lesson6_Backtracking

1.reason

枚举法：make a list of all candidate answers, examine each

适用candidates有限，且不是很多的情况

backtracking(回溯法) : eliminate the explicit examination of a large subset of the candidates（能够消除对大部分候选子集的显式检查）and find the answer

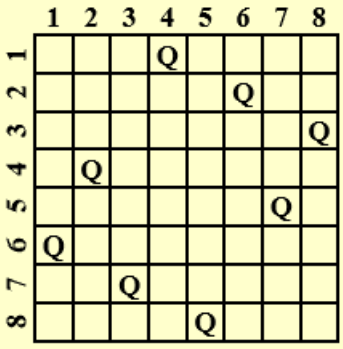
eliminate ~ pruning(剪枝) : cut the whole branch of some tree

2.basic idea

The basic idea is that suppose we have a partial solution $(x_1, \dots, x_i)$ where each $x_k \in S_k$ for $1 \le k \le i < n$. First we add $x_{i+1} \in S_{i+1}$ and check if $(x_1, \dots, x_i, x_{i+1})$ satisfies the constrains. If the answer is "yes" we continue to add the next x, else we delete x_i and backtrack to the previous partial solution $(x_1, \dots, x_{i-1})$.

3.eg1 八皇后问题

Find a placement of 8 queens on an 8 * 8 chessboard such that no two queens attack.(同一行、列、对角线)



Q_i ::= queen in the i-th row

x_i ::= the column index in which Q_i is

Solution = $(x_1, x_2, \dots, x_8)$
= $(4, 6, 8, 2, 7, 1, 3, 5)$

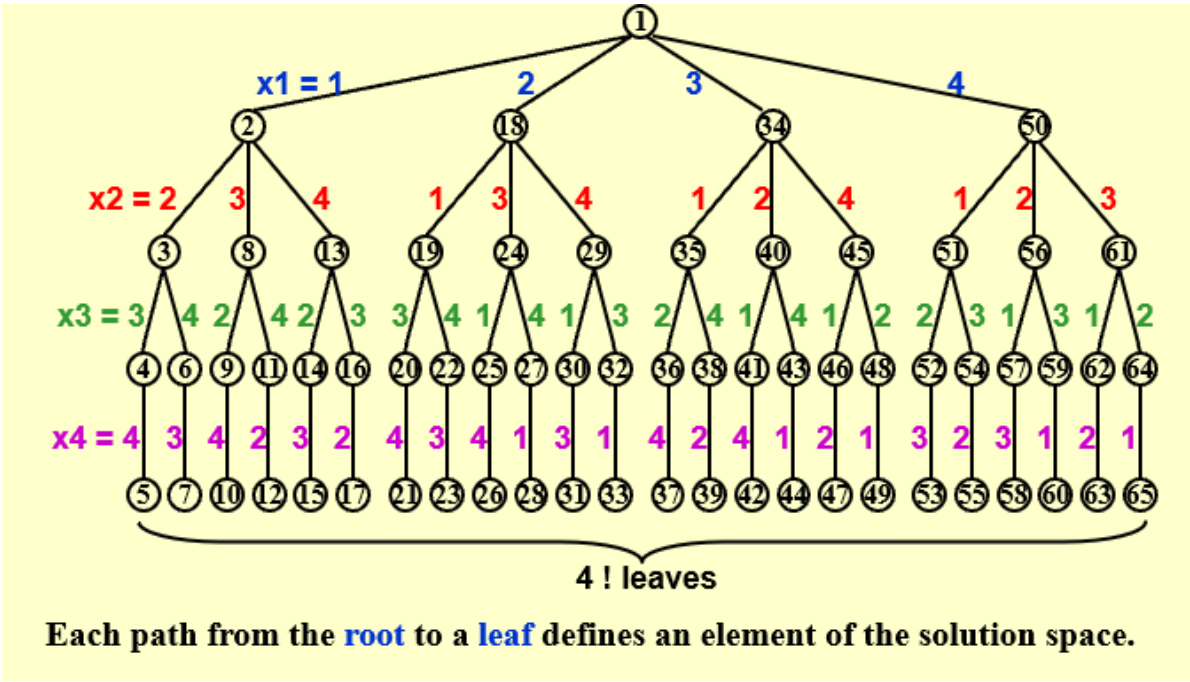
Constrains: ① $S_i = \{1,2,3,4,5,6,7,8\}$ for $1 \le i \le 8$

