1. **Problem Statement**

Create a program that can determine a Hamiltonian cycle of a graph using backtracking.

1. **Requirements**
   1. **Assumptions**

* Input File
  + Every input in a dataset is a non-negative integer
  + Items in a line are separated by a space
  + The input graph is simple, meaning there are no self-loops or parallel edges
  1. **Specifications**
* Must use a backtracking algorithm find the Hamiltonian cycle
* Output all data and actions to the screen and a file
  + Print the Hamiltonian cycle if one exists
    - If one doesn’t exist print that none exists

1. **Decomposition Diagram**

Main

Input file containing the graph datasets

Print all errors to the screen and to a file

Print all actions to the screen and to an output file

Determine if a Hamiltonian cycle exists using backtracking

Generate a graph of each dataset

Input

Output

Process

1. **Test Strategy**

* Valid Data  
  1.x
* File Handling  
  2.x

1. **Test Plan Version 1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Data | 1.0 | Input file containing only one graph |  |  |  |  |
| Data | 1.1 | Input file containing more than one graph |  |  |  |  |
| Data | 1.2 | Graph that creates a Hamiltonian cycle |  |  |  |  |
| Data | 1.3 | Graph that does not create a Hamiltonian cycle |  |  |  |  |
| Data | 1.4 | Datasets that create a disjointed graph |  |  |  |  |
| File Handling | 2.0 | Use an input file that exists |  |  |  |  |
| File Handling | 2.1 | Use an input file that is empty |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

1. **Initial Algorithm**

* Graph class
  + Private attributes:
    - adjMatrix: 2-dimensional vector array representing a matrix
      * Public get and set access funcitons
  + Constructor:
    - Parameters:
      * Integer for the size of the graph
    - set the adjMatrix size to the input parameter amount
    - Set every value in adjMatrix equal to 0
  + addEdge function
    - Parameters:
      * Integer: v1
      * Integer: v2
    - If either of the two vertices are greater than the size of the adjMatrix
      * Print “[ERROR] One of the vertices in the edge do not exist, cannot create edge!”
    - Find the index of both vertices in the vertex vector (v1 and v2)
    - Set the float value of [v1][v2] in the adjMatrix equal to 1
    - Set the float value of [v2][v1] in the adjMatrix equal to 1
  + printGraph function:
    - if adjMatrix is not empty
      * for each row in the matrix
        + print each character in the row separated by 2 spaces
        + move to a new line
    - if the adjMatrix is empty
      * print “[ERROR] Graph is empty, cannot print!”
  + findHamiltonianCycle function:
    - Parameters: integer vector for the current Hamiltonian path, integer index for the current position in the vector
    - If index equals total vertex count
      * If the last vertex in the path vector has a path to the first vertex
        + Return true
      * Return false
    - For each vertex
      * If there is a path to the current indexed vertex and if the vertex is not already included in the path vector
        + Add the vertex to the path vector

Print “Adding vertex [v]”

* + - * + If findHamiltonianCycle returns true

Parameters: current path vector and current index + 1

Then Return true

* + - * + Else remove the most recent vertex from the path vector

Print “Removing vertex: [v]”

* + - Return false since no vertex can be added
  + hamiltonianCycle Function:
    - Parameters: none
    - Create a new vector for the path
    - set the first value of the path vector equal to 0 to indicate starting at the first vertex
    - call findHamiltonianCycle function and pass the path vector as a parameter
    - if findHamiltonianCycle returns false
      * print “Hamiltonian cycle does not exist”
    - else
      * print “Hamiltonian Cycle Exists: “
      * print the values of the path vector
* Main Function
  + Parameters: none
  + Ask the user to enter the name of the input file
  + Attempt to open the input file
    - If it does not exist
      * Print “[ERROR] File does not exist”
      * Return
    - If file is empty
      * Print “[ERROR] File is empty”
      * Return
  + Vector of graph objects
  + Integer for edge count equal to 0
  + for each line in the file
    - if edge count equals 0
      * create new graph with a size of the first integer on the line
      * set edge count equal to the second integer on the line
      * add the graph to the graph vector
    - else
      * add the edge to the current graph
        + first integer as vertex 1 and second integer as vertex 2
      * decrease edge count by 1
  + for each graph in the vector
    - call printGraph function
    - call hamiltonianCycle function

