AI Practical Solutions (Complete) - Slips in sequence (3,8,16 excluded)

Slip 1

Q1. Implement Dijkstra's Algorithm to find the shortest path from a single source for the given graph G. [10 marks]

```
import heapq

def dijkstra(graph, start):
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    pq = [(0, start)]
    parent = {node: None for node in graph}

    while pq:
        dist, node = heapq.heappop(pq)
        if dist > distances[node]:
            continue
        for neighbor, weight in graph[node]:
            new_dist = dist + weight
            if new_dist < distances[neighbor]:
                distances[neighbor] = new_dist
                parent[neighbor] = node
                heapq.heappush(pq, (new_dist, neighbor))
    return distances, parent

# Example usage:
if __name__ == '__main__':
    graph = {
        'A': [('B',4), ('C',1)],
        'B': [('D',1)],
        'C': [('B',2), ('D',5)],
        'D': []
    }
    dist, parent = dijkstra(graph, 'A')
    print('Distances:', dist)
    print('Parents:', parent)
```

Q2. Write a Python program to implement Depth First Search algorithm. [Initial node=1, Goal node=8] [20 marks]

```
def dfs(graph, start, goal, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    if start == goal:
        return [start]
    for neighbor in graph.get(start, []):
        if neighbor not in visited:
            path = dfs(graph, neighbor, goal, visited)
            if path:
                return [start] + path
    return None

# Example:
if __name__ == '__main__':
    graph = {1:[2,3],2:[4],3:[5,6],4:[7],5:[],6:[8],7:[],8:[]}
    print(dfs(graph,1,8))
```

OR

Q2 (OR). Monkey Banana Problem using Python. [20 marks]

```
from collections import deque

def monkey_banana():
    # State: (monkey_pos, box_pos, monkey_on_box, has_banana)
    start = ("Door", "Window", False, False)
    goal = ("UnderBanana", "UnderBanana", True, True)

    def actions(state):
        m, b, on_box, has = state
```

```
        res = []
        # walk
        for loc in ["Door","Window","UnderBanana"]:
            if loc != m:
                res.append((loc, b, on_box, has))
        # push box (only if monkey and box at same pos and not on box)
        if m == b and not on_box:
            for loc in ["Door","Window","UnderBanana"]:
                if loc != b:
                    res.append((m, loc, on_box, has))
        # climb
        if m == b and not on_box:
            res.append((m, b, True, has))
        # grab banana
        if m == "UnderBanana" and on_box:
            res.append((m, b, on_box, True))
        return res

    q = deque([(start, [])])
    visited = set([start])
    while q:
        state, path = q.popleft()
        if state == goal:
            return path + [state]
        for nxt in actions(state):
            if nxt not in visited:
                visited.add(nxt)
                q.append((nxt, path + [state]))
    return None

if __name__ == '__main__':
    sol = monkey_banana()
    for s in sol:
        print(s)
```

Slip 2

Q1. Write a Program to Implement Depth First Search using Python. [10 marks]

```
def dfs(graph, start, goal, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    if start == goal:
        return [start]
    for neighbor in graph.get(start, []):
        if neighbor not in visited:
            path = dfs(graph, neighbor, goal, visited)
            if path:
                return [start] + path
    return None

# Example:
if __name__ == '__main__':
    graph = {1:[2,3],2:[4],3:[5,6],4:[7],5:[],6:[8],7:[],8:[]}
    print(dfs(graph,1,8))
```

Q2. Write a program to implement A* algorithm. [20 marks]

```
import heapq

def astar(start, goal, neighbors_fn, h_fn):
    openh = [(h_fn(start), 0, start, None)]
    came = {}
    gscore = {start:0}
    while openh:
        f, g, node, parent = heapq.heappop(openh)
        if node in came:
            continue
        came[node] = parent
        if node == goal:
            path = []
            n = node
            while n is not None:
                path.append(n)
                n = came[n]
            return list(reversed(path))
        for nbr, cost in neighbors_fn(node):
            ng = g + cost
            if ng < gscore.get(nbr, float('inf')):
                gscore[nbr] = ng
                heapq.heappush(openh, (ng + h_fn(nbr), ng, nbr, node))
    return None

# To use this template, provide neighbors_fn(node) -> list of (nbr, cost)
# and h_fn(node) -> heuristic estimate to goal.
```

