Graph Parser Project Phase 3 – README

Project Overview

This project implements a graph parser program hat processes a DOT file and generates a visual depiction in addition to other features such as modifying nodes, modifying edges, performing BFS and DFS on the graph. Furthermore, this phase contains five individual refactors, applies the template and strategy pattern design, and implements a Random Walk Search algorithm. This project also includes a JUnit test feature to ensure the program passes the expected test cases.

GitHub Link

https://github.com/anishnallani22/CSE-464-2024-anallani

Instructions to Run the Project

1. Clone the repository

```
git clone https://github.com/anishnallani22/CSE-464-2024-anallani.git
```

2. Navigate into project directory

```
cd CSE-464-2024-anallani
```

3. Build the project with Maven

```
mvn package
```

4. Run the Program

```
java -jar target/CSE464-GraphManager-1.0-SNAPSHOT.jar src/main/resources/input.dot
```

5. Run Unit Tests

```
mvn test
```

Input DOT Graph

```
digraph {
   a -> b;
   b -> c;
   c -> d;
   d -> a;
   a -> e;
   e -> f;
   e -> g;
   f -> h;
   g -> h;
}
```

Refactor Description

1. Refactor 1: Extract Method – checkNodeExists in addEdges.java

Summary: Applied the extract method refactoring to improve code readability and reusability in the addEdge method by encapsulating repeated logic for verifying node existence into a new helper method, checkNodeExists. The original addEdge method contained repeated logic for verifying node existence, making the code redundant and harder to maintain. The changes aim to reduce duplication, and make the logic modular.

Commit Link: https://github.com/anishnallani22/CSE-464-2024-anallani/commit/4b6eb0d2fe06aadf6334b356839069f1a605be13

2. Refactor 2: Extract Variable – Neighbor Determination in BFS and DFS methods
Summary: Applied the Extract Variable refactoring to simplify complex expressions in the graphSearchBFS and graphSearchDFS methods by encapsulating the neighbor determination logic into a variable, neighbor. The original methods contained inline logic for determining neighbors, which was repeated and reduced readability, so the change reduced duplication and clarified the logic loop.

Commit Link: https://github.com/anishnallani22/CSE-464-2024-anallani/commit/f1d6f48e6f50cfb883e936edd351b1cc6601dae1

3. Refactor 3: Consolidate Functionality – printGraph methods into GraphPrintUtility
Summary: Refactored the printGraph methods from multiple classes into a single utility method, GraphPrintingUtility.printGraph, to consolidate functionality and reduce duplication. This change improves maintainability by centralizing graph-printing logic in one place and enhances readability by removing repetitive code across the project.

Commit Link: https://github.com/anishnallani22/CSE-464-2024-anallani/commit/e878b5149a3f69e439387554172310e500039fc8

4. Refactor 4: Rename Method – removeGraphNode to removeNode in removeNode.java
Summary: Renamed the method removeGraphNode to removeNode in removeNode.java to align with the naming conventions already present in the codebase. This change improves code consistency and readability by making method names more uniform across the project.

Commit Link: https://github.com/anishnallani22/CSE-464-2024-anallani/commit/196470ed5119d0a584b308caa43196374409b25e

5. Refactor 5: Extract Method – Edge Writing logic in outputDOTGraph in outputGraph.java
Summary: Refactored the outputDOTGaph method in outputGraph.java by extracting the edge-writing logic into a new helper method, writeEdges. This change simplifies the outputDOTGraph method and improves maintainability by centralizing the edge-writing logic, making the code easier to modify.

Commit Link: https://github.com/anishnallani22/CSE-464-2024-anallani/commit/338a72626f31e3e07cadd1efb75e2ce6da328408

Template Pattern Design Description
Description: The template method pattern was applied to refactor the BFS and DFS algorithms by extracting their shared behaviors into a base class, GraphSearchTemplate. This base class encapsulates common steps such as initializing collections, processing nodes, and reconstructing paths using a parentMap to track the parent-child relationships. The search method in the GraphSearchTemplate class serves as a template for traversal, handling high-level logic while delegating algorithm-specific details to abstract methods implemented in subclasses. For instance, the buildPath method in the base class ensures consistent path reconstruction for both algorithms, while methods such as initializeCollection, isCollectionEmpty, getNextNode, and addToCollection are customized in subclasses to define the behavior of BFS and DFS. For example, the BFSTemplate subclass uses a queue to maintain FIFO order, while the DFSTemplate uses a stack for LIFO order and relies on the pop technique for node retrieval.

