CS171 Introduction to Computer Science II

Graphs
Graphs

- Examples
- Definitions
- Implementation/Representation of graphs
Graphs

• Graphs: set of vertices connected pairwise by edges
• Interesting and useful structure
• Many practical applications
  – Maps
  – Web content
  – Schedules
  – Social networks
  – …
Delta Airlines Domestic Routes

Delta Airlines domestic routes

From Atlanta

From Memphis

ATL
LGA
DCA
EWR
LEN
MEM
DWF
LAX
SEA
DEN

11/27/2012
10 million Facebook friends

“Visualizing Friendships” by Paul Butler
Course Prerequisite Graph

Mathematics Major Requirements

Notes
- Underlined courses are required only for the BS.
- BA requires 332 or 321.
- 210 is required for 331 but not 337.

123 Calculus I

110 or 111 Foundations of Computing

124 Calculus II

210 Proof Techniques

231 Linear Algebra & Diff Eq

332 Abstract Algebra I

321 Advanced Analysis I

331 Combinatorics
- Or
- 337 Operations Research

329 Complex Analysis
- Or
- 357 Methods of Applied Math

232 Mathematical Modeling

321 Advanced Analysis II

332 Abstract Algebra II

341

334

337

340

Electives (2)
242 329 331 334 337 341 357 400

8/2010
Graphs

• Undirected graphs
  – simple connections

• Digraphs
  – each connection has a direction

• Edge-weighted graphs
  – each connection has an associated weight

• Edge-weighted digraphs
  – each connection has both a direction and a weight
Undirected Graphs

• A graph is a set of vertices and a collection of edges that each connect a pair of vertices
Graph representation

**Graph drawing.** Provides intuition about the structure of the graph.

**Caveat.** Intuition can be misleading.

*two drawings of the same graph*
**Glossary**

- When there is an edge connecting two vertices, the vertices are **adjacent to** one another and the edge is **incident to** both vertices.
- A **self-loop** is an edge that connects a vertex to itself.
- Two edges that connect the same pair of vertices are **parallel**.
- The **degree** of a vertex is the number of edges incident to the vertex, with loops counted twice.
- A **subgraph** is a subset of a graph’s edges (and associated vertices) that constitutes a graph.
A **path** in a graph is a sequence of vertices connected by edges
- A **simple path** is one with no repeated vertices
- A **cycle** is a path with at least one edge whose first and last vertices are the same
- A **simple cycle** is a cycle with no repeated edges or vertices (except the first and last vertices)
- The **length** of a path is its number of edges

One vertex is **connected to** another if there exists a path that contains both of them

A graph is **connected** if there is a path from every vertex to every other vertex in the graph
- A graph that is **not connected** consists of a set of connected **components**

An **acyclic** graph is a graph with no cycles.
Graphs

• Examples
• Definitions
• Implementation/Representation of graphs
public class Graph

Graph(int V)  
$\quad$ create an empty graph with $V$ vertices

Graph(In in)  
$\quad$ create a graph from input stream

void addEdge(int v, int w)  
$\quad$ add an edge $v$-$w$

Iterable<Integer> adj(int v)  
$\quad$ vertices adjacent to $v$

int V()  
$\quad$ number of vertices

int E()  
$\quad$ number of edges

String toString()  
$\quad$ string representation

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);

read graph from input stream
print out each edge (twice)
Graph API: sample client

Graph input format.

\[\text{tinyG.txt}\]

\[V\]
13
13
0 5
4 3
0 1
9 12
6 4
5 4
0 2
11 12
9 10
0 6
7 8
9 11
5 3

\[E\]

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

```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);
```

read graph from input stream

print out each edge (twice)
How to represent/implement a graph?

• **Space-efficient**
  – Accommodate types of graphs that likely to encounter

• **Time-efficient**
  – Add an edge
  – If there is edge between v and w
  – Iterate over vertices adjacent to v
  – ...
Real-world graphs

- Real-world graphs tend to be “sparse”
  - Huge number of vertices, small average vertex degree

Two graphs (V = 50)
Representation Options

- Edge list
- Adjacency matrix
- Adjacency lists
Set-of-edges graph representation

Maintain a list of the edges (linked list or array).
Adjacency-matrix graph representation

Maintain a two-dimensional $V$-by-$V$ boolean array; for each edge $v \rightarrow w$ in graph: $\text{adj}[v][w] = \text{adj}[w][v] = \text{true}$. 

[Adjacency matrix and graph diagram]

- Two entries for each edge.
Implementation

- http://algs4.cs.princeton.edu/41undirected/AdjMatrixGraph.java.html
Adjacency-list graph representation

Maintain vertex-indexed array of lists.
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];
    }
}
Implementation

• http://algs4.cs.princeton.edu/41undirected/Graph.java.html
Bag API

Main application. Adding items to a collection and iterating (when order doesn't matter).

```
public class Bag<Item> implements Iterable<Item>

Bag() create an empty bag
void add(Item x) insert a new item onto bag
int size() number of items in bag
Iterable<Item> iterator() iterator for all items in bag
```

Implementation. Stack (without pop) or queue (without dequeue).
**Graph representations**

**In practice.** Use adjacency-lists representation.
- Algorithms based on iterating over vertices adjacent to $v$.
- Real-world graphs tend to be “sparse.”

<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>1</td>
<td>$E$</td>
<td>$E$</td>
</tr>
<tr>
<td>adjacency matrix</td>
<td>$V^2$</td>
<td>1 *</td>
<td>1</td>
<td>$V$</td>
</tr>
<tr>
<td>adjacency lists</td>
<td>$E + V$</td>
<td>1</td>
<td>degree($v$)</td>
<td>degree($v$)</td>
</tr>
</tbody>
</table>

* *disallows parallel edges*