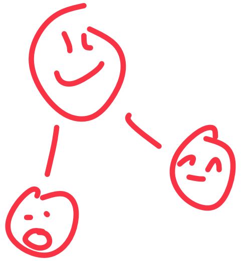
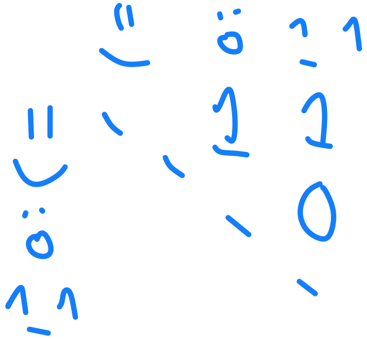


Data Structures

Graph Implementations

CS 225
Brad Solomon

October 23, 2024



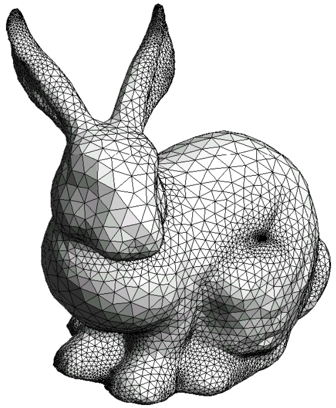
UNIVERSITY OF
ILLINOIS
URBANA-CHAMPAIGN

Department of Computer Science

Learning Objectives

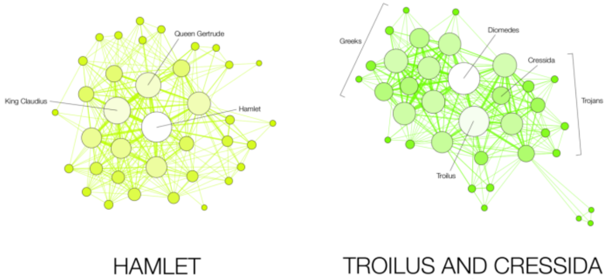
Discuss graph implementation and storage strategies

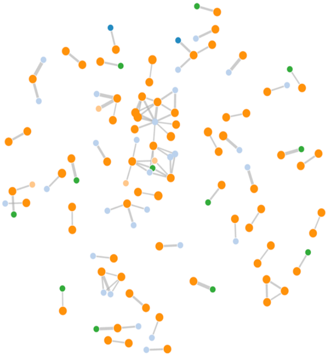
Graphs

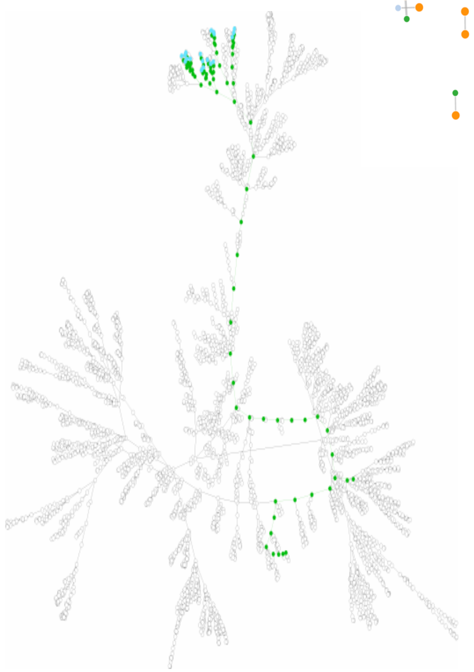


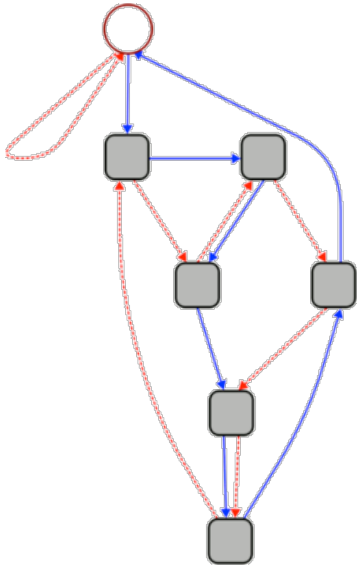
To study all of these structures:

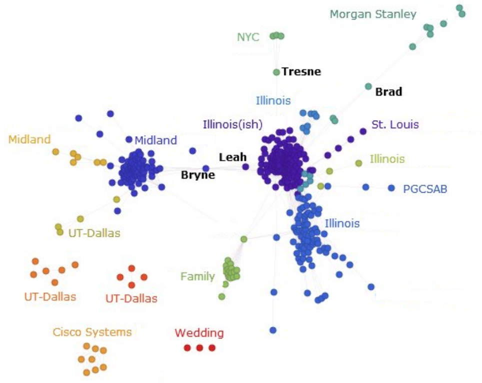
1. A common vocabulary ✓
2. Graph implementations
3. Graph traversals
4. Graph algorithms