OR

Q2 (OR). Tic-Tac-Toe game using Python. [20 marks]

```
import math

def winner(board):
    wins = [(0,1,2),(3,4,5),(6,7,8),(0,3,6),(1,4,7),(2,5,8),(0,4,8),(2,4,6)]
    for a,b,c in wins:
        if board[a]==board[b]==board[c] and board[a] != ' ':
            return board[a]
    return 'Draw' if ' ' not in board else None

def minimax(board, player):
    w = winner(board)
    if w == 'X': return 1
    if w == 'O': return -1
    if w == 'Draw': return 0
    if player == 'X':
        best = -math.inf
        for i in range(9):
            if board[i] == ' ':
                board[i] = 'X'
                val = minimax(board, 'O')
```

```
                board[i] = ' '
                best = max(best, val)
        return best
    else:
        best = math.inf
        for i in range(9):
            if board[i] == ' ':
                board[i] = 'O'
                val = minimax(board, 'X')
                board[i] = ' '
                best = min(best, val)
        return best

def best_move(board):
    best = -math.inf; move = -1
    for i in range(9):
        if board[i] == ' ':
            board[i] = 'X'
            val = minimax(board, 'O')
            board[i] = ' '
            if val > best:
                best = val; move = i
    return move

if __name__ == '__main__':
    board = [' ']*9
    move = best_move(board)
    board[move] = 'X'
    print(board)
```

Slip 4

Q1. Write a Python program to implement Depth First Search algorithm. [Initial node=1, Goal node=8] [10 marks]

```
def dfs(graph, start, goal, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    if start == goal:
        return [start]
    for neighbor in graph.get(start, []):
        if neighbor not in visited:
            path = dfs(graph, neighbor, goal, visited)
            if path:
                return [start] + path
    return None

# Example:
if __name__ == '__main__':
    graph = {1:[2,3],2:[4],3:[5,6],4:[7],5:[],6:[8],7:[],8:[]}
    print(dfs(graph,1,8))
```

Q2. Write a program to implement AO* algorithm. [20 marks]

```
# AO* algorithm sketch: This is a conceptual placeholder.
# A full AO* requires AND-OR graph representation, heuristics, and backtracking.
# For exam purposes, present algorithm steps and a short pseudocode.
def ao_star_sketch():
    pass
```

OR

Q2 (OR). Water-Jug problem using Python. [20 marks]

```
from collections import deque
def water_jug(a_cap, b_cap, target):
    start = (0,0); q = deque([start]); visited = {start}; parent = {start:None}
    while q:
        x,y = q.popleft()
        if x==target or y==target:
            path=[]; cur=(x,y)
            while cur:
                path.append(cur); cur=parent[cur]
            return list(reversed(path))
        moves = []
        moves.append((a_cap,y)); moves.append((x,b_cap))
        moves.append((0,y)); moves.append((x,0))
        t = min(x, b_cap-y); moves.append((x-t, y+t))
        t = min(y, a_cap-x); moves.append((x+t, y-t))
        for m in moves:
            if m not in visited:
                visited.add(m); parent[m]=(x,y); q.append(m)
    return None

if __name__ == '__main__':
    print(water_jug(4,3,2))
```

Slip 5

Q1. Implement Dijkstra's algorithm to find the shortest path from a source node to all other nodes in a weighted graph. [10 marks]

```
import heapq

def dijkstra(graph, start):
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    pq = [(0, start)]
    parent = {node: None for node in graph}

    while pq:
        dist, node = heapq.heappop(pq)
        if dist > distances[node]:
            continue
        for neighbor, weight in graph[node]:
            new_dist = dist + weight
            if new_dist < distances[neighbor]:
                distances[neighbor] = new_dist
                parent[neighbor] = node
                heapq.heappush(pq, (new_dist, neighbor))
    return distances, parent

# Example usage:
if __name__ == '__main__':
    graph = {
        'A': [('B',4), ('C',1)],
        'B': [('D',1)],
        'C': [('B',2), ('D',5)],
        'D': []
    }
    dist, parent = dijkstra(graph, 'A')
    print('Distances:', dist)
    print('Parents:', parent)
```

