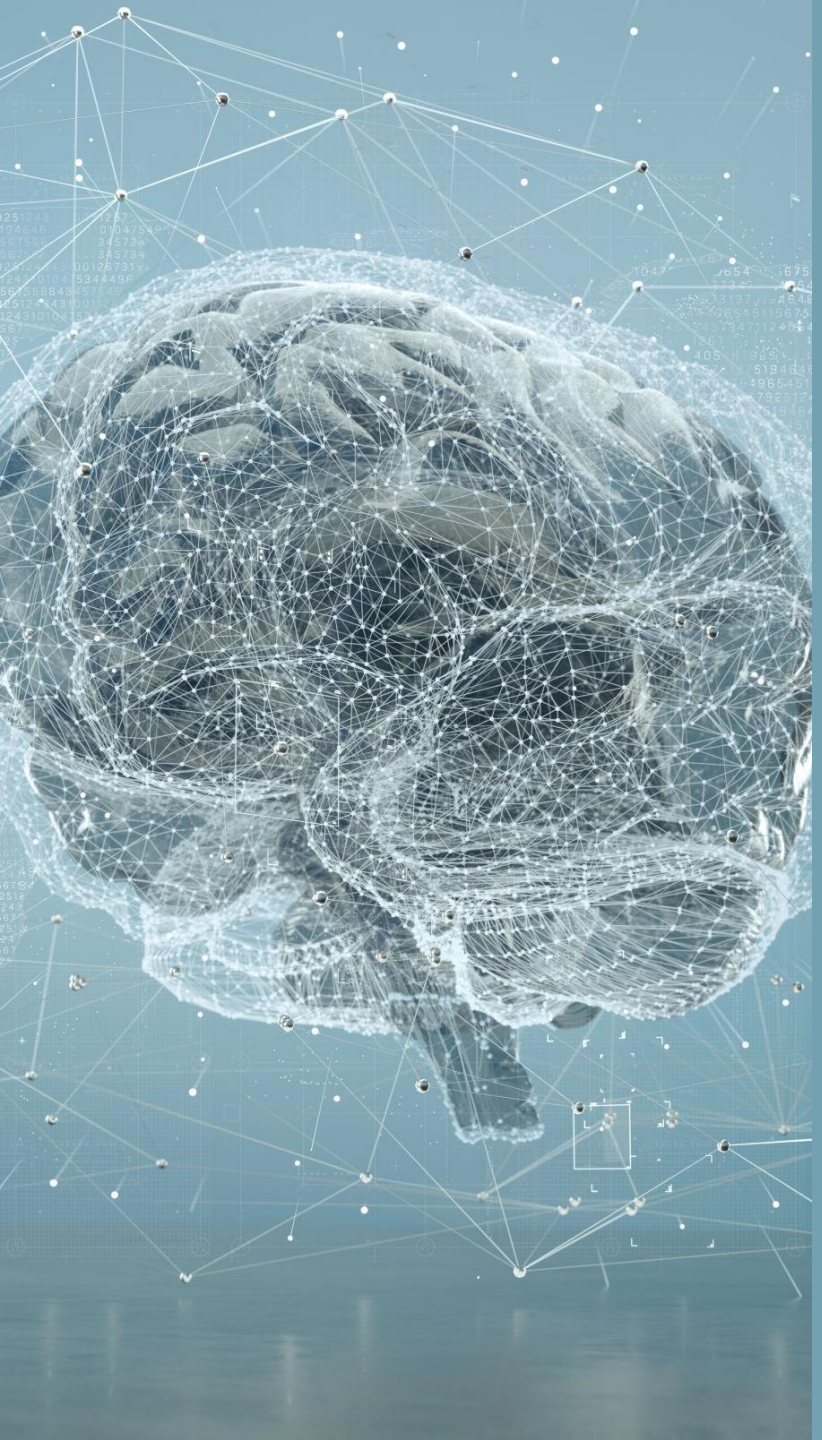


كلية الحاسبات والذكاء الاصطناعي
FACULTY OF COMPUTERS & ARTIFICIAL INTELLIGENCE
BENHA UNIVERSITY

ARTIFICIAL INTELLIGENCE
LAB 06 – ADVERSARIAL SEARCH

Eng. Sammer Kamal

Eng. Yousef Elbaroudy

2024/2025

GUIDELINES

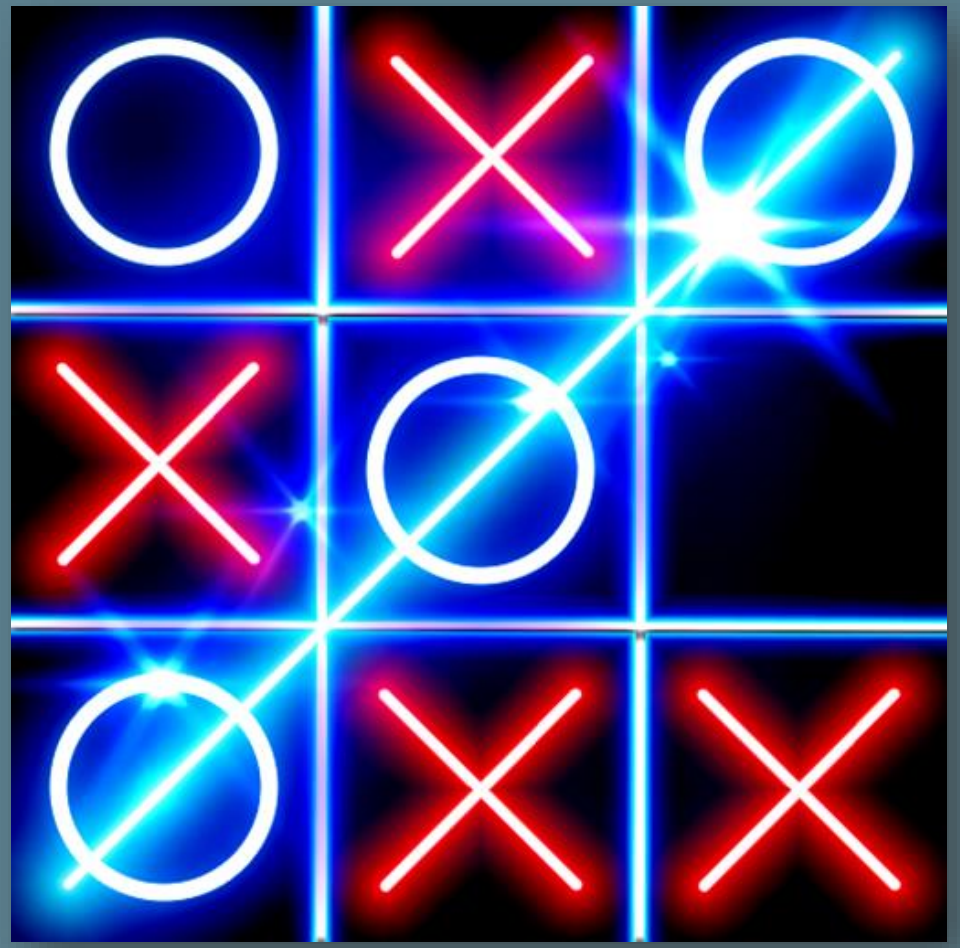
You don’t need to memorize what will be mentioned. Instead, try to understand

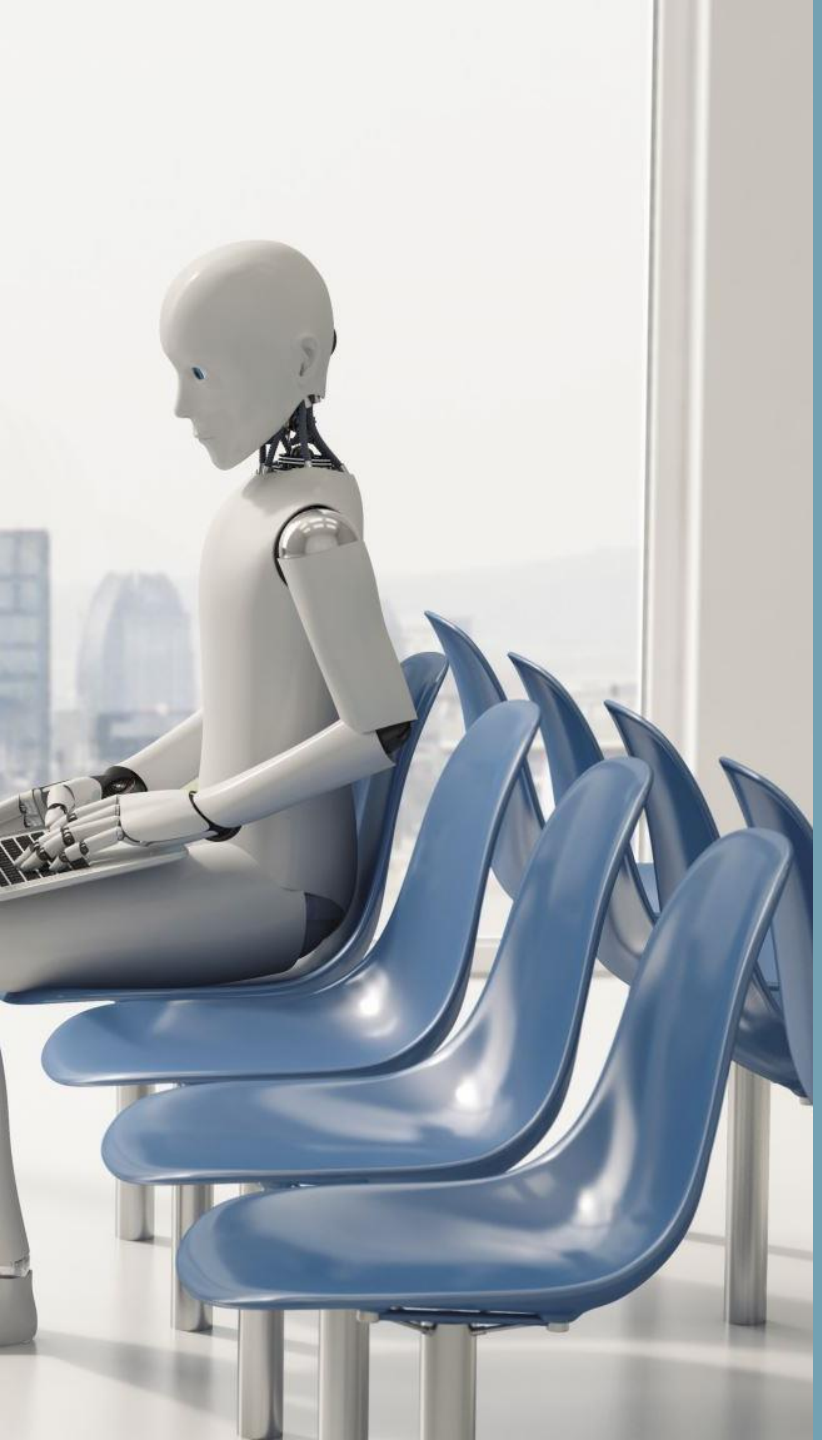
Each PowerPoint Presentation will be available as PDF file on Google Drive Folder of the Course

Give attention as much as you can for this lab, as long as you will implement a cooperative project with your mates

ADVERSARIAL SEARCH

Design your own AI Game





ADVERSARIAL SEARCH

A type of search in artificial intelligence used for decision-making in environments where *multiple agents* (often two players) compete with each other

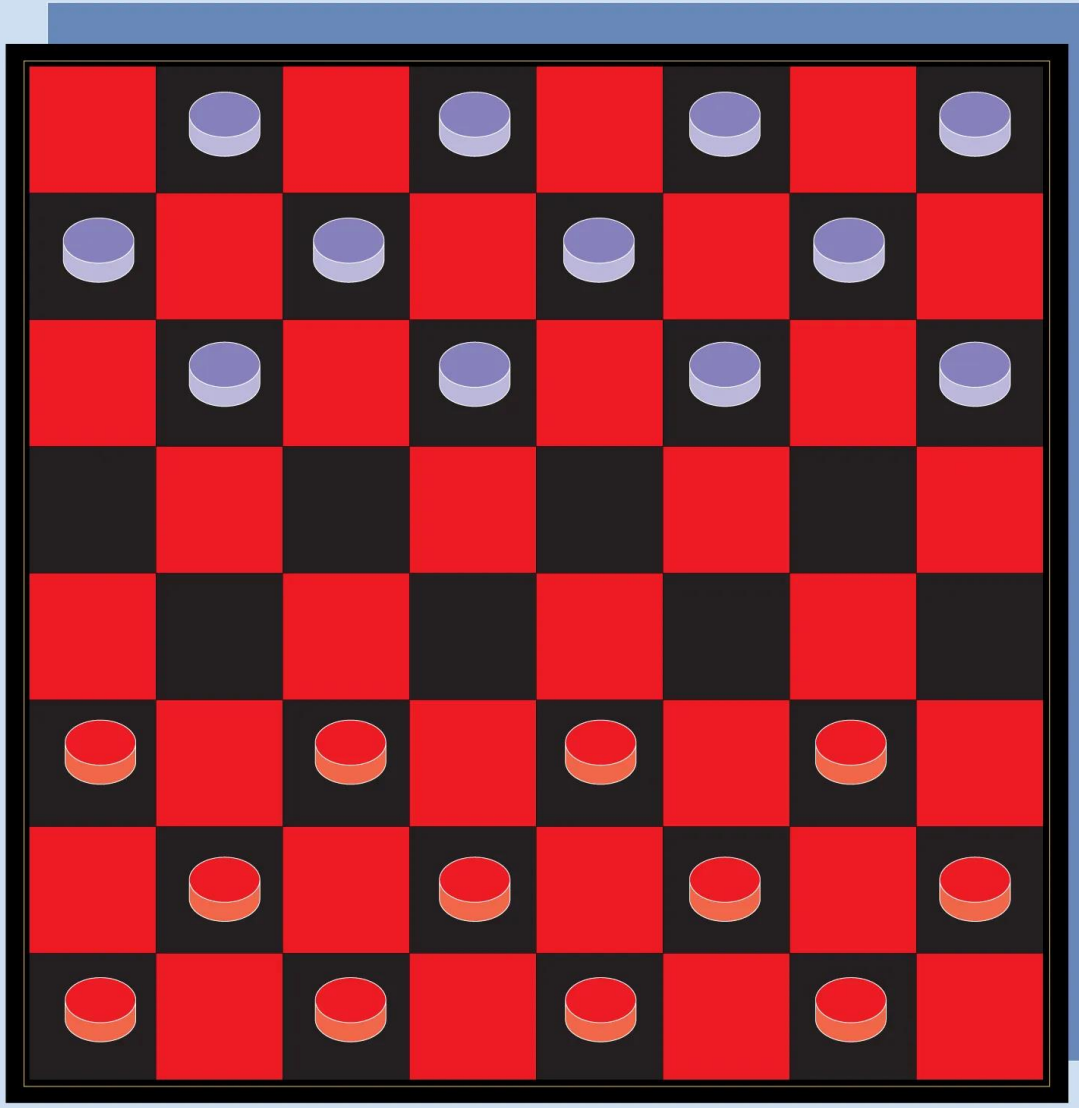
each player's goal is to maximize their own chances of winning while minimizing the chances of the opponent.

ANALYZE THE ENVIRONMENT

ADVERSARIAL SEARCH: FULLY OBSERVABLE ENVIRONMENT

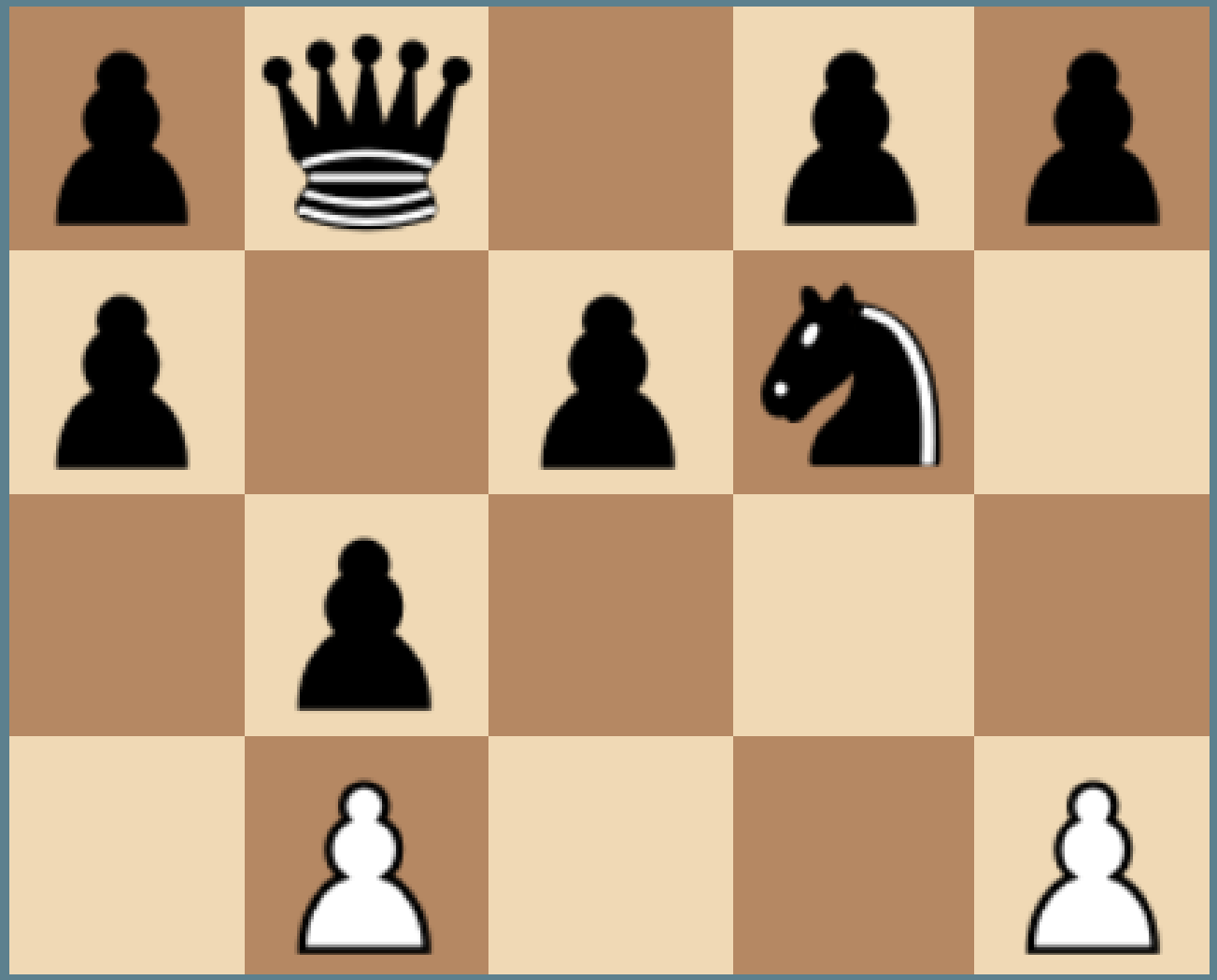
ADVERSARIAL SEARCH: DISCRETE ENVIRONMENT

- A finite set of states
- A finite set of legal actions
- The transition from one state to another is well-defined



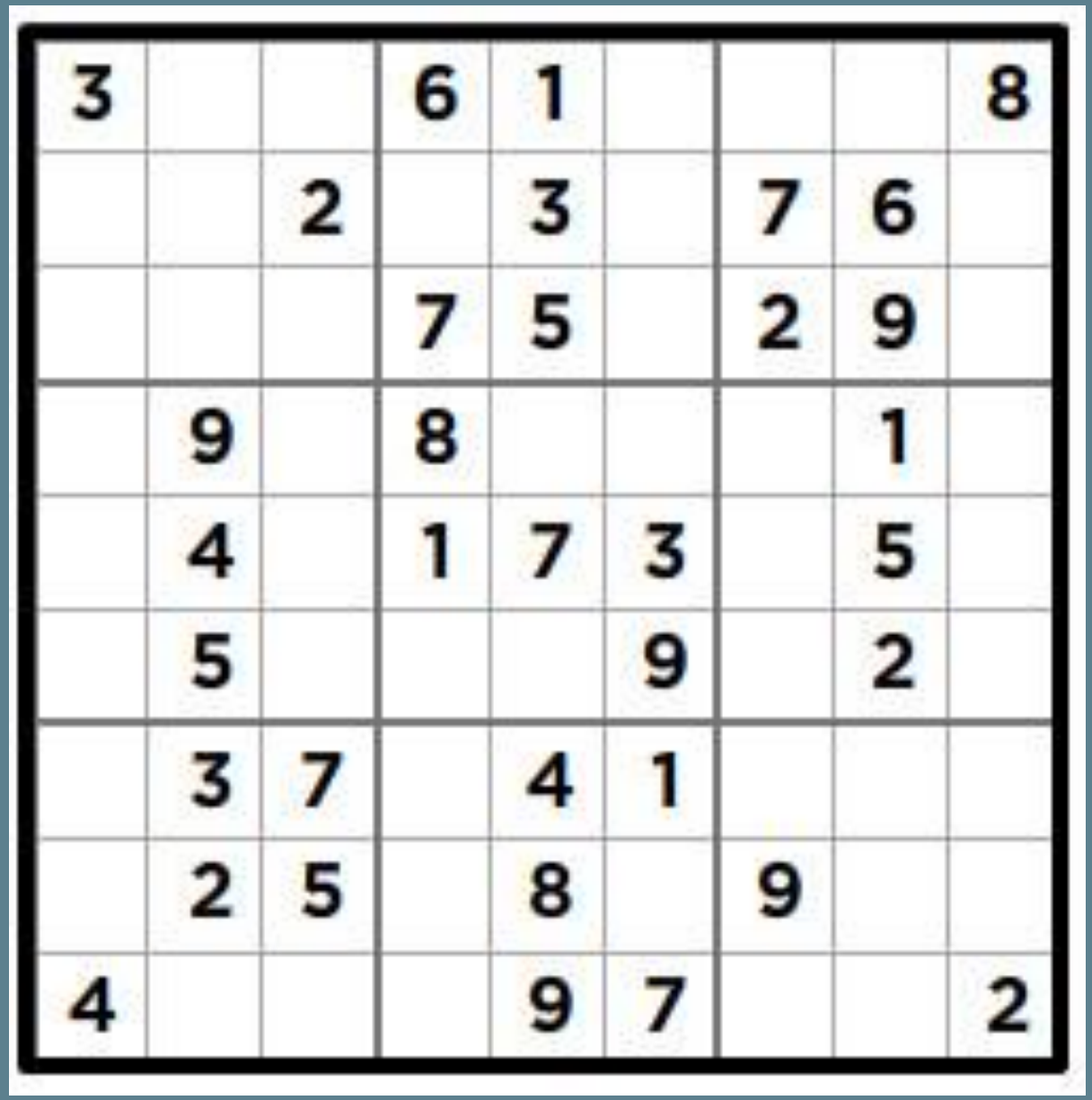
ADVERSARIAL SEARCH: DETERMINISTIC ENVIRONMENT

- The outcome of every action is fully predictable and does not involve any randomness.



