COMP 250

Lecture 28

graphs

Nov. 13, 2017
Example
Same Example – different notation
Weighted Graph

![Diagram of a weighted graph with nodes a, b, c, d, e, f, g, and h, and weighted edges between them.](image)
A *directed graph* is a set of vertices

\[ V = \{v_i : i \in 1, ..., n \} \]

and set of ordered pairs of these vertices called *edges*.

\[ E = \{(v_i, v_j) : i, j \in 1, ..., n \} \]

In an *undirected graph*, the edges are *unordered* pairs.

\[ E = \{ \{v_i, v_j\} : i, j \in 1, ..., n \} \]
# Examples

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>airports</td>
<td></td>
</tr>
<tr>
<td>web pages</td>
<td></td>
</tr>
<tr>
<td>Java objects</td>
<td></td>
</tr>
</tbody>
</table>
Examples

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>airports</td>
<td>flights</td>
</tr>
<tr>
<td>web pages</td>
<td></td>
</tr>
<tr>
<td>Java objects</td>
<td></td>
</tr>
</tbody>
</table>
### Examples

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>airports</td>
<td>flights</td>
</tr>
<tr>
<td>web pages</td>
<td>links (URLs)</td>
</tr>
<tr>
<td>Java objects</td>
<td></td>
</tr>
</tbody>
</table>
### Examples

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>airports</td>
<td>flights</td>
</tr>
<tr>
<td>web pages</td>
<td>links (URLs)</td>
</tr>
<tr>
<td>Java objects</td>
<td>references</td>
</tr>
</tbody>
</table>
linked lists

trees

graphs
Terminology: “in degree”

In the diagram:
- **v** represents the vertices (a, b, c, d, e, f, g, h).
- The in-degree of each vertex is shown in the table:
  - a: 1
  - b: 2
  - c: 2
  - d: 0
  - e: 1
  - f: 3
  - g: 0
  - h: 1
Terminology: “out degree”

<table>
<thead>
<tr>
<th>v</th>
<th>out degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>2</td>
</tr>
<tr>
<td>e</td>
<td>2</td>
</tr>
<tr>
<td>f</td>
<td>2</td>
</tr>
<tr>
<td>g</td>
<td>1</td>
</tr>
<tr>
<td>h</td>
<td>0</td>
</tr>
</tbody>
</table>
Example: web pages

In degree: How many web pages link to some web page (e.g. f) ?

Out degree: How many web pages does some web page (e.g. f) link to ?
Terminology: path

A path is a sequence of edges such that end vertex of one edge is the start vertex of the next edge and no vertex repeated except maybe first and last.

Examples
• acfeb
• dac
• febf
• .....
Graph algorithms in COMP 251

Given a graph, what is the shortest (weighted) path between two vertices?
A cycle is a path such that the last vertex is the same as the first vertex.

Examples
- febf
- efe
- fbf
- ...

Terminology: cycle
“Travelling Salesman” COMP 360 (Hamiltonian circuit)

Find the shortest cycle that visits all vertices once.

How many potential cycles are there in a graph of n vertices?
Directed *Acyclic* Graph

There are three paths from \textit{a} to \textit{d}.

Used to capture dependencies.
202 Intro Program

206 Software Sys

250 Intro CompSci

273 Comp. Sys.

303 Software Design

302 Program Lang

251 Data Str & Alg

350 Num. Meth

240 Disc. Str. 1

223 Linear Alg.

222 Cal III

323 Prob.

310 Oper. Sys.

421 Databases

424 Artif. Intel.

360 Alg. Design

330 Theory Comp.

SYSTEMS
(compilers, networks, distributed sys, concurrency, web,..)

APPLICATIONS
(graphics, vision, bioinf, games, machine learning,..)

THEORY
(crypto, optimization, game theory, logic, correctness, computability,..)
Graph ADT

• addVertex( ... ), addEdge( ... )
• containsVertex( ... ), containsEdge( ... )
• getVertex( ... ), getEdge( ... )
• removeVertex( ... ), removeEdge( ... )
• numVertices( ), numEdges( )
• ...

