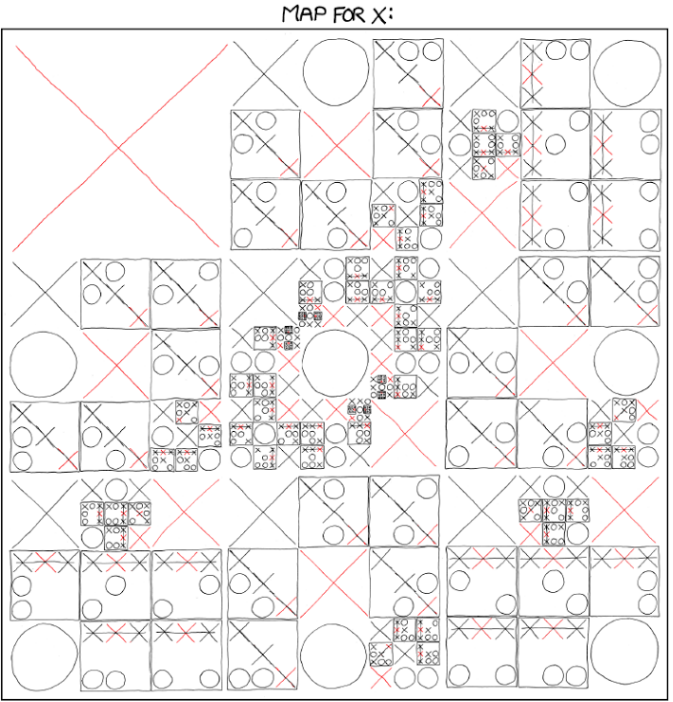
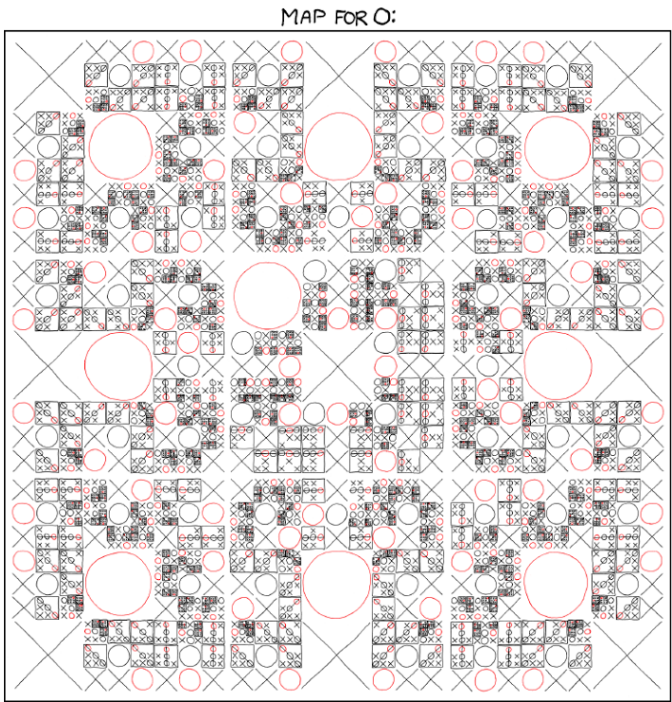
Game Tree Search

1/6/17





Frameworks for Decision-Making

1. Goal-directed planning
   - Agents want to accomplish some goal.
   - The agent will use search to devise a plan.

2. Utility maximization
   - Agents ascribe a utility to various outcomes.
   - The agent attempts to maximize expected utility.

Advantages of Utility Modeling

- Handles uncertainty better
  - Choose actions to maximize *expected* utility.
  - We'll take advantage of this in a few weeks.

- Simplifies modeling other agents
  - Assume all agents are utility maximizers.
    - And all agents know all other agents are utility maximizers.
  - We just have to figure out their utilities.

