Gabriel Valenzuela

CSE 464 Project Part 3 README

jgvalen2@asu.edu
1212912075
jgvalen2

Project Overview

This document provides instructions and examples for running and testing the GraphParser project. This project involves refactoring code and implementing new features using design patterns for BFS, DFS, and Random Walk graph search algorithms.

GitHub REFACTOR Repository Link: https://github.com/gvalen45/464-Project-2/tree/refactor

GitHub Pull Request done by:

Dattatri, Vikram 1215083103 vdattatr

Link: https://github.com/gvalen45/464-Project-2/pull/5

Refactoring and Design Pattern Implementation

1. **Refactor 1**:
   Link: Refactoring 1:Method Reusability (GraphParser.java)

   Improved efficiency and maintainability by refactoring existing methods for better reusability. This practice ensures that methods serve a single, defined purpose and can be used in multiple contexts without modification.
   Impact: Increases code clarity and reduces redundancy, making future changes and additions easier to implement.

2. **Refactor 2**:
   Link: Refactoring 2: Enhancing Test Clarity/Maintainability

   Enhanced the clarity and maintainability of test scripts. This refactoring involved restructuring tests for readability and ensuring they align well with the features they test.
   Impact: Tests are more understandable and easier to maintain. It simplifies the process of updating tests in response to changes in application code.

   Refactor 3:
   Link: Refactoring 3: Encapsulation and Data Hiding

Gabriel Valenzuela

CSE 464 Project Part 3 README

jgvalen2@asu.edu
1212912075
jgvalen2

Applied encapsulation and data hiding principles, particularly in data structures and their manipulation methods. This involved using private access modifiers and providing public getter methods where necessary.
Impact: Increases the safety and integrity of the data within the application, which is a fundamental principle in object-oriented programming. It also reduces the likelihood of unintended side effects from external access.

3. **Refactor 4**:
   Link: Refactor 4: Remove Dead Code

   Identified and removed unused or redundant pieces of code from the codebase. This process involved careful analysis to ensure that the removed code did not impact the application's functionality.
   Impact: Simplifies the codebase, making it easier to read and maintain. It also helps in reducing potential bugs and improves overall code quality.
4. **Refactor 5**:
   Link: Refactor 5: Removed Unnecessary Dependency

   Analyzed the project's dependencies and removed those that were no longer necessary. This was part of an effort to streamline the project and improve its efficiency.
   Impact: Reduces the project's complexity and dependency footprint. This can lead to improved application performance, including reduced memory usage and faster load times.

Design Patterns

1. **Template Pattern**: Implemented for abstracting common steps in BFS and DFS.
   - **Commit Link:** Template pattern
   - Explanation: The Template pattern is used to define the skeleton of the algorithm in the base class **GraphSearchTemplate**, with specific steps implemented in the subclasses **BFS** and **DFS**.
   - Code Snippet:

```
public abstract class GraphSearchTemplate {
        int var = 1;
        protected Graph<String, DefaultEdge> graph;
        protected String srcLabel, dstLabel;
        protected Map<String, String> prev;
        protected Set<String> visited;
        public GraphSearchTemplate(Graph<String, DefaultEdge> graph) {
        this.graph = graph;
        }
        protected abstract void initializeSearch(String srcLabel, String dstLabel);
        protected abstract String getNextNode();
        protected abstract boolean hasNextNode();
        protected abstract boolean isDestinationReached(String node);
        public GraphParser.Path search(String srcLabel, String dstLabel) {
        initializeSearch(srcLabel, dstLabel);
        while (hasNextNode()) {
        String current = getNextNode();
        if (isDestinationReached(current)) {
                return reconstructPath(prev, dstLabel);
        }
        }
        return null;
        }
        protected GraphParser.Path reconstructPath(Map<String, String> prev, String
dstLabel) {
        …
        …
```

2. **Strategy Pattern**: Applied to choose between BFS and DFS using the **algo** parameter.
    - **Commit Link:** Strategy pattern
    - Explanation: The Strategy pattern allows the selection of the algorithm at runtime. The**GraphSearchStrategy** interface is implemented by **BFS** and **DFS**.
    - Code Snippet:

```
public interface GraphSearchStrategy {
        GraphParser.Path search(String srcLabel, String dstLabel);
}
```

3. **Random Walk Search Algorithm**: Integration of Random Walk using both Strategy and Template patterns.
    - **Commit Link:** Random Walk and random testing
    - Example Graph: An example graph is created with nodes A, B, C, D, E, F and edges forming various paths.

