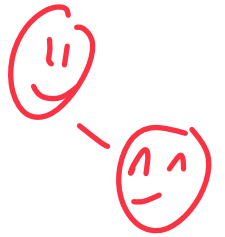
Data Structures

Graph Implementations



CS 225

Brad Solomon

March 30, 2026

Department of Computer Science

Exam 4 on 4/6 — 4/8

Autograded MC and one coding question

Manually graded short answer prompt

Practice exam releases this week on PL

Topics covered can be found on website

Register now!

https://courses.engr.illinois.edu/cs225/exams/

Lab this week is exam 4 review session

Like exam 2, some new questions will be released for lab

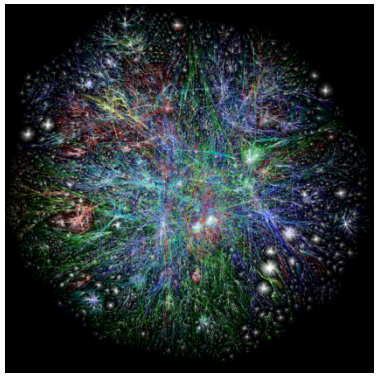
A great opportunity to go over practice exam too!

Form a study group to peer grade FRQs

Learning Objectives

Discuss graph implementation and storage strategies

Graphs

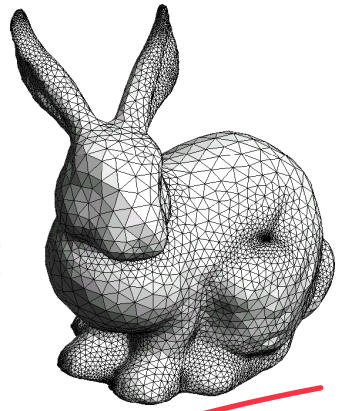


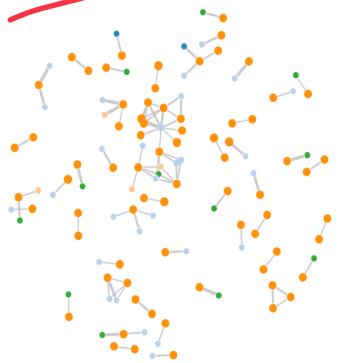
HAMLET

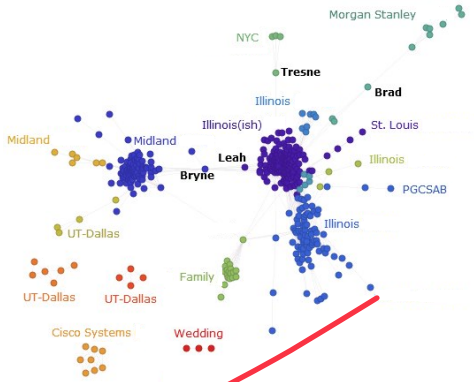
TROILUS AND CRESSIDA

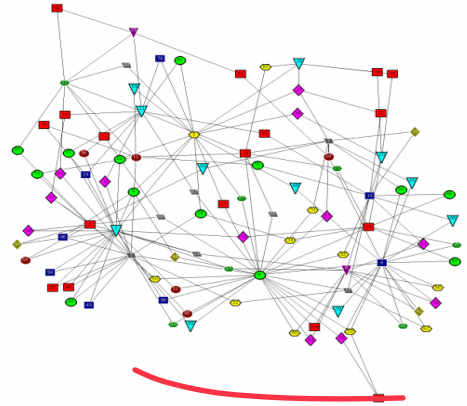
To study all of these structures:

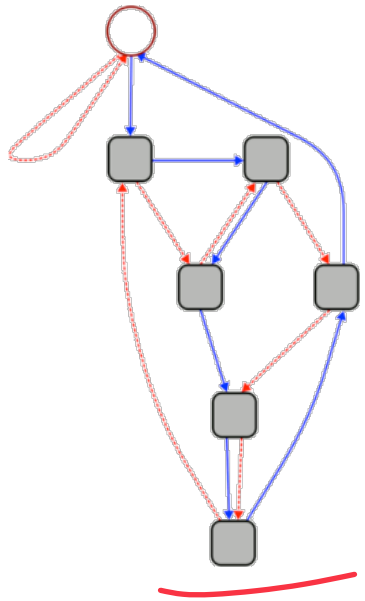
1. A common vocabulary
2. Graph implementations
3. Graph traversals
4. Graph algorithms

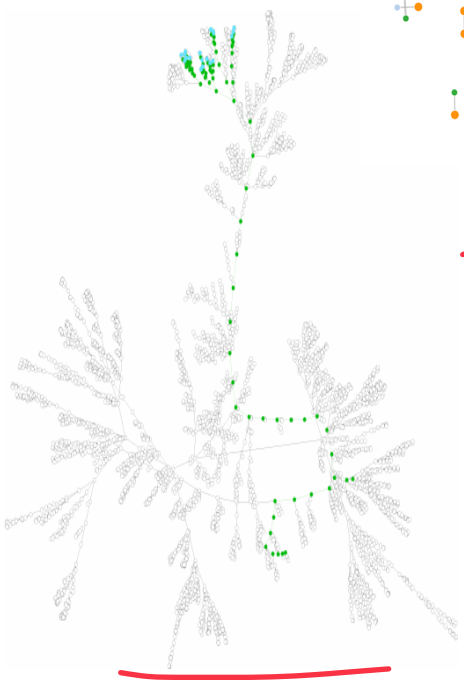








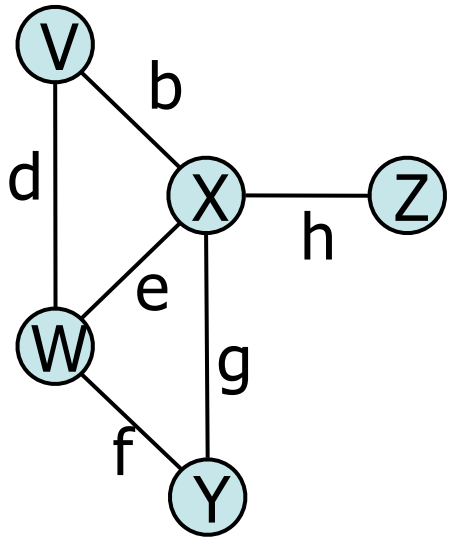




Graph ADT

Data:

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



Functions:

Tradeoffs!