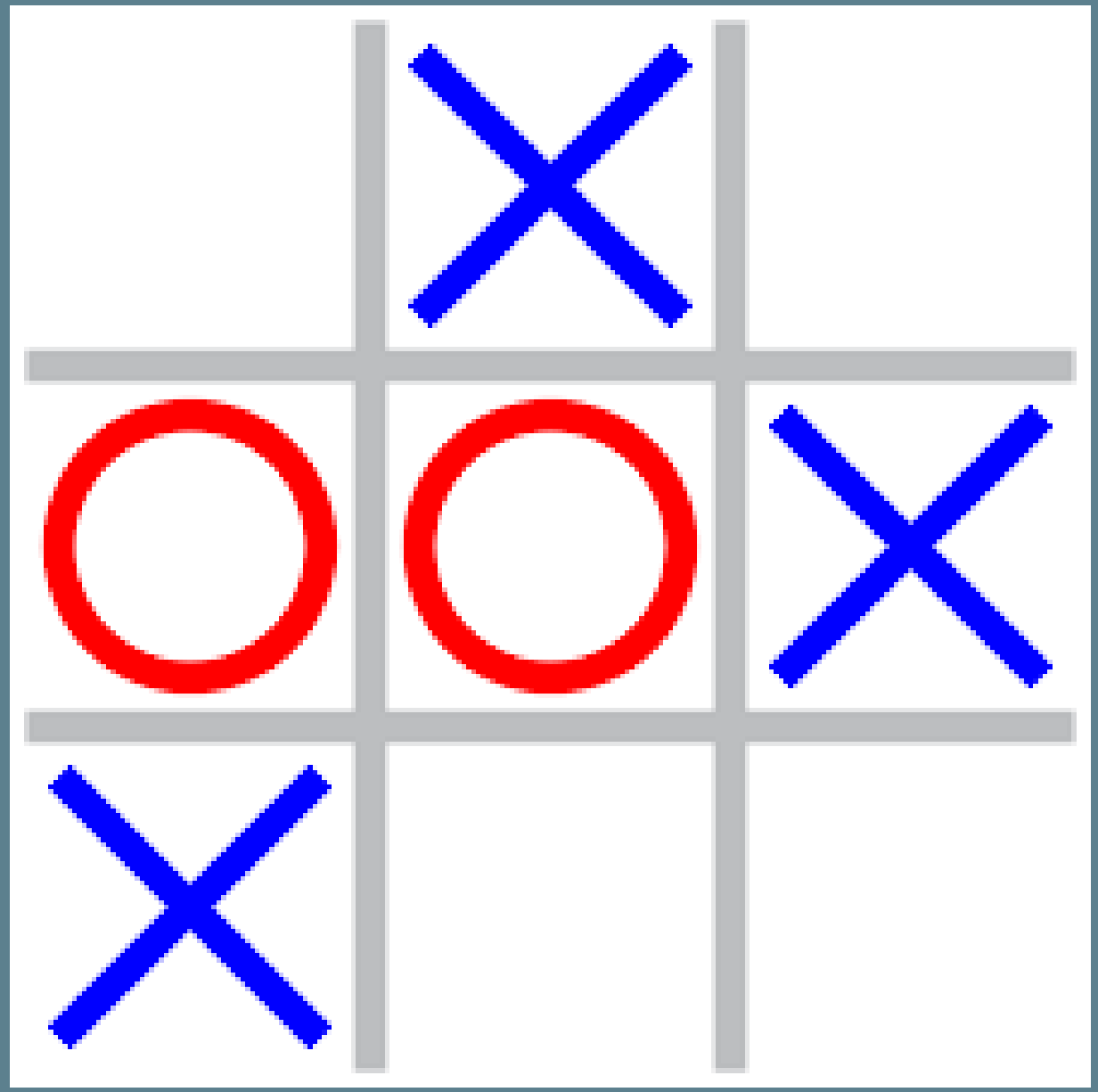
ADVERSARIAL SEARCH: STATIC ENVIRONMENT

- The environment does not change while the agent is thinking or making a decision.



ADVERSARIAL SEARCH:
SEQUENTIAL ENVIRONMENT

- The current decision affects future decisions. Actions are part of a sequence, and their consequences unfold over time.



ADVERSARIAL SEARCH: MULTI-AGENTS

- More than one agent exists and interacts within the same environment – each with its own goals, perceptions, and actions.
- Can be agents that work together toward a shared goal (called Cooperative)
- Can be one agent's gain = another's loss (called Competitive/Adversarial)

ADVERSARIAL ENVIRONMENT PROPERTIES

Fully-observable

Discrete

Deterministic

Sequential

Static

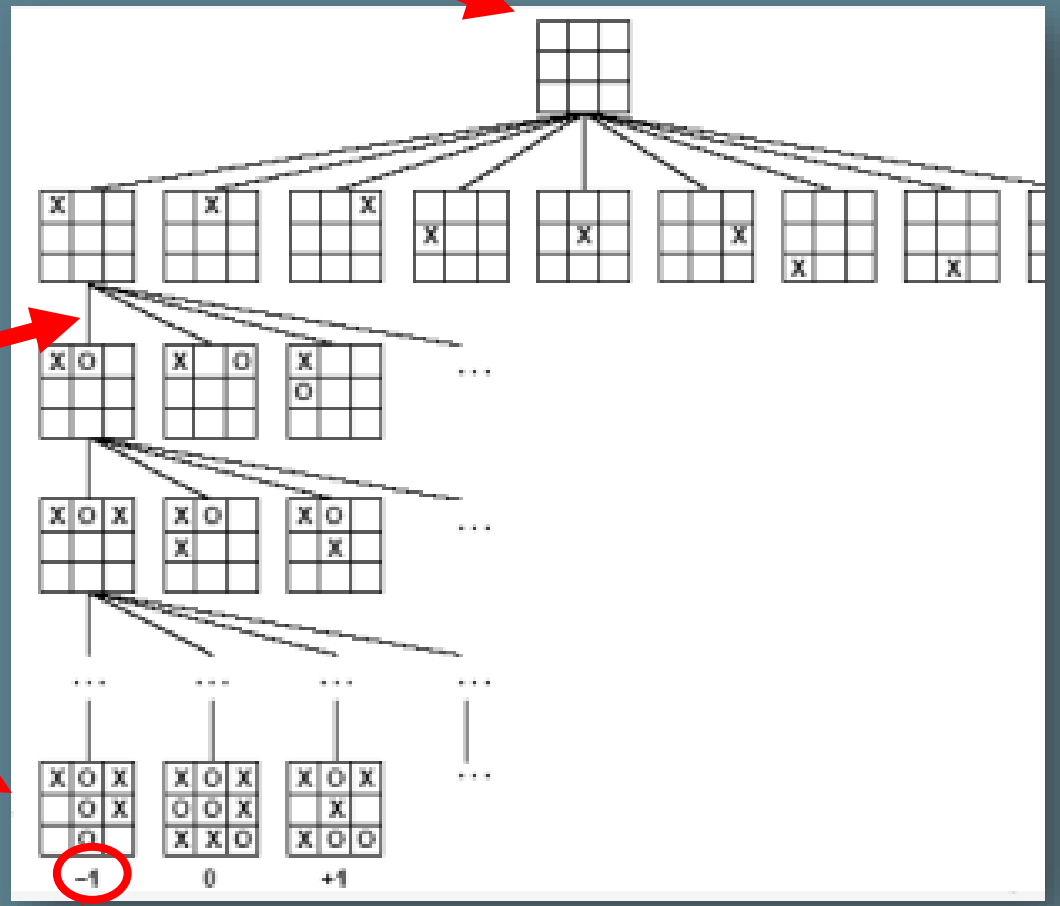
Multi-agent (Competitive)

GAMES VS. SEARCH

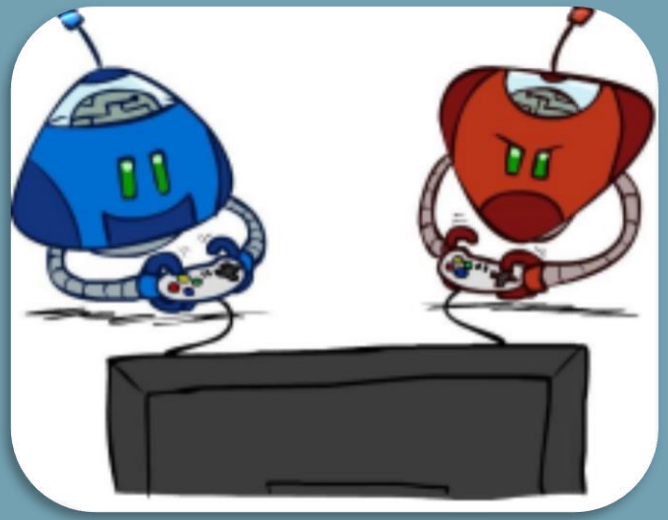
Feature	Games	Search Problems
Number of agents	Two or more (multi-agent)	Single-agent
Environment type	Adversarial (opponents involved)	Non-adversarial (No opponent)
Agent goals	Conflicting (Win/Loss)	Goal is to reach a specific state or solve a task
Nature of moves	Players alternate turns	Agents makes a sequence of moves alone
Uncertainty	From opponent's decisions	Usually determinstic
Solution method	Minimax, Alpha-beta pruning	DFS, BFS, A* … etc
Objective	Maximize/Minimize score or win	Reach a goal state efficiently
State evaluation	Often uses heuristic evaluation functions	Can use cost functions and heuristics
Turns/Sequential plays	Yes – agents take turns	No – agent moves continuously toward goal

ADVERSARIAL KEY CONCEPTS (BUILD THE ENV.)

- Initial State
  - The starting position of the game or the board
- Operators/Decisions/Edges/Arcs/Transitions
  - Legal moves that the player can make
- Terminal Nodes
  - Leaf nodes in the tree, which indicates that the game is over
- Utility function/Reward (Heuristic/Hint)
  - Payoff function (objective function)
  - Value of the outcome of the game
  - Ex: for tic-tac-toe it is 1 for winning, -1 for losing or 0 for draw



THE PROCEDURE IN OUR COURSE



- We will name two players: MAX and MIN
- Max moves first and they take turns until the game is over
- Max will search for the best move to maximize the utility (WINNING)
- Min will search for the best move to minimize the utility (MAKE MAX LOSE)
- *Perfect Decision Games*: if the search algorithm searches the complete search tree.
- *Imperfect Decision Games*: no complete search, search is cut-off earlier.

ADVERSARIAL SEARCH ALGORITHMS

MINMAX ALGORITHM

MINMAX ALGORITHM

Remember !

Max wants to MAXIMIZE the utility to win over Min

Min wants to MINIMIZE the utility to win over Max

Both are playing PERFECTLY (no mistakes are made)

1

Generate the game tree in depth-first order until terminal states are reached.

2

Apply the utility/reward function to the terminal states to get utility value

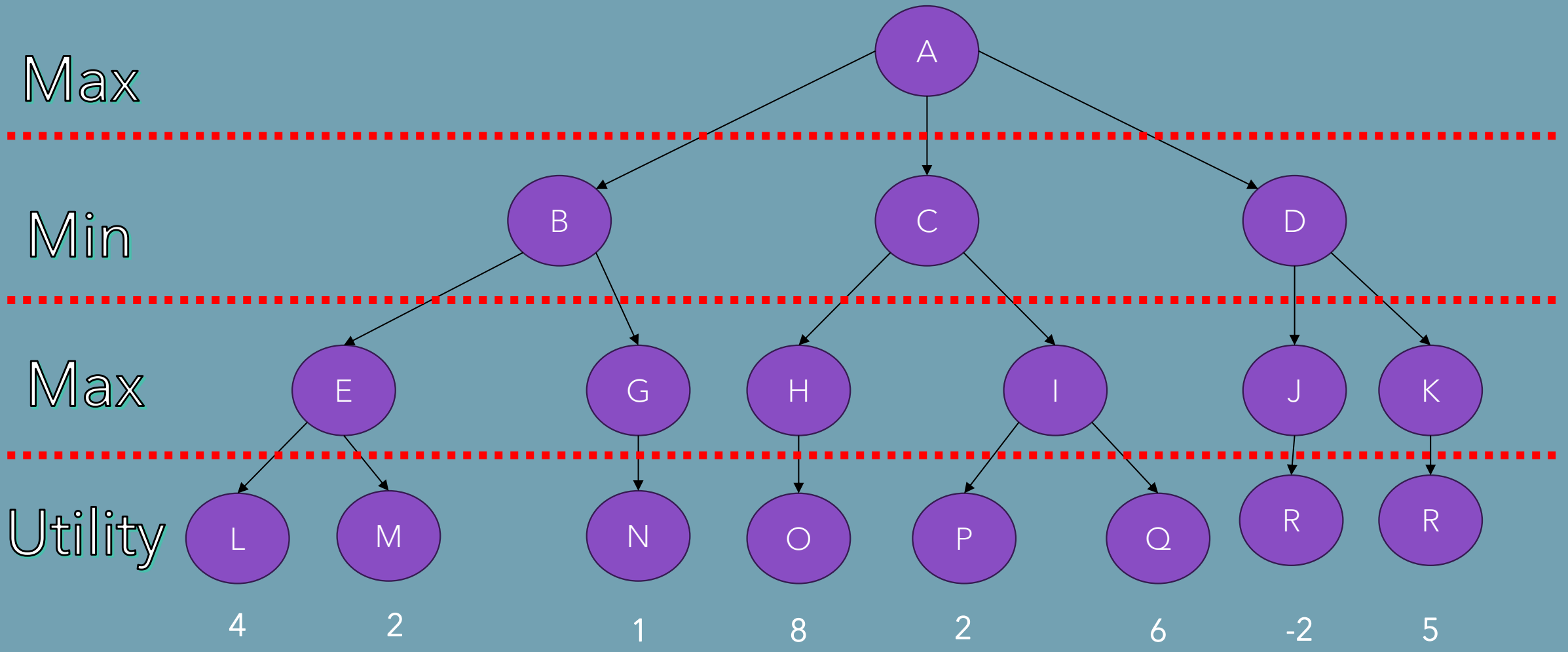
3

MinMax_Value(n):

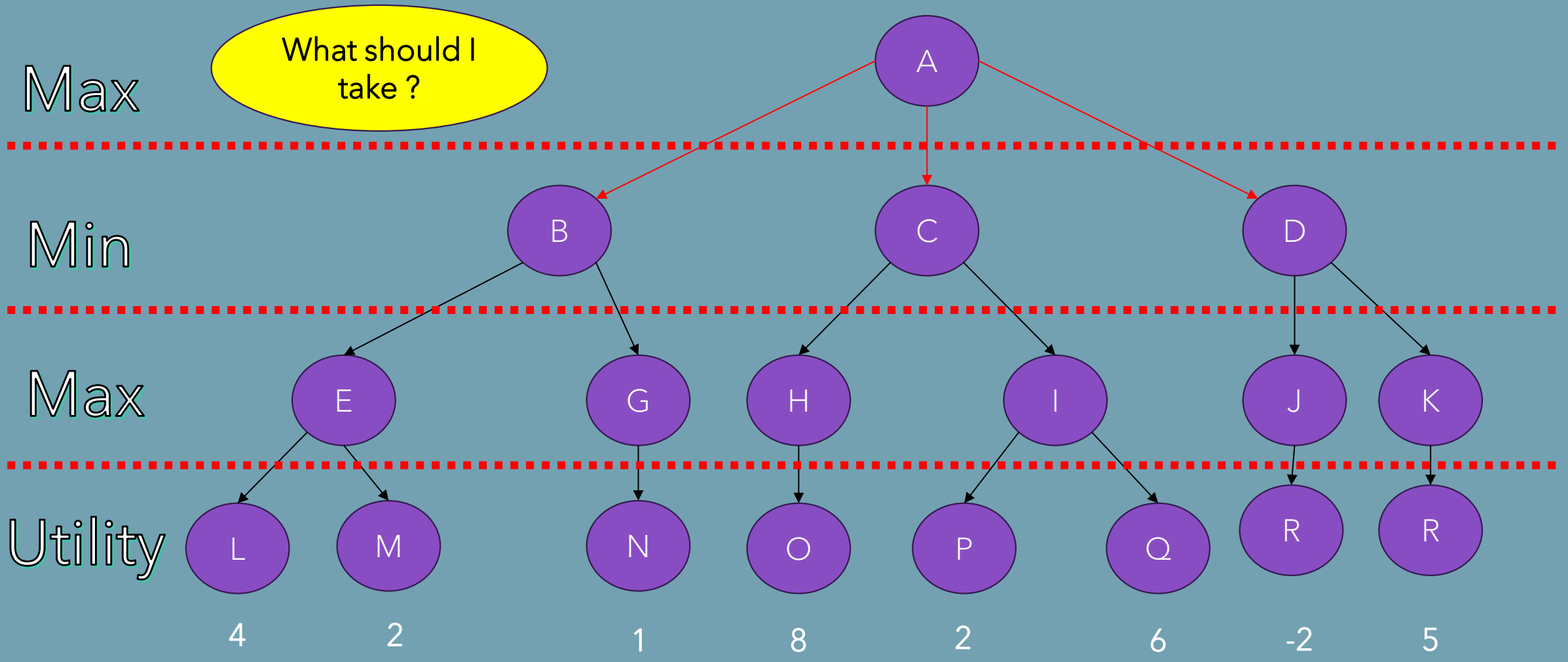
if it is Max turn, maximize the utility of the next state.

If it is Min turn, minimize the utility of the next state.