Gabriel Valenzuela

CSE 464 Project Part 3 README

jgvalen2@asu.edu
1212912075
jgvalen2

- Code Snippet:

```
public class DFS extends GraphSearchTemplate implements GraphSearchStrategy{
        private Stack<String> stack;
        public DFS(Graph<String, DefaultEdge> graph) {
        super(graph);
        }
        @Override
        protected void initializeSearch(String srcLabel, String dstLabel) {
        this.srcLabel = srcLabel;
        this.dstLabel = dstLabel;
        prev = new HashMap<>();
          visited = new HashSet<>();
        stack = new Stack<>();
        stack.push(srcLabel);
        visited.add(srcLabel);
        prev.put(srcLabel, null);
        }
        @Override
        protected String getNextNode() {
        return stack.pop();
        }
        @Override
        protected boolean hasNextNode() {
        return !stack.isEmpty();
        }
        @Override
        protected boolean isDestinationReached(String node) {
        return node.equals(dstLabel);
        }
        public Path search(String srcLabel, String dstLabel) {
        initializeSearch(srcLabel, dstLabel);
        while (hasNextNode()) {
        String current = getNextNode();
        if (isDestinationReached(current)) {
                return reconstructPath(prev, dstLabel);
        }
         addNeighborsToStack(current);
        }
        return null;
```

Gabriel Valenzuela

CSE 464 Project Part 3 README

jgvalen2@asu.edu
1212912075
jgvalen2

```
        }
        // Additional method to add neighbors to the stack
        protected void addNeighborsToStack(String current) {
        …
        …
```

Running the Code

Ensure Maven is installed and navigate to the project directory. Compile the code using, the following command:

mvn package

Test Execution

Run the tests using:

mvn test

Running BFS/DFS/Random Walk Search

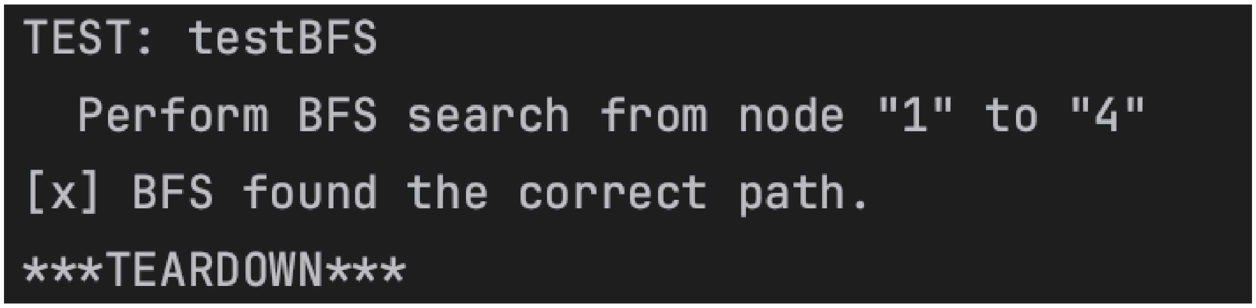
- When you call graphSearch, it instantiates the appropriate search strategy based on the algo parameter.
- It then calls the search method of the instantiated strategy class to perform the actual search.
- For example, to run a BFS search from node "A" to "C", you would use:

```
GraphParser.Path result = graphParser.graphSearch("A", "C",GraphParser.Algorithm.BFS);
```

*Replace **BFS** with **DFS** or **RANDOM_WALK** as needed.*

Expected Outputs

1. **BFS Test Output**:



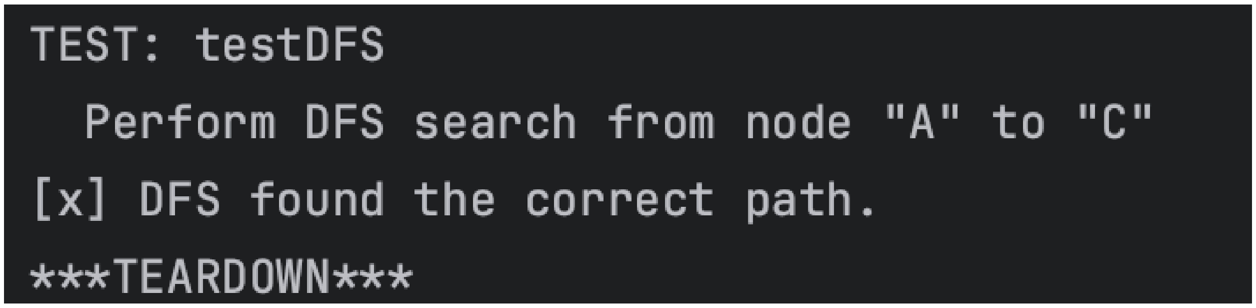
```
TEST: testBFS
  Perform BFS search from node "1" to "4"
[x] BFS found the correct path.
***TEARDOWN***
```

2. **DFS Test Output**:

Gabriel Valenzuela

CSE 464 Project Part 3 README

jgvalen2@asu.edu
1212912075
jgvalen2



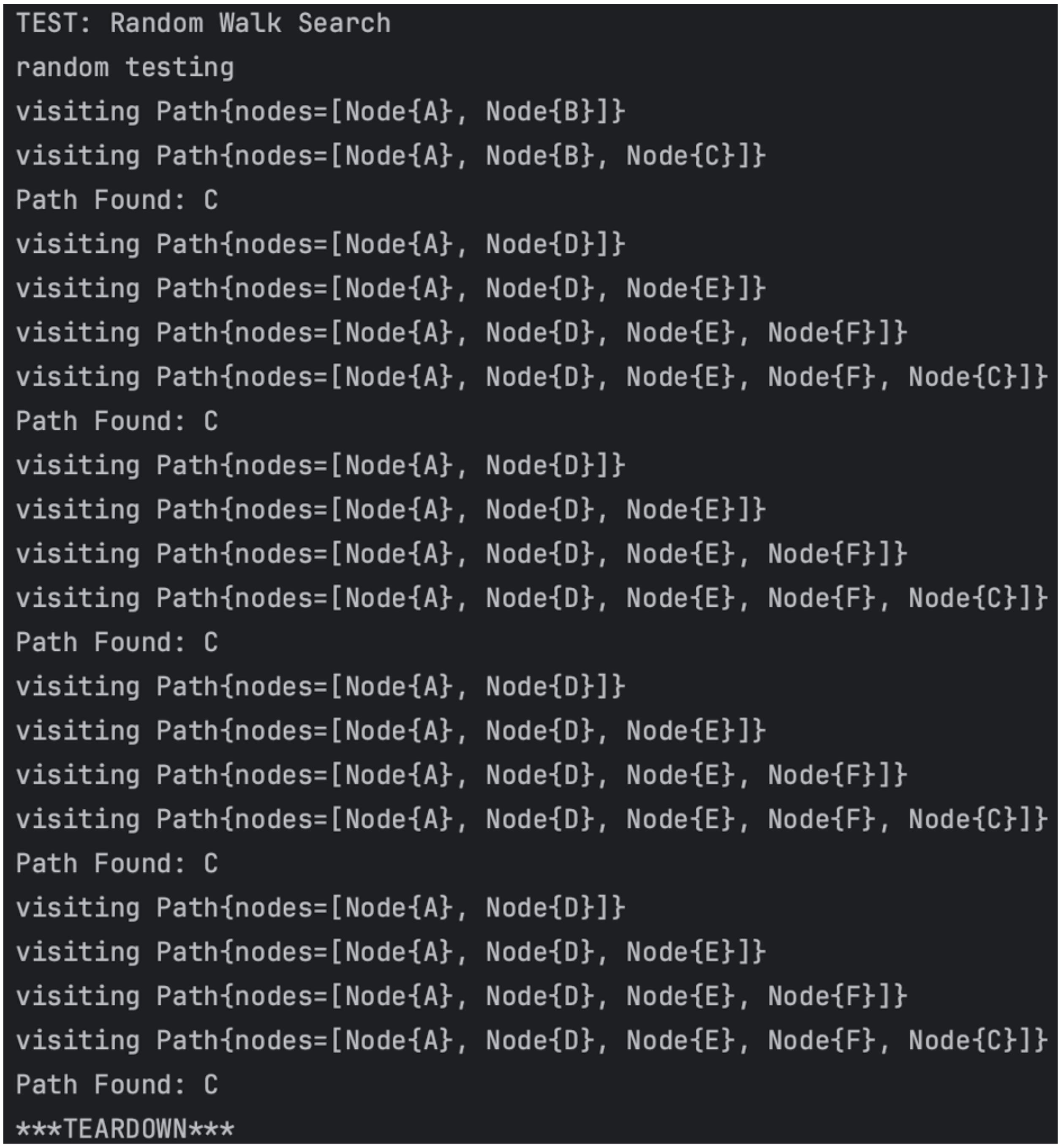
```
TEST: testDFS
  Perform DFS search from node "A" to "C"
[x] DFS found the correct path.
***TEARDOWN***
```

