Graphs – ADTs and Implementations
Applications of Graphs

- Electronic circuits
  - Printed circuit board
  - Integrated circuit
- Transportation networks
  - Highway network
  - Flight network
- Computer networks
  - Local area network
  - Internet
  - Web
- Databases
  - Entity-relationship diagram
Outline

- Definitions
- Graph ADT
- Implementations
Outline

- Definitions
- Graph ADT
- Implementations
Edge Types

- Directed edge
  - ordered pair of vertices \((u,v)\)
  - first vertex \(u\) is the origin
  - second vertex \(v\) is the destination
  - e.g., a flight

- Undirected edge
  - unordered pair of vertices \((u,v)\)
  - e.g., a flight route

- Directed graph (Digraph)
  - all the edges are directed
  - e.g., route network

- Undirected graph
  - all the edges are undirected
  - e.g., flight network
Vertices and Edges

- End vertices (or endpoints) of an edge
  - U and V are the endpoints of a

- Edges incident on a vertex
  - a, d, and b are incident on V

- Adjacent vertices
  - U and V are adjacent

- Degree of a vertex
  - X has degree 5

- Parallel edges
  - h and i are parallel edges

- Self-loop
  - j is a self-loop
A graph is a pair \((V, E)\), where

- \(V\) is a set of nodes, called vertices
- \(E\) is a collection of pairs of vertices, called edges
- Vertices and edges are positions and store elements

Example:

- A vertex represents an airport and stores the three-letter airport code
- An edge represents a flight route between two airports and stores the mileage of the route
Paths

- Path
  - sequence of alternating vertices and edges
  - begins with a vertex
  - ends with a vertex
  - each edge is preceded and followed by its endpoints

- Simple path
  - path such that all its vertices and edges are distinct

- Examples
  - $P_1 = (V, b, X, h, Z)$ is a simple path
  - $P_2 = (U, c, W, e, X, g, Y, f, W, d, V)$ is a path that is not simple
Cycles

- Cycle
  - circular sequence of alternating vertices and edges
  - each edge is preceded and followed by its endpoints

- Simple cycle
  - cycle such that all its vertices and edges are distinct

- Examples
  - $C_1 = (V, b, X, g, Y, f, W, c, U, a, V)$ is a simple cycle
  - $C_2 = (U, c, W, e, X, g, Y, f, W, d, V, a, U)$ is a cycle that is not simple
A subgraph $S$ of a graph $G$ is a graph such that

- The vertices of $S$ are a subset of the vertices of $G$
- The edges of $S$ are a subset of the edges of $G$

A spanning subgraph of $G$ is a subgraph that contains all the vertices of $G$.
Connectivity

- A graph is connected if there is a path between every pair of vertices.
- A connected component of a graph G is a maximal connected subgraph of G.

Connected graph

Non connected graph with two connected components
Trees

A tree is a **connected**, **acyclic**, **undirected** graph.

A forest is a **set** of trees (not necessarily connected)
Spanning Trees

- A spanning tree of a connected graph is a spanning subgraph that is a tree.
- A spanning tree is not unique unless the graph is a tree.
- Spanning trees have applications to the design of communication networks.
- A spanning forest of a graph is a spanning subgraph that is a forest.
Reachability in Directed Graphs

- A node \( w \) is **reachable** from \( v \) if there is a directed path originating at \( v \) and terminating at \( w \).
  - E is reachable from B
  - B is not reachable from E
Properties

Property 1

\[ \sum_v \deg(v) = 2|E| \]

Proof: each edge is counted twice

Property 2

In an undirected graph with no self-loops and no multiple edges

\[ |E| \leq |V| (|V| - 1)/2 \]

Proof: each vertex has degree at most \(|V| - 1\)

Example

- \(|V| = 4\)
- \(|E| = 6\)
- \(\deg(v) = 3\)

Q: What is the bound for a digraph?

A: \(|E| \leq |V|(|V| - 1)\)
Outline

- Definitions
- Graph ADT
- Implementations
Main Methods of the (Undirected) Graph ADT

- Vertices and edges
  - are positions
  - store elements

- Accessor methods
  - endVertices(e): an array of the two endvertices of e
  - opposite(v, e): the vertex opposite to v on e
  - areAdjacent(v, w): true iff v and w are adjacent
  - replace(v, x): replace element at vertex v with x
  - replace(e, x): replace element at edge e with x

- Update methods
  - insertVertex(o): insert a vertex storing element o
  - insertEdge(v, w, o): insert an edge (v,w) storing element o
  - removeVertex(v): remove vertex v (and its incident edges)
  - removeEdge(e): remove edge e

- Iterator methods
  - incidentEdges(v): edges incident to v
  - vertices(): all vertices in the graph
  - edges(): all edges in the graph
Directed Graph ADT

- Additional methods:
  - isDirected(e): return true if e is a directed edge
  - insertDirectedEdge(v, w, o): insert and return a new directed edge with origin v and destination w, storing element o
END OF LECTURE
MARCH 25, 2014
Outline

- Definitions
- Graph ADT
- Implementations
Running Time of Graph Algorithms

- Running time often a function of both $|V|$ and $|E|$.

- For convenience, we sometimes drop the $|$ . $|$ in asymptotic notation, e.g. $O(V+E)$.
Implementing a Graph (Simplified)

Space complexity:  \( \Theta(V + E) \)  \( \Theta(V^2) \)

Time to find all neighbours of vertex \( u \):  \( \Theta(\text{degree}(u)) \)  \( \Theta(V) \)

Time to determine if \( (u, v) \in E \):  \( \Theta(\text{degree}(u)) \)  \( \Theta(1) \)
Representing Graphs (Details)

- Three basic methods
  - Edge List
  - Adjacency List
  - Adjacency Matrix
Edge List Structure

- **Vertex object**
  - element
  - reference to position in vertex sequence

- **Edge object**
  - element
  - origin vertex object
  - destination vertex object
  - reference to position in edge sequence

- **Vertex sequence**
  - sequence of vertex objects

- **Edge sequence**
  - sequence of edge objects
Adjacency List Structure

- Edge list structure
- Incidence sequence for each vertex
  - sequence of references to edge objects of incident edges
- Augmented edge objects
  - references to associated positions in incidence sequences of end vertices
Adjacency Matrix Structure

- Edge list structure
- Augmented vertex objects
  - Integer key (index) associated with vertex
- 2D-array adjacency array
  - Reference to edge object for adjacent vertices
  - Null for non-adjacent vertices
Asymptotic Performance
(assuming collections V and E represented as
doubly-linked lists)

- $|V|$ vertices, $|E|$ edges
- no parallel edges
- no self-loops
- Bounds are “big-Oh”

<table>
<thead>
<tr>
<th></th>
<th>Edge List</th>
<th>Adjacency List</th>
<th>Adjacency Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>$</td>
<td>V</td>
<td>+</td>
</tr>
<tr>
<td>incidentEdges($v$)</td>
<td>$</td>
<td>E</td>
<td>$</td>
</tr>
<tr>
<td>areAdjacent ($v$, $w$)</td>
<td>$</td>
<td>E</td>
<td>$</td>
</tr>
<tr>
<td>insertVertex($o$)</td>
<td>1</td>
<td>1</td>
<td>$</td>
</tr>
<tr>
<td>insertEdge($v$, $w$, $o$)</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>removeVertex($v$)</td>
<td>$</td>
<td>E</td>
<td>$</td>
</tr>
<tr>
<td>removeEdge($e$)</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Outline

- Definitions
- Graph ADT
- Implementations