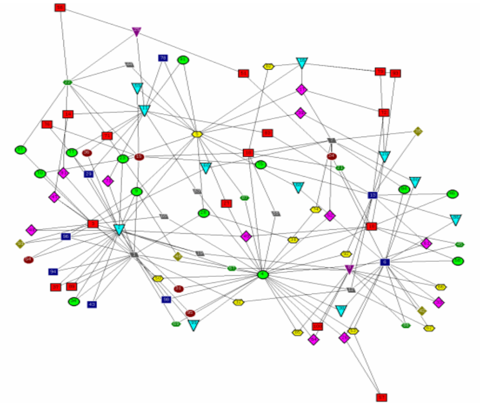








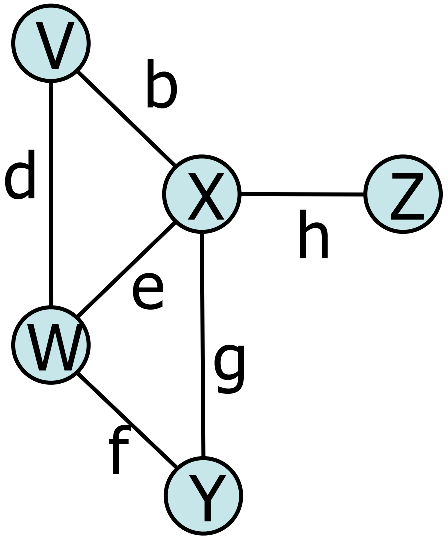




Graph ADT

Data:

- **Vertices** $|V| = n$
- **Edges** $|E| = m$
- **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);**

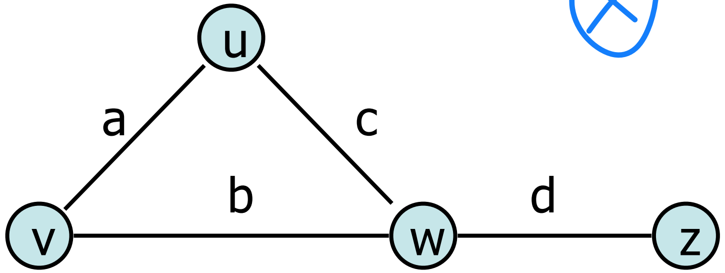
find

- **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



u
v
w
z

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

Vertex Storage:

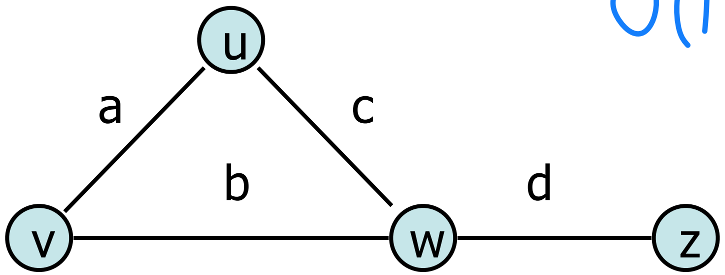
An optional list of vertices

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



|n|

u
v
w
z

x

|m|

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

x z w2

O(1)*

insertVertex(K key)

insertEdge(Vertex v1, Vertex v2, K key)

O(m)

incidentEdges(Vertex v)

areAdjacent(Vertex v1, Vertex v2)

removeVertex(Vertex v) O(n+m)

removeEdge(Vertex v1, Vertex v2)

Graph Implementation: Edge List

Pros:

↳ simple to implement | easy to store ≡ minimal storage cost

↳ Adding edge or vertex is $O(1)^*$

Cons:

↳ Hard to use as a graph

↳ cant lookup vertices easily

↳ edges easily

Graph Implementation: Brainstorming better

What operations might I want to do very quickly?

↳ Find / look up edge or vertex quickly

↳ Removal of vertices & edges

No wrong answer!

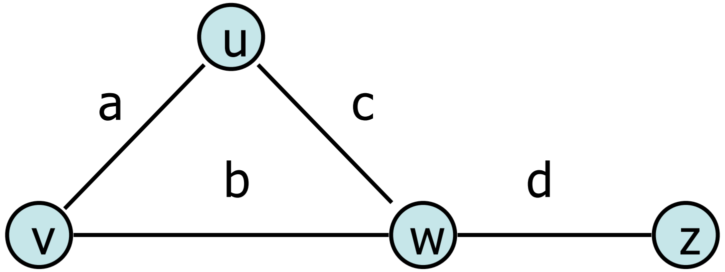
What modifications might allow me to do these things faster?

↳ If unsorted ≡ edge list, what is 'sorted'

unordered ≡ what is 'ordered'?

Graph Implementation: Adjacency Matrix

↳ Improve   is adjacent(u,v)

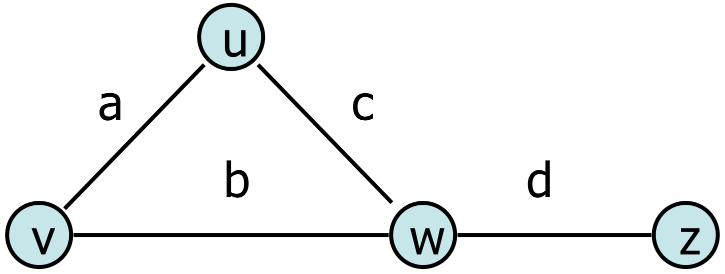


u
v
w
z

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

	u	v	w	z
u	F	T	T	F
v	T	F	T	F
w	T	T	F	T
z	F	F	T	F

Graph Implementation: Adjacency Matrix



u

v

w

z

0

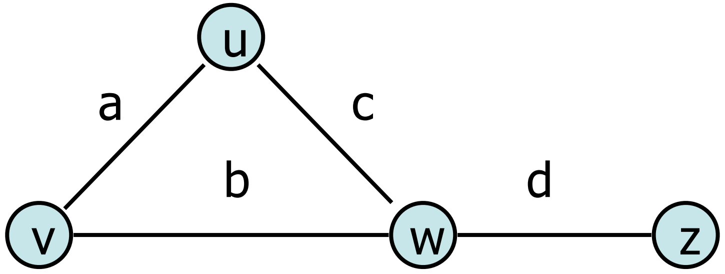
2

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

Diagonal Mirror!

↳ Store upper diagonal only

Graph Implementation: Adjacency Matrix



Implicity or explicity

↳ Hash table

↳ Vector

(vertex label , index)

↗ O(1)***

O(1)

u ← i=0

v ← i=1

w

z

Vector = ordered

u	0
v	1
w	2
z	3

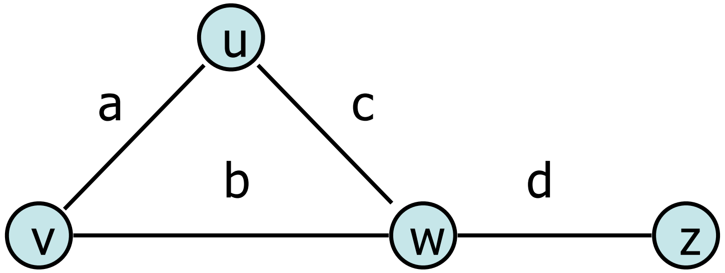
	0	1	2	3
0	-	a	c	0
1		-	b	0
2			-	d
3				-

← Not stored!

Point: Can lookup (u,v) in O(1) time

Graph Implementation: Adjacency Matrix

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



Storage big O

Vertex Storage:

A hash table of vertices

Implicitly or explicitly store index

$O(1)$

u	0
v	1
w	2
z	3

$n-3 + n-2 + n-1 \approx \frac{n(n-1)}{2} \approx O(n^2)$

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

Edge Storage:

A matrix of edges (size n)

Weight is stored at position (u, v)

$O(n^2)$

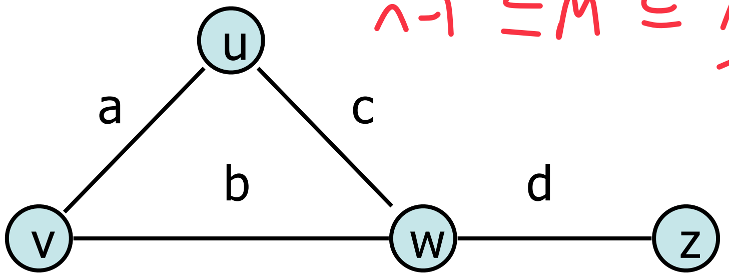
$O(\frac{n^2}{2}) \approx O(n^2)$

upper diagonal only for undirected graphs

Graph Implementation: Adjacency Matrix

|V|= n, |E|= m

$n-1 \le m \le \sim n^2$



u	0
v	1
w	2
z	3

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

d

(get Neighbors) w $O(n)$

incidentEdges(Vertex v):

If full matrix, look up row or col

upper diagonal, look up row and col

either way $O(n)$ Look up

w z

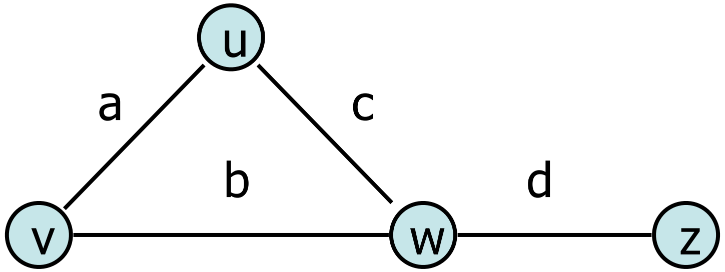
areAdjacent(Vertex v1, Vertex v2): $O(1)$

Look up specific coords in 2D matrix

!! A big win!

