Gebze Technical University

CSE222 – Data Structures and Algorithms

HW8 – Question2

Subject : Graph implementation with 2D Linked-List.

Nevzat Seferoglu

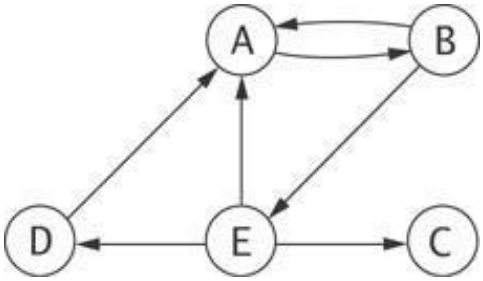
171044024

*Problem Description :

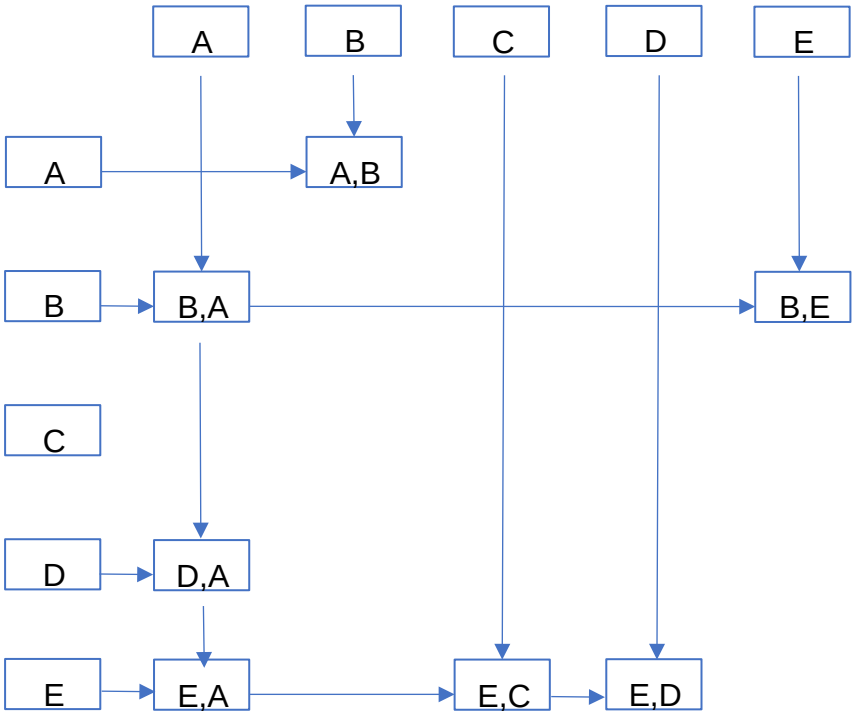
Extend the graph ADT defined in the book so that it includes the following operations.

- Deletion of an individual edge.
- Insertion and deletion of an individual vertex. Give proper definition of the operations. Note that if you can delete a node, node IDs cannot be from 0…(n-1) anymore. - Perform breadth-first search of the graph - Perform depth-first search of the graph.

Implement the extended graph ADT using 2-D linked-list structure. In 2-D linked-list structure, each node has two row links (**rprev** to row-predecessor and **rnext** to row-successor) and two column links (**cprev** to column-predecessor and **cnext** to column-successor). In our graph representation, each row linked-list represents adjacent vertices to a vertex and each column linked-list represents vertices adjacent to a vertex. Consider the example graph below;



2-D linked-list representation is given below. Note that predecessor links are not shown for simplicity. The first column represents the source vertices (in ascending order) and the first row represents the destination vertices (in ascending order). Each edge is represented by a node which belong to two linked lists; a row linked-list and a column linked-list.



Write a program which tests your implementation using a randomly created directed and undirected graphs. You may create random adjacency matrices and convert it to your representation. You can also use random vertices and edges to test your operations.

*Solution Approach :

As described in the problem definition , I had to implement a graph with 2D linked list. At the beginning I thought that I need to implement a new linked-list data structure which's node has 4 different directions reference. But after the lots of search and thinking , I concluded that I just needed to implement a single Node class as an inner class which has 4 different direction reference.

```
/**
 * Raw vertex keeper.
 */
private Node[] firstR;

/**
 * Column vertex keeper.
 */
private Node[] firstC;
```

*Test Cases:

*Directed Graph Test:

Test ID	Scenario	Test Data	Expected Result	Actual Result	Pass / Fail
T01	Creating graph with constructor.		Graph created with size 10	Expected result has been occurred.	Pass

T02	`insert( new Edge(0 , 1) )`
	`insert( new Edge(1 , 0) )`
	`insert( new Edge(1 , 4) )`
	`insert( new Edge(3 , 0) )`
	`insert( new Edge(4 , 0) )`
	`insert( new Edge(4 , 3) )`
	`insert( new Edge(4 , 2) )`
		0,1,2,3,4 Vertexes.	Given edge will be created.	Expected result has been occurred.	Pass

T03	`isEdge( 0 , 1 ) = true`
	`isEdge( 1 , 0 ) = true`
	`isEdge( 4 , 3 ) = true`
	`isEdge( 3 , 4 ) = false`
		Test whether given edges exist.	`isEdge( 0 , 1 ) = true`
			`isEdge( 1 , 0 ) = true`
			`isEdge( 4 , 3 ) = true`
			`isEdge( 3 , 4 ) = false`
				Expected result has been occurred.	Pass

T04	getEdge(0 , 1) getEdge(1 , 0) getEdge(4 , 3) getEdge(3 , 4)	return given edges that is already existing.	(source : 0, dest : 1) (source : 1, dest : 0) (source : 4, dest : 3) null	Expected result has been occurred.	Pass
T05	Iterator-Test		All operation has been done successfully.	Expected result has been occurred.	Pass

T06	insertVertex()	indexes	Number of vertex : 5. insertVertex() = 5 can be used as new vertex. Number of vertex : 6.	Expected result has been occurred.	Pass

T07	insertVertex()	indexes	Number of vertex : 6. insertVertex() = 6 can be used as new vertex. Number of vertex : 7.	Expected result has been occurred.	Pass

T08	insertVertex()	indexes	Number of vertex : 7. insertVertex() = 7 can be used as new vertex. Number of vertex : 8.	Expected result has been occurred.	Pass
T09	insert(0 , 1) insert(0 , 2) insert(0 , 3)	edges	New vertexes inserted to graph.	Expected result has been occurred.	Pass
T10	insert(2 , 0) insert(2 , 1) insert(2 , 3)	edges	New vertexes inserted to graph.	Expected result has been occurred.	
T11	insert(3 , 0) insert(3 , 1) insert(3 , 2)	edges	New vertexes inserted to graph.	Expected result has been occurred.	Pass

*Undirected Graph Test:

T12	`insert( new Edge(0 , 1 ))`
	`insert( new Edge(0 , 2 ))`
	`insert( new Edge(2 , 3 ))`
	`insert( new Edge(2 , 1 ))`
	`insert( new Edge(3 , 1 ))`
			Given edges will be inserted.	Expected result has been occurred.	Pass

*Note:

Due to lack of time , I could not show the test of exception case. All exception cases are **handled properly**. I showed vertexes as an **Integer**. Integer can be encoded as other structures. **BreadthFirst** and **DepthFirst** search are tested in the code properly.

*Dependency:

Graph cannot contain loop. Therefore , destination and source cannot be equivalent at the same time.

*Class Diagram:

*Attached to the assignment file as png.

*Running Command and Result:

*Attached to the assignment file as txt.