



北京大学
PEKING UNIVERSITY

北京大学暑期课 《ACM/ICPC竞赛训练》

北京大学信息学院 郭炜

guo_wei@PKU.EDU.CN

<http://weibo.com/guoweiofpku>

课程网页: http://acm.pku.edu.cn/summerschool/pku_acm_train.htm



高级搜索算法

A*, 迭代加深, Alpha-Beta剪枝

本讲义并非全部原创，拷贝并编辑整理了大量网络资源，仅用于内部授课

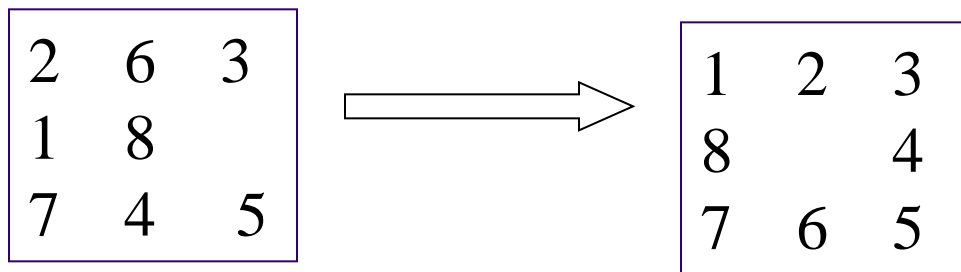


A*算法

启发式搜索算法 (A算法)

- 在BFS算法中，若对每个状态 n 都设定估价函数 $f(n)=g(n)+h(n)$ ，并且每次从Open表中选节点进行扩展时，都选取 f 值最小的节点，则该搜索算法为启发式搜索算法，又称A算法。
- $g(n)$ ：从起始状态到当前状态 n 的代价
- $h(n)$ ：从当前状态 n 到目标状态的估计代价

A算法的例子--八数码



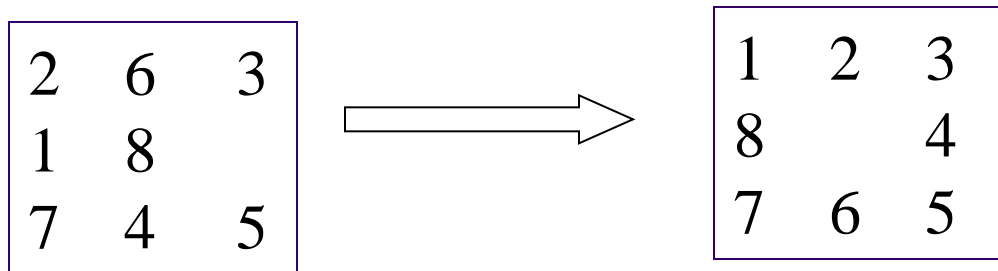
定义估价函数：

$$f(n) = g(n) + h(n)$$

$g(n)$ 为从初始节点到当前节点的步数

$h(n)$ 为当前节点“不在位”的方块数

h计算举例



2,6,1,8,4,5都不在位，因此 $h(n) = 6$

A*算法

- A算法中的估价函数若选取不当，则可能找不到解，或找到的解也不是最优解。因此，需要对估价函数做一些限制，使得算法**确保找到最优解**（步数，即状态转移次数最少的解）。A*算法即为对估价函数做了特定限制，且确保找到最优解的A算法。

A*算法

- $f^*(n) = g^*(n) + h^*(n)$

$f^*(n)$: 从初始节点S0出发，经过节点n到达目标节点的最小步数（真实值）。

$g^*(n)$: 从S0出发，到达n的最少步数（真实值）

$h^*(n)$: 从n出发，到达目标节点的最少步数（真实值）

估价函数 $f(n)$ 则是 $f^*(n)$ 的估计值。

A*算法

- $f(n) = g(n) + h(n)$, 且满足以下限制:

$g(n)$ 是从 s_0 到 n 的真实步数（未必是最优的），因此：

$$g(n) > 0 \text{ 且 } g(n) \geq g^*(n)$$

$h(n)$ 是从 n 到目标的估计步数。估计总是过于乐观的，即

$$h(n) \leq h^*(n)$$

且 $h(n)$ 相容，则A算法转变为A*算法。A*正确性证明略。

A*算法

$h(n)$ 的相容:

如果 h 函数对任意状态 s_1 和 s_2 还满足:

$$h(s_1) \leq h(s_2) + c(s_1, s_2)$$

$c(s_1, s_2)$ 是 s_1 转移到 s_2 的步数, 则称 h 是相容的。

h 相容能确保随着一步步往前走, f 递增, 这样A*能更高效找到最优解。

h 相容 \Rightarrow

$$g(s_1) + h(s_1) \leq g(s_1) + h(s_2) + c(s_1, s_2) = g(s_2) + h(s_2)$$

\Rightarrow

$f(s_1) \leq f(s_2)$ 即 f 是递增的。

A * 算法伪代码(在节点信息中记录了其父节点):

open=[Start]

closed=[]

while open不为空 {

 从open中取出估价值f最小的结点n

 if n == Target then

 return 从Start到n的路径 // 找到了!!!