1. **Test Plan Version 2**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Data | 1.0 | Input file containing only one graph | Graphs1.txt | Reads the dataset for the graph and displays the Hamiltonian cycle |  |  |
| Data | 1.1 | Input file containing more than one graph | Graphs2.txt | Successfully reads in each dataset and generates a graph for each |  |  |
| Data | 1.2 | Graph that creates a Hamiltonian cycle | Graphs2.txt  Graph 1 | Cycle: 1,2,4,3 |  |  |
| Data | 1.3 | Graph that does not create a Hamiltonian cycle | Graphs2.txt  Graph 3 | No possible cycle |  |  |
| Data | 1.4 | Datasets that create a disjointed graph of with several vertices | Graphs2.txt  Graph 5 | No possible cycle |  |  |
| Data | 1.5 | Dataset that contains edge vertex that do not exist | Graphs2.txt  Graph 7 | Prints an error for non-existent edges and generates cycle of 1,2,3,4 |  |  |
| Data | 1.6 | Dataset that contains no edges | Graphs2.txt  Graph 8 | No possible cycle |  |  |
| Data | 1.7 | Graph that does not create a Hamiltonian cycle | Graphs2.txt  Graph 9 | No possible cycle |  |  |
| Data | 1.8 | Datasets that create a disjointed graph with only one disjointed vertex | Graphs2.txt  Graph 6 | No possible cycle |  |  |
| Data | 1.9 | Graph that creates a Hamiltonian cycle | Graphs2.txt  graph 2 | Cycle: 1,2,3,4,5,6 |  |  |
| Data | 1.10 | Graph that creates a Hamiltonian cycle | Graphs2.txt  graph 4 | Cycle: 1,4,3,7,5,2,6,8 |  |  |
| File Handling | 2.0 | Use an input file that exists | Graph1.txt  Graph2.txt | Successfully opens each file |  |  |
| File Handling | 2.1 | Use an input file that is empty | Empty.txt | Prints “[ERROR] file is empty” |  |  |
| File Handling | 2.2 | Use an input file that does not exist | Somefile.txt | Prints “[ERROR] file does not exist |  |  |

1. **Code**

/\*!

\* Author: Jason Hogan

\* Description: Developed for CIS 3501 program #4. Generates a graph based on datasets inputted and finds the hamiltonian cycle.

\*

\*/

#include <string>

#include <iostream>

#include <fstream>

#include <sstream>

#include <iterator>

#include "Graph.h"

#include "DualStreams.h"

using namespace std;

/\*!

\* Main function that is called on applicaiton startup to test finding hamiltonian cycles on a graph (adjacency matrix).

\*

\*/

void main() {

DualStreams dsout("output.txt");

dsout.clearfile();

dsout << "Enter the name of the input file: \n";

string input;

getline(cin, input);

dsout << "Opening: " << input << "\n";

ifstream in(input);

if (in) {

vector<Graph> graphs;

int currentGraph = 0;

int edgeCount = 0;

while (getline(in, input)) {

if (edgeCount == 0) {

stringstream ss(input);

vector<string> strs(istream\_iterator<string>{ss}, istream\_iterator<string>());

Graph temp(stoi(strs[0]));

edgeCount = stoi(strs[1]);

graphs.push\_back(temp);

dsout << "Creating graph #" << to\_string(graphs.size()) << " with " << strs[0] << " vertices and " << strs[1] << " edges\n";

}

else {

stringstream ss(input);

vector<string> strs(istream\_iterator<string>{ss}, istream\_iterator<string>());

dsout << " Adding edge: " << strs[0] << ", " << strs[1] << "\n";

graphs.back().addEdge(stoi(strs[0])-1, stoi(strs[1])-1);

edgeCount = edgeCount - 1;

}

}

for (int x = 0; x < graphs.size(); x++) {

dsout << "============ Graph #" << to\_string(x + 1) << " ============\n";

graphs[x].printGraph();

graphs[x].hamiltonianCycle();

}

}

else {

dsout << "[ERROR] Could not open input file\n";

}

system("PAUSE");

}

/\*!

\* Author: Jason Hogan

\* Description: Developed for CIS 3501 program #4. Generates a graph based on datasets inputted and finds the hamiltonian cycle.

\*

\*/

#pragma once

#include <fstream>

#include <iostream>

using namespace std;

/\*!

\* Designed to mimic std::cout functionality.

\* Writes data to both the cout stream and the specified file output stream.

\*

\*/

class DualStreams

{

private:

const char\* filename;

ofstream file\_stream;

bool valid\_state;

public:

DualStreams() {};

~DualStreams() {};

DualStreams(const char\* file) : valid\_state(file\_stream) { filename = file; };

explicit operator bool() const {

return valid\_state;

}

template<typename T>

DualStreams& operator<<(T&& t);

/\*!

\* Overloads the "<<" operator to allow for DualStreams to function as a single stream.

\*

\*/

void clearfile() {

file\_stream.open(filename, ofstream::out | ofstream::trunc);

file\_stream.close();

};

};

/\*!

