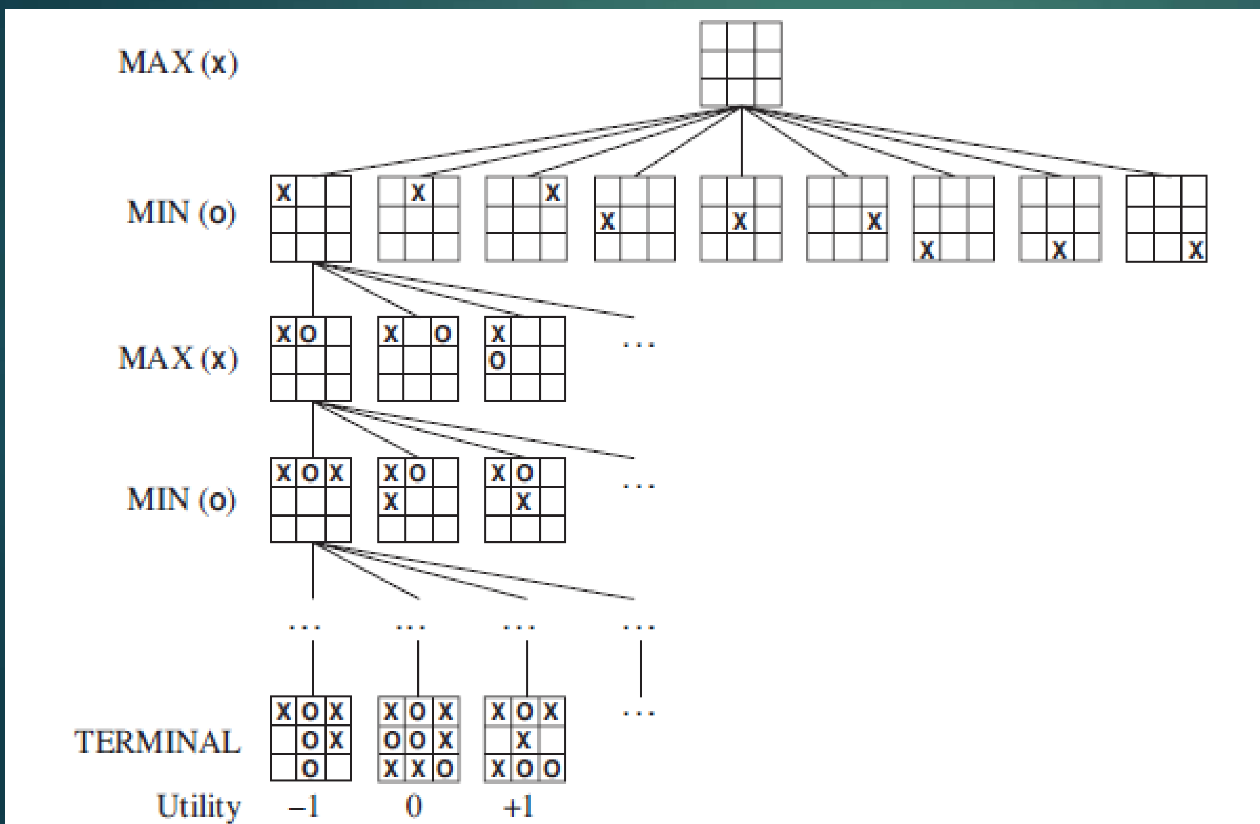
GAMES

- **In Competitive** environment, agents' goals are in conflict, giving rise to ***adversarial (legal proceedings) search*** *problems*—often known as **games**.
- In two player game, both players try to win the game i.e. both of them try to make the best move possible at each turn.
- Both the uninformed and informed Searching techniques are not accurate because of multi-agent environment. And also for multi-agent gaming problems such as chess, the branching factor is very high, so searching will take a lot of time.
- So, we need another search procedures that can generate only good moves and best move can be explored first.

GAME TREE OR SEARCH TREE OR SPACE GRAPH - tree where the nodes are game states and the edges are moves.



