Yukta Sarode
1225266406
CSE 464
Project Part 3

1) Github link - https://github.com/yuktasarode/CSE4642023_ysarode
2) Refactorings:
    a) Refactor 1 -
    https://github.com/yuktasarode/CSE4642023_ysarode/commit/7cd44396ec11c3811eb003643f6da724d2ffcf17

    Magic strings like "PNG" is replaced with constants to improve code maintainability. By defining PNG_FORMAT as a constant, the code becomes more readable, and it reduces the risk of typos and inconsistencies.

    b) Refactor 2-
    https://github.com/yuktasarode/CSE4642023_ysarode/commit/62c3e04a9ad2543e812f9cb9483b8abad45a2ca4

    Made variable declaration location consistent and clear at the top of the code.Enhancing code readability.

    c) Refactor 3-
    https://github.com/yuktasarode/CSE4642023_ysarode/commit/06f12aa566c3f19a7fa038bc918cbdc8a506f56d

    Separate the graph parsing logic from the graph file reading in the parseGraph method. This allows for better error handling and more meaningful error messages.

    d) Refactor 4-
    https://github.com/yuktasarode/CSE4642023_ysarode/commit/ad079d0a0d9d09422f2f7ce5336d39e0fac83f65

    Simplified the construction of the edges section in the toString method using the forEach loop for clarity and simplicity. Also removed the unnecessary substring manipulation for removing the trailing comma and space.

    e) Refactor 5-
    https://github.com/yuktasarode/CSE4642023_ysarode/commit/8c7d7386294b626eaefcc2d75c88bac3125214b4

Improved the logic of toString() in Path class by making it concise and improving readability.

f) Refactor 6- https://github.com/yuktasarode/CSE4642023_ysarode/commit/4459f56aa2633f69ad6fefe160dd852818a90b05

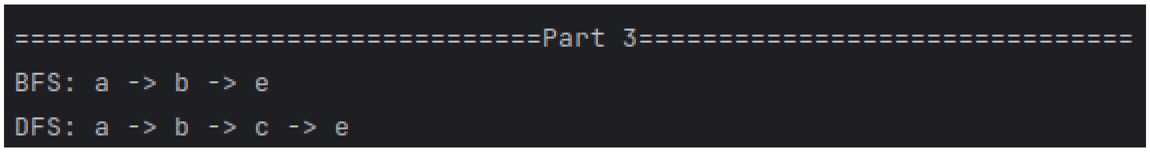
   Refactored the Path class to use the List interface and used contains in containsNode function, made it concise and simplified it. Also encapsulated the nodes field for better encapsulation.

3) Answer:
   a) In my code, I've applied the template pattern to capture the common behaviors of BFS (Breadth-First Search) and DFS (Depth-First Search) while allowing for the customization of specific details in each algorithm.
   b) Firstly, I created an abstract class called GraphSearchTemplate. This class serves as a template for graph search algorithms and contains a template method called processResult. Inside this method, I've encapsulated the common steps involved in searching a graph which is building the path after BFS/ DFS. To accommodate the differences between BFS and DFS, I declared an abstract method graphSearch, which acts as a hook method to be implemented by concrete subclasses.
   c) The processResult method within the abstract class handles the common steps for processing the result of the search. It's called within the template method but can be overridden by subclasses if necessary.
   d) Next, I implemented two concrete classes: BFSGraphSearch and DFSGraphSearch, both of which extend the GraphSearchTemplate. In these concrete classes, I provided specific implementations for the searchAlgorithm method to represent the details of BFS and DFS search strategies. The constructors of these classes call the superclass constructor to set up the graph for the search.
   e) Finally, in the main code, I have a method called Path GraphSearch(String start, String end, algo a). This method takes a start node, end node, and an algorithm type (algo a). Depending on the specified algorithm type, it creates an instance of the corresponding concrete class (BFSGraphSearch or DFSGraphSearch) and invokes the graphSearch method.
   f) This design using the template pattern allows for a clean separation of common behaviors and specific details in graph search algorithms, promoting code reusability and ease of extension or modification.
   g) Commits - https://github.com/yuktasarode/CSE4642023_ysarode/commit/474ce2d8002e4c0277400b2304f647c993b80b0b

4) Answer:
    a) I've implemented the Strategy Pattern in my code to dynamically choose between Breadth-First Search (BFS) and Depth-First Search (DFS) algorithms for graph traversal. Let me break down how I applied this pattern:
    b) Strategy Interface (Algo):
        i) I defined the Algo interface to outline the contract for various graph traversal algorithms. It includes a single method, execute(), which takes a graph and two vertices (source and destination) and returns a GraphMani.Path.
    c) Concrete Strategy Classes (BFSAlgo and DFSAlgo):
        i) I created two classes, BFSAlgo and DFSAlgo, both implementing the Algo interface. Each class provides a specific implementation for the graph traversal algorithm using BFS and DFS, respectively. They both extend a class named GraphSearchTemplate, suggesting shared functionality.
    d) Context Class (context):
        i) I designed the context class to hold a reference to an Algo object (algointerface). The execute method in the context class delegates the execution to the Algo object, allowing me to switch between different strategies at runtime.
    e) Usage in GraphMani Class:
        i) Inside the GraphMani class, I included an enumeration algo to represent different algorithms, including BFS and DFS.
        ii) The GraphSearch method takes the selected algorithm as a parameter and creates the corresponding Algo object using a switch statement.
        iii) Then, I instantiate the context with the selected Algo object and call the execute method, obtaining the result.
    f) Main Method:
        i) In the main method, I created instances of GraphMani, parsed graphs, and called the GraphSearch method for both BFS and DFS, generating and printing the results.
    g) This implementation using the Strategy Pattern provides flexibility and maintainability, allowing me to easily extend the code with new graph traversal algorithms without modifying the existing structure.
    h) Commits - https://github.com/yuktasarode/CSE4642023_ysarode/commit/00f4e6f34312d9632181340f2c55ecc85c43f8b1

       https://github.com/yuktasarode/CSE4642023_ysarode/commit/ec770acc1a36ba8fcc20bb84820357dfd081a300

