Test #2 is Wednesday
• Topics: BSTs (incl. balanced BSTs), PQs, heaps, struct. ind., hash tables.
• For each data structure, you should know:
  • the supported operations
  • any assumptions about the data (e.g. comparable?)
  • RT of various implementations (our “ideas” from slides)
  • advantages and disadvantages of various implementations
• From the labs, you should know
  • AVL rotations
  • heaps percolateUp, percolateDown, heapify
  • Comparators, Iterators
  • Markov model (applied to text generation)

Prelab 9 is due Monday after break (you can do it now)
A graph is a way of specifying relationships among a collection of items. It consists of a set of objects, called nodes, with certain pairs of these objects connected by links, called edges.

Two nodes are neighbours or adjacent if they are connected by an edge.
We define a **directed graph** to consist of a set of nodes, as before, together with a set of **directed edges** (where the direction on the edge matters).

If a graph is not directed, we call it an **undirected graph**.

(a) A graph on 4 nodes.  
(b) A directed graph on 4 nodes.
Examples of Graphs

Example 1: network structure of the Internet — then called the Arpanet — in December 1970

Image from F. Heart, A. McKenzie, J. McQuillian, and D. Walden
Examples of Graphs

Example 1: network structure of the Internet — then called the Arpanet — in December 1970

Image from J. Kleinberg and D. Easley
Example 2: the social network of friendships within a 34-person karate club

Image from D. Easley and J. Kleinberg
Example 3: links between political blogs prior to the 2004 U.S. Presidential election

Image from http://www-personal.umich.edu/ladamic/img/politicalblogs.jpg
Examples of Graphs

Example 4: links between tic-tac-toe game states

Image from wikipedia
Graphs appear in many domains, whenever it is useful to represent how things are either physically or logically linked to one another in a network structure.

communication networks
- nodes are computers or other devices that can relay messages
- edges represent direct links along which messages can be transmitted

social networks
- nodes are people or groups of people
- edges represent some kind of social interaction

information networks
- nodes are information resources such as web pages or documents
- edges represent logical connections such as hyperlinks, citations, refs

game networks
- nodes are states in a game, such as placements of x/o’s in tic-tac-toe
- edges represent a move that transitions from one state to another

plenty others!
Examples of Graphs

Example 5: the spread of an epidemic disease (such as the tuberculosis outbreak shown here)

Image from Andre et al.
Example 6: airline routes

(a) Airline routes

Image from Northwest Airline
Examples of Graphs

Example 7: metro map

Image from www.wmata.com/metrorail/systemmap.cfm
Example 8: prerequisite chart among CS courses at Oberlin College

Non-Major Courses
- 100
- 140
- FYSP 155
- 299

Modeling courses
- 190
- 290
- 390

Math Electives (sub 1 for CSCI 3xx)
- 331
- 345
- 348

Math 132 or higher

(c) Flowchart of college courses
Examples of Graphs

Example 9: the structure of a bridge

(d) *Tank Street Bridge in Brisbane*

Image from D. Easley and J. Kleinberg
A path is a sequence of nodes with the property that each consecutive pair in the sequence is connected by an edge. Can think of path as either sequence of nodes, or sequence of edges, or both.

For ex: MIT, BBN, RAND, UCLA is a path in the Internet graph.

According to this definition, a path can repeat nodes: for example, SRI, STAN, UCLA, SRI, UTAH, MIT is a path.

A simple path is a path that does not repeat nodes.

A length of a path is the number of edges in that path.
A cycle is a path with at least three edges, in which the first and last nodes are the same, but otherwise all nodes are distinct.

For ex: SRI,STAN,UCLA,SRI is a cycle in the Internet graph

In fact, every edge in the 1970 Arpanet belongs to a cycle, and this was by design: if any edge were to fail, there would still be a way to get from any node to any other node.

Cycles in communication and transportation networks are often present to allow for redundancy, and provide alternate routings that go the other way around the cycle. Also have cycles in social networks...
Given a graph, it is natural to ask whether every node can reach every other node by a path.

A graph is **connected** if there is a path between every pair of nodes.

A directed graph is **strongly connected** if there is a directed path between every pair of nodes. It is **weakly connected** if the underlying undirected graph is connected.
Components

If a graph is not connected, then it breaks apart naturally into a set of connected “pieces” that are as big as possible and don’t overlap.

A connected component of a graph is a subset of the nodes such that:
- every node in the subset has a path to every other, and
- the subset is not part of some larger set with the connected property.
Dividing a graph into its components is just one, global way of describing its structure. Within a component, there may be richer internal structure.

For example, look at largest example in collaboration network
  • if remove prominent node at centre, disconnects into 3 separate comp’s
Giant Components

Consider social network of the entire world.

1. Is this global friendship network connected?
2. How big is the component that you are in?

The largest component is likely to include a significant fraction of the world’s population.

This is true in many different types of network datasets — large, complex networks often have a giant component, i.e. a connected component that contains a significant fraction of all the nodes.

Moreover, when a network contains a giant component, it almost always contains only one:

• if there were 2 giant components, then just a single edge between any two nodes in separate components connects them, and this is likely to occur

In rare cases when two large components exist, their merging has been sudden, dramatic, and ultimately catastrophic. (see Guns, Germs, and Steel)
A **degree** of a vertex is the number of edges incident to that vertex.

- If the graph is directed, we can further talk about a vertex’s indegree and outdegree, which represent the number of in-edges and out-edges of a vertex, respectively.

For example, the prominent node has degree 22. All nodes in cycles have degree 2. All internal nodes in a path have degree 2, and the two endpoints degree 1.
A graph $G=(V,E)$ is a set of vertices $V$ and set of edges $E$.

- directed graph has directed edges $e=(u,v)$ that are ordered pairs of $V$
- undirected graphs have undirected edges $e\{u,v\}$ of unordered pairs of $V$

Two nodes are neighbours or adjacent if they are connected by an edge.

A path is a sequence of nodes with the property that each consecutive pair in the sequence is connected by an edge. A simple path is a path that does not repeat nodes. A length of a path is the number of edges in that path.

A graph is connected if there is a path between every pair of nodes. A directed graph is strongly connected if there is a directed path between every pair of nodes, and weakly connected if the underlying undirected graph is connected.

A connected component of a graph is a subset of the nodes such that:

- every node in the subset has a path to every other, and
- the subset is not part of some larger set with the connected property.

The degree of a node is the number of edges incident to the node.
Graph Operations

- size() - return the number of nodes in the graph
- isEmpty() - return whether there are 0 nodes in the graph
- isEdge(Vertex v, Vertex w) - return true if (v, w) is an edge
- areConnected(Vertex v, Vertex w) - return whether v, w are connected
- shortestPath(Vertex v, Vertex w) - return len of the shortest v-w path
- isConnected() - return whether graph is one connected component
- numComponents() - return the number of connected components
- findConnComponent(Vertex v) - return vertices in v’s conn’d component
- getAvgDegree() - determine the average degree of all vertices
- findSpanningTree() - find the fewest number of edges of G to keep so that graph is still connected
- isAcyclic() - return whether the graph contains no cycles
- traverse() - iterate through nodes in an ordered fashion