Utility values are given from the point of view of MAX; high values are assumed to be good for MAX and bad for MIN

Formal definition of game

A game can be formally defined as a kind of search problem with the following elements:

- $S0$: The **initial state**, which specifies how the game is set up at the start.
- PLAYER(s): Defines which player has the move in a state.
- ACTIONS(s): Returns the set of legal moves in a state.
- RESULT(s, a): The **transition model**, which defines the result of a move.
- TERMINAL-TEST(s): A **terminal test**, which is true when the game is over and false otherwise. States where the game has ended are called **terminal states**.
- UTILITY(s, p): A **utility function** defines the final numeric value for a game that ends in terminal state for a player. In tic tac toe, the outcome is a win, loss, or draw, with values +1, -1, or 0.

OPTIMAL DECISIONS IN GAMES

- In a normal search problem, the optimal solution would be a sequence of actions leading to a goal state—a terminal state that is a win.
- In adversarial (conflicting goal) search, some strategy is required. **Strategy** specifies that
  - MAX’s move in the initial state, then
  - MAX’s moves in the states resulting from every possible response by MIN, then MAX’s moves in the next states resulting from every possible response by MIN to *those* moves, and so on.

Minimax search

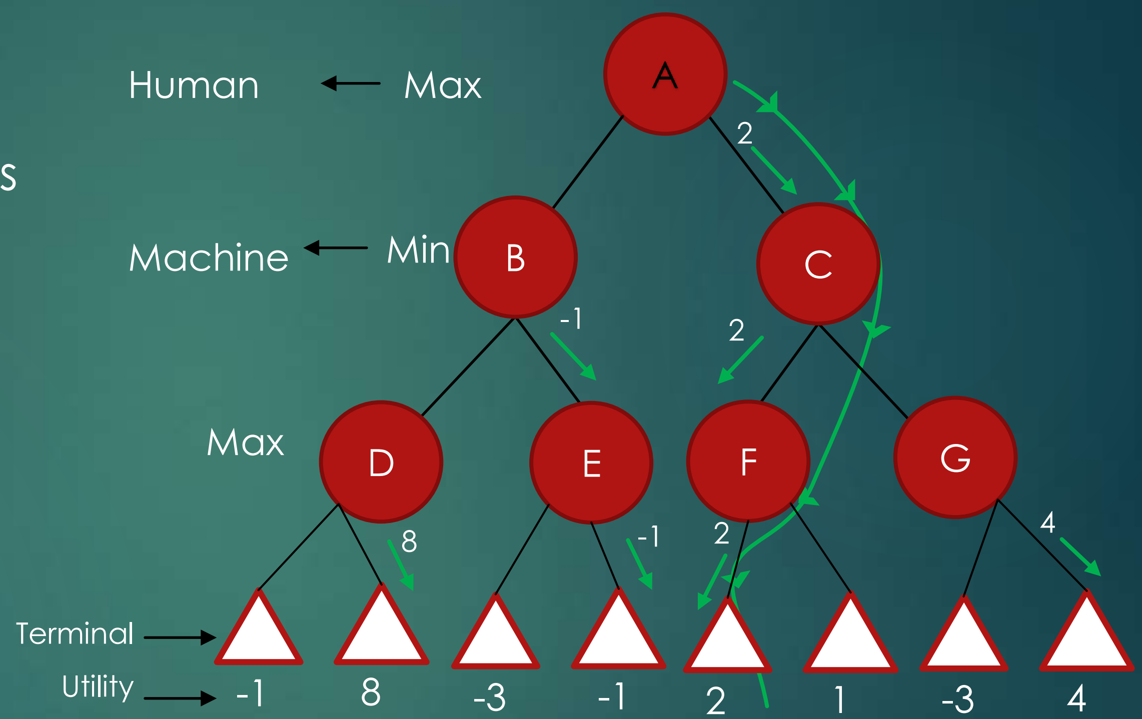
- At the end of the game, points are awarded to the winning player and penalties are given to the loser.
- In every move (*MAX will try to maximize the utility and MIN will try to minimize the utility*) –
  - MAX will try to choose best move to maximize its utility(winning).
  - MIN will try to chose the worst move(according to MAX) to minimize the MAX's utility.
- It is a recursive algorithm, as same procedure occurs at each level.

