Readme file for searching sub graph isomorphism

Author: Alvin Chia

Design of Data Structure

Before deciding what data structure to be implemented, let's look at the search for sub graph isomorphism. The search for sub graph isomorphism needs to compare with edges (directed or undirected) for all vertices (node), and needs to know what is the next vertex from the remaining vertex set from either Graph model GM or Graph data GD to be used for backtracking search, and needs to know what are all the isomorphic mapping between vertex from Graph model GM and Graph data GD to be printed once one instance of sub graph isomorphism is found. The previous statement mentioned three important details that defined the purpose of creating a new type of object (data structure) to store information. Also, a data structure is required to store all information from the Graph text file.

First, I need to have an object that behaves like a mathematical vertex set that contains all the vertex from either Graph Model GM or Graph Data GD. Second, I need to have an object that behaves like a mathematical isomorphic set that contains all information of the isomorphism mapping between Graph model and Graph data, that is, i.e. Vertex 1 in Graph Model GM is mapped to Vertex a in Graph Data GD. Lastly, I need to have a data structure to store information of edges between vertex (node), which I call adjacency matrix. Matrix is more towards of what kind of structure to be used.

Decision of what data structure to be implemented

Next, I will discuss the factors in efficiency and complexity of operation of each kind of object that decides what data structure to be implemented. Most importantly, I want to have an extremely fast access to all the data in the respective objects to be created. My goal is have each operation's complexity to be O(1). For this reason, **I choose to have array based implementation in three of the above discussed data structure to be created.**

Naming the Data Structure

The naming convention of each data structure is based on camelCase. For the vertex set to contain vertices from either Graph Model GM or Graph Model GD is named as **VertexSet**. For the isomorphic set to contain all isomorphic mapping between GM and GD is named as **IsomorphicSet**. For the adjacency matrix is named as **adjMatrixGM** and **adjMatrixGD**. For data structure to store information after reading the graph text file is called **GraphData**.

Organization of files

- src [folder]
    - Match.class
        - GraphData.class [Outer class within Match.class]
    - VertexSet.class
    - IsomorphicSet.class

Encapsulation

Since GraphData is only used by Match once, it will best to encapsulate GraphData.class within Match.class.

Search Algorithm

The algorithm I used is based on the pseudo-code given in the Friday May 17th lecture, but I made changes to optimize the complexity of Search Algorithm.

Implementation of Data Structure

1. **GraphData** consists of many integers and 1 dimesional integer array
    a. Create a new GraphData object with all the necessary information via constructor. Important to note that this object is not in charge of reading text file. Class that create the GraphData object is response for reading the text file.
    b. The only information not store is the string name which is irrelevant for the purpose of the search for sub graph isomorphism
    c. All data stored in this object are used later in VertexSet, IsomorphicSet, and Adjacency Matrix primarily for the purpose to initialize all these object with the appropriate size.
    d. Wherever mentioned GM refers to Graph Model; GM is refer in this readme context as abstraction of Model Graph of undirected vertices; whereas, GM is refer as GraphData Object in the .java code.
    e. Wherever mentioned GD refers to Graph Model; GD is refer in this readme context as abstraction of Data Graph of undirected vertices; whereas, GD is refer as GraphData Object in the .java code.

2. **VertexSet** primarily use of 1 dimensional integer array.
    a. Able to initialize all vertices of Graph GM/GD based on the Gx.numNodes
    b. Since in the search recursive method always passed in copy of VertexSet with one less vertex, namely, v (vertex v has an isomorphic mapping to the Gx'), able to create a new clone object with all remaining vertices from the original object without vertex v.
    c. Able to retrieve value of vertex v by providing the index where the vertex is store in the array

3. **IsomorphicSet** primarily use of 2 dimensional integer array.
    a. Have 2 rows and x columns (depending on GM.numEdges)
    b. Row 0 belongs to all vertices in the VertexSet VM from Graph model GM
    c. Row 1 belongs to all vertices in the VertexSet VD from Graph model GD
    d. The use of integer pointer (to add vertex vm and vertex vd to the correct mapping location in array whenever call) to have O(1) add operation to the mapping. The integer pointer in IsomorphicSet is called curIndexPtr.
    e. All variables are made private except curIndexPtr for easy of use, and also prevents other classes to change the mapping.
    f. Able to initialize the size of mapping array that containing isomorphic mapping between vertex vm and vertex vd based on the Gx.numEdges.
    g. Since in the search recursive method always passed in copy of the original IsomorphicSet h, IsomorphicSet must be able to create a new clone object that copy all information from the original.
    h. Since IsomorphicSet doesn't have any check in place to determine what mapping is valid, so IsomorphicSet needs to have addMapping and removeMapping method to add and remove the pair vertex (vm, vd).
    i. Since IsomorphicSet doesn't have any check in place to determine what mapping is valid, IsomorphicSet need to allow other classes to access the value of this vertex of a particular type of Graph (GM or GD) via a getMapValue Method. Type (0 for VM, 1 for VD) and index is where the location of vertex is stored.
    j. Lastly, since IsomorphicSet store all isomorphic mapping between graphs, IsomorphicSet needs to tell the recursive search when to stop, that is, the flag method isComplete.

4. **Adjacency Matrix** primarily use a 2 dimensional integer array
    a. Initialize by GM.numNodes
    b. Row store undirected edges information to other vertex except itself
    c. Column represent which vertex the undirected edge connects to this vertex (row).
    d. 1 represent there exists an undirected edge
    e. 0 represent there is no undirected edge
    f. Although undirected edges information is only stored in the upper triangular of the Adjacency Matrix, but storing the redundant information in the lower triangular of the Adjacency Matrix helps to access information faster that is required in determining what mapping is valid or not.