Course Project Part 2

Gabriel Valenzuela Jgvalen2

Building the Project

To compile the code using Maven, run the following command:

mvn package

The class GraphParser is the main code and runs to confirm the working of features 1-4.

My tests are in the StackTest.

Run Project with Maven, install -> clean -> package

Table of Contents

Feature 1: testRemoveNodeSuccessfully

Verifies that a node can be successfully removed from the graph.
Checks that the total node count decreases by one after removal.
Ensures that the method returns true when a node is successfully removed.

Junit Test

```
 @Test
public void testRemoveNodeSuccessfully() {
    System.out.println("TEST: removal of a node that exists.");
    int initialNodeCount = graphParser.getNumNodes();
    assertTrue(graphParser.removeNode("B"));
    assertEquals(initialNodeCount - 1, graphParser.getNumNodes());
    System.out.println("[x]Node B removed Correctly.");
}
```

Junit Expected Test

EXAMPLE:

```
***Setup done***
TEST: testRemoveNodesSuccessfully
 removal of multiple nodes that exist.
[x]Nodes A and C removed successfully.
***TEARDOWN***
```

Feature 2: testRemoveNodesSuccessfully (Multiple Nodes)

Tests the removal of multiple nodes from the graph at once.

Confirms that the node count is correctly reduced by the number of nodes removed.

Validates that the nodes are indeed removed from the graph.

Junit Test

```java
@Test
public void testRemoveNodesSuccessfully() {
    System.out.println("TEST: testRemoveNodesSuccessfully \n removal of
multiple nodes that exist.");
    int initialNodeCount = graphParser.getNumNodes();
    graphParser.removeNodes(new String[]{"A", "C"});
    assertEquals(initialNodeCount - 2, graphParser.getNumNodes());
    System.out.println("[x]Nodes A and C removed successfully.");
}
```

Junit Expected Test

EXAMPLE:

```
***TEARDOWN***

***Setup done***
TEST: testRemoveNodesSuccessfully
 removal of multiple nodes that exist.
[x]Nodes A and C removed successfully.
***TEARDOWN***
```

Feature 3: testRemoveEdgeSuccessfully

Ensures that an edge can be successfully removed from the graph.

Checks that the total edge count decreases by one after removal.

Asserts that the method returns true when an edge is successfully removed.

Junit Test

```java
@Test
public void testRemoveEdgeSuccessfully() {
    System.out.println("TEST: testRemoveEdgeSuccessfully \n removal of an
edge that exists.");
    int initialEdgeCount = graphParser.getNumEdges();
    assertTrue(graphParser.removeEdge("A", "B"));
    assertEquals(initialEdgeCount - 1, graphParser.getNumEdges());
    System.out.println("Edge A->B removed successfully.");
}
```

Junit Expected Test

EXAMPLE:

```
***Setup done***
TEST: testRemoveEdgeSuccessfully
 removal of an edge that exists.
Edge A->B removed successfully.
***TEARDOWN***
```

Feature 4: testRemoveNonExistentEdge

Tests the graph's response to the removal of a non-existent edge.

Confirms that the edge count remains unchanged.

Validates that the method returns false for a non-existent edge removal attempt.

Junit Test

```java
@Test
public void testRemoveNonExistentEdge() {
    System.out.println("TEST: testRemoveNonExistentEdge \n  removal of an
edge is DNE");
    int initialEdgeCount = graphParser.getNumEdges();
    assertFalse(graphParser.removeEdge("A", "Z"));
```

```
    assertEquals(initialEdgeCount, graphParser.getNumEdges());
    System.out.println("[x]Correctly identified that edge A->Z DNE.");
}
```

Junit Expected Test

EXAMPLE

```
***Setup done***
TEST: testRemoveNonExistentEdge
  removal of an edge is DNE
Edge from: A to Z does not exist.
[x]Correctly identified that edge A->Z DNE.
***TEARDOWN***
```

Feature 5: testRemoveNonExistentNode

Tests the behavior of the graph when attempting to remove a node that does not exist.