 else {

 for n的每个子结点x {

 if x in open {

 计算新的f(x)

 比较open表中的旧f(x)和新f(x)

 if 新f(x) < 旧f(x) {

 删掉open表里的旧x，加入新x

 } //新旧x的父亲，以及f值不同

 }

A * 算法伪代码(在节点信息中记录了其父节点):

```
    else if  $x$  in closed {  
        计算新的 $f(x)$   
        比较closed表中的旧 $f(x)$  和新 $f(x)$   
        if 新 $f(x) < 旧f(x)$  {  
            remove  $x$  from closed  
            add  $x$  to open  
        }  
    }
```

// 比较新 $f(x)$ 和旧 $f(x)$ 实际上比的就是新旧 $g(x)$, 因 $h(x)$ 相等

```
    else {  
        //  $x$ 不在open, 也不在close, 是遇到的新结点  
        计算 $f(x)$  add  $x$  to open  
    }
```

```
} // for  $n$ 的每个子结点 $x$   
add  $n$  to closed
```

```
}
```

```
}
```

A*算法

A*算法的搜索效率很大程度上取决于估价函数 $h(n)$ 。一般说来，在满足 $h(n) \leq h^*(n)$ 的前提下， $h(n)$ 的值越大越好。

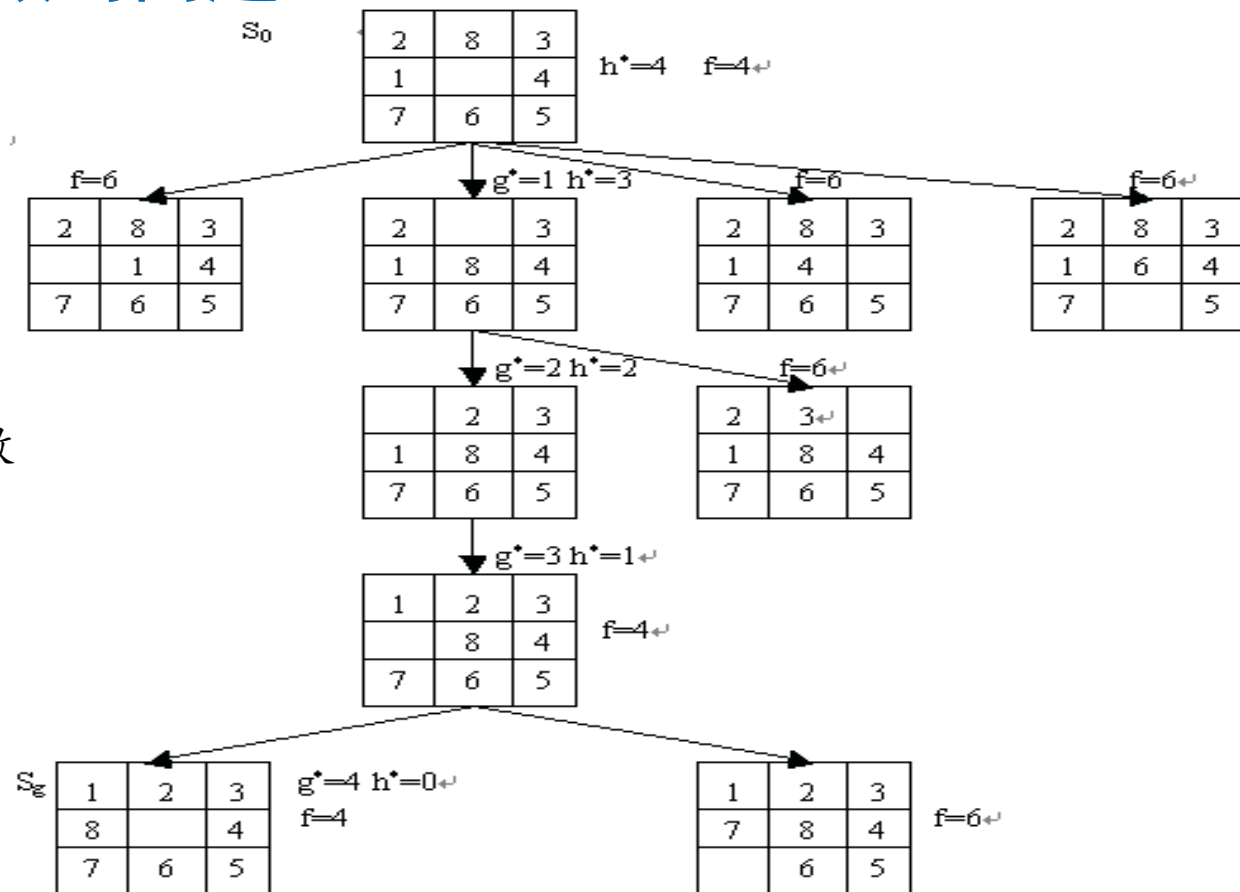
八数码问题：

方案一. $h(n)$ 为不在位的数字个数

方案二. $h(n)$ 为不在位的数字到其该呆的位置的曼哈顿距离和

后者优于前者

A*算法解决八数码问题



$h(n)$ 为不在位的数字到其该呆的位置的曼哈顿距离和

A*算法解决八数码问题实现的要点

multiset中元素的迭代器不因增删其他元素而失效

```
struct Node {
```

```
    int status;