\* Template funciton that is used to write the data to the cout stream and the file stream

\* Due to multiple instances of the file being open at once, it has to reopen the file on each call to DualStreams.

\*

\* \param t

\* \return

\*/

template<typename T>

inline DualStreams & DualStreams::operator<<(T && t)

{

file\_stream = ofstream(filename, ios\_base::app);

if (!valid\_state) {

return \*this;

}

if (!(cout << t)) {

valid\_state = false;

return \*this;

}

if (!(file\_stream << t)) {

valid\_state = false;

cout << "fstream error";

return \*this;

}

file\_stream.close();

return \*this;

}

/\*!

\* Author: Jason Hogan

\* Description: Developed for CIS 3501 program #4. Generates a graph based on datasets inputted and finds the hamiltonian cycle.

\*

\*/

#pragma once

#include <vector>

#include <iostream>

#include <string>

#include "DualStreams.h"

using namespace std;

/\*!

\* Handles all functionality related to creating, modifying, and processing the graph.

\*

\*/

class Graph

{

private:

vector<vector<int>> adjMatrix;

public:

Graph();

Graph(int size);

~Graph();

void addEdge(int v1, int v2);

void printGraph();

bool findHamiltonianCycle(vector<int>\* cycle);

void hamiltonianCycle();

};

/\*!

\* Author: Jason Hogan

\* Description: Developed for CIS 3501 program #4. Generates a graph based on datasets inputted and finds the hamiltonian cycle.

\*

\*/

#include "Graph.h"

Graph::Graph()

{

}

Graph::Graph(int size)

{

vector<vector<int>> m(size, vector<int>(size));

adjMatrix = m;

}

Graph::~Graph()

{

}

/\*!

\* Adds an edge between the two vertices (v1 and v2) to the adjacency matrix.

\*

\* \param v1

\* \param v2

\*/

void Graph::addEdge(int v1, int v2)

{

DualStreams dsout("output.txt");

bool valid = true;

if (v1 < 0 || v1 >= adjMatrix.size()) {

dsout << " Vertex #1 does not exist: v" + to\_string(v1+1) << "\n";

valid = false;

}

if (v2 < 0 || v2 >= adjMatrix.size()) {

dsout << " Vertex #2 does not exist: v" + to\_string(v2+1) << "\n";

valid = false;

}

if (valid == false) {

return;

}

// set the edge in the matrix

adjMatrix[v1][v2] = 1;

adjMatrix[v2][v1] = 1;

}

/\*!

\* Prints the ajdacency graph to the screen and the output.txt file.

\*

\*/

void Graph::printGraph()

{

DualStreams dsout("output.txt");

if (adjMatrix.size() != 0) {

dsout << "Printing adjacency matrix for the graph: " << "\n";

for (int x = 0; x < adjMatrix.size(); x++) {

for (int i = 0; i < adjMatrix.size(); i++) {

cout << to\_string(adjMatrix[x][i]) << ", ";

}

cout << endl;

}

}

}

/\*!

\* Recursively finds the hamiltonian cycle using a backtracking algorithm.

\* At each step of the algorithm it will either add a vertex to the cycle vector

\* or remove the most recent one if no futher paths are possible.

\*

\* \param cycle

\* \return

\*/

bool Graph::findHamiltonianCycle(vector<int>\* cycle)

{

DualStreams dsout("output.txt");

if (cycle->size() == adjMatrix.size()) {

if (adjMatrix[cycle->back()][cycle->front()]) {

cycle->push\_back(cycle->front());

return true;

}

return false;

}

for (int i = 0; i < adjMatrix.size(); i++) {

if (adjMatrix[i][cycle->back()] == 1) {

bool contains = false;

for (int x = 0; x < cycle->size(); x++) {

if ((\*cycle)[x] == i) {

contains = true;

}

}

if (contains == false) {

cycle->push\_back(i);

dsout << "Adding vertex " << to\_string(i+1) << "\n";

if (findHamiltonianCycle(cycle)) {

return true;

}

else {

cycle->pop\_back();

dsout << "Removing vertex " << to\_string(i+1) << "\n";

}

}

}

}

return false;

}

/\*!

\* The function to call from outside the graph class when attempting to find the hamiltonian cycle.

\* Calls the recursive function findHamiltonianCycle to use backtracking to find the cycle.

