Lecture 6

Game Playing

6.1 Games

Games or **adversarial search problems** focuses on handling a multi-agent competitive environment.

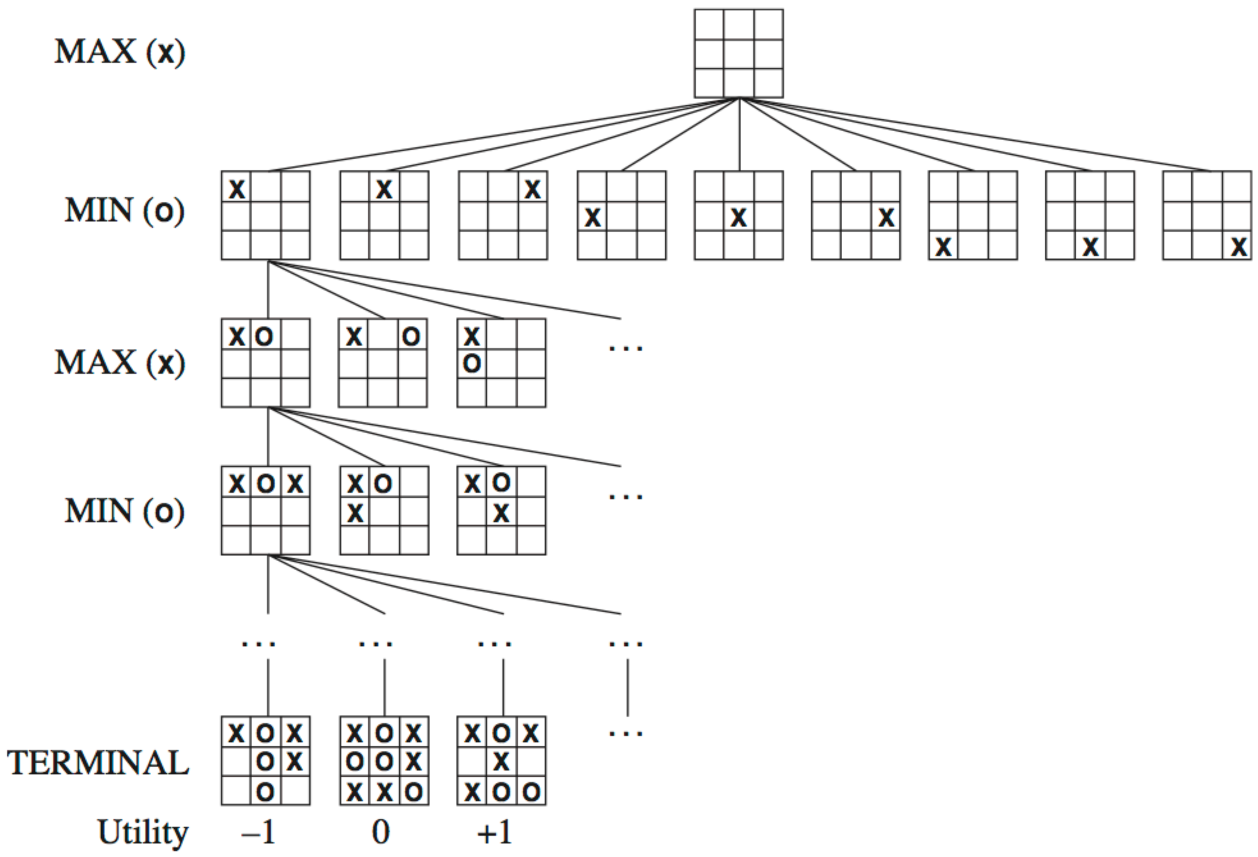
- Fully observable, and Deterministic environment
- Each player selects an action then changes to another one
- Two-player: human and computer
- Zero-sum: Win $= +1$ and Lose $= -1$

Searching to win the game is different from the general search because of the *unpredictable opponent*, and *time limits*.

