Administrative Details

- Six classes left!
  - One real lab, one optional lab, one final exam (self-scheduled)
  - Darwin tourney – Fri or Mon…TBD
- Darwin lab and creature due today
- Any questions?
- Lab this week: Exam Scheduling
  - Focuses on using and manipulating graphs
- You’ll get back Lab 8 on Wed in lab
- Midterms are partially graded

Last Time

- Started talking about graphs
  - Last “major” data structure of the semester!
  - Defined key graph terminology

Today’s Outline

- Continue talking about graphs
  - Implementing graphs
    - GraphMatrix
    - GraphList

Reachability

- What does it mean for a dest vertex to be reachable from a src vertex?
- Path exists from src to dest
- Example
  - Is B reachable from A?
- How do we implement

```java
public boolean reachable(Graph<V,E> g, V src, V dst) {
  g.reset();
  visitReachableFrom(g, src);
  return g.isVisited(dst);
}
```

Use “visited” flag on vertices to help determine reachability.
Implementing Graphs: Graph Interface

- You'll gain a better understanding of the Graph interface in lab this week.
- What is it used for?
  1. Creating graphs
  2. Adding nodes/edges
  3. Testing connectivity
  4. Traversing nodes/edges (iterators)

Graph Interface Methods

- boolean visit(V vertexLabel)
  - Mark vertex as "visited" and return previous value of visited flag
- void visitEdge(Edge<V,E> e)
  - Mark edge as "visited"
- boolean isVisited(V vtx), boolean isVisitedEdge(Edge<V,E> e)
  - Returns true iff vertex/edge has been visited
- Iterator<V> neighbors(V vtx1)
  - Get iterator for all neighbors to vtx1
  - For directed graphs, out-edges only
- Iterator<V> iterator()
  - Get vertex iterator
- void reset()
  - Remove visited flags for all nodes/edges

Example Graph/Edge Usage

- You'll see one example in lab this week...

For our map with edge labels:

```java
Graph<String, Integer> g = new GraphMatrixDirected<String, Integer>();
g.add("SF");
g.add("Dallas");
g.addEdge("SF", "Dallas", new Integer(1468));
```

Edge<String, Integer> SFtoDallas = g.getEdge("SF", "Dallas");
int dist = (SFtoDallas.label()).intValue();

Graph Interface Methods

- void add(V vtx), V remove(V vtx)
  - Add/remove vertex to graph
- void addEdge(V vtx1, V vtx2, E edgeLabel), E removeEdge(V vtx1, V vtx2)
  - Add/remove edge between vtx1 and vtx2
- boolean containsEdge(V vtx1, V vtx2)
  - Returns true iff there is an edge between vtx1 and vtx2
- Edge<V,E> getEdge(V vtx1, V vtx2)
  - Returns edge between vtx1 and vtx2
- void clear()
  - Remove all nodes (and edges) from graph

Edge Class

- Graph edges are defined in their own public class
  - Edge<V,E>( V vtx1, V vtx2, E label, boolean directed)
  - Construct a (possibly directed) edge between two labeled vertices (vtx1 -> vtx2)
- Useful methods:
  - label(), here(), there()
  - setLabel(), isVisited(), isDirected()

Representing Graphs

- Two options
  - Option 1: GraphMatrix (Directed and Undirected)
  - Option 2: GraphList (Directed and Undirected)
- We're going to look at GraphMatrix first
  - Represent graph as a vertex adjacency matrix
- Challenge: How to represent vertices and edges?
  - Solution: Maintain a dictionary that translates a "normal" vertex label into an index in matrix
GraphMatrix

- Abstract class – partially implements Graph

```java
public abstract class GraphMatrix<V,E> implements Graph<V,E>
```

- Instance variables

```java
protected int size; // max size of matrix
protected Object data[]];[][]; // matrix of edges
protected Map<V, GMV<V>> dict; // labels -> vertices
protected List<Integer> freeList; // avail indices
protected boolean directed;
```

GraphMatrix Constructor

```java
protected GraphMatrix(int size, boolean dir) {
  this.size = size; // set maximum size
directed = dir; // fix direction of edges

  // the following constructs a size x size matrix
  // (the 'Objects' will be 'Edges')
  // (can't use generics with arrays!)
data = new Object[size][size];

  // label to index translation table
dict = new HashMap<V,GraphMatrixVertex<V>>().size();
  // put all indices in the free list
  freeList = new SinglyLinkedList<Integer>();
  for (int row = size-1; row >= 0; row--)
    freeList.add(new Integer(row));
}
```

Vertex and GraphMatrixVertex

- We already looked at the Edge class
- Now we need to define a Vertex class
  - Unlike the Edge class, Vertex class is not public
- Useful Vertex methods:
  - V label(), boolean visit(), void reset()
- GraphMatrixVertex class adds one more useful attribute to Vertex class
  - Index of node (int) in adjacency matrix
  - Why do we only need one int to represent index!

GraphMatrixDirected

- Represent graph as a vertex adjacency matrix
- GraphMatrixUndirected is very similar...
- How do we implement GraphMatrixDirected?
  - Note: We are not going to go over every detail of GraphMatrixDirected!
  - Today: add and addEdge (maybe remove...)
  - Please read Ch 16 for complete implementation details...

GraphMatrixDirected

```java
public GraphMatrixDirected(int size) {
  // pre: size > 0
  // post: constructs an empty graph that may be
  // expanded to at most size vertices. Graph
  // is directed if dir true and undirected
  // otherwise

  // call GraphMatrix constructor
  super(size,true);
}
```

```java
public void add(V label) {
  if (dict.containsKey(label)) return;
  int row = freeList.removeFirst().intValue;
  dict.put(label, new GraphMatrixVertex<V>(label, row));
}
```

```java
public void addEdge(V vLabel1, V vLabel2, E label) {
  GraphMatrixVertex<V> vtx1,vtx2;
  vtx1 = dict.get(vLabel1);
  vtx2 = dict.get(vLabel2);
  Edge<V,E> e = new Edge<V,E>(vtx1.label(), vtx2.label(),
                             label, true);
  data[vtx1.index()][vtx2.index()] = e;
}
```