\*

\*/

void Graph::hamiltonianCycle()

{

DualStreams dsout("output.txt");

vector<int>\* cycle = new vector<int>();

cycle->push\_back(0); // Add the first vertex to the vector

dsout << "Adding vertex " << to\_string(1)<< "\n";

if (findHamiltonianCycle(cycle) == true) {

dsout << "Hamiltonian Cycle Exists: " << "\n";

for (int x = 0; x < cycle->size(); x++) {

dsout << to\_string((\*cycle)[x]+1) << " ";

}

dsout << "\n";

}

else {

dsout << "No Hamiltonian Cycle found" << "\n";

}

}

1. **Updated Algorithm**

* Graph class
  + Private attributes:
    - adjMatrix: 2-dimensional vector array representing a matrix
      * Public get and set access funcitons
  + Constructor:
    - Parameters:
      * Integer for the size of the graph
    - set the adjMatrix size to the input parameter amount
    - Set every value in adjMatrix equal to 0
  + addEdge function
    - Parameters:
      * Integer: v1
      * Integer: v2
    - ~~If either of the two vertices are greater than the size of the adjMatrix~~
      * ~~Print “[ERROR] One of the vertices in the edge do not exist, cannot create edge!”~~
    - Boolean value for valid vertices set to true
    - If v1 is out of range of the adjMatrix
      * Print “ Vertex #1 does not exist: v[vertex]”
      * Set the valid vertices Boolean equal to false
    - If v2 is out of range of the adjMatrix
      * Print “ Vertex #2 does not exist: v[vertex]”
      * Set the valid vertices Boolean equal to false
    - If valid vertices is false
      * Return from the function
    - ~~Find the index of both vertices in the vertex vector (v1 and v2)~~
    - Set the float value of [v1][v2] in the adjMatrix equal to 1
    - Set the float value of [v2][v1] in the adjMatrix equal to 1
  + printGraph function:
    - if adjMatrix is not empty
      * for each row in the matrix
        + print each character in the row separated by a comma
        + move to a new line
    - if the adjMatrix is empty
      * print “[ERROR] Graph is empty, cannot print!”
  + findHamiltonianCycle function:
    - Parameters: integer vector (cycle) for the current Hamiltonian path, ~~integer index for the current position in the vector~~
    - If the current size of the cycle vector equals the size of the adjMatrix
      * If the last vertex in the path vector has a path to the first vertex
        + Add the first vertex of the cycle vector to the back as well
        + Return true
      * Return false
    - For each vertex
      * If there is a path to the last vertex in the cycle vector and if the vertex is not already included in the path vector
        + Add the vertex to the path vector

Print “Adding vertex [v]”

* + - * + If findHamiltonianCycle returns true

Parameters: current path vector and current index + 1

Then Return true

* + - * + Else remove the most recent vertex from the path vector

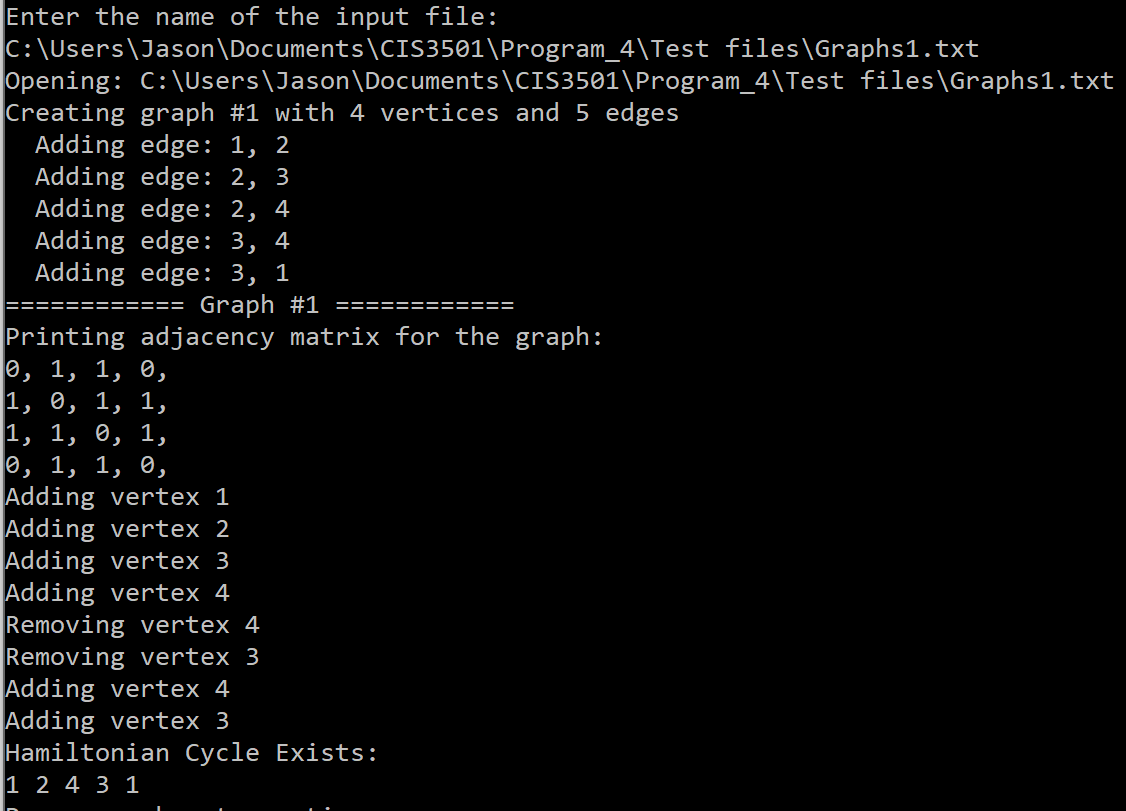
Print “Removing vertex: [v]”