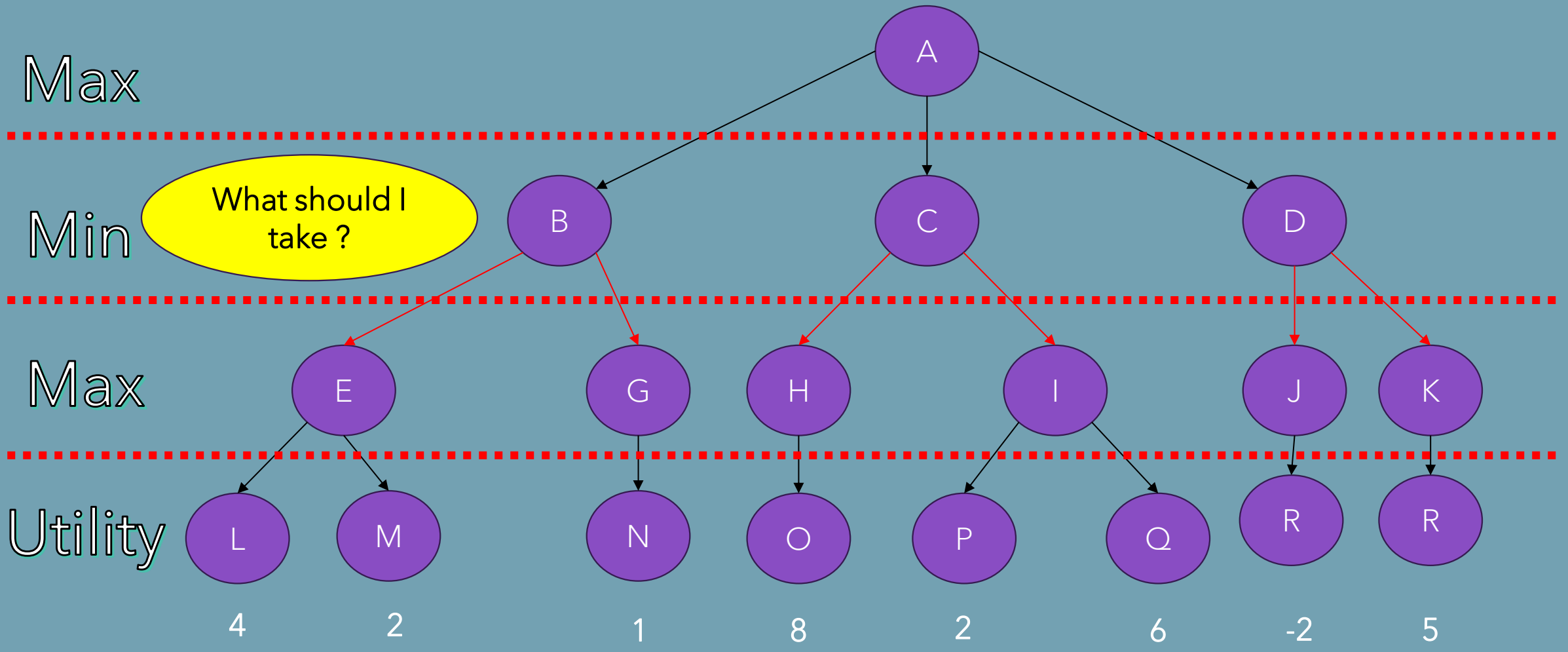
MINMAX ALGORITHM: A SEARCHING PROBLEM



MINMAX ALGORITHM: MAX TURN

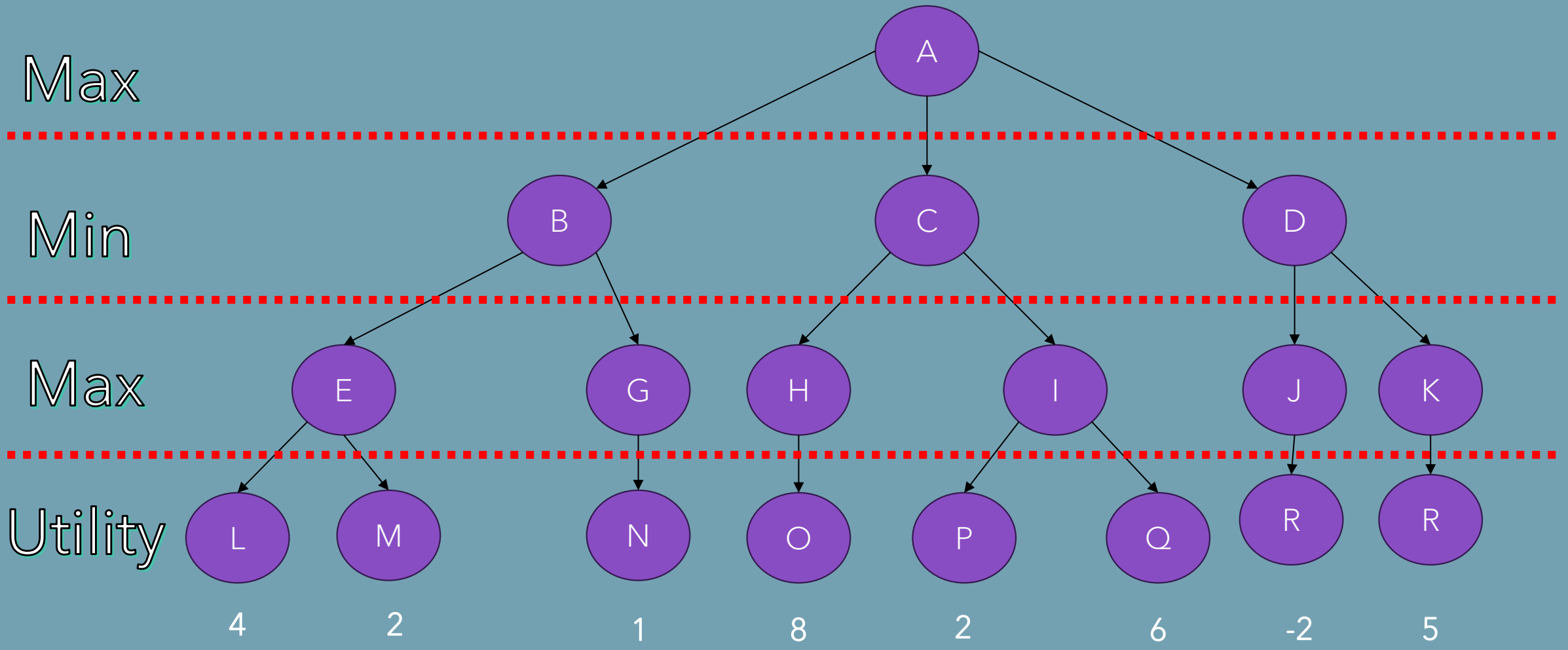


MINMAX ALGORITHM: MIN TURN

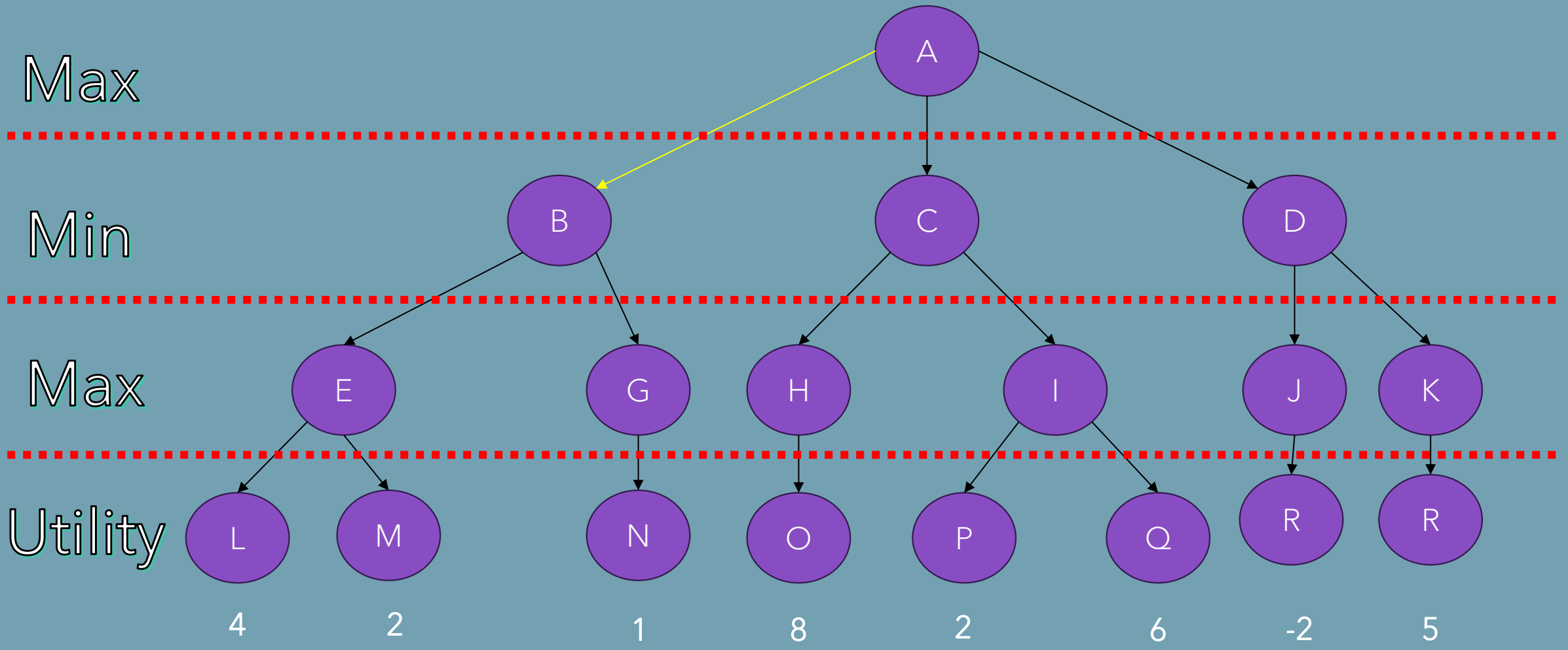


RECURSIVE EVALUATION

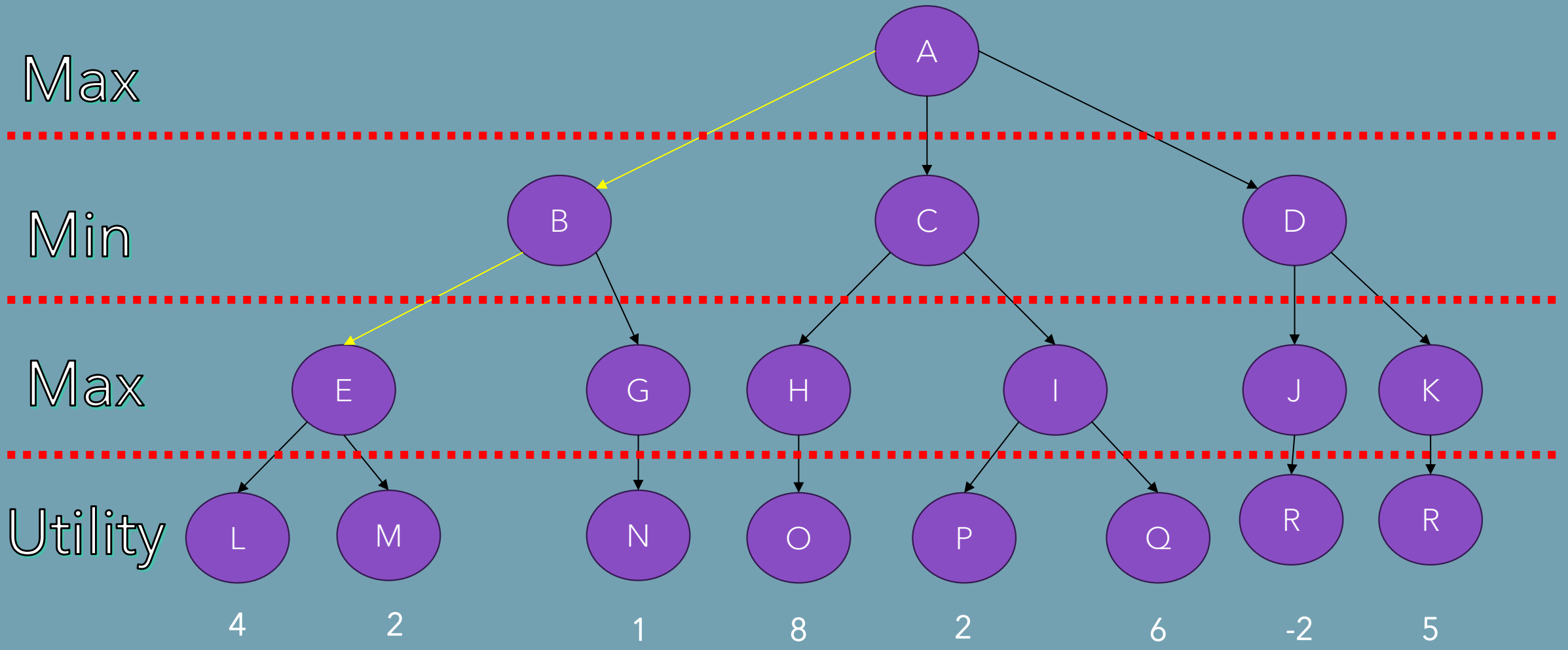
MINMAX ALGORITHM: RECURSIVE FROM ROOT



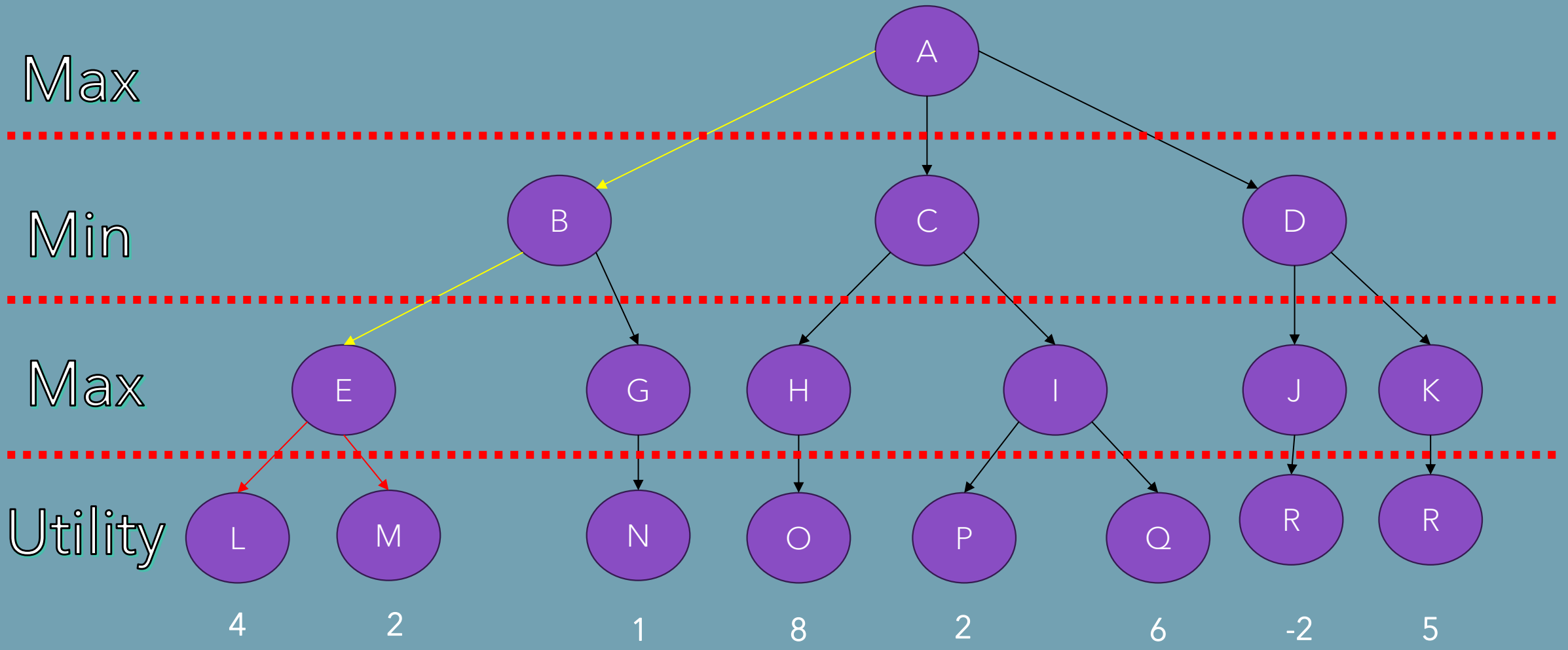
MINMAX ALGORITHM: RECURSIVE FROM ROOT



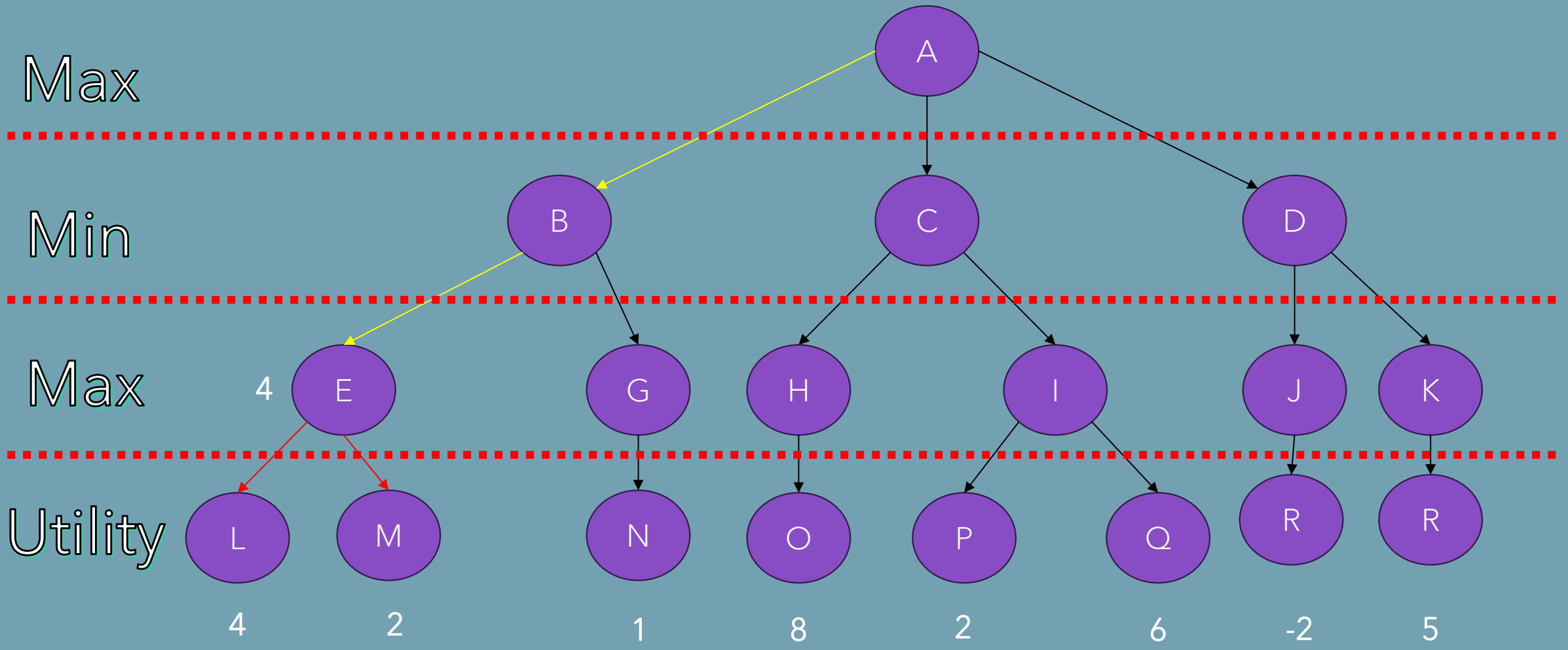
MINMAX ALGORITHM: RECURSIVE FROM ROOT



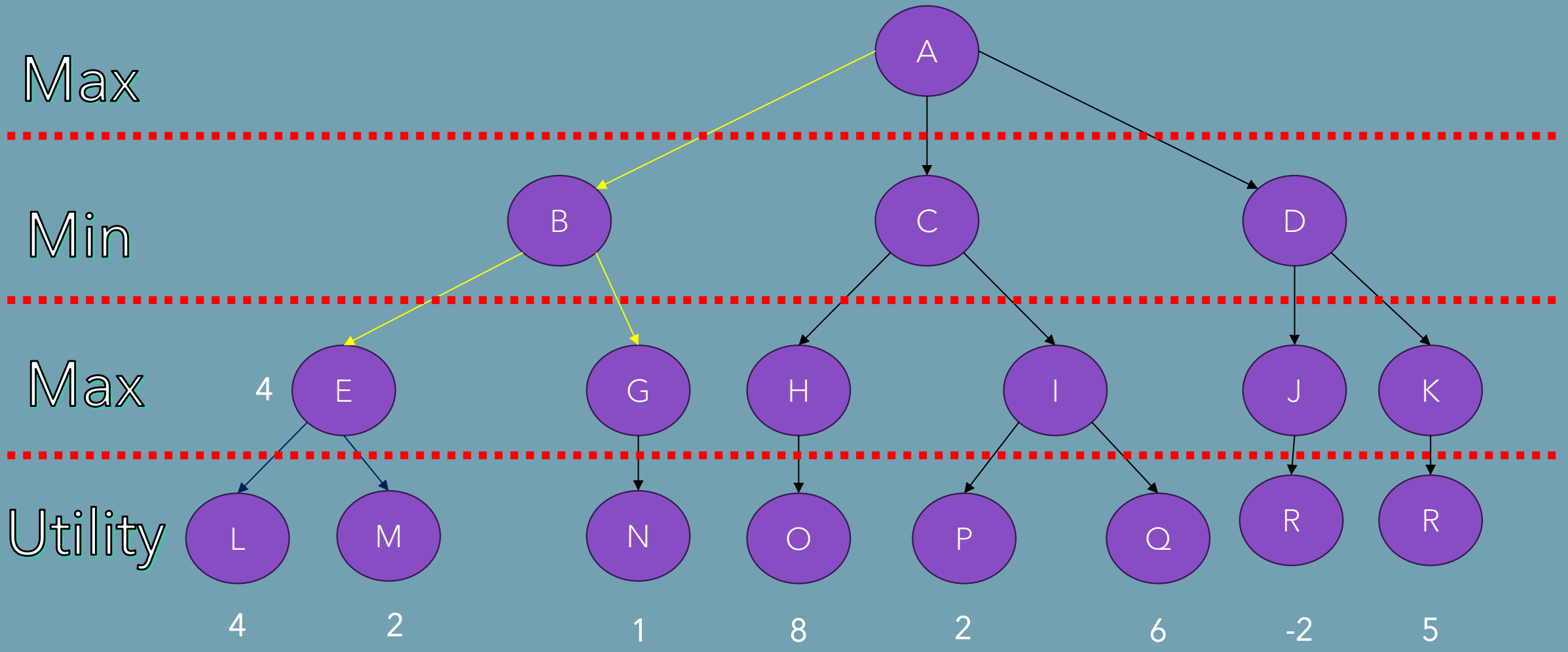