- insertVertex(K key);
- insertEdge(Vertex v1, Vertex v2, K key);

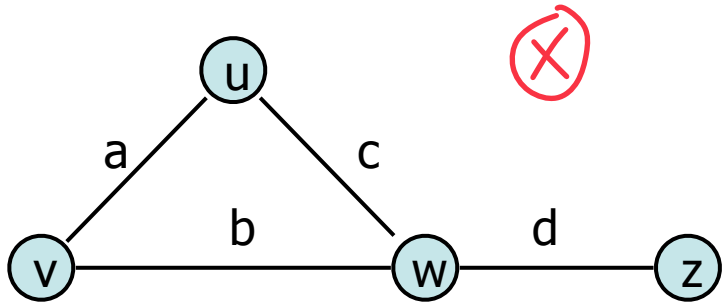
- removeVertex(Vertex v);
- removeEdge(Vertex v1, Vertex v2);

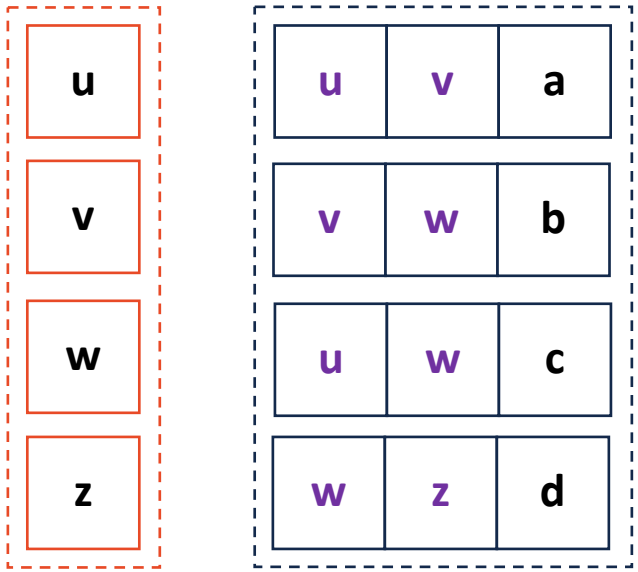
- incidentEdges(Vertex v);
- areAdjacent(Vertex v1, Vertex v2);

- origin(Edge e);
- destination(Edge e);

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

The equivalent of an 'unordered' data structure





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

Vertex Storage:

An optional list of vertices

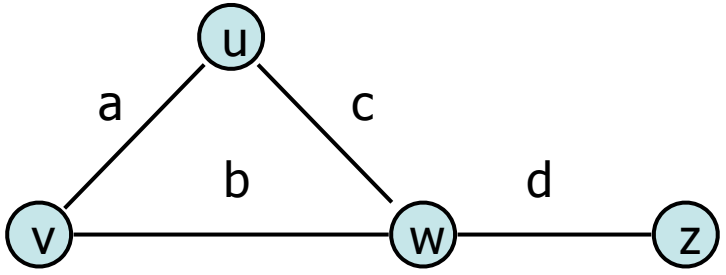
* ↳ optional only if all vertices connected

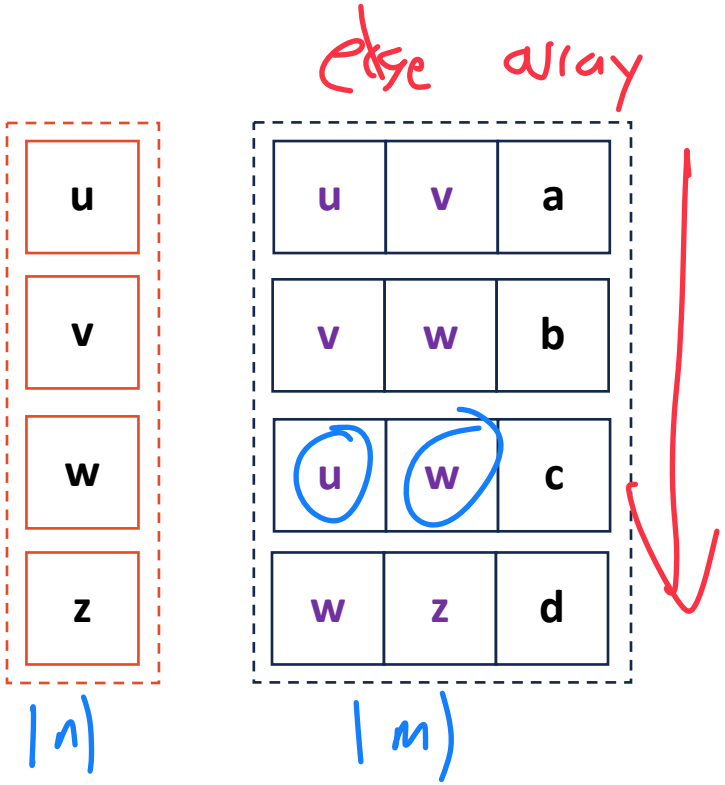
Edge Storage:

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

Most graphs are stored as just an edge list!

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





getEdges(Vertex v)

Search edge storage for 'v'

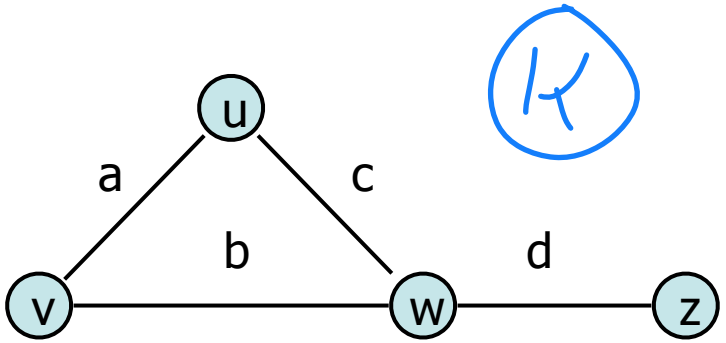
Since unsorted array: O(m)

areAdjacent(Vertex v1, Vertex v2)

Search edge storage for 'v1' AND 'v2'

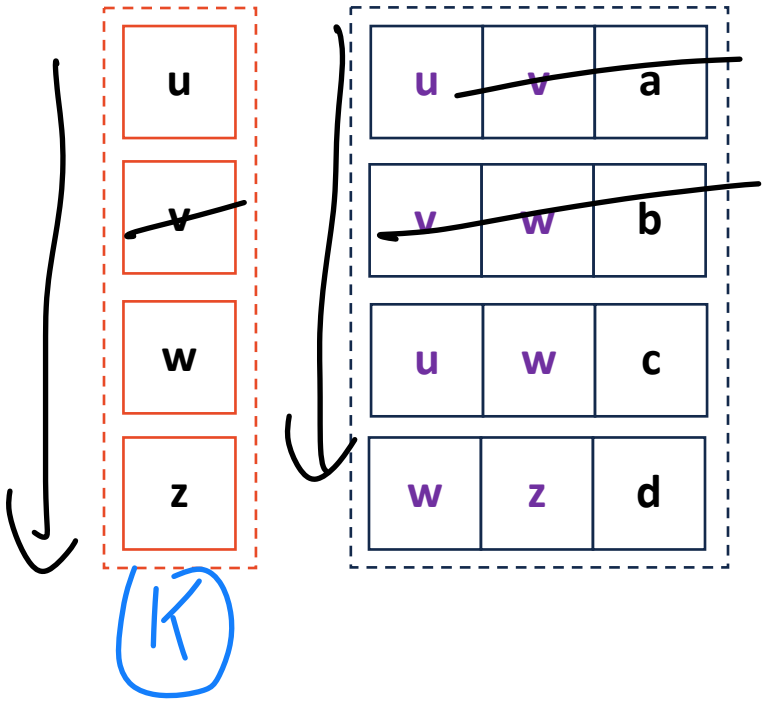
Since unsorted array: O(m)

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



insertVertex(K key)