Commit Link: https://github.com/anishnallani22/CSE-464-2024-anallani/commit/3807483dc8264f6082cfeb1cf7323c56cdf2341f

Strategy Pattern Design Description
Description: The strategy pattern was applied to enable dynamic selection of graph traversal algorithms, allowing the system to switch between BFS and DFS at runtime based on the algo parameter in the GraphSearch method of the Path class. A GraphStrategyPattern interface was introduced to define a common contract for all traversal strategies, including methods like getGraph() and search(). Concrete classes, BFSTemplate and DFSTemplate, implement this interface and encapsulate their respective traversal behaviors, with BFS using a queue for level-by-level exploration and DFS using a stack for depth-first traversal. The GraphSearch method is responsible for determining the appropriate strategy based on the algo parameter and passing it to the context. For example, if Algorithm.BFS is provided, a BFSTemplate object is created and used for search. This ensures modularity and that the graph traversal implementation can be dynamically switched.

BFS and DFS Sample Output:

```
=== Performing BFS Search ===
Path found using BFS: Path{nodes=[a, b, c, d, l]}

=== Performing DFS Search ===
Path found using DFS: Path{nodes=[a, b, c, d, l]}
```

Commit Links:
https://github.com/anishnallani22/CSE-464-2024-anallani/commit/45f5beff5b32eabbe3db7a26beec60fa7597d6b0
https://github.com/anishnallani22/CSE-464-2024-anallani/commit/ce621856195d7f5859f533fb63729bacc4b04187
https://github.com/anishnallani22/CSE-464-2024-anallani/commit/bdd7f0efe062a1618859926ada8119eb0d441abb

Random Walk Search Description
Description: The Random Walk Search algorithm performs traversal by randomly selecting a neighbor of the current node at each step and continues until the destination node is reached or no more unvisited neighbors remain.

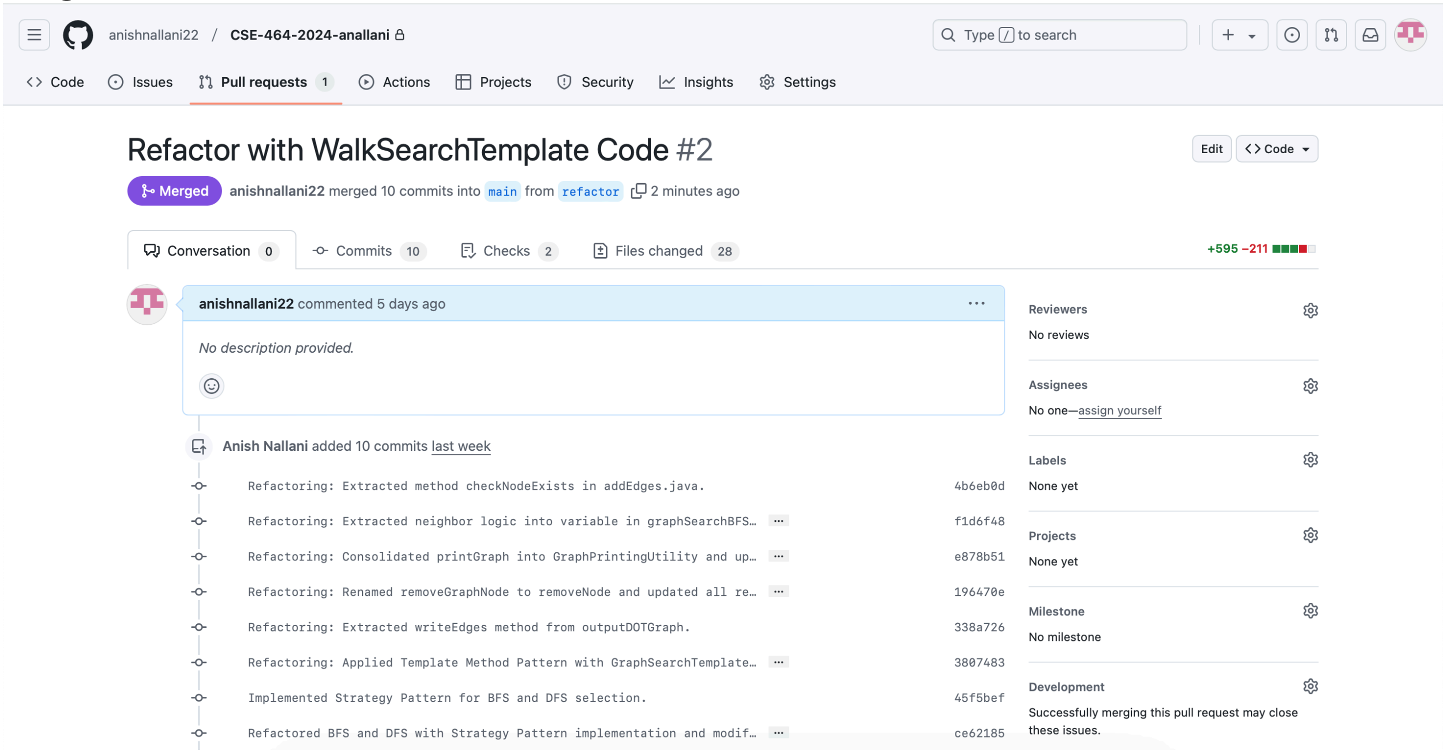
Sample Output:

```
=== Performing Random Walk Search ===

random testing
visiting Path{nodes=[Node{a}]}
visiting Path{nodes=[Node{a}, Node{b}]}
visiting Path{nodes=[Node{a}, Node{b}, Node{j}]}
No more unvisited neighbors. Random walk failed.

random testing
visiting Path{nodes=[Node{a}]}
visiting Path{nodes=[Node{a}, Node{b}]}
visiting Path{nodes=[Node{a}, Node{b}, Node{c}]}
Path{nodes=[Node{a}, Node{b}, Node{c}]}
```

```
Path{nodes=[a, b, c]}

random testing
visiting Path{nodes=[Node{a}]}
visiting Path{nodes=[Node{a}, Node{e}]}
visiting Path{nodes=[Node{a}, Node{e}, Node{f}]}
visiting Path{nodes=[Node{a}, Node{e}, Node{f}, Node{h}]}
No more unvisited neighbors. Random walk failed.

random testing
visiting Path{nodes=[Node{a}]}
visiting Path{nodes=[Node{a}, Node{e}]}
visiting Path{nodes=[Node{a}, Node{e}, Node{g}]}
visiting Path{nodes=[Node{a}, Node{e}, Node{g}, Node{h}]}
No more unvisited neighbors. Random walk failed.

random testing
visiting Path{nodes=[Node{a}]}
visiting Path{nodes=[Node{a}, Node{b}]}
visiting Path{nodes=[Node{a}, Node{b}, Node{j}]}
No more unvisited neighbors. Random walk failed.

random testing
visiting Path{nodes=[Node{a}]}
visiting Path{nodes=[Node{a}, Node{b}]}
visiting Path{nodes=[Node{a}, Node{b}, Node{j}]}
No more unvisited neighbors. Random walk failed.

random testing
visiting Path{nodes=[Node{a}]}
visiting Path{nodes=[Node{a}, Node{b}]}
visiting Path{nodes=[Node{a}, Node{b}, Node{j}]}
No more unvisited neighbors. Random walk failed.

random testing
visiting Path{nodes=[Node{a}]}
visiting Path{nodes=[Node{a}, Node{b}]}
visiting Path{nodes=[Node{a}, Node{b}, Node{j}]}
No more unvisited neighbors. Random walk failed.

random testing
visiting Path{nodes=[Node{a}]}
visiting Path{nodes=[Node{a}, Node{b}]}
visiting Path{nodes=[Node{a}, Node{b}, Node{c}]}
Path{nodes=[Node{a}, Node{b}, Node{c}]}
Path{nodes=[a, b, c]}

random testing
visiting Path{nodes=[Node{a}]}
visiting Path{nodes=[Node{a}, Node{i}]}
No more unvisited neighbors. Random walk failed.
```

Commit Link: https://github.com/anishnallani22/CSE-464-2024-anallani/commit/51769f0b36eb03cd664d2d94b64367a6f92143d9

Merge refactor into main


anishnallani22 / CSE-464-2024-anallani

Code | Issues | Pull requests 1 | Actions | Projects | Security | Insights | Settings

Refactor with WalkSearchTemplate Code #2

Merged anishnallani22 merged 10 commits into `main` from `refactor` 2 minutes ago

