Graphs

Graph
Set of vertices connected pairwise by edges

- Formally: $G = (V, E)$, where $V$ is a set and $E \subseteq V \times V$
- $G = (V, E)$ is undirected if for all $v, w \in V$:
  $(v, w) \in E \iff (w, v) \in E$
- Otherwise graph is directed
Undirected graph
Directed graph
Why graphs?

Ideas?

- Surprisingly many practical applications
- Hundreds of graph algorithms known
- Challenging branch of computer science and discrete math
Applications: Maps
Applications: Electrical Circuits

Vertices represent diodes, transistors, capacitors, switches, etc., and edges represent wires connecting them.
Applications: Protein-protein interaction

Jeong et al, Nature Review — Genetics
Applications: Internet
Applications: Map of science clickstreams
Applications: The evolution of FCC lobbying coalitions, 2010
When to use Graphs?

Graphs are a good representation for any collection of objects and binary relation among them:

- The relationship in space of places or objects
- The ordering in time of events or activities
- Family relationships
- Taxonomy (e.g. animal - mammal - dog)
- Precedence (x must come before y)
- Conflict (x conflicts or is incompatible with y)
- Etc.
Graph terminology

- **Path**: Sequence of vertices connected by edges
- Two vertices are **connected** if there is a path between them
Graph Representation

▶ **Graph drawing:** Provides intuition about the structure of the graph
▶ Intuition can be misleading
## Graph API

```java
class Graph {
    Graph(int V) {
        // create an empty graph with V vertices
    }
    Graph(In in) {
        // create a graph from input stream
    }
    void addEdge(int v, int w) {
        // add an edge v-w
    }
    Iterable<Integer> adj(int v) {
        // vertices adjacent to v
    }
    int V() {
        // number of vertices
    }
    int E() {
        // number of edges
    }
    String toString() {
        // string representation
    }
}
```

```java
In in = new In(args[0]);
Graph G = new Graph(in);

for (int v = 0; v < G.V(); v++)
    for (int w : G.adj(v))
        StdOut.println(v + "-" + w);
```

- **In in = new In(args[0]);**
  - read graph from input stream
- **Graph G = new Graph(in);**
  - print out each edge (twice)
API: Sample client

Graph input format.

```
% java Test tinyG.txt
0-6
0-2
0-1
0-5
1-0
2-0
3-5
3-4
...
12-11
12-9
```

```
In in = new In(args[0]);
Graph G = new Graph(in);

for (int v = 0; v < G.V(); v++)
    for (int w : G.adj(v))
        StdOut.println(v + "-" + w);
```
Graph processing code

```java
public static int degree(Graph G, int v) {
    int degree = 0;
    for (int w : G.adj(v)) degree++;
    return degree;
}

public static int maxDegree(Graph G) {
    int max = 0;
    for (int v = 0; v < G.V(); v++)
        if (degree(G, v) > max)
            max = degree(G, v);
    return max;
}

public static double averageDegree(Graph G) {
    return 2.0 * G.E() / G.V();
}

public static int numberOfSelfLoops(Graph G) {
    int count = 0;
    for (int v = 0; v < G.V(); v++)
        for (int w : G.adj(v))
            if (v == w) count++;
    return count/2;  // each edge counted twice
}
```
Graph representation: Set of edges
Applications: Adjacency matrix data structure

1 - there is an edge between vertex $v_i$ and $v_j$, 0 - otherwise
Graph representation: Adjacency matrix

Maintain a 2D $V \times V$ boolean array

$adj[v][w] = adj[w][v] = true$
Graph representation: Adjacency list

Maintain a vertex-indexed array of lists
Java: Adjacency list

Maintain a vertex-indexed array of lists

```java
public class Graph {
    private final int V;
    private Bag<Integer>[] adj;

    public Graph(int V) {
        this.V = V;
        adj = (Bag<Integer>[]) new Bag[V];
        for (int v = 0; v < V; v++)
            adj[v] = new Bag<Integer>();
    }

    public void addEdge(int v, int w) {
        adj[v].add(w);
        adj[w].add(v);
    }

    public Iterable<Integer> adj(int v) {
        return adj[v];
    }
}
```

- Adjacency lists (using Bag data type)
- Create empty graph with V vertices
- Add edge v-w (parallel edges allowed)
- Iterator for vertices adjacent to v
In practice adjacency lists representation is used the most
- Algorithms based on iterating over vertices adjacent to \( v \)
- Real-world graphs tend to be sparse (large number of vertices, small average vertex degree)

<table>
<thead>
<tr>
<th>representation</th>
<th>space</th>
<th>add edge</th>
<th>edge between ( v ) and ( w )?</th>
<th>iterate over vertices adjacent to ( v )?</th>
</tr>
</thead>
<tbody>
<tr>
<td>list of edges</td>
<td>( E )</td>
<td>( I )</td>
<td>( E )</td>
<td>( E )</td>
</tr>
<tr>
<td>adjacency matrix</td>
<td>( V^2 )</td>
<td>( I^* )</td>
<td>( 1 )</td>
<td>( V )</td>
</tr>
<tr>
<td>adjacency lists</td>
<td>( E + V )</td>
<td>( I )</td>
<td>degree(( v ))</td>
<td>degree(( v ))</td>
</tr>
</tbody>
</table>
Class activity: Post your solution in slack to get attendance points
Class activity: Post your solution in slack to get attendance points

1. Identify and provide the indegree of all the vertices in the graph shown in previous slide. List the vertices using the ascending order.

2. Provide the list of edges representation of the graph shown in previous slide.

3. Provide the adjacency matrix representation of the graph shown in previous slide.

4. Provide the adjacency list representation of the graph shown in previous slide.