↳ Add to end of vertex list

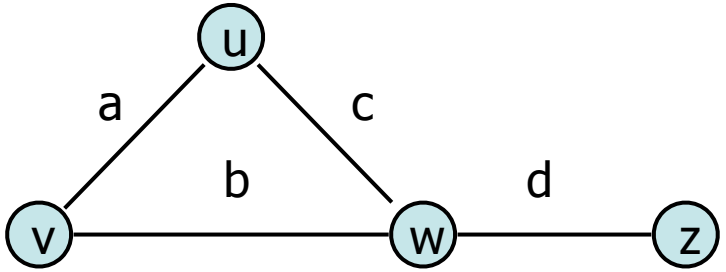


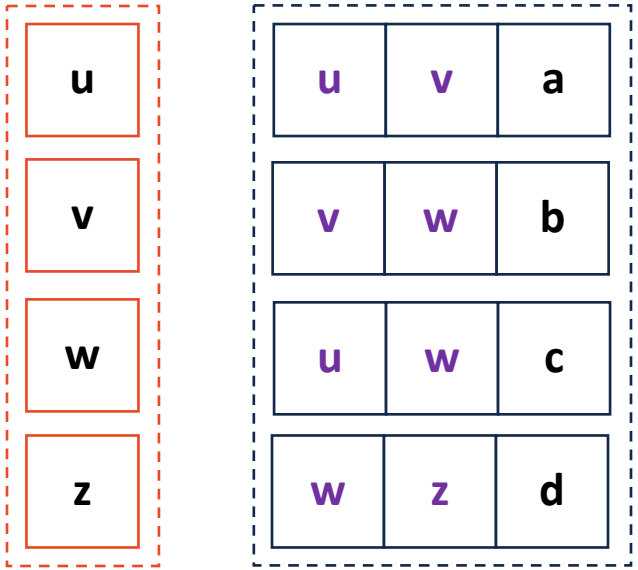
removeVertex(Vertex v)

↳ walk through vertex array, remove ✓

↳ walk through edge array, remove edges w/ v

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





insertVertex(K key)

Add 'key' to vertex array

Since unsorted array: O(1)*

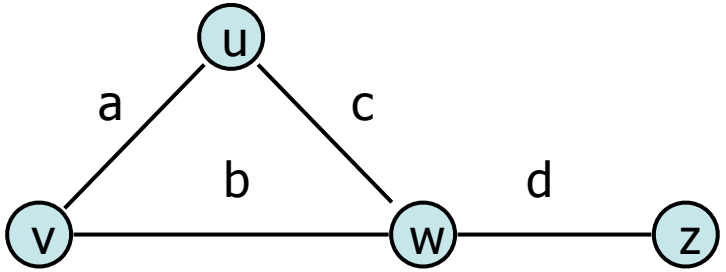
removeVertex(Vertex v)

Remove vertex from vertex array

Remove vertex from edge array

Since unsorted array: O(n+m)

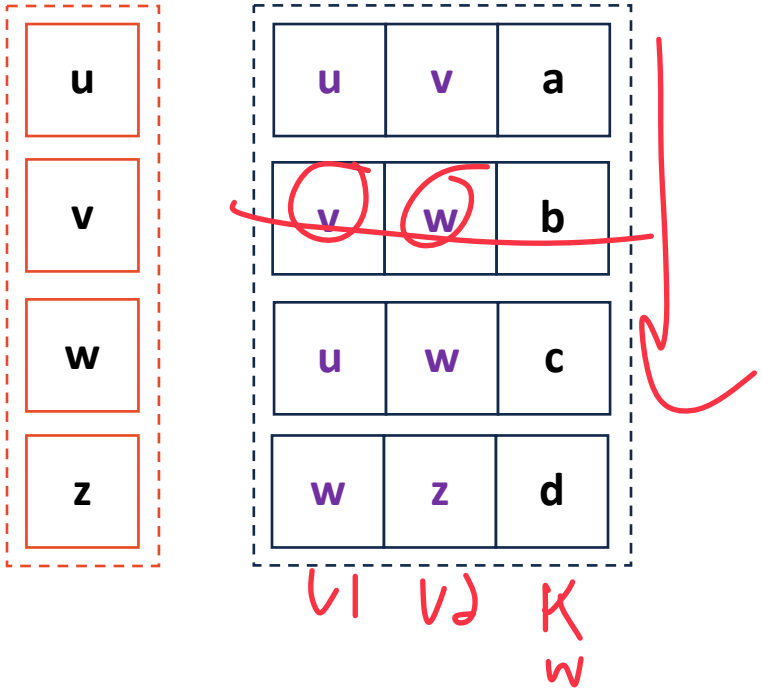
Graph Implementation: Edge List $|V| = n, |E| = m$



insertEdge(Vertex v1, Vertex v2, K key)

↳ Insert back to edge array

$O(1)^*$

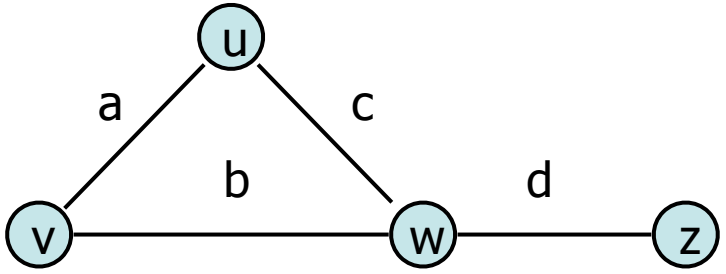


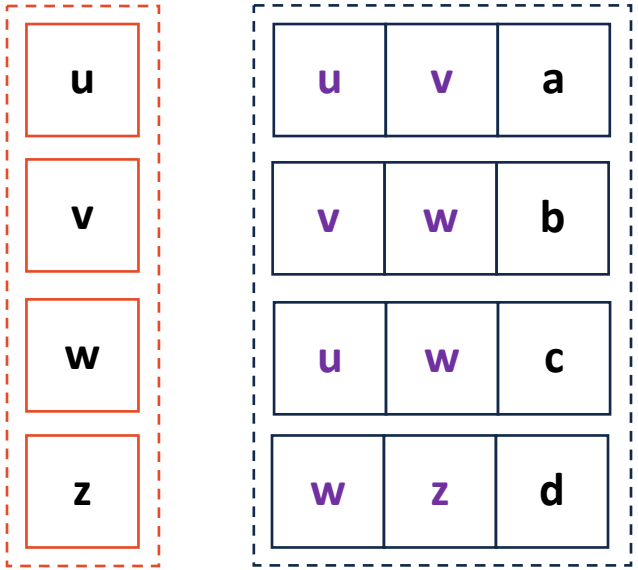
removeEdge(Vertex v1, Vertex v2)

↳ Find the edge

$O(m)$

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





insertEdge(Vertex v1, Vertex v2, K key)

Tricky! Can we assume edge doesn't exist?