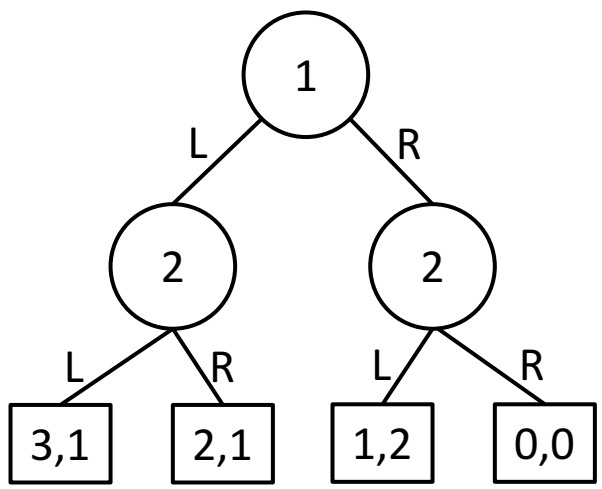
Sometimes this is really hard, but this week it's easy.

Behaving Optimally with Multiple Agents

We need game theory!

If agents act sequentially:

- Extensive form games



- Our focus this week.

If agents act simultaneously:

- Normal form games

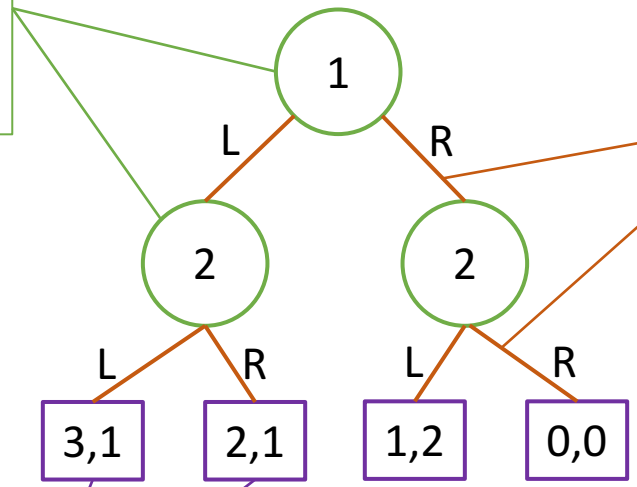
2 \ 1	R	P	S
R	0,0	-1,1	1,-1
P	1,-1	0,0	-1,1
S	-1,1	1,-1	0,0

- We'll come back to this at the end of the semester.

Extensive form game terminology

decision nodes (states)

Each node belongs to a specific agent (player).



actions (moves)

terminal nodes (outcomes)

Each outcome lists a utility for every player.

Example Game: Nimm

- There are initially N pieces.
- Each turn a player must remove 1, 2, or 3 pieces.
- The player who removes the last piece loses.

Let’s play a game where N=9, you go first.

Exercise: play a few games of Nimm

- Try different values of N.
  - 1, 2, 3, …, 9, 10, …

- Who wins under optimal play?
- How does it depend on N?