② $x_i \neq x_j$ if $i \neq j$ ③ $(x_i - x_j) / (i - j) \neq \pm 1$

candidates：N!

backtracking eg: 4 queens

step1:construct a game tree



step2:DFS(深度优先、先序遍历 节点数字是check顺序)

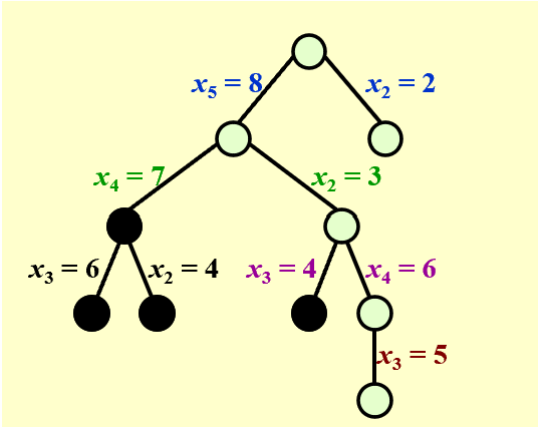
违反规则就剪枝，孩子都死了就剪枝；

因为没有访问所有的结点，所以节省了时间；不需要真的构建树，game tree只是个概念；

4.eg2 Turnpike Reconstruction (收费公路重建)

Given N points on the x-axis with coordinates x1 < x2 < ...< xN . Assume that x1 = 0. There are N (N – 1) / 2 distances between every pair of points.

Given N (N – 1) / 2 distances. Reconstruct a point set from the distances.



eg: Given D = { 1, 2, 2, 2, 3, 3, 3, 4, 5, 5, 5, 6, 7, 8, 10 }

Step 1: N (N – 1) / 2 = 15 implies N = 6

Step 2: x1 = 0 and x6 = 10

Step 3: find the next largest distance and check

可以用递归实现，如果no,恢复删去的边；

```
bool Reconstruct ( DistType X[ ], DistSet D, int N, int left, int right )
```

```
{ /* X[1]...X[left-1] and X[right+1]...X[N] are solved */
/*left,right规定未解决的问题的范围*/
/*初始化*/
    bool Found = false;
/*结束遍历条件*/
    if ( Is_Empty( D ) ) /*边的集合空*/
        return true; /* solved */
/*找到合适遍历点*/
    D_max = Find_Max( D );/*找到最大边*/
    /* option 1: X[right] = D_max */
    /* check if |D_max-X[i]|∈D is true for all X[i]'s that have been solved */
 /*判断是否违背条件，不违背再往下搜索，否则尝试另一可能性*/
    OK = Check( D_max, N, left, right ); /* pruning */
    if ( OK ) { /* add X[right] and update D */
        X[right] = D_max;
        /*在集合中删去加入节点后产生的边*/
        /*减少遍历的元素，更新遍历集合*/
        for ( i=1; i<left; i++ )  Delete( |X[right]-X[i]|, D);
        for ( i=right+1; i<=N; i++ )  Delete( |X[right]-X[i]|, D);
        /*继续found*/
        Found = Reconstruct ( X, D, N, left, right-1 );
        /*如果后续找不到解决方法，撤回删去对应边的操作*/
        if ( !Found ) {
            for ( i=1; i<left; i++ )  Insert( |X[right]-X[i]|, D);
            for ( i=right+1; i<=N; i++ )  Insert( |X[right]-X[i]|, D);
        }
    }
    /* finish checking option 1 */
if ( !Found ) { /* if option 1 does not work */
        /* option 2: X[left] = X[N]-D_max */
        OK = Check( X[N]-D_max, N, left, right );
        if ( OK ) {
            X[left] = X[N] - D_max;
            for ( i=1; i<left; i++ )  Delete( |X[left]-X[i]|, D);
            for ( i=right+1; i<=N; i++ )  Delete( |X[left]-X[i]|, D);
            Found = Reconstruct (X, D, N, left+1, right );
            if ( !Found ) {
                for ( i=1; i<left; i++ ) Insert( |X[left]-X[i]|, D);
                for ( i=right+1; i<=N; i++ ) Insert( |X[left]-X[i]|, D);
            }
        }
        /* finish checking option 2 */
    } /* finish checking all the options */

    return Found;
}
```

