博弈论

湖南师范大学
罗迅

博弈论（GAME THEORY）

- 《孙子兵法》
- 冯·诺依曼

- 约翰·纳什

囚徒困境

- A、B两个嫌疑人分别单独接受审讯

- 其中一个坦白，而另一个抵赖，则坦白人释放，抵赖者判刑10年
- 双方都坦白，均判刑8年
- 双方都抵赖，由于没有其他证据以证明更严重的罪行，双方均判刑2年

合作案例

- 43个参会人，1个主持人
- 每个参会人分别单独的缴纳一定的会费
- 如果总会费超过250元，则主持人将向每参会人返还10元
- 如果总会费未超过250元，则全部归公，一分不还

- 这是一个有趣的实验，有条件可以做一下

ACM之博弈问题

- ACM的博弈问题诸如打牌、下棋或者是某种两人胜负游戏
- 博弈双方绝对聪明和理性，每一步都是当时的全局最优，即使输也要输得最少
- 毫无疑问，给定一个局面唯一确定一个结果，而不管这个局面是怎么来的，这是很明显的DP性质

ACM之博弈问题

- 令胜负值为先手得分减去后手得分
- 则先手的目标是胜负值越大越好
- 后手的目标则是该数值越小越好
- 显然你要考虑很多很多步，这是一个典型的DFS思路
- 由于局面导致结果的唯一性，这是一个记忆化搜索

DFS函数的标准写法

- dfs(depth)
  - if (depth已经到底) return;
  - for所有可能的操作
    - 进行操作
    - dfs(depth+1)
    - 恢复

- 结论可以由全局变量记录，也可以由dfs函数返回

10884

- 4×4的棋盘，双方轮流落子，先形成3子连线（横竖斜均可）的获胜
- 给定残局，问谁能获胜

10884

- 这就是一个典型的DFS过程
- 对每一步棋，穷尽考虑所有可能的落子，每次落子继续往下递归考虑下一步所有可能的落子
- 与标准dfs的区别在于每次递归需要变换角度

10884

- dfs(step)
  - 如果棋盘已满，return
  - 对棋盘上所有的空位
    - 落子
    - dfs(step+1)
    - 怎么判断胜负、返回结论？
    - 悔棋

最大最小值函数

```
int Max(int depth) {
  int best = -INFINITY;
  if (depth <= 0) {
    return Evaluate();
  }
  GenerateLegalMoves();
  while (MovesLeft()) {
    MakeNextMove();
    val = Min(depth - 1);
    UnmakeMove();
    if (val > best) {
      best = val;
    }
  }
  return best;
}
```

```
int Min(int depth) {
  int best = INFINITY;  // 注意这里不同于“最大”算法
  if (depth <= 0) {
    return Evaluate();
  }
  GenerateLegalMoves();
  while (MovesLeft()) {
    MakeNextMove();
    val = Max(depth - 1);
    UnmakeMove();
    if (val < best) {  // 注意这里不同于“最大”算法
      best = val;
    }
  }
  return best;
}
```

负值最大函数

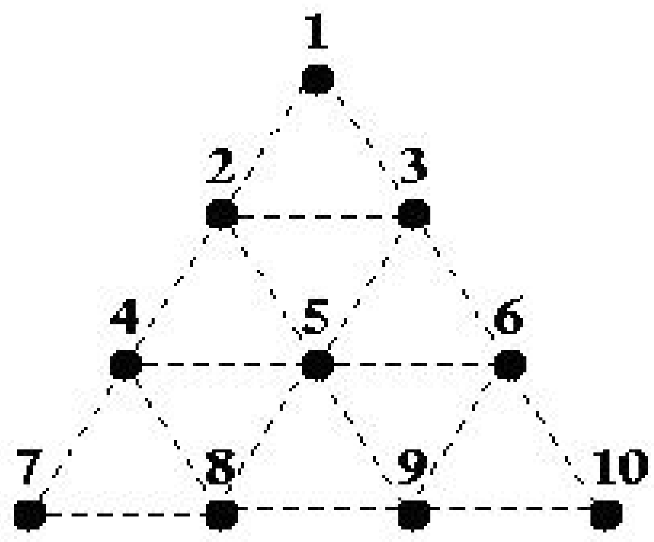
```
int NegaMax(int depth) {
  int best = -INFINITY;
  if (depth <= 0) {
    return Evaluate();
  }
  GenerateLegalMoves();
  while (MovesLeft()) {
    MakeNextMove();
    val = -NegaMax(depth - 1); // 注意这里有个负号
    UnmakeMove();
    if (val > best) {
      best = val;
    }
  }
  return best;
}
```

