Part 1: Status of Code

Required .java files:
Node.java: Operational and fully implemented
Partition.java: Operational and fully implemented
IslandSurvey.java: Operational and fully implemented
IslayLakeSurvey.java: Operational and fully implemented

Required test files:
map1.txt: Required test case
map2.txt: Required test case
map3.txt: Required test case
map4.txt: Required test case
map5.txt: Custom test case
map6.txt: Custom test case

Output files
map1Output.txt: Required test case, completed with 2B output.
map2Output.txt: Required test case, completed with 2B output.
map3Output.txt: Required test case, completed with 2B output.
map4Output.txt: Required test case, completed with 2B output.
map5Output.txt: Custom test case, completed with 2B output.
map6Output.txt: Custom test case, completed with 2B output.

Known bugs and issues:

1. (NOT ISSUE WITH CODE): On Windows Command prompt/power shell, running java IslandSurvey < mapX.txt > mapXOutput.txt, sometimes produces random UTF-16 output (consists of random text if opened as UTF-8).

   Note: This is an encoding/viewer mismatch, not a logic error. Easiest work around is re-open file with proper UTF encoding.

Part 2: Testing

Testing was done by using assignment provided test cases (map1.txt-map4.txt) and self-designed test cases. See Part 1 regarding which files are what.

The Custom Test cases:

- map5.txt (4×4 grid) - Tested full boundary conditions: no islands, multiple isolated islands, and full merging into a single island.
- map6.txt (6×5 grid) - Tested incremental union logic without lake formation, confirming no false positives in lake detection.

All outputs from these files have matched proper computations, confirming that the algorithm specified in the assignment is correctly implemented. Specifically the 4-neighbor and 8-neighbor partition behavior, accurate phase updates, and properly formatted output are all functionals.

Part 3: Program

3.1 Partition ADT

The Partition class implements a sequence based disjoint set structure using singly linked lists to represent clusters. The class has a nested static class (named ‘Cluster’) with properties of a head and tail nodes and size as an integer. The partition class has an attribute (an ArrayList named ‘clusters’) to keep track of all clusters in a sequenced based manner. Note that the Partition ADT utilizes an external class (Node.java) which points back to its owning cluster while containing an element. As instructed in the assignment, the partition ADT contains all required methods operating at the most efficient possible way. The makeCluster() method creates a new singleton node and adds its cluster to the ArrayList in O(1) time. The find() method returns the cluster’s leader (head) in O(1). The union() method identifies the smaller cluster, then merges the smaller cluster into the larger one by re-arranging pointers (updating all node references to maintain cluster integrity); running at $O(\min(n_1, n_2))$. The clusterSizes() method collects and sorts cluster sizes in decreasing order with time complexity (O(nlogn). ClusterPositions() traverses a single cluster’s nodes (O(n)). An ArrayList was utilized to preserve sequence order and enable efficient access, merging, and reporting operations that align directly with the assignment’s required algorithms.

3.2 Part 2A – IslandSurvey

Part 2A uses the assignment provided algorithm. It first parses the given input file, creating a 2D char array. After, it loops over the array, utilizing a Partition ADT to model connected components of black cells (“islands”) on a grid. Each '1' cell forms a cluster, and if there is any adjacency on neighbouring clusters (up, down, left, right), they are unioned together. After constructing all clusters, the program reports the number of islands, their sizes in decreasing order, and the total island area. Rather than checking every neighbour (up, down, left, right), the program uses a more efficient approach by beginning in the top left of the array, checking for adjacency on every right, and bottom to connect land masses. This is to avoid redundancy and ensure linear-time performance relative to grid size.

3.3 Part 2B – IslandLakeSurvey

Part 2B is building off of IslandSurvey, by introducing a second partition (WP) for white cells, using 8-neighbor connectivity to track potential lakes. Note that a lake is defined as a white cluster that does not touch the outer border, and is side-adjacent to exactly one island. The program counts all such lakes, assigns each to its containing island, and adds its area to that island’s total size. The white partition is rebuilt each phase to maintain accurate topology as new land appears.

Part 4: Resources

Referenced Old Code regarding LinkedLists from ITI 1121
Appendix provided in assignment.
Program1csi2110F2025.pdf provided in assignment.
https://stackoverflow.com/questions/13185727/reading-a-txt-file-using-scanner-class-in-java
https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html
https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html
https://www.w3schools.com/java/java_arraylist.asp