Q2. Implement Iterative Deepening DFS algorithm. [20 marks]

```
def dls(node, goal, depth, graph, path=None):
    if path is None: path = []
    path = path + [node]
    if node == goal:
        return path
    if depth == 0:
        return None
    for n in graph.get(node, []):
        res = dls(n, goal, depth-1, graph, path)
        if res: return res
    return None

def iddfs(start, goal, graph, maxd=20):
    for d in range(maxd+1):
        r = dls(start, goal, d, graph)
        if r: return r
    return None
```

OR

Q2 (OR). Travelling Salesman Problem using Python (heuristic). [20 marks]

```
def nearest_neighbor(dist):
    n = len(dist); visited = {0}; path=[0]; cur=0
    while len(visited) < n:
        nxt = min((dist[cur][j], j) for j in range(n) if j not in visited)[1]
        visited.add(nxt); path.append(nxt); cur = nxt
    path.append(0); return path

if __name__ == '__main__':
    print(nearest_neighbor([[0,10,15,20],[10,0,35,25],[15,35,0,30],[20,25,30,0]]))
```

Slip 6

Q1. Selection Sort (descending). [10 marks]

```
def selection_sort_desc(arr):
    a = arr[:]; n = len(a)
    for i in range(n):
        max_i = i
        for j in range(i+1,n):
            if a[j] > a[max_i]: max_i = j
        a[i], a[max_i] = a[max_i], a[i]
    return a

if __name__ == '__main__':
    print(selection_sort_desc([64,25,12,22,11]))
```

Q2. Develop an elementary Chatbot for any suitable customer interaction application. [20 marks]

```
def bot(query):
    q = query.lower()
    if 'hello' in q or 'hi' in q: return "Hello! How can I help?"
    if 'price' in q or 'cost' in q: return "Our prices are on the website."
    if 'book' in q or 'reserve' in q: return "Please provide date and time."
    return "Sorry, I don't understand. Please contact support."

if __name__ == '__main__':
    print(bot("I want to book a table"))
```

OR

Q2 (OR). Water-Jug problem using Python. [20 marks]

```
from collections import deque
def water_jug(a_cap, b_cap, target):
    start = (0,0); q = deque([start]); visited = {start}; parent = {start:None}
    while q:
        x,y = q.popleft()
        if x==target or y==target:
            path=[]; cur=(x,y)
            while cur:
                path.append(cur); cur=parent[cur]
            return list(reversed(path))
        moves = []
        moves.append((a_cap,y)); moves.append((x,b_cap))
        moves.append((0,y)); moves.append((x,0))
        t = min(x, b_cap-y); moves.append((x-t, y+t))
        t = min(y, a_cap-x); moves.append((x+t, y-t))
        for m in moves:
            if m not in visited:
                visited.add(m); parent[m]=(x,y); q.append(m)
    return None

if __name__ == '__main__':
    print(water_jug(4,3,2))
```

Slip 7

Q1. Breadth First Search using Python. [10 marks]

```
from collections import deque
def bfs(graph, start, goal):
    q = deque([[start]]); visited = {start}
    while q:
        path = q.popleft(); node = path[-1]
        if node == goal: return path
        for n in graph.get(node, []):
            if n not in visited:
                visited.add(n); q.append(path + [n])
    return None
```

Q2. Iterative Deepening DFS algorithm. [20 marks]

```
def dls(node, goal, depth, graph, path=None):
    if path is None: path = []
    path = path + [node]
    if node == goal:
        return path
    if depth == 0:
        return None
    for n in graph.get(node, []):
        res = dls(n, goal, depth-1, graph, path)
        if res: return res
    return None

def iddfs(start, goal, graph, maxd=20):
    for d in range(maxd+1):
        r = dls(start, goal, d, graph)
        if r: return r
    return None
```

OR

Q2 (OR). Minimax algorithm. [20 marks]

```
def minimax(node, depth, maximizingPlayer, children_fn, eval_fn):
    if depth==0 or not children_fn(node):
        return eval_fn(node)
    if maximizingPlayer:
        best = -float('inf')
        for ch in children_fn(node):
            best = max(best, minimax(ch, depth-1, False, children_fn, eval_fn))
        return best
    else:
        best = float('inf')
        for ch in children_fn(node):
            best = min(best, minimax(ch, depth-1, True, children_fn, eval_fn))
        return best
```

