Backtracking

Rationale of the Backtracking Algorithms

A sure-fire way to find the answer to a problem is to make a list of all candidate answers, examine each, and following the examination of all or some of the candidates, declare the identified answer.

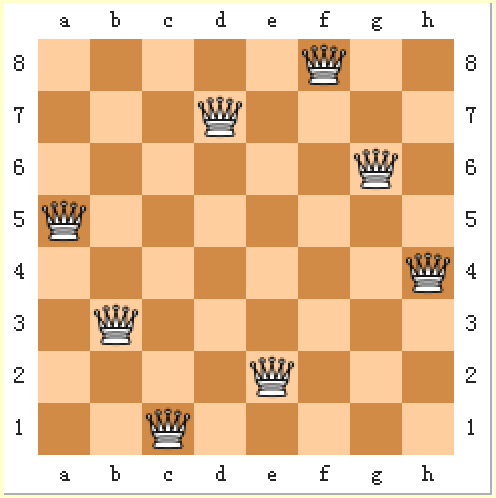
Backtracking enables us to eliminate the explicit examination of a large subset of the candidates while still guaranteeing that the answer will be found if the algorithm is run to termination.

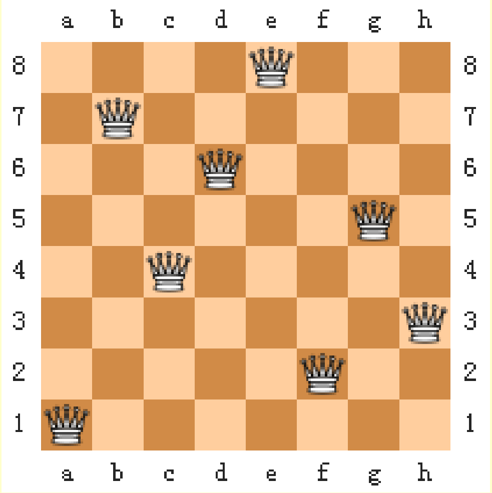
The basic idea is that suppose we have a partial solution $(x_1, \dots , x_i)$ where each $x_k \in S_k$ for $1 \le k \le i < n$. First we add $x_{i+1} \in S_{i+1}$ and check if $(x_1, \dots , x_i, x_{i+1})$ satisfies the constrains. If the answer is "yes" we continue to add the next x, else we delete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

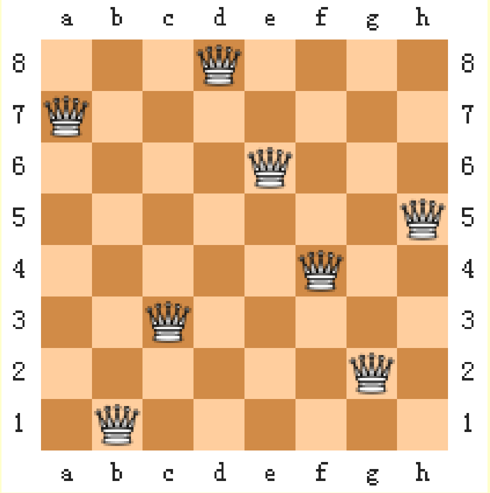
Eight Queens

Find a placement of 8 queens on an 8×8 chessboard such that no two queens attack.

Two queens are said to attack iff they are in the same row, column, diagonal, or antidiagonal of the chessboard.