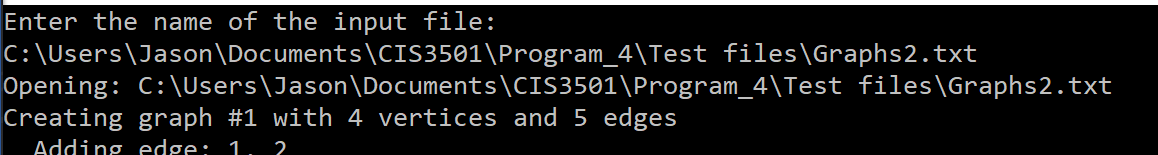
* + - Return false since no vertex can be added
  + hamiltonianCycle Function:
    - Parameters: none
    - Create a new vector for the path
    - set the first value of the path vector equal to 0 to indicate starting at the first vertex
    - call findHamiltonianCycle function and pass the path vector as a parameter
    - if findHamiltonianCycle returns True
      * print “Hamiltonian Cycle Exists: “
      * print the values of the path vector
      * print “Hamiltonian cycle does not exist”
    - else
      * print “Hamiltonian cycle does not exist”
* DualStreams Class
  + Constructor
    - Parameters:
      * Constant character array for the file name to write to
    - Sets the private variable for the filename equal to the value passed as a parameter
  + clearfile function
    - Opens the file stream of the filename stored in the private variable for the class
    - Erases the file so that it is empty
  + Operator “<<” overload
    - Parameters:
      * Value t of some template type
    - Attempts to open the file of the filename specified in the private variable
    - If it is not in a valid state, then it will return from the function
    - Attempts to write the value to the screen using cout from the std library
    - Attempts to write the value to the file stream that was created
    - Closes the file stream and returns from the function
* Main Function
  + Parameters: none
  + Ask the user to enter the name of the input file
  + Attempt to open the input file
    - If it does not exist
      * Print “[ERROR] File does not exist”
      * Return
    - If file is empty
      * Print “[ERROR] File is empty”
      * Return
  + Vector of graph objects
  + Integer for edge count equal to 0
  + for each line in the file
    - if edge count equals 0
      * create new graph with a size of the first integer on the line
      * set edge count equal to the second integer on the line
      * add the graph to the graph vector
    - else
      * add the edge to the current graph
        + first integer as vertex 1 and second integer as vertex 2
      * decrease edge count by 1
  + for each graph in the vector
    - call printGraph function
    - call hamiltonianCycle function

