Printout for cs320-11-A24-Graph.cpp

```
// File: Graph/Graph.cpp
// Mike Lim 2017/12/5

#include <iostream>
#include "Graph.hpp"
#include "QueueL.hpp"
using namespace std;

// ========= Constructor =========
Graph::Graph(bool isDigraph, int numVert) :
    _isDigraph(isDigraph),
    _numVertices(numVert),
    _vertex(numVert),
    _graph(numVert),
    _graphIter(numVert) {
    for (int i = 0; i < numVert; i++) {
        _graphIter[i].setIterListL(&_graph[i]);
    }
    }

// ========= Destructor =========
Graph::~Graph() {
    for (int i = 0; i < _numVertices; i++) {
        _graph[i].clear();
    }
}

// ========= Breadth first search =========
void Graph::breadthFirstSearch(int s, ostream &os) {
    initGraph();
    os << endl;
    bfs(s, os);
    writeVerticesPostBreadth(os);
}

void Graph::bfs(int s, ostream &os) {
    _vertex[s].color = GRAY;
    _vertex[s].distance = 0;
    _vertex[s].predecessor = -1;
    QueueL<int> queue;
    _vertex[s].discovered = ++_time;
    queue.enqueue(s);
    while (!queue.isEmpty()){
        int i = queue.dequeue();
        os << i << " ";
        _vertex[i].finished = ++_time;
        for (_graphIter[i].first(); !_graphIter[i].isDone(); _graphIter[i].next()){
            int j = _graphIter[i].currentItem();
            if(_vertex[j].color == WHITE){
                _vertex[j].color = GRAY;
                _vertex[j].distance = _vertex[i].distance + 1;
                _vertex[j].predecessor = i;
                _vertex[j].discovered = ++_time;
                queue.enqueue(j);
            }
        }
        _vertex[i].color = BLACK;
    }
}
```

```
// ========= Depth first search =========
void Graph::depthFirstSearch(int s, ostream &os) {
    initGraph();
    os << endl;
    dfs(s, os);
    writeVerticesPostDepth(os);
}

void Graph::dfs(int u, ostream &os) {
    os << u << " ";
    _vertex[u].discovered = ++_time;
    _vertex[u].color = GRAY;
    for (_graphIter[u].first(); !_graphIter[u].isDone(); _graphIter[u].next()){
        int i = _graphIter[u].currentItem();
        if(_vertex[i].color == WHITE){
            _vertex[i].predecessor = u;
            dfs(i, os);
        }
    }                                   Assignment =.
    _vertex[u].color == BLACK;          You are lucky this did not affect your output.
    _vertex[u].finished = ++_time;      -2
}

// ========= initGraph =========
void Graph::initGraph() {
    _time = 0;
    for (int i = 0; i < _numVertices; i++) {
        _vertex[i].color = WHITE;
        _vertex[i].discovered = -1;
        _vertex[i].finished = -1;
        _vertex[i].distance = -1;
        _vertex[i].predecessor = -1;
    }
}

// ========= insertEdge =========
void Graph::insertEdge(int from, int to) {
    if ((from < 0) || (_numVertices <= from) || (to < 0) || (_numVertices <= to)) {
        cerr << "insertEdge precondition violated: from or to out of range." << endl;
        cerr << "from == " << from << "  to == " << to << endl;
        throw -1;
    }
    if (!_graph[from].contains(to)) {
        _graph[from].prepend(to);
    }
    if (!_isDigraph && !_graph[to].contains(from)) {
        _graph[to].prepend(from);
    }
}

// ========= numEdges =========
int Graph::numEdges() {
    int result = 0;
    for (int i =0; i<_numVertices; i++){
        result += _graph[i].length();
    }
    return _isDigraph ? result : result/2;
}

// ========= removeEdge =========
```

```
void Graph::removeEdge(int from, int to) {
        if ((from<0) || (_numVertices <= from) || (to < 0) || (_numVertices <= to)){
            cerr << "removeEdge precondition violated" << endl;