Verifies that the node count remains unchanged.

Confirms that the method returns false when trying to remove a non-existent node.

Junit Test

```
@Test
public void testRemoveNonExistentNode() {
    System.out.println("TEST: testRemoveNonExistentNode \n removal of a node
that DNE.");
    int initialNodeCount = graphParser.getNumNodes();
```

```
    assertFalse(graphParser.removeNode("Z"));
    assertEquals(initialNodeCount, graphParser.getNumNodes());
    System.out.println("[x]Correctly identified that node Z DNE.");
}
```

Junit Expected Test

EXAMPLE

```
***Setup done***
TEST: testRemoveNonExistentNode
 removal of a node that DNE.
-Node Z does not exist. Cannot remove node.
[x]Correctly identified that node Z DNE.
***TEARDOWN***
```

Feature 6: testRemoveNonExistentNodes (multiple)

Checks the removal operation on multiple non-existent nodes.

Ensures that the node count is not affected when attempting to remove nodes that are not present in the graph.

Junit Test

```
@Test
public void testRemoveNonExistentNodes() {
    System.out.println("TEST: testRemoveNonExistentNodes \n removal of
multiple nodes that DNE");
    int initialNodeCount = graphParser.getNumNodes();
    graphParser.removeNodes(new String[]{"X", "Y"});
    assertEquals(initialNodeCount, graphParser.getNumNodes());
    System.out.println("[x]Correctly identified that nodes X and Y DNE.");
}
```

Junit Expected Test

EXAMPLE

```
***Setup done***
TEST: testRemoveNonExistentNodes
 removal of multiple nodes that DNE
-Node X does not exist. Cannot remove node.
-Node Y does not exist. Cannot remove node.
[x]Correctly identified that nodes X and Y DNE.
***TEARDOWN***
```

Feature 7: testBFS

Performs a Breadth-First Search (BFS) from one specified node to another.

Verifies that the BFS algorithm returns the correct path between the two nodes.

Checks that the path returned is not null and matches the expected sequence of nodes.

Junit Test

```
@Test
public void testBFS() {
    System.out.println("TEST: testBFS \n  Perform BFS search from node \"1\"
to \"4\"");

    //  the graph with nodes and edges
    graphParser.addNode("1");
    graphParser.addNode("2");
    graphParser.addNode("3");
    graphParser.addNode("4");
    graphParser.addEdge("1", "2");
    graphParser.addEdge("2", "3");
    graphParser.addEdge("3", "4");
```

```
    // Perform BFS search from node "1" to "4"
    GraphParser.Path result = graphParser.graphSearchBFS("1", "4");

    // Verify the path is as expected
    List<String> expectedPath = Arrays.asList("1", "2", "3", "4");
    assertNotNull(result);
    assertEquals(expectedPath, result.getNodes());
    System.out.println();
}
```

Junit Expected Test

EXAMPLE

```
[INFO] Running StackTest
***Setup done***
TEST: testBFS
  Perform BFS search from node "1" to "4"
Visiting Node: 1
Visiting Node: 2
Visiting Node: 3
Visiting Node: 4
Path Found: 1 -> 2 -> 3 -> 4
```

Feature 8: testDFS

Executes a Depth-First Search (DFS) recursively from one node to another.
Ensures that the DFS algorithm finds the correct path between the nodes.
Validates that the path returned by DFS is as expected and that the method does not return null.

