CHAPTER 6

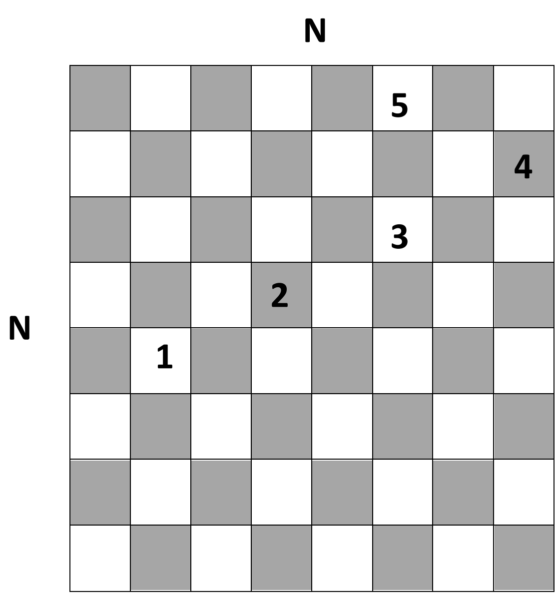
Back Tracking Procedures

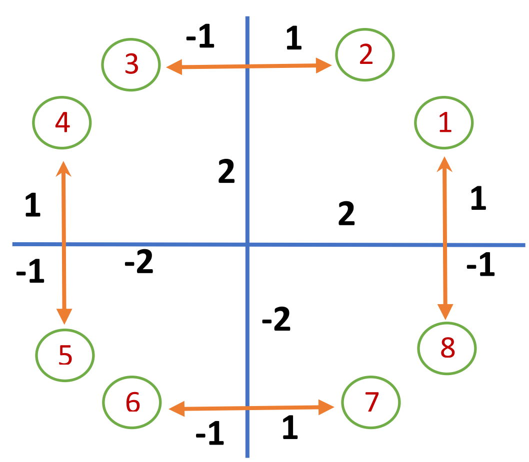
- Algorithms for finding solutions to specify problems, not by following a fixed rule of computation, but by trial and error.

Example:

Knight's tour

- The problem is to find if the knight can tour entire N*N board by visiting every field in the board exactly once.
- Starting from one point.
- The problem can be reduced from converting N2 fields to the problem of either performing the next move or finding out that none is possible.





```
Algorithm tryNextMove
Begin
    Initialize selection of moves
    Repeat
        Select next candidate move
        If accepted then
            Record the move
            If board is not full then
                tryNextMove
                If not successful then
                    Erase previous recording
                End if
            Else
                Successful = true;
            End if
        End if
    Until ( Successful ) OR noMoreMoves
End.
```

- Data representation and initial values
  - board: matrix of integer

    To keep track of history, of successive board occupations.
  - const int index = 8;
  - int MTX[index][index]
  - MTX[i][j] = 0 , means field(I, j) is not visited
  - MTX[i][j] = k , means field(I, j) is visited in the k^{th} move

    $1 \leq k \leq N^2$
  - The MTX initial value to zero
  - Parameters of tryNextMove
    - Current field [(x, y coordinates) $1 \leq x, y \leq N$]
    - Move number
    - Boolean variable (successful or not)

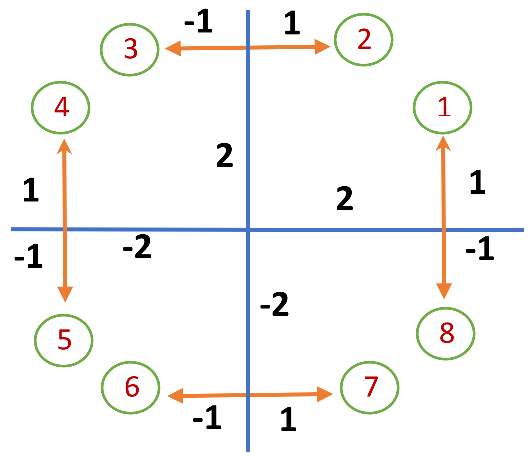
```
Procedure tryNextMove( int i, int x, int y, boolean yes )
Begin
    int u, v;
    boolean  ok;
    Initialize selection of moves
    Repeat
        ok = false;
        let u, v be the coordinates of the next move
        defined by the chess values
        if (1 ≤ u ≤ n) AND (1 ≤ v ≤ n) AND MTX[u][v] = 0 then
            MTX[ u ][ v ] = i;
            if ( i < N^2 ) then
                tryNextMove( i+1, u, v, ok );
                if Not ok then
                    MTX[ u ][ v ] = 0;
                end if
            else
                ok = true;
            end if
        end if
    until ( ok ) OR ( no_More_Moves )
    yes = ok;
end.
```

- Given a starting point x, y then there are 8 potential coordinates for (u, v).

