Data Structures and Algorithms

End-semester Examination (Memory-Based)

Q1. Matrix Common Element Problem [10 marks]

You are given a square matrix of size n × n. Your task is to identify the maximum value that occurs in every single row of the matrix. If no such value exists, indicate this in your solution.

Example:
Consider the following 5×5 matrix:

```
[9  6  3  8  5]
[3  5  1  6  8]
[0  7  5  3  5]
[3  5  7  8  6]
[4  3  5  7  9]
```

The solution for this matrix is **5**, as it appears in all rows and is the largest such number.

Part A [8 marks]

Design and describe your solution approach using any of the following formats:

- Step-by-step algorithmic description
- Pseudocode implementation
- Complete code solution

You may optionally include a worked example to demonstrate your algorithm.

Part B [2 marks]

Analyze your algorithm's computational complexity. Provide both time and space complexity with justification. Your response will be evaluated based on accuracy, optimization, and clear explanation.

Q2. Anti-Reflection Graph Properties Problem [12 marks]

A directed graph G with n vertices has an adjacency matrix A that satisfies the **anti-reflection property** when both of the following conditions are met:

1. **Zero diagonal**: A[i][i] = 0 for all vertices i (no self-loops exist)
2. **Complementary edges**: For any pair of distinct vertices i and j, exactly one of A[i][j] or A[j][i] equals 1, while the other equals 0

Illustrative Example:
The following 5×5 adjacency matrix demonstrates the anti-reflection property:

```
     [0  1  0  1  1]
     [0  0  0  1  0]
A =  [1  1  0  1  1]
     [0  0  0  0  1]
     [0  1  0  0  0]
```

Part A [2 marks]

Construct the directed graph represented by the given adjacency matrix A using 5 vertices. Then demonstrate that this graph contains a Hamiltonian path (a path visiting all 5 vertices exactly once, using 4 edges).

Part B [4 marks]

Provide a mathematical proof showing that any directed graph G on n vertices whose adjacency matrix has the anti-reflection property must contain a Hamiltonian path (a path with n-1 edges that visits all n vertices).

Part C [6 marks]

Design an algorithm that takes an n×n anti-reflection adjacency matrix as input and outputs a Hamiltonian path in the corresponding directed graph. Your submission should include:

- Algorithm implementation (code or pseudocode)
- Time complexity analysis
- Correctness proof

Performance Requirements: Target $O(n \log n)$ time complexity, though $O(n^2)$ solutions will receive equal credit.

Q3. Longest Path Algorithms Analysis [16 marks]

This problem explores computing longest paths in graphs using polynomial-time approaches by adapting shortest path algorithms. We'll analyze four different strategies.

Part A [4 marks]

You have a directed graph G with positive edge weights and want to find a longest path. Consider modifying Dijkstra's shortest path algorithm by changing just one line:

Original Dijkstra's Algorithm:

```
if(dist(u) + wt((u,v)) < dist(v))
    dist(v) ← dist(u) + wt((u,v))
```

Modified "Longest Path" Algorithm:

```
if(dist(u) + wt((u,v)) > dist(v))
    dist(v) ← dist(u) + wt((u,v))
```

Question: Does this modified algorithm correctly compute longest paths in G? If yes, provide a step-by-step proof. If no, explain the failure and provide a counter-example.

Part B [4 marks]

You have a directed graph G with positive edge weights and want to find a longest path. Create a new graph G^neg by multiplying all edge weights in G by -1. The vertices and edges remain identical to G. For example, an edge with weight 17 in G becomes weight -17 in G^neg.

The idea: A longest path in G corresponds to a shortest path in G^neg. Use the Bellman-Ford-Moore algorithm to compute shortest paths in G^neg.

Question: Does this approach always correctly compute longest paths in G? If yes, provide a step-by-step proof. If no, identify the flaw and provide a counter-example.

Part C [4 marks]

You have a directed graph G with positive integer edge weights and want to find a longest path. Create a new graph G^inv by taking the reciprocals of all edge weights in G. The vertices and edges remain identical to G. For example, an edge with weight 17 in G becomes weight 1/17 in G^inv.