How to implement a Graph class? A graph is a generalization of a tree, so …
Recall: How to implement a rooted tree in Java?

class Tree<T>{
    TreeNode<T> root;
}

// inner class

class TreeNode<T>{
    T element;
    ArrayList<TreeNode<T>> children;
    TreeNode<T> parent;
}

// alternatively...

class TreeNode<T>{
    T element;
    TreeNode<T> firstChild;
    TreeNode<T> nextSibling;
}
Adjacency List
(generalization of children for graphs)

Here each adjacency list is sorted, but that is not always possible (or necessary).
How to implement a Graph class in Java?

class Graph<T> {
    class Vertex<T> {
        ArrayList<Vertex> adjList;
        T element;
    }
}

This is a very basic Graph class.
How to implement a Graph class in Java?

```java
class Graph<T>  {

    class Vertex<T>   {
        ArrayList<Edge>     adjList;
        T                     element;
        boolean             visited;
    }

    class Edge {
        Vertex         endVertex;
        double         weight;
    }

    // Implementation details...
}

Unlike a rooted tree, there is no notion of a root vertex in a graph.
```
How to reference vertices?

Suppose we have a string name (key) for each vertex.

[Code]
```java
class Graph<T> {
    HashMap<String, Vertex<T>> vertexMap;
    class Vertex<T> { ... }
    class Edge<T> { ... }
}
```

We could also just enumerate vertices.
How many objects?

![Diagram showing objects and connections]
Assume we have a mapping from vertex names to 0, 1, ..., n-1.

boolean adjMatrix[6][6]

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>f</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Adjacency Matrix

```
boolean adjMatrix[6][6]
```

```
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>f</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```
Suppose a graph has \( n \) vertices.

The graph is *dense* if number of edges is close to \( n^2 \).

The graph is *sparse* if number of edges is close to \( n \).

(These are not formal definitions.)
Exercise

Would you use an *adjacency list* or *adjacency matrix* for each of the following?

- The graph is sparse e.g. 10,000 vertices and 20,000 edges and we want to use as little space as possible.
- You need to answer the query `areAdjacent()` as quickly as possible, no matter how much space you use.
- You need to perform operation `insertVertex`.
- You need to perform operation `removeVertex`.
Exercise

Would you use an *adjacency list* or *adjacency matrix* for each of the following?

• The graph is sparse e.g. 10,000 vertices and 20,000 edges and we want to use as little space as possible.

• The graph is dense e.g. 10,000 vertices and 20,000,000 edges, and we want to use as little space as possible.
Exercise

Would you use an *adjacency list* or *adjacency matrix* for each of the following?

- The graph is sparse e.g. 10,000 vertices and 20,000 edges and we want to use as little space as possible.
- The graph is dense e.g. 10,000 vertices and 20,000,000 edges, and we want to use as little space as possible.
- Answer the query `areAdjacent()` as quickly as possible, no matter how much space you use.
Exercise

Would you use an *adjacency list* or *adjacency matrix* for each of the following?

- The graph is sparse e.g. 10,000 vertices and 20,000 edges and we want to use as little space as possible.
- The graph is dense e.g. 10,000 vertices and 20,000,000 edges, and we want to use as little space as possible.
- Answer the query `areAdjacent()` as quickly as possible, no matter how much space you use.
- Perform operation `insertVertex(v)`.
Exercise

Would you use an *adjacency list* or *adjacency matrix* for each of the following?

- The graph is sparse e.g. 10,000 vertices and 20,000 edges and we want to use as little space as possible.
- The graph is dense e.g. 10,000 vertices and 20,000,000 edges, and we want to use as little space as possible.
- Answer the query areAdjacent() as quickly as possible, no matter how much space you use.
- Perform operation insertVertex( v ).
- Perform operation removeVertex( v ).
Next lecture

• Recursive graph traversal
  • depth first

• Non-recursive graph traversal
  • depth first
  • breadth first