Lecture 13: Graphs
Outline

- New data structure: Graphs
  - used to model a variety of problems
  - different graph than “plotting a function graph”
- Definitions
  - lots of terminology with more to come in later lectures
  - even more terminology exists, but not covered in this course
- Basic Properties
- Graph ADT
Graphs: Motivation

- Graph is a natural representation to visualize flight roots.
  - Each city is represented with a node.
    - Can label each node with a three letter airport code.
  - Two cities with a direct flight between them are connected with an edge.
    - Can label edge with the mileage of the route, time to fly, etc.
Graphs: Motivation

- A way to think about graphs:
  - Nodes specify some entities we are interested in
  - Edges specify the relationships between these entities
Graphs: Motivation

- Can answer many interesting questions using graphs
  - Can we reach one city from another city?
  - What is the route with minimum number of connections between 2 cities?
  - What is the minimum mileage route between 2 cities?
- Many interesting questions can be answered efficiently, but for some there is no hope of efficient answer
A graph is a pair \((V, E)\), where
- \(V\) is a collection of nodes, or vertices
- \(E\) is a collection of pairs of vertices, called edges

In this example
- \(V = \{a, b, c, d, f\}\)
- \(E = \{(a, c), (b, c), (c, f), (b, d), (d, f), (c, d)\}\)
Two 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
  - \((v, u)\) and \((u, v)\) are two different edges

- Undirected edge
  - unordered pair of vertices \((u, v)\)
  - e.g., a network of friends
  - If \textit{Sam} is a friend of \textit{Bob}, then \textit{Bob} is also a friend of \textit{Sam}
  - \((u, v)\) and is \((v, u)\) the same edge
Graph Types

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

- Undirected graph
  - all edges are undirected
  - “friends” network
Applications

- Transportation networks
  - City map
  - Highway network
  - Flight network

- Computer networks
  - Local area network
  - Internet
  - Web

- Computer Vision
  - Image pixels are graph nodes, neighboring pixels connected by edges
Graph Terminology

- **End vertices** (or **endpoints**) of an edge are 2 vertices that joined by an edge
  - $u$ and $v$ are the endpoints of $(u,v)$
- Edge is **incident** on a vertex if the vertex is one of this edge’s endpoints
  - $(v,w)$ is incident on $w$
- 2 vertices are **adjacent** if there is an edge between them
  - $v$ and $x$ are adjacent
- **Degree** of a vertex is the number of adjacent vertices
  - $x$ has degree 4
More Graph Terminology

- **Parallel** edges are edges that
  - undirected graph: have the same endpoints
  - directed graph: same origin and destination
  - There are 2 parallel edges between $x$ and $z$

- **Self-loop** is an edge whose endpoints coincide
  - $(z,z)$ is a self-loop

- **Simple** graph: no parallel edges and no self-loops

- We will deal almost exclusively with simple graphs
Even More Terminology

- **Path**
  - sequence of vertices such that consecutive vertices are adjacent
  - Example path: $U, W, X, Y, W, V$

- **Simple path**
  - path such that all its vertices are distinct
  - Example: $V, X, Z$
Yet Even More Terminology

- **Cycle**
  - Path on which the first vertex is equal to the last vertex and

- **Simple cycle**
  - cycle such that all its vertices, except the first and the last one, are distinct
  - No repeating edges on the path
  - Example: $V, X, Y, W, U, V$
Yet Even More Terminology Still

- **Connected** graph
  - Graph where any two vertices are connected by some path

- **Subgraph** of a graph \((V,E)\)
  - \((V',E')\) s.t. \(V'\) is a subset of \(V\), \(E'\) is a subset of \(E\), and both endpoints of edges in \(E'\) are in \(V'\)
  - A **spanning** subgraph of \(G\) is a subgraph that contains all vertices of \(G\)
A connected component $G'$ of a graph $G$ is a maximal connected subgraph of $G$

- **Connected**: There is a path between any 2 vertices in the connected component $G'$
- **Maximal**: no way to add into $G'$ any vertices and/or edges of $G$ which are not currently in $G'$ in such a way that the resulting subgraph is connected
Connected Component

Graph G:

**not** a connected component in thick red lines

added a new vertex $c$ and edge $(b, c)$, and still connected
Connected Component

Graph G

not a connected component in thick red lines

added a new edge (a, c), and still connected
Connected Components
Almost the Last Terminology Slide

- **Tree** graph is any *connected* graph without cycles
  - this is different from the “rooted” trees studied previously
  - to make a distinction from “rooted” trees, sometimes say *free tree*

- **Forest** graph is any graph without cycles
  - connected components of a forest are trees
The Last Terminology Slide

- 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.
  - In not connected graph we have spanning forest.
  - In connected graph we can have a spanning tree.
Properties

Let

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

Property 1: \( \sum_v \text{deg}(v) = 2m \)

Proof:

- Every edge \((w,u)\) has 2 end points: \( w \) and \( u \)
- Endpoint \( w \) contributes exactly 1 to \( \text{deg}(w) \)
- Endpoint \( u \) contributes exactly 1 to \( \text{deg}(u) \)
- Thus each edge contributes exactly 2 to the sum on the left
Properties

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

- Property 2: In an undirected graph with no self-loops and no parallel edges \( m \leq n (n - 1)/2 \)

  **Proof:** Property 1 says: \( 2m = \sum_v \text{deg}(v) \)

  each vertex has degree at most \( (n - 1) \)

  \[ m = \frac{1}{2} \sum_v \text{deg}(v) \leq \frac{1}{2} \sum_v (n - 1) = \frac{1}{2} n (n - 1) \]

  Property 2 says that \( m \) is \( O(n^2) \)
Euler Cycle and the 7 Bridges of Koenigsberg

- The year is 1735. City of Koenigsberg has a funny layout of 7 bridges across the river

- Is it possible to walk across each bridge exactly once and return to the same starting point?
  - thought impossible, but no one can prove it

- Eulerian Cycle
  - path that traverses every edge exactly once and returns to the first vertex

- Euler proves a theorem:
  
  *A graph has a Eulerian Cycle if and only if all vertices have even degree*
Main Methods of the Graph ADT

- Vertices and edges
  - are objects
  - can store elements

- Accessor methods
  - `endVertices(e)`: returns array of the two endvertices of `e`
  - `opposite(v, e)`: return the vertex opposite of `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 and return a vertex storing element `o`
  - `insertEdge(v, w, o)`: insert and return an edge `(v,w)` storing `o`
  - `removeVertex(v)`: remove vertex `v` (and its incident edges) and return element stored at `v`
  - `removeEdge(e)`: remove edge `e` and return element stored at `e`

- Iterator methods
  - `incidentEdges(v)`: return iterator over edges incident on `v`
  - `vertices()`: return iterator over vertices
  - `edges()`: return iterator over edges