Graph Implementation: Adjacency Matrix

|V|= n, |E|= m



u	0
v	1
w	2
z	3

	0	1	2	3
0	-	a	c	~~0~~ 5
1		-	~~b~~ 0	0
2			-	d
3				-

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

↳ Look up (v_1, v_2) & replace weight

$O(1)$ ☺

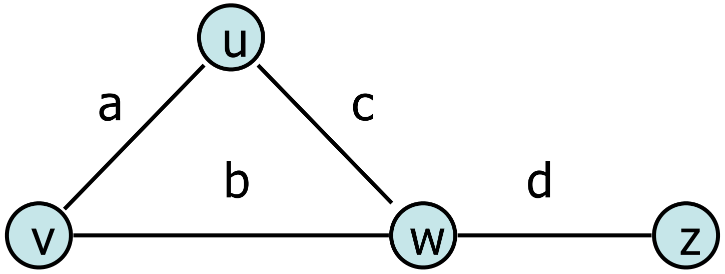
removeEdge(Vertex v1, Vertex v2, K key):

↳ Look up (v_1, v_2) & replace weight

$O(1)$ ☺

Graph Implementation: Adjacency Matrix

|V|= n, |E|= m



u	0
v	1
w	2
z	3
X	4

	0	1	2	3	4
0	-	a	c	0	
1		-	b	0	
2			-	d	
3				-	
4					

insertVertex(K key):

Add to vertex table $O(1)^*$

If full matrix, add row & column $O(n^2)$

upper diagonal, add col $O(n)$

removeVertex(Vertex v):

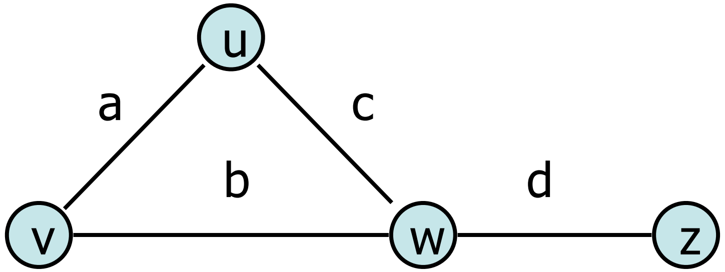
↳ Show is also $O(n)$

↳ Tombstoning to not resize array.

Graph Implementation: Adjacency Matrix

|V|= n, |E|= m

upper diagonal storage



u	0
v	1
w	2
z	3

X 4

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

4

5

4

As cols

V= [A]

W= [c, b]

Z = [0, 0, d]

X = [-, -, -, -]

O(n)

As rows

u= [a, c, 0]

V= [b, 0]

W= [d]

Z= [ ]

O(1)*

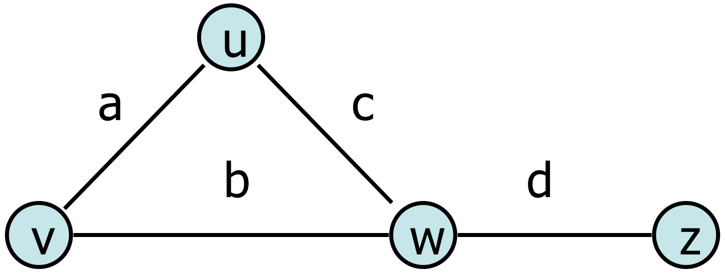
Graph Implementation: Adjacency Matrix

Pros:

Cons:

Graph Implementation Brainstorming

We want something…



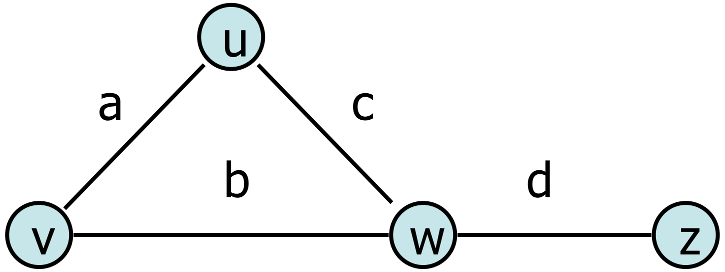
Faster than an edge list

Less space than an adjacency matrix

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

Graph Implementation: Edge List + ?