```

```
    int f;    int g;    int h;
```

```
    int parent; // 父状态
```

```
    char move; //经由何种动作到达本状态
```

```
};
```

```
multiset<Node > open; //按f从小到大排序
```

```
multiset<Node > closed; //按f从小到大排序
```

```
map<int, multiset<Node>::iterator> inOpen;
```

```
//元素的 first代表状态， second代表first在open表中的地址
```

```
map<int, multiset<Node>::iterator> inClosed;
```

```
//元素的 first代表状态， second代表first在closed表中的地址
```

POJ上可用A*算法解决的题:

1376

1324

1084

2449

1475



迭代加深搜索算法

迭代加深搜索算法

- 算法思路
 - 总体上按照深度优先算法方法进行
 - 对搜索深度需要给出一个深度限制 dm ，当深度达到了 dm 的时候，如果还没有找到解答，就停止对该分支的搜索，换到另外一个分支继续进行搜索。
 - dm 从1开始，从小到大依次增大（因此称为迭代加深）
- 迭代加深搜索是最优的，也是完备的

例：旋转游戏 (POJ2286)

在如下图的棋盘上，摆放着8个1，8个2和8个3，每一步你可以沿着A、B、C、D、E、F、G、H任意一个方向移动该字母所指的长块。移出边界的小块会从另一端移进来。如图，最左边的棋盘经过操作A，就会变成中间的棋盘布局，再进行操作C，就会变成右边的棋盘布局。

你需要设法使的最中间的8个格子的数字相同，问最少需要多少步。如何移动？

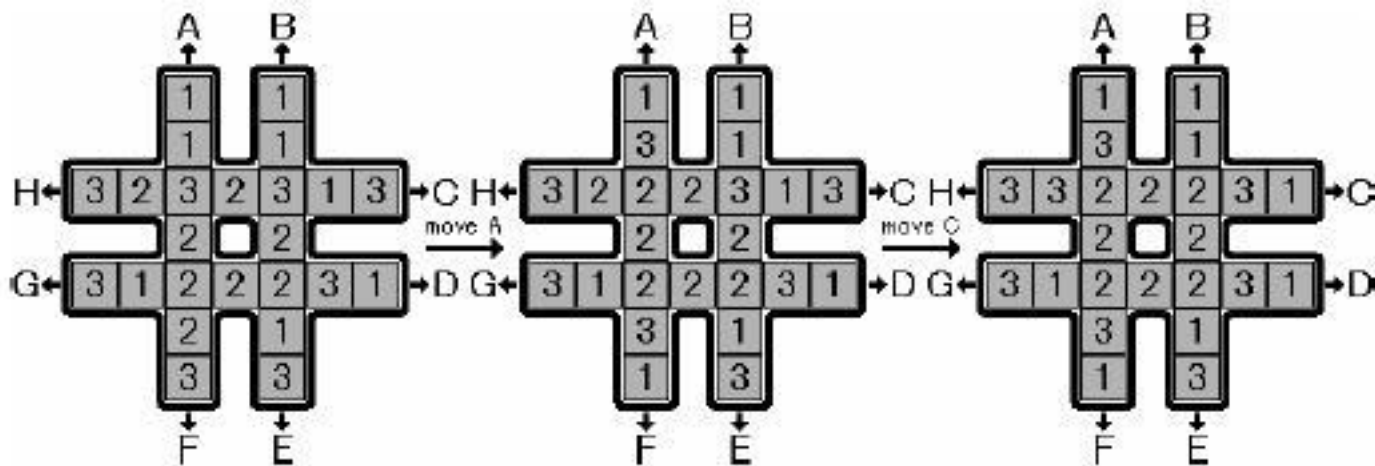


Fig.1

例：旋转游戏 (POJ2286)

1. 用迭代加深dfs做，每次在限定一个最大深度的情况下做完全的dfs搜索（除非已经找到解）
2. 如何剪枝？如何选取估价函数？（乐观预测从一个局面到最终界面至少需要的代价）