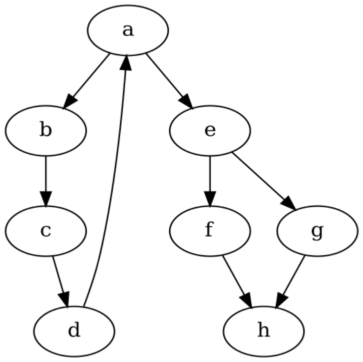
Output for template and strategy pattern:



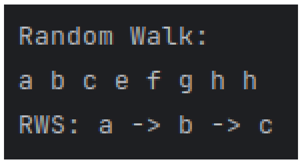
```
================================Part 3================================
BFS: a -> b -> e
DFS: a -> b -> c -> e
```

5)

a) Random Walk on input2.dot:

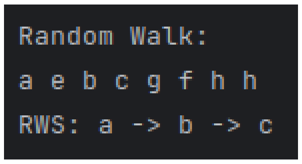


Run 1:



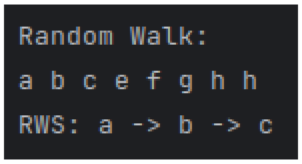
```
Random Walk:
a b c e f g h h
RWS: a -> b -> c
```

Run 2:



```
Random Walk:
a e b c g f h h
RWS: a -> b -> c
```

Run 3:



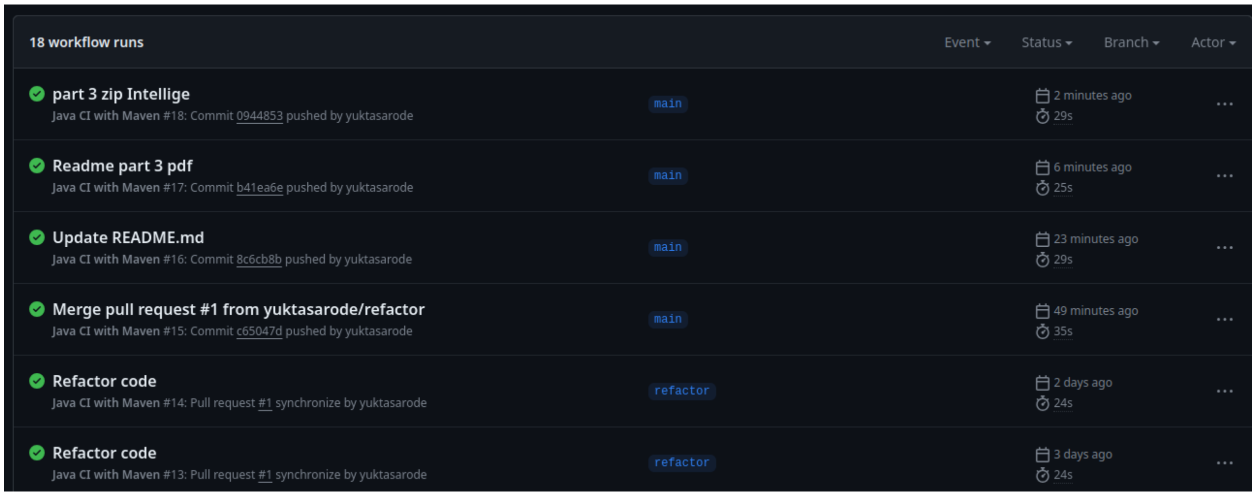
```
Random Walk:
a b c e f g h h
RWS: a -> b -> c
```

Commits -

https://github.com/yuktasarode/CSE4642023_ysarode/commit/630c9ddbca0d589bdbe0f70f7aa075f4e5cf7692

https://github.com/yuktasarode/CSE4642023_ysarode/commit/db00c75b509b5b2a7a295ebc585c5a115af71e78

6) CI/CD



18 workflow runs

part 3 zip Intellige
Java CI with Maven #18: Commit 0944853 pushed by yuktasarode
main
2 minutes ago
29s

Readme part 3 pdf
Java CI with Maven #17: Commit b41ea6e pushed by yuktasarode
main
6 minutes ago
25s

Update README.md
Java CI with Maven #16: Commit 8c6cb8b pushed by yuktasarode
main
23 minutes ago
29s

Merge pull request #1 from yuktasarode/refactor
Java CI with Maven #15: Commit c65047d pushed by yuktasarode
main
49 minutes ago
35s

Refactor code
Java CI with Maven #14: Pull request #1 synchronize by yuktasarode
refactor
2 days ago
24s

Refactor code
Java CI with Maven #13: Pull request #1 synchronize by yuktasarode
refactor
3 days ago
24s