CSCI 136
Data Structures & Advanced Programming

Jeannie Albrecht

Lecture 32

May 7, 2014

Administrative Details

- Lab today: Exam Scheduling
  - Focuses on using and manipulating graphs
- Darwin tournament on Friday
  - I'll need your final creatures by tomorrow at noon!

Lab 10 Notes: Graphs in structure5

Lab 10 Notes: Using Graphs

- Create a new graph in structure5
  - GraphList, GraphListDirected, GraphListUndirected,
  - GraphMatrix, GraphMatrixDirected, GraphMatrixUndirected
  - Graph<V,E> dep = new GraphListUndirected<V,E>();

Lab 10 Notes: Greedy Algorithm

- Pick a starting vertex – add it to list
- Walk through other vertices in graph
  - If vertex is not connected to all other vertices in list, add it to list
- Remove all vertices in list from graph
- Repeat until all vertices are gone

Lab 10 Notes: Useful Graph Methods

- `void add(V label)`
  - add vertex to graph
- `void addEdge(V vtx1, V vtx2, E label)`
  - add edge between vtx1 and vtx2
- `Iterator<V> neighbors(V vtx1)`
  - Get iterator for all neighbors to vtx1
- `boolean isEmpty()`
  - Returns true iff graph is empty
- `Iterator<V> iterator()`
  - Get vertex iterator
- `V remove(V label)`
  - Remove a vertex from the graph
- `E removeEdge(V vLabel1, V vLabel2)`
  - Remove an edge from graph

Last Time

- Looked at GraphMatrix implementation
  - Define graph using an adjacency matrix
  - Matrix keeps track of edge weights

7

Recap: GraphMatrix

- Abstract class – partially implements Graph

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

- Instance variables

```
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;
```

8

Recap: GraphMatrix Constructor

```
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
   data = new Object[size][size];

   // label to index translation table
   dict = new Hashtable<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));
}
```

9

Recap: GraphMatrixDirected

- Constructor

```
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
   super(size, true);
}
```

10

Recap: GraphMatrixDirected

- add and addEdge

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

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;
}
```

11

Today's Outline

- Continue talking about graphs
  - Finish up GraphMatrix
  - Discuss GraphList

12

Removing

- Now let's look at removing vertices and edges in GraphMatrixDirected
  - `public V remove(V label)`
  - `public E removeEdge(V vlabel1, Vlabel2)`

13

Removing Edges

```
public E removeEdge(V vlabel1, Vlabel2) {
   // get indices
   int row = dict.get(vLabel1).index();
   int col = dict.get(vLabel2).index();
   // save old value
   Edge<V,E> e = (Edge<V,E>)data[row][col];
   // update matrix
   data[row][col] = null;
   if (e == null) return null;
   else return e.label(); // return old value
}
```

14

Removing Vertices
(same for GMD and GMU)

```
public V remove(V label) {
   // find and extract vertex
   GraphMatrixVertex<V> vert;
   vert = dict.remove(label);
   if (vert == null) return null;
   // remove vertex from matrix
   int index = vert.index();
   // clear row and column entries
   for (int row=0; row<size; row++) {
       data[row][index] = null;
       data[index][row] = null;
   }
   // add node index to free list
   freeList.add(new Integer(index));
   return vert.label();
}
```

15

Efficiency

- Assume map operations are O(1) (for now)
  - |E| = number of edges
  - |V| = number of vertices
- Runtime of add, addEdge, getEdge, removeEdge?
- Runtime of remove?
- Space usage?
- Most operations are fast, but we waste a lot of space
  - Can we do better?
  - YES! Don't store empty slots in array.

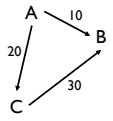
16

GraphList

- Rather than keep an adjacency matrix, maintain an *adjacency list of edges* at each vertex (only keep outgoing edges for directed graphs)
- Support both directed and undirected graphs (GraphListDirected, GraphListUndirected)

17

Example



Adjacency Matrix:

	A	B	C
A	-	10	20
B	-	-	-
C	-	30	-

Adjacency List:

A: [A->B,10] [A->C,20]

B:

C: [C->B,30]

18

GraphListDirected

- To implement GLD, we use the GraphListVertex (GLV) class
- GraphListVertex class
  - Maintain linked list of edges at each vertex
  - Instance vars: label, visited flag, linked list of edges
- GraphList abstract class
  - Instance vars:
    - Map<V,GraphListVertex<V,E>> dict; // label -> vertex
    - boolean directed; // is graph directed?
- How do we implement key GLD methods?
  - add, addEdge, removeEdge, remove

19

```
// add is same for GraphListDirected and GraphListUndirected,
// so implement in GraphList.java
public void add(V label) {
   // check to see if vertex already exists
   if (dict.containsKey(label)) return;
   GraphListVertex<V,E> v = new GraphListVertex<V,E>(label);
   dict.put(label,v);
}