	1	2	3	4	5	6	7	8
1				Q				
2						Q		
3								Q
4		Q						
5							Q	
6	Q							
7			Q					
8					Q			

Q_i ::= queen in the i-th row

x_i ::= the column index in which Q_i is

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

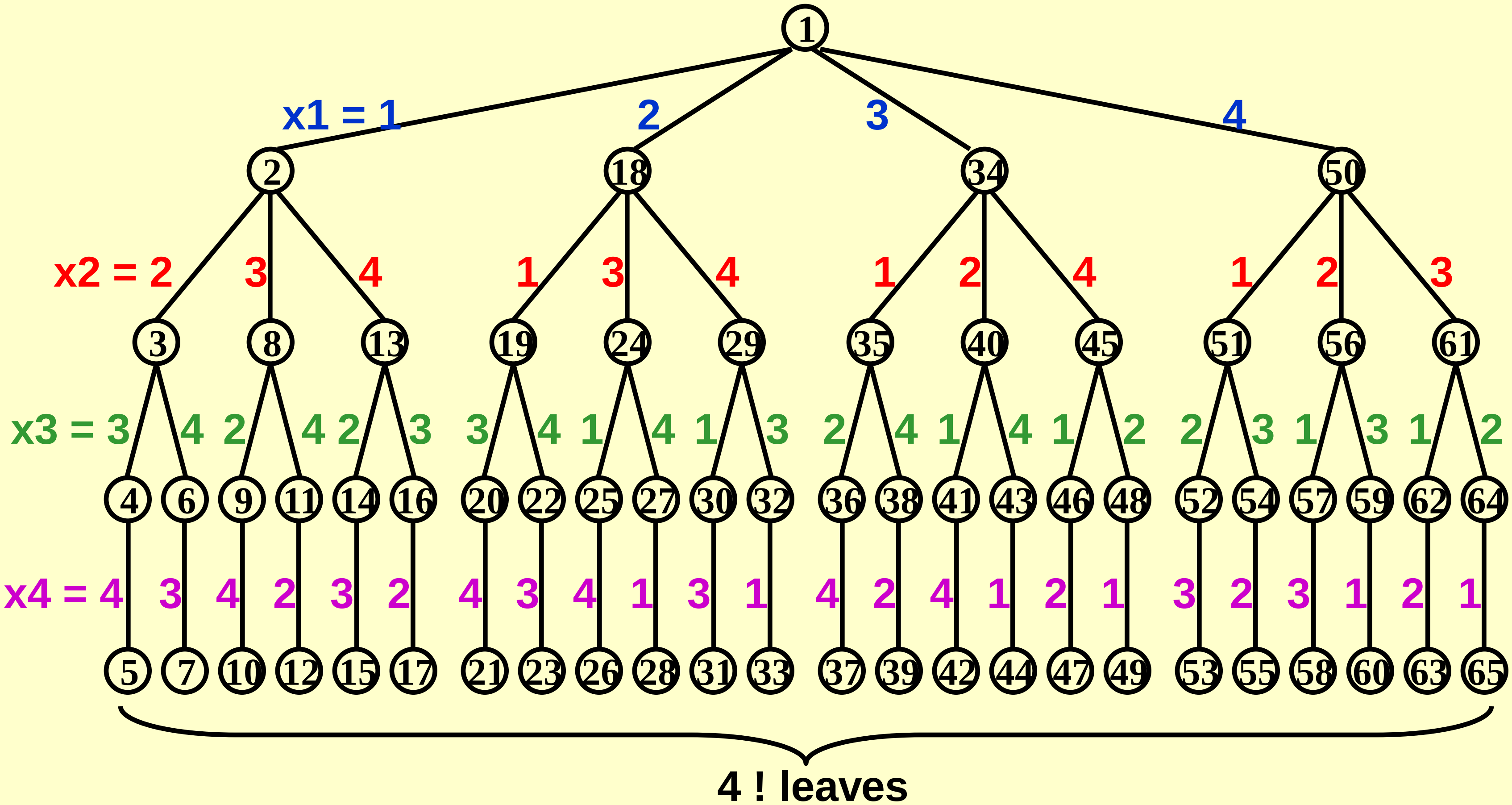
Constrains: ① $S_i = \{ 1,2,3,4,5,6,7,8 \}$ for $1 \le i \le 8$