backtracking （整体思想）

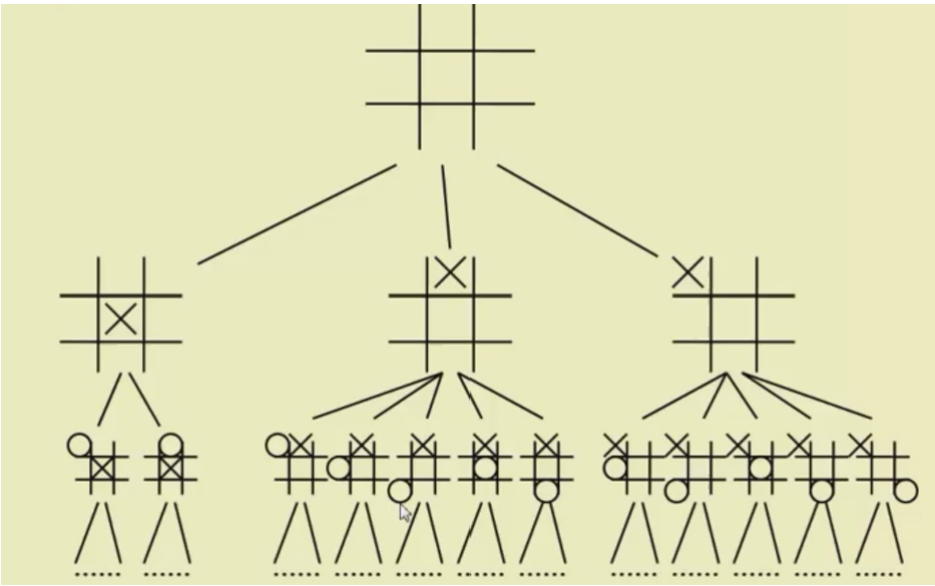
```
bool Backtracking ( int i )
{   Found = false;
    if ( i > N )
        return true; /* solved with (x1, …, xN) */
    for ( each xi ∈Si ) { /*factor1*/
        /* check if satisfies the restriction R */
        OK = Check((x1, …, xi) , R ); /* pruning */ /*factor2*/
```

```
        if ( OK ) {/*factor3*/
            Count xi in;
            Found = Backtracking( i+1 );
            if ( !Found )
                Undo( i ); /* recover to (x1, …, xi-1) */
        }
        if ( Found ) break;
    }
    return Found;
}
```

factor: 1.candidates的规模 2.check设置的好坏 3.有多少ok的情况

Smaller Si first ? 一棵树的孩子数量越多，一次剪掉的很少，剪枝效率越低；2 children > 3 children

5.a game tic tac toe



step1: 3 options (考虑对称性，所以不是9)

step2: 根据上一步，新的选项数不同

分析：3^9种棋局 （每个格子X or O or NULL）； 9！种放置序列

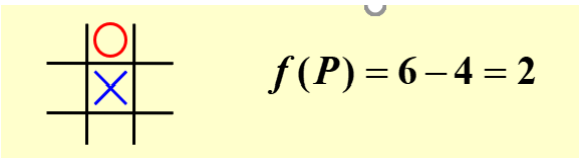
Minimax Strategy

f (P) = W computer - W human;