6.2 Game as a Search Problem

- S_0: **Initial state**
- $\textsc{Player}(s)$: define which player has the move in a state
- $\textsc{Actions}(s)$: legal moves in a state
- $\textsc{Result}(s, a)$: the result of a move
- $\textsc{Terminal-Test}(s)$: true when the game is over, and false otherwise.
- $\textsc{Utility}(s, p)$: final numeric value for a game ending with s for a player p

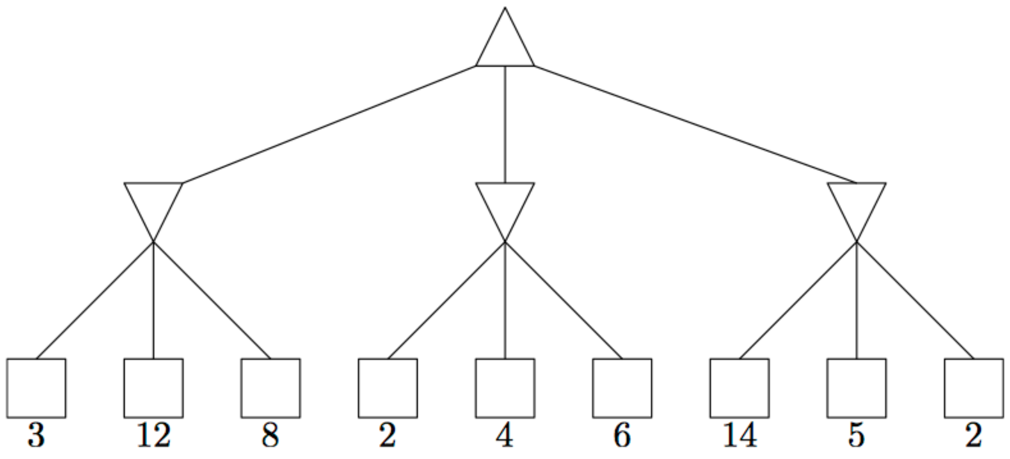
6.2.1 Game Tree



6.3 Optimal Decisions in Games

$$\text{MINIMAX}(s) = \begin{cases} \text{Utility}(s) & \text{if } s \text{ is a terminal state,} \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

Example 6.1 Find the most appropriate action using **depth-first search** and **minimax**.



6.3.1 The minimax Algorithm

Before making decision, the minimax algorithm is applied to compute the minimax decision from the current state. The algorithm recursively computes the minimax values of each successor state. This is similar to conducting **depth-first** or **depth-limited** search.

```
function MINIMAX-DECISION(state) returns an action
    return $\arg\max_{a \in \text{ACTIONS}(a)}$ MIN-VALUE(RESULT(state, a))
```

```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow -\infty$
    for each a in ACTIONS(state) do
        $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s, a)))
    return v
```

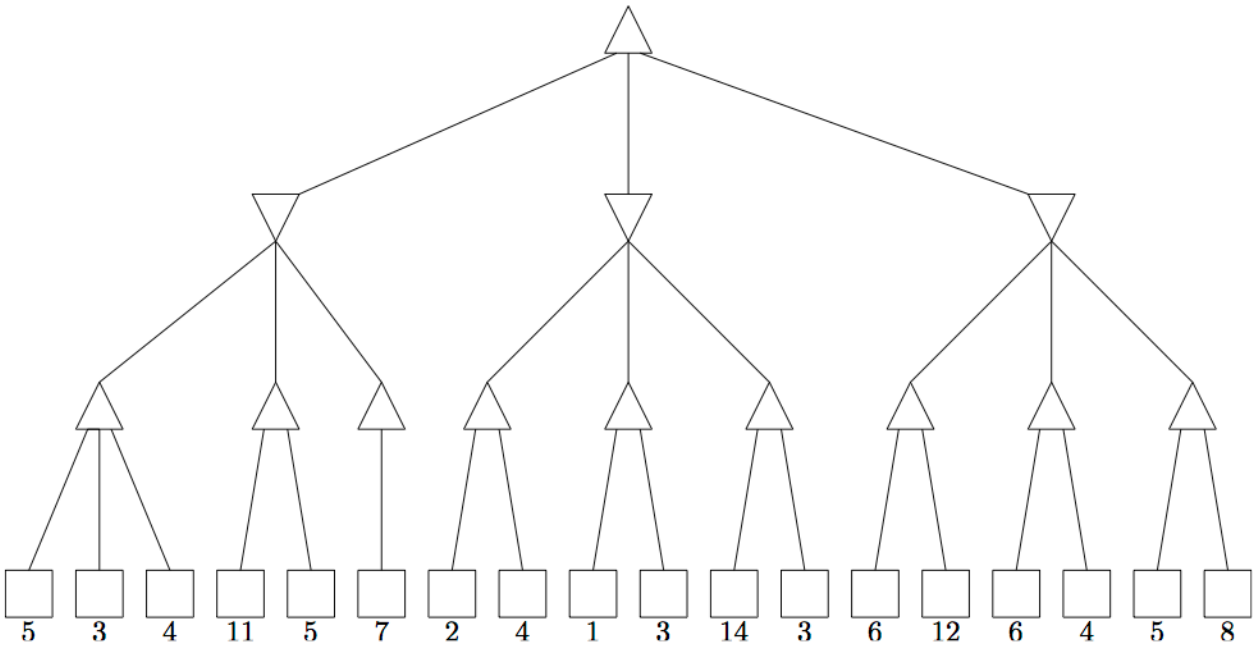
```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow +\infty$
    for each a in ACTIONS(state) do
        $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s, a)))
    return v
```

Exercise 6.1 A two-player game begins with a common number N shared by both players. Each player takes turn to select a number 1, 2, or 3 to be subtracted from N. By repeatedly doing this, the value of N is gradually decreased. The game ends when N becomes 0 and the player who takes the last value loses obtaining utility -1. The one who wins receives utility 1.

1. How can a state be represented? What are the actions that each player can select?

2. Draw a complete game tree for the first player. The game starts from $N = 4$. The tree must include utilities for all terminal states. Then, use the minimax algorithm to decide the first move.

Exercise 6.2 Use **depth-first search** to fill in the minimax values for the following game trees. Write the visiting order for each node. Here, a rectangle represents a terminal state.



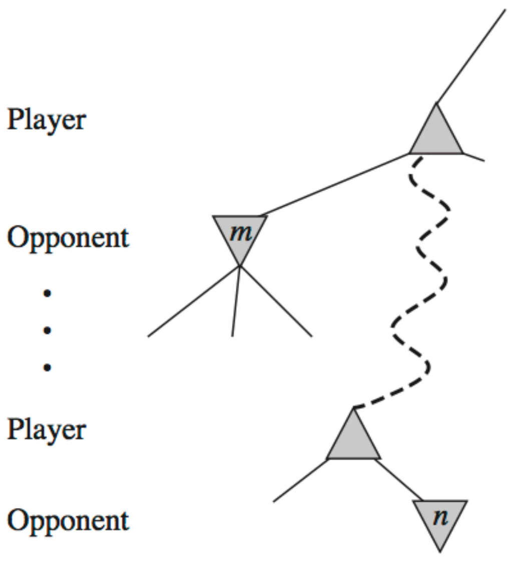
In this exercise, the search goes deep until a terminal state is found. This is not practical. A game tree in the real-world games like Chess contains a huge number of branches and it is very deep. It would take too long to reach all the terminal states.

Therefore, the search is practically limited to a particular depth and a heuristic function is applied to estimate the utility values of nodes at that depth.

6.4 Alpha-Beta Pruning

Minimax search has a drawback that the number of states it has to examine is huge, and exponential in the depth of the game tree. A technique called “**alpha-beta pruning**” aims to correctly explore the game tree without looking at every node.

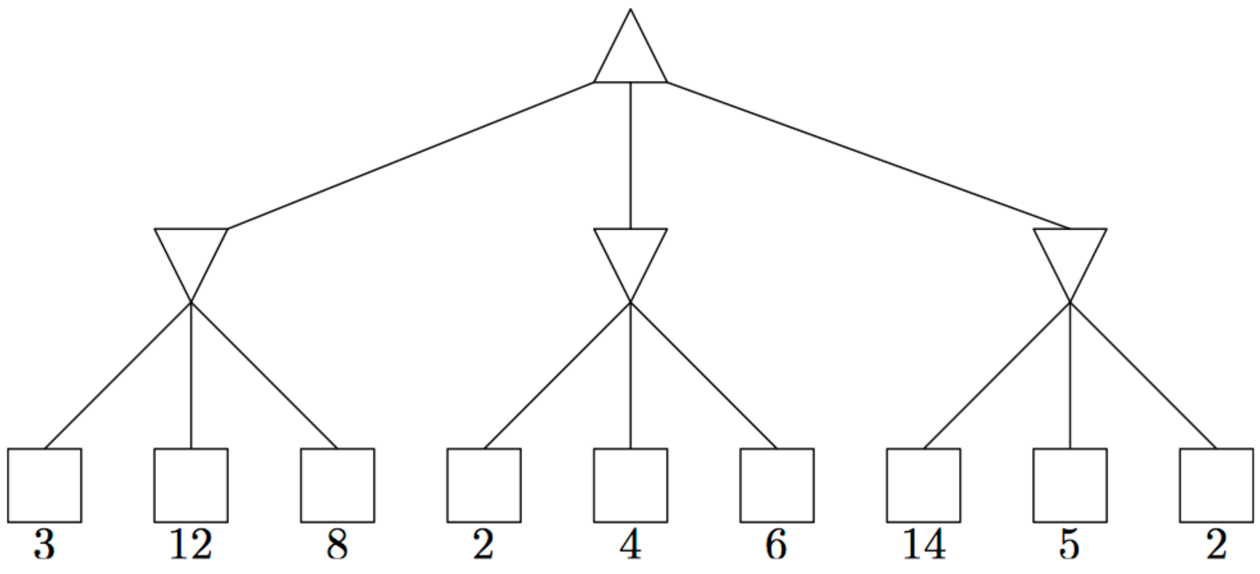
> Consider a node n somewhere in the tree, such that Player has a choice to move to that node. If Player has a better choice m either at the parent node of n or at any choice point further up, then n *will never be reached in actual play.* So once we have found out enough about n (by examining some of its descendants) to reach this conclusion, we can prune it. (AIMA, p.168)



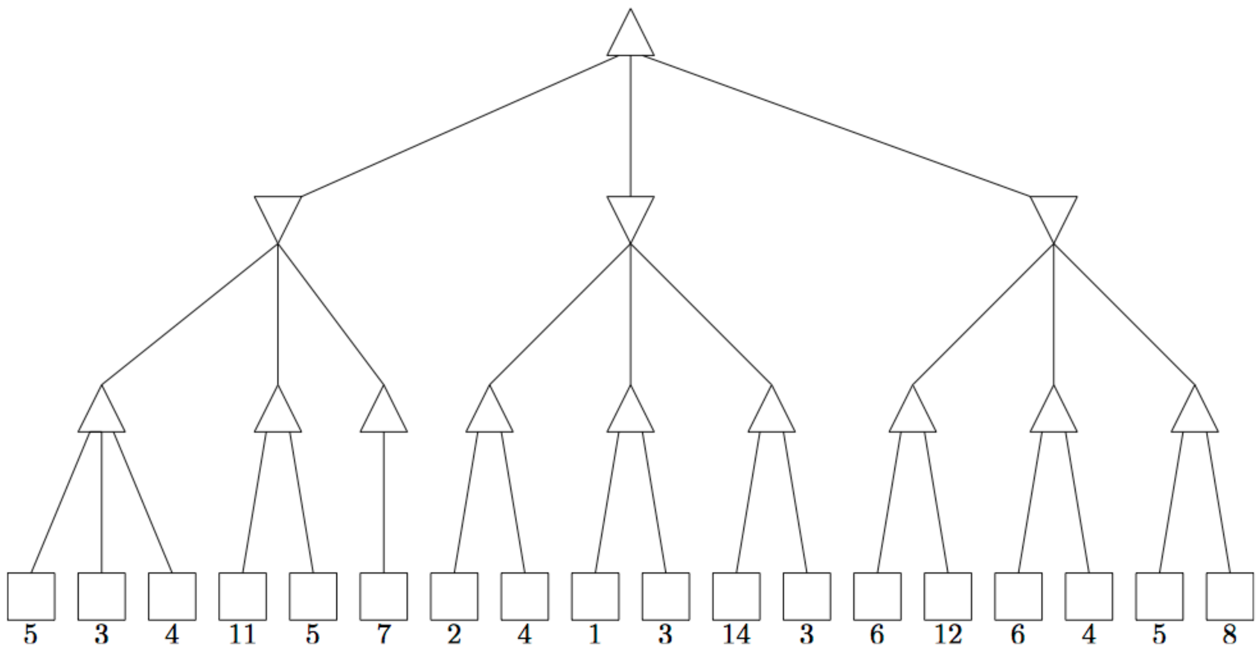
The alpha-beta pruning highly depends on the order in which the states are examined.

Due to this idea, we keep track of *a range of minimax value* for each node, and stop searching some subtrees when the range is invalid.

Example 6.2 Cross out unnecessary subtrees based on the alpha-beta pruning.



Exercise 6.3 Use **depth-first search** and **alpha-beta pruning** to find the most appropriate action.



References

Russell, S. and Norvig, P. (2010). Artificial Intelligence: A Modern Approach (3rd edition). Pearson/Prentice Hall.
Michalewicz, F. and Fogel, D. B. (1998). How to Solve It: Modern Heuristics. Springer.
Aarts and Lenstra (Eds). Local Search in Combinatorial Optimization. Princeton University Press. 1997.