Graphs

Updated: 7th December, 2021

Aims

- To create a general purpose graph class.
- To implement search algorithms in the graph class.

Before the Practical

- Read this practical sheet fully before starting.
- Ensure you have completed the activities from previous practicals.

Activities

1. UML Diagram

We will be using an *Adjacency List* implementation of graphs in this practical.
Following the notes from the lecture slides as a guide, draw up the UML diagrams for `DSAGraph`, `DSAGraphNode` and their test harnesses. Make sure to include any other classes they make use of. Update this diagram as you work through the practical.

2. Graph Implementation

Create a `DSAGraph` class using linked lists to store the list of nodes and a `DSAGraphNode` class using linked lists within each node to store the adjacency list.
At a minimum, implement all the methods outlined in the lecture slides for `DSAGraph` and `DSAGraphNode`. You should implement additional methods as necessary to further develop your graph implementation. Write a test harness to test each method thoroughly, be sure to test all cases.

Note:

- Ensure you have implemented your `displayAsList()` and `displayAsMatrix()` methods to help your testing. Test your graph with a small graph first, display it, then add more nodes/edges and display it again.
- There are many choices you may make in developing your graph implementation. Examples include:
  - Directed or undirected graph?
  - Edge creation with non-existent vertices. Should you throw an exception or implement a try/catch to create the vertices?

The implementation of a `DSAGraphEdge` *is optional for this practical.*

3. Read Graph From File

Two input files for graphs (prac6_1.al and prac6_2.al) have been uploaded on Blackboard. Create a program to read in the graph information from these files and create a DSAGraph object. Work out manually what they should look like and check it against your display output.

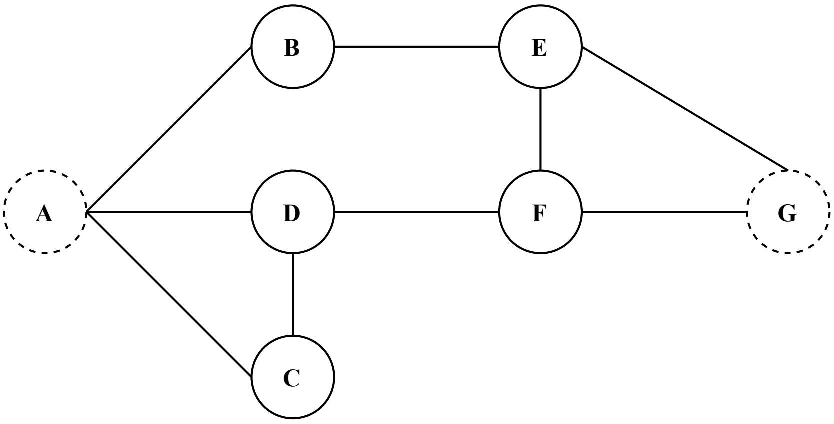
Note: An interactive menu system similar to those in previous practicals could also be implemented.

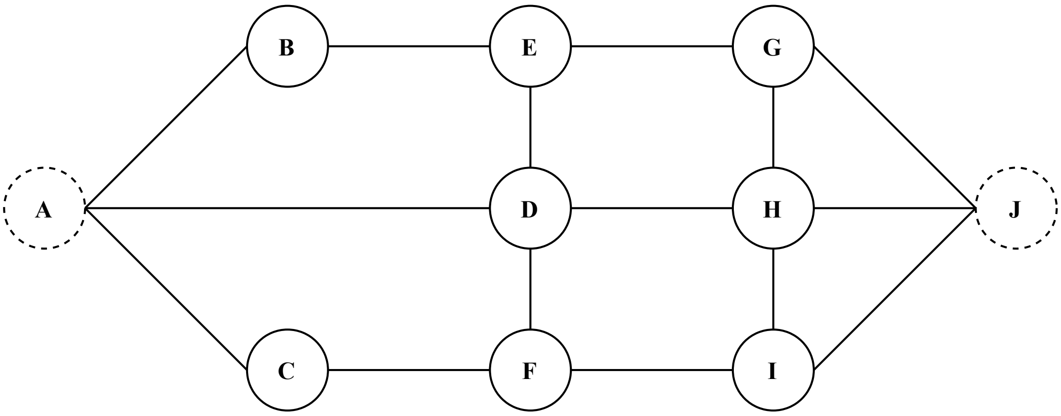
The format for the file is given below:

```
Edge_vertex1 Edge_vertex2
Edge_vertex1 Edge_vertex2
Edge_vertex1 Edge_vertex2
```

4. Manual Depth First Search and Breadth First Search

Consider the following graphs and carry out a Breadth First Search and a Depth First Search *manually* based on the algorithms in the lecture notes. (Assume that they are sorted alphabetically - so you will choose vertices in alphabetical order)





5. Depth First Search and Breadth First Search Implementation

Following the notes from the lecture slides and the pseudocode as a guide, implement methods for `depthFirstSearch()` and `breadthFirstSearch()` in your `DSAGraph` class. Test them against the graphs read in from Activity 3 and compare your results to those obtained in Activity 4.

```
breadthFirstSearch()
        Declare T = DSAQueue and Q = DSAQueue
        Iterate through your vertices list and clear visited
        Reference a vertex from your vertices list as v
        Set v as visited
        Enqueue v into Q
        while Q is not empty
                v = Q.dequeue()
                for each vertex w in v's adjacency list that is unvisited
                        T.enqueue(v)
                        T.enqueue(w)
                        Set w as visited
                        Enqueue w into Q

depthFirstSearch()
        Declare T = DSAQueue and S = DSAStack
        Iterate through your vertices list and clear visited
        Reference a vertex from your vertices list as v
        Set v as visited
        Push v onto S
        while S is not empty
                while there is an unvisited vertex w in v's adjacency list
                (w is the next unvisited vertex in v's adjacency list)
                        T.enqueue(v)
                        T.enqueue(w)
                        Set w as visited
                        Push w onto S
                        v = w
                v = S.pop()
```

Note:

- A helper method can assist with returning w for Depth First Search.
- For alphabetical order preference, you may wish sort your vertices and adjacency lists using a sorting algorithm from Practical 1.

Submission Deliverable

- Your code and UML diagrams are due 2 weeks from your current tutorial session.
  - You will demonstrate your work to your tutors during that session
  - If you have completed the practical earlier, you can demonstrate your work during the next session
- You must **submit** your code and any test data that you have been using **electronically via Blackboard** under the *Assessments* section before your demonstration.
  - Java students, please do not submit the *.class files

Marking Guide

Your submission will be marked as follows:

- [2] Your UML diagram for all implemented classes and methods.
- [2] Your DSAGraph is implemented correctly - your test harness will show this.
- [2] You can read in a file and create a corresponding graph.
- [2] You have manually worked through the depth first search and breadth first search problems - submit a .pdf or image file.
- [2] You have implemented the depth first search and breadth first search methods.

End of Worksheet