CMPT 280

Tutorial: Graphs in lib280

Mark G. Eramian

University of Saskatchewan

Based on notes by G. Cheston and J. P. Tremblay

Implementing Graphs

What classes do we need?

- Would like to use custom nodes and custom edges, with a generic default.
- Would like have notion of "position" (current node).
- Would like to have notion of "edge position" (current edge of current node).
- Would like to allow either an adjacency list or adjacency matrix representation of edges with some common functionality between these.

Implementing Graphs

`Vertex280` Generic vertex with basic functionality.

Implementing Graphs

`Vertex280` Generic vertex with basic functionality.

`Edge280<V>` An edge between two vertices of type V.

Implementing Graphs

`Vertex280` Generic vertex with basic functionality.

`Edge280<V>` An edge between two vertices of type V.

`Graph280<V,E>` Abstract class that contains most of the functionality for vertex management. Methods for edge management are abstract.

Implementing Graphs

`Vertex280` Generic vertex with basic functionality.

`Edge280<V>` An edge between two vertices of type V.

`Graph280<V,E>` Abstract class that contains most of the functionality for vertex management. Methods for edge management are abstract.

`GraphWithCursors280<V,E>` Abstract class subclass of `Graph280` that adds vertex and edge cursor methods.

Implementing Graphs

`Vertex280` Generic vertex with basic functionality.

`Edge280<V>` An edge between two vertices of type V.

`Graph280<V,E>` Abstract class that contains most of the functionality for vertex management. Methods for edge management are abstract.

`GraphWithCursors280<V,E>` Abstract class subclass of `Graph280` that adds vertex and edge cursor methods.

`GraphPosition<V,E>` A class that stores all of the information about cursors in a `GraphWithCursors280` instance.

Implementing Graphs

`Vertex280` Generic vertex with basic functionality.

`Edge280<V>` An edge between two vertices of type V.

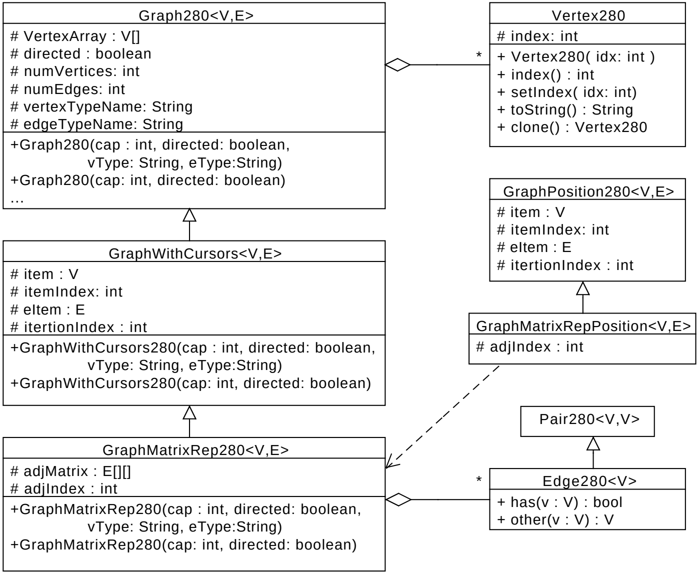
`Graph280<V,E>` Abstract class that contains most of the functionality for vertex management. Methods for edge management are abstract.

`GraphWithCursors280<V,E>` Abstract class subclass of `Graph280` that adds vertex and edge cursor methods.

`GraphPosition<V,E>` A class that stores all of the information about cursors in a `GraphWithCursors280` instance.

`GraphMatrixRep280<V,E>` Descendent of GraphWithCursors280 with edges stored as adjacency matrix.

Implementing Graphs



Vertex280

Instance Variables

Each vertex is associated with an integer index corresponding to its position in the vertex array in the graph to which it belongs.

`index`
: Index of the vertex (`int`).

Vertex280

Methods

Each vertex is associated with an integer index corresponding to its position in `vertexArray`.

`Vertex280(int idx)`
: Create a new vertex with index `idx`.

`int index()`
: Get the index of the vertex.

`setIndex(int idx)`
: Set the index of the vertex.

Graph280

Instance Variables

`vertexArray`
: Array of vertices in the graph (`V[]`).

`directed`
: True if the graph is directed, false otherwise (`boolean`).

`numVertices`
: Number of vertices in the graph (`int`).

`numEdges`
: Number of edges in the graph (`int`).

`vertexTypeName`
: The name of the class representing edges (`String`).

`edgeTypeName`
: The name of the class representing edges (`String`).

Graph280

`Graph280` implements `Container280` because `vertexArray` has a maximum capacity.

Type Parmeters:

V	Vertex type.
E	Edge Type.

