COSC 2011 Section N

Thursday, May 3 2001

Overview

- Graphs
  - Review
    - ⋆Definitions, Terminology
  - ADT
  - Data Structures for Graphs

Graphs - Definitions: (1)

- *endVertices, endPoints*:
  - Two vertices joined by an edge.
- *origin*:
  - First endpoint of a directed edge.
- *destination*:
  - Second endpoint of a directed edge.

Graphs - Definitions: (2)

- *adjacent*:
  - Two vertices are "adjacent" if they are connected by the same edge.
- *incident*:
  - An edge is "incident" on a vertex if the vertex is an endpoint of the edge..
- *Outgoing edges of a vertex*:
  - Directed edges whose *origin* is that vertex.

Graphs - Definitions: (3)

- *Incoming edges of a vertex*:
  - Directed edges whose *destination* is that vertex.
- *Degree of a vertex v*:
  - Number of incident edges of *v.*
  - Denoted by *deg(v).*
- *in-degree of a vertex v*:
  - Number of incoming edges of *v.* Denoted by *indeg(v).*

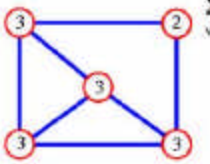
Graphs - Definitions: (4)

- *out-degree of a vertex v*:
  - Number of *outgoing* edges of *v*. Denoted by *outdeg(v).*

Graph Terminology

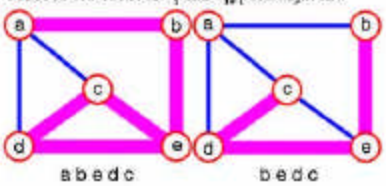
- adjacent vertices: connected by an edge
- degree (of a vertex): # of adjacent vertices

$\sum_{v \in V} \deg(v) = 2(\#\text{ edges})$



- Since adjacent vertices each count the adjoining edge, it will be counted twice

path: sequence of vertices $v_1, v_2, \ldots v_k$ such that consecutive vertices v_i and v_{i+1} are adjacent.



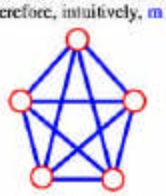
Connectivity

Let n = #vertices
m = #edges

- complete graph - all pairs of vertices are adjacent

$m = (1/2)\sum_{v \in V} \deg(v) = (1/2)\sum_{v \in V} (n-1) = n(n-1)/2$

- Each of the n vertices is incident to n - 1 edges, however, we would have counted each edge twice!!! Therefore, intuitively, $m = n(n-1)/2$.



n = 5
m = (5 * 4)/2 = 10

- Therefore, if a graph is *not* complete, $m < n(n-1)/2$

More Connectivity

n = #vertices
m = #edges

- For a tree m = n - 1

n = 5
m = 4

- If m < n - 1, G is not connected

n = 5
m = 3

Graphs – ADT:

- Many, many methods!
- Three Main Categories of Methods:
  - General methods
  - Methods dealing with directed edges.
  - Methods for updating and modifying graphs.

The Graph ADT

- The **Graph ADT** is a positional container whose positions are the vertices and the edges of the graph.

- size() Return the number of vertices plus the number of edges of G.
- isEmpty()
- elements()
- positions()
- swap()
- replaceElement()

Notation: Graph G; Vertices v, w; Edge e; Object o

- numVertices() Return the number of vertices of G.
- numEdges() Return the number of edges of G.
- vertices() Return an enumeration of the vertices of G.
- edges() Return an enumeration of the edges of G.

Graphs 15

The Graph ADT (contd.)

- directedEdges() Return an enumeration of all directed edges in G.
- undirectedEdges() Return an enumeration of all undirected edges in G.
- incidentEdges(v) Return an enumeration of all edges incident on v.
- inIncidentEdges(v) Return an enumeration of all the incoming edges to v.
- outIncidentEdges(v) Return an enumeration of all the outgoing edges from v.
- opposite(v, e) Return an endpoint of e distinct from v
- degree(v) Return the degree of v.
- inDegree(v) Return the in-degree of v.
- outDegree(v) Return the out-degree of v.

Graphs 16

More Methods ...

- adjacentVertices(v) Return an enumeration of the vertices adjacent to v.
- inAdjacentVertices(v) Return an enumeration of the vertices adjacent to v along incoming edges.
- outAdjacentVertices(v) Return an enumeration of the vertices adjacent to v along outgoing edges.
- areAdjacent(v,w) Return whether vertices v and w are adjacent.
- endVertices(e) Return an array of size 2 storing the end vertices of e.
- origin(e) Return the end vertex from which e leaves.
- destination(e) Return the end vertex at which e arrives.
- isDirected(e) Return true iff e is directed.