If yes — easy: O(1)*

If no — still easy but slower: O(m)

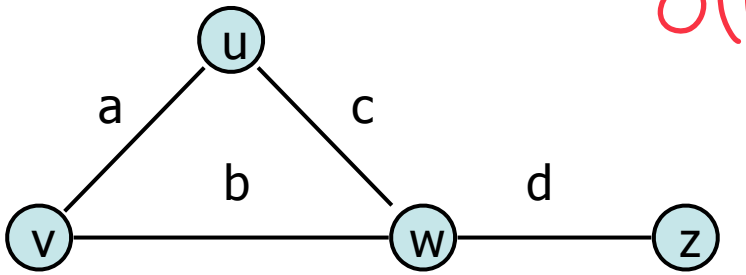
removeEdge(Vertex v1, Vertex v2)

Search edge array for 'v1' and 'v2'

Remove it if it exists

Since unsorted array: O(m)

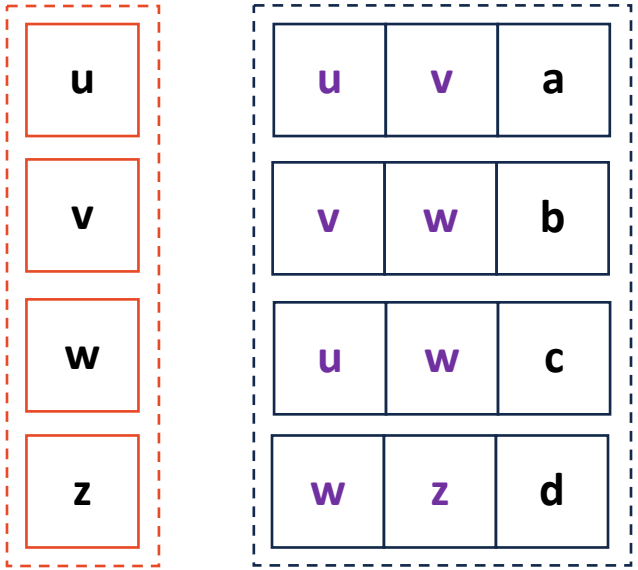
Graph Implementation: Edge List $|V| = n, |E| = m$



O(1)

insertVertex(K key)

insertEdge(Vertex v1, Vertex v2, K key)



O(n)

incidentEdges(Vertex v)

areAdjacent(Vertex v1, Vertex v2)

removeVertex(Vertex v) ← O(1+n)

removeEdge(Vertex v1, Vertex v2)

Graph Implementation: Edge List

$n-1 \leq m \leq n^2$

Pros:

- very fast vertex/edge insert ← * useful as storage structure
- Flexible → easy to implement/use
- Storage costs are minimal ← *

Cons:

- Remove Vertex very slow
- Most graph access functions slow O(n)

Graph Implementation: Brainstorming better

What operations might I want to do very quickly?

↳ Find specific vertex

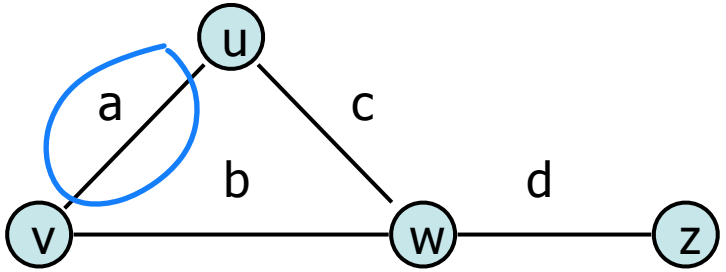
↳ Get edges (so we can traverse)

↳ Remove can be done faster (vertex / edge)

What modifications might allow me to do these things faster?

↳ Lets add the equivalent of 'sort' or 'order'

Graph Implementation: Adjacency Matrix



can be O(1) if done right!

T/F for if edge exists

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

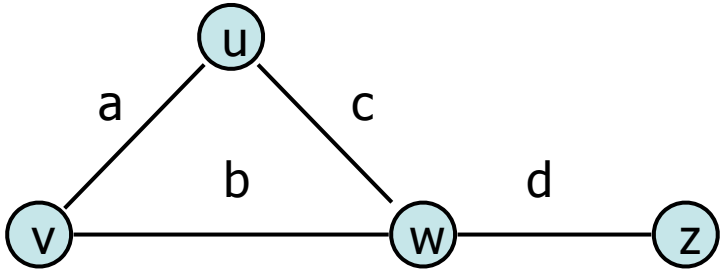
if u.d!=v.d+2

upper diagonal only

why not mirror both

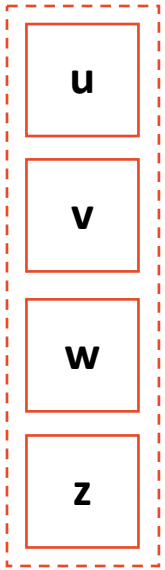
Graph Implementation: Adjacency Matrix

↳ Replace edge array w/ edge matrix!



ordered matrix

V



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

(2)

How can we get row/col index given label in O(1) time?

1) Use map — O(1)***

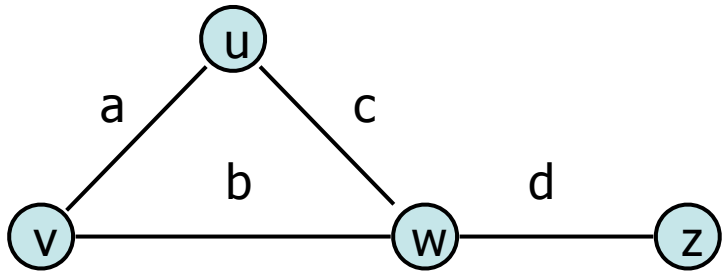
Vertex label → index

2) Implicit in array

V.index(u) → 0

O(1)

Graph Implementation: Adjacency Matrix



O(1)

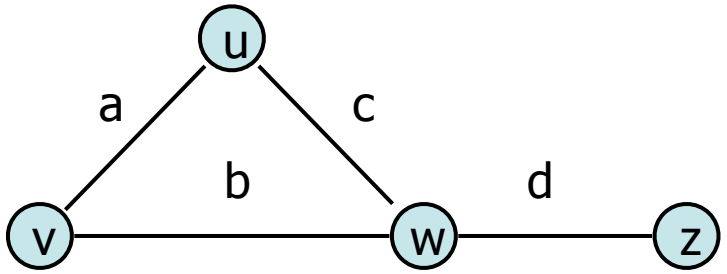
u	0
v	1
w	2
z	3

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

Any lookup in table in O(1) time! !!