// addEdge in GraphListDirected.java
// first vertex is source, second is destination
 public void addEdge(V vLabel1, V vLabel2, E label) {
   // first get the vertices
   GraphListVertex<V,E> v1 = dict.get(vLabel1);
   GraphListVertex<V,E> v2 = dict.get(vLabel2);
   // create the new edge
   Edge<V,E> e = new Edge<V,E>(v1.label(), v2.label(), label, true);
   // add edge only to source vertex linked list (aka adjacency list)
   v1.addEdge(e);
}
```

20

```
public V remove(V label) {
   //Get vertex out of map/dictionary
   GraphListVertex<V,E> v = dict.get(label);

   //Iterate over all vertex labels (called the map "keyset")
   Iterator<V> vi = iterator();
   while (vi.hasNext()) {
        //Get next vertex label in iterator
        V v2 = vi.next();

        //Skip over the vertex label we're removing
        //(Nodes don't have edges to themselves...)
        if (!label.equals(v2)) {
            //Remove all edges to "label"
            //If edge does not exist, removeEdge returns null
            removeEdge(v2,label);
        }
    }
    //Remove vertex from map
    dict.remove(label);
    return v.label();
}
```

21

```
public E removeEdge(V vLabel1, V vLabel2) {
   //Get vertices out of map
   GraphListVertex<V,E> v1 = dict.get(vLabel1);
   GraphListVertex<V,E> v2 = dict.get(vLabel2);

   //Create a "temporary" edge connecting two vertices
   Edge<V,E> e = new Edge<V,E>(v1.label(), v2.label(), null, true);

   //Remove edge from source vertex linked list
   e = v1.removeEdge(e);
   if (e == null) return null;
   else return e.label();
}
```

22

Efficiency Revisited

- Assume map operations are O(1) (for now)
  - |E| = number of edges
  - |V| = number of vertices
- Runtime of add, addEdge, getEdge, removeEdge, remove?
- Space usage?
- Conclusions
  - Matrix is better for dense graphs
  - List is better for sparse graphs
  - For graphs "in the middle" there is no clear winner

23

Efficiency

	Matrix	Sparse GraphList	Dense GraphList
add	O(1)	O(1)	O(1)
addEdge	O(1)	O(1)	O(1)
getEdge	O(1)	O(1)	O(|V|)
removeEdge	O(1)	O(1)	O(|V|)
remove	O(|V|)	O(|E|) = O(|V|)	O(|E|) = O($\|V\|^2$)
space	O($\|V\|^2$)	O(|E|) = O(|V|)	O(|E|) = O($\|V\|^2$)

24

Moving On…

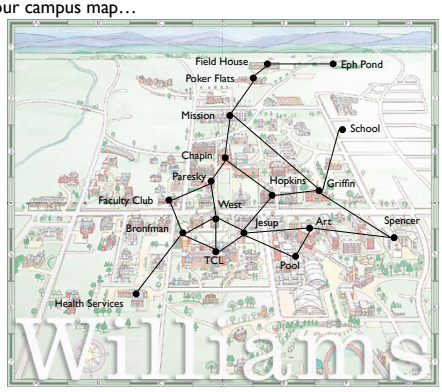
25

Graph Traversal Algorithms

- General concepts
  - Reachability
  - Cycle detection
  - Shortest path
  - …
  - (Read Ch 16!)

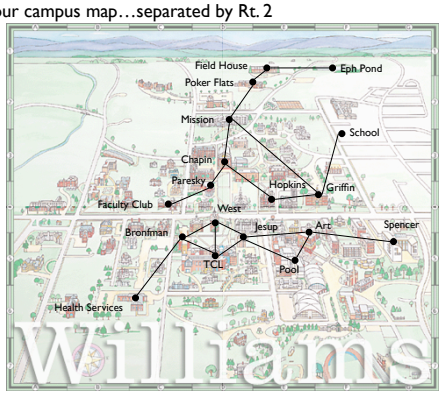
26

Recall our campus map…



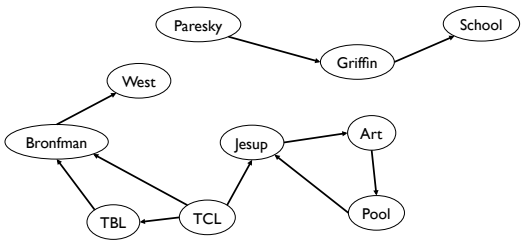
27

Recall our campus map…separated by Rt. 2



28

Suppose we applied a direction to our edges…



29

Reachability

- There are two ways to measure reachability in our graph
  - Depth-first search and breadth-first search
- How did we do DFS and BFS in trees?
- DFS uses a stack
  - Stack records path from src to current node
  - Like pre-order tree traversal with visited flags to only visit nodes once
  - Runtime: O(|E|)
- BFS uses a queue
  - Queue records nodes whose out edges have not been explored
  - Like level-order tree traversal
  - Runtime: O(|E|)

30

Depth-First Search

```
protected void reachableFrom(Graph<V,E> g, V src) {

  if (!g.visited(src)) {
    g.visit(src);

    Iterator<V> neighbors = g.neighbors(src);
    while (neighbors.hasNext()) {
      V next = neighbors.next();
      if (!g.visited(next))
        reachableFrom(g, next);
    }
  }

}
```

31