Graph280
Public Methods

`int numVertices()`	Number of vertices.
`int numEdges()`	Number of edges.
`boolean directed()`	Are edges directed?
`boolean capacity()`	Max # of vertices (`vertexArray.length`).
`V vertex(int i)`	Get the vertex with index `i`.
`addVertex(int idx)`	Add a vertex with index `idx`.
`ensureVertices(int maxIdx)`	Add all vertices with index between 1 and `maxIdx` that are not already in the graph.

Graph280

Public Methods

`addEdge(int src, int dst)`
: Add an edge from vertex with index `src` to vertex with index `dst`.

`clear()`
: Remove all nodes and edges.

`isEmpty()`
: True if there are 0 vertices.

`isFull()`
: True if number of vertices is equal to the capacity.

Graph280

Abstract Public Methods

Methods dealing with edges have to be abstract since we don't know what the edge representation is.

`boolean isAdjacent(int src, int dst)`
: Is the vertex with index `src` adjacent to the vertex with index `dst`?

`boolean isAdjacent(V v1, V v2)`
: Is the vertex `v1` adjacent to the vertex `v2`?

`addEdge(V v1, V v2)`
: Add an edge from `v1` to `v2`.

Graph280

Create a graph programmatically:

```
// Create undirected graph with up to 10 vertices.
GraphMatrixRep280<Vertex280, Edge280<Vertex280>> G =
    new GraphMatrixRep280<Vertex280, Edge280<Vertex280>>(10, false);

// Add vertices 1 through 3.
G.ensureVertices(3);

// Add edge (1, 2)
G.addEdge(1, 2);

// Add edge (2,3)
G.addEdge(2,3);

// Add edge (3,1)
G.addEdge(3,1);

// Display edge list.
System.out.println(G);
```

Output:

```
3   undirected
1 :  2 3
2 :  1 3
3 :  1 2
```

Graph280

Note difference when the graph is directed:

```java
// Create undirected graph with up to 10 vertices.
GraphMatrixRep280<Vertex280, Edge280<Vertex280>> G =
    new GraphMatrixRep280<Vertex280, Edge280<Vertex280>>(10,true);

// Add vertices 1 through 3.
G.ensureVertices(3);

// Add edge (1, 2)
G.addEdge(1, 2);

// Add edge (2,3)
G.addEdge(2,3);

// Add edge (3,1)
G.addEdge(3,1);

// Display edge list.
System.out.println(G);
```

Output:

```
3   directed
1 :  2
2 :  3
3 :  1
```

GraphWithCursors280

- Descendent of `Graph280`.
- Same type parameters `V` and `E`.
- Same constructors.
- Adds vertex and edge cursors, but still leaves edge-related methods abstract.

GraphWithCursors280

Instance Variables

`item`
: Vertex at the vertex cursor. (`V`).

`itemIndex`
: Index of the vertex at the vertex cursor. (`int`).

`eItem`
: Current edge incident on vertex with index `iterationIndex` (`E`).

`iterationIndex`
: Index of vertex over whose edges the edge cursor `eItem` is iterating (i.e. the index of `item`). (`int`).

`adjIndex`
: The index of the destination vertex of the edge `eItem`.

GraphWithCursors280

Public Methods

`boolean after()`
Is the vertex cursor after the last vertex?

`boolean before()`
Is the vertex cursor before the first vertex?

`goAfter()`
Position vertex cursor after the end.

`goBefore()`
Position vertex cursor before the first vertex.

`goFirst()`
Position the vertex cursor at the first vertex.

`GraphWithCursors280`

Public Methods

`goForth()`
: Advance vertex cursor.

`V item()`
: Vertex at the vertex cursor.

`boolean itemExists()`
: Is there a vertex at the vertex cursor?

`E eItem()`
: The edge at the edge cursor.

`boolean eItemExists()`
: Is the edge cursor positioned at an edge?

GraphWithCursors280

Public Methods

`int eIterationIndex()`
: Index of vertex whose edges are being iterated by the edge cursor.

`V eItemAdjacentVertex()`
: Destination vertex of the current edge (source vertex is always the vertex with index `eIterationIndex()`).

`int eItemAdjacentIndex()`
: Index of destination vertex of current edge.

`int itemIndex()`
: Index of the vertex at the vertex cursor.

`GraphWithCursors280`

Public Methods

`goVertex(V newV)`
    Position vertex cursor at the vertex `newV`.

`goIndex(int idx)`
    Position vertex cursor at the vertex with index `idx`.

`deleteItem()`
    Delete the vertex at the vertex cursor.

`initGraphFromFile(String file)`
    Load graph from a data file.

`GraphWithCursors280`

Abstract Public Methods

These methods must be abstract because they depend on how the edges are stored.

`eSearch(V u, V v)`
: Find the (directed) edge with endpoints `u` and `v`

`deleteEItem()`
: Delete the edge at the edge cursor and move the cursor to the next edge (if it exists).

`eGoFirst(V v)`
: Position edge cursor at first edge incident on vertex `v`.

GraphWithCursors280

Abstract Public Methods

These methods must be abstract because they depend on how the edges are stored.

`eGoForth()`
: Advance to the next edge in the edges outgoing from vertex with index `eIterationIndex()`.

`boolean eAfter()`
: Is the edge cursor after the last edge?

GraphWithCursors280

Iterating over all Vertices.

```java
// Create undirected graph with up to 10 vertices.
GraphMatrixRep280<Vertex280, Edge280<Vertex280>> G =
    new GraphMatrixRep280<Vertex280, Edge280<Vertex280>>(10,true);

// ... Construct directed graph ...

// Start at first vertex
G.goFirst()
while(!G.after()) {
    // Get the current vertex.
    Vertex280 currentVertex = G.item();

    // ... Do stuff using currentVertex...

    // Advance to next vertex
    G.goForth();
}
```

GraphWithCursors280

Iterating over all Edges.

For each vertex, iterate edges:

```
// Create undirected graph with up to 10 vertices.
GraphMatrixRep280<Vertex280, Edge280<Vertex280>> G =
    new GraphMatrixRep280<Vertex280, Edge280<Vertex280>>(10,true);

// ... Construct directed graph ...

// Start at first vertex
G.goFirst()
while(!G.after()) {
    // Iterate edges of current vertex.
    G.eGoFirst(G.item());
    while(!G.eAfter()) {
        Edge280<Vertex280> currentEdge = G.eItem();

        // ... Do stuff using the current edge ...

        G.eGoForth();
    }

    // Advance to next vertex
    G.goForth();
}
```

`GraphPosition280`

- Stores information about cursors in a `GraphWithCursors280` instance.
- A class that takes a copy of the cursor-related instance variables.
- Instance variables are public.

GraphPosition280

Public Instance Variables

item

    Vertex at the vertex cursor. (V).

itemIndex

    Index of the vertex at the vertex cursor. (int).

eItem

    Current edge with source vertex iterationIndex (E).

iterationIndex

    Index of vertex over whose edges the edge cursor eItem is iterating. (int).

adjIndex

    The index of the destination vertex of the edge eItem.

GraphMatrixRep280

- Descendent of `GraphWithCursors280`.
- Implements abstract methods.
- Contains adjacency matrix data structure for edges.

GraphMatrixRep280

Instance Variables

`adjMatrix`

The adjacency matrix. Each entry stores edge object. (`E[][]`).

GraphWithCursors280

Public Methods

- No new methods are defined.
- Abstract methods from `Graph280` and `GraphWithCursors280` are implemented.

Reading Graphs from a File

We define a graph file format:

- First line contains number of node n, nodes in the graph are numbered continuously from 1 to n.
- Each subsequent line contains two vertex indices indicating the source and destination of one edge.

Example graph file (`testgraph.gra` in `lib280-asn5.graph` package):

```
10
1 2
2 3
3 1
3 5
9 8
9 7
6 4
4 8
5 6
1 1
```

Reading Graphs from a File

```java
// Code fragment demonstrating reading a graph from a file

// Create undirected graph with up to 10 vertices.
GraphMatrixRep280<Vertex280, Edge280<Vertex280>> G =
    new GraphMatrixRep280<Vertex280, Edge280<Vertex280>>(10,true);

     G.initGraphFromFile("src/lib280/graph/testgraph.gra");
```

Using Custom Nodes and Edges

Suppose we want to attach data to an vertex.

```
// Descendent of default edge, Edge280<V>
public class MyVertex extends Vertex280 {

    // Set when a vertex has been reached in a BFS or DFS
    private boolean reached;

    public MyVertex(int idx) {
        super(idx);
    }

    public boolean isReached() {
        return reached;
    }

    public void setReached(boolean r) {
        this.reached = r;
    }
}
```

Using Custom Nodes and Edges

DFT from vertex s

```
public void dft(int startVertexIdx) {

for(int i=0; i < this.numVertices; i++)
      vertexArray[i].setReached(false);
      depthFirstTraverse(s);
}


private void depthFirstTraverse(int v) {
      vertexArray[v].setReached(true);

      // do stuff for node v

      for(this.eGoFirst(vertexArray[v]); this.eItemExists(); this.eGoForth()) {
            if(!this.eItem().secondItem().isReached())
                  depthFirstTraverse(this.eItem().index())
      }
}
```

Using Custom Nodes and Edges

DFT from vertex s

```
// Code fragment demonstrating reading a graph from a file

// Create undirected graph with up to 10 vertices.
GraphMatrixRep280<Vertex280, Edge280<Vertex280>> G =
    new GraphMatrixRep280<MyVertex, Edge280<Vertex280>>(10,true, "MyVertex",
          "lib280.graph.Edge280");

G.initGraphFromFile("src/lib280/graph/testgraph.gra");

G.dft(1);
```