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

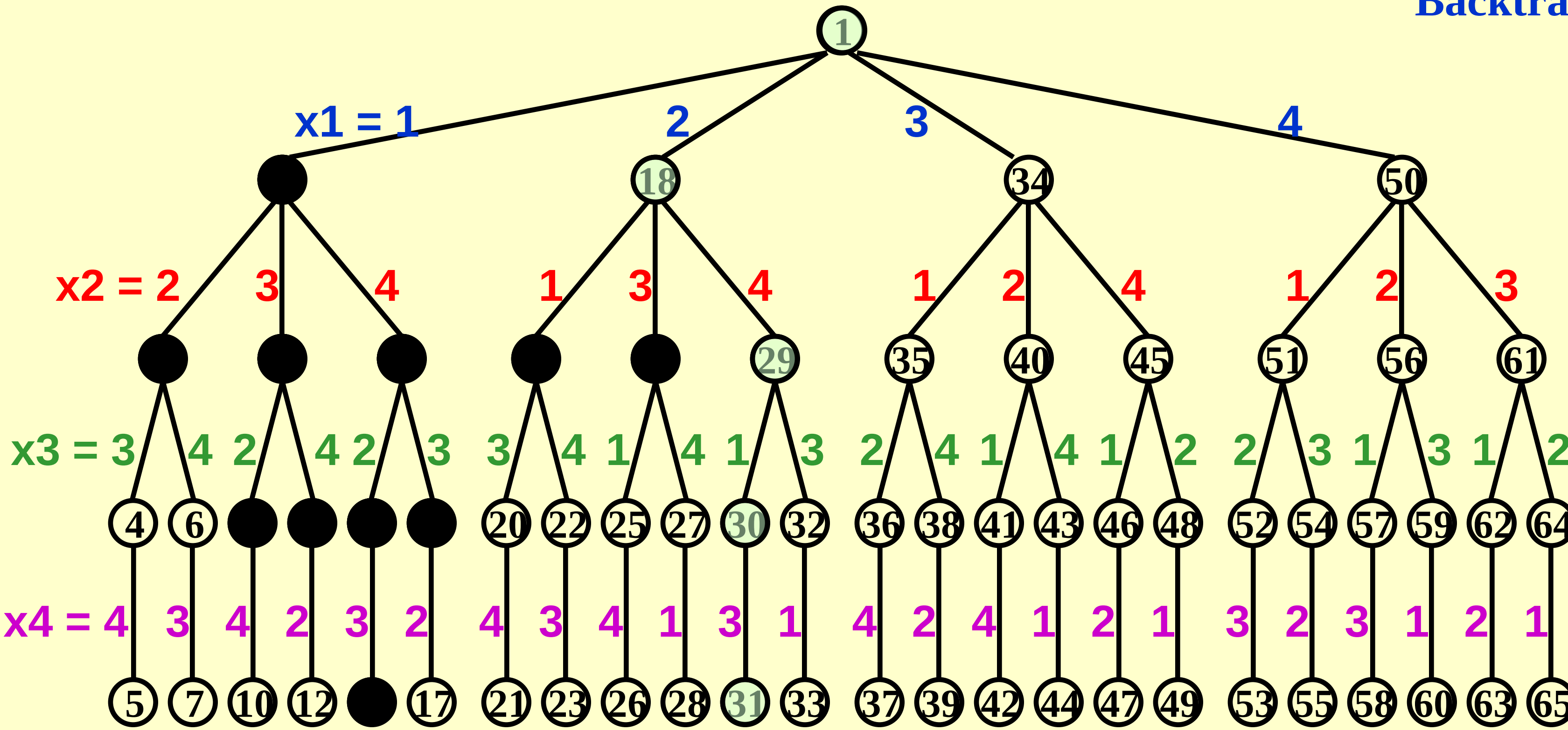
For the problem with *n* queens, there are *n*! candidates in the solution space.

Method: **Take the problem of 4 queens as an example**

Step 1: **Construct a game tree**



Each path from the root to a leaf defines an element of the solution space.



Step 2: Perform a **depth-first** search (post-order traversal) to examine the paths

(2, 4, 1, 3)

	Q		
			Q
Q			
		Q	

Note: No tree is actually constructed. The game tree is just an abstract concept.

The Turnpike Reconstruction Problem

Given N points on the x-axis with coordinates $x_1 < x_2 < \ldots < x_N$. Assume that $x_1 = 0$. There are $N (N - 1) / 2$ distances between every pair of points.

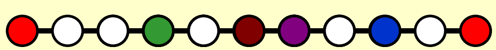
Given $N (N - 1) / 2$ distances. Reconstruct a point set from the distances.