Junit Test

```java
@Test
public void testDFS() {
    System.out.println("TEST: testDFS \n  Perform recursive DFS search from node \"A\" to \"C\"");
    System.out.println("Graph before DFS:");
    System.out.println(graphParser.toString()); // Print the graph before performing DFS

    // Perform recursive DFS search from node "A" to "C"
    GraphParser.Path result = graphParser.graphSearchDFSRecursive("A", "C");

    // Verify the path is as expected
    List<String> expectedPath = Arrays.asList("A", "B", "C"); // This is just an example
    assertNotNull(result);
    assertEquals(expectedPath, result.getNodes());
    System.out.println("[x]Recursive DFS found the correct path.");

    System.out.println("Graph after DFS:");
    System.out.println(graphParser.toString());
}
```

Junit Expected Test

EXAMPLE

```
***Setup done***
TEST: testDFSS
  Perform  DFS search from node "A" to "C"
Graph before DFS:
Graph:
Num of Nodes: 3
Labels of Nodes:
A
B
C
Num of Edges: 2
Nodes and Edge Directions:
A -> B
B -> C

[x] DFS found the correct path.
Graph after DFS:
Graph:
Num of Nodes: 3
Labels of Nodes:
A
B
C
Num of Edges: 2
Nodes and Edge Directions:
A -> B
B -> C

***TEARDOWN***
```

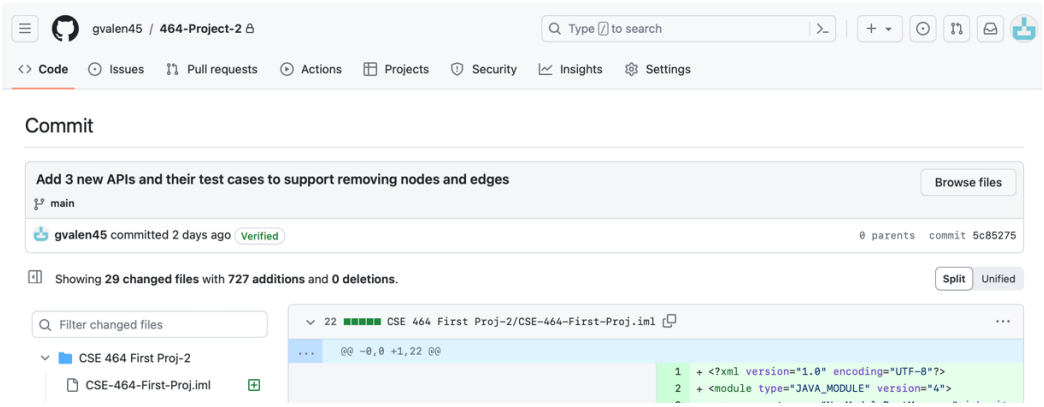
Contribution History Github/Continuous Integration Setup