Graph Implementation: Adjacency Matrix

|V|= n, |E|= m



Vertex Storage:

A 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				-

Edge Storage:

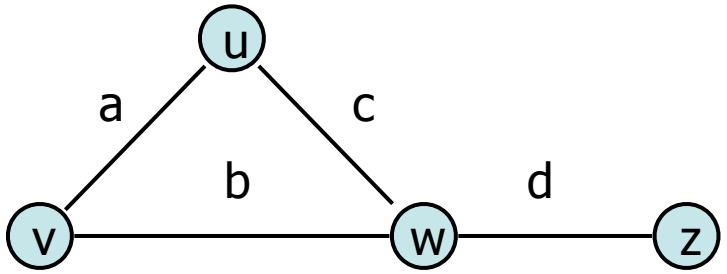
A |V| x |V| matrix of edges

Weight is stored at position (u, v)

Blank? Because undirected graph upper diagonal

Graph Implementation: Adjacency Matrix

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



u	0
v	1
w	2
z	3

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

incidentEdges(Vertex v):

↳ look up v row / col

areAdjacent(Vertex v1, Vertex v2):

↳ Look up in matrix

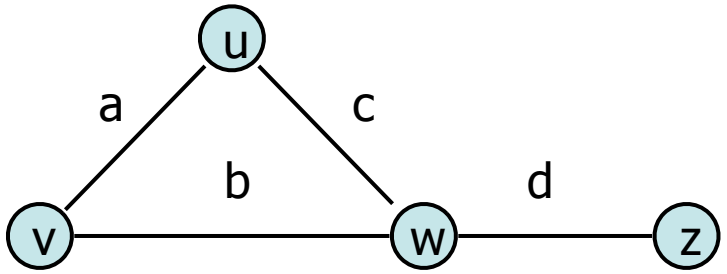
↳ implicitly lookup in map first

↳ Get index

Graph Implementation: Adjacency Matrix

Join Code: 225

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



u	0
v	1
w	2
z	3

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

incidentEdges(Vertex v):

↳ from 0 edges to n-1 edges will be run

Look up row (or column) in matrix

↳ # of items I look up is fixed to n vertices

↳ Always look through null spaces (n) $O(n)$

areAdjacent(Vertex v1, Vertex v2):

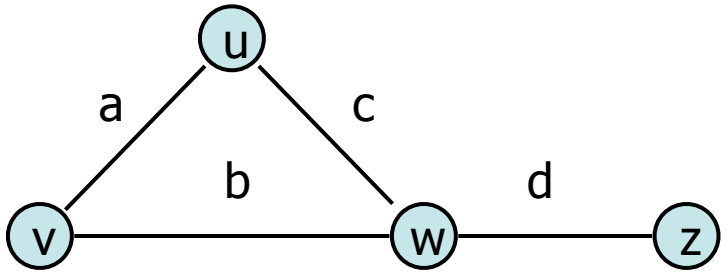
Look up row (and column) in matrix $O(1)$

$O(1)$ $O(n)$ $O(m)$ $O(n^2)$ $O(m^2)$

What are my Big Os?

Graph Implementation: Adjacency Matrix

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



u	0
v	1
w	2
z	3

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

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

↳ look up (v_1, v_2)

↳ change value

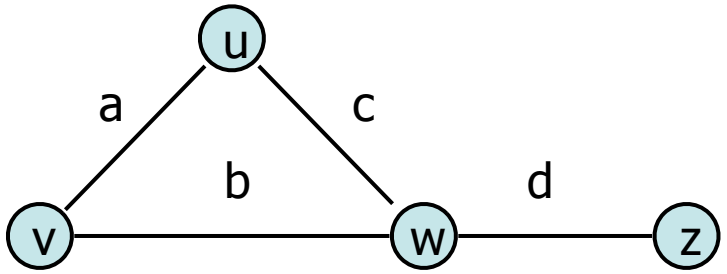
removeEdge(Vertex v1, Vertex v2, K key):

↳ look up specific (v_1, v_2)

↳ change value (to 0)

Graph Implementation: Adjacency Matrix

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



u	0
v	1
w	2
z	3

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

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

Look up row and column

Change value

O(1)

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

Look up row and column

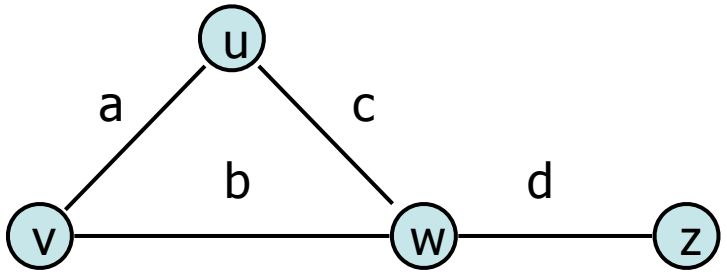
Change value

O(1)

What are my Big Os?

Graph Implementation: Adjacency Matrix

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



u	0
v	1
w	2
z	3

4 | 4

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

insertVertex(K key):

↳ Add new key to map

↳ Add new row to matrix

↳ Add new column to every row

removeVertex(Vertex v):

↳ find in map & remove

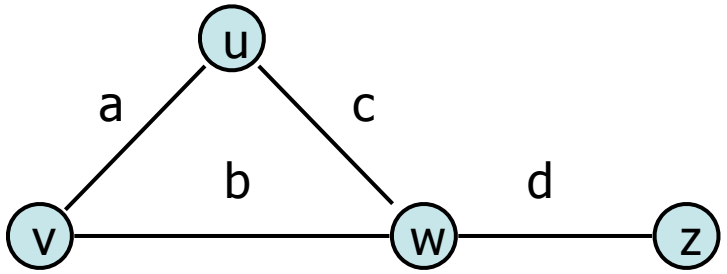
↳ remove a row & a column in every row

uh oh....

Graph Implementation: Adjacency Matrix

Join Code: 225

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



u	0
v	1
w	2
z	3

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

ready to delete

Tombstoning: Do this later remember it was done

insertVertex(K key):

Insert new key to vertex mapping

Add a new row and column to matrix

If rebuild matrix: $O(n^2)$

If upper diagonal as list of lists: $O(n)$

removeVertex(Vertex v): $O(n^2)$ overall

Remove v from vertex mapping $O(1)$***

Remove an entire row / entire column

↳ Often requires rebuilding matrix

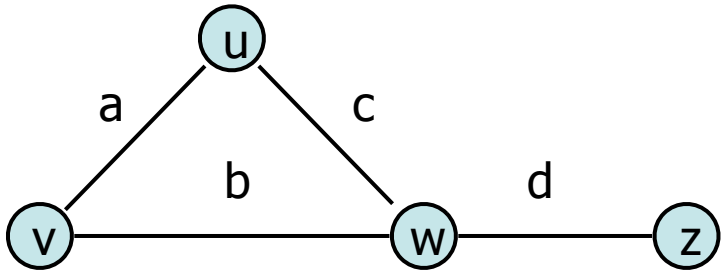
$\geq O(n^2)$

What are my Big Os?

Graph Implementation: Adjacency Matrix

Join Code: 225

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



u	0
v	1
~~w~~	~~2~~
z	3

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

Insert vertex?

upper diagonal

u: [a, c, 0] ← O(1)*

v: [b, 0] ← O(1)*

w: [d] ←

z: [] ←

K: []

n-1

O(1)*

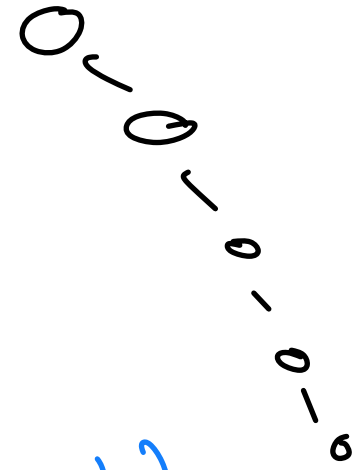
O(n) if so

What are my Big Os?