Graphs 17

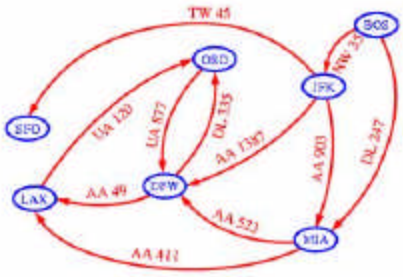
Update Methods

- makeUndirected(e)
  Set e to be an undirected edge.
- reverseDirection(e)
  Switch the origin and destination vertices of e.
- setDirectionFrom(e, v)
  Sets the direction of e away from v, one of its end vertices.
- setDirectionTo(e, v)
  Sets the direction of e toward v, one of its end vertices.
- insertEdge(v, w, o)
  Insert and return an undirected edge between v and w, storing o at this position.
- insertDirectedEdge(v, w, o)
  Insert and return a directed edge between v and w, storing o at this position.
- insertVertex(o)
  Insert and return a new (isolated) vertex storing o at this position.
- removeEdge(e)
  Remove edge e.

Graphs 18

Data Structures for Graphs

- A Graph! How can we represent it?
- To start with, we store the vertices and the edges into two containers, and each edge object has references to the vertices it connects.



- Additional structures can be used to perform efficiently the methods of the Graph ADT

Data Structures for Graphs 2

Graphs – Data Structures:

- Three Main Approaches:
  - Edge List.
  - Adjacency List.
  - Adjacency Matrix.

Graphs – Edge List: (1)

- Simplest but not efficient.
- Vertex *v* storing element *o* is represented by vertex object.
- Vertex Objects:
  - Stored in container V.
  - A reference to *o*.
  - Counters for number of incident undirected edges, incoming directed edges, outgoing directed edges.

Graphs – Edge List: (2)

- Distinguishing feature is how it represents edges!.
- Edge *e* storing element *o* is represented by edge object.
  - Edge objects are also stored in a container E.
- Edge Objects:
  - Reference to o.
  - Boolean indicator tells if edge is directed.

Graphs – Edge List: (3)

- References to the vertex objects in V associated with the endpoint vertices of *e* (undirected) or references to the *origin* and *destination* (directed).

Graphs – Edge List: (4)

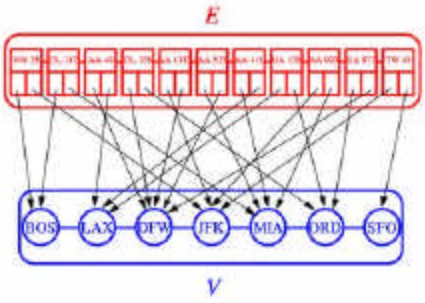
- Provides direct access to the edges and to the two vertices it is adjacent to.
- Allows for simple and efficient algorithms for edge based methods of the ADT:
  - *endVertices*, *origin* and *destination.*
- Not efficient when we want to access edges that are incident to some vertex.

Graphs – Edge List: (5)

- Need to look through all the edges!
  - incidentEdges(*v*) requires a search to look for all edges incident to *v.*
  - areAdjacent(*v*, *w*) also requires a serach!
  - removeVertex also requires a search of edges.

Edge List

- The edge list structure simply stores the vertices and the edges into unsorted sequences.
- Easy to implement.
- Finding the edges incident on a given vertex is inefficient since it requires examining the entire edge sequence.



Data Structures for Graphs 3

Performance of the Edge List Structure

Operation	Time
size, isEmpty, replaceElement, swap	O(1)
numVertices, numEdges	O(1)
vertices	O(n)
edges, directedEdges, undirectedEdges	O(m)
elements, positions	O(n+m)
endVertices, opposite, origin, destination, isDirected	O(1)
incidentEdges, inIncidentEdges, outIncidentEdges, adjacentVertices, inAdjacentVertices, outAdjacentVertices, areAdjacent, degree, inDegree, outDegree	O(m)
insertVertex, insertEdge, insertDirectedEdge, removeEdge, makeUndirected, reverseDirection, setDirectionFrom, setDirectionTo	O(1)
removeVertex	O(m)

Data Structures for Graphs 4

Graphs – Adjacency List: (1)

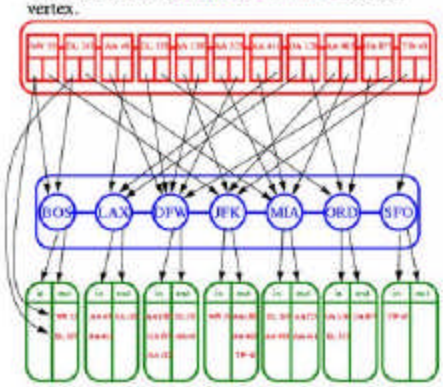
- Extends the edge list structure except it adds extra information to support direct access to incident edges of each vertex.
- Includes all structures of edge list including:
  - Each vertex object v holds reference to container $I(v)$ that stores references to the edges incident on v.

