In [1]:
```
#3
def backtrack(assignment, variables, constraints):
    if len(assignment) == len(variables):
        return assignment

    unassigned_var = next(var for var in variables if var not in assignment)

    for value in range(1, 4):
        if is_consistent(unassigned_var, value, assignment, constraints):
            assignment[unassigned_var] = value
            result = backtrack(assignment, variables, constraints)
            if result is not None:
                return result
            assignment.pop(unassigned_var)
    return None
def is_consistent(variable, value, assignment, constraints):
    return all((constraint[0] != variable or (constraint[1] not in assignment
                                     or assignment[constraint[1]] != value))
               and (constraint[1] != variable or (constraint[0] not in assignment
                                      or assignment[constraint[0]] != value))
               for constraint in constraints)
variables = ['A', 'B', 'C']
constraints = [('A', 'B'), ('B', 'C')]
solution = backtrack({}, variables, constraints)
if solution is not None:
    print("Solution found:")
    for var, value in solution.items():
        print(f"{var}: {value}")
else:
    print("No solution found.")
```

```
Solution found:
A: 1
B: 2
C: 1
```

In [5]:
```
#4
MAX, MIN = 1000, -1000
def minimax(depth, nodeIndex, maximizingPlayer,
            values, alpha, beta):
    if depth == 3:
        return values[nodeIndex]
    if maximizingPlayer:
        best = MIN
        for i in range(0, 2):
            val = minimax(depth + 1, nodeIndex * 2 + i,
                          False, values, alpha, beta)
            best = max(best, val)
            alpha = max(alpha, best)
            if beta <= alpha:
                break
        return best

    else:
        best = MAX
        for i in range(0, 2):
            val = minimax(depth + 1, nodeIndex * 2 + i,
                            True, values, alpha, beta)
            best = min(best, val)
            beta = min(beta, best)
            if beta <= alpha:
                break
```

```
        return best
if __name__ == "__main__":

    values = [3, 5, 6, 9, 1, 2, 0, -1]
    print("The optimal value is :", minimax(0, 0, True, values, MIN, MAX))
```

```
The optimal value is : 5
```

In [ ]: