极大极小值搜索+AlphaBeta剪枝

算法概念：

极小极大搜索算法即minimax搜索算法

主要应用于零和博弈（非胜即负，如围棋，象棋，井子棋等），完全信息（玩家知道之前所有的步骤。象棋就是完全信息，因为玩家是交替着落子，且之前的步骤都能在棋盘上体现）

这个算法采用搜索算法递归实现，一层为先手，记为a, 一层为后手，记为b, 交替出现

对于最终局面，有一个分数（比如：先手胜分数为1， 平局分数为0，先手输分数为-1）

先手a想要让这个分数越大越好，后手b想要让这个分数越小越好，于是搜索到先手这一层，取最大返回，搜索到后手这一层，取最小返回

关于alpha-beta剪枝：

如果当前层为取最小，如果取最小后比上一层当前最大值还小，则不需要往下搜索，因为上一层根本不会选择当前节点往下搜，还有更好的选择

同理，如果当前层为取最大，如果取最大后比上一层当前最小值还大，则不需要往下搜索，因为上一层根本不会选择当前节点往下搜

参考文献：[https://blog.csdn.net/SKnight\_31/article/details/81040345 //](https://blog.csdn.net/SKnight_31/article/details/81040345%20//)关于概念理解

<https://blog.csdn.net/u013351484/article/details/50810224> //游戏博弈规则设置

实验内容：

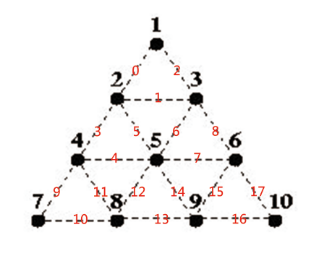
游戏有A、B两人参与。A先走，每人每次任选一条虚线填成实线。而如果某人填完一条线段后，该线段与另外两条相邻的实线组成了一个单位三角形，该三角形被标记为该游戏者所有，且该游戏者必须接着再填一条虚线。当18条线段被填充完毕后，拥有三角形多的玩家获胜。

        首先输入一个数 T ，表示试验次数；接着一个数 n，表示当前局面已经走了 n 条边；接着输入 n 对数，每对数字表示一条填充边，由 A 开始。问从给出的局面开始，双方都采取最优策略，最终谁会取胜。

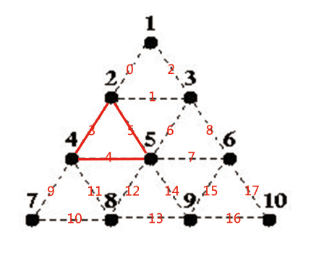
思路：

     思路：

  显然是博弈问题，也是 alpha-beta 搜索算法的基础应用（博弈基础：点击打开链接）。一个比较关键的问题是局面的存储。整个棋盘有18条边，因此可以用一个整型来存储一个局面，具体就是：



        如果每条边用红色数字表示（当然也可以不这样排列），则用 2^0 = 1 表示 edge(1, 2) 或 edge(2, 1)，用 2^1 = 2(即二进制10)表示 edge(2, 3) 或 edge(3, 2)，用2^2 = 4(即二进制100)表示 edge(1, 3) 或 edge(3, 1) ...



        那这个局面用哪个数存储？用 2^3 + 2^4 + 2^5 = 56 表示；其余单位三角形可用同样的方法计算。

代码：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// 保存每条边代表的数字

int edge[11][11]={

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 1, 3, 5, 0, 0, 0, 0, 0},

{0, 2, 1, 0, 0, 6, 8, 0, 0, 0, 0},

{0, 0, 3, 0, 0, 4, 0, 9, 11,0, 0},

{0, 0, 5, 6, 4, 0, 7, 0, 12,14,0},

{0, 0, 0, 8, 0, 7, 0, 0, 0, 15,17},

{0, 0, 0, 0, 9, 0, 0, 0, 10,0, 0},

{0, 0, 0, 0, 11,12,0, 10,0, 13, 0},

{0, 0, 0, 0, 0, 14,15,0, 13, 0, 16},

{0, 0, 0, 0, 0, 0, 17,0, 0, 16,0},

};

// 保存每个单位三角形代表的数字

int tri[9] = {7, 56, 98, 448, 3584, 6160, 28672, 49280, 229376};

int end\_state = (1<<18)-1; // 终结状态 2^18 - 1 ，即所有边均被填充

int inf = (1<<20);

int next\_state(int cur\_state, int edge, int \*cnt)

{

int i;

int new\_state = (cur\_state | edge); // 当前局面并上一条边形成新局面

for(i = 0; i < 9; i++) // 如果新局面能形成一个新的单位三角形，则 cnt++

if(((cur\_state & tri[i]) != tri[i]) && ((new\_state & tri[i]) == tri[i]))

(\*cnt)++;

return new\_state;

}

int alpha\_beta(int player, int cur\_state, int alpha, int beta, int ca, int cb)

{

int remain;

// 如 A 得到 5 分以上则 A 赢

// 如 B 得到 5 分以上则 A 输

if(ca >= 5) return 1;

if(cb >= 5) return -1;

remain = ((~cur\_state) & end\_state); // 计算剩余可走的边

if(player){ // A 走

while(remain){ // 有可走边

int move = (remain & (-remain)); // 选择一条可走边

int ta = ca;

int val;

