GRAPHS

- Definitions
- Examples
- The Graph ADT
What is a Graph?

- A graph $G = (V,E)$ is composed of:
  - $V$: set of vertices
  - $E$: set of edges connecting the vertices in $V$

- An edge $e = (u,v)$ is a pair of vertices

- Example:

  $V = \{a,b,c,d,e\}$

  $E = \{(a,b),(a,c),(a,d), (b,e),(c,d),(c,e), (d,e)\}$
Applications

- electronic circuits

*find the path of least resistance to CS16*

- networks (roads, flights, communications)
mo’ better examples
A Spike Lee Joint Production

- scheduling (project planning)

A typical student day

- wake up
- eat
- work
- more cs16
- play
- cs16 meditation
- cs16 program
- battletris
- make cookies for cs16 HTA
- sleep
- dream of cs16
Graph Terminology

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

\[ \sum \text{deg}(v) = 2(\# \text{ edges}) \quad v \in V \]

- 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.
More Graph Terminology

- **simple path**: no repeated vertices

- **cycle**: simple path, except that the last vertex is the same as the first vertex
Even More Terminology

- **connected graph**: any two vertices are connected by some path

  ![Connected Graph](image)

  ![Not Connected Graph](image)

- **subgraph**: subset of vertices and edges forming a graph

- **connected component**: maximal connected subgraph. E.g., the graph below has 3 connected components.
¡Caramba! Another Terminology Slide!

- **(free) tree** - connected graph without cycles
- **forest** - collection of trees
Connectivity

Let $n = \#\text{vertices}$
\[ m = \#\text{edges} \]

- complete graph - all pairs of vertices are adjacent

\[ m = \frac{1}{2} \sum_{v \in V} \deg(v) = \frac{1}{2} \sum_{v \in V} (n - 1) = \frac{n(n-1)}{2} \]

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

\[ n = 5 \]
\[ m = \frac{5 \times 4}{2} = 10 \]

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

\[ n = \text{#vertices} \]
\[ m = \text{#edges} \]

- For a tree \( m = n - 1 \)

\[
\begin{array}{c}
  \text{\( n = 5 \)} \\
  \text{\( m = 4 \)} \\
\end{array}
\]

- If \( m < n - 1 \), \( G \) is not connected

\[
\begin{array}{c}
  \text{\( n = 5 \)} \\
  \text{\( m = 3 \)} \\
\end{array}
\]
Spanning Tree

• A **spanning tree** of $G$ is a subgraph which
  - is a tree
  - contains all vertices of $G$

• Failure on any edge disconnects system (least fault tolerant)
AT&T vs. RT&T

(Roberto Tamassia & Telephone)

- Roberto wants to call the TA’s to suggest an extension for the next program...

- One fault will disconnect part of graph!!

- A cycle would be more fault tolerant and only requires $n$ edges

But Plant-Ops ‘accidentally’ cuts a phone cable!!!
Euler and the Bridges of Koenigsberg

Can one walk across each bridge exactly once and return at the starting point?

- Consider if you were a UPS driver, and you didn’t want to retrace your steps.
- In 1736, Euler proved that this is not possible
Graph Model (with parallel edges)

- **Eulerian Tour**: path that traverses every edge exactly once and returns to the first vertex

- **Euler’s Theorem**: A graph has a Eulerian Tour if and only if all vertices have even degree
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 \( G \).
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 $v$. 
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.
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 $e$. 