Let us now investigate…

Two-Player Games

Two-Player Games with Complete Trees

We can use search algorithms to write “intelligent” programs that **play games** against a human opponent.

Just consider this extremely simple (and not very exciting) game:

- At the beginning of the game, there are seven coins on a table.
- Player 1 makes the first move, then player 2, then player 1 again, and so on.
- One move consists of removing 1, 2, or 3 coins.
- The player who removes all remaining coins wins.

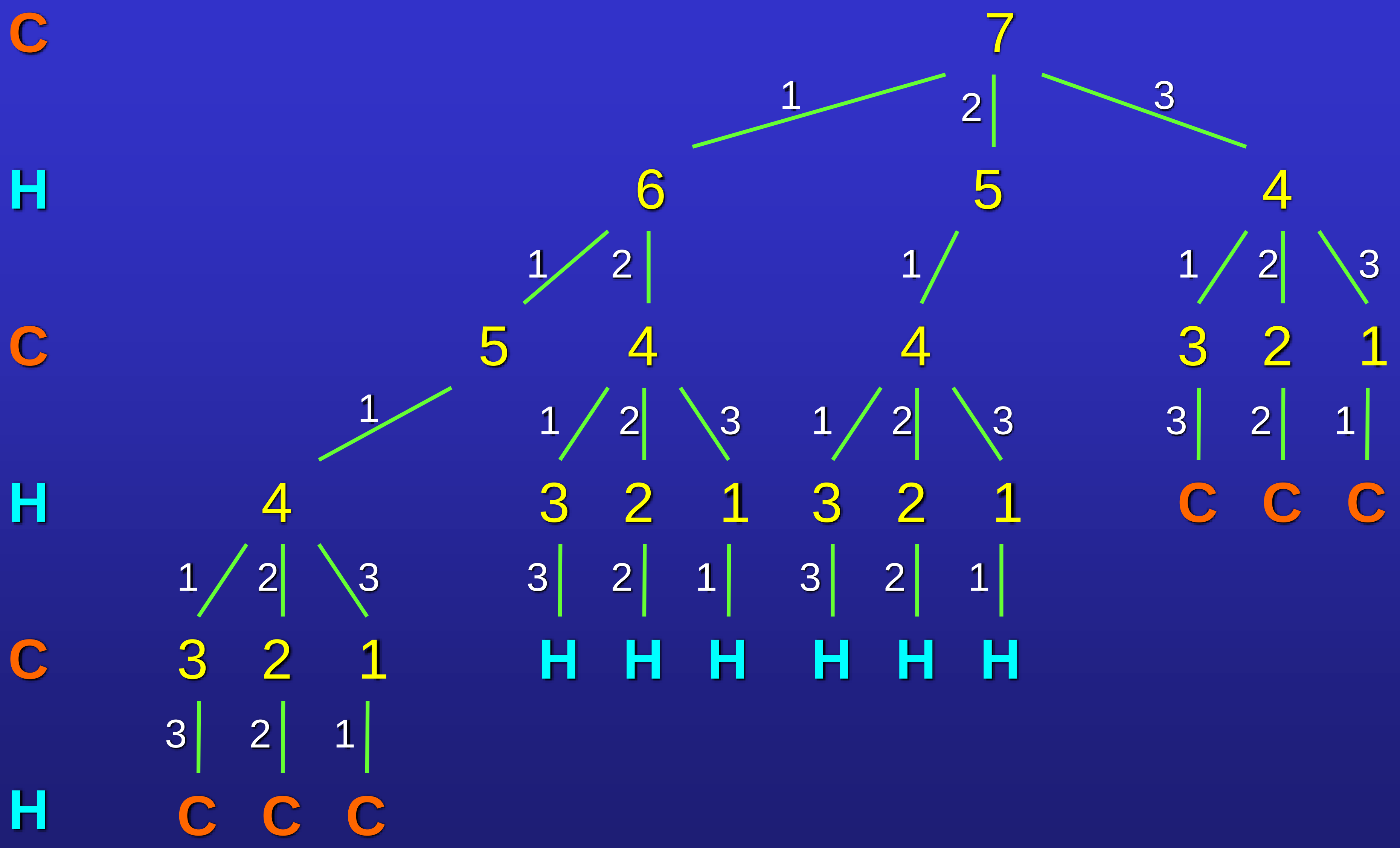
Two-Player Games with Complete Trees

Let us assume that the computer has the first move. Then, the game can be described as a **series of decisions**, where the first decision is made by the computer, the second one by the human, the third one by the computer, and so on, until all coins are gone.

The **computer** wants to make decisions that **guarantee its victory** (in this simple game).

The underlying assumption is that the **human** always finds the **optimal move**.