MINMAX ALGORITHM: RECURSIVE FROM ROOT



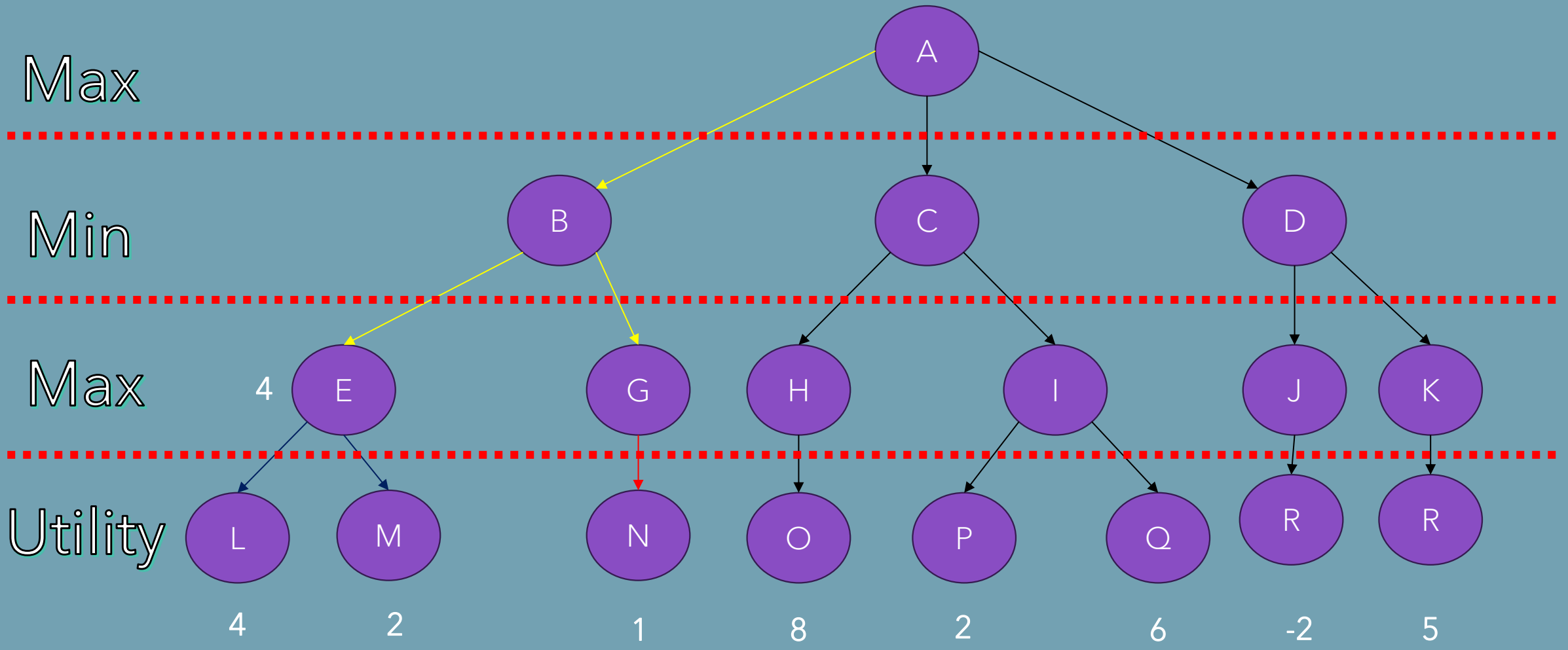
MINMAX ALGORITHM: MAXIMIZE !



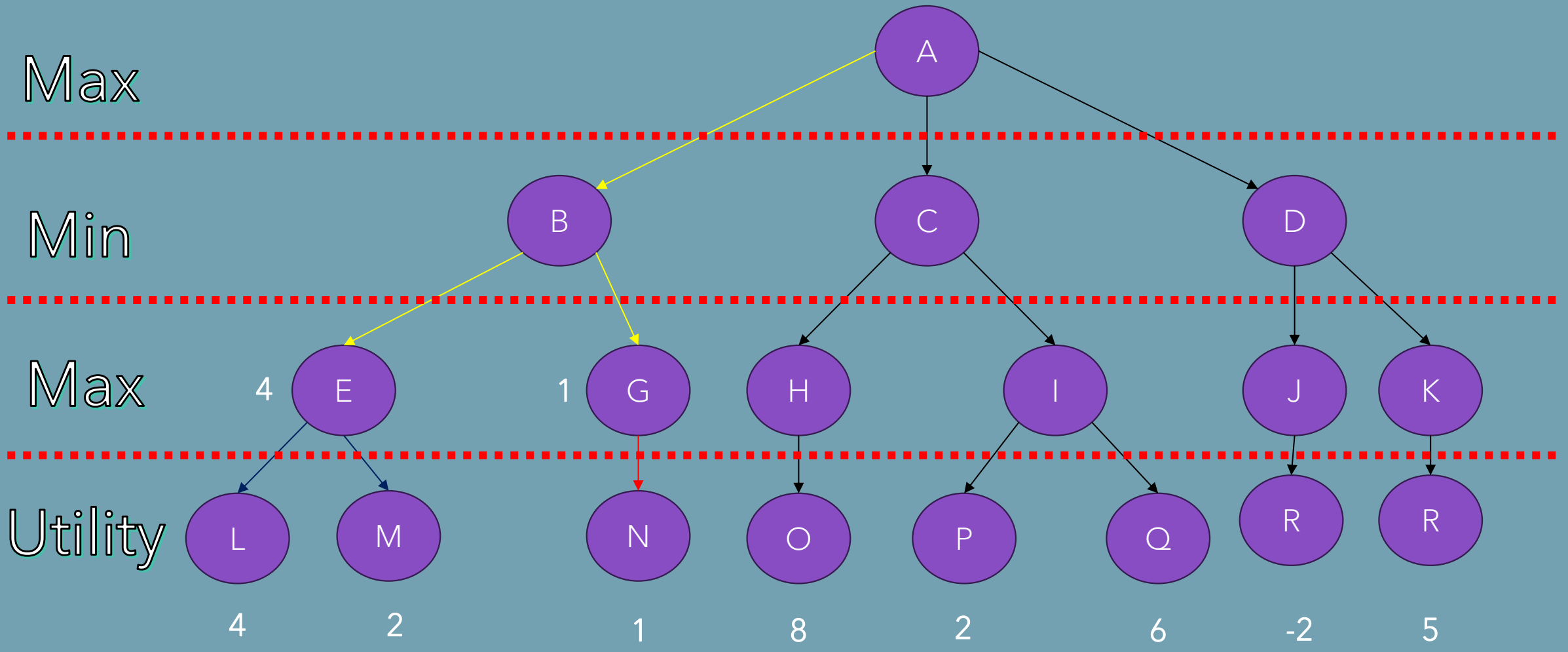
MINMAX ALGORITHM: RECURSIVE FROM ROOT



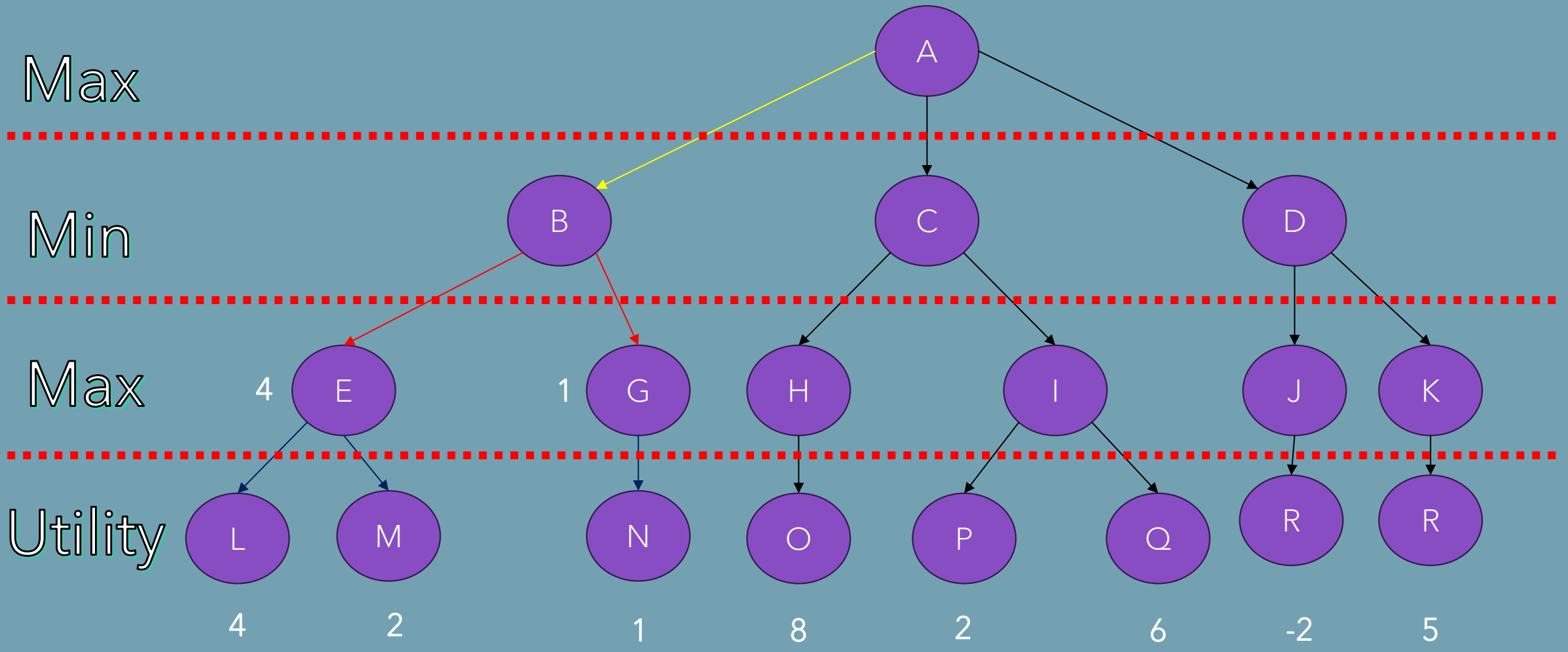
MINMAX ALGORITHM: RECURSIVE FROM ROOT



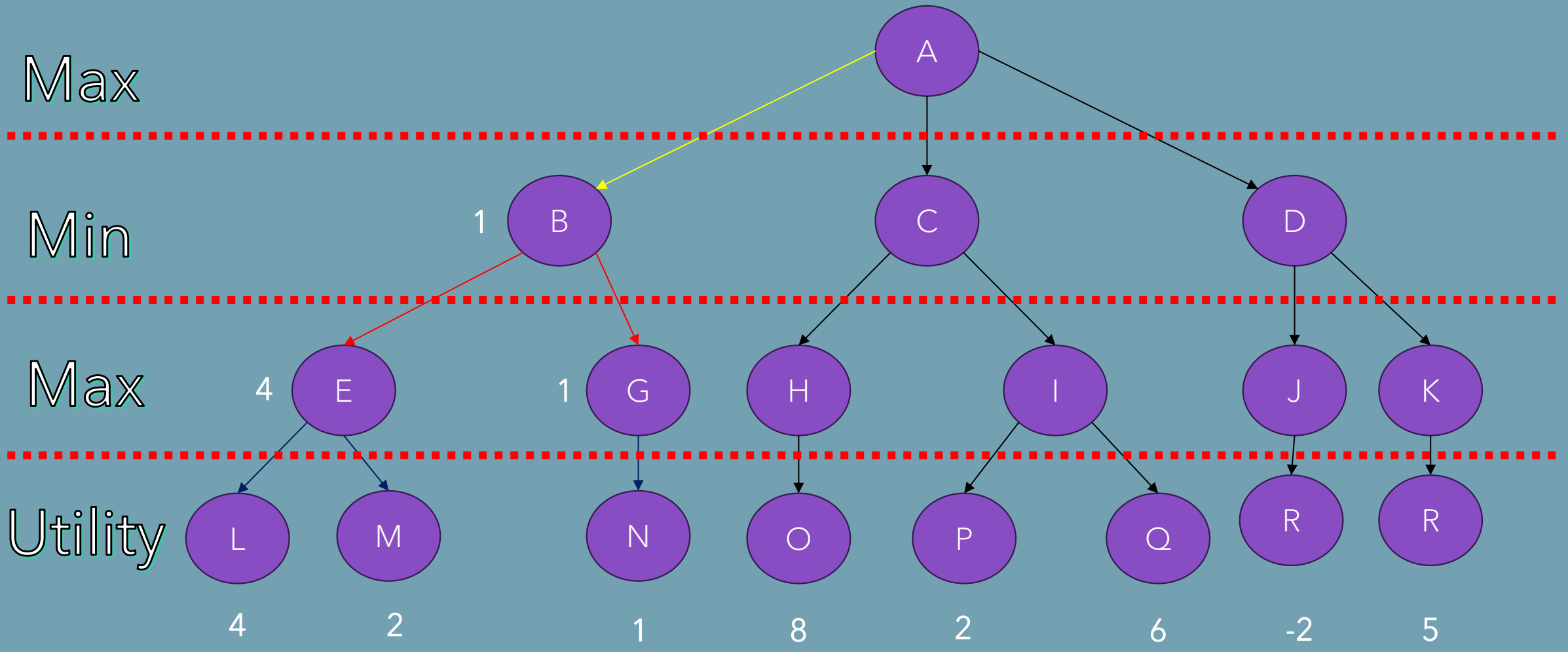
MINMAX ALGORITHM: MAXIMIZE !



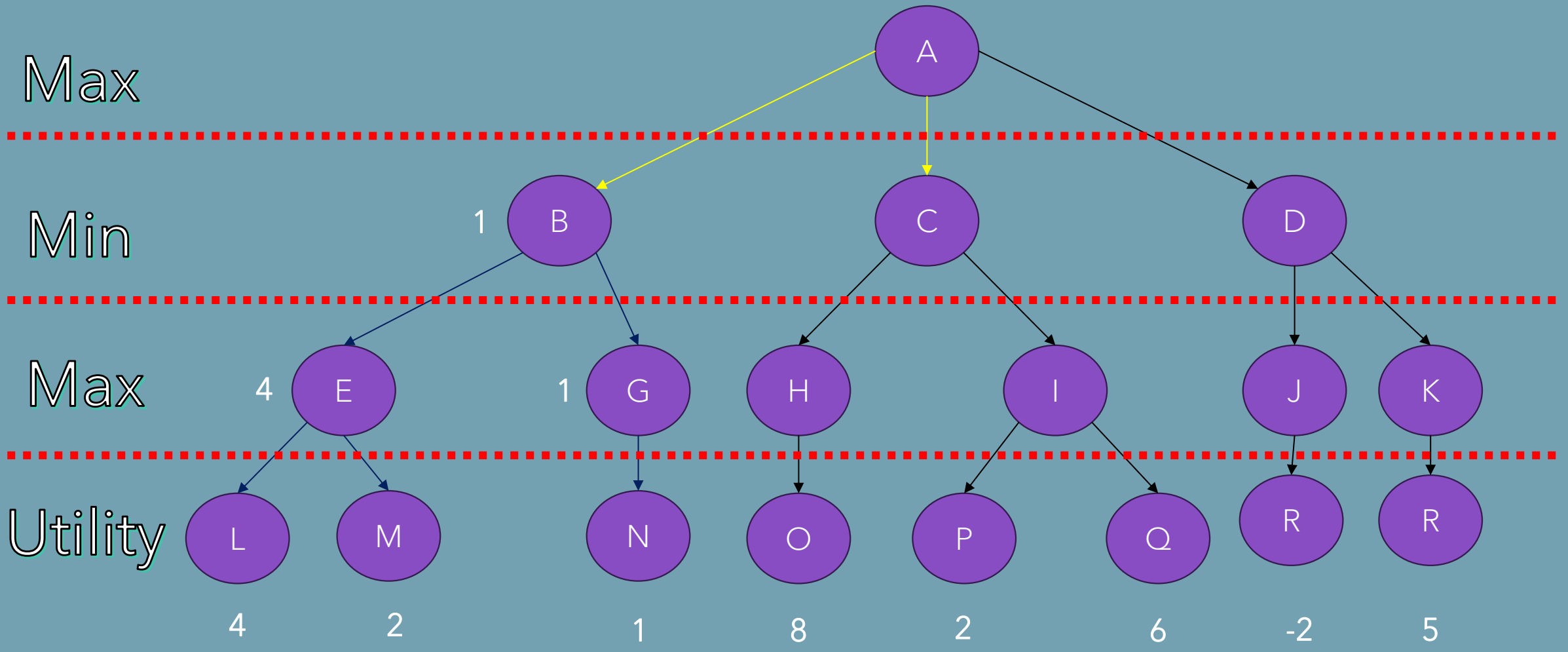
MINMAX ALGORITHM: RECURSIVE FROM ROOT



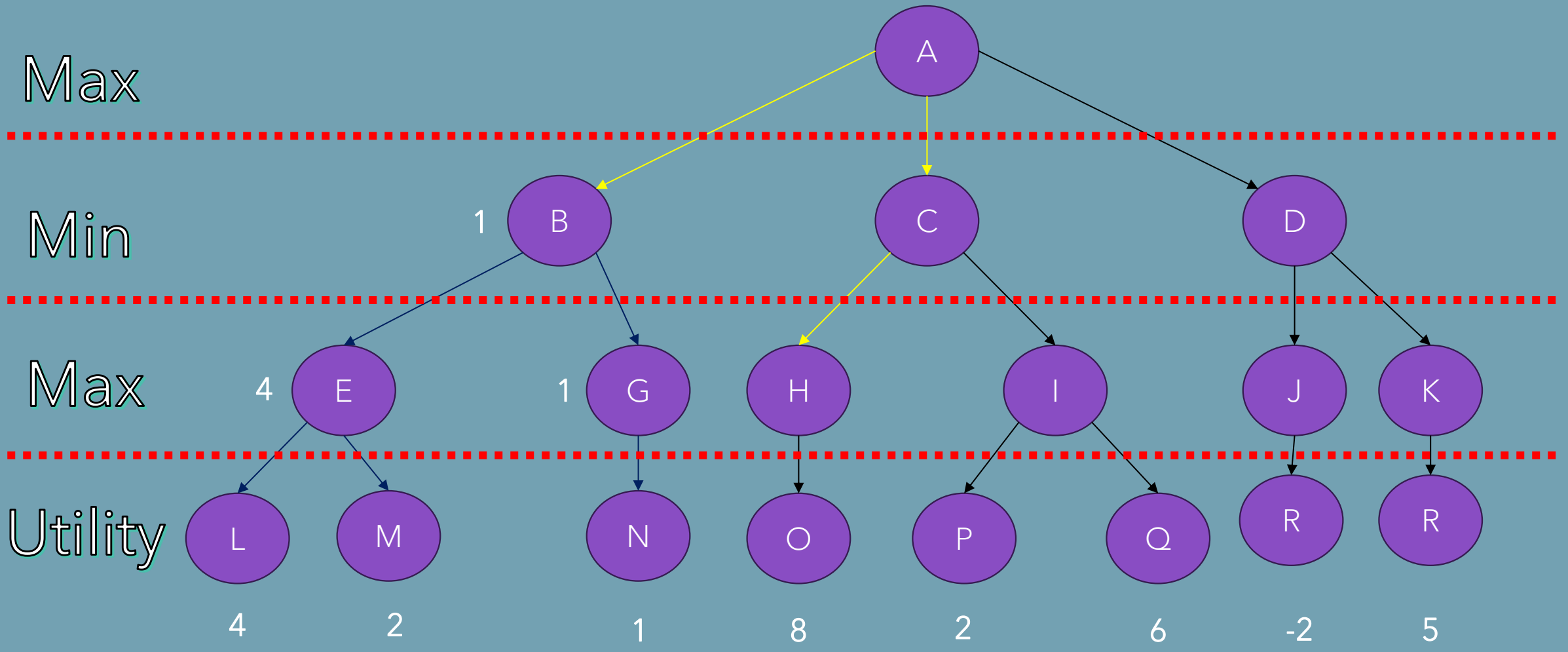
MINMAX ALGORITHM: MINIMIZE !