3. **Random Walk Test Output**:

Gabriel Valenzuela

CSE 464 Project Part 3 README

jgvalen2@asu.edu
1212912075
jgvalen2



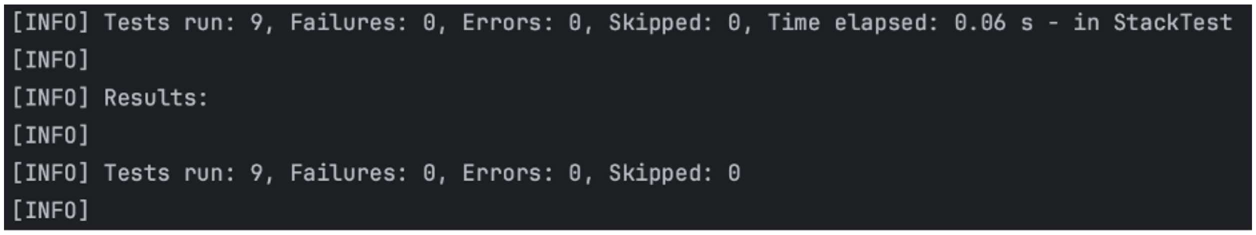
```
TEST: Random Walk Search
random testing
visiting Path{nodes=[Node{A}, Node{B}]}
visiting Path{nodes=[Node{A}, Node{B}, Node{C}]}
Path Found: C
visiting Path{nodes=[Node{A}, Node{D}]}
visiting Path{nodes=[Node{A}, Node{D}, Node{E}]}
visiting Path{nodes=[Node{A}, Node{D}, Node{E}, Node{F}]}
visiting Path{nodes=[Node{A}, Node{D}, Node{E}, Node{F}, Node{C}]}
Path Found: C
visiting Path{nodes=[Node{A}, Node{D}]}
visiting Path{nodes=[Node{A}, Node{D}, Node{E}]}
visiting Path{nodes=[Node{A}, Node{D}, Node{E}, Node{F}]}
visiting Path{nodes=[Node{A}, Node{D}, Node{E}, Node{F}, Node{C}]}
Path Found: C
visiting Path{nodes=[Node{A}, Node{D}]}
visiting Path{nodes=[Node{A}, Node{D}, Node{E}]}
visiting Path{nodes=[Node{A}, Node{D}, Node{E}, Node{F}]}
visiting Path{nodes=[Node{A}, Node{D}, Node{E}, Node{F}, Node{C}]}
Path Found: C
visiting Path{nodes=[Node{A}, Node{D}]}
visiting Path{nodes=[Node{A}, Node{D}, Node{E}]}
visiting Path{nodes=[Node{A}, Node{D}, Node{E}, Node{F}]}
visiting Path{nodes=[Node{A}, Node{D}, Node{E}, Node{F}, Node{C}]}
Path Found: C
***TEARDOWN***
```