Two-Player Games with Complete Trees



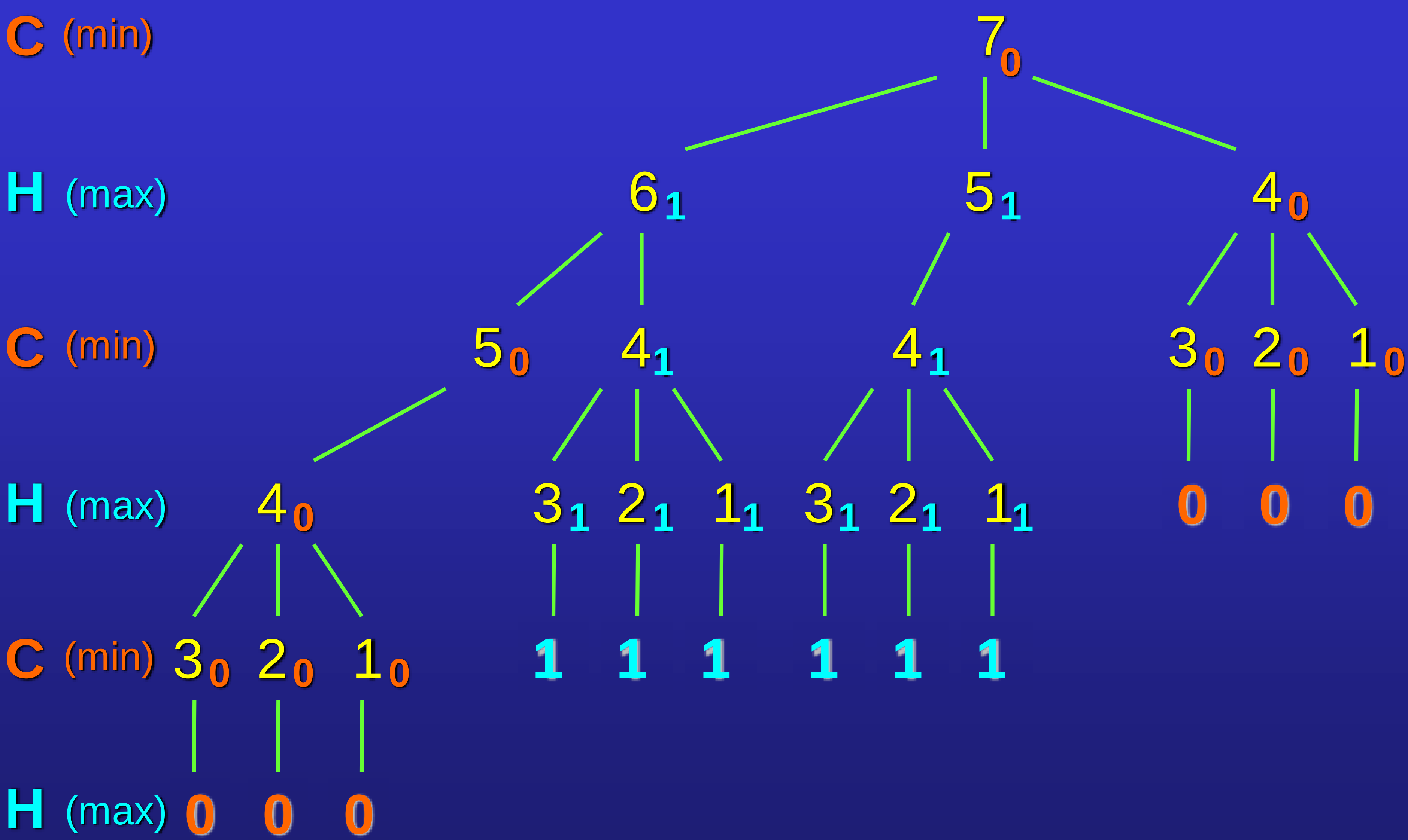
Two-Player Games with Complete Trees

So the computer will start the game by taking three coins and is **guaranteed to win** the game.

The most practical way of implementing such an algorithm is the **Minimax procedure**:

- Call the two players MIN and MAX.
- Mark each leaf of the search tree with 0, if it shows a victory of MIN, and with 1, if it shows a victory of MAX.
- Propagate these values up the tree using the rules:
  - If the parent state is a MAX node, give it the maximum value among its children.
  - If the parent state is a MIN node, give it the minimum value among its children.

Two-Player Games with Complete Trees



Two-Player Games with Complete Trees

The previous example shows how we can use the Minimax procedure to determine the computer's best move.

It also shows how we can apply depth-first search and a variant of backtracking to prune the search tree.

Before we formalize the idea for pruning, let us move on to more interesting games.

For such games, it is **impossible** to check every possible sequence of moves. The computer player then only looks ahead a certain number of moves and **estimates** the chance of winning after each possible sequence.

Two-Player Games

Therefore, we need to define a **static evaluation function e(p)** that tells the computer how favorable the current game position p is from its perspective.

In other words, e(p) will assume large values if a position is likely to result in a win for the computer, and low values if it predicts its defeat.

In any given situation, the computer will make a move that guarantees a maximum value for e(p) after a certain number of moves.

For this purpose, we can use the Minimax procedure with a specific maximum search depth (**ply-depth k** for k moves of each player).

Two-Player Games

For example, let us consider **Tic-Tac-Toe** (although it would still be possible to search the complete game tree for this game).

What would be a suitable evaluation function for this game?

We could use the **number of lines** that are still open for the computer (X) minus the ones that are still open for its opponent (O).

Two-Player Games

$e(p) = 8 – 8 = 0$

X		
O	X	

$e(p) = 6 – 2 = 4$

O	O	X
X	O	
X		

$e(p) = 2 – 2 = 0$

shows the weak-ness of this e(p)

How about these?

O	O	X
	X	
X		

$e(p) = \infty$

X	X	
O	O	O
	X	

$e(p) = - \infty$