Slip 9

Q1. Depth First Search using Python. [10 marks]

```
def dfs(graph, start, goal, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    if start == goal:
        return [start]
    for neighbor in graph.get(start, []):
        if neighbor not in visited:
            path = dfs(graph, neighbor, goal, visited)
            if path:
                return [start] + path
    return None

# Example:
if __name__ == '__main__':
    graph = {1:[2,3],2:[4],3:[5,6],4:[7],5:[],6:[8],7:[],8:[]}
    print(dfs(graph,1,8))
```

Q2. Iterative Deepening DFS algorithm. [20 marks]

```
def dls(node, goal, depth, graph, path=None):
    if path is None: path = []
    path = path + [node]
    if node == goal:
        return path
    if depth == 0:
        return None
    for n in graph.get(node, []):
        res = dls(n, goal, depth-1, graph, path)
        if res: return res
    return None

def iddfs(start, goal, graph, maxd=20):
    for d in range(maxd+1):
        r = dls(start, goal, d, graph)
        if r: return r
    return None
```

OR

Q2 (OR). Healthcare Appointment Bot (simple chatbot). [20 marks]

```
def healthcare_bot(query):
    q = query.lower()
    if 'book' in q or 'appointment' in q:
        return "To book, provide preferred date and doctor. Or call 12345."
    if 'timings' in q: return "Clinic hours: 9am-5pm Mon-Fri."
    return "Please contact clinic reception."

if __name__ == '__main__':
    print(healthcare_bot("How to book appointment?"))
```

Slip 10

Q1. Prim's Algorithm to connect cities (minimum cost). [10 marks]

```
import heapq
def prim(adj, start=0):
    visited = {start}; edges = []
    for v,w in adj[start]:
        heapq.heappush(edges, (w, start, v))
    mst = []
    while edges:
        w,u,v = heapq.heappop(edges)
        if v in visited: continue
        visited.add(v)
        mst.append((u,v,w))
        for to,c in adj[v]:
            if to not in visited:
                heapq.heappush(edges, (c, v, to))
    return mst

if __name__ == '__main__':
    adj = {0:[(1,2),(2,3)],1:[(0,2),(2,1)],2:[(0,3),(1,1)]}
    print(prim(adj))
```

Q2. A* Algorithm. [20 marks]

```
import heapq

def astar(start, goal, neighbors_fn, h_fn):
    openh = [(h_fn(start), 0, start, None)]
    came = {}
    gscore = {start:0}
    while openh:
        f, g, node, parent = heapq.heappop(openh)
        if node in came:
            continue
        came[node] = parent
        if node == goal:
            path = []
            n = node
            while n is not None:
                path.append(n)
                n = came[n]
            return list(reversed(path))
        for nbr, cost in neighbors_fn(node):
            ng = g + cost
            if ng < gscore.get(nbr, float('inf')):
                gscore[nbr] = ng
                heapq.heappush(openh, (ng + h_fn(nbr), ng, nbr, node))
    return None

# To use this template, provide neighbors_fn(node) -> list of (nbr, cost)
# and h_fn(node) -> heuristic estimate to goal.
```

OR

Q2 (OR). Mini-Max Algorithm. [20 marks]

```
def minimax(node, depth, maximizingPlayer, children_fn, eval_fn):
    if depth==0 or not children_fn(node):
        return eval_fn(node)
    if maximizingPlayer:
        best = -float('inf')
        for ch in children_fn(node):
            best = max(best, minimax(ch, depth-1, False, children_fn, eval_fn))
        return best
    else:
        best = float('inf')
        for ch in children_fn(node):
            best = min(best, minimax(ch, depth-1, True, children_fn, eval_fn))
        return best
```

Slip 11

Q1. Selection Sort (ascending). [10 marks]

```
def selection_sort_asc(arr):
    a = arr[:]
    n = len(a)
    for i in range(n):
        min_i = i
        for j in range(i+1, n):
            if a[j] < a[min_i]:
                min_i = j
        a[i], a[min_i] = a[min_i], a[i]
    return a

if __name__ == '__main__':
    print(selection_sort_asc([64,25,12,22,11]))
```

Q2. Breadth First Search (graph input start=1 goal=8). [20 marks]

```
from collections import deque
def bfs(graph, start, goal):
    q = deque([[start]]); visited = {start}
    while q:
        path = q.popleft(); node = path[-1]
        if node == goal: return path
        for n in graph.get(node, []):
            if n not in visited:
                visited.add(n); q.append(path + [n])
    return None
```

