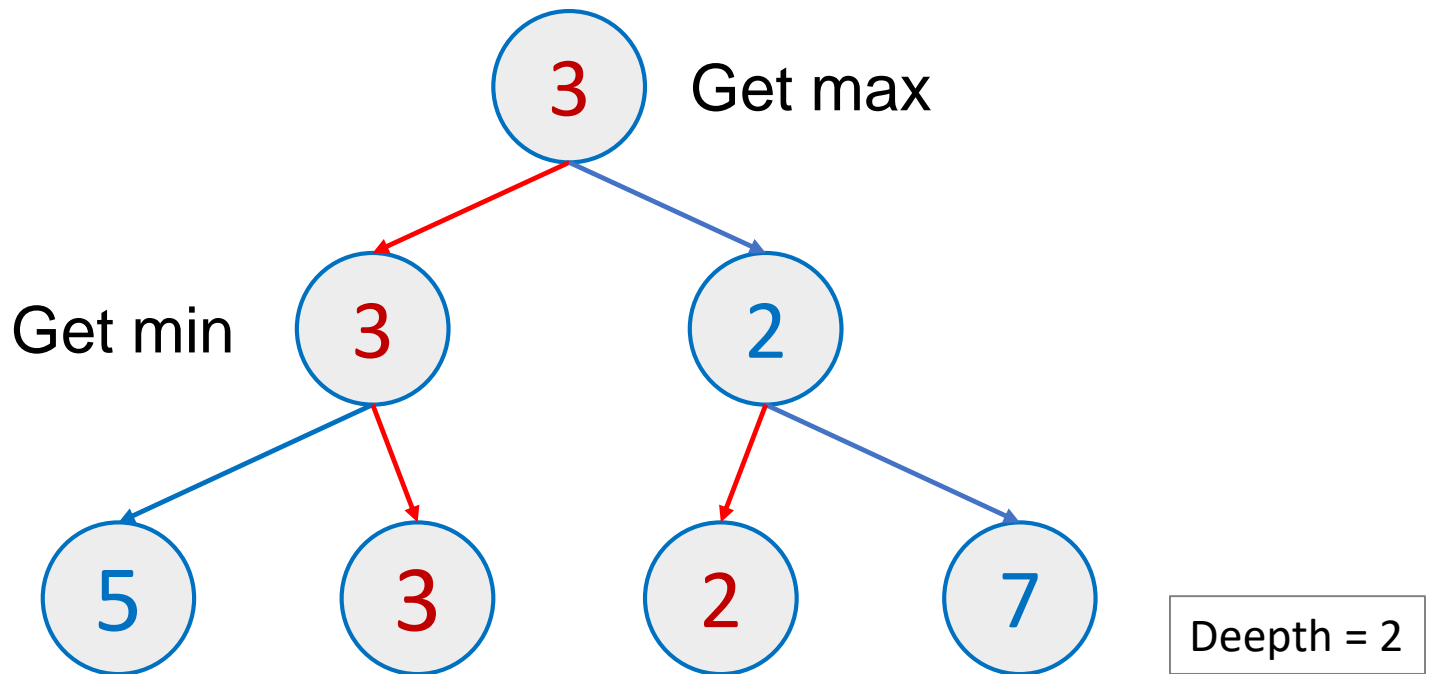


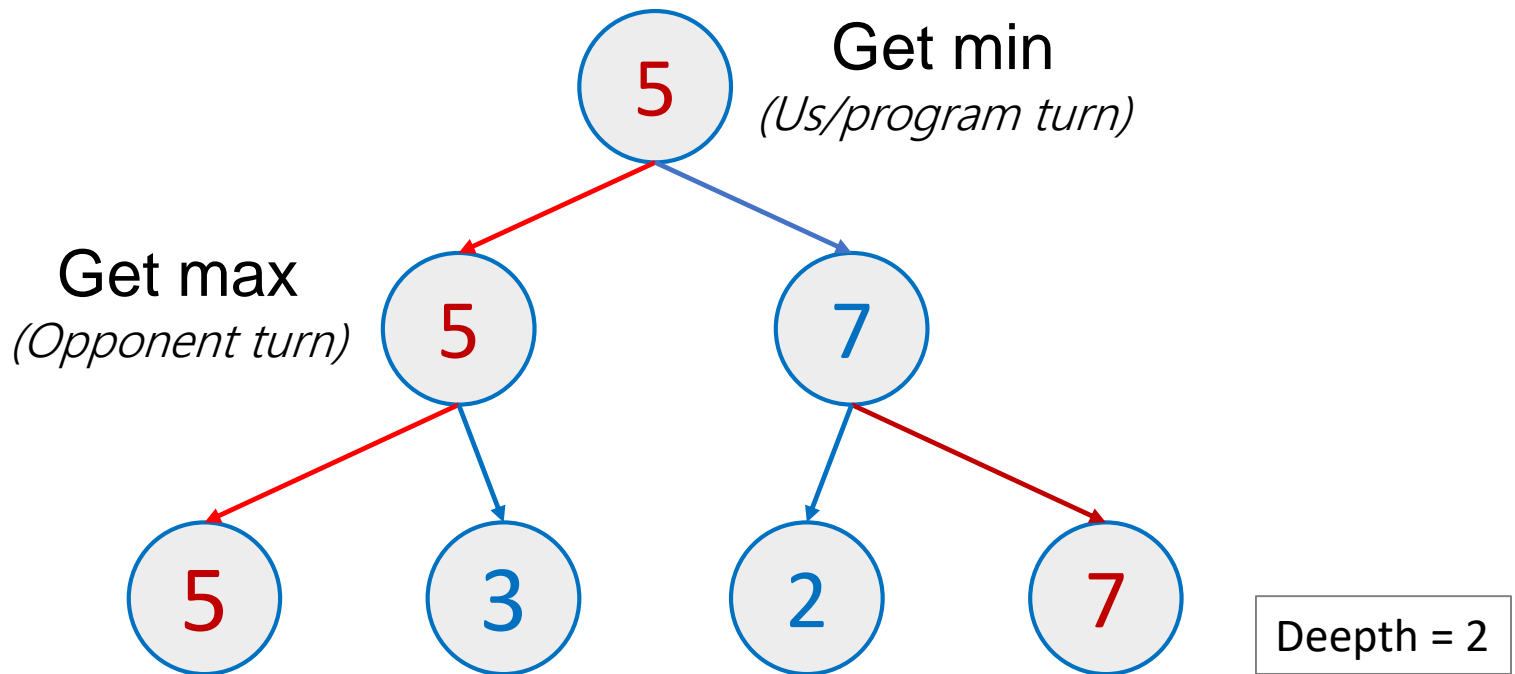
MiniMax Algorithm

- MiniMax algorithm help us/program make a decision which is the best move in all of state can happen.
- To basically, the whole algorithm is divided into two-phase. Get max and get min from children of each state.



MiniMax Algorithm

- When we combine this algorithm with [Adversarial Search](#), we must predict our's opponent move. Suppose they always select the best move for them. So, we change our algorithm a little bit.