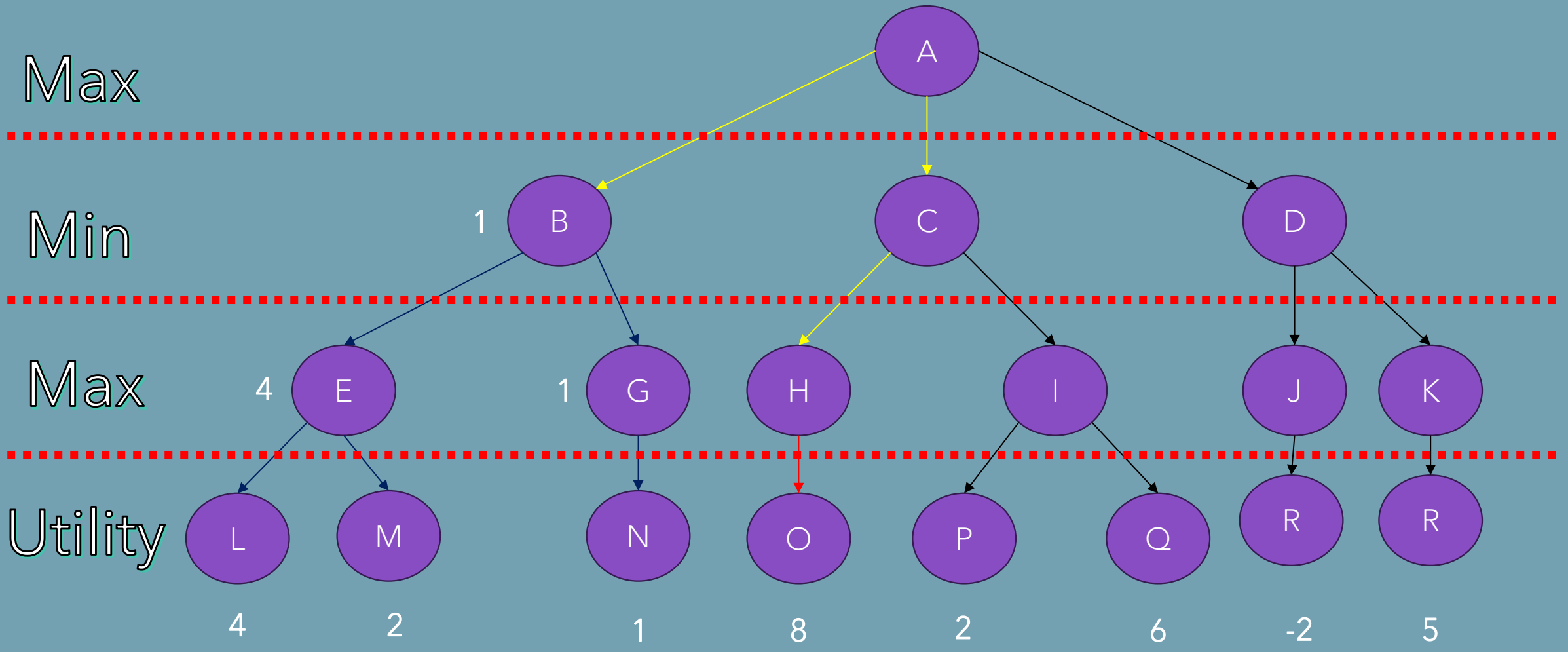
MINMAX ALGORITHM: RECURSIVE FROM ROOT



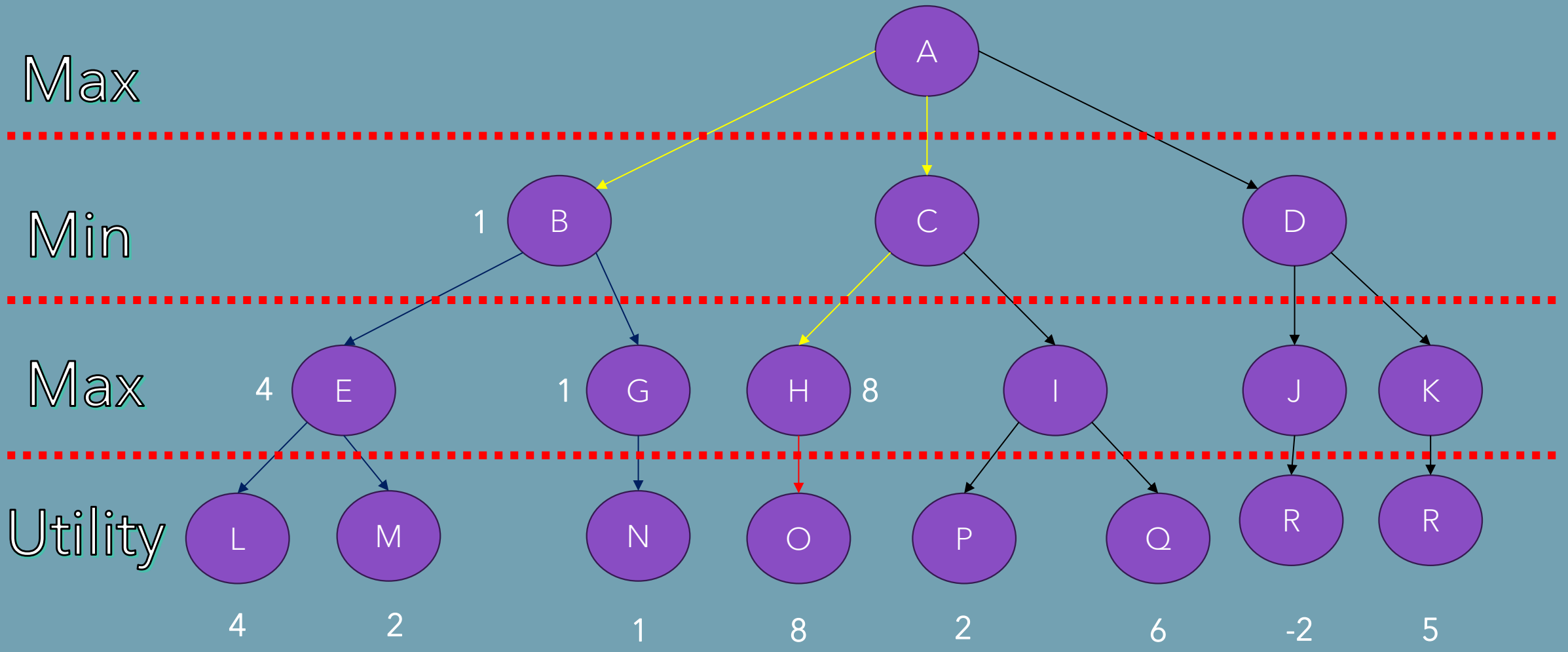
MINMAX ALGORITHM: RECURSIVE FROM ROOT



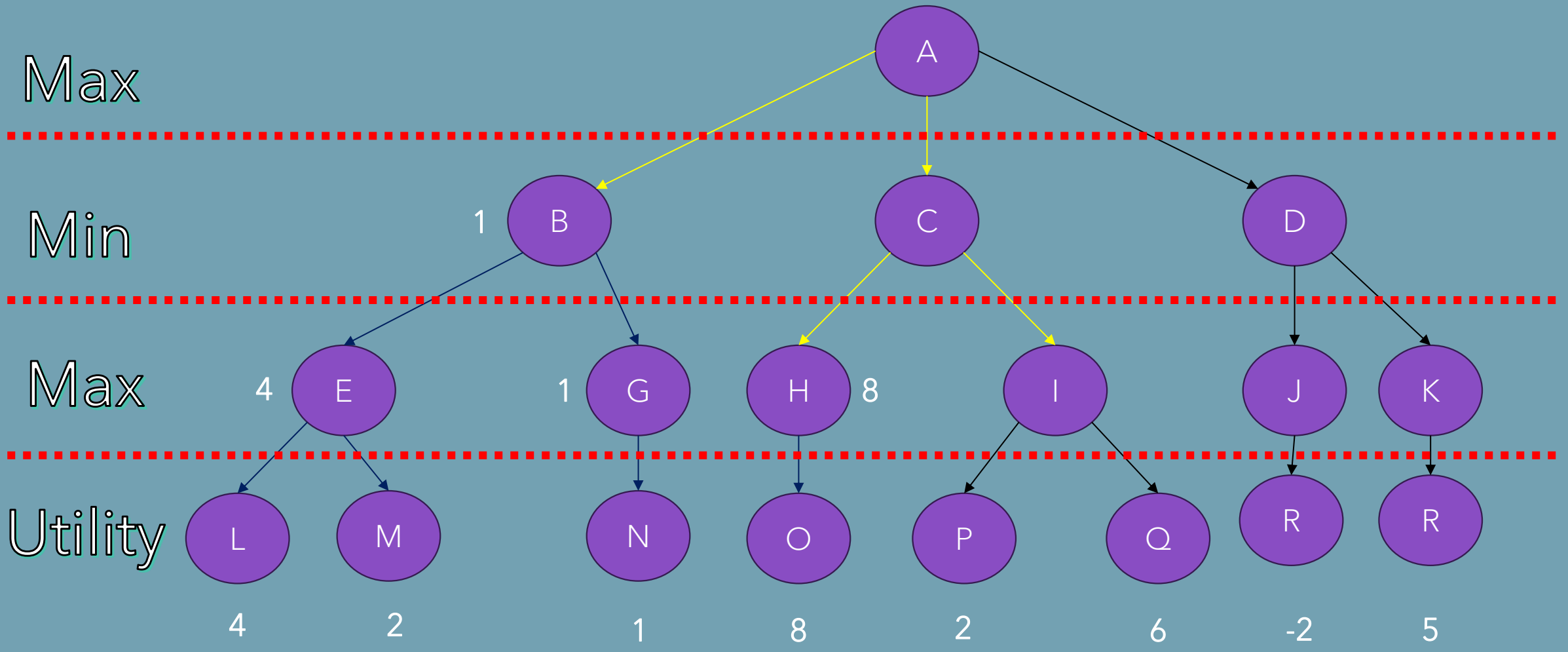
MINMAX ALGORITHM: RECURSIVE FROM ROOT



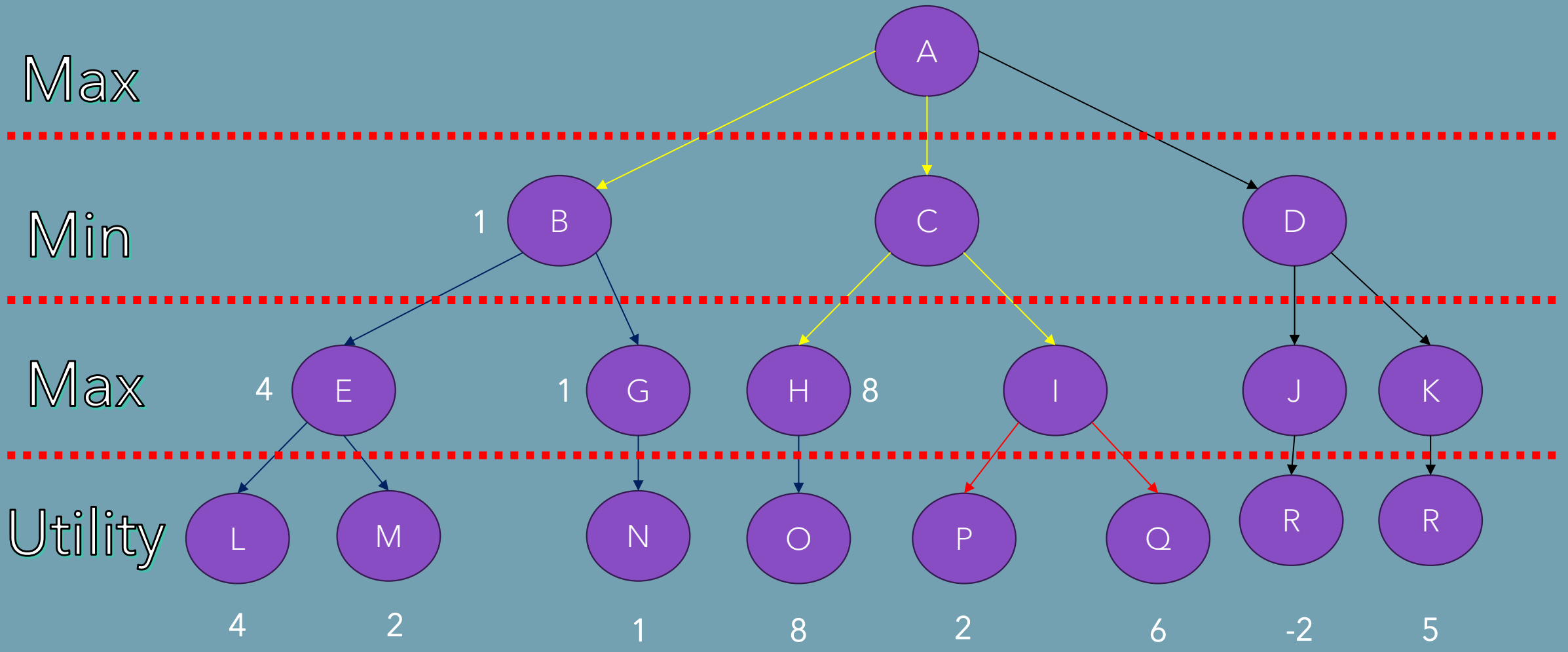
MINMAX ALGORITHM: MAXIMIZE !



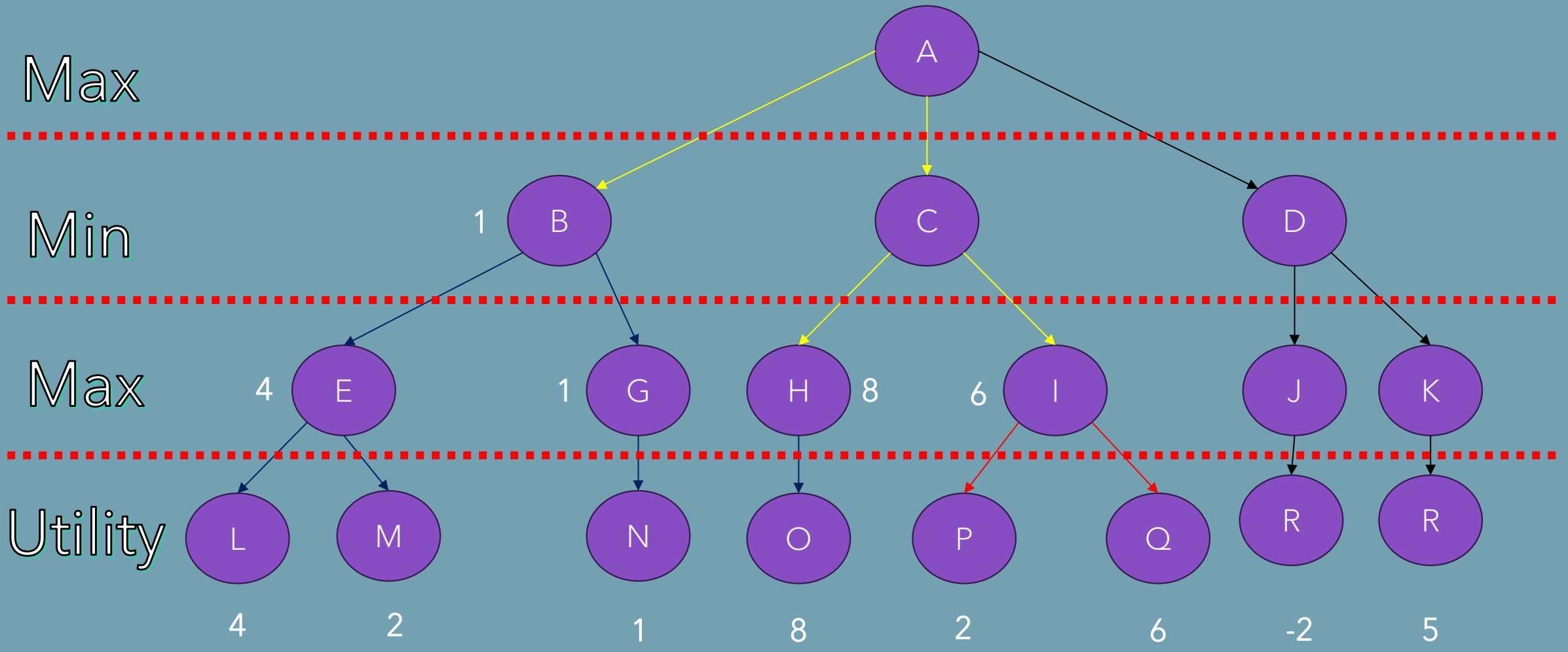
MINMAX ALGORITHM: RECURSIVE FROM ROOT



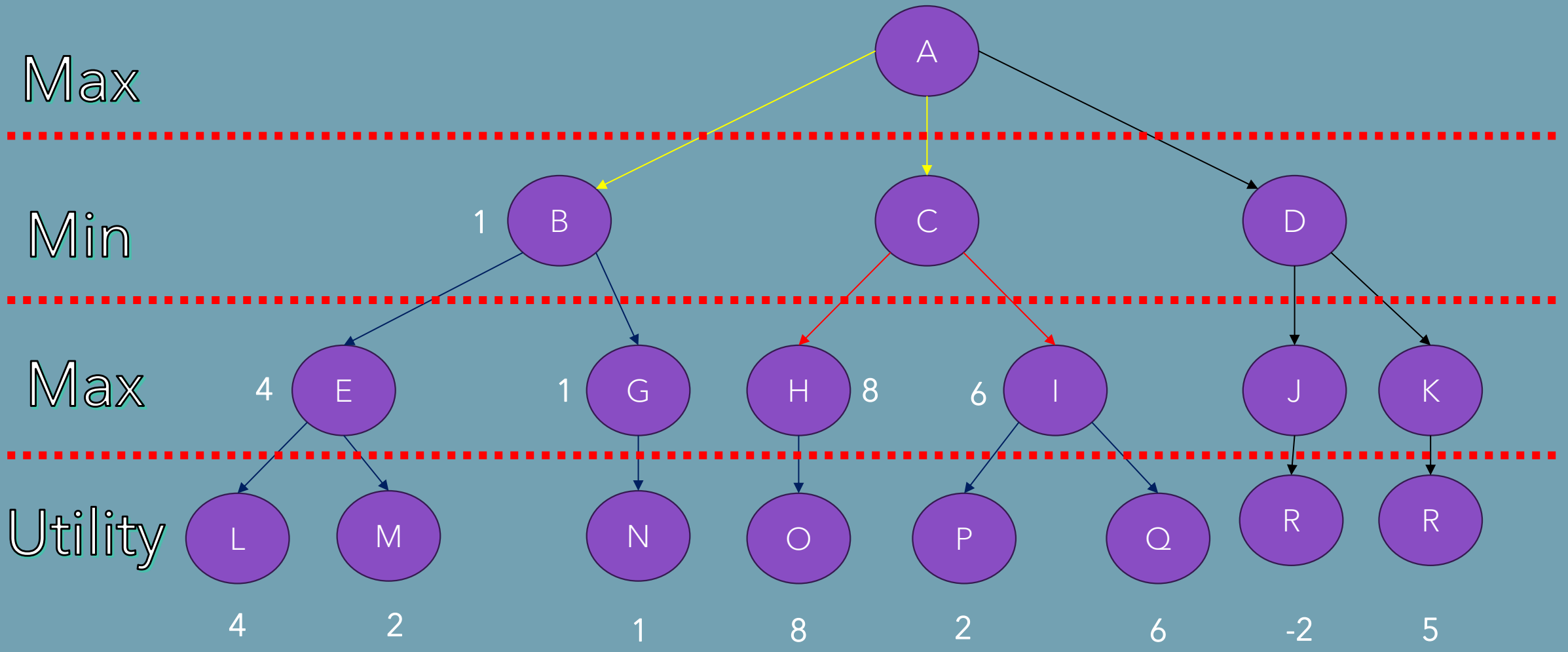
MINMAX ALGORITHM: RECURSIVE FROM ROOT



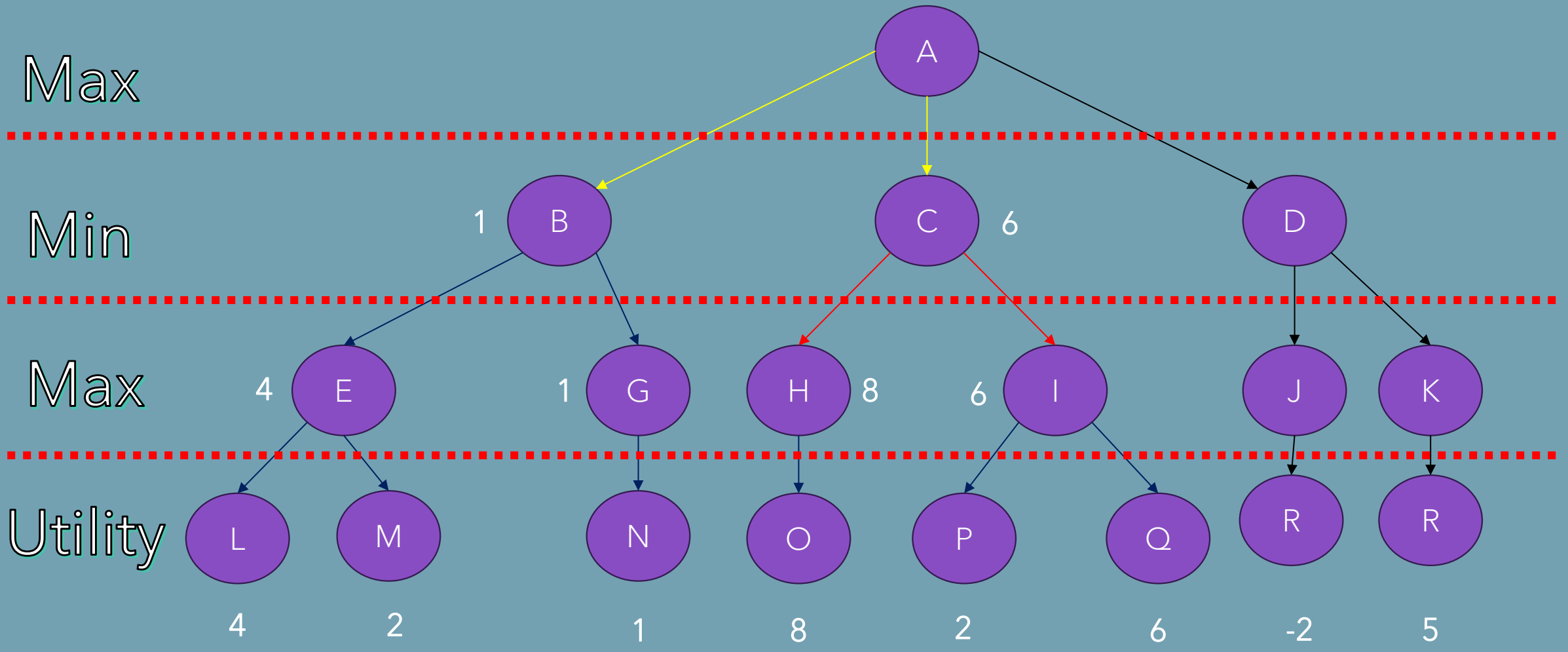
MINMAX ALGORITHM: MAXIMIZE !