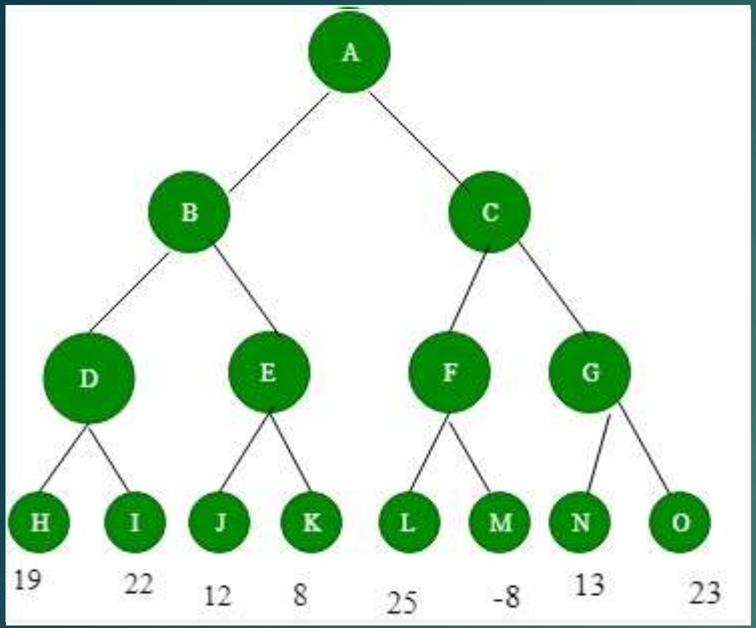
MiniMax Algorithm:

→Best Move Strategy used

→Max will try to maximize its utility (best Move)

→Min will try to minimize its utility (worst Move)

