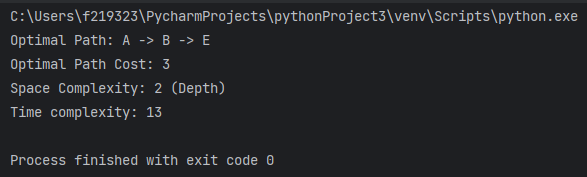
# TASK 1:

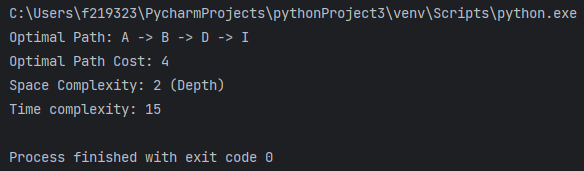
graph = {  
 'A': ['B', 'C', 'D'],  
 'B': ['E', 'F', 'G'],  
 'C': ['H', 'I', 'J'],  
 'D': ['K', 'L', 'M'],  
 'E': [],  
 'F': [],  
 'G': [],  
 'H': [],  
 'I': [],  
 'J': [],  
 'K': [],  
 'L': [],  
 'M': []  
}  
  
cost = {  
 'A': 3,  
 'B': 3,  
 'C': 2,  
 'D': 2,  
 'E': 3,  
 'F': 12,  
 'G': 8,  
 'H': 2,  
 'I': 4,  
 'J': 6,  
 'K': 14,  
 'L': 5,  
 'M': 2,  
}  
  
  
class GraphMinimax:  
 def \_\_init\_\_(self, graph, cost):  
 self.graph = graph  
 self.cost = cost  
 self.nodes\_explored = 0  
  
 def min\_max(self, node, is\_maximizing):  
 self.nodes\_explored += 1  
  
 if self.graph[node] == []:  
 return self.cost[node], [node]  
  
 if is\_maximizing:  
 max\_eval = float('-inf')  
 best\_path = []  
 for child in self.graph[node]:  
 eval\_child, path\_child = self.min\_max(child, False)  
 if eval\_child > max\_eval:  
 max\_eval = eval\_child  
 best\_path = [node] + path\_child  
 return max\_eval, best\_path  
  
 else:  
 min\_eval = float('inf')  
 best\_path = []  
 for child in self.graph[node]:  
 eval\_child, path\_child = self.min\_max(child, True)  
 if eval\_child < min\_eval:  
 min\_eval = eval\_child  
 best\_path = [node] + path\_child  
 return min\_eval, best\_path  
  
 def find\_optimal\_path(self, start\_node):  
 optimal\_path\_cost, optimal\_path = self.min\_max(start\_node, True)  
 return optimal\_path\_cost, optimal\_path  
  
  
  
  
minimax = GraphMinimax(graph, cost)  
optimal\_path\_cost, optimal\_path = minimax.find\_optimal\_path('A')  
print("Optimal Path:", ' -> '.join(optimal\_path))  
print("Optimal Path Cost:", optimal\_path\_cost)  
print("Space Complexity: 2 (Depth)")  
print("Time complexity:", minimax.nodes\_explored)

OUTPUT:  


# TASK 2:

graph = {  
 'A': ['B', 'C'],  
 'B': ['D', 'E'],  
 'C': ['F', 'G'],  
 'D': ['H', 'I'],  
 'E': ['J', 'K'],  
 'F': ['L', 'M'],  
 'G': ['N', 'O'],  
 'H': [],  
 'I': [],  
 'J': [],  
 'K': [],  
 'L': [],  
 'M': [],  
 'N': [],  
 'O': []  
}  
  
cost = {  
 'A': 4,  
 'B': 4,  
 'C': -3,  
 'D': 4,  
 'E': 6,  
 'F': -3,  
 'G': 7,  
 'H': -1,  
 'I': 4,  
 'J': 2,  
 'K': 6,  
 'L': -3,  
 'M': -5,  
 'N': 0,  
 'O': 7,  
}  
  
  
class GraphMinimax:  
 def \_\_init\_\_(self, graph, cost):  
 self.graph = graph  
 self.cost = cost  
 self.nodes\_explored = 0  
  