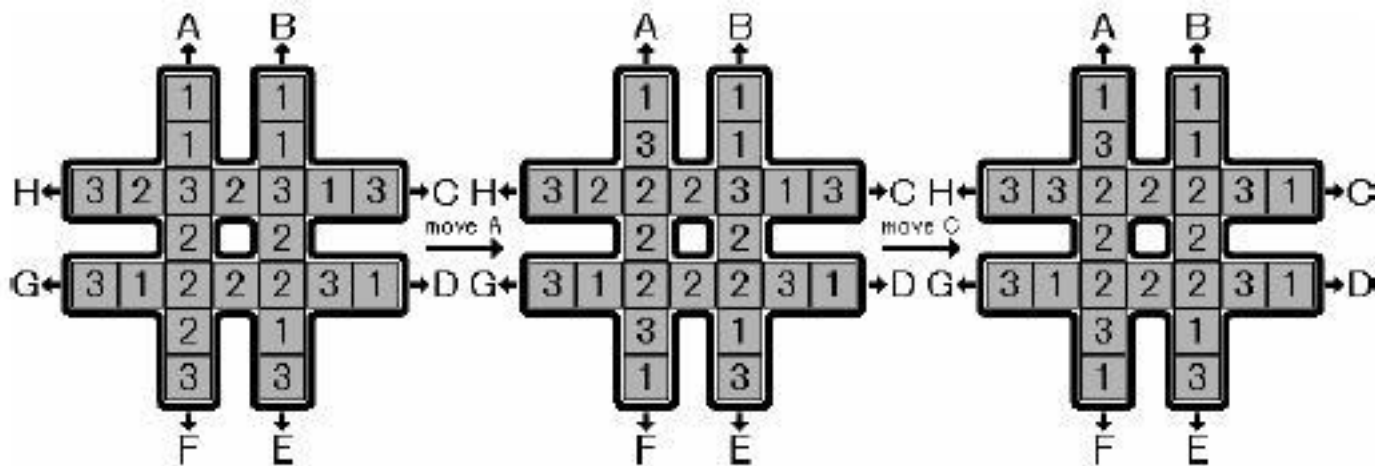


Fig.1

例：旋转游戏 (POJ2286)

剪枝：

- 每次移动最多只会使得中央区域包含的数字种类减少1种。求出中央区域个数最多的那个数字的个数 n ， 要达到中央区域数字都相同， 至少需要 $8-n$ 次操作， 此即估价函数值

- 可以用于可行性剪枝。已经移动的步数加上估价函数值， 超过本次dfs限定的深度， 则剪枝

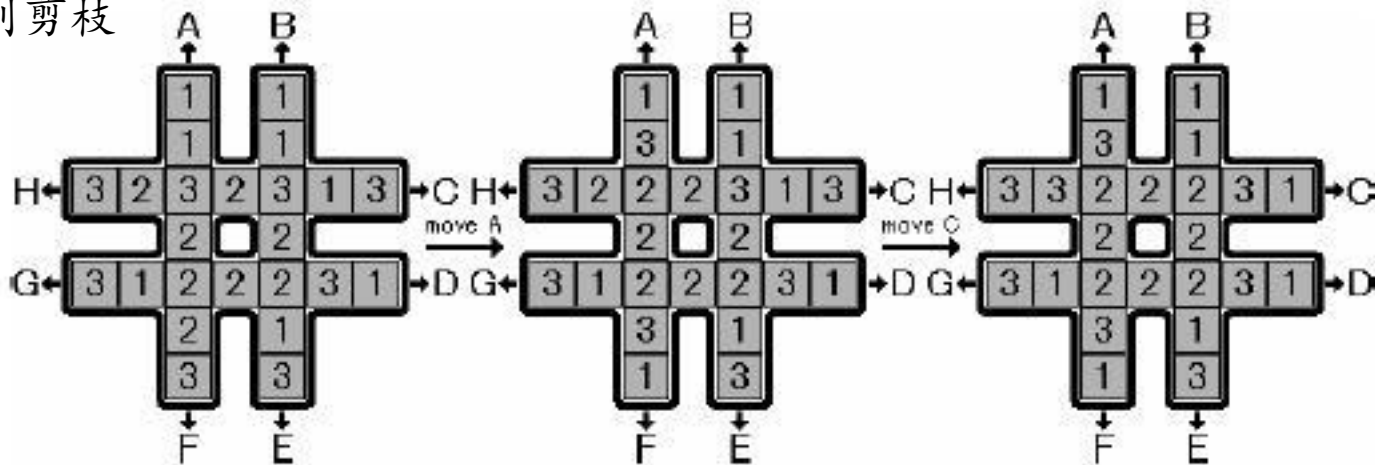


Fig.1



Alpha-Beta剪枝

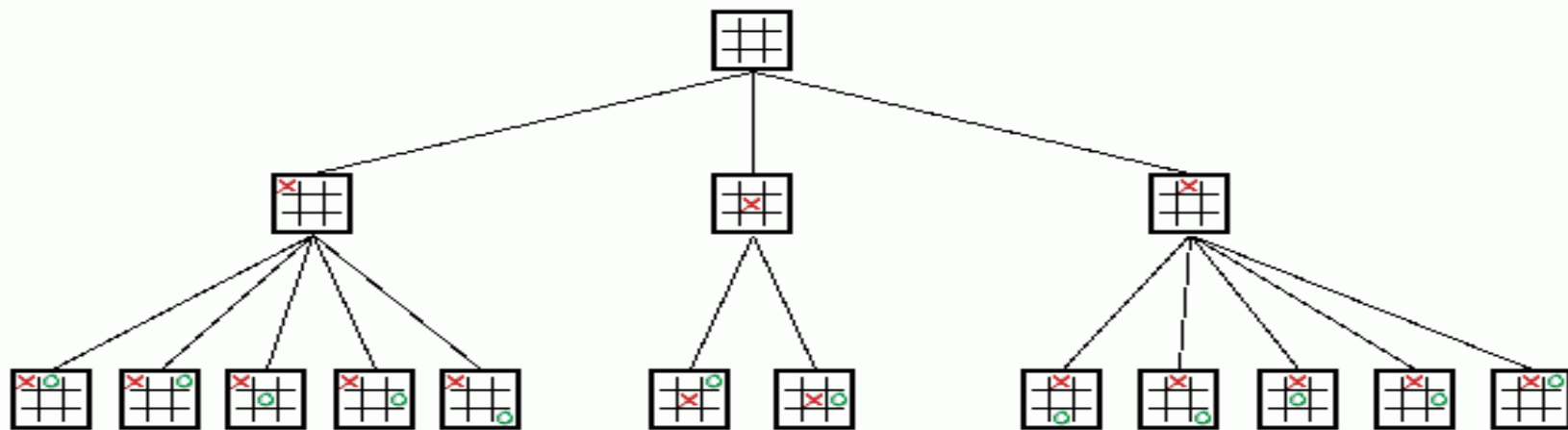
极大极小搜索法

在博弈搜索中，比如：围棋，五子棋，象棋等，结果有三种可能：胜利、失败和平局。

理论上可以穷举所有的走法，这就需要生成整棵博弈树。实际上不可行。因此搜索时可以限定博弈树的深度，到达该深度则不再往下搜，相当于只往前看 n 步。

极大极小搜索法

抢先连成三点一线的一字棋的三层博弈树：



极大极小搜索法

假设：MAX和MIN对弈，轮到MAX走棋了，那么我们会遍历MAX的每一个可能走棋方法，然后对于前面MAX的每一个走棋方法，遍历MIN的每一个走棋方法，然后接着遍历MAX的每一个走棋方法，……直到分出胜负或者达到了搜索深度的限制。若达到搜索深度限制时尚未分出胜负，则根据当前局面的形式，给出一个得分，计算得分的方法被称为估价函数，不同游戏的估价函数如何设计和具体游戏相关。

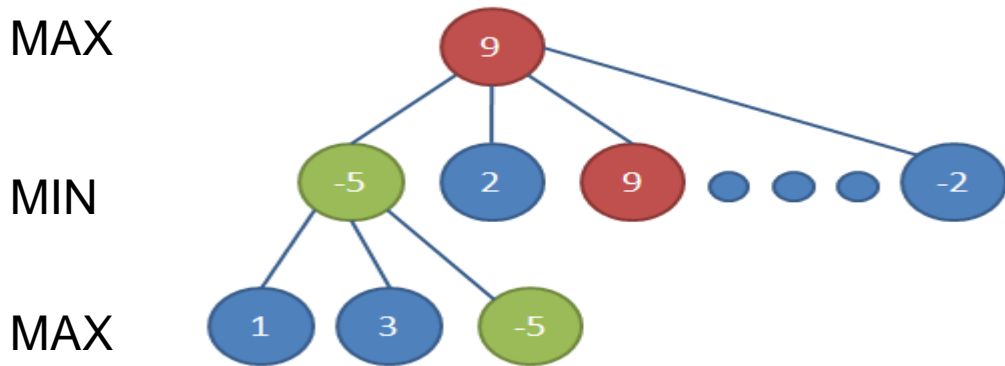
在搜索树中，轮到MAX走棋的节点即为极大节点，轮到MIN走棋的节点为极小节点。

极大极小搜索法

- 1) 确定估价值函数，来计算每个棋局节点的估价值。对MAX方有利，估价值为正，对MAX方越有利，估价值越大。对MIN方有利，估价值为负，对MIN方越有利，估价值越小。
- 2) 从当前棋局的节点要决定下一步如何走时，以当前棋局节点为根，生成一棵深度为n的搜索树。不妨总是假设当前棋局节点是MAX节点。
- 3) 用局面估价值函数计算出每个叶子节点的估价值
- 4) 若某个非叶子节点是极大节点，则其估价值为其子节点中估价值最大的那个节点的估价值
若某个非叶子节点是极小节点，则其估价值为其子节点中估价值最小的那个节点的估价值

极大极小搜索法

5) 选当前棋局节点的估价值最大的那个子节点，作为此步行棋的走法。



```
function minimax(node, depth) // 指定当前节点和还要搜索的深度
// 如果胜负已分或者深度为零，使用评估函数返回局面得分
if node is a terminal node or depth = 0
    return the heuristic value of node
// 如果轮到对手走棋，即node是极小节点，选择一个得分最小的走法
if the adversary is to play at node
    let  $\alpha := +\infty$ 
    foreach child of node
         $\alpha := \min(\alpha, \text{minimax}(\text{child}, \text{depth}-1))$ 
// 如果轮到自己走棋，是极大节点，选择一个得分最大的走法
else {we are to play at node}
    let  $\alpha := -\infty$ 
    foreach child of node
         $\alpha := \max(\alpha, \text{minimax}(\text{child}, \text{depth}-1))$ 
return  $\alpha$ ;
```

具体的做法：

int MinMax(int depth) { // 函数的评估都是以MAX方的角度来评估的

if (SideToMove() == MAX_SIDE)

return Max(depth);

else

return Min(depth);

}

int Max(int depth) {

int best = -INFINITY;

if (depth <= 0) return Evaluate();

GenerateLegalMoves();

while (MovesLeft()) {

MakeNextMove();

val = Min(depth - 1);

UnmakeMove();

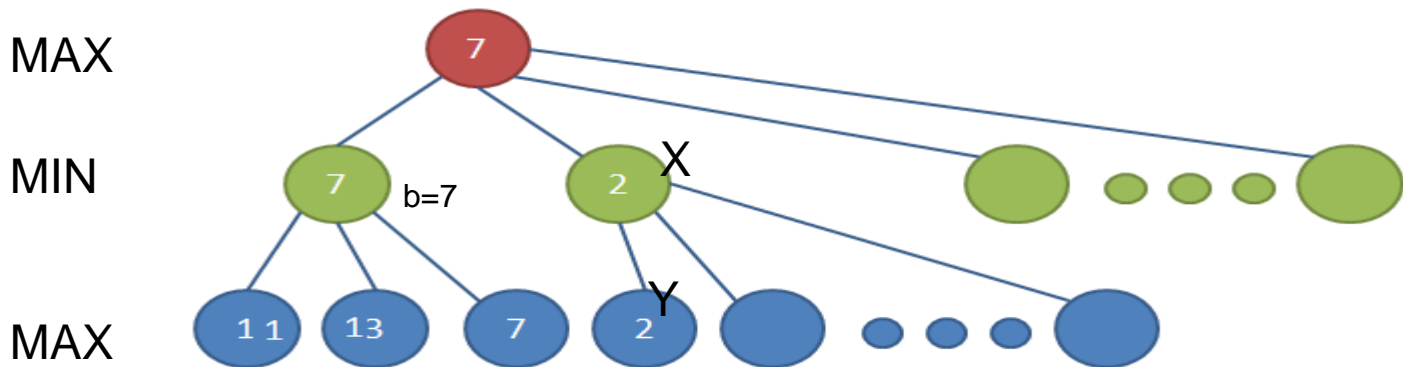
if (val > best) best = val;

}

return best; }

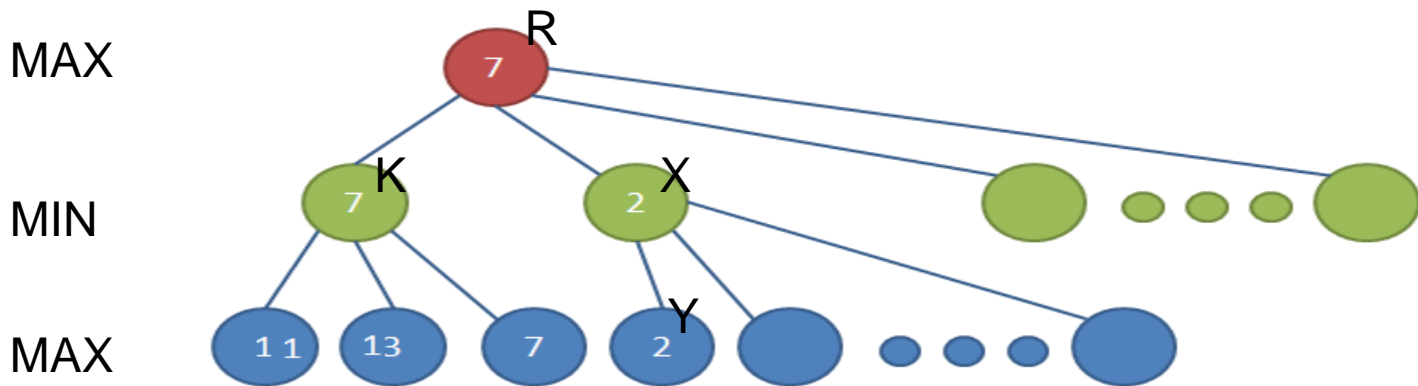
```
int Min(int depth) {  
    int best = INFINITY; // 注意这里不同于“最大”算法  
    if (depth <= 0)  
        return Evaluate();  
    GenerateLegalMoves();  
    while (MovesLeft()) {  
        MakeNextMove();  
        val = Max(depth - 1);  
        UnmakeMove();  
        if (val < best) // 注意这里不同于“最大”算法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

Alpha - beta剪枝



若结点x是Min节点，其兄弟节点(父节点相同的节点)中，已经求到的**最大估价值是b**（有些兄弟节点的估价值，可能还没有算出来），那么在对x的子节点进行考查的过程中，如果一旦发现某子节点的估价值 $\leq b$ ，则不必再考查后面的x的子节点了。

alpha剪枝



当搜索节点X时，若已求得某子节点Y的值为2，因为X是一个极小节点，那么X节点得到的值肯定不大于2。因此X节点的子节点即使都搜索了，X节点值也不会超过2。而节点K的值为7，由于R是一个Max节点，那么R的取值已经可以肯定不会选X的取值了。因此X节点的Y后面子节点可以忽略，即图中第三层没有数字的节点可被忽略。此即为alpha剪枝 ---- 因被剪掉的节点是极大节点。相应的也有beta剪枝，即被剪掉的节点是极小节点。

beta剪枝

若结点 x 是Max节点，其兄弟节点(父节点相同的节点)中，已经求到的最小估价值是 a （有些兄弟节点的估价值，可能还没有算出来）

那么在对 x 的子节点进行考查的过程中，如果一旦发现某子节点的估价值 $\geq a$ ，则不必再考查后面的 x 的子节点了。