Graphs – Adjacency List: (2)

  - If directed edges, the $I(v)$ splits into $I_{in}(v)$, $I_{out}(v)$ and $I_{un}(v)$.
- Provides direct access from the edges to the vertices and from the vertices to the edges.
  - Allows for speed up for several methods.

Adjacency List (modern)

- The adjacency list structure extends the edge list structure by adding incidence containers to each vertex.



- The space requirement is O(n + m).

Data Structures for Graphs 6

Performance of the Adjacency List Structure

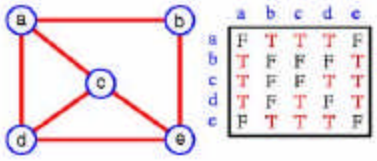
Operation	Time
size, isEmpty, replaceElement, swap	O(1)
numVertices, numEdges	O(1)
vertices	O(n)
edges, directedEdges, undirectedEdges	O(m)
elements, positions	O(n+m)
endVertices, opposite, origin, destination, isDirected, degree, inDegree, outDegree	O(1)
incidentEdges(v), inIncidentEdges(v), outIncidentEdges(v), adjacentVertices(v), inAdjacentVertices(v), outAdjacentVertices(v)	O(deg(v))
areAdjacent(u, v)	O(min(deg(u), deg(v)))
insertVertex, insertEdge, insertDirectedEdge, removeEdge, makeUndirected, reverseDirection,	O(1)
removeVertex(v)	O(deg(v))

Data Structures for Graphs 7

Graphs – Adjacency Matrix: (1)

- Extends the edge structure with additional component:
  - Matrix A allows to determine adjacencies between pairs of vertices in constant time.
  - Think of the vertices as being integers in the set {0, 1, …, n-1}.
  - Edges are pairs of these integers.

Adjacency Matrix (traditional)



- matrix M with entries for all pairs of vertices
- M[i,j] = true means that there is an edge (i,j) in the graph.
- M[i,j] = false means that there is no edge (i,j) in the graph.
- There is an entry for every possible edge, therefore: Space = $\Theta(N^2)$

Data Structures for Graphs 8

Graphs – Adjacency Matrix: (1)

- Vertex object *v* also stores a distinct integer key in the range {0, 1, …, n-1}, called the ***index*** of *v*.
- Keep a 2D n × n array A such that cell A[*i*,*j*] holds reference to edge *e* that goes from vertex *i* to vertex *j* if such an edge exists.
  - If *e* is undirected store reference to A[i,j] and A[j,i]!

Adjacency Matrix (modern)

- The adjacency matrix structures augments the edge list structure with a matrix where each row and column corresponds to a vertex.

	0	1	2	3	4	5	6
0	∅	∅	NW 35	∅	DL 247	∅	∅
1	∅	∅	∅	AA 49	∅	DL 335	∅
2	∅	AA 1387	∅	∅	AA 903	∅	TW 45
3	∅	∅	∅	∅	∅	UA 120	∅
4	∅	AA 523	∅	AA 411	∅	∅	∅
5	∅	UA 877	∅	∅	∅	∅	∅
6	∅	∅	∅	∅	∅	∅	∅

BOS 0 DFW 1 JFK 2 LAX 3 MIA 4 ORD 5 SFO 6

- The space requirement is $O(n^2 + m)$

Data Structures for Graphs 9

Performance of the Adjacency Matrix Structure

Operation	Time
size, isEmpty, replaceElement, swap	O(1)
numVertices, numEdges	O(1)
vertices	O(n)
edges, directedEdges, undirectedEdges	O(m)
elements, positions	O(n+m)
endVertices, opposite, origin, destination, isDirected, degree, inDegree, outDegree	O(1)
incidentEdges, inIncidentEdges, outIncidentEdges, adjacentVertices, inAdjacentVertices, outAdjacentVertices,	O(n)
areAdjacent	O(1)
insertEdge, insertDirectedEdge, removeEdge, makeUndirected, reverseDirection, setDirectionFrom, setDirectionTo	O(1)
insertVertex, removeVertex	$O(n^2)$

Data Structures for Graphs 10