            cerr << "from = " << from << " to = " << to << endl;
            throw -1;
        }
        _graph[from].remove(to);
        if(!_isDigraph){
            _graph[to].remove(from);
        }
    }


// ========= writeAdjacencyLists =========
void Graph::writeAdjacencyLists(ostream &os) {
    os << "Adjacency lists" << endl;
    for (int i = 0; i < _numVertices; i++) {
        os << i << ": ";
        os << _graph[i] << endl;
    }
}

// ========= writeComponents =========
void Graph::writeComponents(ostream &os) {
    int numComponents = 0;
    initGraph();
    for (int u = 0; u < _numVertices; u++){
        if(_vertex[u].color==WHITE){
            os << "Connected component: " << endl;
            dfs (u, os);
            os << endl;
            numComponents++;
        }
    }
    if (numComponents == 1) {
        os << "\n There is one connected component." << endl;
    } else {
        os << "\n There are " << numComponents << "connected." << endl;
    }
    cout << endl;
}

// ========= writePath =========
void Graph::writePath(int from, int to, ostream &os) {
    if ((from < 0) || (_numVertices <= from) || (to < 0) || (_numVertices <= to)) {
        cerr << "minimumDistance precondition violated: from or to out of range." <<
endl;
        cerr << "from == " << from << "  to == " << to << endl;
        throw -1;
    }
    initGraph();
    os << "\nBreadth-first search from " << from << ": ";
    bfs(from, os);
    os << "\nPath from " << from << " to " << to << " is: ";
    writePathHelper(from, to, os);
    if (_vertex[to].distance != -1) {
        os << "\nDistance = " << _vertex[to].distance << endl;
    }
}

void Graph::writePathHelper(int from, int to, ostream &os) {
```

```
    if (to == from) {
        os << from << " ";
    } else if (_vertex[to].predecessor == -1){
        os << "\n No Path from " << from << " to " << to << " exists " << endl;
    } else {
        writePathHelper(from, _vertex[to].predecessor, os);
        os << to << " ";
    }
}

// ========= write vertices =========
void Graph::writeVerticesPostBreadth(ostream &os) {
    os << "\n\nDiscovered/finished, predecessor, distance";
    os << endl;
    for (int i = 0; i < _numVertices; i++) {
        os << i << ": "
           << _vertex[i].discovered << "/" << _vertex[i].finished << ", "
           << _vertex[i].predecessor << ", "
           << _vertex[i].distance << endl;
    }
}

void Graph::writeVerticesPostDepth(ostream & os) {
    os << "\n\nDiscovered/finished, predecessor";
    os << endl;
    for (int i = 0; i < _numVertices; i++) {
        os << i << ": "
           << _vertex[i].discovered << "/" << _vertex[i].finished << ", "
           << _vertex[i].predecessor << endl;
    }
}

// new page
```

```
// new page
```

```
clang++ -stdlib=libc++ -std=c++11 -c GraphMain.cpp \
        -I ../dp4dsDistribution/Utilities \
        -I ../dp4dsDistribution/ListL \
        -I ../dp4dsDistribution/ASeq \
        -I ../dp4dsDistribution/ArrayT \
        -I ../dp4dsDistribution/ArrayP 2>&1
clang++ -stdlib=libc++ -std=c++11 -c Graph.cpp \
        -I ../dp4dsDistribution/Utilities \
        -I ../dp4dsDistribution/ListL \
        -I ../dp4dsDistribution/ASeq \
        -I ../dp4dsDistribution/ArrayT \
        -I ../dp4dsDistribution/ArrayP 2>&1
Graph.cpp:80:22: warning: equality comparison result unused [-Wunused-comparison]
    _vertex[u].color == BLACK;
    ~~~~~~~~~~~~~~~~~^~~~~~~~
Graph.cpp:80:22: note: use '=' to turn this equality comparison into an assignment
    _vertex[u].color == BLACK;
                     ^~
                     =
1 warning generated.