https://github.com/gvalen45/464-Project-2

Feature Commits

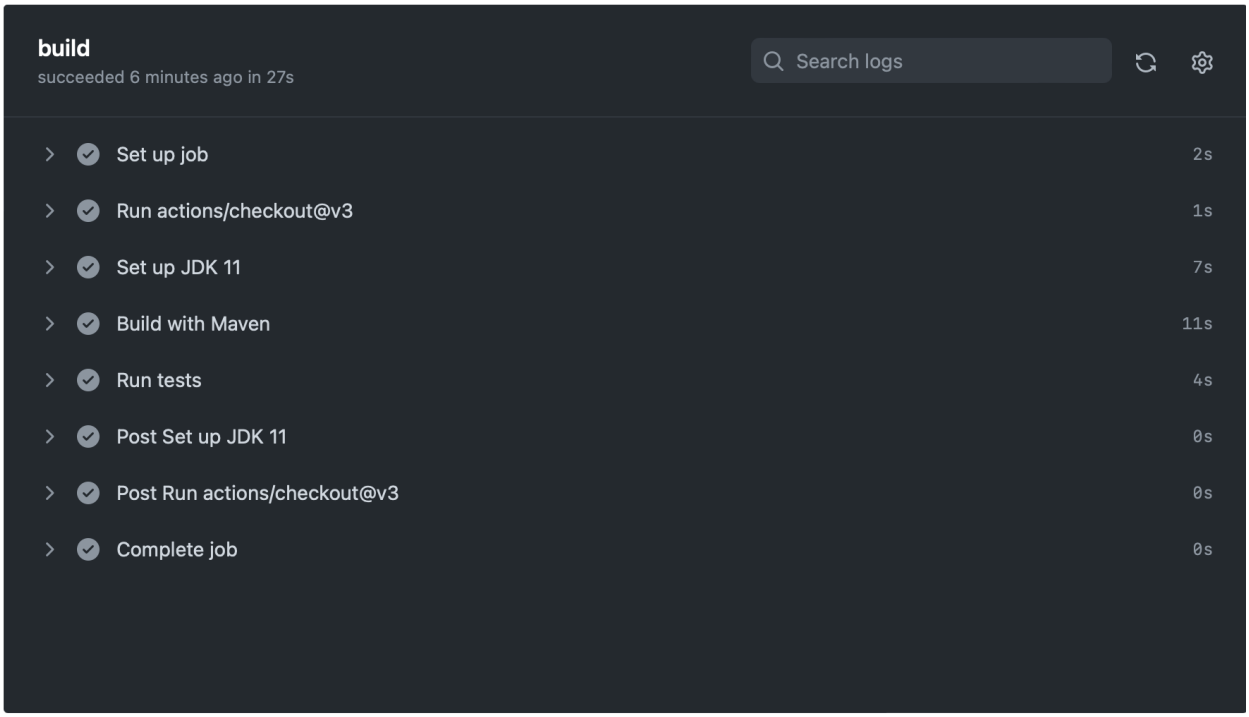
Project Part 1: Add 3 new APIs and their test cases to support removing nodes and edges

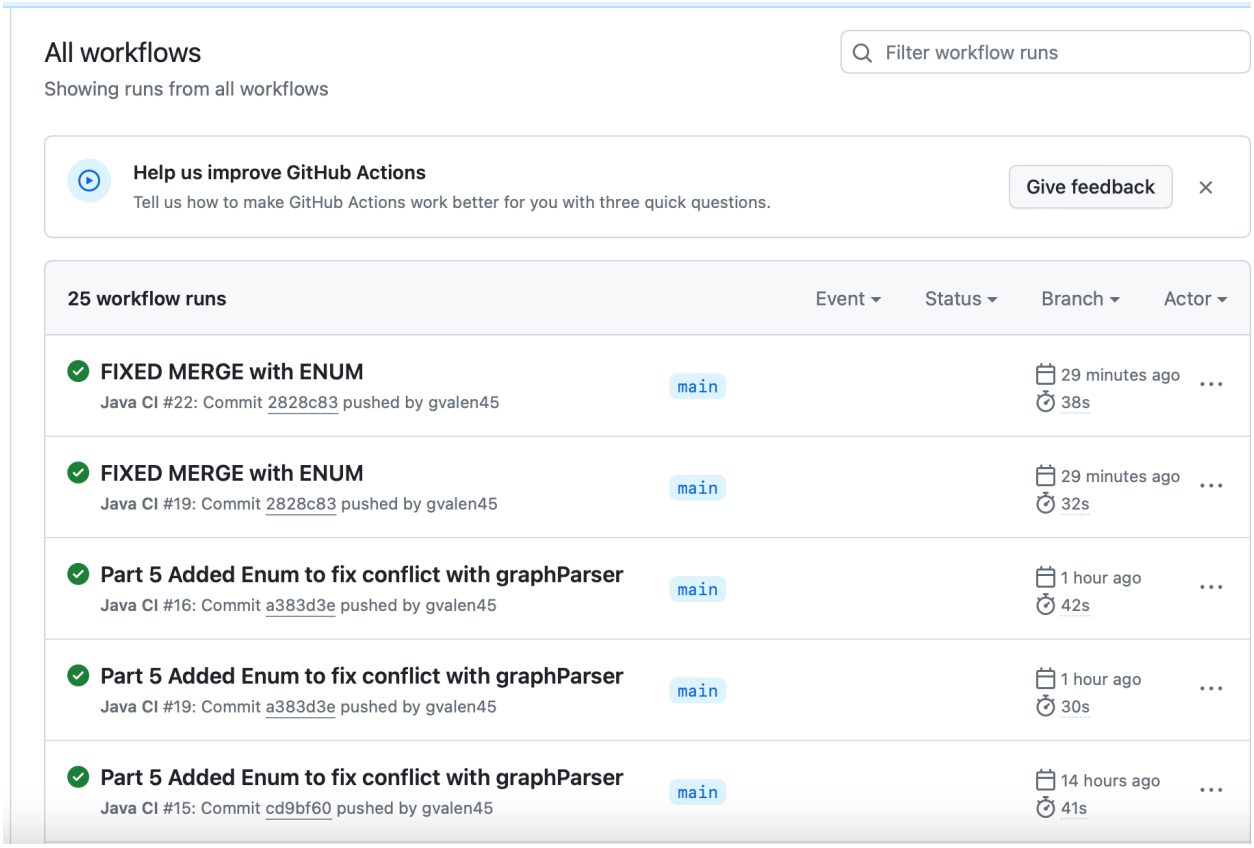
https://github.com/gvalen45/464-Project-2/commit/5c852752c8da3b762dd685f8b4df658ac86a9469



