

Data Structures

Graph Implementations

CS 225

October 23, 2024

Brad Solomon



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

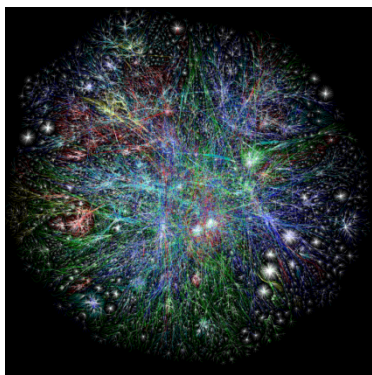
Department of Computer Science



Learning Objectives

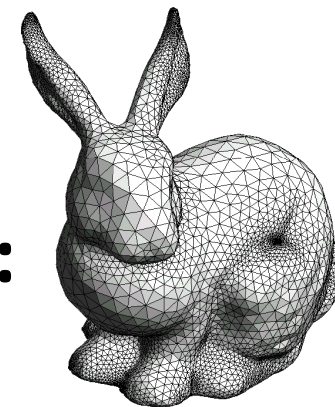
Discuss graph implementation and storage strategies

Graphs



To study all of these structures:

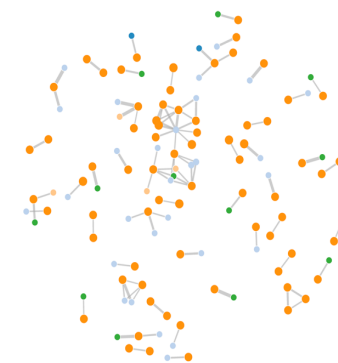
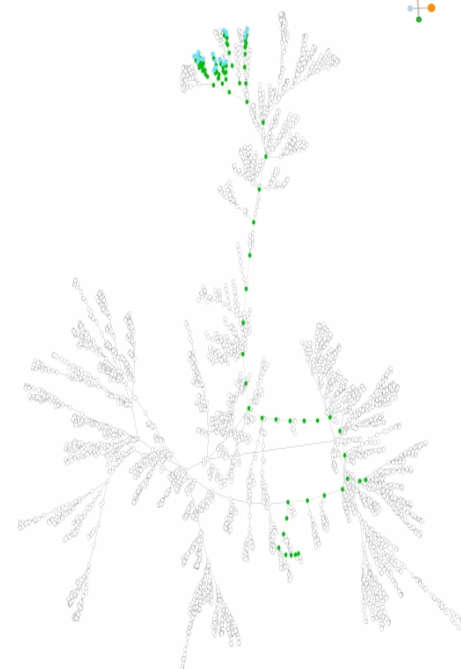
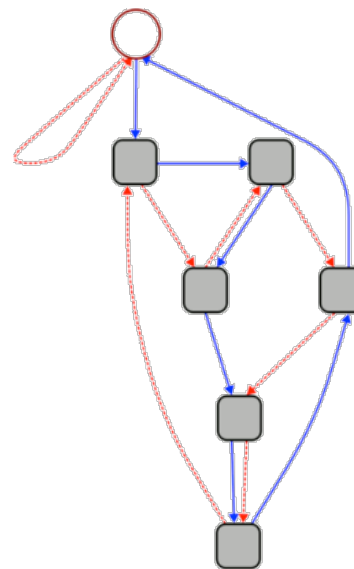
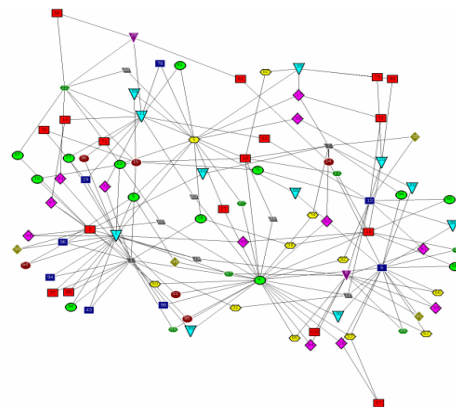
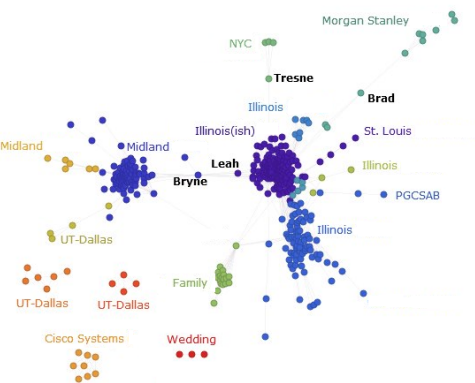
1. A common vocabulary
2. Graph implementations
3. Graph traversals
4. Graph algorithms



