test.cpp

```
// Name: Blake McGahee
// UFID: 82924917
// Project 2 Testing Check-In

// I'm using Catch2 for unit testing. This macro tells Catch2 to provide the main()
function for my test executable.
// This was important to avoid conflicts with my own main() function in main.cpp.
#define CATCH_CONFIG_MAIN
#include <catch2/catch_test_macros.hpp> // Using Catch2 v3 headers. This was a fix
for C1083 errors.
#include <sstream>        // Needed for std::stringstream to capture output.
#include <iostream>       // Needed for std::cout (for redirection, and occasional
debug prints).

// Including AdjacencyList.h to make my AdjacencyList class available for tests.
#include "../src/AdjacencyList.h"
// I now include program.h to access the runProgram function.
// This was a key fix after encountering "unresolved external symbol" errors
(LNK2019) for runProgram,
// and also helped resolve issues where tests weren't showing or were hanging due
to main() conflicts.
#include "../src/program.h"

// The runProgram function is now declared in program.h, so I don't need a forward
declaration here.

// This helper function allows me to run my core program logic (runProgram) with
custom input
// and capture its output into a string, which I can then compare against expected
results.
// This was crucial for testing functions that print to cout.
std::string runProgramWithInput(const std::string& input) {
    std::stringstream ssInput(input);
    std::stringstream ssOutput;

    // I call the core program logic here.
    // Debug prints were temporarily added to runProgram and test cases to diagnose
hangs,
    // but have since been removed or moved to be captured by ssOutput to avoid
interfering with Test Explorer.
    runProgram(ssInput, ssOutput);

    return ssOutput.str();
```

```
}

// Wrapping all tests in a namespace to provide a more descriptive grouping in Test
Explorer.
namespace PageRankTests {

    // --- My Original 5 Distinct Tests (Expected to Pass with Dummy Output) ---
    // These tests are designed to pass with the dummy PageRank output I have
implemented.

    TEST_CASE("Test 1: Basic Graph Example - 2 Iterations", "[PageRank][Basic]") {
        // This test uses the example graph provided in the assignment.
        // It verifies that my program can correctly process a standard input
format
        // and produce the expected dummy output for a basic graph with 2 power
iterations.
        std::string input = R"(7 2
google.com gmail.com
google.com maps.com
facebook.com ufl.edu
ufl.edu google.com
ufl.edu gmail.com
maps.com facebook.com
gmail.com maps.com)";
        std::string expectedOutput = R"(facebook.com 0.20
gmail.com 0.20
google.com 0.10
maps.com 0.30
ufl.edu 0.20
)";
        std::string actualOutput = runProgramWithInput(input);
        REQUIRE(actualOutput == expectedOutput);
    }

    TEST_CASE("Test 2: Single Node Graph - 5 Iterations", "[PageRank][EdgeCase]") {
        // This test checks an edge case: a graph with only one node that links to
itself.
        // It ensures my program handles minimal graph structures and a higher
number of iterations gracefully,
        // even with the dummy PageRank output.
        std::string input = R"(1 5
singlepage.com singlepage.com)";
        std::string expectedOutput = R"(facebook.com 0.20
gmail.com 0.20
google.com 0.10
maps.com 0.30
ufl.edu 0.20
```

```
)";
        std::string actualOutput = runProgramWithInput(input);
        REQUIRE(actualOutput == expectedOutput);
    }

    TEST_CASE("Test 3: Empty Graph - 3 Iterations", "[PageRank][EdgeCase]") {
        // This test covers another edge case: an empty graph (zero lines of
edges).
        // It confirms that my program can handle no input edges without crashing
        // and still produces the default dummy output.
        std::string input = R"(0 3)";
        std::string expectedOutput = R"(facebook.com 0.20
gmail.com 0.20
google.com 0.10
maps.com 0.30
ufl.edu 0.20
)";
        std::string actualOutput = runProgramWithInput(input);
        REQUIRE(actualOutput == expectedOutput);
    }

    TEST_CASE("Test 4: Disconnected Graph - 1 Iteration", "[PageRank][Complex]") {
        // This test examines a graph with multiple disconnected components (e.g.,
A->B and X->Y).