OR

Q2 (OR). Monkey Banana Problem. [20 marks]

```
from collections import deque

def monkey_banana():
    # State: (monkey_pos, box_pos, monkey_on_box, has_banana)
    start = ("Door", "Window", False, False)
    goal = ("UnderBanana", "UnderBanana", True, True)

    def actions(state):
        m, b, on_box, has = state
        res = []
        # walk
        for loc in ["Door","Window","UnderBanana"]:
            if loc != m:
                res.append((loc, b, on_box, has))
        # push box (only if monkey and box at same pos and not on box)
        if m == b and not on_box:
            for loc in ["Door","Window","UnderBanana"]:
                if loc != b:
                    res.append((m, loc, on_box, has))
        # climb
        if m == b and not on_box:
            res.append((m, b, True, has))
        # grab banana
        if m == "UnderBanana" and on_box:
            res.append((m, b, on_box, True))
        return res

    q = deque([(start, [])])
    visited = set([start])
    while q:
        state, path = q.popleft()
        if state == goal:
            return path + [state]
        for nxt in actions(state):
            if nxt not in visited:
                visited.add(nxt)
                q.append((nxt, path + [state]))
    return None

if __name__ == '__main__':
    sol = monkey_banana()
```

```
for s in sol:
    print(s)
```

Slip 12

Q1. Recursive algorithm to search all vertices of an undirected graph. [10 marks]

```
def dfs_all(graph, node, visited=None):
    if visited is None: visited = set()
    visited.add(node)
    for n in graph.get(node, []):
        if n not in visited:
            dfs_all(graph, n, visited)
    return visited
```

Q2. Depth First Search (start=2, goal=7). [20 marks]

```
def dfs(graph, start, goal, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    if start == goal:
        return [start]
    for neighbor in graph.get(start, []):
        if neighbor not in visited:
            path = dfs(graph, neighbor, goal, visited)
            if path:
                return [start] + path
    return None

# Example:
if __name__ == '__main__':
    graph = {1:[2,3],2:[4],3:[5,6],4:[7],5:[],6:[8],7:[],8:[]}
    print(dfs(graph,1,8))
```

OR

Q2 (OR). Tic-Tac-Toe game. [20 marks]

```
import math

def winner(board):
    wins = [(0,1,2),(3,4,5),(6,7,8),(0,3,6),(1,4,7),(2,5,8),(0,4,8),(2,4,6)]
    for a,b,c in wins:
        if board[a]==board[b]==board[c] and board[a] != ' ':
            return board[a]
    return 'Draw' if ' ' not in board else None

def minimax(board, player):
    w = winner(board)
    if w == 'X': return 1
    if w == 'O': return -1
    if w == 'Draw': return 0
    if player == 'X':
        best = -math.inf
        for i in range(9):
            if board[i] == ' ':
                board[i] = 'X'
                val = minimax(board, 'O')
                board[i] = ' '
                best = max(best, val)
        return best
    else:
        best = math.inf
        for i in range(9):
            if board[i] == ' ':
                board[i] = 'O'
                val = minimax(board, 'X')
                board[i] = ' '
                best = min(best, val)
        return best

def best_move(board):
    best = -math.inf; move = -1
    for i in range(9):
        if board[i] == ' ':
            board[i] = 'X'
            val = minimax(board, 'O')
            board[i] = ' '
```

```
            if val > best:
                best = val; move = i
    return move

if __name__ == '__main__':
    board = [' ']*9
    move = best_move(board)
    board[move] = 'X'
    print(board)
```

Slip 13

Q1. Breadth First Search using Python. [10 marks]

```
from collections import deque
def bfs(graph, start, goal):
    q = deque([[start]]); visited = {start}
    while q:
        path = q.popleft(); node = path[-1]
        if node == goal: return path
        for n in graph.get(node, []):
            if n not in visited:
                visited.add(n); q.append(path + [n])
    return None
```

Q2. Implement AO* Algorithm. [20 marks]

```
# AO* algorithm sketch: This is a conceptual placeholder.
# A full AO* requires AND-OR graph representation, heuristics, and backtracking.
# For exam purposes, present algorithm steps and a short pseudocode.
def ao_star_sketch():
    pass
```