HAMLET



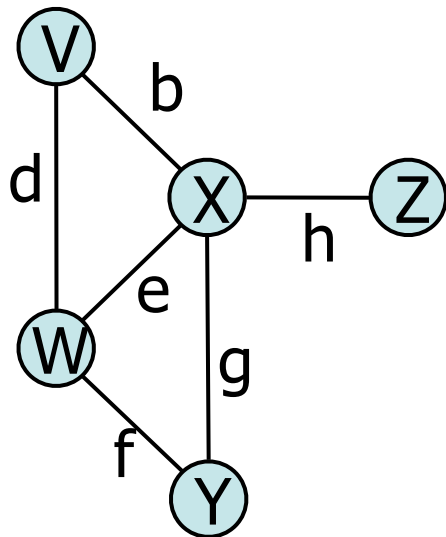
TROILUS AND CRESSIDA



Graph ADT

Data:

- Vertices
- Edges
- Some data structure maintaining the structure between vertices and edges.

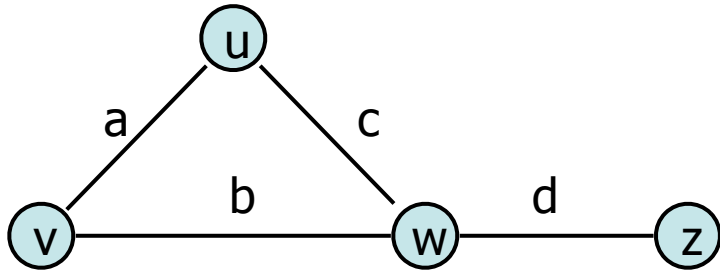


Functions:

- insertVertex(K key);
- insertEdge(Vertex v1, Vertex v2, K key);
- removeVertex(Vertex v);
- removeEdge(Vertex v1, Vertex v2);
- incidentEdges(Vertex v);
- areAdjacent(Vertex v1, Vertex v2);
- origin(Edge e);
- destination(Edge e);

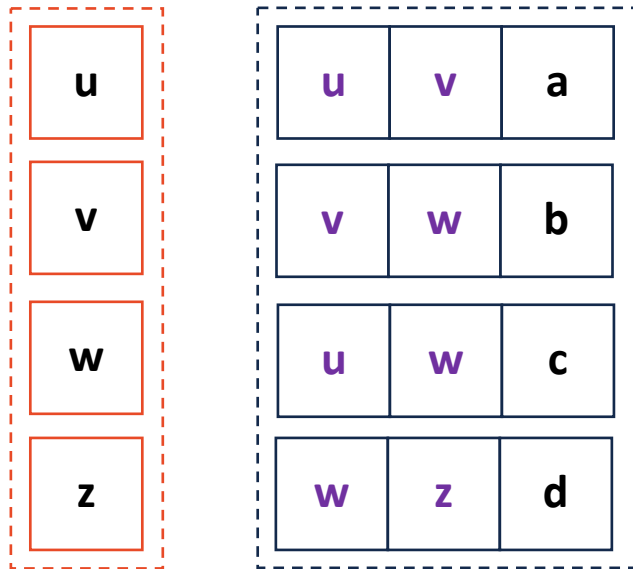
Graph Implementation: Edge List $|V| = n, |E| = m$

The equivalent of an 'unordered' data structure



Vertex Storage:

An optional list of vertices

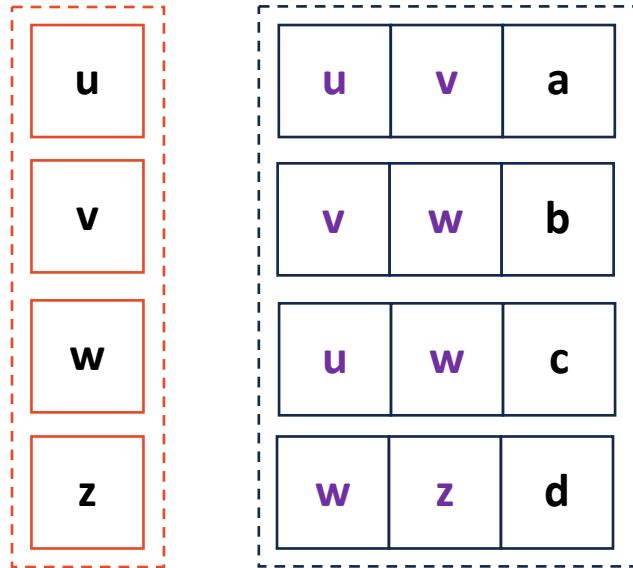
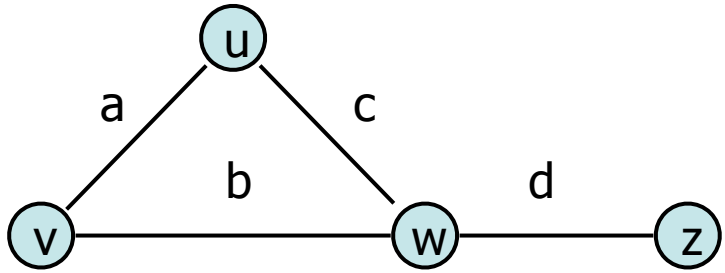