The idea: A longest path in G corresponds to a shortest path in G^inv. Use Dijkstra's algorithm to compute shortest paths in G^inv.

Question: Does this approach always correctly compute longest paths in G? If yes, provide a step-by-step proof. If no, identify the flaw and provide a counter-example.

Part D [4 marks]

You have a directed graph G with positive integer edge weights and want to find longest paths from a starting vertex s to all other vertices.

Proposed Algorithm:

- **Step 1:** For each vertex v, compute all paths from s to v containing exactly one edge (with their weights)
- **Step 2:** Extend each path from Step 1 by one edge to compute all paths from s to v containing exactly two edges
- **Step 3:** Extend each path from Step 2 by one edge to compute all paths from s to v containing exactly three edges
- **...continuing this pattern...**
- **Step (n-1):** Compute all paths containing exactly (n-1) edges by extending paths from Step (n-2)
- **Step n:** For each vertex v, find the longest path among all paths from s to v (computed in Steps 1 through (n-1))

Analysis: The algorithm notes that Step 1 has O(n) total paths, and each subsequent step has at most O(m) times the size of the previous step, where m is the number of edges. Since we perform n steps, this should be polynomial time.

Questions:

1. Does this algorithm run in polynomial time?
2. Does it correctly solve the longest path problem?
3. If not, identify the flaw in the analysis and provide a counter-example.

Q4. Delicate Graph Construction Problem [7 marks]

Note: Throughout this problem, assume n is even and $n \geq 4$.

An edge (u,v) in a directed graph G is **delicate** if removing that specific edge eliminates all paths from u to v. A directed graph is **delicate** if every edge in the graph is delicate.

Example: The graph below with n = 4 vertices and $n^2/4 = 4$ edges is delicate:

```
u ──→ v
↑     ↓
x ←── w
```

Figure 1 Analysis: This graph is delicate because:

- If (u,v) is removed: no path from u to v exists
- If (v,w) is removed: no path from v to w exists
- If (w,x) is removed: no path from w to x exists
- If (x,u) is removed: no path from x to u exists

Part A [2 marks]

Draw a directed graph with n = 6 vertices and $n^2/4 = 9$ edges that satisfies the delicate property.

Part B [5 marks]

Develop a general construction strategy for creating delicate directed graphs with n vertices and $n^2/4$ edges, where n is any even number ≥ 4. Your strategy should provide a systematic approach that works for all valid values of n.

Q5. Majority Element Detection Problem [10 marks]

Design an algorithm to determine in O(n) time whether an unsorted array of n integers contains more than n/4 copies of any single value. Your task is to identify if there exists a number that appears more than n/4 times in the array.

Example: In the array below, the number 7 appears more than n/4 times:

```
[2, 7, 3, 5, 7, 1, 7, 7, 4, 3, 9, 6, 2, 7, 8, 4, 7, 0, 7, 5]
```

Note: You cannot make assumptions about the array entries other than that they are integers.

Part A [8 marks]

Provide your algorithm description using any of the following formats:

- Step-by-step algorithmic description
- Pseudocode implementation
- Complete code solution

Including a worked example is optional but may be helpful.

Part B [2 marks]

Analyze and explain your algorithm's complexity. Your evaluation will be based on correctness, efficiency, and clarity of explanation.

Q6. Strongly Connected Components Algorithm Analysis [5 marks]

A new algorithm is proposed for finding strongly connected components of a directed graph. The algorithm works as follows:

1. **First DFS scan:** Perform DFS on the original graph to mark finishing times for each node (as in the standard algorithm)
2. **Second DFS scan:** Conduct DFS on the **original graph** (not the transpose), processing vertices in decreasing order of their finishing times from step 1
3. **Output:** Each DFS tree found in the second scan is reported as a strongly connected component

Question:

Evaluate this proposed algorithm. Does it correctly identify strongly connected components? Provide either:

- A proof of correctness, OR
- A counter-example demonstrating where the algorithm fails

Your analysis should clearly explain why the algorithm works or provide a specific graph example showing incorrect behavior.