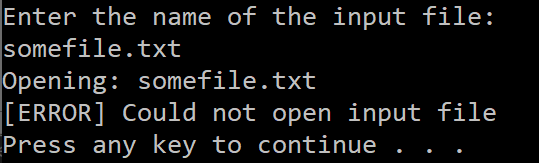
1. **Test Plan Version 3**

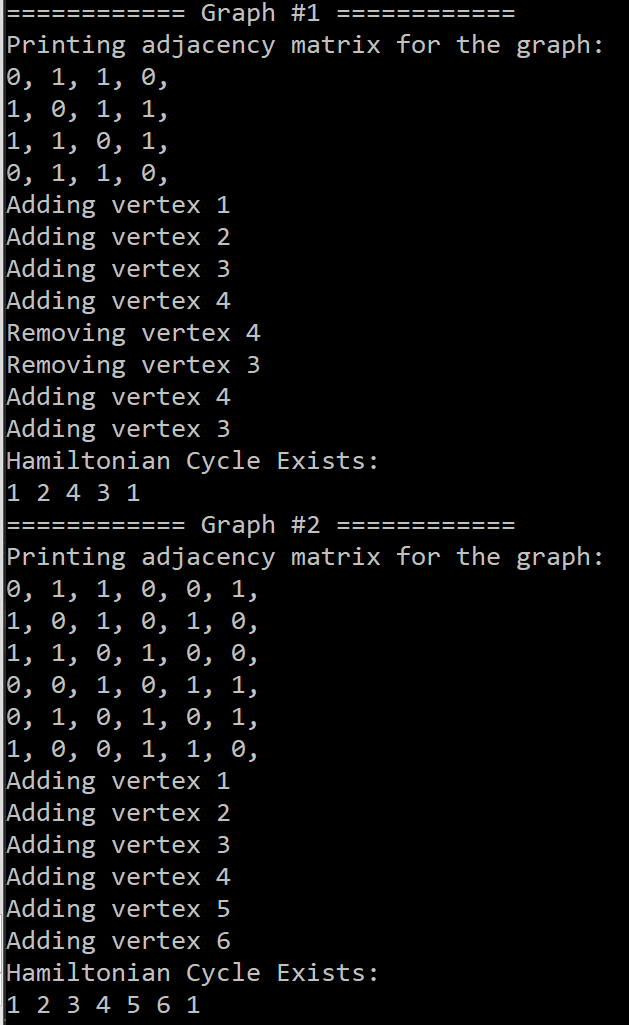
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Data | 1.0 | Input file containing only one graph | Graphs1.txt | Reads the dataset for the graph and displays the Hamiltonian cycle | Hamiltonian Cycle Exists:  1 2 4 3 1 | PASS |
| Data | 1.1 | Input file containing more than one graph | Graphs2.txt | Successfully reads in each dataset and generates a graph for each | Processed all datasets.  See output2.txt | PASS |
| Data | 1.2 | Graph that creates a Hamiltonian cycle | Graphs2.txt  Graph 1 | Cycle: 1,2,4,3 | See output2.txt  Hamiltonian Cycle Exists:  1 2 4 3 1 | PASS |
| Data | 1.3 | Graph that does not create a Hamiltonian cycle | Graphs2.txt  Graph 3 | No possible cycle | See output2.txt  ….  Removing vertex 5  Removing vertex 4  No Hamiltonian Cycle found | PASS |
| Data | 1.4 | Datasets that create a disjointed graph of with several vertices | Graphs2.txt  Graph 5 | No possible cycle | See output2.txt  ….  Removing vertex 3  Removing vertex 4  No Hamiltonian Cycle found | PASS |
| Data | 1.5 | Dataset that contains edge vertex that do not exist | Graphs2.txt  Graph 7 | Prints an error for non-existent edges and generates cycle of 1,2,3,4 | See output2.txt  Printed Error for edges.  Hamiltonian Cycle Exists:  1 2 3 4 1 | PASS |
| Data | 1.6 | Dataset that contains no edges | Graphs2.txt  Graph 8 | No possible cycle | See output2.txt  ….  Adding vertex 1  No Hamiltonian Cycle found | PASS |
| Data | 1.7 | Graph that does not create a Hamiltonian cycle | Graphs2.txt  Graph 9 | No possible cycle | See output2.txt  ….  Removing vertex 4  Removing vertex 5  No Hamiltonian Cycle found | PASS |
| Data | 1.8 | Datasets that create a disjointed graph with only one disjointed vertex | Graphs2.txt  Graph 6 | No possible cycle | See output2.txt  ….  Removing vertex 3  Removing vertex 4  No Hamiltonian Cycle found | PASS |
| Data | 1.9 | Graph that creates a Hamiltonian cycle | Graphs2.txt  graph 2 | Cycle: 1,2,3,4,5,6 | See output2.txt  ….  Adding vertex 6  Hamiltonian Cycle Exists:  1 2 3 4 5 6 1 | PASS |
| Data | 1.10 | Graph that creates a Hamiltonian cycle | Graphs2.txt  graph 4 | Cycle: 1,4,3,7,5,2,6,8 | See output2.txt  ….  Adding vertex 6  Adding vertex 8  Hamiltonian Cycle Exists:  1 4 3 7 5 2 6 8 1 | PASS |
| File Handling | 2.0 | Use an input file that exists | Graph1.txt  Graph2.txt | Successfully opens each file | See output1.txt and output2.txt  Each File opened | PASS |
| File Handling | 2.1 | Use an input file that is empty | Empty.txt | Prints “[ERROR] file is empty” | Opening: somefile.txt  [ERROR] Could not open input file | PASS |
| File Handling | 2.2 | Use an input file that does not exist | Somefile.txt | Prints “[ERROR] file does not exist | Opening: somefile.txt  [ERROR] Could not open input file | PASS |