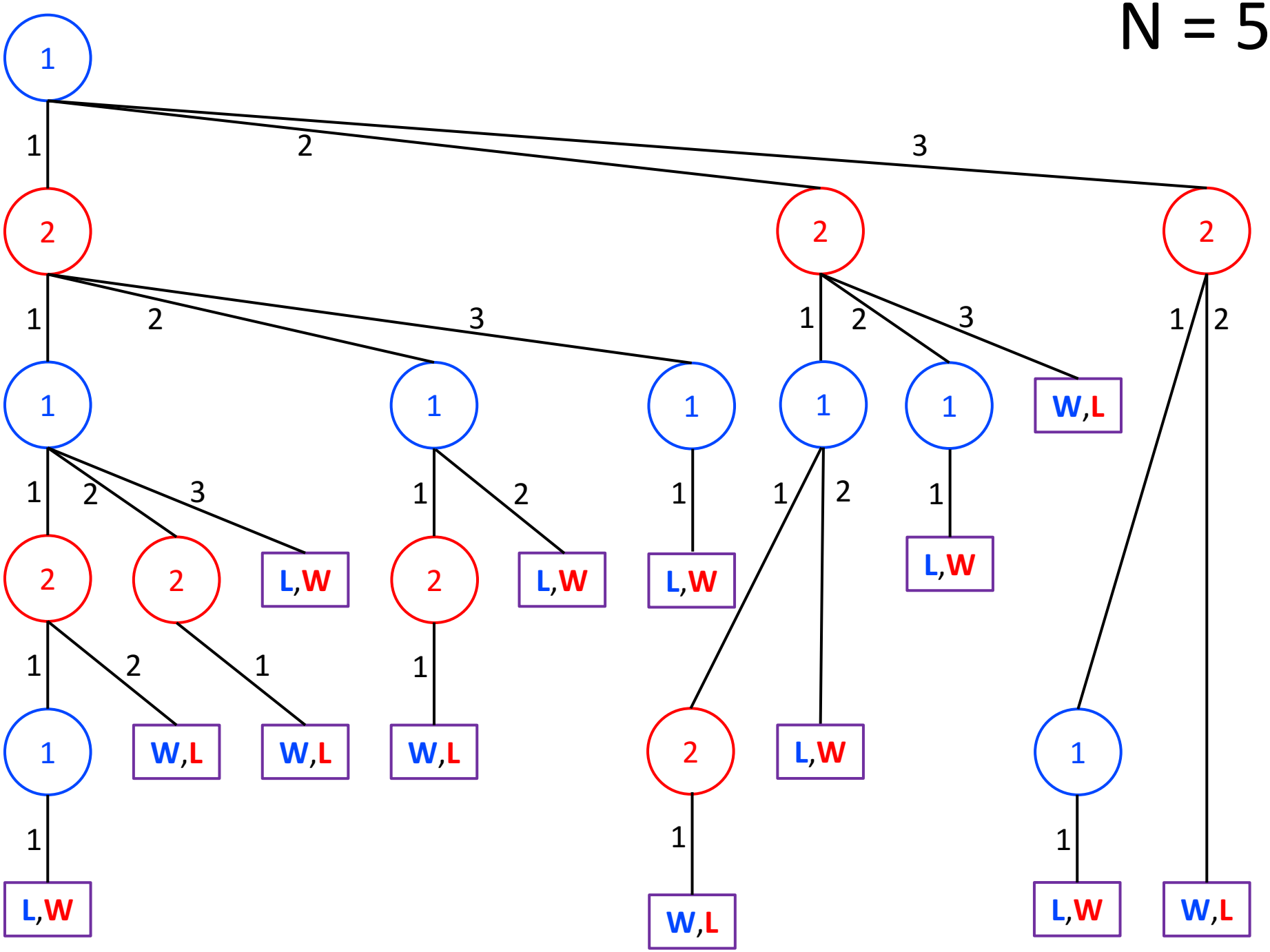
N	Outcome for P1	First move
1	L	1
2	W	1
3	W	2
4	W	3
5	L	?
6	W	1
7	W	2
8	W	3
9	L	?
10		
11		
12		
13		
14		

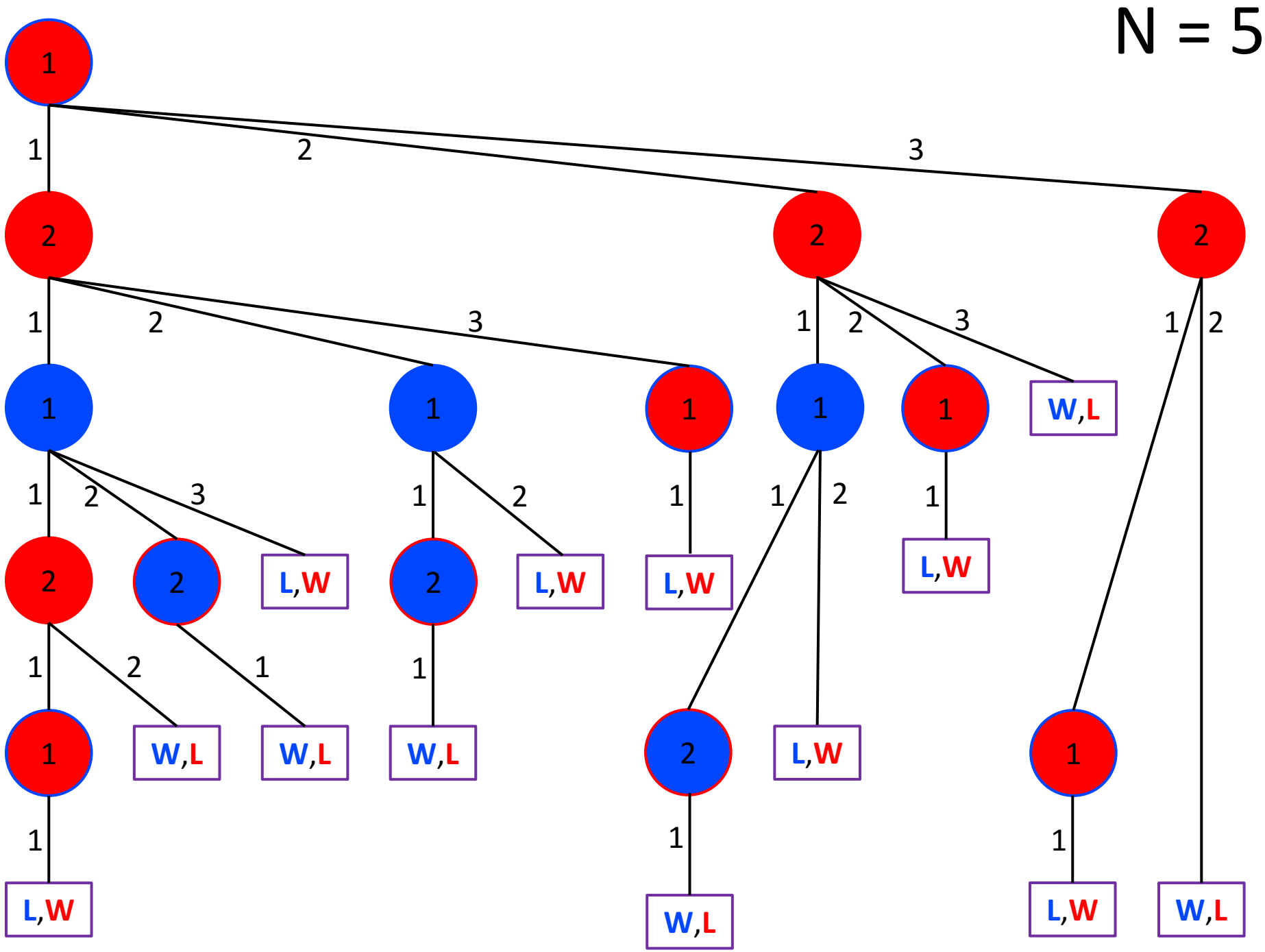
Backward Induction

Key idea: start from outcomes and work your way up.

- At leaf nodes, return the outcome.
- At decision nodes, recursively determine the outcome of each action.
- The optimal move is the one that gives the best outcome for the current player.

N = 5


1
1
2
3
2
2
1
2
3
1
2
3
1
2
1
1
1
1
W,L
1
2
3
1
2
1
1
2
1
2
2
L,W
2
L,W
L,W
L,W
1
2
1
1
1
W,L
W,L
W,L
2
L,W
1
1
1
L,W
W,L
L,W
W,L

