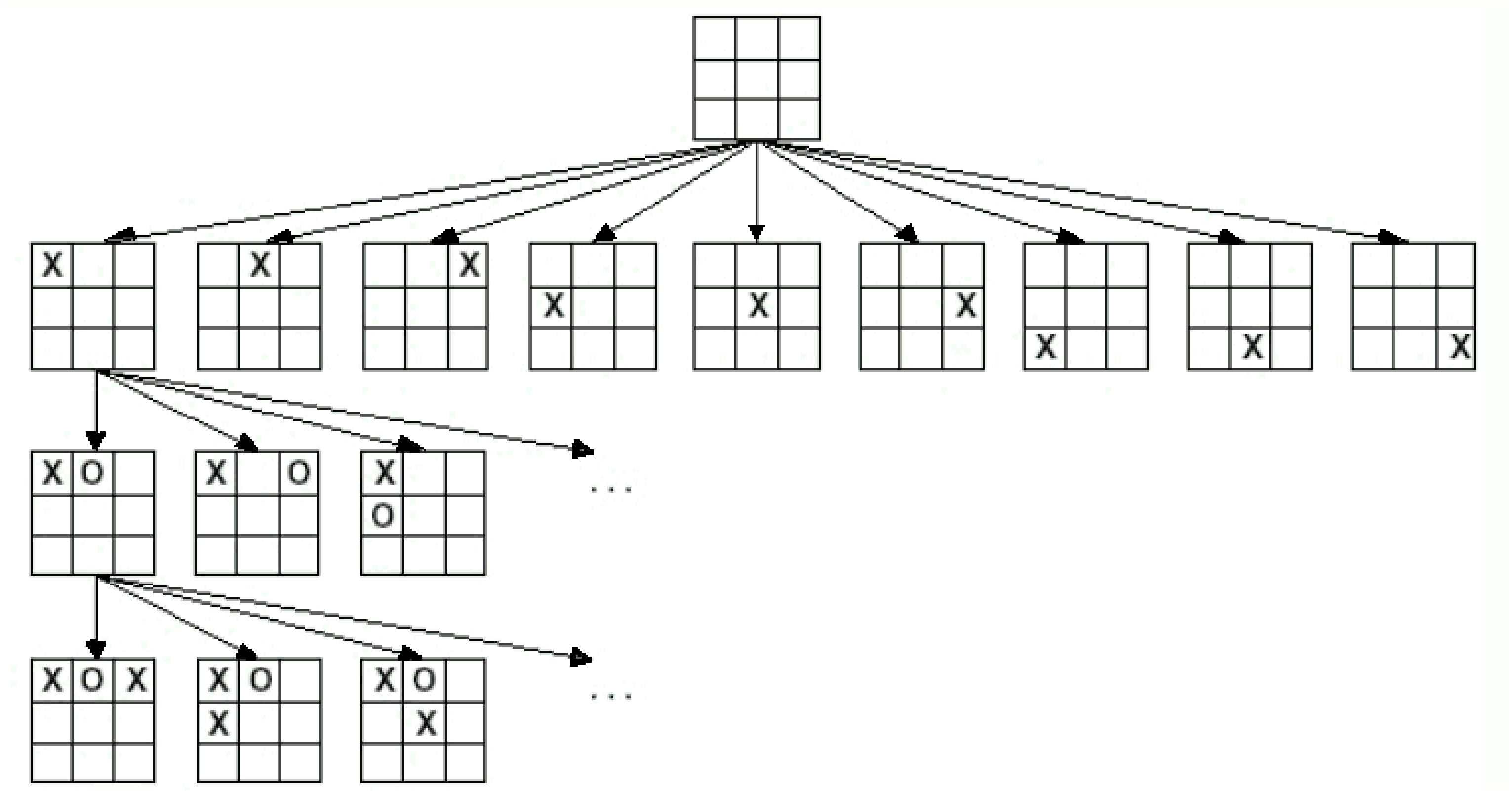
7. Trees :: Applications :: Game Trees

Trees have many applications and are widely used in programming. In this part of the course, we will briefly discuss two applications: game trees and Huffman coding.

A **game tree** is a tree that can be used to analyze certain types of games, e.g., tic-tac-toe, where players take turns making moves:



7. Trees :: Applications :: Game Trees :: Tic-Tac-Toe

Tic-tac-toe is a simple enough game (in the sense that there is a only a small number of moves that can be made at each turn) that the entire game tree could be generated in advance. It would have 10 levels, with the root node being the empty grid at level 0, and all of the nodes at level 10 consisting of completely filled grids.

The number of nodes at level 1 would be 9, since X has 9 squares in which to make a move. At level 2, O would have 8 squares to make a move in, so the total number of nodes would be $9 \cdot 8 = 72$. At level 3, X would have 7 squares to make a move in, so the total number of nodes would be $9 \cdot 8 \cdot 7 = 504$. At level 10, X would be forced to choose the only remaining squares, and the number of leaf nodes in the tree would be $9! = 362{,}880$.

Of course, not all of those leaves represent games states that would be reached; this would happen when an ancestor node of the leaf node represents a game state where X or O wins. A little pruning would reduce the number of nodes in the entire tree.

7. Trees :: Applications :: Game Trees :: Tic-Tac-Toe (continued)

However, in practice, it is not required to generate the entire tree in advance. When X (playing computer) is confronted with a move, we only need to generate enough of the tree to find a node which represents a winning state for X:

For more complex games (chess, checkers, go) where the number of moves at any time—while still finite—leads to a game tree that is too large to be searched for a guaranteed winning move, there are various search strategies that can be used to determine the optimal move to make. The standard algorithm to find the best move while looking at a small part of the game tree is called **minimax**, but that is beyond the scope of this course.