Edge Storage:

A list storing edges as (V1, V2, Weight)

Most graphs are stored as just an edge list!

Graph Implementation: Edge List $|V| = n, |E| = m$



insertVertex(K key)

insertEdge(Vertex v1, Vertex v2, K key)

incidentEdges(Vertex v)

areAdjacent(Vertex v1, Vertex v2)

removeVertex(Vertex v)

removeEdge(Vertex v1, Vertex v2)

Graph Implementation: Edge List



Pros:

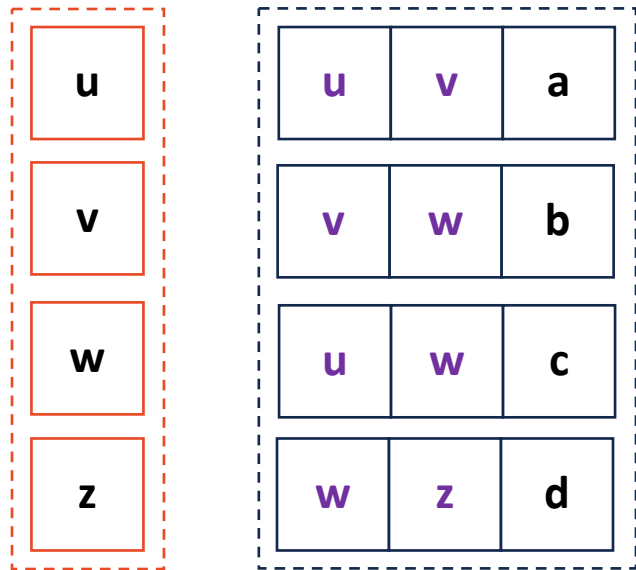
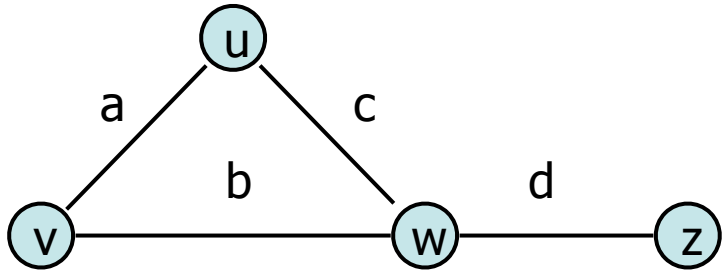
Cons:

Graph Implementation: Brainstorming better

What operations might I want to do very quickly?

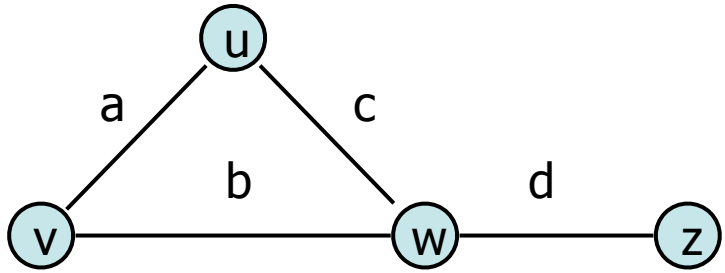
What modifications might allow me to do these things faster?