〖Example〗 Given $D = \{ 1, 2, 2, 2, 3, 3, 3, 4, 5, 5, 5, 6, 7, 8, 10 \}$

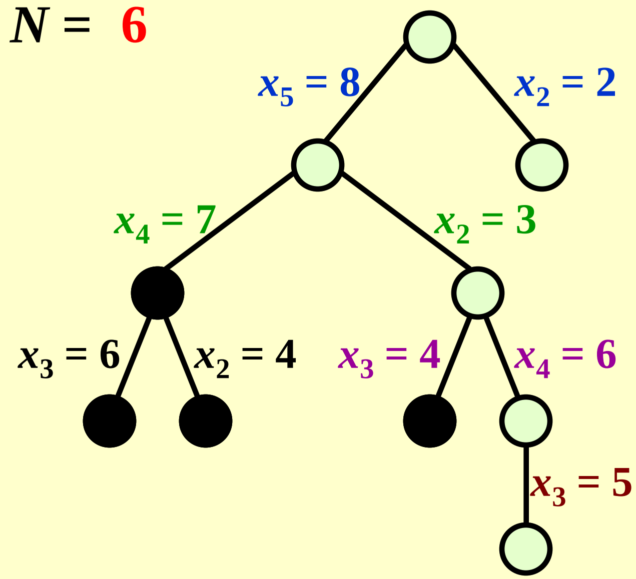
Step 1: $N (N - 1) / 2 = 15$ implies $N = 6$

Step 2: $x_1 = 0$ and $x_6 = 10$

Step 3: find the next largest distance and check



$(0, 3, 5, 6, 8, 10)$



```
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 */
    bool Found = false;
    if ( Is_Empty( D ) )
        return true; /* solved */
    D_max = Find_Max( D );
    /* option 1：X[right] = D_max */
    /* check if |D_max-X[i]|∈D is true for all X[i]'s that have been solved */
    OK = Check( D_max, N, left, right ); /* pruning */
    if ( OK ) { /* add X[right] and update D */
        X[right] = D_max;
        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);
        Found = Reconstruct ( X, D, N, left, right-1 );
        if ( !Found ) { /* if does not work, undo */
            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 */
```

