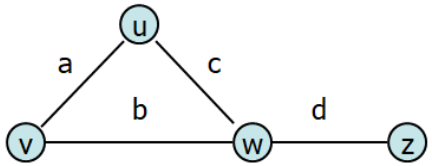
CS 225

#36: Graph Implementation

Graph Implementation #1: Edge List

Vert.	Edges
u v w z	a b c d



Data Structures:

Vertex Collection:

Edge Collection:

Operations on an Edge List implementation:

insertVertex(K key):
- What needs to be done?

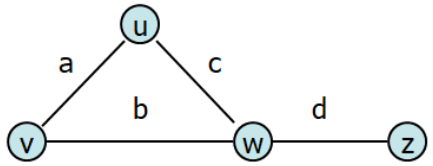
removeVertex(Vertex v):
- What needs to be done?

incidentEdges(Vertex v):
- What needs to be done?

areAdjacent(Vertex v1, Vertex v2):
- Can this be faster than `G.incidentEdges(v1).contains(v2)`?

insertEdge(Vertex v1, Vertex v2, K key):
- What needs to be done?

Graph Implementation #2: Adjacency Matrix



Vert.	Edges	Adj. Matrix
u v w z	a b c d	u v w z u v w z

Data Structures:

Operations on an Adjacency Matrix implementation:

insertVertex(K key):
- What needs to be done?

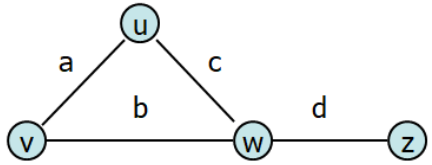
removeVertex(Vertex v):
- What needs to be done?

incidentEdges(Vertex v):
- What needs to be done?

areAdjacent(Vertex v1, Vertex v2):
- Can this be faster than `G.incidentEdges(v1).contains(v2)`?

insertEdge(Vertex v1, Vertex v2, K key):
- What needs to be done?

Graph Implementation #3: Adjacency List



Vertex List	Edges
u	a
v	b
w	c
z	d

Operations on an Adjacency Matrix implementation:

insertVertex(K key):

removeVertex(Vertex v):

incidentEdges(Vertex v):

areAdjacent(Vertex v1, Vertex v2):

insertEdge(Vertex v1, Vertex v2, K key):

Running Times of Classical Graph Implementations

	Edge List	Adj. Matrix	Adj. List
Space	**n+m**	**n^2**	**n+m**
insertVertex	**1**	**n**	**1**
removeVertex	**n+m**	**n**	**deg(v)**
insertEdge	**1**	**1**	**1**
removeEdge	**m**	**1**	**min(deg(v), deg(w))**
incidentEdges	**m**	**n**	**deg(v)**
areAdjacent	**m**	**1**	**min(deg(v), deg(w))**

Q: If we consider implementations of simple, connected graphs, what relationship between n and m?

- On connected graphs, is there one algorithm that underperforms the other two implementations?

Q: Is there clearly a single best implementation?

- Optimized for fast construction:

- Optimized for areAdjacent operations: