CS200: Graphs

Rosen Ch. 9.1-9.4, 9.6, 10.4-10.5
Walls and Mirrors Ch. 14

Graphs

A collection of nodes and edges

What can this represent?
- A computer network
- Abstraction of a map
- Social network

Directed Graphs

A collection of nodes and directed edges

Sometimes we want to represent directionality:
- Unidirectional network connections
- One way streets
- The web

Graph Terminology

G=(V, E)
Vertices/Nodes
Edges

Two vertices are adjacent if they are connected by an edge
An edge is incident on two edges
Degree of a vertex: number of edges incident on it

Directed Graphs

Indegree: number of incoming edges
Outdegree: number of outgoing edges

Self loop (loop): an edge that connects a vertex to itself
Simple graph: no self loops and no two edges connect the same vertices
Multigraph: may have multiple edges connecting the same vertices
Pseudograph: multigraph with self-loops
Simple Facts About Graphs

- **Handshaking**: Let $G=(V,E)$ be an undirected graph. Then
  \[ \sum_{v \in V} \deg(v) = 2|E| \]
- An undirected graph has an even number of vertices of odd degree.
- Let $G=(V,E)$ be a graph with directed edges. Then
  \[ \sum_{v \in V} \deg^+(v) = \sum_{v \in V} \deg^-(v) = |E| \]

Special Graphs

- **Complete**
- **Hypercube**

Path

- **Path**: a sequence of edges
  - $(e_1, e_2, e_3)$ is a path of length 3 from $v_1$ to $v_4$
  - In a simple graph a path can be represented as a sequence of vertices
- **Cycle**: begins and ends at the same vertex

Six Degrees of Kevin Bacon

- Actor $x$ has a Kevin Bacon Number of $n$ if the shortest path between $x$ and Kevin Bacon has length $n$

Graph ADT

- Create
- Empty?
- Number of vertices?
- Number of edges?
- Edge exists between two vertices?
- Add a vertex
- Add an edge
- Delete a vertex (and any edges adjacent to it)
- Delete an edge
- Retrieve a vertex
Classes for a Weighted Undirected Graph

- **Edge:**
  - vertex1, vertex2
  - weight

- **Graph:**
  - organized collection of vertices and edges

Adjacency Matrix Implementation

- Directed graph: \( A[i][j] \) is 1 if there is an edge from vertex \( i \) to vertex \( j \).
- Undirected graph: \( A[i][j] \) is 1 if there is an edge between vertex \( i \) and vertex \( j \).
- Weighted graph: \( A[i][j] \) gives the weight of the edge.
- Useful for dense graphs.

Example

- Adjacency matrix?

```
  d e f g

  d 0 1 1 1
  e 1 0 0 1
  f 1 0 0 0
  g 1 1 0 0
```

Adjacency List Implementation

- Each vertex has a list of outgoing edges

```
  Index  Label
  0     A
  1     B
  2     C
  3     D
  4     E

  mapping of vertex labels to list of edges
```
Adjacency List Implementation

- For undirected graphs each edge appears twice. Why?
- Useful for sparse graphs.

Which Implementation?

- Which implementation best supports common Graph Operations:
  - Is there an edge between vertex i and vertex j?
  - Find all vertices adjacent to vertex j
  - Which best uses space?

Implementation: Edge Class

```java
class Edge {
    private Integer v, w; // vertices
    private int weight;
    public Edge(Integer first, Integer second, int edgeWeight) {
        v = first; w = second; weight = edgeWeight;
    }
    public int getWeight() {
        return weight;
    }
    public Integer getV() {
        return v;
    }
    public Integer getW() {
        return w;
    }
}
```

Implementation: Graph Class

```java
class Graph {
    private int numVertices;
    private int numEdges;
    private Vector<TreeMap<Integer, Integer>> adjList;
    public Graph(int n) {
        numVertices = n; numEdges = 0;
        adjList = new Vector<TreeMap<Integer, Integer>>() {
            for (int i = 0; i < numVertices; i++) {
                adjList.add(new TreeMap<Integer, Integer>());
            }
        }
    }
    public void addEdge(Integer v, Integer w, int weight) {
        // precondition: the vertices v and w must exist
        // postcondition: the edge (v,w) is part of the graph
        adjList.get(v).put(w, weight);
        adjList.get(w).put(v, weight);
        numEdges++;
    }
}
```