clang++ -stdlib=libc++ -std=c++11 -o GraphMain GraphMain.o Graph.o \
        ../dp4dsDistribution/Utilities/Utilities.o
strip GraphMain

// new page
```

```
Testing cs320-11

Testing cs320-11 unit-breadth-first-digraph
==========================================
Adjacency lists
0: (3, 1)
1: (5)
2: (1, 4)
3: (1)
4: (3, 5)
5: (2)

0 3 1 5 2 4

Discovered/finished, predecessor, distance
0: 1/2, -1, 0
1: 4/6, 0, 1
2: 9/10, 5, 3
3: 3/5, 0, 1
4: 11/12, 2, 4
5: 7/8, 1, 2

1 5 2 4 3

Discovered/finished, predecessor, distance
0: -1/-1, -1, -1
1: 1/2, -1, 0
2: 5/6, 5, 2
3: 9/10, 4, 4
4: 7/8, 2, 3
5: 3/4, 1, 1

// new page
```

```
Testing cs320-11 unit-breadth-first
===================================
Adjacency lists
0: (1)
1: (5, 4, 3, 0)
2: (5)
3: (4, 1)
4: (5, 3, 1)
5: (4, 2, 1)

0 1 5 4 3 2

Discovered/finished, predecessor, distance
0: 1/2, -1, 0
1: 3/4, 0, 1
2: 9/12, 5, 3
3: 7/11, 1, 2
4: 6/10, 1, 2
5: 5/8, 1, 2

2 5 4 1 3 0

Discovered/finished, predecessor, distance
0: 10/12, 1, 3
1: 6/9, 5, 2
2: 1/2, -1, 0
3: 8/11, 4, 3
4: 5/7, 5, 2
5: 3/4, 2, 1

// new page
```

```
Testing cs320-11 unit-components
================================
Adjacency lists
0: (1)
1: (5, 2, 0)
2: (5, 1)
3: ()
4: (5)
5: (4, 2, 1)
6: (7)
7: (6)

Connected component:
0 1 5 4 2
Connected component:
3
Connected component:
6 7

 There are 3connected.


// new page
```

```
Testing cs320-11 unit-depth-first-digraph
==========================================
Adjacency lists
0: (3, 1)
1: (5)
2: (1, 4)
3: (1)
4: (3, 5)
5: (2)

4 3 1 5 2

Discovered/finished, predecessor
0: -1/-1, -1
1: 3/8, 3
2: 5/6, 5
3: 2/9, 4
4: 1/10, -1
5: 4/7, 1

5 2 1 4 3

Discovered/finished, predecessor
0: -1/-1, -1
1: 3/4, 2
2: 2/9, 5
3: 6/7, 4
4: 5/8, 2
5: 1/10, -1

// new page
```

```
Testing cs320-11 unit-depth-first
=================================
Adjacency lists
0: (1)
1: (5, 4, 3, 0)
2: (5)
3: (4, 1)
4: (5, 3, 1)
5: (4, 2, 1)

0 1 5 4 3 2

Discovered/finished, predecessor
0: 1/12, -1
1: 2/11, 0
2: 8/9, 5
3: 5/6, 4
4: 4/7, 5
5: 3/10, 1

2 5 4 3 1 0

Discovered/finished, predecessor
0: 6/7, 1
1: 5/8, 3
2: 1/12, -1
3: 4/9, 4
4: 3/10, 5
5: 2/11, 2

// new page
```

```
Testing cs320-11 unit-paths
===========================
Adjacency lists
0: (4)
1: (2)
2: (5)
3: (4, 0)
4: (2, 0)
5: (4)

Breadth-first search from 0: 0 4 2 5
Path from 0 to 5 is: 0 4 2 5
Distance = 3

Breadth-first search from 5: 5 4 2 0
Path from 5 to 0 is: 5 4 0
Distance = 2

Breadth-first search from 3: 3 4 0 2 5
Path from 3 to 5 is: 3 4 2 5
Distance = 3

Breadth-first search from 5: 5 4 2 0
Path from 5 to 3 is:
 No Path from 5 to 3 exists

// new page
```

```
Testing cs320-11 unit-remove-digraph
====================================
Number of edges: 9

Adjacency lists
0: (3, 1)
1: (5)
2: (1, 4)
3: (1)
4: (3, 5)
5: (2)

Number of edges: 7

Adjacency lists
0: (3, 1)
1: (5)
2: (1, 4)
3: ()
4: (3)
5: (2)

// new page
```

```
Testing cs320-11 unit-remove
============================
Number of edges: 7

Adjacency lists
0: (1)
1: (5, 4, 3, 0)
2: (5)
3: (4, 1)
4: (5, 3, 1)
5: (4, 2, 1)

Number of edges: 5

Adjacency lists
0: (1)
1: (5, 3, 0)
2: ()
3: (4, 1)
4: (5, 3)
5: (4, 1)

// new page
```