OR

Q2 (OR). Depth First Search using Python. [20 marks]

```
def dfs(graph, start, goal, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    if start == goal:
        return [start]
    for neighbor in graph.get(start, []):
        if neighbor not in visited:
            path = dfs(graph, neighbor, goal, visited)
            if path:
                return [start] + path
    return None

# Example:
if __name__ == '__main__':
    graph = {1:[2,3],2:[4],3:[5,6],4:[7],5:[],6:[8],7:[],8:[]}
    print(dfs(graph,1,8))
```

Slip 14

Q1. Prim's Algorithm. [10 marks]

```
import heapq
def prim(adj, start=0):
    visited = {start}; edges = []
    for v,w in adj[start]:
        heapq.heappush(edges, (w, start, v))
    mst = []
    while edges:
        w,u,v = heapq.heappop(edges)
        if v in visited: continue
        visited.add(v)
        mst.append((u,v,w))
        for to,c in adj[v]:
            if to not in visited:
                heapq.heappush(edges, (c, v, to))
    return mst

if __name__ == '__main__':
    adj = {0:[(1,2),(2,3)],1:[(0,2),(2,1)],2:[(0,3),(1,1)]}
    print(prim(adj))
```

Q2. A* Algorithm. [20 marks]

```
import heapq

def astar(start, goal, neighbors_fn, h_fn):
    openh = [(h_fn(start), 0, start, None)]
    came = {}
    gscore = {start:0}
    while openh:
        f, g, node, parent = heapq.heappop(openh)
        if node in came:
            continue
        came[node] = parent
        if node == goal:
            path = []
            n = node
            while n is not None:
                path.append(n)
                n = came[n]
            return list(reversed(path))
        for nbr, cost in neighbors_fn(node):
            ng = g + cost
            if ng < gscore.get(nbr, float('inf')):
                gscore[nbr] = ng
                heapq.heappush(openh, (ng + h_fn(nbr), ng, nbr, node))
    return None

# To use this template, provide neighbors_fn(node) -> list of (nbr, cost)
# and h_fn(node) -> heuristic estimate to goal.
```

OR

Q2 (OR). Missionaries and Cannibals Problem. [20 marks]

```
from collections import deque
def missionaries_cannibals():
    start=(3,3,1); goal=(0,0,0)
    def valid(m,c): return 0<=m<=3 and 0<=c<=3 and (m==0 or m>=c)
    moves = [(1,0),(2,0),(0,1),(0,2),(1,1)]
    q = deque([start]); parent={start:None}
    while q:
        m,c,b = q.popleft()
        if (m,c,b) == goal:
            path=[]; cur=(m,c,b)
            while cur:
                path.append(cur); cur=parent[cur]
            return list(reversed(path))
        for dm,dc in moves:
            if b==1:
                nm, nc = m-dm, c-dc; nb=0
            else:
                nm, nc = m+dm, c+dc; nb=1
```

```
            if valid(nm,nc) and valid(3-nm,3-nc):
                st=(nm,nc,nb)
                if st not in parent:
                    parent[st]=(m,c,b); q.append(st)
    return None

if __name__ == '__main__':
    print(missionaries_cannibals())
```

Slip 15

Q1. Depth First Search. [10 marks]

```
def dfs(graph, start, goal, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    if start == goal:
        return [start]
    for neighbor in graph.get(start, []):
        if neighbor not in visited:
            path = dfs(graph, neighbor, goal, visited)
            if path:
                return [start] + path
    return None

# Example:
if __name__ == '__main__':
    graph = {1:[2,3],2:[4],3:[5,6],4:[7],5:[],6:[8],7:[],8:[]}
    print(dfs(graph,1,8))
```

Q2. Alpha-Beta Pruning. [20 marks]

```
def alphabeta(node, depth, alpha, beta, maximizingPlayer, children_fn, eval_fn):
    if depth == 0 or not children_fn(node):
        return eval_fn(node)
    if maximizingPlayer:
        value = -float('inf')
        for child in children_fn(node):
            value = max(value, alphabeta(child, depth-1, alpha, beta, False, children_fn, eval_fn))
            alpha = max(alpha, value)
            if alpha >= beta:
                break
        return value
    else:
        value = float('inf')
        for child in children_fn(node):
            value = min(value, alphabeta(child, depth-1, alpha, beta, True, children_fn, eval_fn))
            beta = min(beta, value)
            if beta <= alpha:
                break
        return value
```