  (x ± 2, y ± 1)

  (x ± 1, y ± 2)

xIncrement	yIncrement
2	1
1	2
-1	2
-2	1
-2	-1
-1	-2
1	-2
2	-1




```
Procedure tryNextMove ( …)
Begin
        Int k;
        xIncrement[ 8 ] = { 2, 1, -1, -2, -2, -1, 1, 2 } ;
        yIncrement[ 8 ] = { 1, 2, 2, 1, -1, -2, -2, -1 } ;
        k = 0;
        Repeat
                k = k + 1;
                u = x + xIncrement[ k ];
                v = y + yIncrement[ k ];
                …
        Until ( ok ) OR ( k == 8 )
end.
```

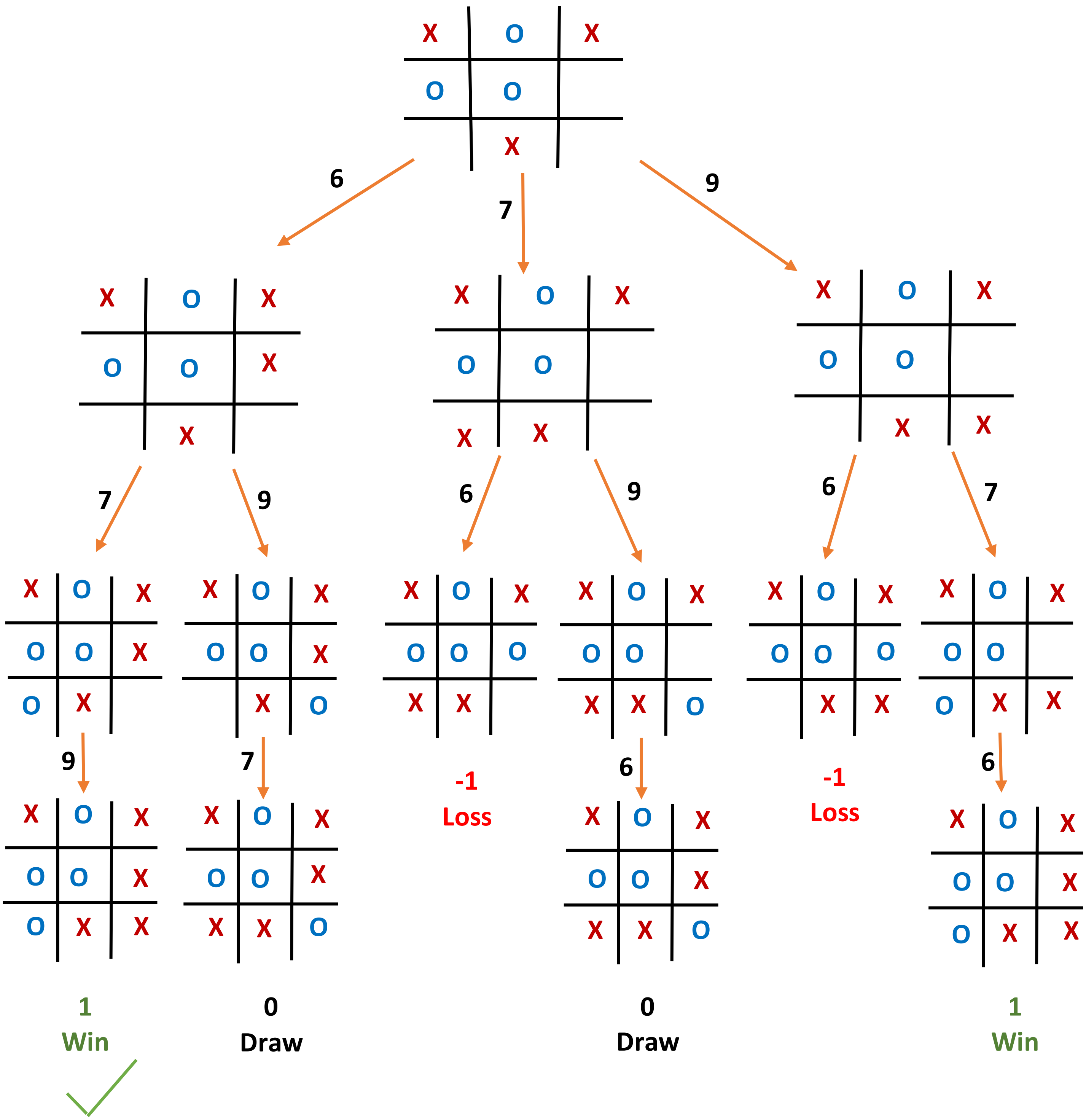
Game trees and the minmax Algorithm

- In complicated games such as chess, the computer can analyze only a few moves deep (usually fewer than 10), become the huge number of possible moves make the number of variations immonse.
- However, in the game of tic_tac_toe the computer can examine every variation, all the way to the final position, because the number of moves is always small (less than 9).
- The number of variation will be less than 9*8*7*6*5*4*3*2 = 362,880
- The computer chooses its move using a minmax algorithm. At positions where the game is over (either a win, loss, or draw), the final position is given a value by using what is called the static evaluation function.
- Static evaluation function

Value	Game result
1	Win
0	Draw
-1	Loss

1	2	3
4	5	6
7	8	9

- This is the basis of the minimax algorithm, which start at the bottom of the tree, evaluating final positions with the static evaluation function. Then, for each internal node, the rules of its child nodes are either maximized or minimized (depending whose move it is at this node), and the internal node is given this value.