Graph Implementation: Adjacency Matrix



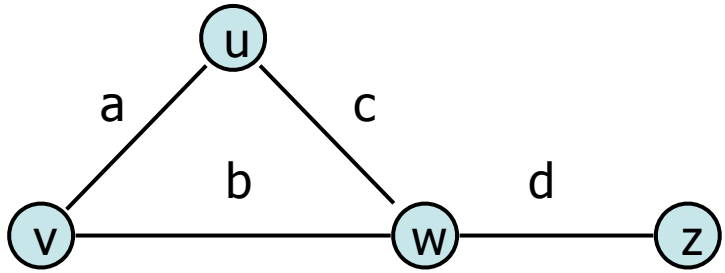
	u	v	w	z
u				
v				
w				
z				

Graph Implementation: Adjacency Matrix



	u	v	w	z
u				
v				
w				
z				

Graph Implementation: Adjacency Matrix



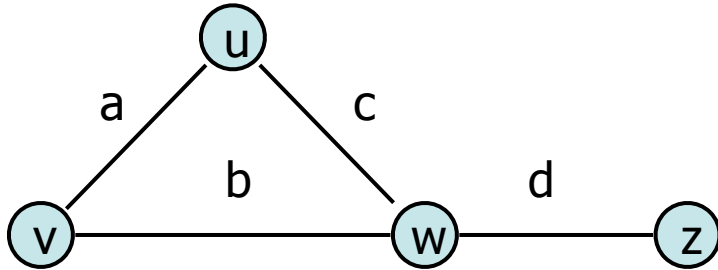
u	0
v	1
w	2
z	3

	0	1	2	3
0	-	a	c	0
1		-	b	0
2			-	d
3				-



Graph Implementation: Adjacency Matrix

$$|V| = n, |E| = m$$



Vertex Storage:

A hash table of vertices

Implicitly or explicitly store index

Edge Storage:

A list (or hash table) of vertices

Implicitly or explicitly store index

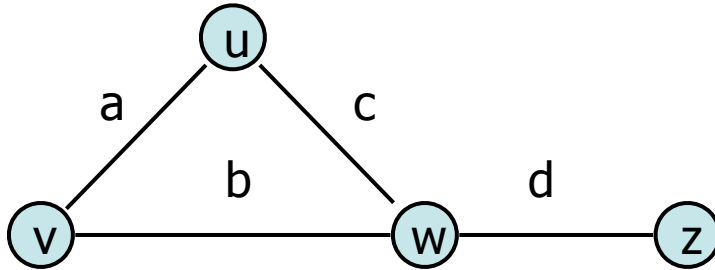
u	0
v	1
w	2
z	3

	0	1	2	3
0	-	a	c	0
1		-	b	0
2			-	d
3				-

Graph Implementation: Adjacency Matrix

$|V| = n, |E| = m$

incidentEdges(Vertex v):



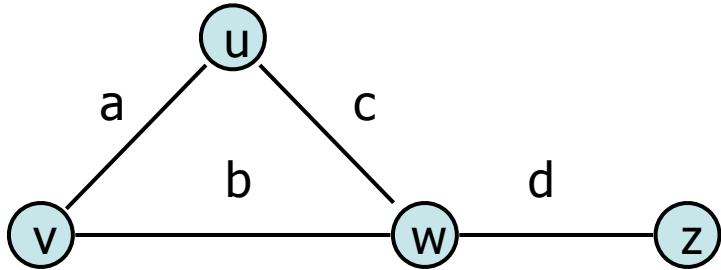
areAdjacent(Vertex v1, Vertex v2):

u	0
v	1
w	2
z	3

	0	1	2	3
0	-	a	c	0
1		-	b	0
2			-	d
3				-

Graph Implementation: Adjacency Matrix

$|V| = n, |E| = m$



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

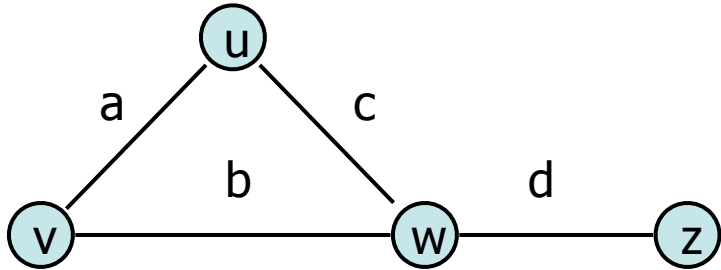
removeEdge(Vertex v1, Vertex v2, K key):

u	0
v	1
w	2
z	3

	0	1	2	3
0	-	a	c	0
1		-	b	0
2			-	d
3				-

Graph Implementation: Adjacency Matrix

$|V| = n, |E| = m$



insertVertex(K key):

removeVertex(Vertex v):

u	0
v	1
w	2
z	3

	0	1	2	3
0	-	a	c	0
1		-	b	0
2			-	d
3				-

Graph Implementation: Adjacency Matrix



Pros:

Cons:

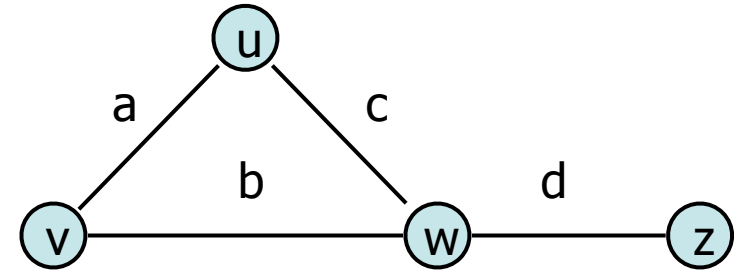
Graph Implementation Brainstorming

We want something...