Graph Implementation: Adjacency Matrix

Pros:

↳ Very efficient edge lookup / modification

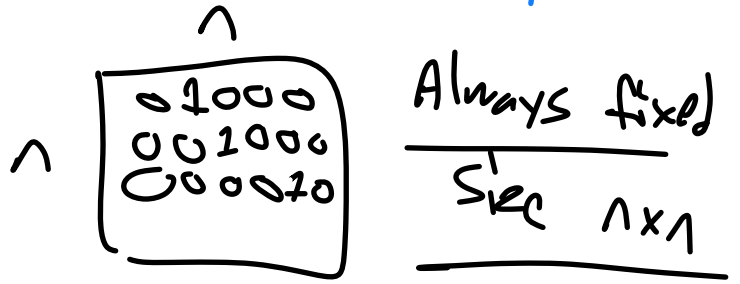


Cons:

↳ Worst memory cost by far (especially sparse graph)

↳ Adding vertices for resizing is bad)

↳ Looking up all adjacent is still O(n)

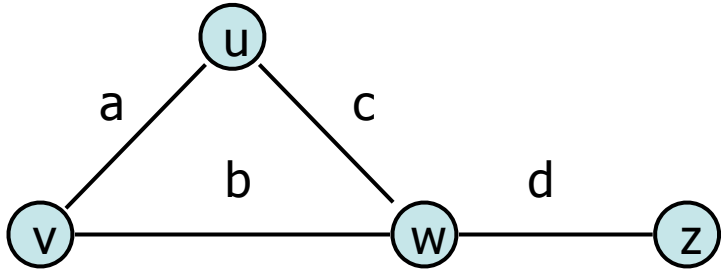


Dense graphs which do not change vertices == adjacency matrix

Graph Implementation Brainstorming

We want something…

Faster than an edge list

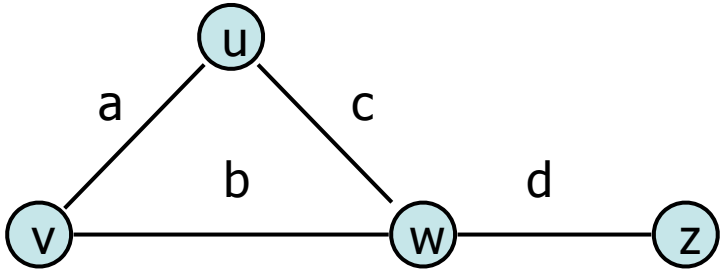


Less space than an adjacency matrix

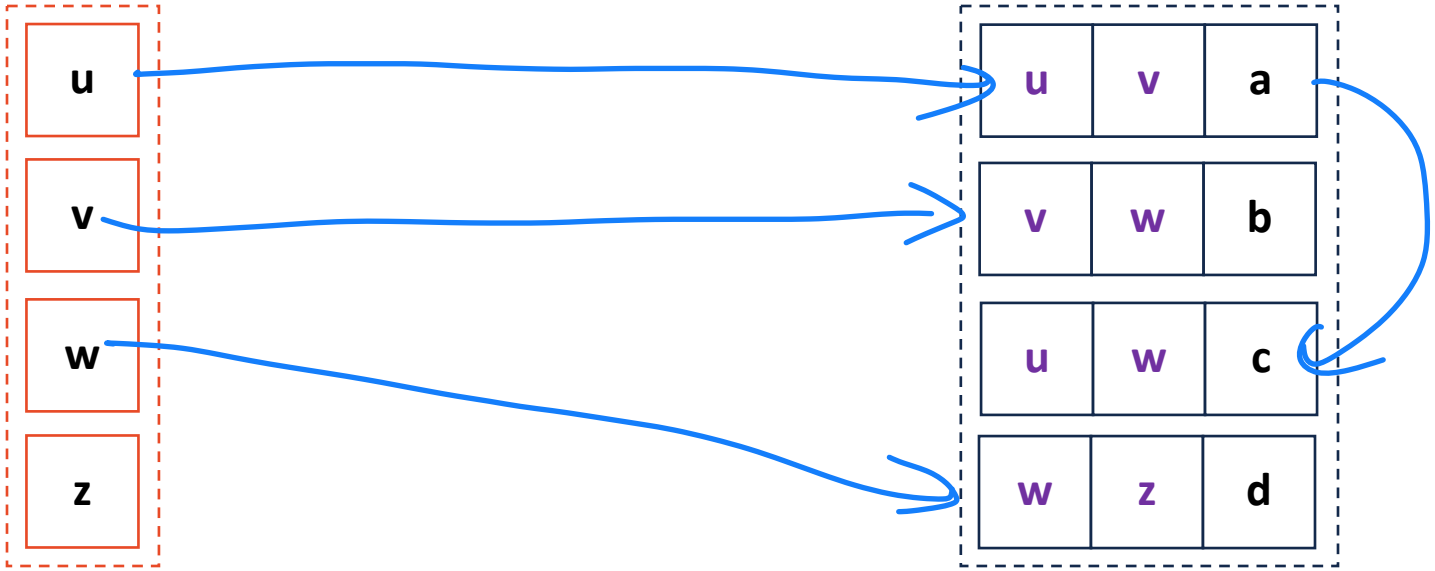
Particularly good at **finding all adjacent elements (neighbors)**

Graph Implementation: Edge List + ?