Conversation 0 | Commits 10 | Checks 2 | Files changed 28 | +595 −211

anishnallani22 commented 5 days ago

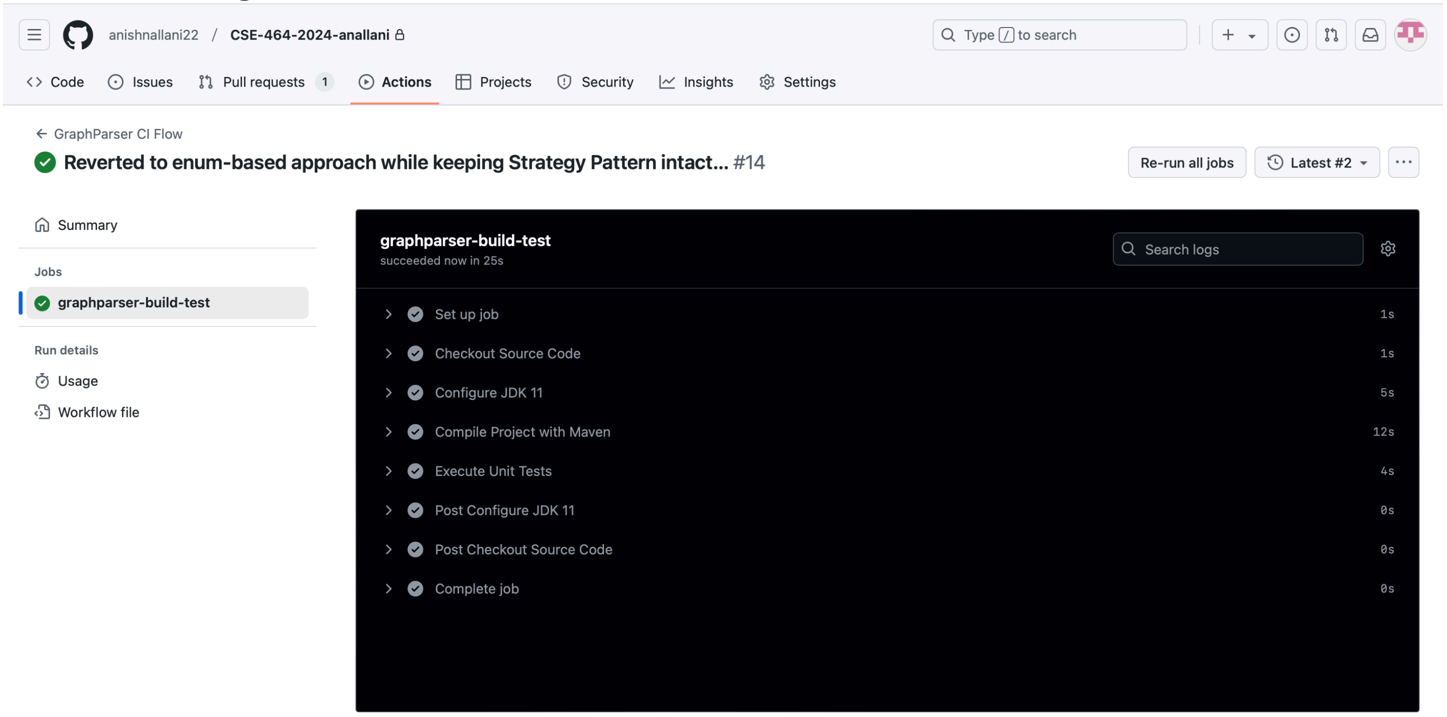
No description provided.

Anish Nallani added 10 commits last week

```
Refactoring: Extracted method checkNodeExists in addEdges.java.                    4b6eb0d
Refactoring: Extracted neighbor logic into variable in graphSearchBFS…             f1d6f48
Refactoring: Consolidated printGraph into GraphPrintingUtility and up…             e878b51
Refactoring: Renamed removeGraphNode to removeNode and updated all re…             196470e
Refactoring: Extracted writeEdges method from outputDOTGraph.                      338a726
Refactoring: Applied Template Method Pattern with GraphSearchTemplate…             3807483
Implemented Strategy Pattern for BFS and DFS selection.                            45f5bef
Refactored BFS and DFS with Strategy Pattern implementation and modif…             ce62185
```

Reviewers: No reviews
Assignees: No one—assign yourself
Labels: None yet
Projects: None yet
Milestone: No milestone
Development: Successfully merging this pull request may close these issues.

Continuous Integration Workflow


anishnallani22 / CSE-464-2024-anallani

Code | Issues | Pull requests 1 | Actions | Projects | Security | Insights | Settings

GraphParser CI Flow

Reverted to enum-based approach while keeping Strategy Pattern intact... #14

Summary
Jobs: graphparser-build-test
Run details: Usage, Workflow file

graphparser-build-test
succeeded now in 25s

```
Set up job                    1s
Checkout Source Code          1s
Configure JDK 11              5s
Compile Project with Maven   12s
Execute Unit Tests            4s
Post Configure JDK 11         0s
Post Checkout Source Code     0s
Complete job                  0s
```