Faster than an edge list

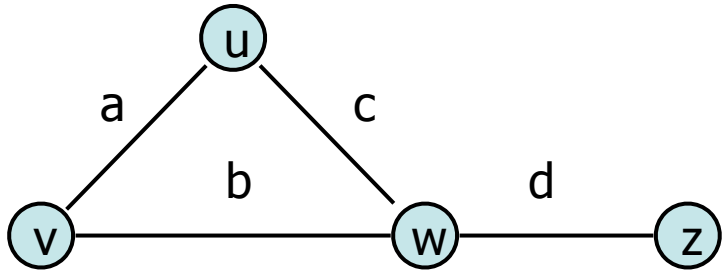
Less space than an adjacency matrix

Particularly good at **finding all adjacent elements (neighbors)**



Graph Implementation: Edge List + ?

$$|V| = n, |E| = m$$

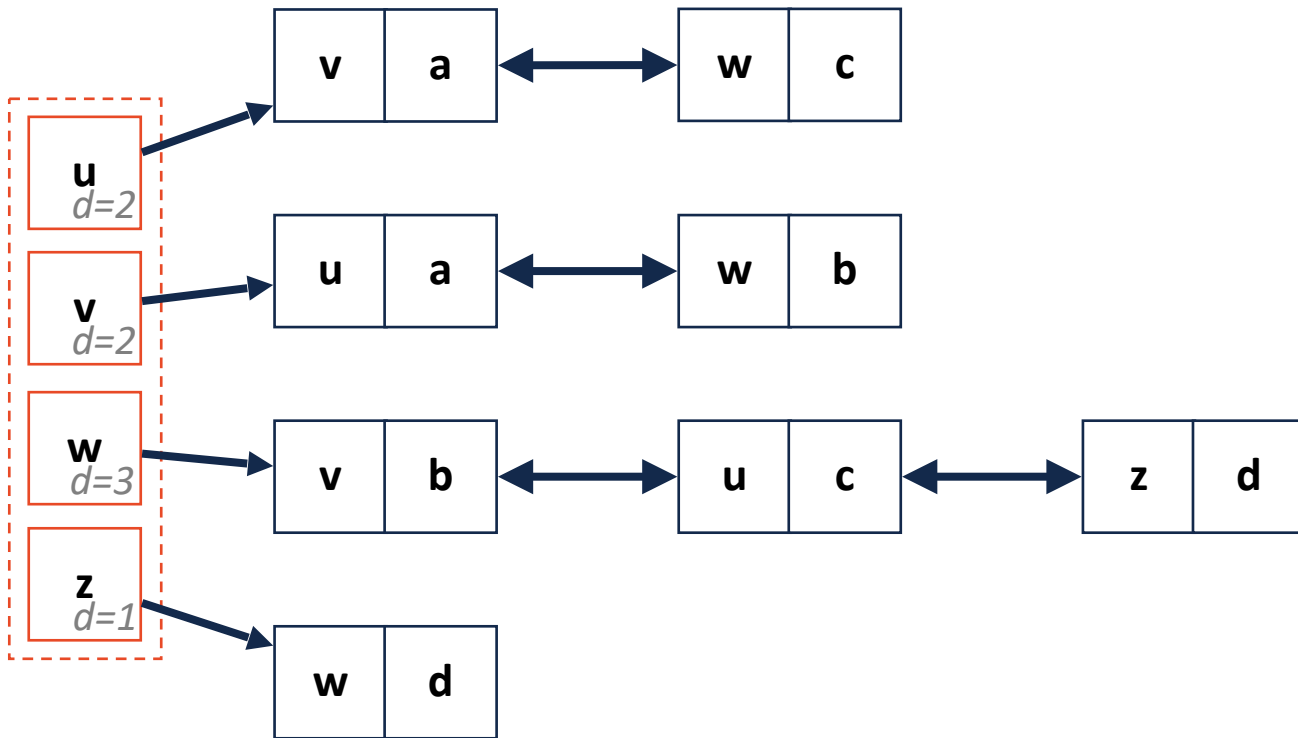
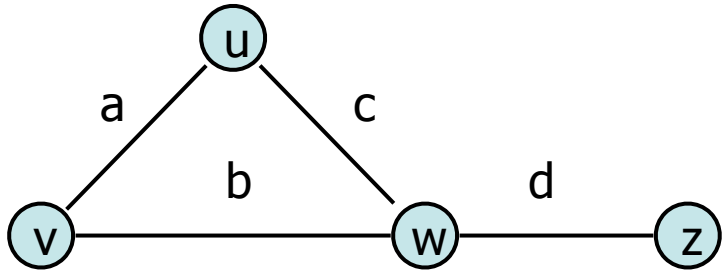