Project Part 2: Add continuous integration support for your github project:

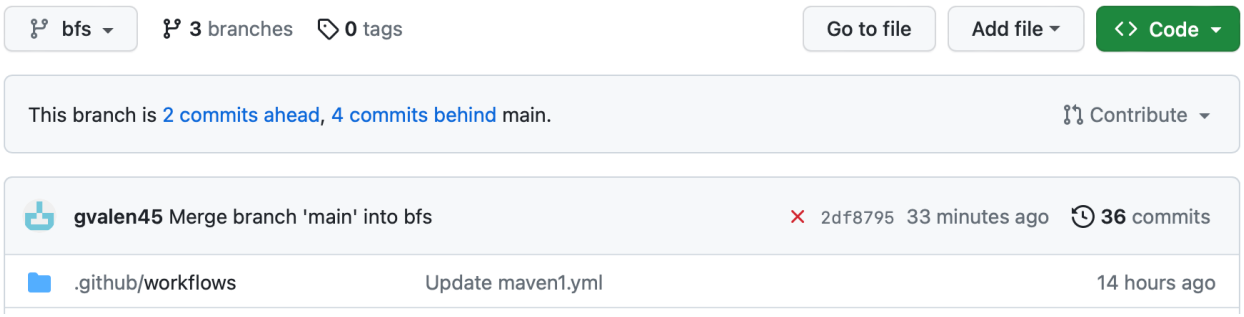
https://github.com/gvalen45/464-Project-2/actions/runs/6763603737/job/18380917854





