Yue Fang ASUID: 1215500923 Java -version : java11
Jgrapht -version: 1.5.0

requirement 1: I perform 5 refactoring, specifically as follows:

1. Refactoring: Extract Method notempty to isEmpty

   - Type: Extract Method
   - Reason: The original method name notempty is negative, and it is better to use a positive method name. Therefore, notempty is refactored to isEmpty.
   - Commit Link: https://github.com/AttackBlueSteel/CSE464YueFang/pull/1/commits/a110443edb35fd3bbd237f8bd0ce025109a72432

2. Refactoring: Convert All Underline To Hump

   - Type: Rename
   - Reason: To conform with the naming conventions used in Java where camelCase is the preferred naming style for method names and variable names. This makes the code more readable and easier to understand.
   - Commit Link: https://github.com/AttackBlueSteel/CSE464YueFang/pull/1/commits/06c8d9c9ceaf1bd142a026496b4fa3ae6b619b87

3. Refactoring: Replace for loop with enhanced for

   - Type: Readability
   - Reason: The enhanced for loop is a more concise and readable way to iterate over arrays or collections compared to a traditional for loop. It eliminates the need for index variables and reduces the potential for off-by-one errors. Additionally, it can make the code easier to maintain and understand by removing unnecessary details and focusing on the actual iteration logic.
   - Commit Link: https://github.com/AttackBlueSteel/CSE464YueFang/pull/1/commits/e82b559b477fb7745217c5068b82888d2a0f5633

4. Refactoring: Remove unnecessary cast

   - Type: Readability and Performance
   - Rreason: the type of temp.getPath() is string, declaring it as an object is not necessary
   - Commit Link: https://github.com/AttackBlueSteel/CSE464YueFang/pull/1/commits/84f9f712c979cb4fd7e714cefff30e12d8403d2b

5. Refactoring: String concatenation with StringBuilder

   - Type: Performance
   - Reason: The reason for using StringBuilder instead of the string concatenation operator is that the concatenation operator creates a new string object each time it is used. This can result in a large number of objects being created in memory, which can impact performance and memory usage. In contrast, the StringBuilder class creates a single buffer that can be appended to, resulting in fewer objects being created.
   - Commit Link: https://github.com/AttackBlueSteel/CSE464YueFang/pull/1/commits/9a2965514fabc2b9b9ac4cc09d57d8519a7b3e96

requirement 2: implementation details are as follows

- Explain: the GraphSearchAlgorithm is an abstract class that defines the basic structure of the search algorithm using a graph. It has an abstract method addNodeToQueue and addNeighborsToQueue, which subclasses BFS and DFS implement to provide their specific implementations. The search method in the GraphSearchAlgorithm class provides the overall structure of the search algorithm using the template method pattern. It initializes a queue and a set of visited nodes and adds the start node to the queue. It then enters a loop that dequeues the next path, checks if the last node in the path has already been visited, adds it to the visited nodes set, and checks if it is the destination node. If it is, it returns the path. Otherwise, it adds the neighbors of the current node to the queue using the addNeighborsToQueue method, if they have not been visited before. The BFS and DFS classes extend the GraphSearchAlgorithm class and implement the addNodeToQueue and addNeighborsToQueue methods based on the specific requirements of each search algorithm. In particular, BFS adds nodes to the end of the queue, while DFS adds them to the front of the queue. Also, BFS uses a FIFO queue to traverse the graph while DFS uses a LIFO

stack.
- Commit Link: https://github.com/AttackBlueSteel/CSE464YueFang/pull/1/commits/a592994c853182059ecc740102655cc0e137c69b

requirement 3: implementation details are as follows

- Explain: The main interface GraphSearch provides the search method to search for a path between two nodes in the graph. The GraphSearchAlgorithm is an abstract class that implements the GraphSearch interface and provides the common functionality for both BFS and DFS algorithms. It has an abstract method addNodeToQueue that is implemented by its concrete subclasses to add the first node to the search queue, and another abstract method addNeighborsToQueue that is implemented to add the neighbors of a node to the search queue. The search method is implemented by the GraphSearchAlgorithm and performs the actual search using the given algorithm.The GraphSearchStrategy class acts as a factory that creates an instance of GraphSearchAlgorithm based on the selected algorithm. It takes in the graph, source and destination nodes, and the selected algorithm as input and returns the path between the source and destination nodes using the selected algorithm. It uses a switch statement to choose between BFS and DFS and creates an instance of the corresponding algorithm.The BFS and DFS classes are concrete subclasses of GraphSearchAlgorithm that implement the addNodeToQueue and addNeighborsToQueue methods based on their respective algorithms. BFS implements breadth-first search algorithm by adding the nodes to the queue in a breadth-first manner. DFS implements depth-first search algorithm by adding the nodes to the queue in a depth-first manner.
- Commit Link: https://github.com/AttackBlueSteel/CSE464YueFang/pull/1/commits/220781fe498d6f7e12067bb5ca0e248689d929cc

requirement 4: implementation details are as follows

- Commit Link: https://github.com/AttackBlueSteel/CSE464YueFang/pull/1/commits/f8be8502d07d01b5bacc8e5776174f06bc811bb7