|V|= n, |E|= m

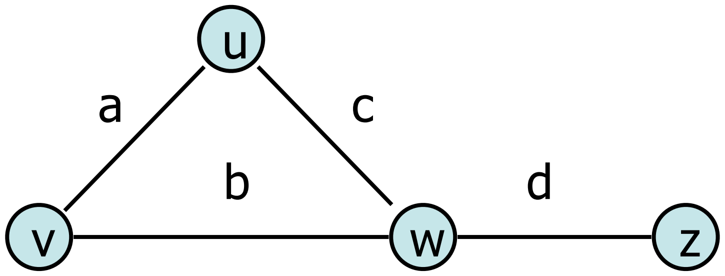


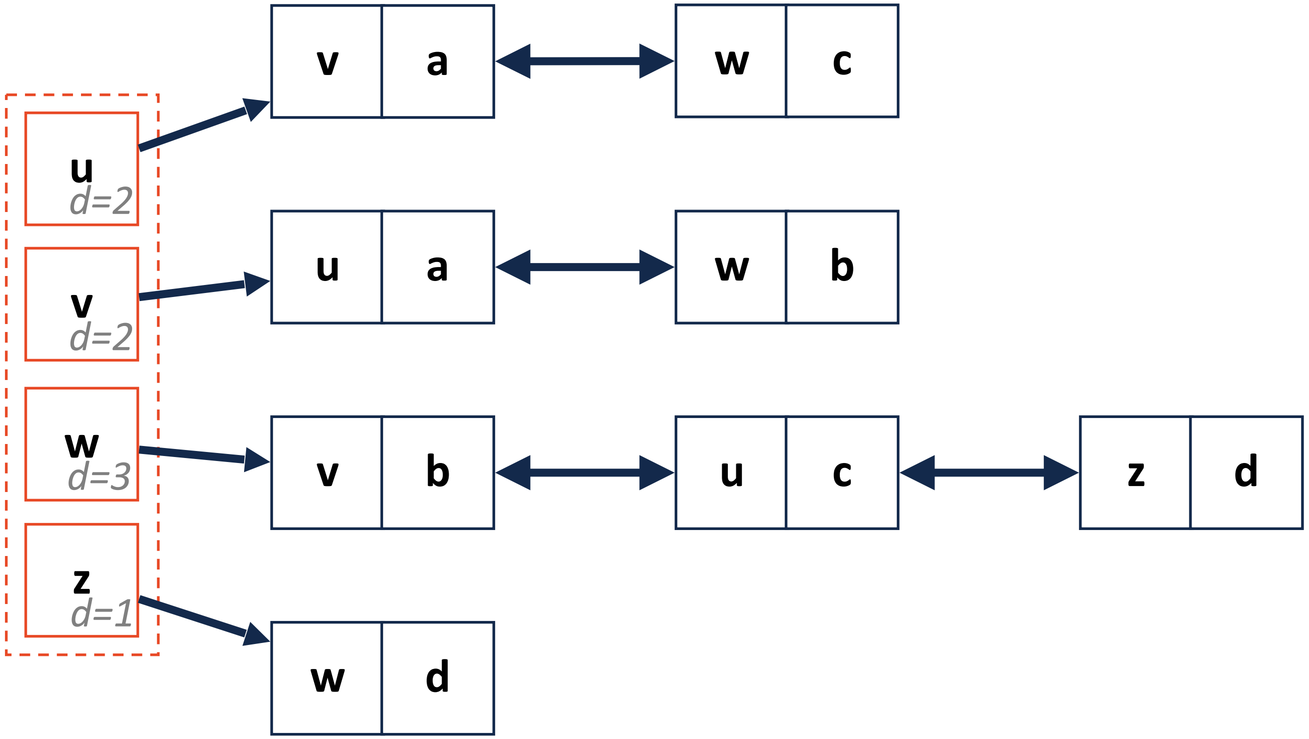
u
v
w
z

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

Graph Implementation: Adjacency List

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





u *d=2* → v a ⟷ w c