1. **Screenshots**

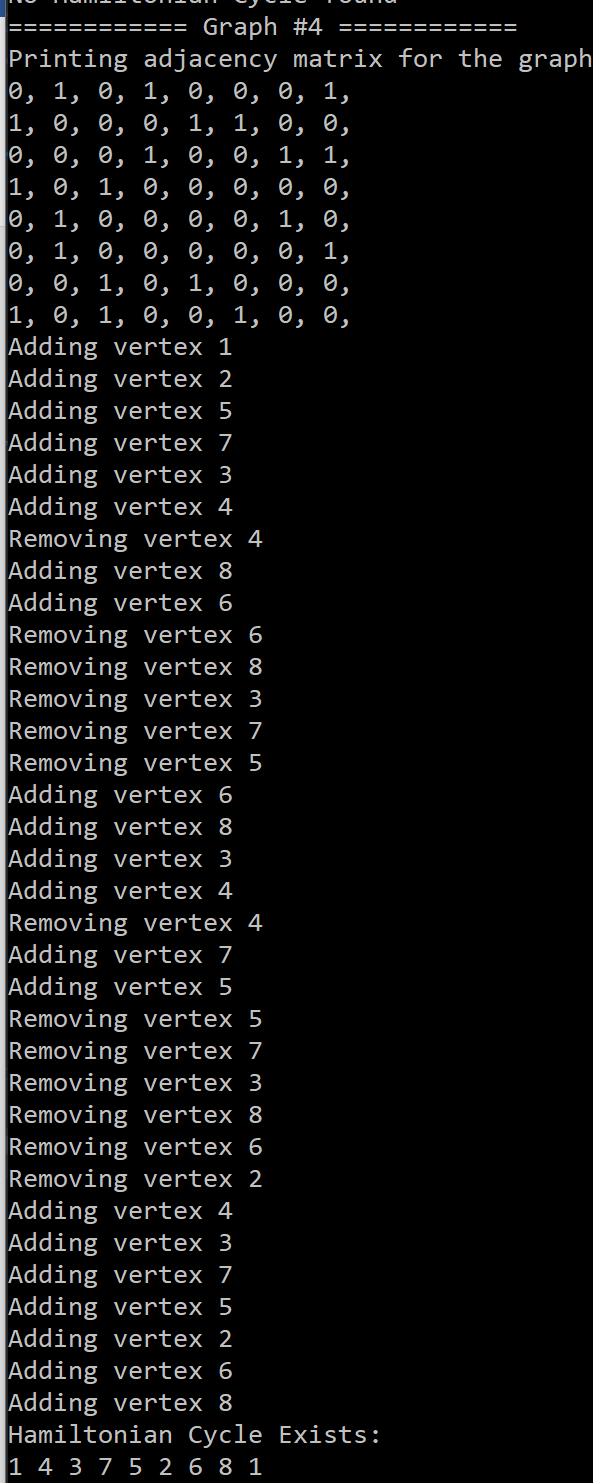
Test 1.0

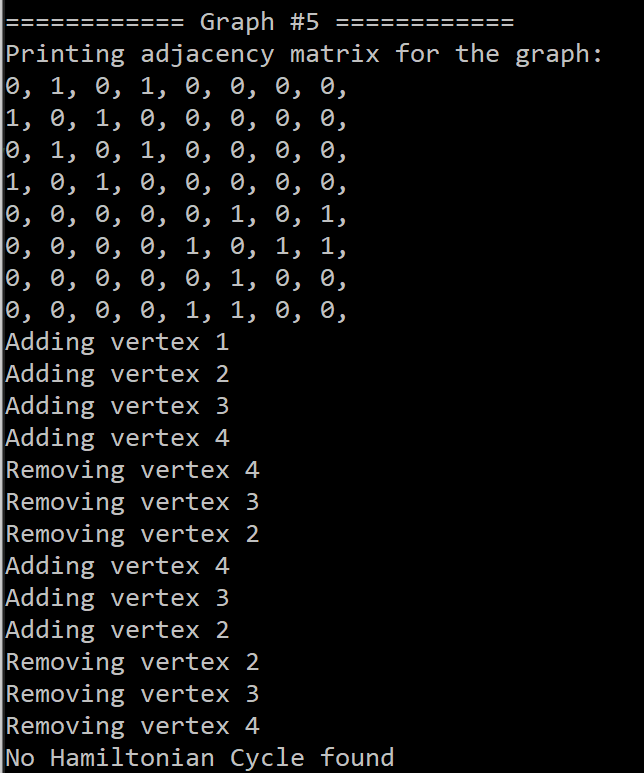
Test 1.1 & 2.0

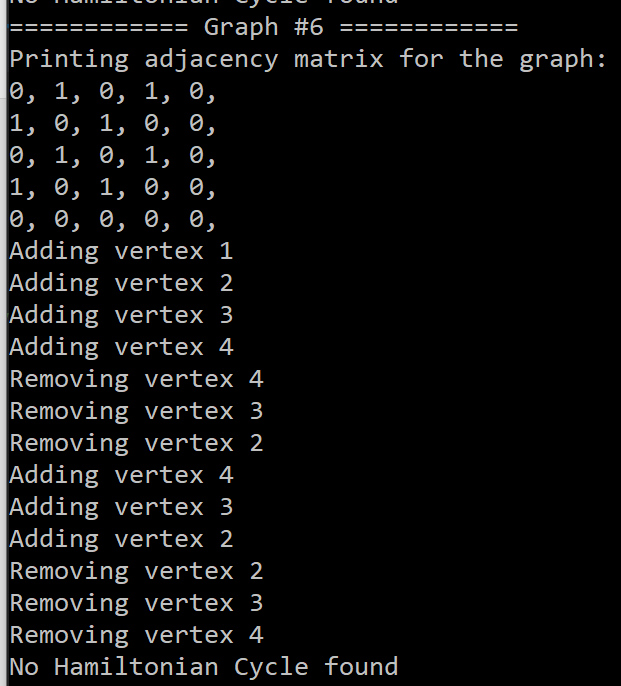
Test 2.1 & 2.2

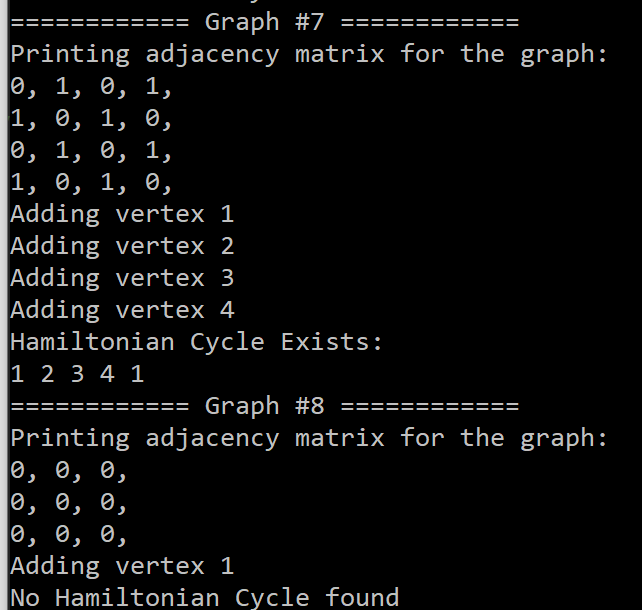
Test 1.2 & 1.9

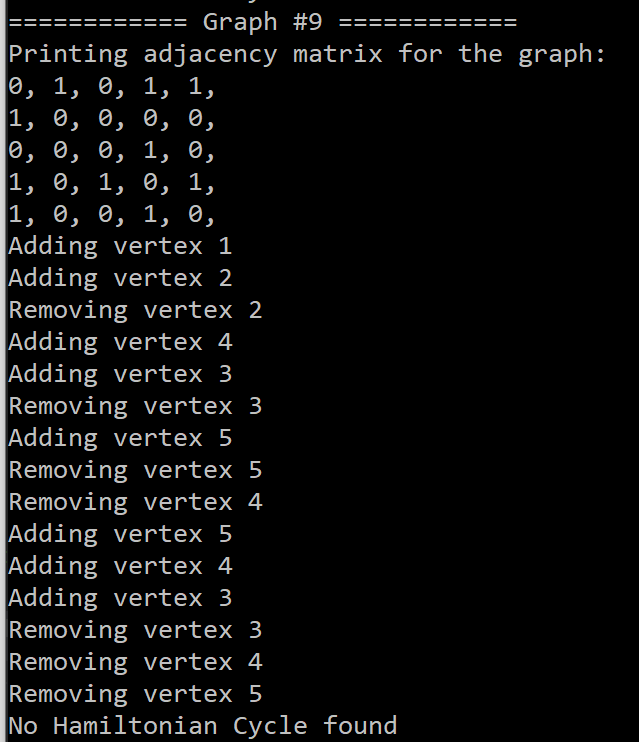
Test 1.3

Test 1.10

Test 1.4

Test 1.8

Test 1.5 & 1.6

Test 1.7

1. **Error Log**

|  |  |  |
| --- | --- | --- |
| Error Type | Cause of Error | Solution to Error |
|  |  |  |
|  |  |  |

1. **Status**

The application functions successfully and meets all requirements. It does run slower and is not quite efficient due to the way that the DualStreams class opens and closes the file on each reference, this is done because otherwise there would be multiple streams open to the same file during runtime causing a bunch of other problems; such as data being wrote to the file out of order.