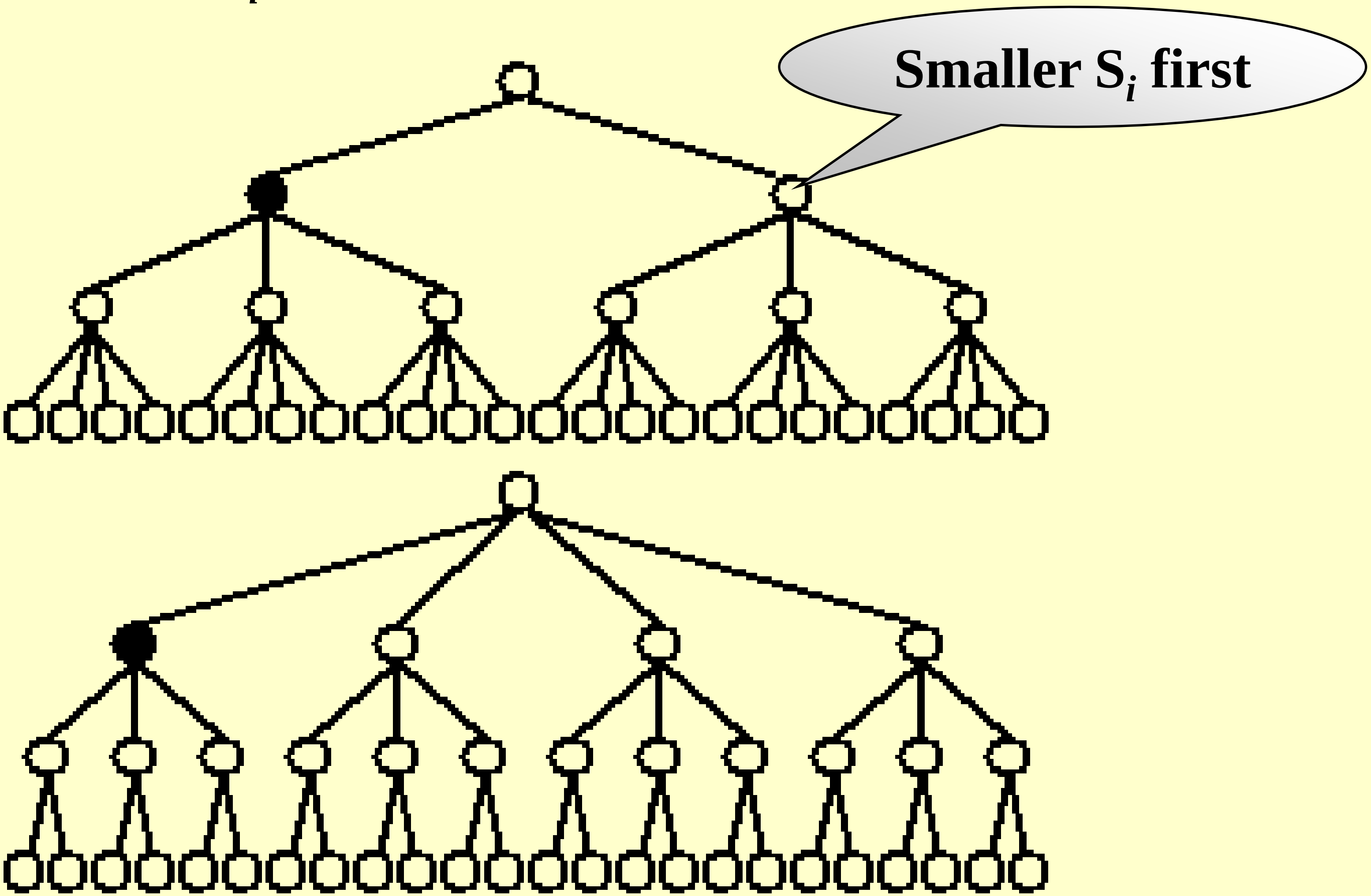
```
    if ( !Found ) { /* if option 1 does not work */
        /* option 2: X[left] = X[N]-D_max */
        OK = Check( X[N]-D_max, N, left, right );
        if ( OK ) {
            X[left] = X[N] – D_max;
            for ( i=1; i<left; i++ )  Delete( |X[left]-X[i]|, D);
            for ( i=right+1; 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;
}
```

A Template

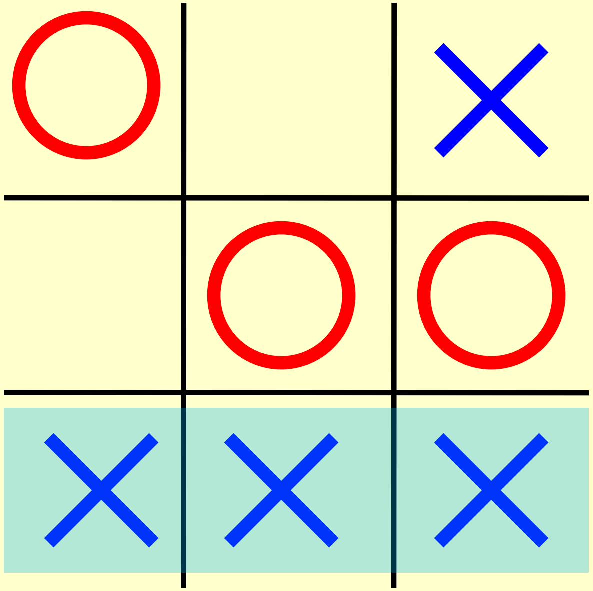
```
bool Backtracking ( int i )
{   Found = false;
    if ( i > N )
        return true; /* solved with (x1, …, xN) */
    for ( each xi ∈ Si ) {
        /* check if satisfies the restriction R */
        OK = Check((x1, …, xi) , R ); /* pruning */
        if ( OK ) {
            Count xi in;
            Found = Backtracking( i+1 );
            if ( !Found )
                Undo( i ); /* recover to (x1, …, xi-1) */
        }
        if ( Found ) break;
    }
    return Found;
}
```