u
v
w
z

u	v	a
v	w	b
u	w	c
w	z	d

Graph Implementation: Adjacency List

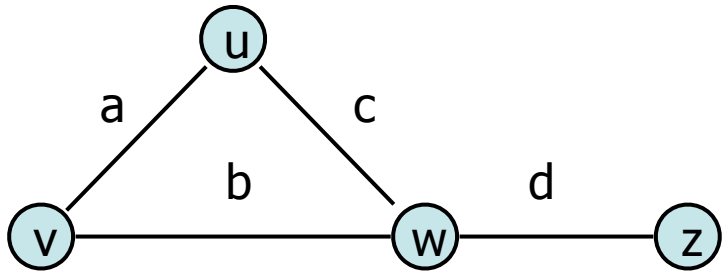
$$|V| = n, |E| = m$$



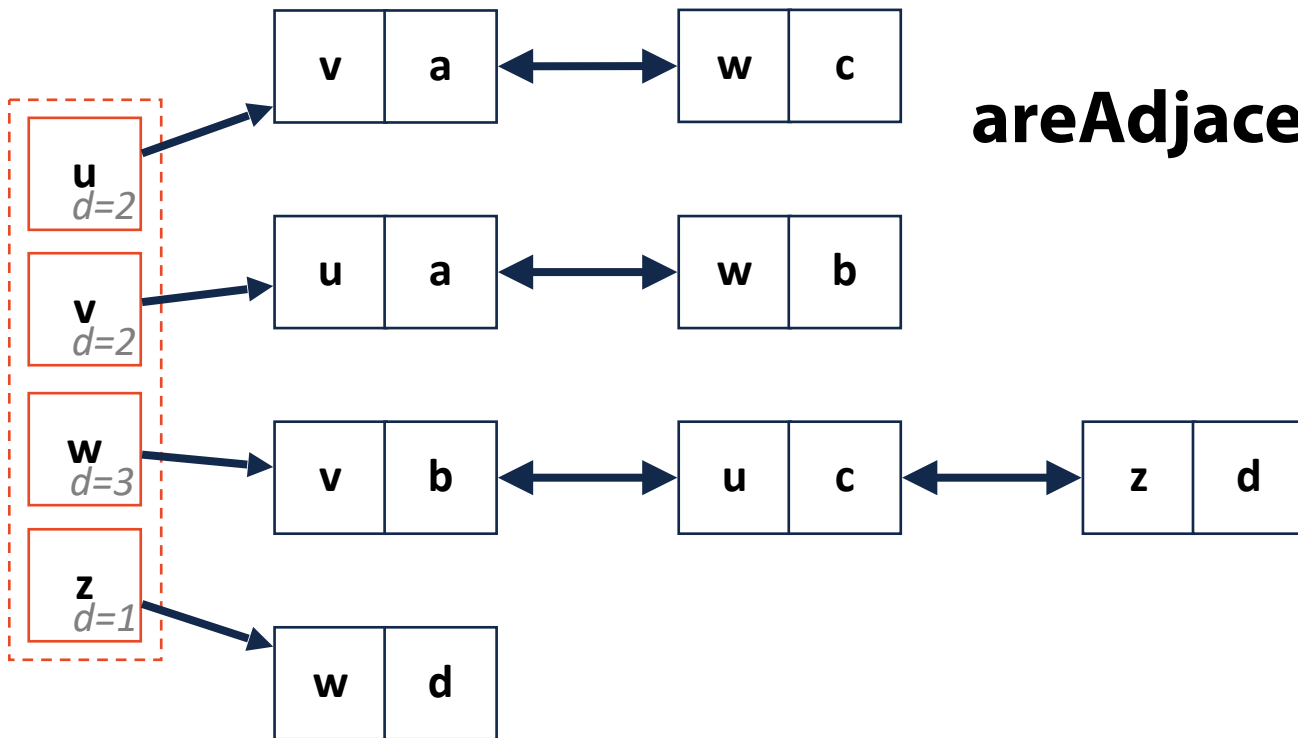
Graph Implementation: Adjacency List

$|V| = n, |E| = m$

incidentEdges(Vertex v):