OR

Q2 (OR). Restaurant Reservation Assistant (chatbot). [20 marks]

```
def reservation_bot(q):
    q = q.lower()
    if 'reserve' in q or 'book' in q:
        return "Please send date, time and party size."
    if 'menu' in q:
        return "Menu: Starter, Main Course, Desserts. See website for details."
    return "Contact support for more help."

if __name__ == '__main__':
    print(reservation_bot("Book a table for 2 on Friday"))
```

Slip 17

Q1. Depth First Search. [10 marks]

```
def dfs(graph, start, goal, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    if start == goal:
        return [start]
    for neighbor in graph.get(start, []):
        if neighbor not in visited:
            path = dfs(graph, neighbor, goal, visited)
            if path:
                return [start] + path
    return None

# Example:
if __name__ == '__main__':
    graph = {1:[2,3],2:[4],3:[5,6],4:[7],5:[],6:[8],7:[],8:[]}
    print(dfs(graph,1,8))
```

Q2. Water Jug Problem. [20 marks]

```
from collections import deque
def water_jug(a_cap, b_cap, target):
    start = (0,0); q = deque([start]); visited = {start}; parent = {start:None}
    while q:
        x,y = q.popleft()
        if x==target or y==target:
            path=[]; cur=(x,y)
            while cur:
                path.append(cur); cur=parent[cur]
            return list(reversed(path))
        moves = []
        moves.append((a_cap,y)); moves.append((x,b_cap))
        moves.append((0,y)); moves.append((x,0))
        t = min(x, b_cap-y); moves.append((x-t, y+t))
        t = min(y, a_cap-x); moves.append((x+t, y-t))
        for m in moves:
            if m not in visited:
                visited.add(m); parent[m]=(x,y); q.append(m)
    return None

if __name__ == '__main__':
    print(water_jug(4,3,2))
```

OR

Q2 (OR). Tic-Tac-Toe (take). [20 marks]

```
import math

def winner(board):
    wins = [(0,1,2),(3,4,5),(6,7,8),(0,3,6),(1,4,7),(2,5,8),(0,4,8),(2,4,6)]
    for a,b,c in wins:
        if board[a]==board[b]==board[c] and board[a] != ' ':
            return board[a]
    return 'Draw' if ' ' not in board else None

def minimax(board, player):
    w = winner(board)
    if w == 'X': return 1
    if w == 'O': return -1
    if w == 'Draw': return 0
    if player == 'X':
        best = -math.inf
        for i in range(9):
            if board[i] == ' ':
                board[i] = 'X'
                val = minimax(board, 'O')
                board[i] = ' '
                best = max(best, val)
        return best
    else:
        best = math.inf
```

```
        for i in range(9):
            if board[i] == ' ':
                board[i] = 'O'
                val = minimax(board, 'X')
                board[i] = ' '
                best = min(best, val)
        return best

def best_move(board):
    best = -math.inf; move = -1
    for i in range(9):
        if board[i] == ' ':
            board[i] = 'X'
            val = minimax(board, 'O')
            board[i] = ' '
            if val > best:
                best = val; move = i
    return move

if __name__ == '__main__':
    board = [' ']*9
    move = best_move(board)
    board[move] = 'X'
    print(board)
```

Slip 18

Q1. Breadth First Search. [10 marks]

```
from collections import deque
def bfs(graph, start, goal):
    q = deque([[start]]); visited = {start}
    while q:
        path = q.popleft(); node = path[-1]
        if node == goal: return path
        for n in graph.get(node, []):
            if n not in visited:
                visited.add(n); q.append(path + [n])
    return None
```

Q2. Tic-Tac-Toe Game. [20 marks]

```
import math

def winner(board):
    wins = [(0,1,2),(3,4,5),(6,7,8),(0,3,6),(1,4,7),(2,5,8),(0,4,8),(2,4,6)]
    for a,b,c in wins:
        if board[a]==board[b]==board[c] and board[a] != ' ':
            return board[a]
    return 'Draw' if ' ' not in board else None

def minimax(board, player):
    w = winner(board)
    if w == 'X': return 1
    if w == 'O': return -1
    if w == 'Draw': return 0
    if player == 'X':
        best = -math.inf
        for i in range(9):
            if board[i] == ' ':
                board[i] = 'X'
                val = minimax(board, 'O')
                board[i] = ' '
                best = max(best, val)
        return best
    else:
        best = math.inf
        for i in range(9):
            if board[i] == ' ':
                board[i] = 'O'
                val = minimax(board, 'X')
                board[i] = ' '
                best = min(best, val)
        return best

def best_move(board):
    best = -math.inf; move = -1
    for i in range(9):
        if board[i] == ' ':
            board[i] = 'X'
            val = minimax(board, 'O')
            board[i] = ' '
            if val > best:
                best = val; move = i
    return move

if __name__ == '__main__':
    board = [' ']*9
    move = best_move(board)
    board[move] = 'X'
    print(board)
```