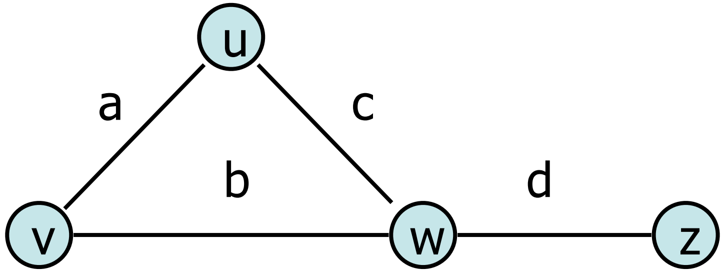
v *d=2* → u a ⟷ w b

w *d=3* → v b ⟷ u c ⟷ z d

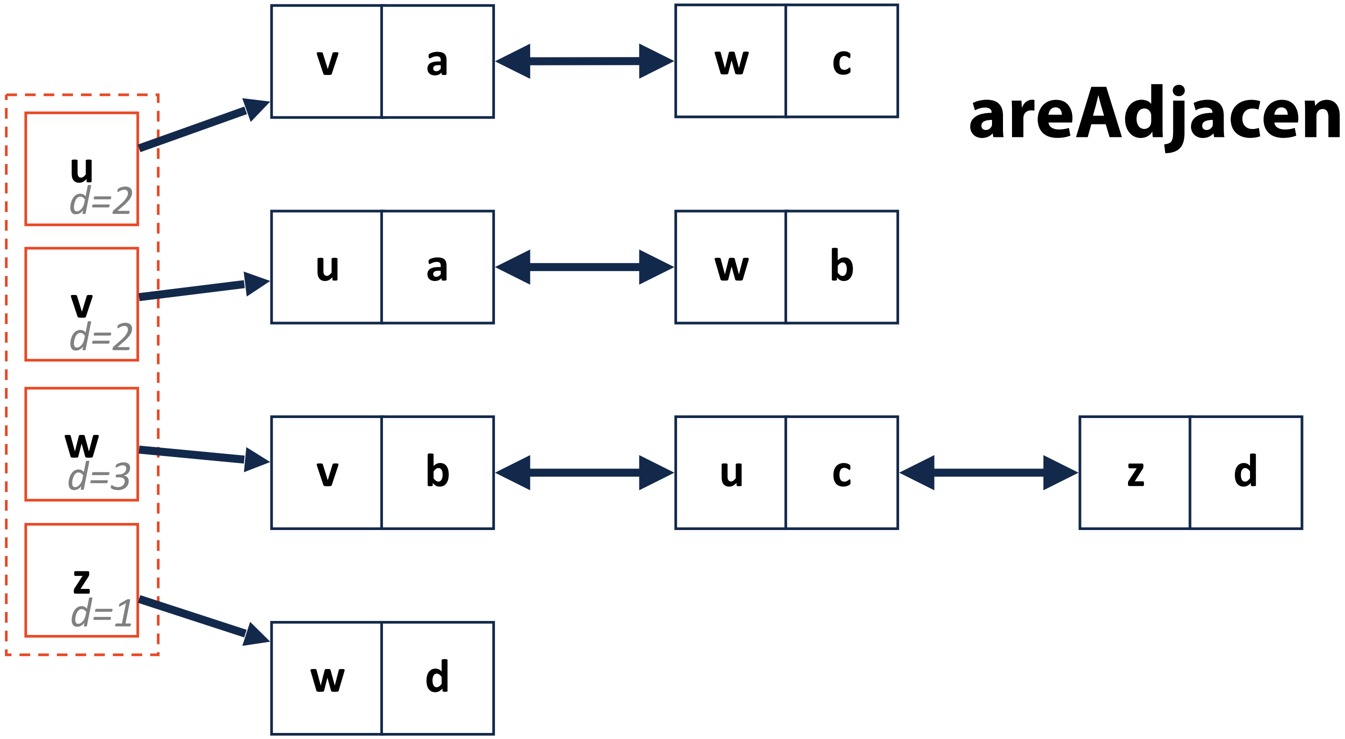
z *d=1* → w d

Graph Implementation: Adjacency List

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



incidentEdges(Vertex v):