|V|= n, |E|= m



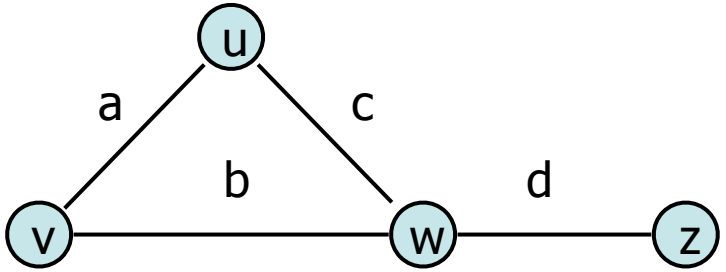
Bring back pointers & linked list

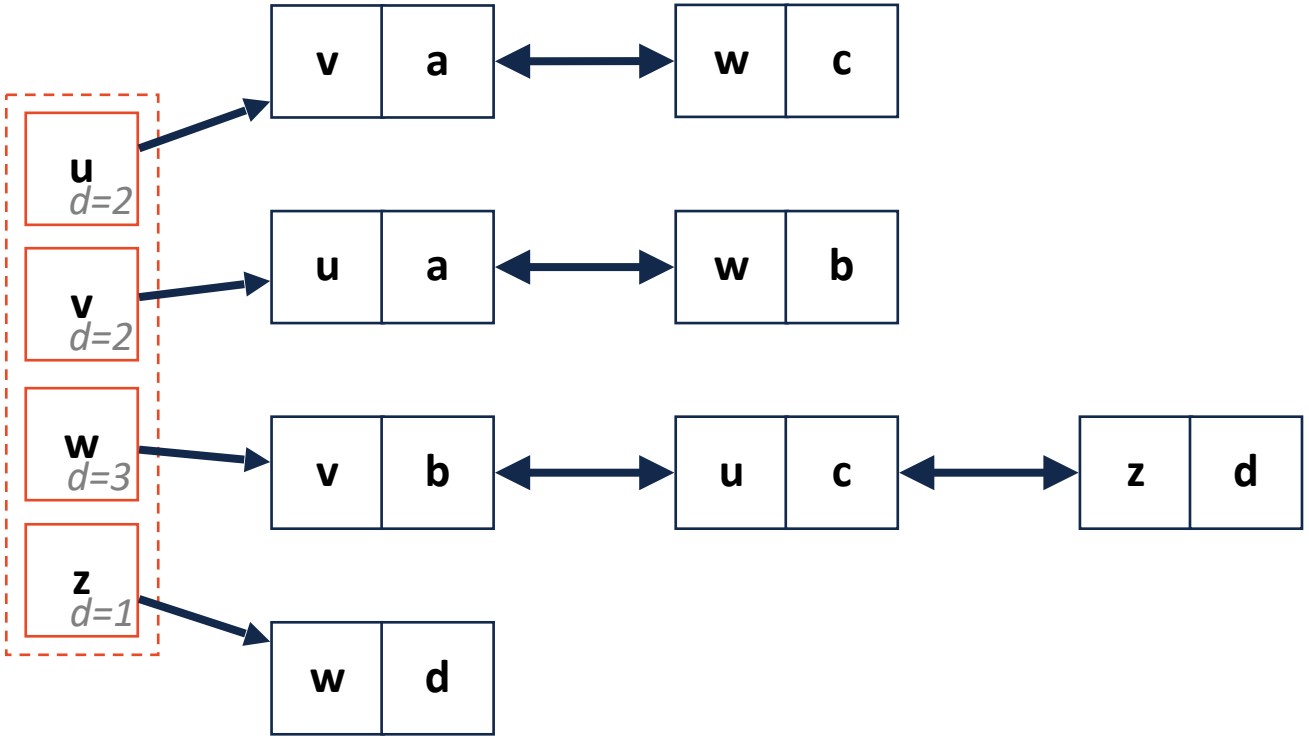


(1)

Graph Implementation: Adjacency List

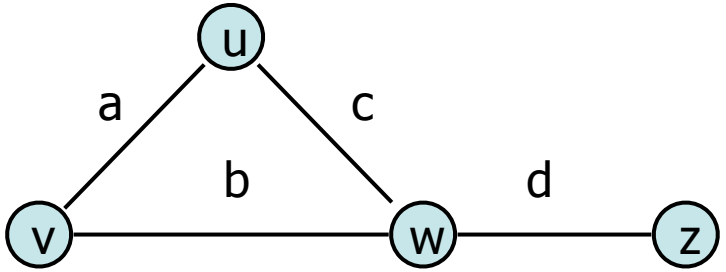
|V|= n, |E|= m



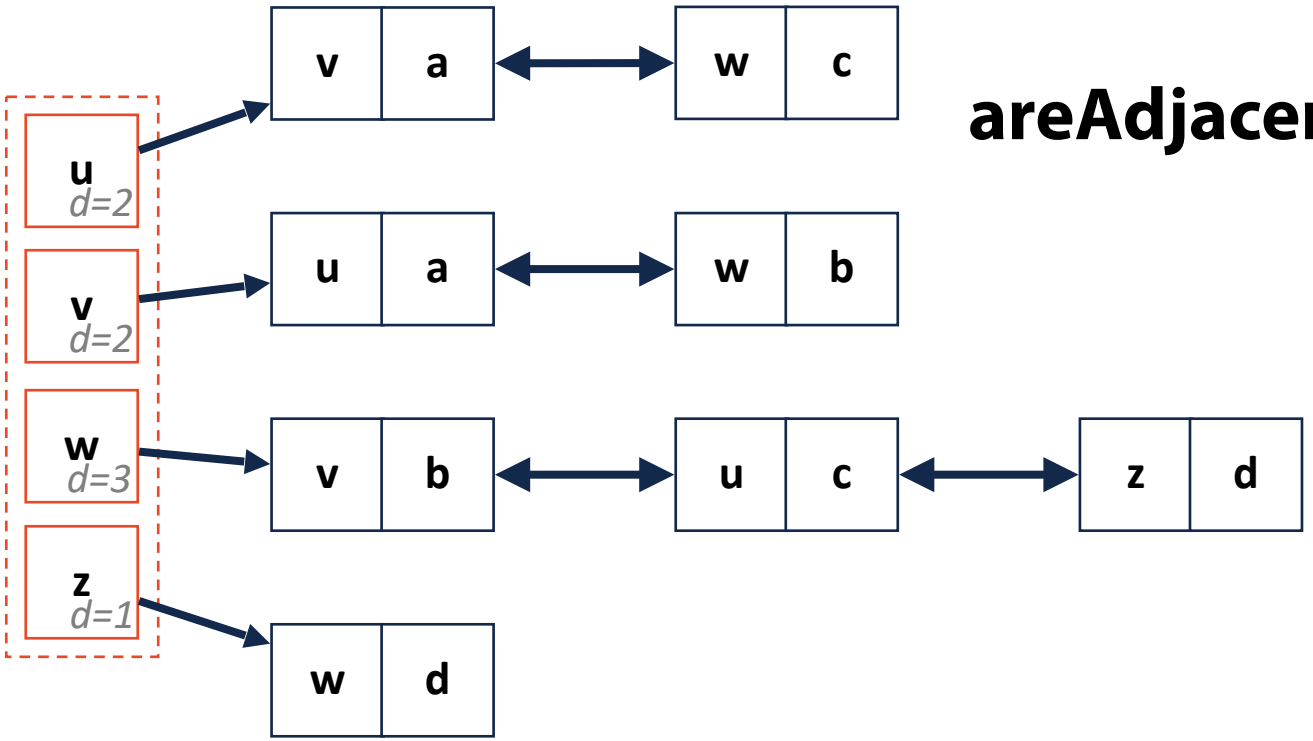


Graph Implementation: Adjacency List

|V|= n, |E|= m



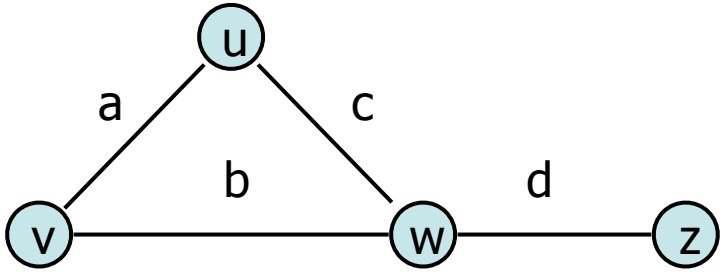
incidentEdges(Vertex v):