When different S_i's have different sizes



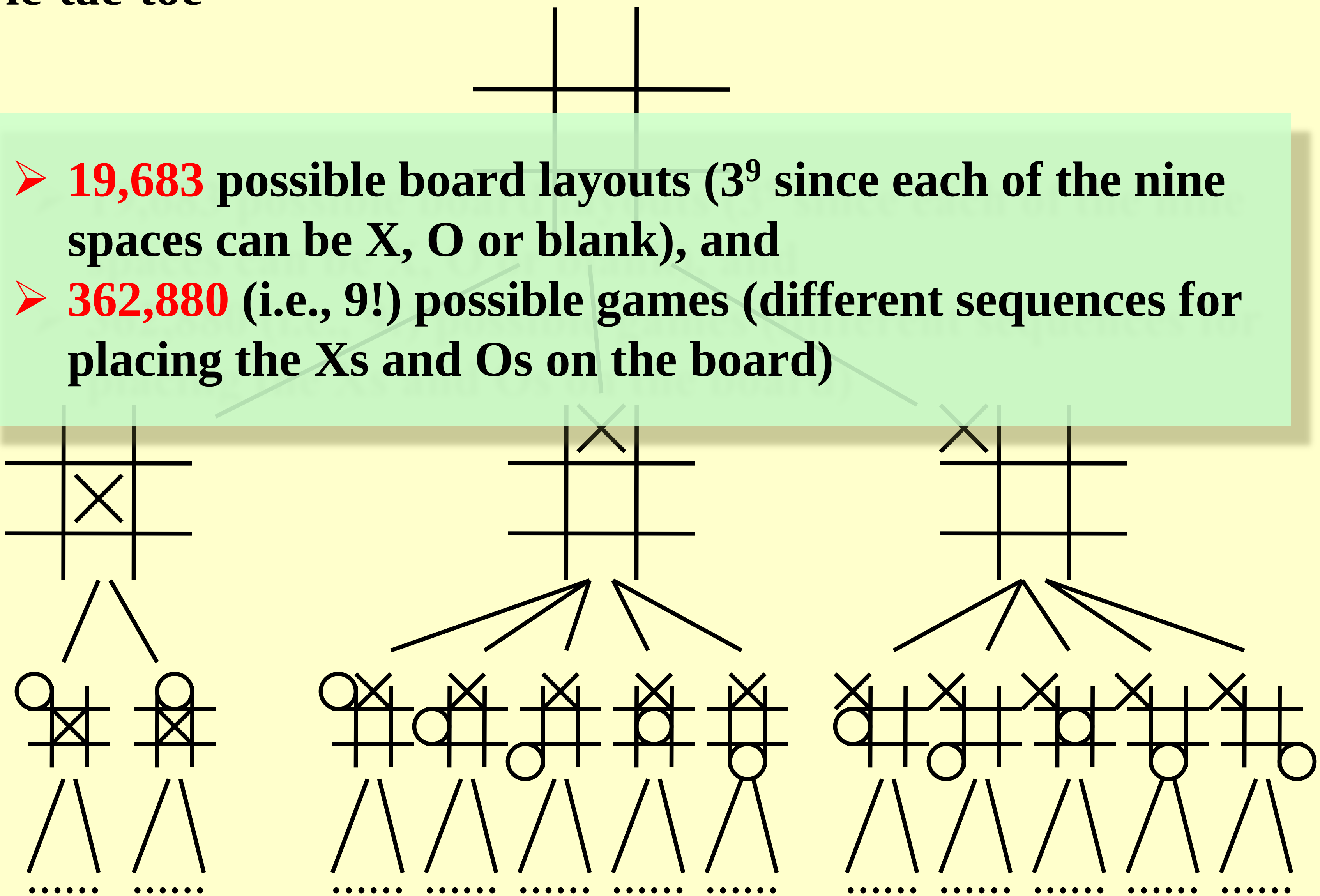
Games – *how did AlphaGo win*

Tic-tac-toe



The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.