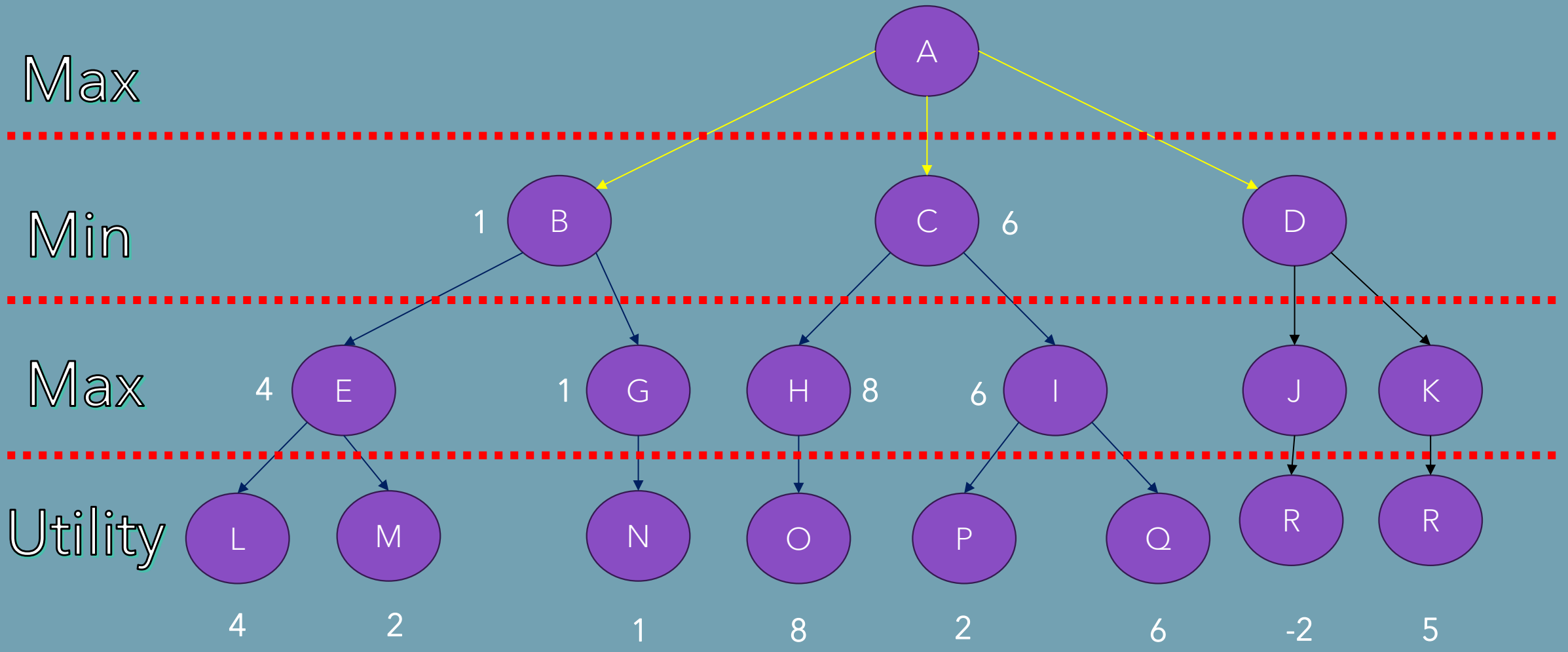
MINMAX ALGORITHM: RECURSIVE FROM ROOT



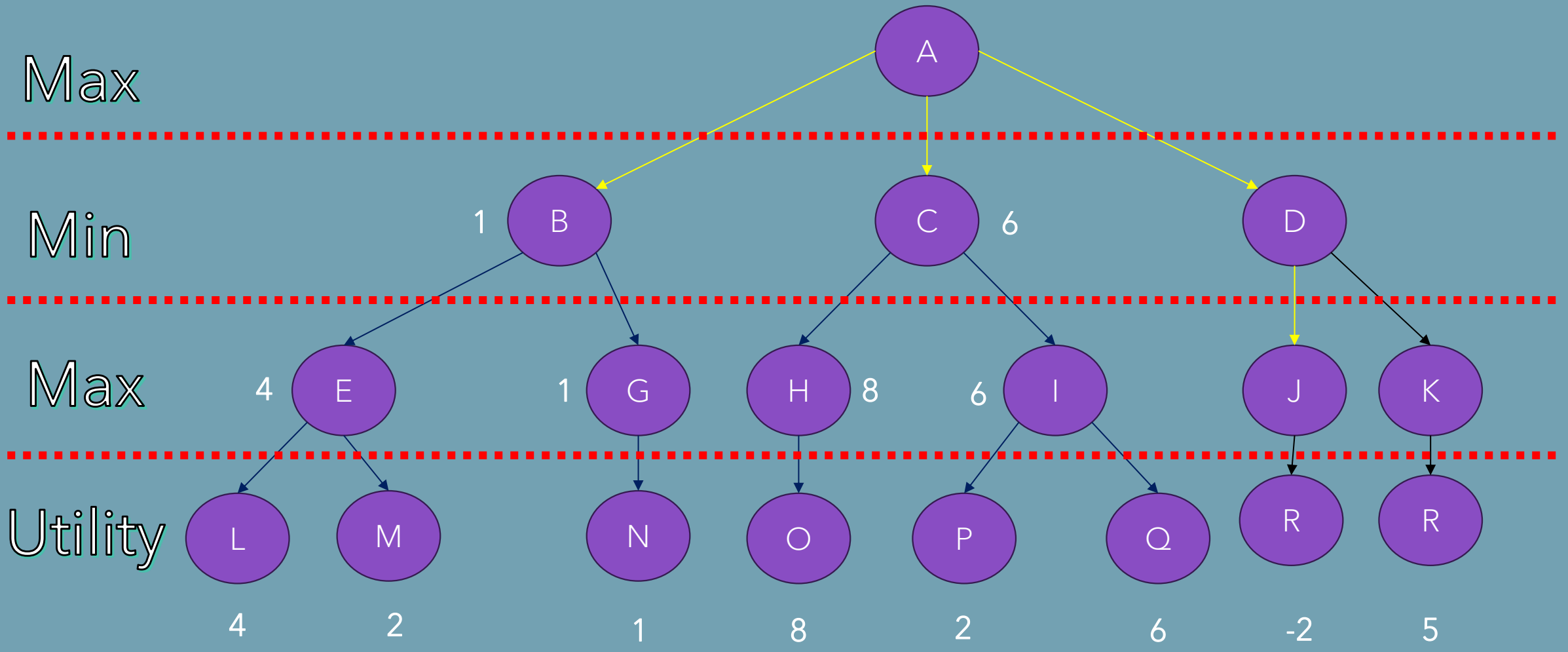
MINMAX ALGORITHM: MINIMIZE !



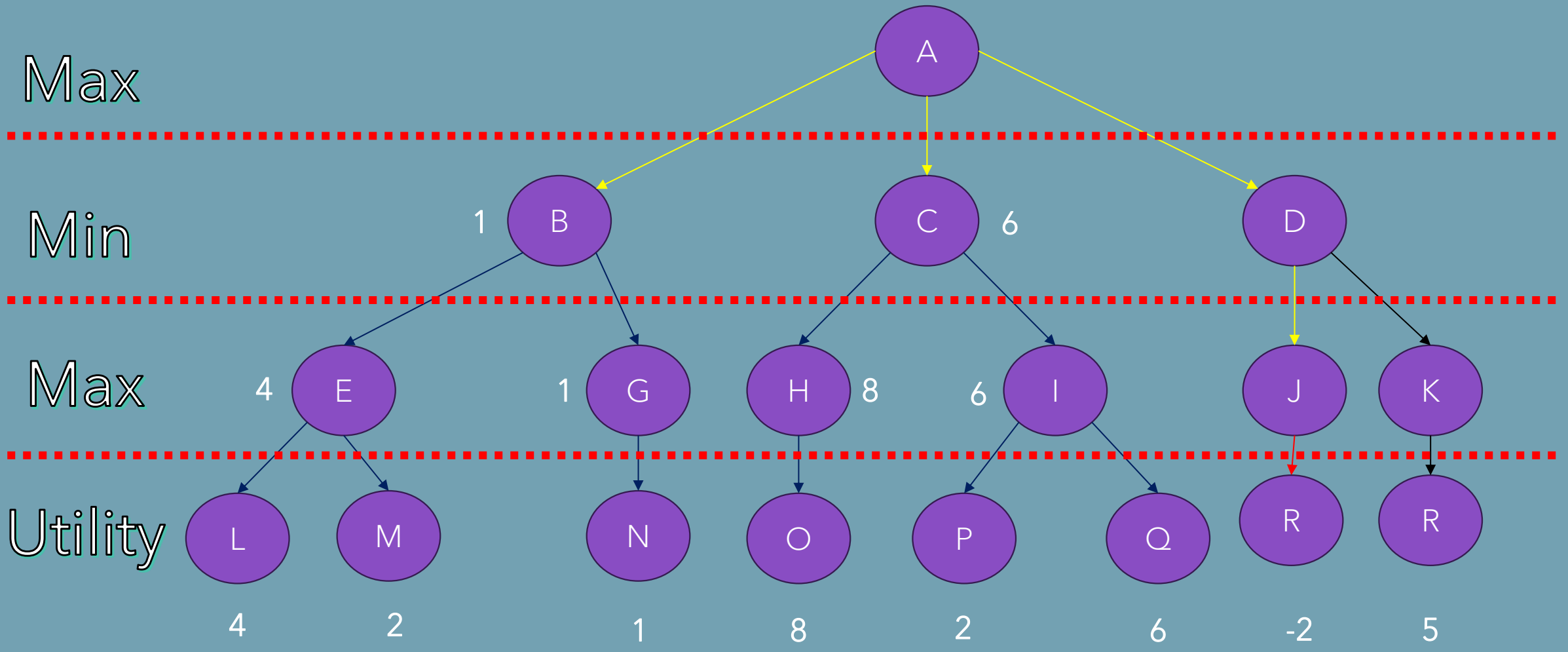
MINMAX ALGORITHM: RECURSIVE FROM ROOT



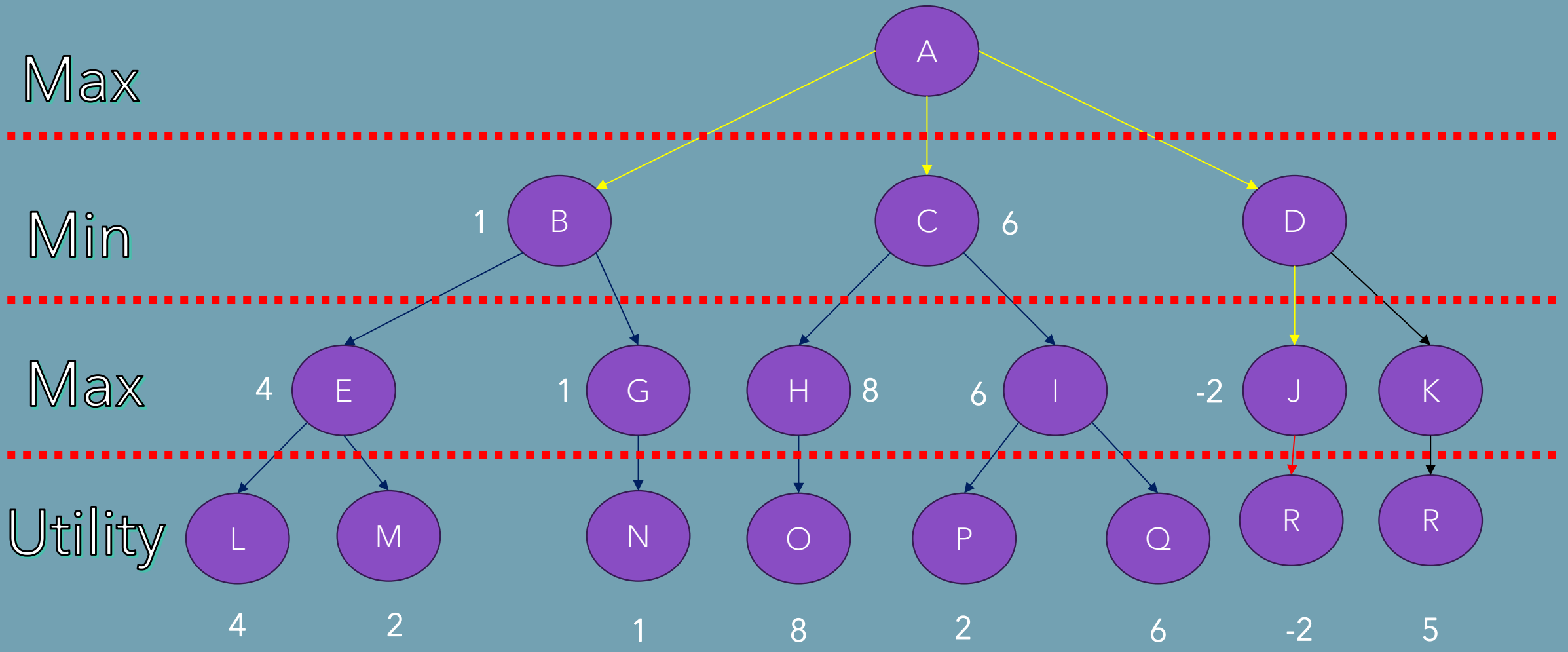
MINMAX ALGORITHM: RECURSIVE FROM ROOT



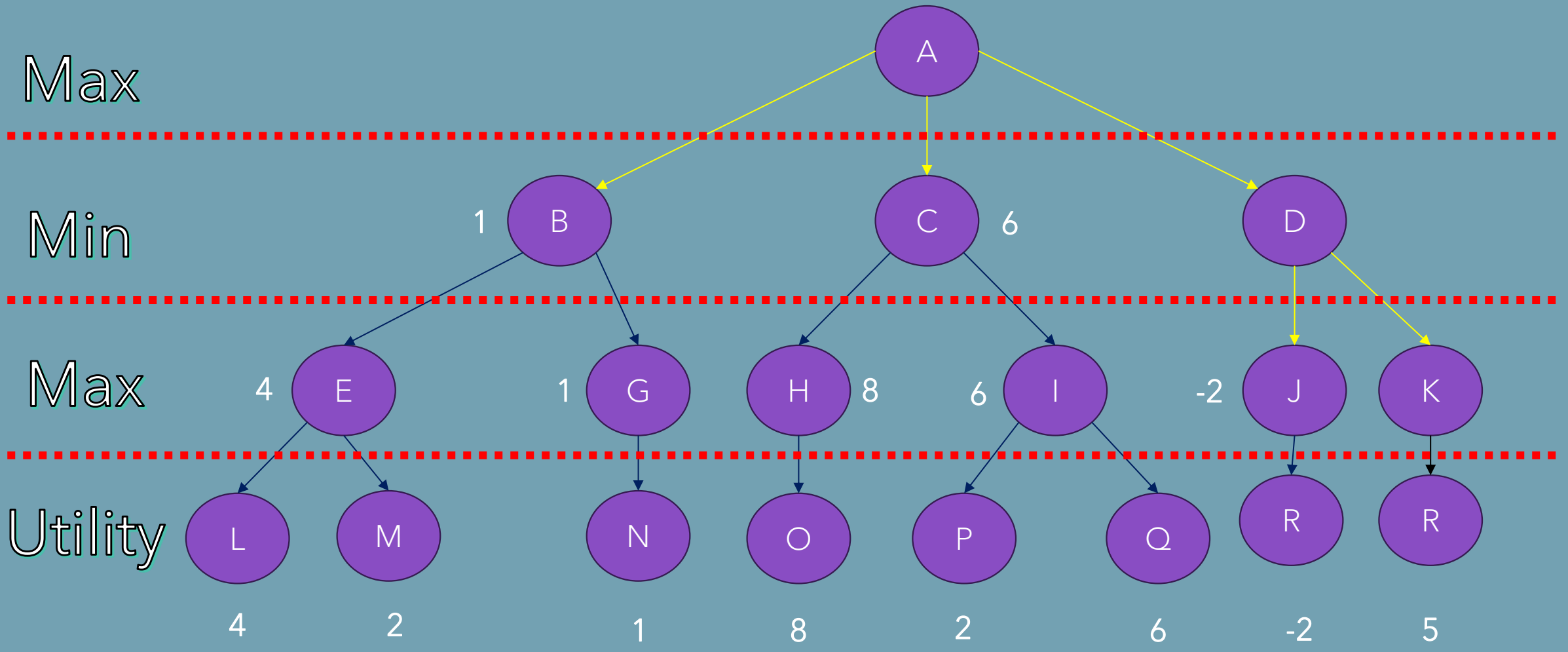
MINMAX ALGORITHM: RECURSIVE FROM ROOT



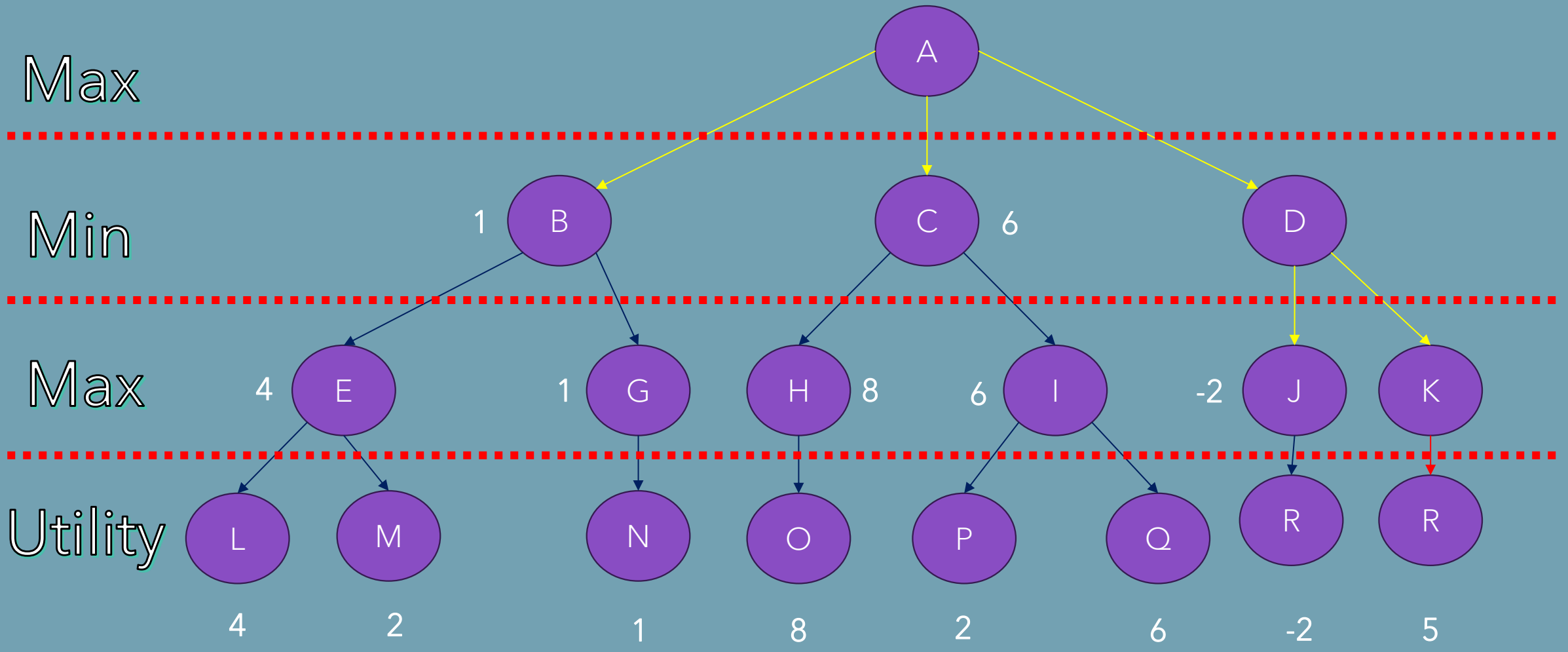
MINMAX ALGORITHM: MAXIMIZE !