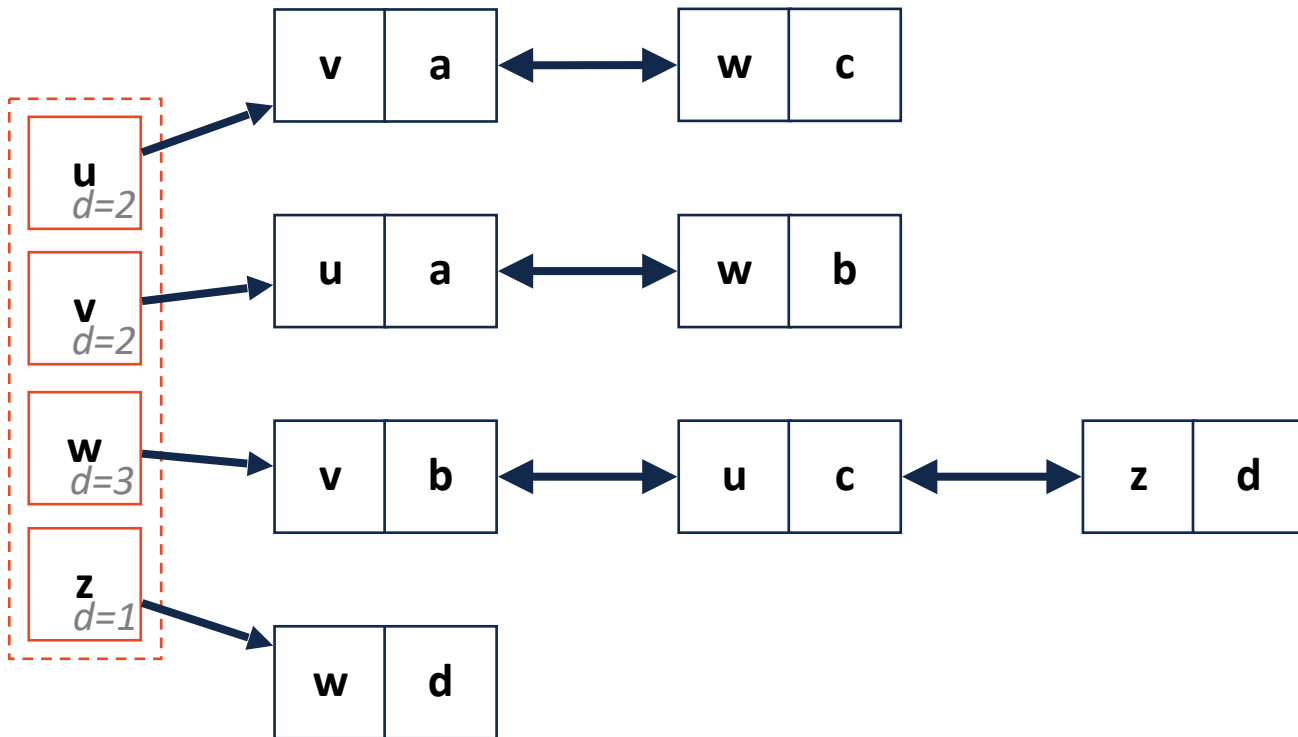
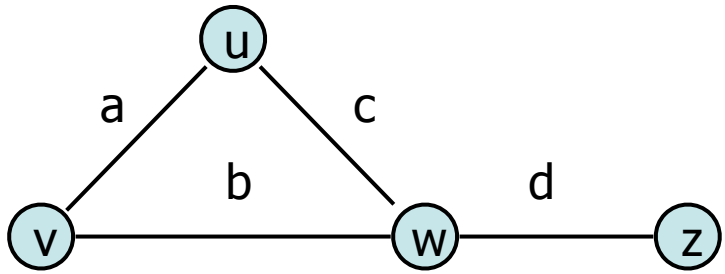
areAdjacent(Vertex v1, Vertex v2):