// A 填了边后形成新的局面

int new\_state = next\_state(cur\_state, move, &ta);

if(ta > ca) // 如果 A 得分了，则 A 继续填一条边

val = alpha\_beta(player, new\_state, alpha, beta, ta, cb);

else // 否则轮到 B 填

val = alpha\_beta(player^1, new\_state, alpha, beta, ca, cb);

if(val > alpha)

alpha = val;

if(alpha >= beta)

return alpha;

remain -= move; // 把边 move 从剩余可选边 remain 中移除

}

return alpha;

}

else{ // B 走

while(remain){

int move = (remain & (-remain));

int tb = cb;

int val;

int new\_state = next\_state(cur\_state, move, &tb);

if(tb > cb)

val = alpha\_beta(player, new\_state, alpha, beta, ca, tb);

else

val = alpha\_beta(player^1, new\_state, alpha, beta, ca, cb);

if(val < beta)

beta = val;

if(alpha >= beta)

return beta;

remain -= move;

}

return beta;

}

}

int main()

{

#if 0

freopen("in.txt","r",stdin);

#endif

int T, w = 0;

scanf("%d", &T);

while(T--){

int i;

int n;

int ans;

int cnt = 0; // 偶数轮到 A 走，奇数轮到 B 走

int cur\_state = 0; // 当前局面

int ca = 0; // A 的得分

int cb = 0; // B 的得分

int ta, tb;

int alpha = -inf;

int beta = inf;

scanf("%d", &n);

for(i = 0; i < n; i++){

int u, v;

ta = ca;

tb = cb;

scanf("%d%d", &u, &v);

cur\_state = next\_state(cur\_state, 1<<edge[u][v], (cnt & 1) ? (&cb) : (&ca));

if(ta == ca && tb == cb) // 不得分，轮到对方走

cnt++;

}

if(cnt & 1)

ans = alpha\_beta(0, cur\_state, alpha, beta, ca, cb);

else

ans = alpha\_beta(1, cur\_state, alpha, beta, ca, cb);

if(ans > 0)

printf("Game %d: A wins.\n", ++w);

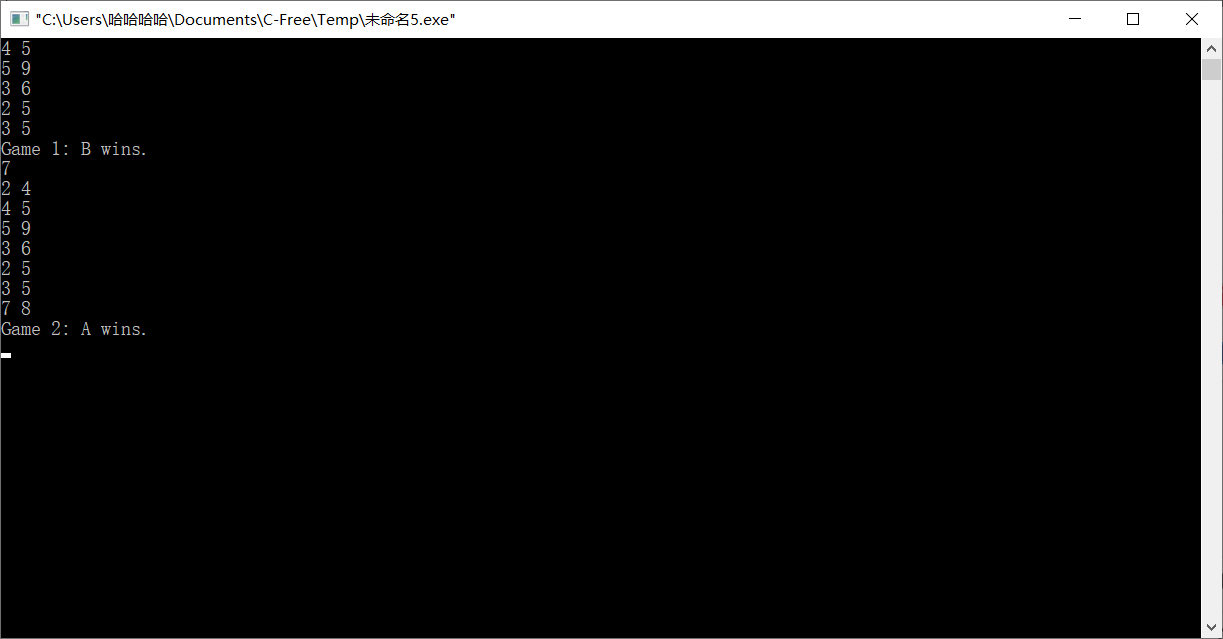
else

printf("Game %d: B wins.\n", ++w);

}

return 0;

}

运行结果:

实验总结：

对于极大值极小值是比较好理解的，但对于ALphaBeat算法相对于比较难理解，在看完网上资料后，掌握了一定的诀窍，就是要考虑好父节点的极大极小值性质，比较兄弟节点来剪枝，这样的话对于该不该剪枝有了更好的判断方法。