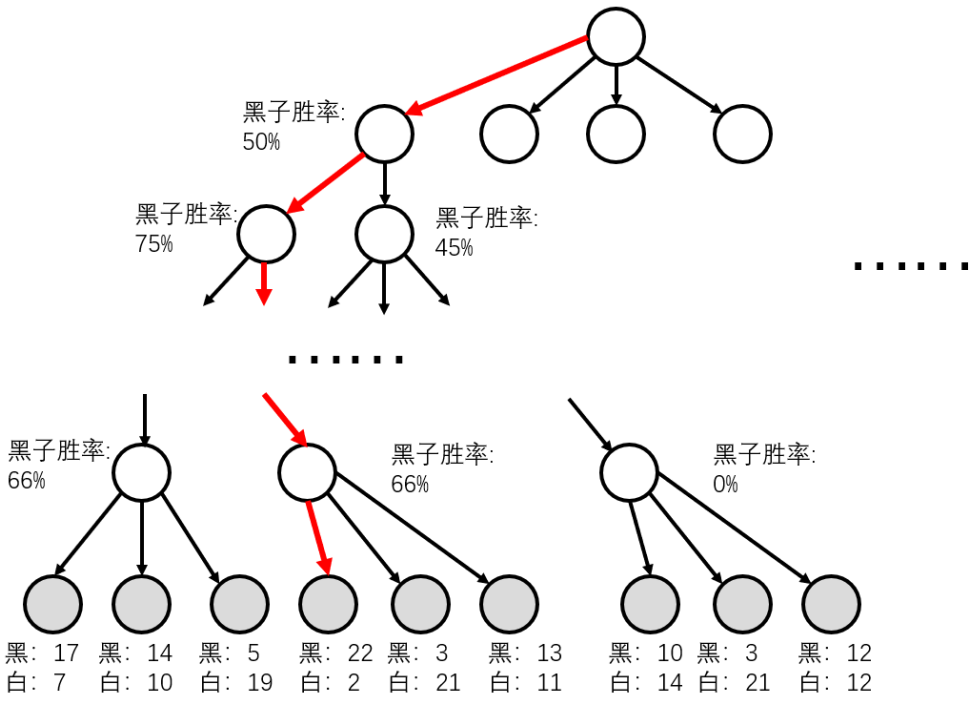

图 7 通过当前局面所能得到的黑子最终胜率来选择落子路径

其中，收益的评估方式有多种，下面提示两种思路，仅供参考，不一定最佳。

（1） 贪心规则，即选取所有模拟过程中己方净胜对方棋子数最多的路径。例如，黑子落子 1 位

置时最后能够模拟出黑子比白子多 14 子的结果，落子 2 位置时最后能够模拟得到比白子多 12 子的结果，则选择落子 1 位置。

（2） 胜率规则，即选取所有模拟过程中胜率最大的路径。例如，黑子落子 1 位置时最后模拟得出的所有 15 个结果中有 10 个结果黑子能多于白子，落子 2 位置时最后模拟得出的所有 12 个结果中有 9 个结果黑子能多于白子，计算胜率得到后者 75%大于前者 66%，则选择落子 2 位置。

当然，完全可以综合使用多种评估方式，例如优先保证胜率时考虑使净胜棋子数量最大，等等。

其中，黑白棋的规则“每次落子都必须要有翻转，反之则不能落子，由对方继续行动”会较大影响回溯树的探索过程和收益评估过程，一定要注意处理。

实现过程中可以尝试减少线条数，将 2×2×2（24 交叉点）简化成 2×1×1（10 交叉点）来进行调整和测试，能够更好的查找错误。

2.4 实现可人机对战的球面黑白棋游戏

假设玩家执黑子，则游戏过程可以简单整理下述状况：

（1） 初始化棋盘；

（2） 判断游戏是否结束：棋盘是否已经下满，或是黑子是否已经全部翻转；

（3） 判断黑子是否可以落子，不可以落子则跳至（6）；

（4） 玩家输入黑子落子位置，执行翻转处理，如果不能翻转，则拒绝该次输入，要求玩家继续输入；

（5） 判断游戏是否结束：棋盘是否已经下满，或是白子是否已经全部翻转；

（6） 判断白子是否可以落子，不可落子则跳至步骤（3）；

（7） AI 计算白子落子位置并落子，执行翻转处理；

（8） 判断所有交叉点是否重复步骤（2）至（7）；

3. 作业要求

1. 尽可能多的使用本学期《数据结构和算法》的知识和技能来解决以上问题。
2. 从实际耗时和人工智能的效果用户使用的角度出发，**尽量采用提示的人工智能算法思路中（2）和（3）的思路，使用第（1）种方法会降低评分**。
3. 游戏过程中的图像表现方式可以使用图 4 的平面展开图来进行表现。
4. 如对题目有疑问，可联系 longzi@sztu.edu.cn 咨询。
5. 要求大家**独立**完成，第 18 周交。提交时每人都需要提交**大作业报告，源代码，ppt 和 5 分钟视频讲解录像**，打包成一个文件，文件名统一为“学号_姓名”。提交方式由各个理论课

老师决定。