ACM之博弈问题

- 博弈问题的DP解法一般与DFS没有区别
- 因为博弈DP一般都是从父结构去递归子结构，所以也就是一个DFS过程
- 另外由于局面的确定性，实际上就是一个记忆化搜索

POJ1085

- 双方轮流连线
- 如果能够完成一个三角形，则再走一步
- 给定残局，问，谁能获胜



- 动态规划

POJ1085

- POJ1085是一个非常好的题目
- 这道题不仅仅是实践博弈、DP、记忆化搜索等思想
- 这道题对代码表达能力也是很有用的，如何用程序语言表达这样一个游戏，包括其状态、走棋、胜负等，相当于一道模拟题

NIM游戏

- POJ2234
- 若干堆、若干个石子
- 双方轮流从一堆中取出可取的数目
- 拿走最后石子者获胜，取无可取者失败

- 给定初始条件，先手能否获胜
  - 3，Yes
  - 1，1，No

NIM问题

- Impartial Combinatorial Games, ICG
  - 两个人参与，交替走棋
  - 可能的走法在一个有限集合里选取
  - 游戏局面无后效性，未来与过去无关
  - 如果某选手无法走动，则判负

NIM理论

- 任何一个局面必然二属其一：当前先手必胜（N位置）、当前先手必败（P位置）

- 数学定义：
  - 终态是P位置
  - 能够移动到P位置的状态是N位置
  - 只能去N位置的状态是P位置
- 注解：走投无路就是必败，就是P

示例

- Nim问题中的（3,3），这是一个P
- 这个位置往下有3种走法
  - （3,0），显然是N
  - （3,1），也是N；因为从（3,1）可以走到（1,1），而（1,1）是P
  - （3,2），简单枚举就知道也是N
- 所以，（3,3）无论怎么走都走到N，则（3,3）是P
- 反过来，（3,1）既可以走到（1,1），也可以走到（2,1）或者其他状态，但只要（1,1）是一个P位置，（3,1）就是一个N位置

NIM问题的解法

- 所有堆的石子数目求异或，为0则是P位置
- 证明：
  - 终态满不满足该条件？满足
  - 异或非0的N位置能不能移动到异或为0的P位置？能，找到非零的最高位进行操作
  - 异或为0的P位置是不是只能移动到异或非0的N位置？是，因为无论怎么移，异或和一定变成非0

MEX函数

- 运算对象是一个集合
- 最小的不属于该集合的非负整数
- mex({0,1,2,4}) = 3
- mex({2,3,5}) = 0
- mex(空集) = 0

SG函数

- 定义在有向无环图
- g(x) = mex({ g(y) | y是x的后继 })

SG的性质

- 所有终态的sg为0
- 如果已知一个点的sg是0，则它的所有后继的sg均不为0
- 如果已知一个点的sg非0，则它必然有一个后继sg为0

- 通过这三条定义可知：
- sg为0的点就是P位置

SG 的应用

- 首先判断是否可以使用SG
- 其次将游戏分解成多个独立的小游戏
- 再次计算每个小游戏的SG值
  - 前驱与后继的关系
- 游戏总的SG值是各小游戏SG值的异或和

题目列表

- hdu1404，DP + DFS
- POJ2960，SG函数
- hdu1729，SG函数，SG函数的求法
- hdu1730，极其简单
- hdu1760，DP+DFS，建立SG函数的概念
- hdu1809，与上题类似，体现SG的价值
- POJ2425hdu1524，SG函数可解，应该有更好的实现思想

题目列表

- ZOJ1913，加减法
- ZOJ1893，乘除法
- ZOJ1024，奇偶性

- 上述题目都可以用SG的思想，但直接使用数学思维会更简便。

题目列表

- 10401
- 10884，可能出现平局
- 11008
- 11306，最大最小值函数
- 11317

OVER