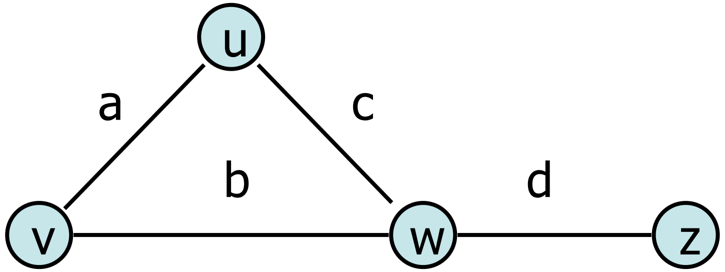


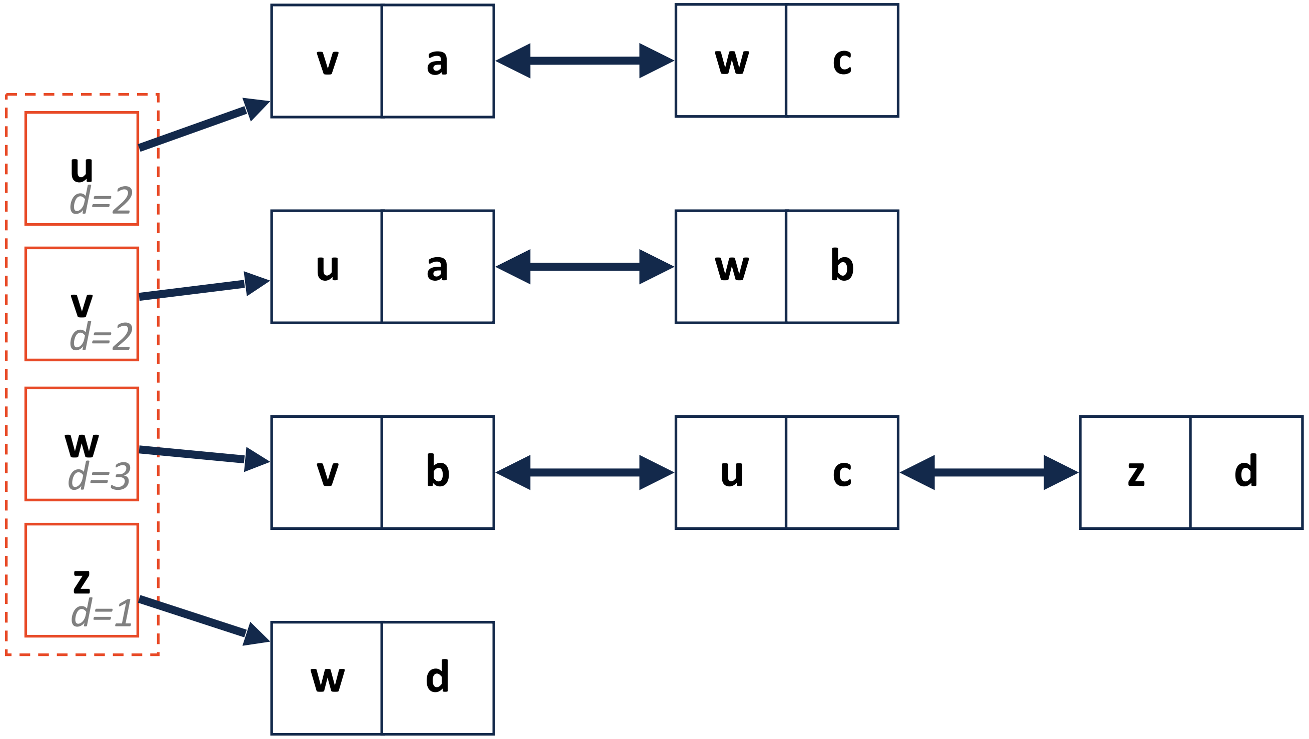
areAdjacent(Vertex v1, Vertex v2):