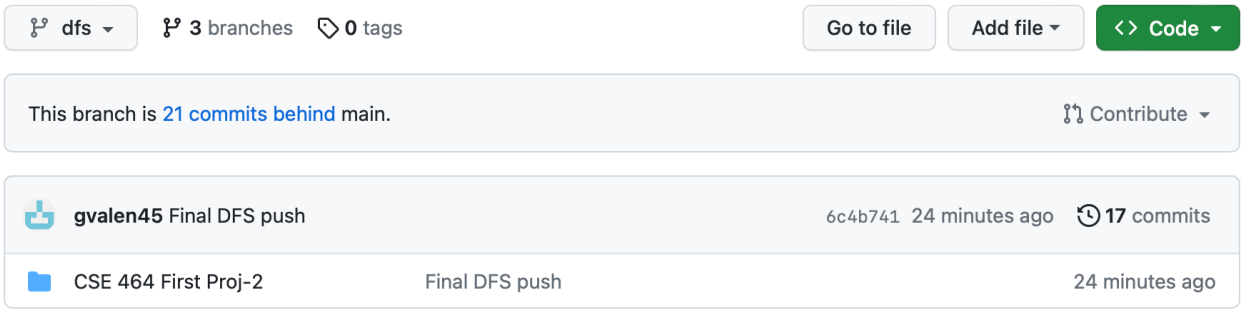
Project Part 3: Create a branch bfs

https://github.com/gvalen45/464-Project-2/tree/bfs

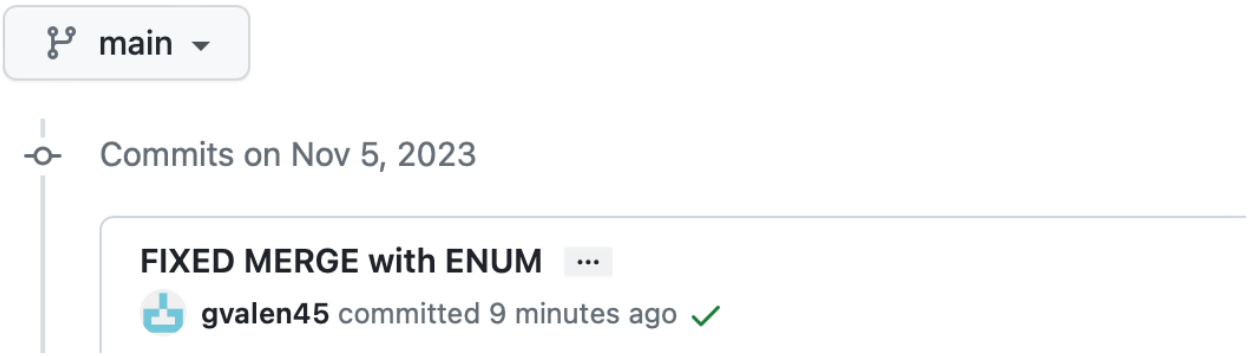


Project Part 4: Create another branch dfs, and implement the same graph search API using DFS algorithm

https://github.com/gvalen45/464-Project-2/tree/dfs



Project Part 5: Merge the bfs branch to the main branch, and then merge the dfs branch to the main branch



Branches and Successful Merges

Branch Name: main

Successful Merge Commit to main ENUM:

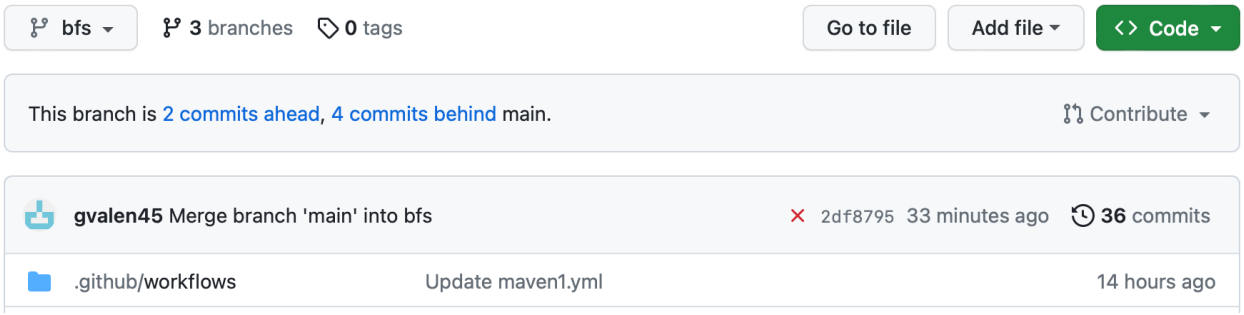
https://github.com/gvalen45/464-Project-2/pull/4

Branch Name: bfs

https://github.com/gvalen45/464-Project-2/tree/bfs

Successful Merge Commit to main:

https://github.com/gvalen45/464-Project-2/commit/4ca509d94b4200c47bf0c4047d6172d81bc37283



Branch Name: dfs

Successful Merge Commit to main:

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