 def min\_max(self, node, is\_maximizing):  
 self.nodes\_explored += 1  
  
 if self.graph[node] == []:  
 return self.cost[node], [node]  
  
 if is\_maximizing:  
 max\_eval = float('-inf')  
 best\_path = []  
 for child in self.graph[node]:  
 eval\_child, path\_child = self.min\_max(child, False)  
 if eval\_child > max\_eval:  
 max\_eval = eval\_child  
 best\_path = [node] + path\_child  
 return max\_eval, best\_path  
  
 else:  
 min\_eval = float('inf')  
 best\_path = []  
 for child in self.graph[node]:  
 eval\_child, path\_child = self.min\_max(child, True)  
 if eval\_child < min\_eval:  
 min\_eval = eval\_child  
 best\_path = [node] + path\_child  
 return min\_eval, best\_path  
  
 def find\_optimal\_path(self, start\_node):  
 optimal\_path\_cost, optimal\_path = self.min\_max(start\_node, True)  
 return optimal\_path\_cost, optimal\_path  
  
  
  
  
minimax = GraphMinimax(graph, cost)  
optimal\_path\_cost, optimal\_path = minimax.find\_optimal\_path('A')  
print("Optimal Path:", ' -> '.join(optimal\_path))  
print("Optimal Path Cost:", optimal\_path\_cost)  
print("Space Complexity: 2 (Depth)")  
print("Time complexity:", minimax.nodes\_explored)

# OUTPUT:



# TASK 3:

graph = {  
 'A': ['B', 'C', 'D'],  
 'B': ['E', 'F', 'G'],  
 'C': ['H', 'I', 'J'],  
 'D': ['K', 'L', 'M'],  
 'E': [],  
 'F': [],  
 'G': [],  
 'H': [],  
 'I': [],  
 'J': [],  
 'K': [],  
 'L': [],  
 'M': []  
}  
  
cost = {  
 'A': 3,  
 'B': 3,  
 'C': 2,  
 'D': 2,  
 'E': 3,  
 'F': 12,  
 'G': 8,  
 'H': 2,  
 'I': 4,  
 'J': 6,  
 'K': 14,  
 'L': 5,  
 'M': 2,  
}  
  
  
class MinMaxAlphaBeta:  
 def \_\_init\_\_(self, graph, cost):  
 self.graph = graph  
 self.cost = cost  
 self.nodes\_explored = 0  
  
 def minimax\_alpha\_beta(self, node, is\_maximizing, alpha, beta):  
 self.nodes\_explored += 1  
  
 if not self.graph[node]:  
 return self.cost[node], [node]  
  
 if is\_maximizing:  
 max\_eval = float('-inf')  
 best\_path = []  
 for child in self.graph[node]:  
 eval\_child, path\_child = self.minimax\_alpha\_beta(child, False, alpha, beta)  
 if eval\_child > max\_eval:  
 max\_eval = eval\_child  
 best\_path = [node] + path\_child  
 alpha = max(alpha, max\_eval)  
 if alpha >= beta:  
 break  
 return max\_eval, best\_path  
  
 else:  
 min\_eval = float('inf')  
 best\_path = []  
 for child in self.graph[node]:  
 eval\_child, path\_child = self.minimax\_alpha\_beta(child, True, alpha, beta)  
 if eval\_child < min\_eval:  
 min\_eval = eval\_child  
 best\_path = [node] + path\_child  
 beta = min(beta, min\_eval)  
 if beta <= alpha:  
 break  
 return min\_eval, best\_path  
  
 def find\_optimal\_path(self, start\_node):  
 optimal\_path\_cost, optimal\_path = self.minimax\_alpha\_beta(start\_node, True, float('-inf'), float('inf'))  
 return optimal\_path\_cost, optimal\_path  
  
  
minimax\_alpha\_beta = MinMaxAlphaBeta(graph, cost)  
optimal\_path\_cost, optimal\_path = minimax\_alpha\_beta.find\_optimal\_path('A')  
print("Optimal Path:", ' -> '.join(optimal\_path))  
print("Optimal Path Cost:", optimal\_path\_cost)  
print("Space Complexity: 2 (Depth)")  
print("Nodes Explored:", minimax\_alpha\_beta.nodes\_explored)

# OUTPUT:

