

Graph ADT Implementation

- code organization

- AbstractGraph, AbstractVertex, and AbstractEdge should contain instance variables common to both adjacency list and adjacency matrix implementations

Adjacency Matrix Implementation	Adjacency List Implementation
graph stores <ul style="list-style-type: none"> a list of vertices a list of edges doubly-linked list allows for O(1) removal given reference to list node 2D array, indexed by vertex key	graph stores <ul style="list-style-type: none"> a list of vertices a list of edges doubly-linked list allows for O(1) removal given reference to list node
vertex stores <ul style="list-style-type: none"> the associated object degree of the vertex reference to the vertex's location in the list of vertices distinct integer key in the range 0..n-1	vertex stores <ul style="list-style-type: none"> the associated object degree of the vertex reference to the vertex's location in the list of vertices list of incident edges doubly-linked list allows for O(1) removal given reference to list node
edge stores <ul style="list-style-type: none"> the associated object endpoint vertices reference to the edge's location in the list of edges 	edge stores <ul style="list-style-type: none"> the associated object endpoint vertices reference to the edge's location in the list of edges references to the edge's location in the incidence lists for its endpoint vertices
array stores <ul style="list-style-type: none"> Adj[i] holds the edge from vertex with index i to vertex with index j (null if no edge) 	

- AbstractGraph should contain bodies for those methods that can be implemented there
- helper classes such as ListNode should be inner classes

3

Graph ADT Implementation

- running times

- be sure to achieve the running times discussed in class
 - avoid unnecessary loops, searching

	adjacency list	adjacency matrix
numVertices(), numEdges()	O(1)	O(1)
vertices(), edges()	O(1) per element	O(1) per element
aVertex()	O(1)	O(1)
degree(v)	O(1)	O(1)
adjacentVertices(v)	O(1) per element	O(n) – to scan row/column of array
incidentEdges(v)	O(1) per element	O(n) – to scan row/column of array
endVertices(e)	O(1)	O(1)
opposite(v,e)	O(1)	O(1)
areAdjacent(v,w)	O(min(deg(v,w))) – search list for vertex with smaller degree	O(1)
insertEdge(v,w,o)	O(1)	O(1)
insertVertex(o)	O(1)	O(n) – to initialize row/col of array O(n ²) – if array needs to grow
removeVertex(v)	O(deg(v)) – to remove each incident edge	O(1) – with clever bookkeeping (and wasted space) O(n ²) – shifting in array
removeEdge(e)	O(1)	O(1)

- to achieve O(1) removal from a list (and for full credit), use your own doubly-linked list implementation for lists and store the reference to the list node

- can't just add prev, next pointers to AbstractVertex/AbstractEdge because in the adjacency list implementation, an edge will appear in three lists – the list of edges in the graph and the list of incident edges for each of the edge's vertices
- no need to search for the list node if there's a reference stored!

4

Graph ADT Implementation

- Java things

- .equals() vs ==
 - a == b is for checking if a and b are the same object (same location in memory)
 - e.g. list nodes
 - use == when *object identity* matters – different objects are different things even if they have the same state
 - a.equals(b) is for checking if a and b are equivalent objects (different locations in memory but viewed as interchangeable)
 - e.g. String
- only cast if you need methods of the more specific type