Code

I used Google's Guava library in order to make use of its collection types. ImmutableList, ImmutableTable, and ImmutableMap were particularly useful for the project. Because references to immutable objects can be cached as they are not going to change. In addition to immutable collections, I used Google's AutoValue library. AutoValue provides an easy way to create immutable classes. Considering that data structures like Map are used in the project, having immutable objects are vital since they are good candidates for being key.

Maven became convenient for managing to build the project. It was quite useful since it makes the build process easy and allows describing dependencies effortlessly.

Source code of the project is present in Assignment1/src/main/java/Main.java.

- readFile and writeFile methods are for I/O operations.
- applyDFS, applyBFS and applyAstar methods searches end coordinate with depth-first search, breadth-first search and A* search respectively.
- isMoveLegal method determines whether a neighbor can be added to frontier or not. If problem is a labyrinth, then move is legal if the neighbor is not a wall. If the problem is mountain, then move is legal if the height difference between the current coordinate and the neighbor coordinate is less than or equal to 1.
- computeDistanceTraveled method computes the distance traveled from the start coordinate to end coordinate. If problem is labyrinth, distance is path length minus one. If the problem is mountain, distance is sum of Euclidean distance between each consecutive coordinate in the path.
- isUsedLess method is used for A* search. It determines whether the new cost of a node is less than the present cost of it or not.

Input files should be located in Assignment1/src/main/resources/data. Output files are generated in Assignment1/output.

Comparison

Search algorithms are commonly evaluated according to the following four criteria (each evaluation is made according to the given problem and data in the assignment):

1. Completeness:Is the algorithm guaranteed to find a solution when there is one?
    - DFS: Solutions exists in all given problems. In addition, all problems have finite space. Therefore, DFS is complete.
    - BFS: Solutions exists in all given problems. In addition branching factor (4) is finite. Therefore, BFS is complete.

    - A*: Solutions exists in all given problems. There are not infinitely many nodes. Therefore, A* search is complete.
2. Time Complexity: How long does it take to find a solution?
    - DFS: O(b^m) where b is branching factor and m is the maximum length of any path in the state space
    - BFS: O(b^(d+1)) where b is branching factor and d is the depth of the shallowest goal node
    - A*: O(b^m) where b is branching factor and m is the maximum length of any path in the state space
3. Space Complexity: How much memory is needed to perform the search?
    - DFS: O(bm) where b is branching factor and m is the maximum length of any path in the state space
    - BFS: O(b^(d+1)) where b is branching factor and d is the depth of the shallowest goal node
    - A*: O(b^m) where b is branching factor and m is the maximum length of any path in the state space
4. Optimality: Does the strategy find the optimal solution?
    - DFS: DFS is not optimal.
    - BFS: BFS is not optimal unless step costs are constant.
    - A*: A* search is optimal because of admissible heuristics and monotonicity.

Optimality is associated with optimal path. Output files show that optimal path is found by BFS and A* in labyrinth problems since step cost in labyrinth problem is constant. On the other hand, path cost of BFS in mountain problems is greater than or equal to the path cost of A* search since step cost is not constant. Hence, optimal path is found by A* in mountain problems.

Time complexity is associated with number of discovered nodes. Number of discovered nodes for each input are in below table

	BFS	**DFS**	**A***
labyrinth_test_1.txt	70	31	31
labyrinth_test_2.txt	59	47	33
labyrinth_test_3.txt	65	57	58
labyrinth_test_4.txt	141	60	50
labyrinth_test_5.txt	65	63	51
mountain_test_1.txt	146	129	113
mountain_test_2.txt	149	24	102
mountain_test_3.txt	128	125	51
mountain_test_4.txt	39	111	9

mountain_test_5.txt	147	31	53

A* search discovers the least number of nodes in 7 (4 labyrinth, 3 mountain) out of 10 inputs. DFS discovers the least number of nodes in 4 (2 labyrinth, 2 mountain) out of 10 inputs (there is a tie in input1). Therefore, we can conclude that though time complexity of DFS is the lowest, A* generally discovers less nodes than DFS in our problem set.