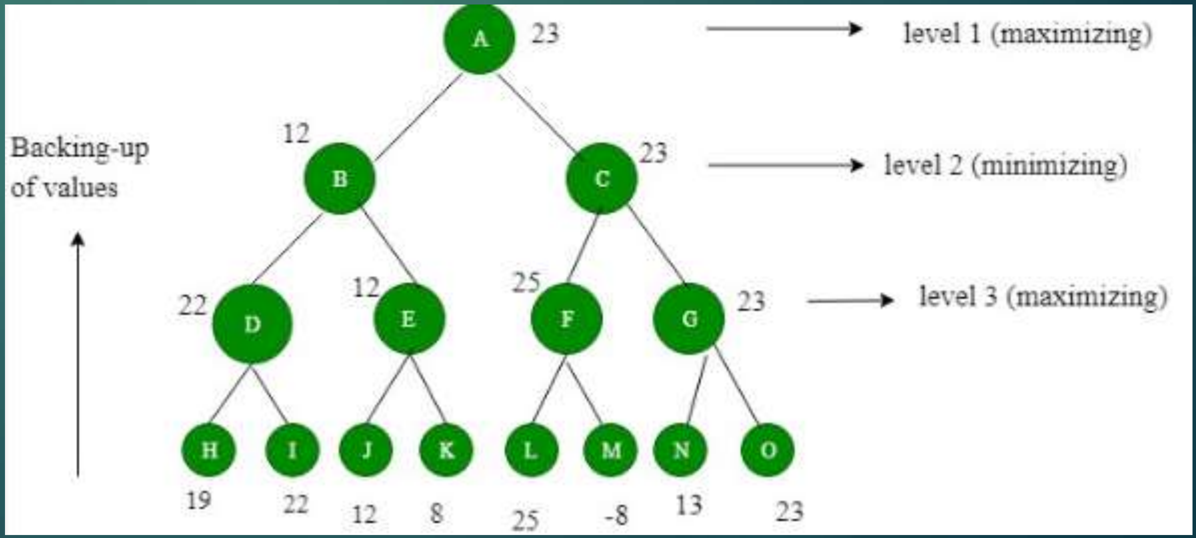



A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
19
22
12
8
25
-8
13
23

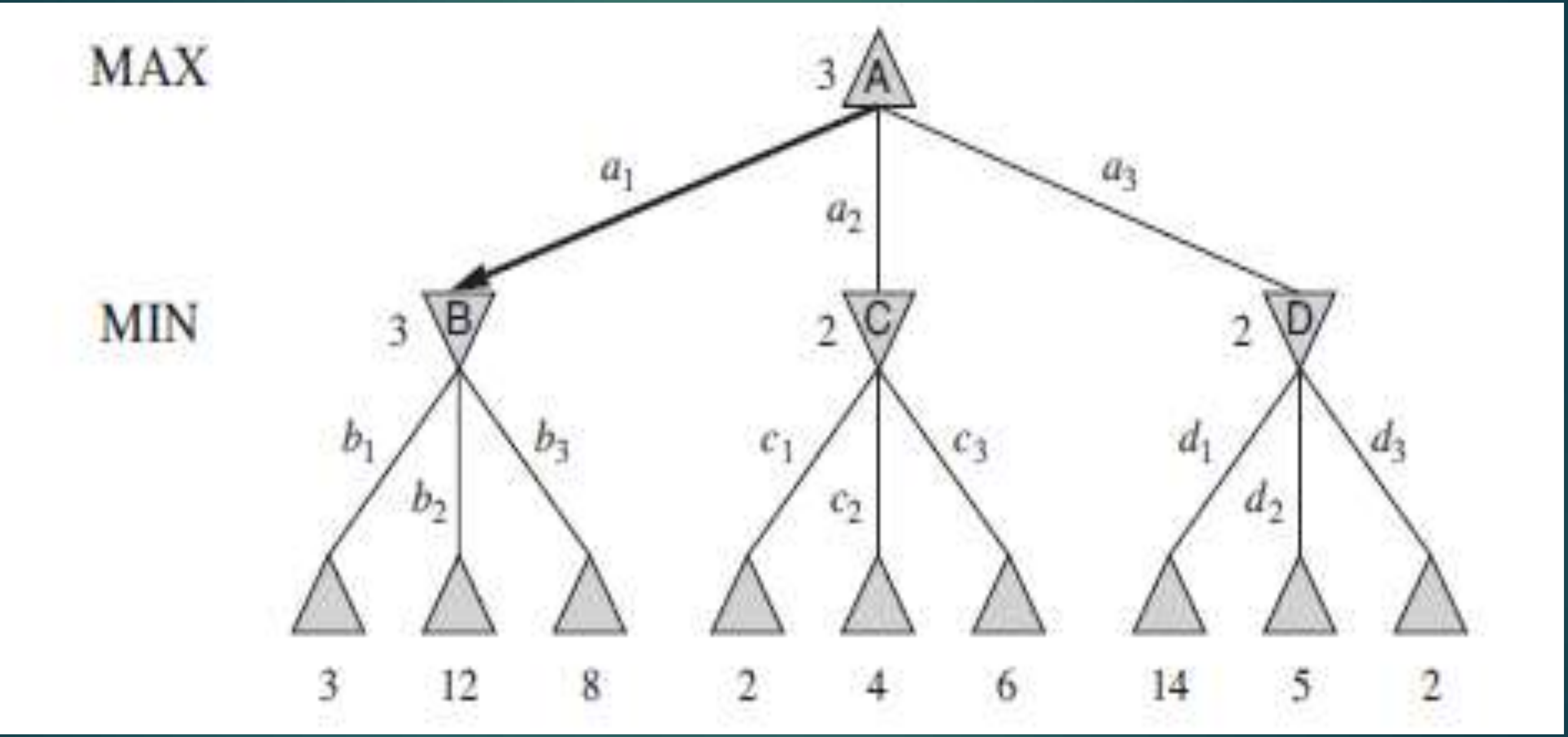
max

min

max


A
23
level 1 (maximizing)
Backing-up
of values
12
B
C
23
level 2 (minimizing)
22
D
12
E
25
F
G
23
level 3 (maximizing)
H
I
J
K
L
M
N
O
19
22
12
8
25
-8
13
23

Two-ply game tree



ALPHA–BETA PRUNING

- The problem with minimax search is that the number of game states it has to examine is exponential in the depth of the tree .
- Unfortunately, we can’t eliminate the exponent, but it turns out we can effectively cut it in half by exploring lesser number of nodes.
- It is possible to compute the correct minimax decision without looking at every node in the game tree.
- When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.

Alpha – Beta pruning

→Cut-off search by exploring less no. of nodes