Graphs

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

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
**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**
  - all the edges are directed
  - e.g., route network
- **Undirected graph**
  - all the edges are undirected
  - e.g., flight network

**Applications**

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

- **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

**Terminology (cont.)**

- **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
**Terminology (cont.)**

- **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**
  - $C_1 = (V, b, X, g, Y, f, W, c, U, a, \ldots)$ is a simple cycle
  - $C_2 = (U, c, W, e, X, g, Y, f, W, d, V, a, \ldots)$ is a cycle that is not simple

**Properties**

**Property 1**

\[ \sum_v \text{deg}(v) = 2m \]

**Notation**

- $n$ number of vertices
- $m$ number of edges
- $\text{deg}(v)$ degree of vertex $v$

**Proof:** each edge is counted twice

**Property 2**

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

\[ m \leq n(n - 1)/2 \]

**Proof:** each vertex has degree at most $(n - 1)$

**Example**

- $n = 4$
- $m = 6$
- $\text{deg}(v) = 3$

What is the bound for a directed graph?
Vertices and Edges

- A **graph** is a collection of **vertices** and **edges**.
- We model the abstraction as a combination of three data types: Vertex, Edge, and Graph.
- A **Vertex** is a lightweight object that stores an arbitrary element provided by the user (e.g., an airport code)
  - We assume it supports a method, `element()`, to retrieve the stored element.
- An **Edge** stores an associated object (e.g., a flight number, travel distance, cost), retrieved with the `element()` method.

Graph ADT

- `numVertices()`: Returns the number of vertices of the graph.
- `vertices()`: Returns an iteration of all the vertices of the graph.
- `numEdges()`: Returns the number of edges of the graph.
- `edge()`: Returns an iteration of all the edges of the graph.
- `getEdge(u, v)`: Returns the edge from vertex u to vertex v, if one exists; otherwise return null. For an undirected graph, there is no difference between `getEdge(u, v)` and `getEdge(v, u)`.
- `endVertices(e)`: Returns an array containing the two endpoint vertices of edge e. If the graph is directed, the first vertex is the origin and the second is the destination.
- `opposite(v, e)`: For edge e incident to vertex v, returns the other vertex of the edge; an error occurs if e is not incident to v.
- `outDegree(v)`: Returns the number of outgoing edges from vertex v.
- `inDegree(v)`: Returns the number of incoming edges to vertex v. For an undirected graph, this returns the same value as `outDegree(v)`.
- `outgoingEdges(v)`: Returns an iteration of all outgoing edges from vertex v.
- `incomingEdges(v)`: Returns an iteration of all incoming edges to vertex v. For an undirected graph, this returns the same collection as `outgoingEdges(v)`.
- `insertVertex(x)`: Creates and returns a new Vertex storing element x.
- `insertEdge(u, v, x)`: Creates and returns a new Edge from vertex u to vertex v, storing element x; an error occurs if there already exists an edge from u to v.
- `removeVertex(v)`: Removes vertex v and all its incident edges from the graph.
- `removeEdge(e)`: Removes edge e from the graph.
**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**

- 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
- The "old fashioned" version just has 0 for no edge and 1 for edge

![Adjacency Matrix](image)

```
    0 1 2 3
  u --------------------------------------
  v       e   g
  w       e   f   g   h
  z       h
```

Performance

- $n$ vertices, $m$ edges
- no parallel edges
- no self-loops

<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>$n + m$</td>
<td>$n + m$</td>
<td>$n^2$</td>
</tr>
<tr>
<td>incidentEdges($v$)</td>
<td>$m$</td>
<td>$\deg(v)$</td>
<td>$n$</td>
</tr>
<tr>
<td>areAdjacent ($v$, $w$)</td>
<td>$m$</td>
<td>$\min(\deg(v), \deg(w))$</td>
<td>1</td>
</tr>
<tr>
<td>insertVertex($o$)</td>
<td>1</td>
<td>1</td>
<td>$n^2$</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>$m$</td>
<td>$\deg(v)$</td>
<td>$n^2$</td>
</tr>
<tr>
<td>removeEdge($e$)</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>