

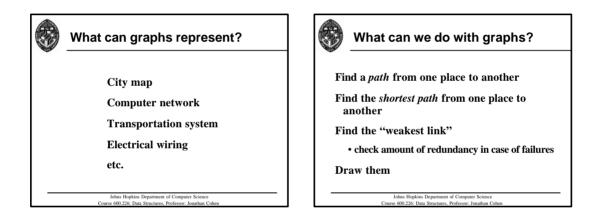


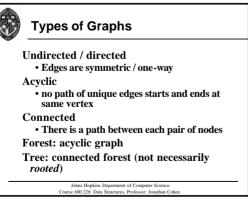
### What is a Graph?

(in computer science, it's not a data plot) General structure for representing positions

- with an arbitrary connectivity structure
- Collection of *vertices* (nodes) and *edges* (arcs)
  - ---Edge is a pair of vertices it connects the two vertices, making them *adjacent*
- A tree is a special type of graph!

Johns Hopkins Department of Computer Science Course 600.226: Data Structures, Professor: Jonathan 0

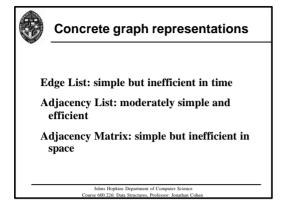






### (undirected) Graph ADT

numVertices(), numEdges(): return # of vertices or edges vertices(), edges(): return iterator of vertices or edges degree(v): return # of incident edges on a vertex incidentEdges(v): return iterator of incident edges on vertex endVertices(e): return two vertices of edge e opposite(v, e): return endpoint of e that is not v areAdjacent(v, w): return whether an edge connects v to w insertEdge(v, w, o): create and return an edge between v and w storing object o insertVertex(o): insert and return new vertex storing o removeVertex(v): remove vertex v and its adjacent edges removeVertex(v): remove edge e





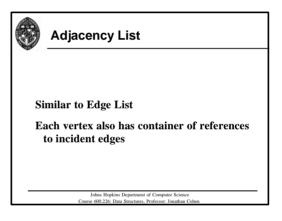
# Edge List

- Container (list/vector/dictionary) of vertices

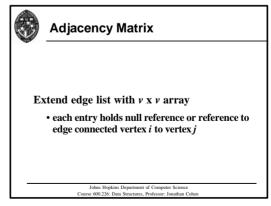
  Each vertex just has its object
- Container (list/vector/dictionary) of edges
  - Each edge has its object
  - Edge also has references to its two endpoint vertices

Johns Hopkins Department of Computer Science Course 600.226: Data Structures, Professor: Jonathan Col

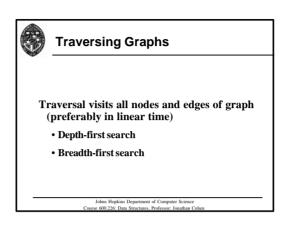
Edge list (linked list) efficiency		
vertices():	O(n)	
edges():	O(m)	
endVertices(e):	<i>O</i> (1)	
incidentEdges(v):	O(m)	
areAdjacent(v, w):	O(m)	
removeEdge(e):	<b>O</b> (1)	
removeVertex(v):	O(m)	
 Johns Hopkins Department of Computer Course 600.226: Data Structures, Professor: Joa		

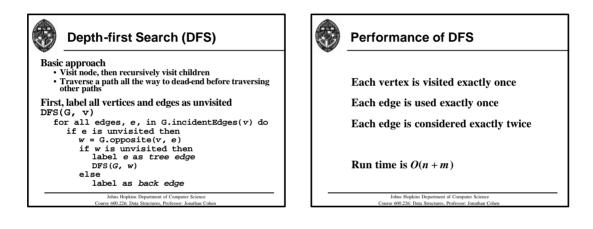


Adjacency lis efficiency	t (linked list)	
vertices():	O(n)	
edges():	O(m)	
endVertices(e):	<i>O</i> (1)	
incidentEdges(v):	$O(\deg(v))$	
areAdjacent(v, w):	$O(\min(\deg(v), \deg(w)))$	
removeEdge(e):	$O(\deg(u) + \deg(v))$ e = (u,v)	
removeVertex(v):	$O(\deg(v) + \sum_{u \in \operatorname{adj}(v)} \operatorname{deg}(u))$	
(note: the last two a	re incorrect in the textbook)	
	rtment of Computer Science ctures, Professor: Jonathan Cohen	



	Adjacency Matrix efficiency		
	vertices():	O(n)	
	edges():	O(m)	
	endVertices(e):	<i>O</i> (1)	
	incidentEdges(v):	O(n)	
	areAdjacent(v, w):	<i>O</i> (1)	
	removeEdge(e):	<i>O</i> (1)	
	removeVertex(v):	$O(n^2)$	
• perhaps $O(n)$ with amortization			
Johns Hopkins Department of Computer Science Course 600.226: Data Structures, Professor, Jonathan Cohen			







#### **Uses for DFS**

All n nodes and m edges are visited

• if graph is not connected, all nodes and edges in connected component are visited

#### Useful for:

- Find a spanning tree of a graph
- · Find a path between two vertices
- · Find all connected components of a graph
- · Finding a cycle (if any) in a graph

Basic approach BFS(G, s)

# **Breadth-first search** • Visit a node, then put all its children on a queue to be visited · Visit nodes in order of queue

-visits "close" nodes first, then "farther" nodes mark all vertices and edges unvisited Initialize queue, Q to contain vertex, s while not Q.isEmpty() do v = Q.dequeue(), mark v visited
for each edge, e of v do if e is unvisited then
w = G.other(v, e) if w is unvisited then label e as tree edge, Q.enqueue(w) else label e as cross edge 00.226: Data S