Graph Implementation: Adjacency List

|V|= n, |E|= m

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

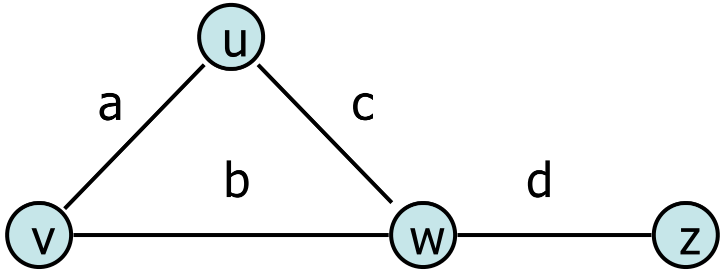


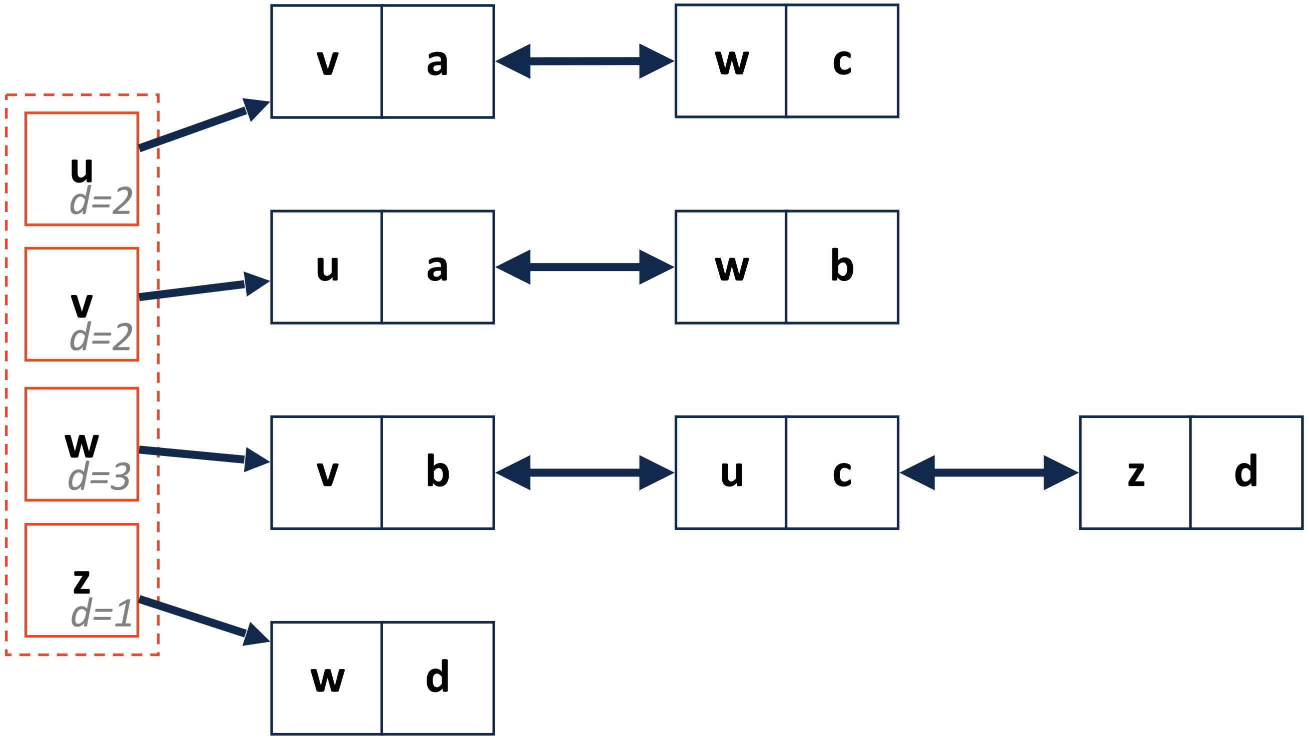


Graph Implementation: Adjacency List

|V|= n, |E|= m

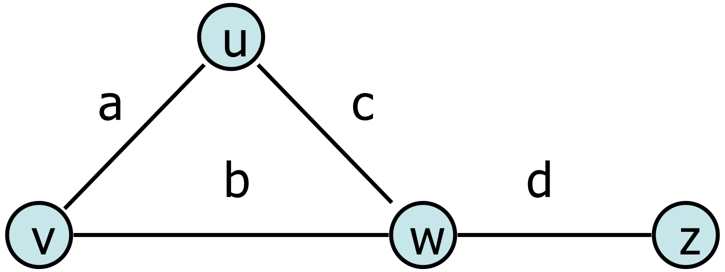
removeVertex(Vertex v):

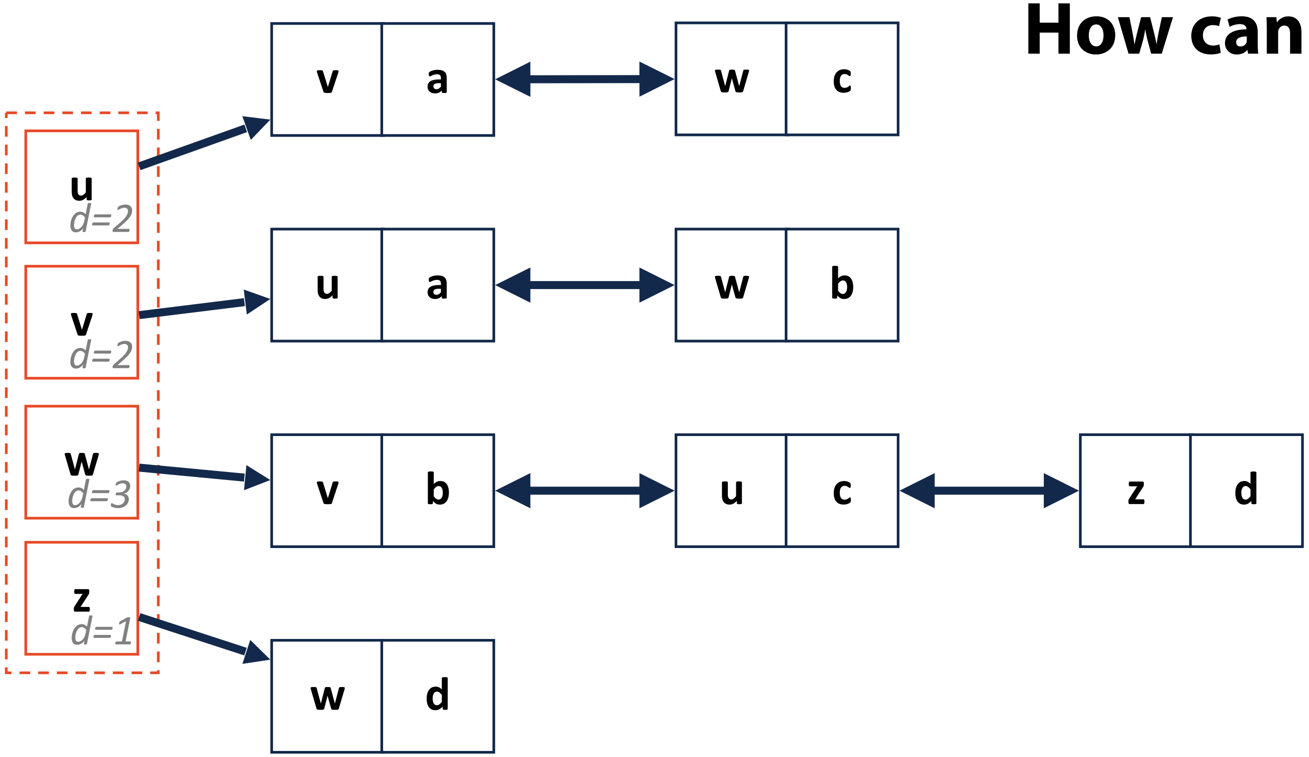




Graph Implementation: Adjacency List

|V|= n, |E|= m





What's wrong with our implementation?

How can we fix it?