Algebraic Algorithm

$$F(x) = a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x^1 + a_0$$

- Representation of data

  A : array [1..n] of real ← dense representation

  $F(x) = x^{1000} + 1$ ← sparse representation

  n, d, n-1, d_{n-1}

  Worst case double storage

Example:

$F(x) = x^6 + 2x^2 + 4$

- Dense representation

  6, 1, 0, 0, 0, 2, 0, 4

  n Coefficient

- Sparse representation

  6, 1, 2, 2, 0, 4

  Power Coefficient

- Algorithm for dense representation

```
term = 1;
sum = a[0];
for ( i = 1; I <= n; i++ )
   term = term * value;
   sum =  sum + a[ i ] * term
end for
```

Mul ➔ 2n

Add ➔ n

Assg ➔ 2n+2

- Horner's Methods

  $8x^5 + 3x^4 + 2x^3 + 6x^2 + 7x + 4$

  ((((8x + 3) x + 2) x + 6) x +7) x + 4

```
i = n-1;
sum = aₙ;
while ( i > 0 )
        sum = sum * value + aᵢ ;
        i--;
end while
```

Mul ➔ n

Add ➔ n

Assg ➔ n+2

- Sparse representation

```
sum = 0;
for ( i = 1; i < m; i++)
   sum = sum + aᵢ * vᵉ
end for
```

represent power

⇨ Improvement

$value^5 = value^4 * value$

Example:

$8X^{12} + 4X^{7} + 6X^{3} + 5X^{0}$

$6 (X^{(3-0)} * X^{0})$

$4 (X^{(7-3)} * X^{3})$

$8 (X^{(12-7)} * X^{7})$

$\mathbf{8 (X^{(12-7)} * X^{7}) + 4 (X^{(7-3)} * X^{3}) + 6 (X^{(3-0)} * X^{0}) + 5X^{0}}$

```
sum = 0;
e0  = 0 ;
term = 1;
for ( i = 1; i <= m; i++)
        r = v↑( ei – ei-1 );
        term = r * term;
        sum = sum + ai * term;
end for
```

⇨ Horner's Methods

$(((a_m\ x^{em-em-1}) * x^{em-1 - em-2}) \ldots$

$\mathbf{(((8\ X^{(12-7)} + 4) * X^{(7-3)} + 6) * X^{(3-0)} + 5)\ X^{0}}$

```
sum = 0;
for ( i = m; i >= 1; i-- )
        sum = ( sum + ai ) * value↑( ei – ei-1)
end for
```