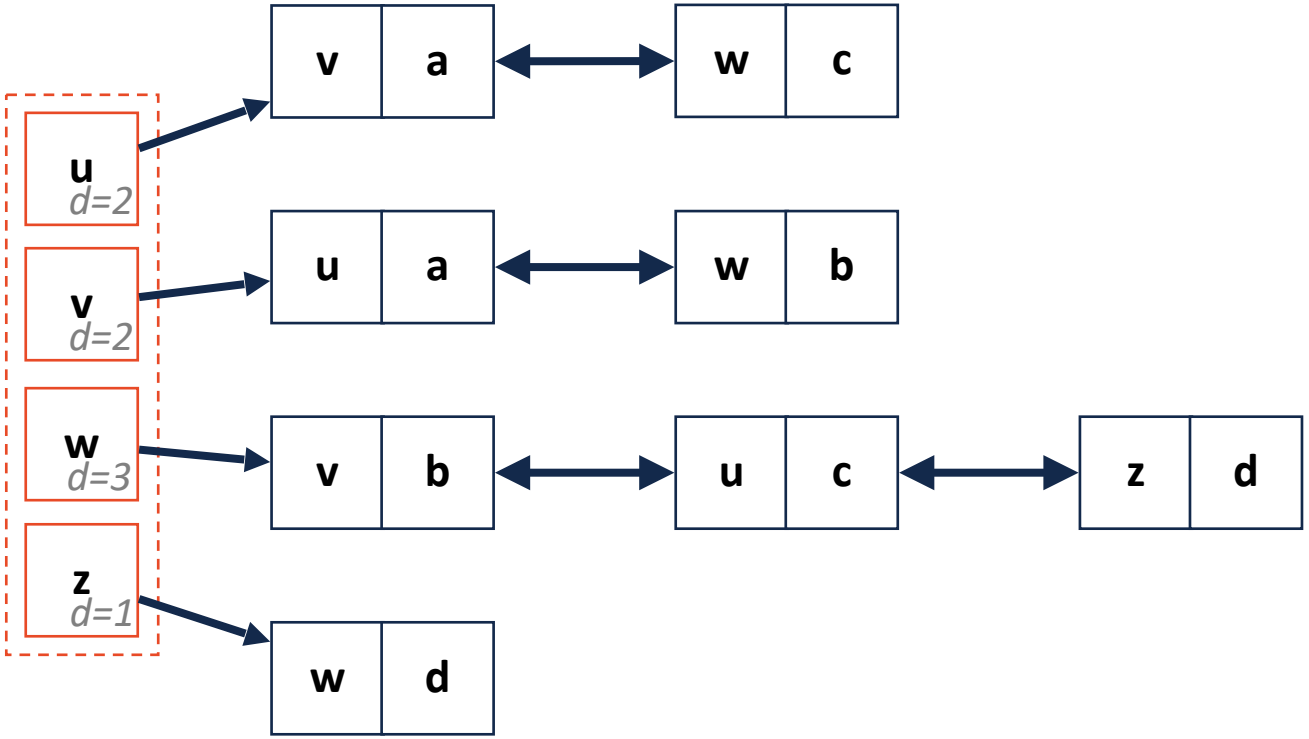
areAdjacent(Vertex v1, Vertex v2):

Graph Implementation: Adjacency List

|V|= n, |E|= m



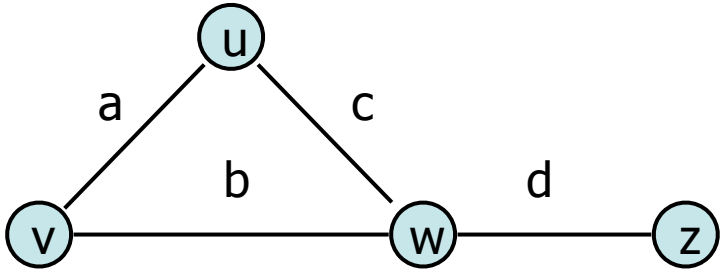
removeEdge(Vertex v1, Vertex v2, K key):

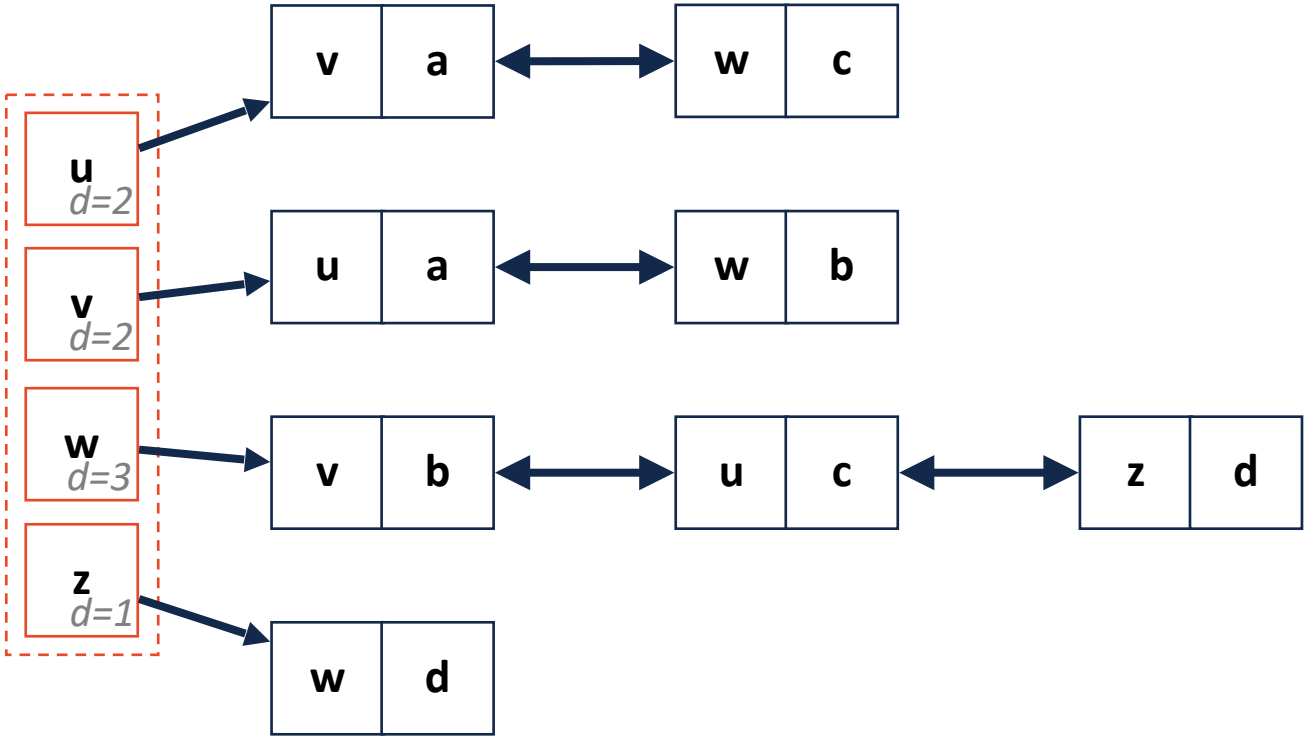


Graph Implementation: Adjacency List

|V|= n, |E|= m

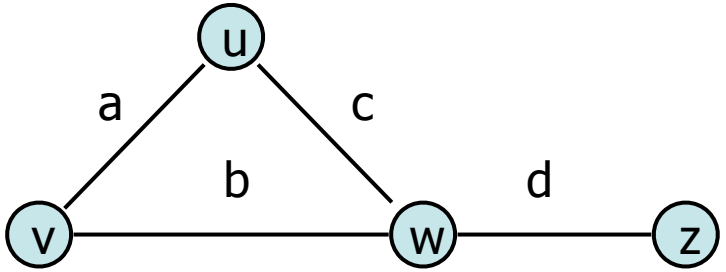
removeVertex(Vertex v):





Graph Implementation: Adjacency List

|V|= n, |E|= m



What's wrong with our implementation?

How can we fix it?