Tic-tac-toe



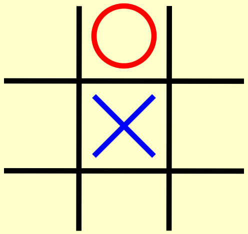
- **19,683** possible board layouts (3^9 since each of the nine spaces can be X, O or blank), and
- **362,880** (i.e., 9!) possible games (different sequences for placing the Xs and Os on the board)

Tic-tac-toe: Minimax Strategy

Use an evaluation function to quantify the "goodness" of a position. For example:

$$f(P) = W_{Computer} - W_{Human}$$

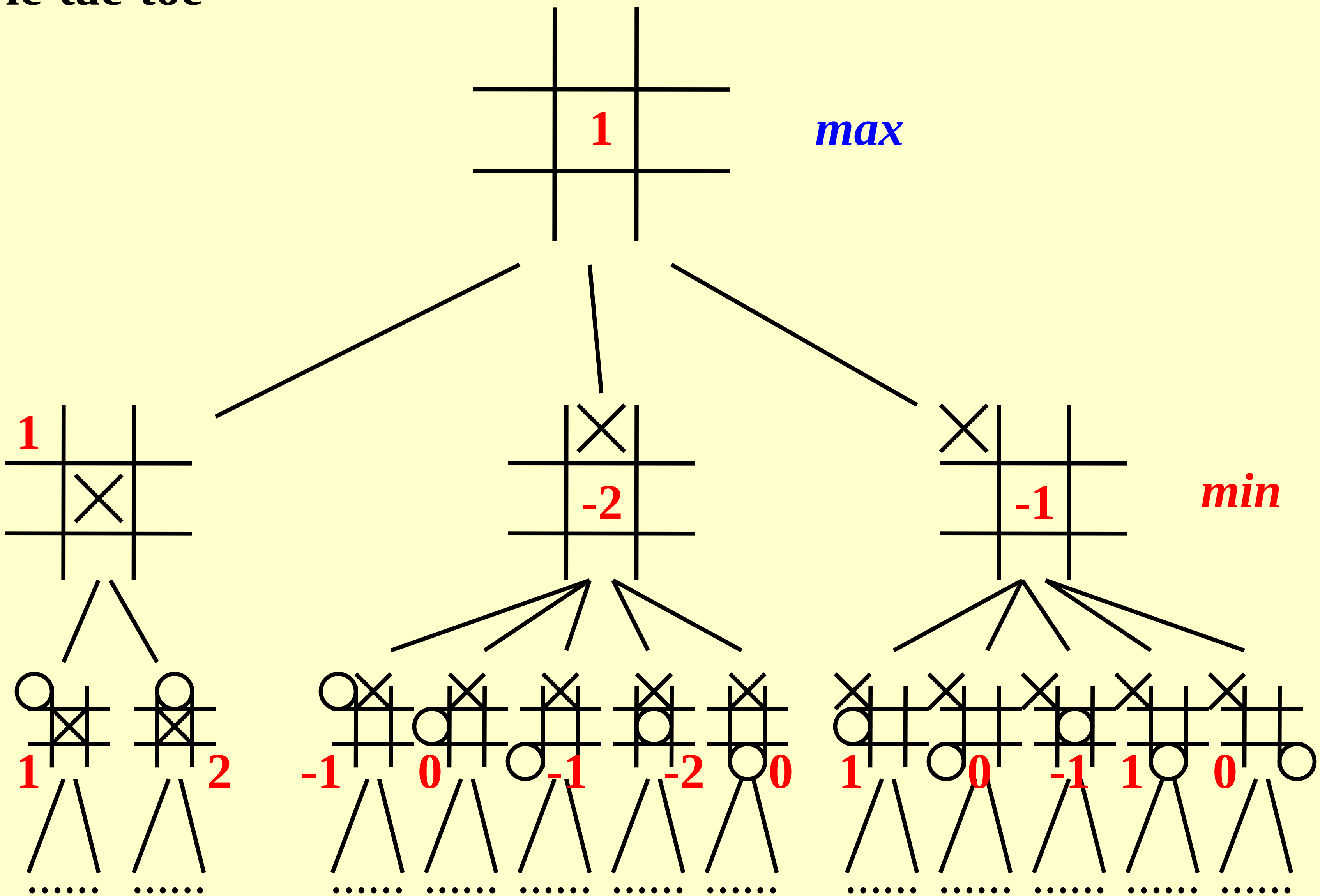
where W is the number of potential wins at position P.