        // It helps verify that my program correctly processes independent graph
segments,
        // even if the PageRank calculation is currently a dummy.
        std::string input = R"(2 1
PageA PageB
PageX PageY)";
        std::string expectedOutput = R"(facebook.com 0.20
gmail.com 0.20
google.com 0.10
maps.com 0.30
ufl.edu 0.20
)";
        std::string actualOutput = runProgramWithInput(input);
        REQUIRE(actualOutput == expectedOutput);
    }

    TEST_CASE("Test 5: Graph with Multiple Outgoing Links - 2 Iterations",
"[PageRank][Complex]") {
        // This test focuses on a scenario where a single page has multiple
outgoing links.
        // It ensures that the input parsing and dummy PageRank handling work as
expected
        // for pages with varying out-degree.
```

```
        std::string input = R"(3 2
PageA PageB
PageA PageC
PageB PageA)";
        std::string expectedOutput = R"(facebook.com 0.20
gmail.com 0.20
google.com 0.10
maps.com 0.30
ufl.edu 0.20
)";
        std::string actualOutput = runProgramWithInput(input);
        REQUIRE(actualOutput == expectedOutput);
    }

    // --- 5 Additional Tests (Some Designed to Fail) ---
    // I added these to demonstrate that my testing environment correctly reports
failures.

    TEST_CASE("Test 6: Input with Zero Iterations (Expected Fail)",
"[PageRank][Failure]") {
        // This test checks an edge case where the number of power iterations is
zero.
        // My dummy output is fixed, so this test will fail because the expected
output is intentionally different.
        std::string input = R"(1 0
test.com test.com)";
        std::string expectedOutput = R"(This is a deliberately wrong output for
failure.
)";
        std::string actualOutput = runProgramWithInput(input);
        REQUIRE(actualOutput == expectedOutput);
    }

    TEST_CASE("Test 7: Input with Negative Iterations (Expected Fail)",
"[PageRank][Failure]") {
        // This test covers an invalid input scenario: a negative number of power
iterations.
        // It is designed to fail, as the expected output is intentionally
mismatched with the dummy output.
        std::string input = R"(1 -1
test.com test.com)";
        std::string expectedOutput = R"(This is another deliberately wrong output.
)";
        std::string actualOutput = runProgramWithInput(input);
        REQUIRE(actualOutput == expectedOutput);
    }
```

```
    TEST_CASE("Test 8: Large Number of Edges (Expected Pass with Dummy Output)",
"[PageRank][Scale]") {
        // This test uses a larger graph with more edges to ensure input parsing
scales.
        // With the current dummy PageRank, the test still passes as the expected
output matches the fixed dummy output.
        std::string input = R"(10 3
a b
b c
c d
d e
e f
f g
g h
h i
i j
j a)";
        std::string expectedOutput = R"(facebook.com 0.20
gmail.com 0.20
google.com 0.10
maps.com 0.30
ufl.edu 0.20
)";
        std::string actualOutput = runProgramWithInput(input);
        REQUIRE(actualOutput == expectedOutput);
    }

    TEST_CASE("Test 9: Simple Two-Way Link (Expected Pass with Dummy Output)",
"[PageRank][Basic]") {
        // This test verifies handling of a simple graph where two pages link to
each other.
        // It passes with the current dummy output, confirming basic input
processing for this structure.
        std::string input = R"(2 2
page1 page2
page2 page1)";
        std::string expectedOutput = R"(facebook.com 0.20
gmail.com 0.20
google.com 0.10
maps.com 0.30
ufl.edu 0.20
)";
        std::string actualOutput = runProgramWithInput(input);
        REQUIRE(actualOutput == expectedOutput);
    }

    TEST_CASE("Test 10: Deliberate Assertion Failure (Expected Fail)",
```

```
"[PageRank][Failure]") {
        // This test is designed to always fail, regardless of the program's
output.
        // It serves to confirm that the testing framework correctly reports a
hardcoded assertion failure.
        std::string input = R"(1 1
dummy.com dummy.com)";
        std::string actualOutput = runProgramWithInput(input);
        REQUIRE(false);
    }

} // End of namespace PageRankTests
```

Screenshot of Tests Running