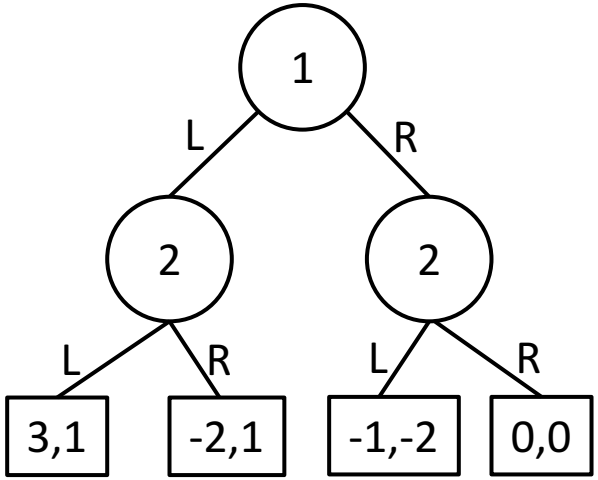

N = 5
1
2
3
W,L
L,W

Backward Induction Pseudocode

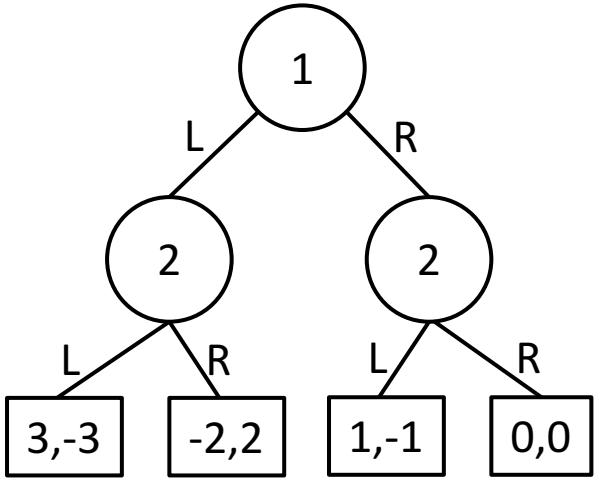
```
function backward_induction(state, player):
  if state is terminal:
    return outcome

  initialize best_outcome, best_utility

  for each action available in state:

    ns, np = make_move(state, action)
    outcome = backward_induction(ns, np)

    if utility(outcome, player) > best_utility:
      update best_outcome, best_utility

  return best_outcome
```

Special Case: Zero-Sum Games

- The sum of utilities is zero for every outcome.



not zero-sum



zero-sum

- In a zero-sum game, my gain is always your loss.
- We can represent one-fewer utility per outcome.
- Is Nimm zero-sum?

Min-Max Pseudocode

```
function min_max(state, player):
  if state is terminal:
    return none, value
  initialize best_action, best_value

  for each action available in state:
    next_state = make_move(state, action)
    act, val = min_max(next_state, other_player)

    if player is maximizer and val > best_value:
      update best_action, best_value
    if player is minimizer and val < best_value:
      update best_action, best_value

  return best_action, best_value
```

Alternative Min-Max Pseudocode

```
function max_value(state):
  if state is terminal:
    return value
  initialize best_val
  for each action available in state:
    next_state = make_move(state, action)
    best_val = max(min_value(next_state), best_val)
  return best_val

function min_value(state):
  ...
    best_val = min(max_value(next_state), best_val)
  ...
```

Problem: game tree size

- For most interesting games the game tree is too large to search to the end and to find optimal moves.
- In chess, the branching factor is approximately 35 and games can last for 100 moves.
- This creates a game tree of 35^100 nodes which is approximately 10,154! .
- Instead we will search to a limited depth and try to approximate the value of states.

How big is the game tree for tic-tac-toe?

Checkers?

Evaluation Function

- Look at a game state without knowing any context and try to assign it a value.
- Performance of a game playing program is highly dependent on this evaluation.
- Using a good evaluation function allows us to make informed decisions about which move now is likely to lead to good situations later.

Features of a good evaluation function

- When a terminal state is reached, score it correctly.
- Should be efficient to calculate since it will be called many, many times.
- Should reflect the actual chances of winning.
- Exactness is less important than trying to get the relative values correct.