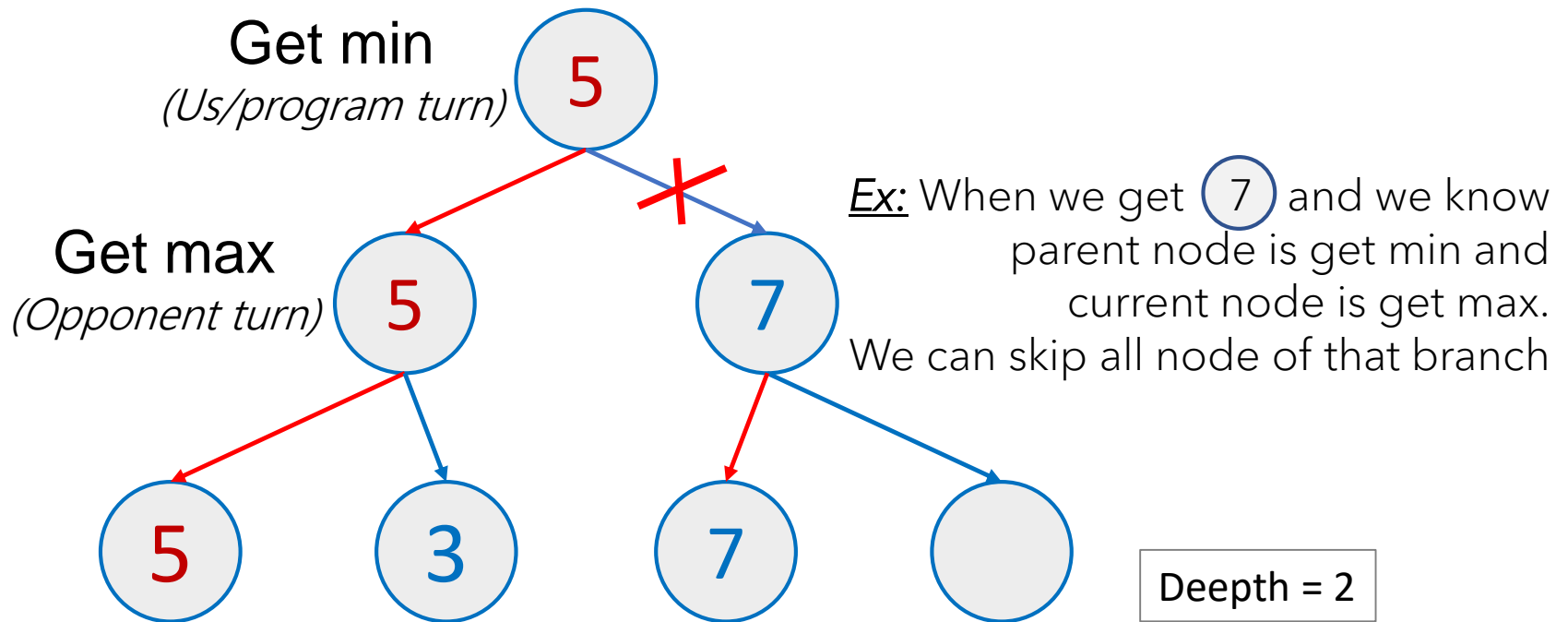
MiniMax Algorithm

Pseudocode

```
def mini_max(crr_state, depth, turn):  
    if depth == 1:  
        return get_max_benefit(crr_state)  
  
    best_move = None  
    crr_benefit = 1000 #if depth is odd, 0 if not  
    for move in can_move(crr_state, turn):  
        get_benefit = mini_max(move, depth - 1,  
                                change_turn)  
  
        if get_benefit > crr_benefit and depth%2==1  
        or get_benefit < crr_benefit and depth%2==0:  
            crr_benefit = get_benefit  
            best_move = move  
  
    return best_move, crr_benefit
```

Alpha-beta Pruning

- Alpha-Beta pruning is an optimization technique for MiniMax algorithm.
- It will cut off all states that no longer necessary to calculate.

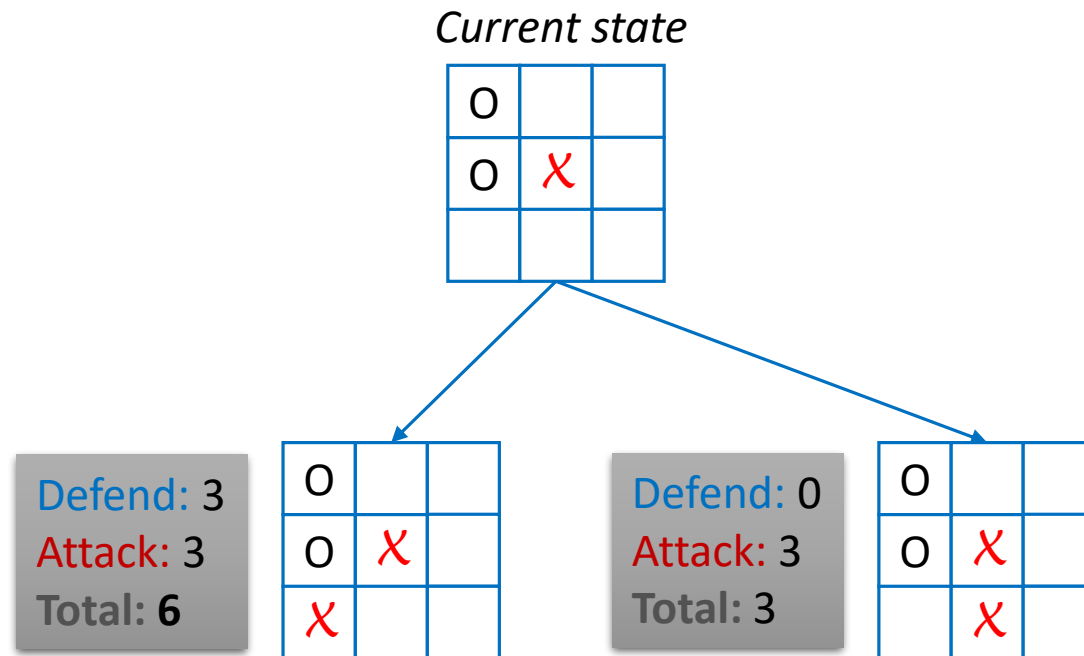


Profit/Benefit Function

The benefit of a state depends on two main part:

- How many point can earn from **defend move**.
- How many point can earn from **attack move**.

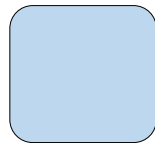
** Do not do stupid move.*



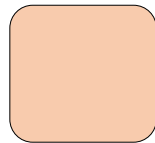
Profit/Benefit Function

Point of **defend move** in gomoku/tic-tac-toe game consist of:

- How many lines you can deflect, prioritize the longest line first.
- Move into the position that our opponent can get the victory.



Deflect line



Deflect opponent victory

O		
O	X	
X		

Profit/Benefit Function

Point of **attack move** in caro/gomoku game consist of:

- How many lines you can make, prioritize the longest line first.
- Move into the position that we can get the victory.

Make a line

O	X	
X	O	X

Get the victory

		X
O	X	
X	O	

Profit/Benefit Function

Stupid move is:

- Move to the position can't get the victory after that. (1)
- Move into the position not necessary to defend. (2)

(1)

O	X	X
	O	

(2)

O		
X	O	X

Profit/Benefit Function

Pseudocode

```
def get_max_benefit(crr_state):  
    best_move = None  
    crr_benefit = 0  
    for move in can_move(crr_state, turn):  
        if not is_stupid_move(move):  
            get_benefit = depend(move) + attack(move)  
  
            if get_benefit > crr_benefit:  
                crr_benefit = get_benefit  
                best_move = move  
  
    return best_move, crr_benefit
```