4. **Summary of Test Output:**

Gabriel Valenzuela

CSE 464 Project Part 3 README

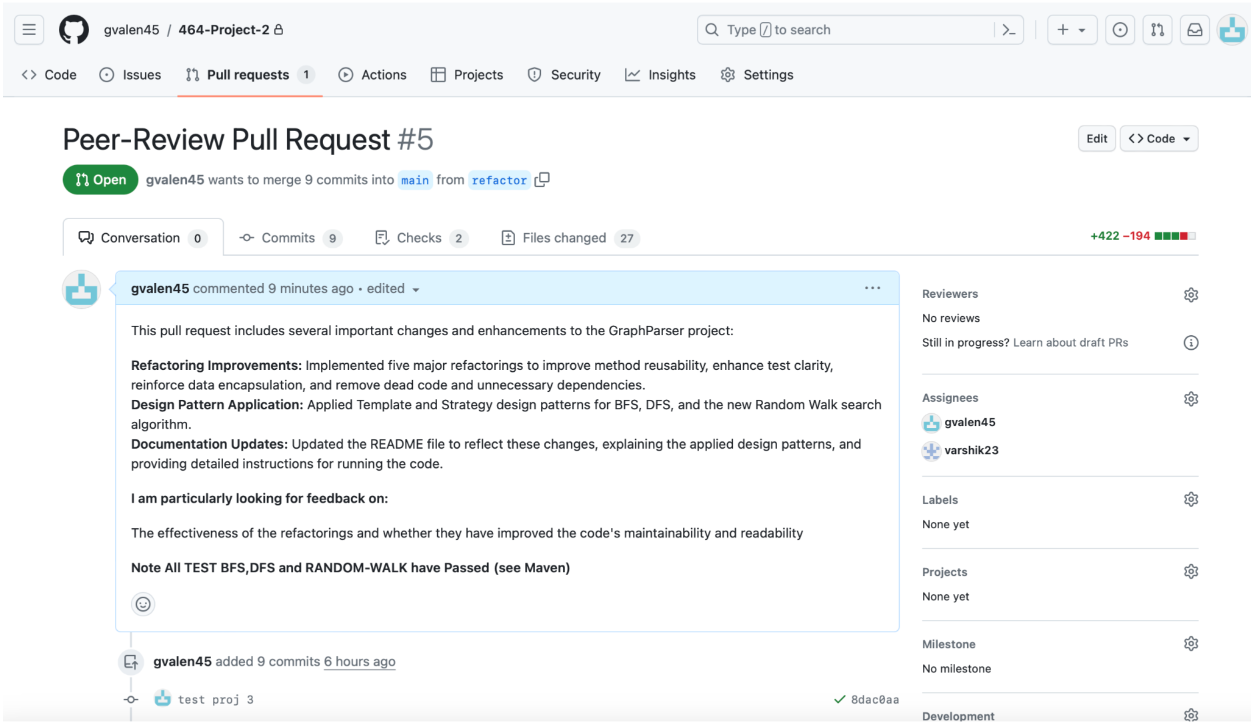
jgvalen2@asu.edu
1212912075
jgvalen2



```
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.06 s - in StackTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0
[INFO]
```

Pull-Request:

LINK: https://github.com/gvalen45/464-Project-2/pull/5



Peer-Review Pull Request #5

gvalen45 wants to merge 9 commits into `main` from `refactor`

gvalen45 commented 9 minutes ago • edited

This pull request includes several important changes and enhancements to the GraphParser project:

Refactoring Improvements: Implemented five major refactorings to improve method reusability, enhance test clarity, reinforce data encapsulation, and remove dead code and unnecessary dependencies.
Design Pattern Application: Applied Template and Strategy design patterns for BFS, DFS, and the new Random Walk search algorithm.
Documentation Updates: Updated the README file to reflect these changes, explaining the applied design patterns, and providing detailed instructions for running the code.

I am particularly looking for feedback on:

The effectiveness of the refactorings and whether they have improved the code's maintainability and readability

Note All TEST BFS,DFS and RANDOM-WALK have Passed (see Maven)

gvalen45 added 9 commits 6 hours ago

`test proj 3` ✓ `8dac0aa`

Reviewers
No reviews
Still in progress? Learn about draft PRs

Assignees
gvalen45
varshik23

Labels
None yet

Projects
None yet

Milestone
No milestone

Development