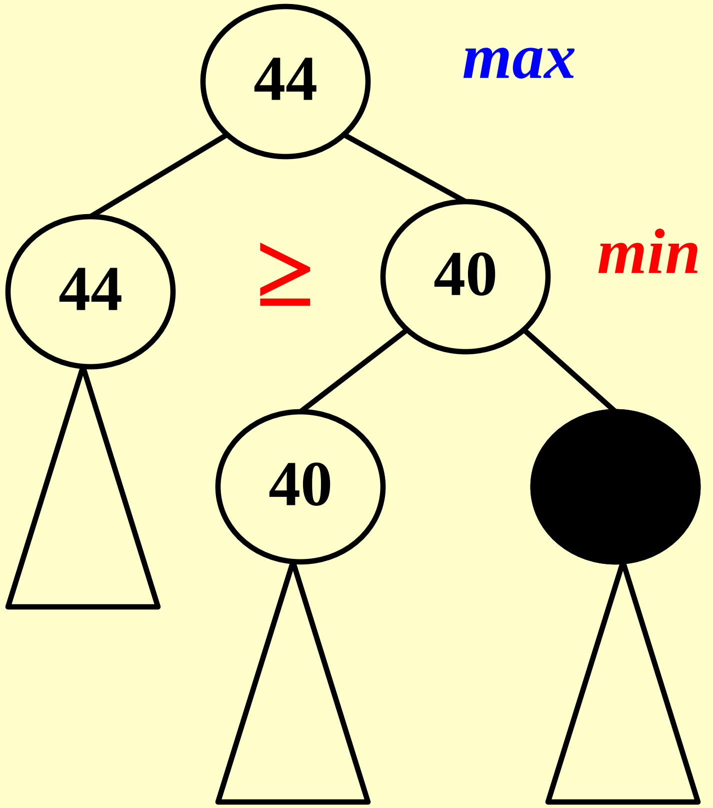
$$f(P) = 6 - 4 = 2$$

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

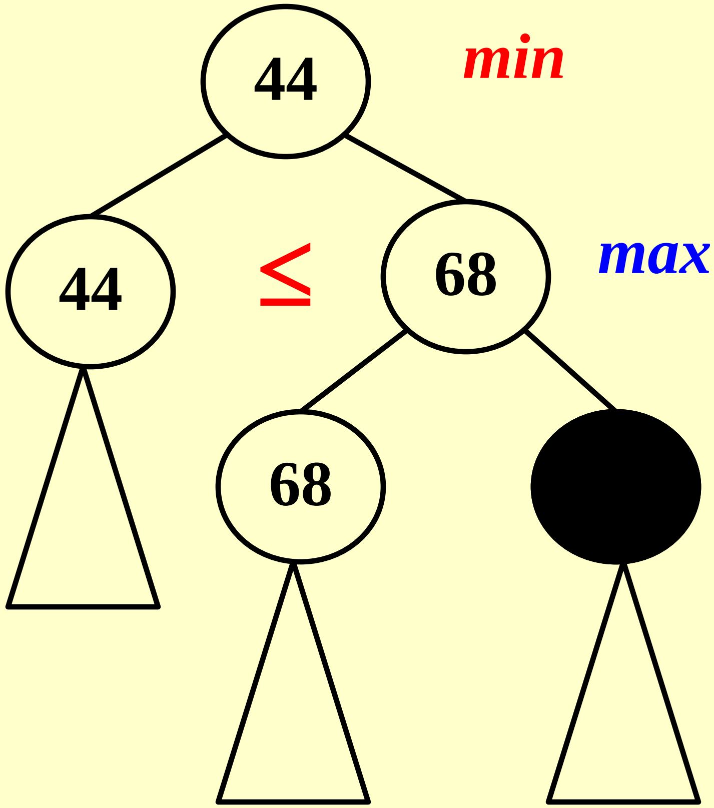
Tic-tac-toe



α pruning



β pruning



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

Reference:

Data Structure and Algorithm Analysis in C (2nd Edition)：Ch.10，p.403-414；*M.A.Weiss著、陈越改编，人民邮件出版社，2005*