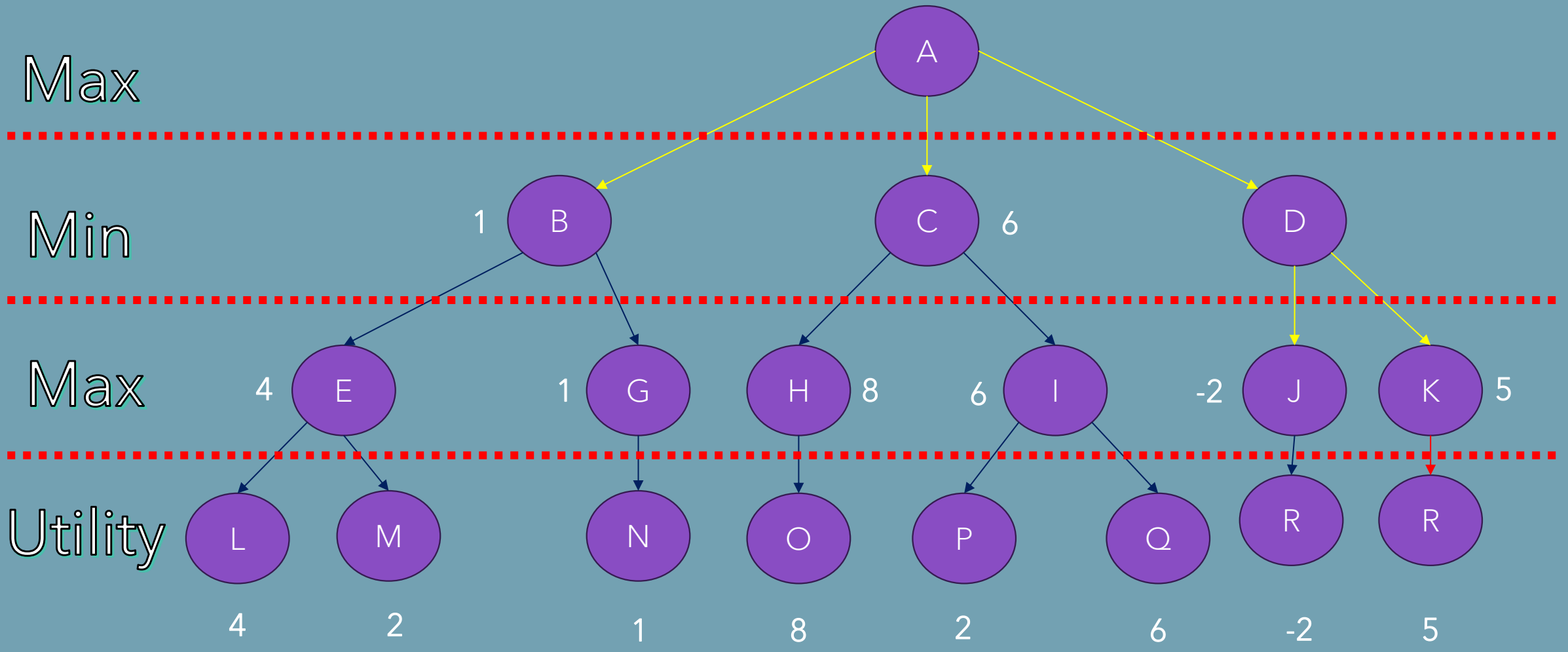
MINMAX ALGORITHM: RECURSIVE FROM ROOT



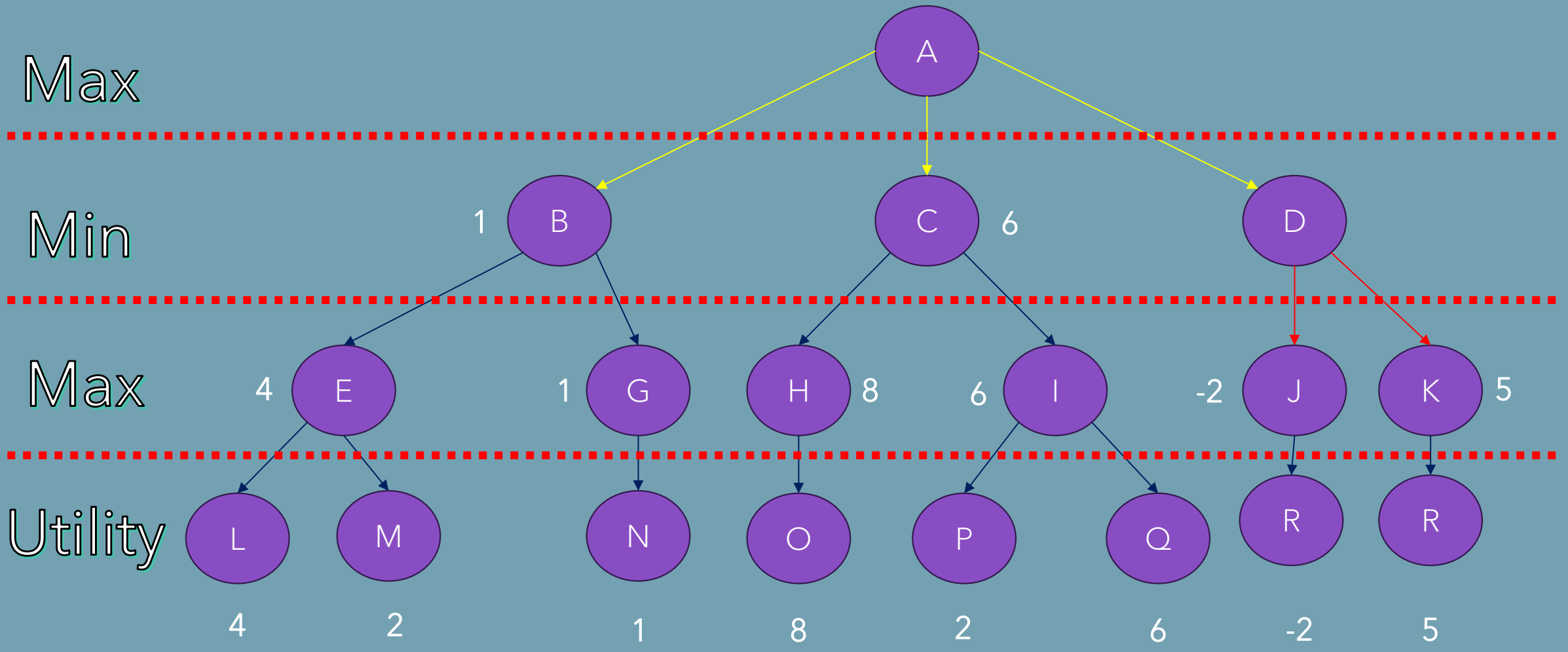
MINMAX ALGORITHM: RECURSIVE FROM ROOT



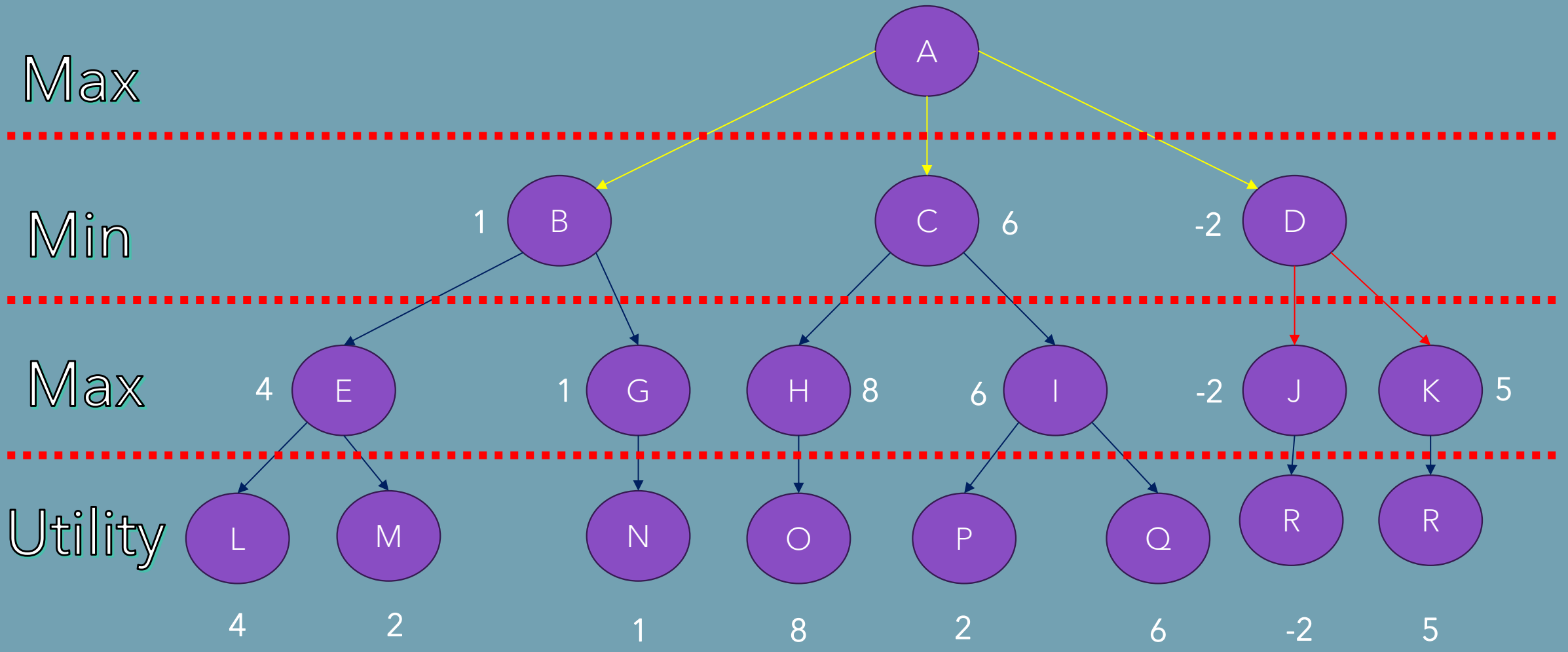
MINMAX ALGORITHM: MAXIMIZE !



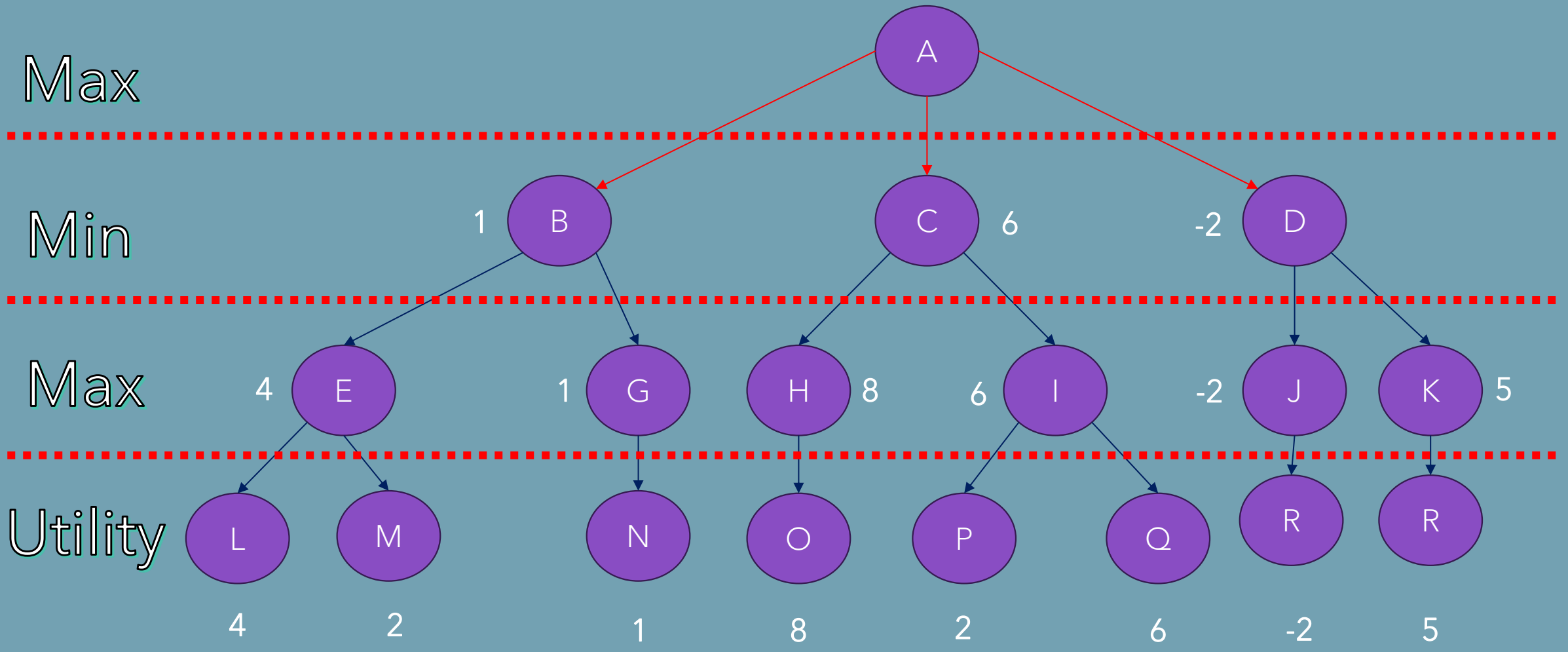
MINMAX ALGORITHM: RECURSIVE FROM ROOT



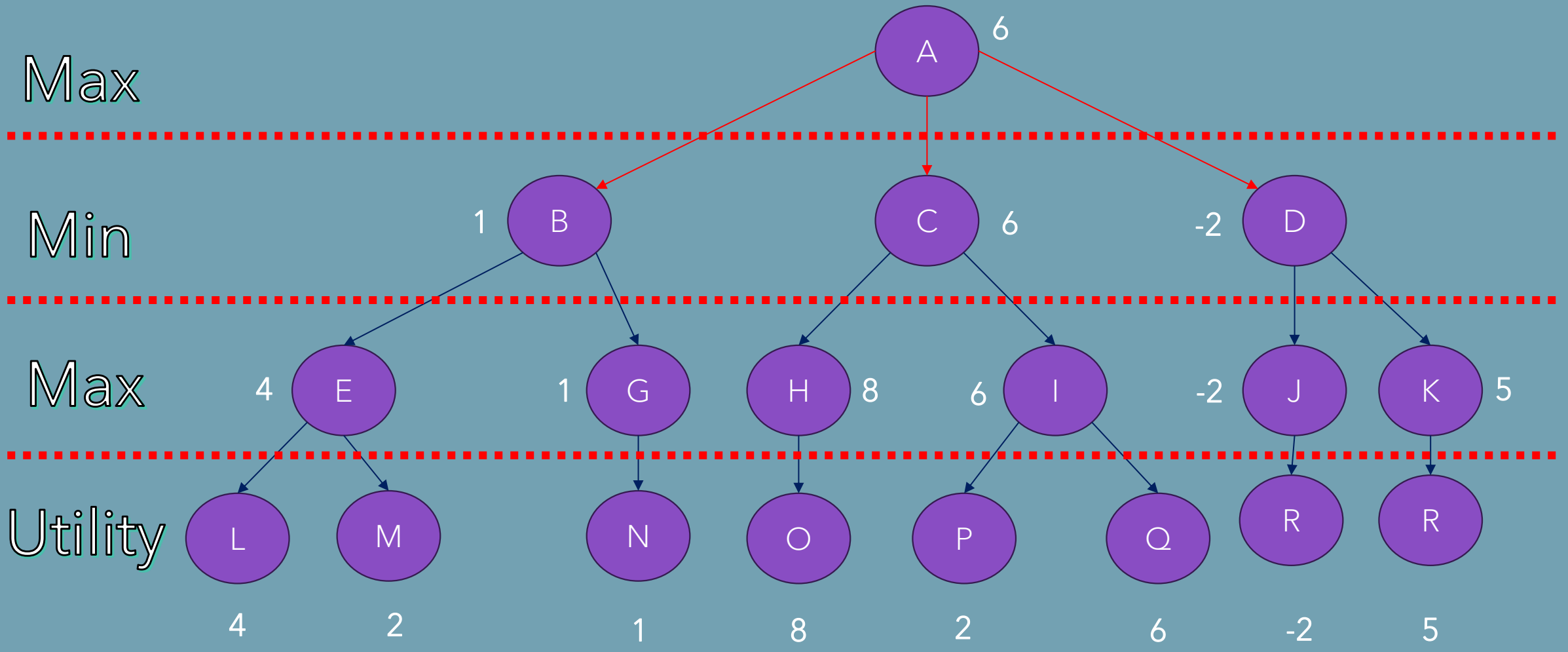
MINMAX ALGORITHM: MINIMIZE !



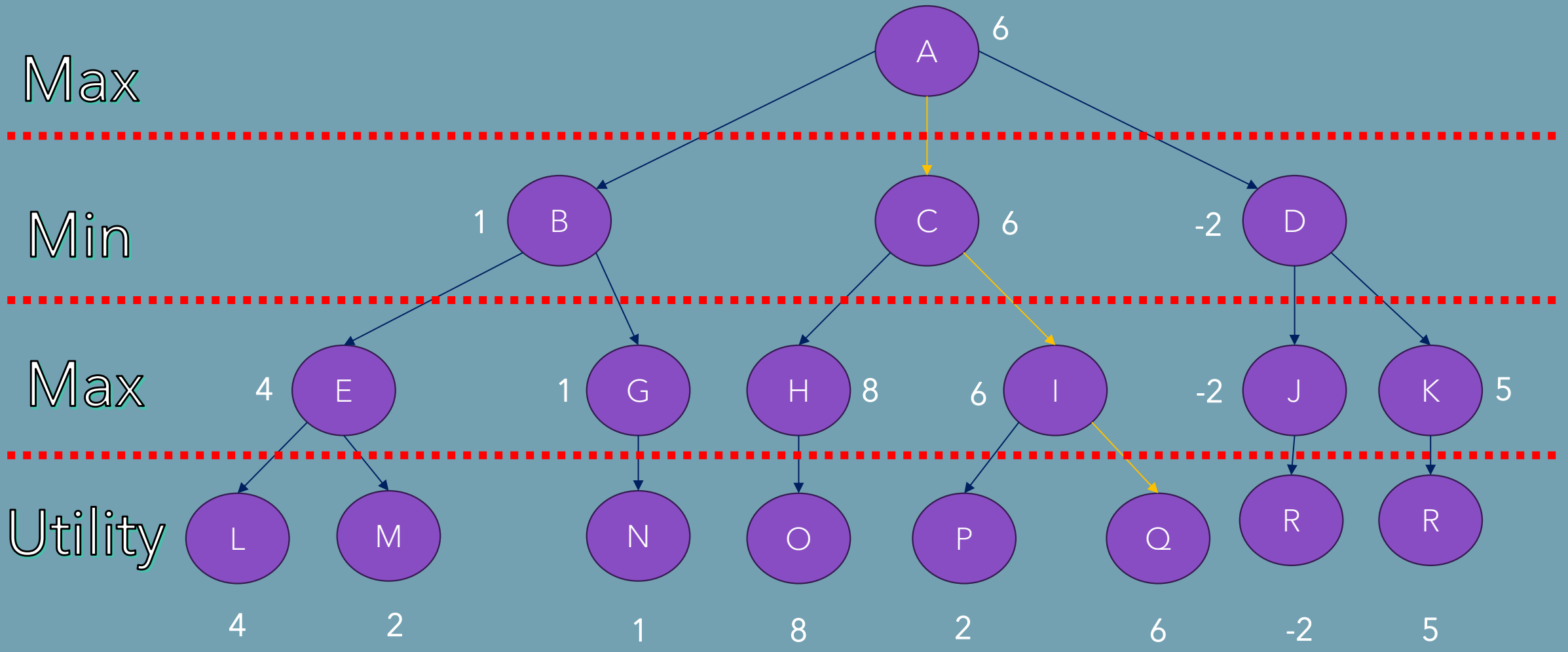
MINMAX ALGORITHM: RECURSIVE FROM ROOT



MINMAX ALGORITHM: MAXIMIZE !