Graph Implementation: Adjacency List

$|V| = n, |E| = m$

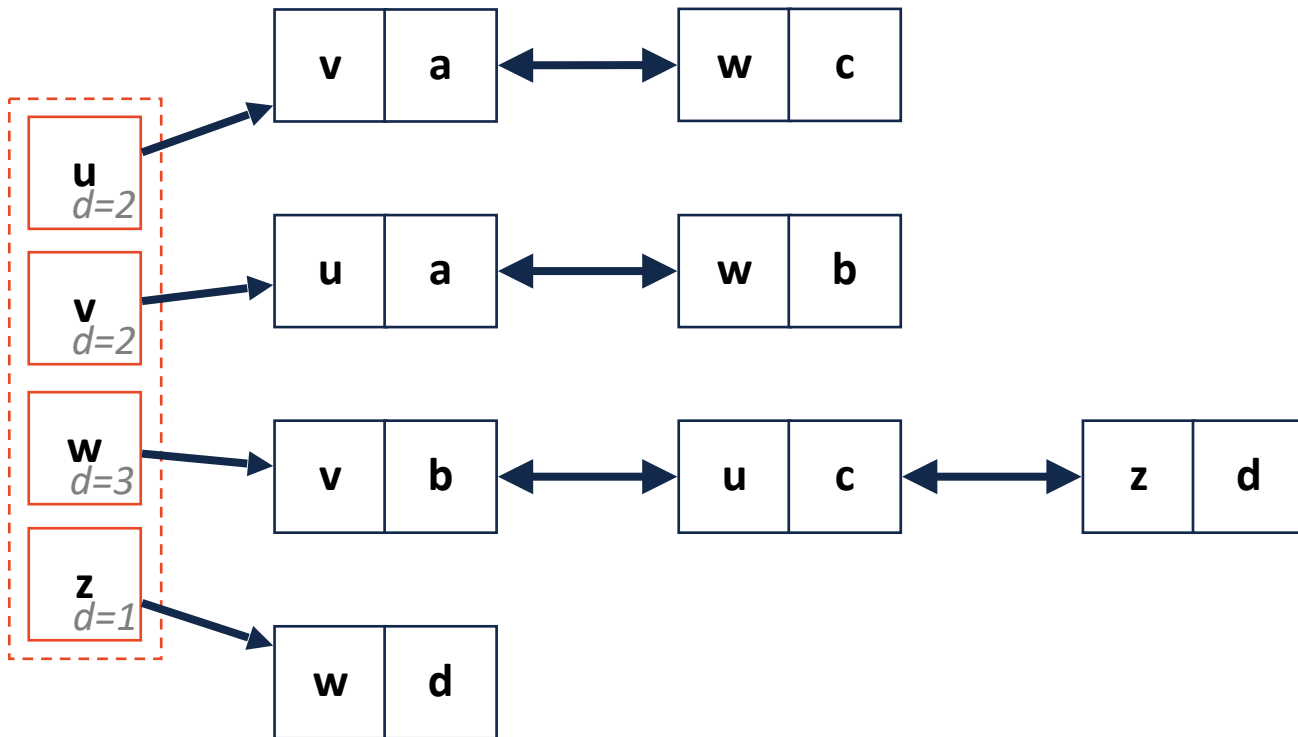
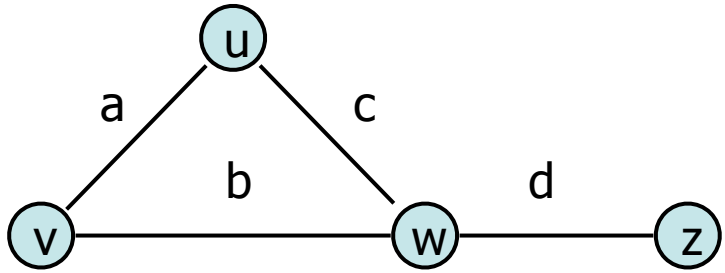
removeEdge(Vertex v1, Vertex v2, K key):



Graph Implementation: Adjacency List

$$|V| = n, |E| = m$$

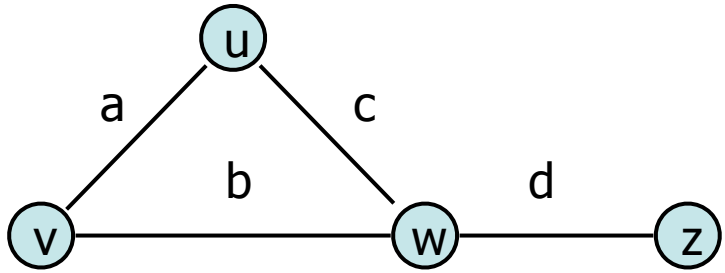
removeVertex(Vertex v):



Graph Implementation: Adjacency List



$$|V| = n, |E| = m$$



What's wrong with our implementation?

How can we fix it?

