Administrative Details

- Lab 10 - Hexapawn
  - Any questions?

- Study Groups
  - A handful of people expressed interest
  - Let me know by 5pm and I’ll send a group email
Last Time

• Began Graphs
• Core concepts:
  • Adjacency Matrix
    • $|V| \times |V|$ matrix, each $(i,j)$ represents a possible edge
    • $\text{matrix}[i][j] = T$ if edge between $v_i, v_j$, $F$ otherwise
    • Good for dense graphs
  • Adjacency List
    • Vector of length $|V|$ 
    • Each index $i$ stores a list of vertex $i$’s outgoing edges
    • Good for sparse graphs
Today’s Outline

• Graph Implementations (details in book)
  • Graph Interface
  • Intuition for abstract graph classes

• Traversing graphs
  • BFS
  • DFS
Graph Implementation Approach

• Interface specifies general graph requirements
  • Graph<V,E>
    • Edge<V,E> (connects two vertices, public)
    • Vertex<E> (labeled vertex, non-public)
  • Abstract classes implement graph interface, provide common code
  • Strategy: provide all the code that can be written without assuming a graph’s “directedness”
Vertex Class

• Graph *vertices* are defined in a *non-public* class
  • `Vertex<E>` *(E label)*

• Just a label and a marker to track “visits” during traversals

• Methods
  
  ```java
  label(),
  visit(), isVisited(),
  reset(),
  equals()
  ```
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)

• Methods:
  
  label(), setLabel(),
  here(), there()
  visit(), isVisited(),
  reset(),
  isDirected()
Graph<V, E> Interface Methods

- void add(V label), V remove(V label)
  - Add/remove vertex to graph
- void addEdge(V vtx1Label, V vtx2Label, E edgeLabel),
  E removeEdge(V vtx1Label, V vtx2Label)
  - Add/remove edge between vtx1 and vtx2
- boolean containsEdge(V vtx1Label, V vtx2Label)
  - Returns true iff there is an edge between vtx1 and vtx2
- Edge getEdge(V vtx1Label, V vtx2Label)
  - Returns edge between vtx1 and vtx2
- void clear()
  - Remove all nodes (and edges) from graph
Graph<V, E> Interface Methods

- boolean visit(V vertexLabel)
  - Mark vertex as “visited” and return previous value of visited flag
- boolean visitEdge(Edge<V, E> edge)
  - Mark edge as “visited”
- boolean isVisited(V vtxLabel), boolean isVisitedEdge(Edge<V,E> e)
  - Returns true iff vertex/edge has been visited
- Iterator<V> neighbors(V vertexLabel)
  - Get iterator for all neighbors to vertex
  - For directed graphs, out-edges only
- Iterator<V> iterator()
  - Get vertex iterator (over all vertices in graph)
- void reset()
  - Remove visited flags for all nodes/edges
Abstract Graph Classes

- **GraphMatrix**\(< V, E >\)
  - GraphMatrixDirected\(< V, E >\)
  - GraphMatrixUndirected\(< V, E >\)

- **GraphList**\(< V, E >\)
  - GraphListDirected\(< V, E >\)
  - GraphListUnDirected\(< V, E >\)
GraphMatrix Intuition

- **NxN array of Edge objects**
- **Dictionary to map Vertex label to array index**
- `add(vLabel)`
  - Add mapping from label to index to dictionary
- `addEdge(v1Label, v2Label, edgeLabel)`
  - Check dictionary for v1, v2 indexes \((i, j)\)
  - Create edge and add it at matrix\([i][j]\)
- `getEdge(v1Label, v2Label)`
  - Check dictionary for v1, v2 indexes \((i, j)\)
  - Check matrix\([i][j]\) for edge

```
0 _ _ _ _ _
1 _ _ _ _ _
2 _ _ _ _ _
3 _ _ _ _ _
4 _ _ _ _ _
{   }
```
GraphList Intuition

• Each Vertex stores a list of Edge objects
• Dictionary to map Vertex label to Vertex
• add(vLabel)
  • Create a new Vertex with an empty list
  • Add label to vertex mapping to dictionary
• addEdge(v1Label, v2Label, edgeLabel)
  • Check dictionary for v1 and v2 vertexes
  • Create edge, add to appropriate vertex lists
• getEdge(v1Label, v2Label)
  • Check dictionary for v1 and v2 vertexes
  • Search v1 vertex list for target edge
Example Graph/Edge Usage

• For example map from last week with edge labels:

```java
Graph<String, Integer> g =
    new GraphMatrixDirected<String, Integer>(10);
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();
```

![Graph Diagram]

- SF
- 1468
- Dallas
Suppose we applied a direction to the edges in our campus map…
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: $O(|E|)$
DFS vs. BFS

DFS:

BFS:
DFS vs. BFS

DFS:

BFS:
DFS vs. BFS

DFS:

BFS:
DFS vs. BFS

DFS:

BFS:
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);
        }
    }
}
Find first unvisited neighbor of TCL…
Depth-First Search

Find first neighbor of TBL…
Find first neighbor of Bronfman…
Find first neighbor of West...none exist! Pop stack...
Find next neighbor of Bronf…none exist! Pop stack…
Find next neighbor of TBL…none exist! Pop stack…
Find next unvisited neighbor of TCL…
Find first neighbor of Jesup…
Depth-First Search

Find first neighbor of Art...
Find first unvisited neighbor of Pool…none exist!
Find next neighbor of Art... none exist!
Find next neighbor of Jesup…none exist!
Find next neighbor of TCL...none exist!
Depth-First Search

We’re done!
Depth-First Search

- Bronfman
  - West
    - TBL
    - TCL
- Jesup
  - Art
  - Pool
- Paresky
- Hollander
- Schapiro
Breadth-First Search

```java
void reachableFrom(Graph<V,E> g, V src) {
    Queue<V> todo = new QueueList<V>();
    g.visit(src);
    todo.enqueue(src);
    while (!todo.isEmpty()) {
        V node = todo.dequeue();
        Iterator<V> neighbors = g.neighbors(node);
        while (neighbors.hasNext()) {
            V next = neighbors.next();
            if (!g.visited(next)) {
                g.visit(next);
                todo.enqueue(next);
            }
        }
    }
}
```
Breadth-First Search

- Bronfman
  - TBL
  - TCL
- West
- Paresky
- Jesup
  - Art
  - Pool
- Hollander
- Schapiro
Find all unqueued/unvisited neighbors of TCL…
Breadth-First Search

Find all unqueued/unvisited neighbors of TBL…
Breadth-First Search

Find all unqueued/unvisited neighbors of Bronf…
Find all unqueued/unvisited neighbors of Jesup...
Breadth-First Search

Find all unqueued/unvisited neighbors of West…
Breadth-First Search

Find all unqueued/unvisited neighbors of Art…
Breadth-First Search

Find all unqueued/unvisited neighbors of Pool…
Breadth-First Search