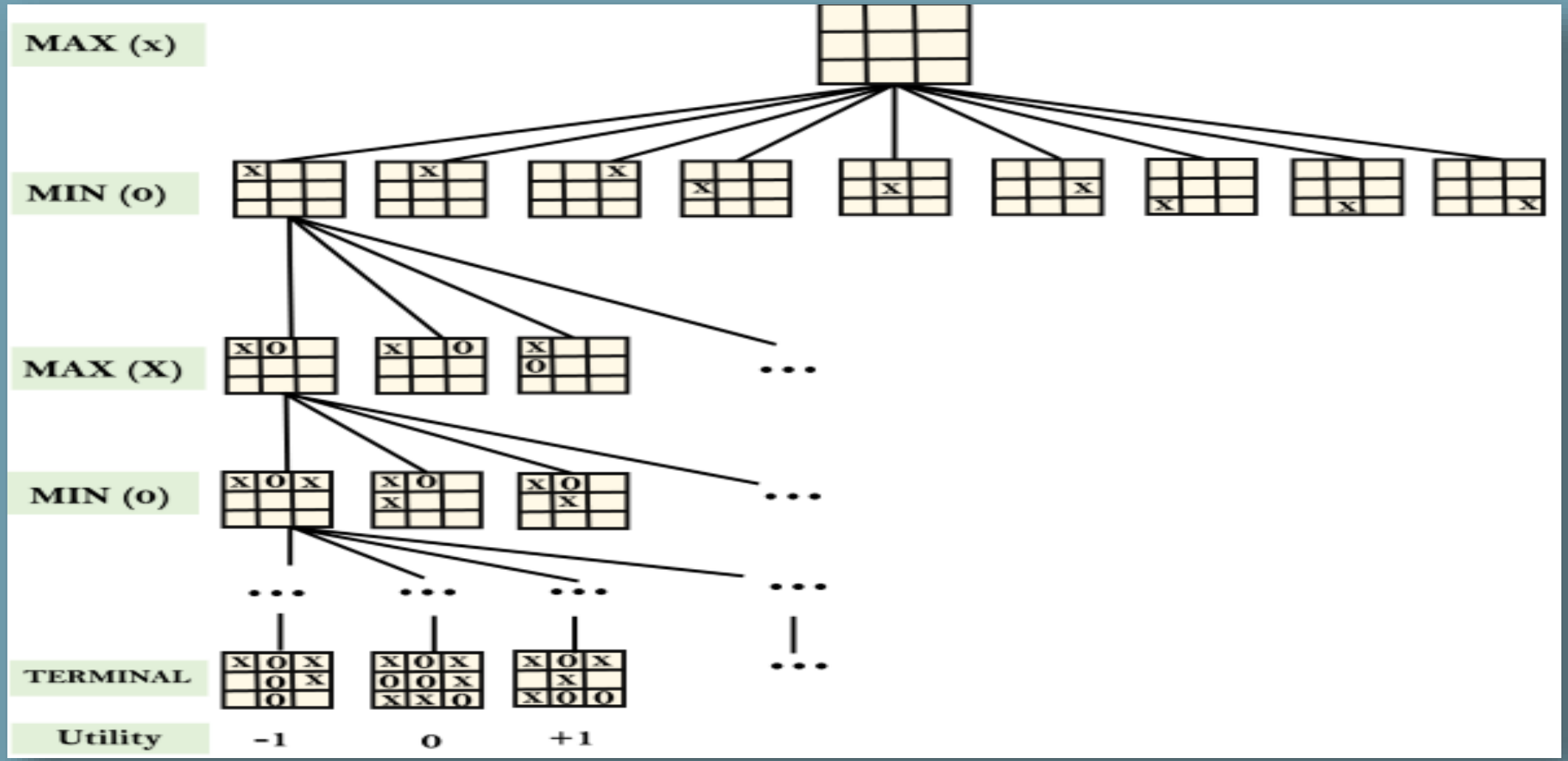
MINMAX ALGORITHM: BEST ACTION [A, C, I, Q]



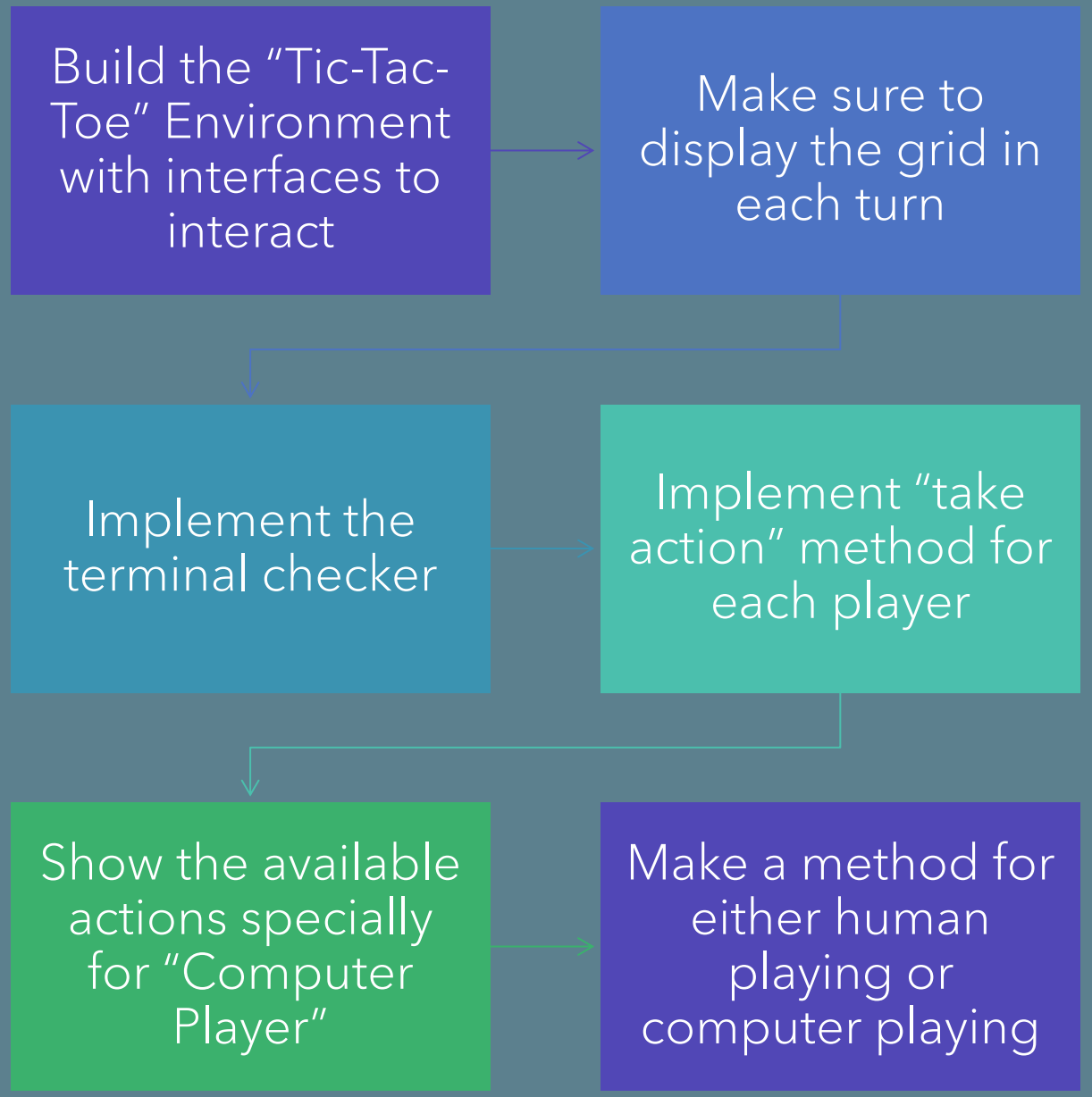
IMPLEMENTATION

Tic – Tac - Toe

TIC-TAC-TOE



IMPLEMENTATION REQUIREMENTS



TIC-TAC-TOE (CLASS IMPLEMENTATION)

```
# Design Tic-Tac-Toe environment
class Tic_Tac_Toe:
    def __init__(self):
        self.initial_grid = [[" ", " ", " "],[" ", " ", " "],[" ", " ", " "]]
```

TIC-TAC-TOE (DISPLAY GRID METHOD)

```
def display_grid(self, state):
    print()
    for row in range(3):
        for col in range(3):
            print(' ', state[row][col], ' ', sep='', end='')
            if col < 2:
                print('|', end='')
        print()
        if row < 2:
            print('---+---+---', end='')
            print()
    print()
```

TIC-TAC-TOE (CURRENT PLAYER METHOD)

Who should play now ?

```
def current_player(self, state):
    # Count the number of Xs and Os
    count_X = 0
    count_O = 0
    for row in range(3):
        for col in range(3):
            symbol = state[row][col]
            if symbol == 'X':
                count_X += 1
            elif symbol == 'O':
                count_O += 1
    # If they are the same, it's player X's turn
    if count_X == count_O:
        return 'X'
    # Otherwise, it's player O's turn
    return 'O'
```

TIC-TAC-TOE (TAKE ACTION METHOD)

```
def take_action(self, current_state, action):
    new_state = [[" ", " ", " "],[" ", " ", " "],[" ", " ", " "]] # Define a new_state to not make a conflict
    # copy the current state into a new state
    for row in range(3):
        for col in range(3):
            new_state[row][col] = current_state[row][col]

    player, row, col = action
    new_state[row][col] = player
    return new_state
```

TIC-TAC-TOE (CHECK TERMINAL METHOD)

```python
def check_terminal(self, current_state):
    terminal = False
    full = False
    player = None
    # if X's or O's in a row, it is over
    for row in range(3):
        if current_state[row][0] == current_state[row][1] == current_state[row][2] and current_state[row][0] != " ":
            terminal = True
            player = current_state[row][0]
            break

    # if X's or O's in a column, it is over
    for col in range(3):
        if current_state[0][col] == current_state[1][col] == current_state[2][col] and current_state[0][col] != " ":
            terminal = True
            player = current_state[0][col]
            break

    # if X's or O's in a diagonal, it is over
    if current_state[0][0] == current_state[1][1] == current_state[2][2] and current_state[0][0] != " ":
        terminal = True
        player = current_state[0][0]

    if current_state[0][2] == current_state[1][1] == current_state[2][0] and current_state[0][2] != " ":
        terminal = True
        player = current_state[0][2]

    # if all cells are not empty, then it is DRAW
    empty_count = 0
    for row in range(3):
        for col in range(3):
            if current_state[row][col] == " ":
                empty_count += 1

    if empty_count == 0:
        full = True

    # check the winner or if it is draw
    if terminal:
        if player == "X":
            return 1
        elif player == "O":
            return -1
    elif full:
        return 0
    else:
        return "Not terminal"
```

TIC-TAC-TOE (AVAILABLE ACTIONS METHOD)

```
def available_actions(self, current_state):
    actions = []
    player = self.current_player(current_state)
    for row in range(3):
        for col in range(3):
            if current_state[row][col] == " ":
                actions.append((player, row, col))

    return actions
```

TIC-TAC-TOE (HUMAN PLAY METHOD)

```python
def human_play(self, current_state):
    self.display_grid(current_state)
    player = self.current_player(current_state)
    print(f"Your turn, you are playing with {player}")
    row, column = list(map(int, input("Choose your action, row column respectively: ").split()))
    action = (player, row, column)
    new_state = self.take_action(current_state, action)
    self.display_grid(new_state)
    return new_state
```

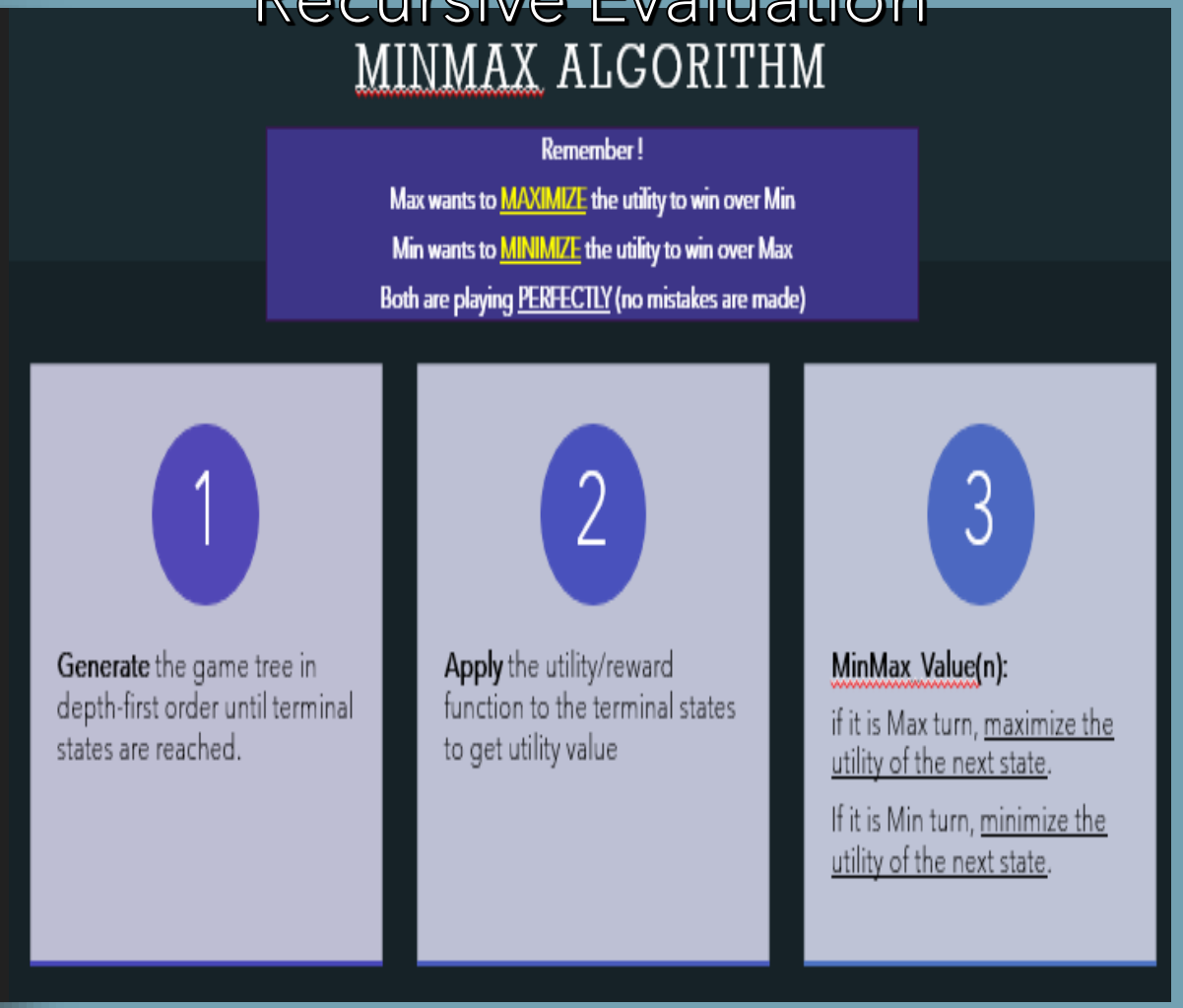
TIC-TAC-TOE (MINMAX METHOD)

```python
def MinMax(self, current_state):
    if self.check_terminal(current_state) != "Not terminal":
        score = self.check_terminal(current_state)
        return score


    utility_values = []
    for action in self.available_actions(current_state):
        next_state = self.take_action(current_state, action)
        new_utility_value = self.MinMax(next_state)
        utility_values.append(new_utility_value)

    if self.current_player(current_state) == "X":
        return max(utility_values)
    elif self.current_player(current_state) == "O":
        return min(utility_values)
```

Recursive Evaluation

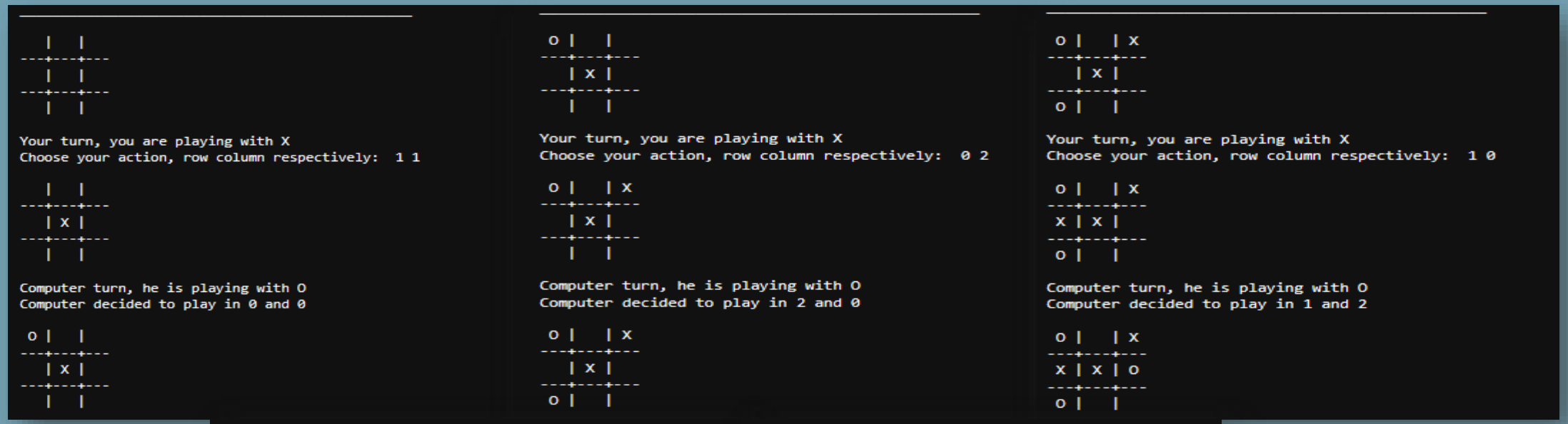


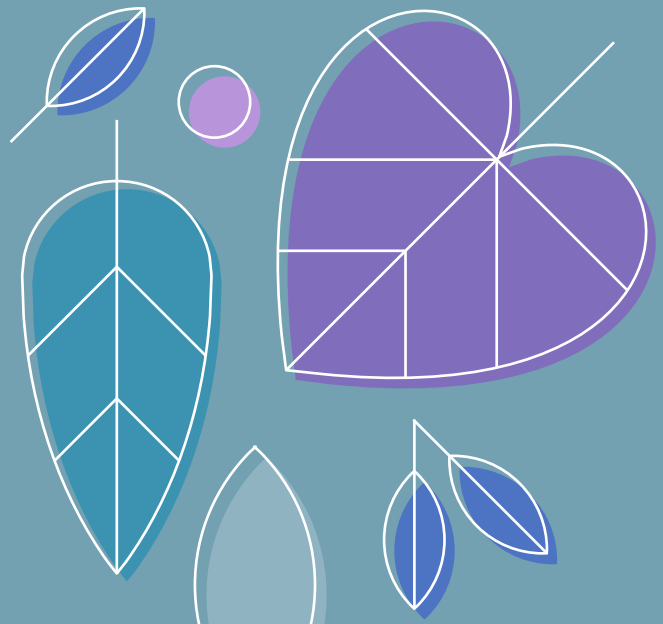
TESTING

```
my_game = Tic_Tac_Toe()
my_grid = my_game.initial_grid.copy()
while my_game.check_terminal(my_grid) == "Not terminal":
    print("_____")
    my_grid = my_game.human_play(my_grid)
    if my_game.check_terminal(my_grid) != "Not terminal":
        if my_game.check_terminal(my_grid) == 1:
            print("You won !")
            break
        elif my_game.check_terminal(my_grid) == -1:
            print("You lost !")
            break
        else:
            print("Draw !")
            break

    my_grid = my_game.computer_play(my_grid)
    if my_game.check_terminal(my_grid) != "Not terminal":
        if my_game.check_terminal(my_grid) == 1:
            print("You won !")
            break
        elif my_game.check_terminal(my_grid) == -1:
            print("You lost !")
            break
        else:
            print("Draw !")
            break
```

TESTING


Your turn, you are playing with X
Choose your action, row column respectively: 1 1
Computer turn, he is playing with O
Computer decided to play in 0 and 0
Your turn, you are playing with X
Choose your action, row column respectively: 0 2
Computer turn, he is playing with O
Computer decided to play in 2 and 0
Your turn, you are playing with X
Choose your action, row column respectively: 1 0
Computer turn, he is playing with O
Computer decided to play in 1 and 2
Your turn, you are playing with X
Choose your action, row column respectively: 2 2
Computer turn, he is playing with O
Computer decided to play in 0 and 1
Your turn, you are playing with X
Choose your action, row column respectively: 2 1
Draw !



NEXT LAB: GENETIC ALGORITHM

Thank you