W:赢的可能数

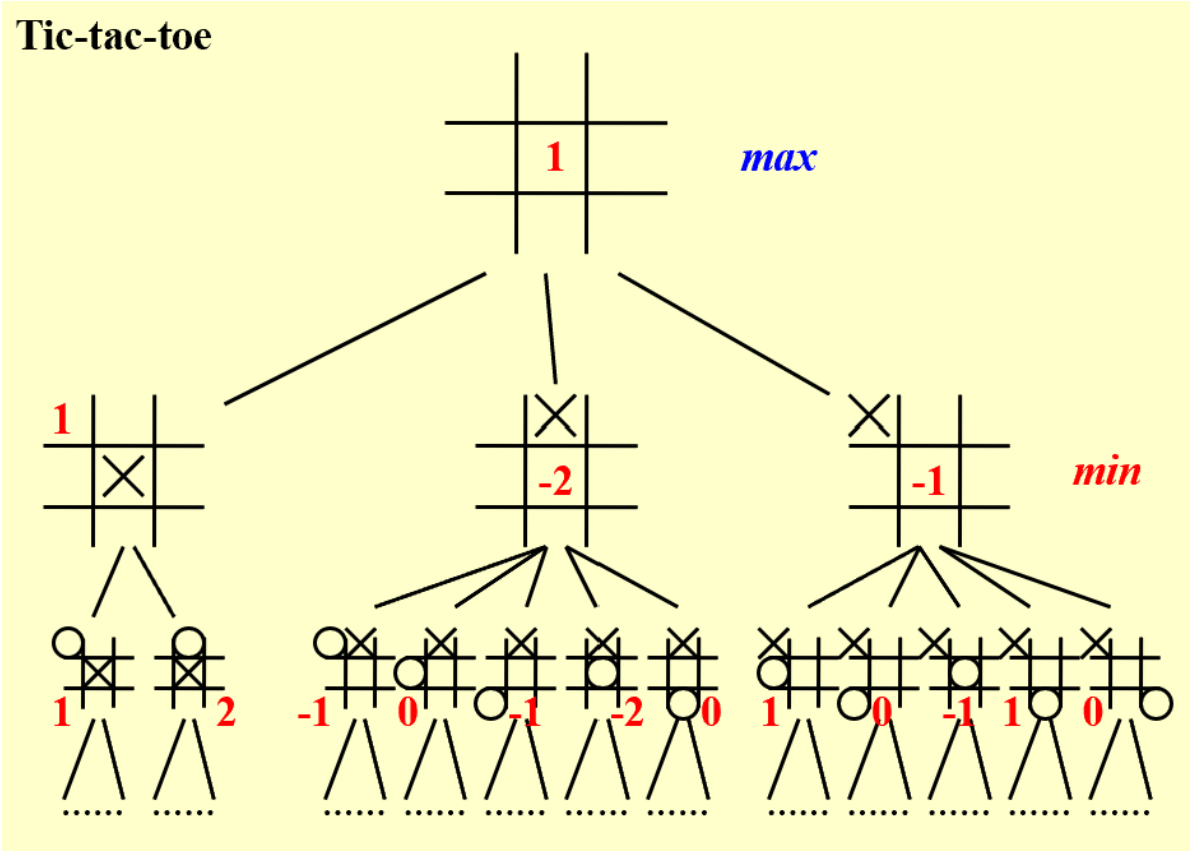
W computer = 6

W human = 4



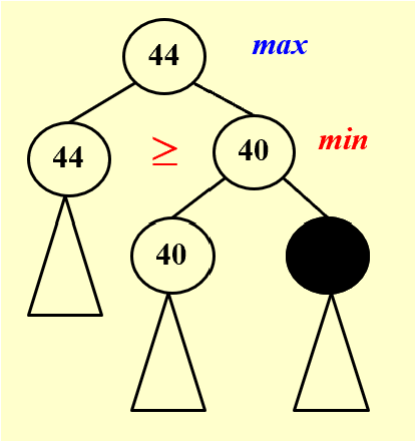
The human is trying to minimize the value of the position P, while the computer is trying to maximize it.

先往后看两步：

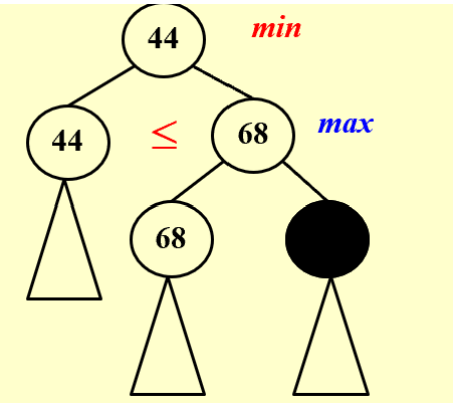


剪枝：

α pruning：



β pruning：



α-β pruning: when both techniques are combined. In practice, it limits the searching to only **O(N^0.5) nodes**, where N is the size of the full game tree.