OR

Q2 (OR). FAQ Bot for University Website. [20 marks]

```
def faq(query):
    q = query.lower()
    if 'admission' in q: return "Admissions open July; see registrar."
    if 'contact' in q: return "Call 020-xxxxx or email support@uni.edu."
    return "Visit website or contact admin."
```

```
if __name__ == '__main__':
    print(faq("How to apply for admission?"))
```

Slip 19

Q1. Depth First Search. [10 marks]

```
def dfs(graph, start, goal, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    if start == goal:
        return [start]
    for neighbor in graph.get(start, []):
        if neighbor not in visited:
            path = dfs(graph, neighbor, goal, visited)
            if path:
                return [start] + path
    return None

# Example:
if __name__ == '__main__':
    graph = {1:[2,3],2:[4],3:[5,6],4:[7],5:[],6:[8],7:[],8:[]}
    print(dfs(graph,1,8))
```

Q2. Breadth First Search (start=1 goal=8). [20 marks]

```
from collections import deque
def bfs(graph, start, goal):
    q = deque([[start]]); visited = {start}
    while q:
        path = q.popleft(); node = path[-1]
        if node == goal: return path
        for n in graph.get(node, []):
            if n not in visited:
                visited.add(n); q.append(path + [n])
    return None
```

OR

Q2 (OR). A* Algorithm. [20 marks]

```
import heapq

def astar(start, goal, neighbors_fn, h_fn):
    openh = [(h_fn(start), 0, start, None)]
    came = {}
    gscore = {start:0}
    while openh:
        f, g, node, parent = heapq.heappop(openh)
        if node in came:
            continue
        came[node] = parent
        if node == goal:
            path = []
            n = node
            while n is not None:
                path.append(n)
                n = came[n]
            return list(reversed(path))
        for nbr, cost in neighbors_fn(node):
            ng = g + cost
            if ng < gscore.get(nbr, float('inf')):
                gscore[nbr] = ng
                heapq.heappush(openh, (ng + h_fn(nbr), ng, nbr, node))
    return None

# To use this template, provide neighbors_fn(node) -> list of (nbr, cost)
# and h_fn(node) -> heuristic estimate to goal.
```

Slip 20

Q1. Depth First Search. [10 marks]

```
def dfs(graph, start, goal, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    if start == goal:
        return [start]
    for neighbor in graph.get(start, []):
        if neighbor not in visited:
            path = dfs(graph, neighbor, goal, visited)
            if path:
                return [start] + path
    return None

# Example:
if __name__ == '__main__':
    graph = {1:[2,3],2:[4],3:[5,6],4:[7],5:[],6:[8],7:[],8:[]}
    print(dfs(graph,1,8))
```

Q2. AO* Algorithm. [20 marks]

```
# AO* algorithm sketch: This is a conceptual placeholder.
# A full AO* requires AND-OR graph representation, heuristics, and backtracking.
# For exam purposes, present algorithm steps and a short pseudocode.
def ao_star_sketch():
    pass
```

OR

Q2 (OR). Kruskal's Algorithm. [20 marks]

```
def kruskal(edges, n):
    parent = list(range(n))
    rank = [0]*n
    def find(x):
        while parent[x] != x:
            parent[x] = parent[parent[x]]
            x = parent[x]
        return x
    def union(a,b):
        ra, rb = find(a), find(b)
        if ra == rb: return False
        if rank[ra] < rank[rb]:
            parent[ra] = rb
        else:
            parent[rb] = ra
            if rank[ra] == rank[rb]:
                rank[ra] += 1
        return True
    edges = sorted(edges)
    mst = []; total = 0
    for w,u,v in edges:
        if union(u,v):
            mst.append((u,v,w)); total += w
    return mst, total

if __name__ == '__main__':
    edges = [(1,0,1),(2,0,2),(3,1,2)]
    print(kruskal(edges,3))
```