```

function alphabeta(node, depth,  $\alpha$ ,  $\beta$ , maximizingPlayer)
    if depth = 0 or node is a terminal node
        return the heuristic value of node
    if maximizingPlayer
         $\alpha$  := -INF; //负无穷大
        for each child of node
             $\alpha$  := max( $\alpha$ , alphabeta(child, depth - 1,  $\alpha$ ,  $\beta$ ,
                                not(maximizingPlayer)))
            if  $\beta \leq \alpha$  //  $\beta$ 是node的兄弟节点到目前为止的最小估价值
                break // Beta cut-off, 剪掉剩下的极小节点
        return  $\alpha$ 
    else
         $\beta$  := INF; //无穷大
        for each child of node
             $\beta$  := min( $\beta$ , alphabeta(child, depth - 1,  $\alpha$ ,  $\beta$ ,
                                not(maximizingPlayer)))
            if  $\beta \leq \alpha$  //  $\alpha$ 是node的兄弟节点到目前为止的最大估价值
                break // Alpha cut-off, 剪掉剩下的极大节点
        return  $\beta$ 

```

初始调用: alphabeta(origin, depth, -infinity, +infinity, TRUE)

注解

- 红色字母处，随便写什么值都可以
- 在搜到底的情况下，infinity不一定是无穷大 infinity应该是主角赢的那个状态(胜负已分的状态)的估价值，而-infinity应该是主角输的那个状态(胜负已分的状态)的估价值。

Alpha - beta剪枝例题 POJ 1568

给定一个4 * 4 的四子连珠棋棋盘的局面，现在轮到x先走，问x下在哪里可以**必胜**

.....

. XO.

. OX.

.....

Alpha - beta剪枝例题 POJ 1568

在穷尽搜索，而非搜几层就停止的情况下，估价函数只有三种取值，正(inf)，0，负($-\text{inf}$)。分别代表己方胜，平，负。

必胜：

无论对手(B, o)怎么走，己方(A, x)最后都能找到获胜办法。不是自己无论怎么走，都能获胜。

此步行棋，若能找到一个走法，其对应的节点估价值为 inf ，则此为一个必胜的走法。

```
#include<iostream>
#include<cstdio>
#include<map>
#include<cstring>
#include<cmath>
#include<vector>
#include<queue>
#include<algorithm>
#include<set>
using namespace std;
```

//程序改编自网上博客源代码

```
#define inf 1<<30    //inf取1也可以
```

```
char str[5][5];
```

```
int X,Y,chess;
```

```
bool check(int x,int y){    //...略    }
```

//判断一个局面是否是棋局结束的局面

```
int MinSearch(int x,int y,int alpha);
```

```
int MaxSearch(int x,int y,int beta);
```

```

int MinSearch(int x,int y, int alpha)
{ /*本节点是A刚下完，刚刚下在了(x,y)该轮到B下的节点
MinSearch的返回值 是B最佳走法的估价值。具体办法，枚举本步所有B走法，对每种走法用
MaxSearch求其估价值，返回估价值最小的那个走法(B的选择)的估价值。
参数alpha，表示本步（本节点）的兄弟节点里面，已经算出来的最大估价值。
如果枚举走法的过程中，发现有 MaxSearch的值如ans已经小于等于alpha,那么立即停止
枚举，返回 ans,表示本节点的估价值就是ans
如果 ans < alpha, 那么本节点就不会被自己选择 */
    int ans = inf;
    if( check(x,y)) //自己刚走了一步，如果棋局分出胜负，那么定是自己胜利
        return inf;
    if( chess == 16 )
        return 0; //平局
    for( int i = 0;i < 4; ++i)
        for( int j = 0; j < 4; ++ j ) {
            if( str[i][j] == '.' ) {
                str[i][j] = 'o'; ++ chess;
                int tmp = MaxSearch(i,j,ans); //此处的
ans是beta 值，即在本节点的各个子节点中，目前已经找到的最小的 估价值
                str[i][j] = '.'; -- chess;
                ans = min(ans,tmp);
            }
        }
}

```

```

        if( ans <= alpha ) {
            return ans;
        }
    }
    return ans;
}

```

```

int MaxSearch(int x,int y, int beta)

```

/*本节点是B刚下完,下在(x,y),该轮到A下的节点

在MaxSearch中,要返回本步A最佳走法的估价值。具体办法,枚举本步所有走法,对每种走法用 MinSearch求其估价值,返回估价值最大的那个走法的估价值。

beta表示本步的兄弟节点中,已经算出来的最小的估价值。

如果枚举走法的过程中,发现有 MinSearch的值如ans已经大于等于 beta,那么立即停止枚举,返回 ans,表示本节点的估价值就是ans

如果 ans > beta,那么本节点就不会被B选择 */

```

    int ans = -inf; //本节点展开的各子节点中,最大的估价函数值
    if( check(x,y) )
        return -inf;

```



```

    if( chess == 16 )
        return 0;
    for( int i = 0; i < 4; ++ i)
        for( int j = 0; j < 4; ++ j ) {
            if( str[i][j] == '.' ) {
                str[i][j] = 'x'; ++chess;
                int tmp  = MinSearch(i,j,ans );
                str[i][j] = '.'; --chess;
                ans = max(tmp,ans);
                if( ans >= beta )
                    return ans;
            }
        }
    return ans;
}

```

//ans是 beta值，即在本节点的各个子节点中，目前找到的最大的估价值

```
bool Solve()
{
    int alpha = -inf;
    for(int i = 0; i < 4; ++ i)
        for( int j = 0; j < 4; ++j ) {
            if( str[i][j] == '.' ) {
                str[i][j] = 'x'; ++ chess;
                int tmp = MinSearch(i,j,alpha);
                str[i][j] = '.'; -- chess;
                if( tmp == inf ) {
                    X = i;
                    Y = j;
                    return true;
                }
            }
        }
    return false;
}
```

```
int main() {
    char ch[5];
    while (scanf("%s", ch) != EOF && ch[0] != '$') {
        chess = 0;
        for (int i = 0; i < 4; i++) {
            scanf("%s", str[i]);
            for (int j = 0; j < 4; j++)
                chess += str[i][j] != '.';
        }
        if (chess <= 4) { //这一步直接从2s+到0ms
            printf("#####\n");
            continue;
        }
        if (Solve()) printf("(%d,%d)\n", X, Y);
        else printf("#####\n"); //无必胜走法
    }
    return 0;
}
```

例：2012金华邀请赛Problem F. Chess

给定4*4棋盘上四子棋的局面（至少已经有6个子），现在轮到Bob下。

Bob可能执 X 也可能执 O,
问Bob是否能做到必胜(想赢就赢),
或问Bob是否能做到想败就一定败
或问Bob是否能做到想和就一定和

例：2012金华邀请赛Problem F. Chess

误区：采用和POJ1568一样的估价函数定义来做此题。

○必能败，等价于 X 必胜吗？

○必能败 等价于 X 不是必败，且不是必和 吗？

○必能和 等价于 X 不是必胜，也不是必败 吗？

例：2012金华邀请赛Problem F. Chess

误区：采用和POJ1568一样的估价函数定义来做此题。

○必能败，等价于 X 必胜吗？ 否

○必能败 等价于 X 不是必败，且不是必和 吗？ 否

○必能和 等价于 X 不是必胜，也不是必败 吗？ 否

例：2012金华邀请赛Problem F. Chess

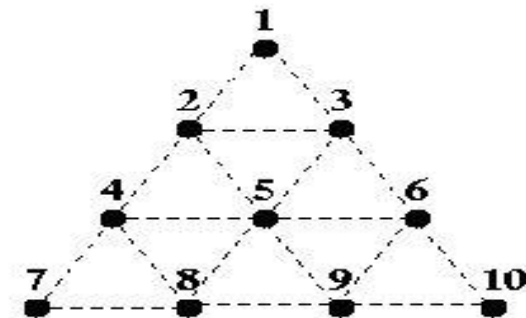
误区：采用和POJ1568一样的估价函数定义来做此题。

O 必能如何，无法等价于 X 必能或不必能怎么样的组合

所以，应根据Bob是X还是O,以及Bob想怎么样，改变估价函数的定义。比如，若Bob是O,且Bob想输，那么和棋终局或是Bob赢的终局，估价函数值都是 $-\infty$ ，Bob输的终局，估价函数值都是 ∞

例：三角形大战 (POJ1085)

- 两个游戏者轮流填充左边的虚线三角形。每次只能填充一条短边。
- 若某游戏者填充一条短边之后组成了一个“小三角形”，则该游戏者“拥有”这个三角形，并且可以继续填充。
- 当所有边都被填充之后，拥有三角形数目多的游戏者获胜。
- 给定局面，问谁能赢



- 博弈的思路
 - alpha-beta剪枝
 - 估价函数：A拥有的三角形数 - B拥有的三角形数