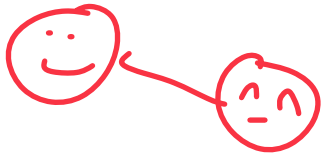
Data Structures

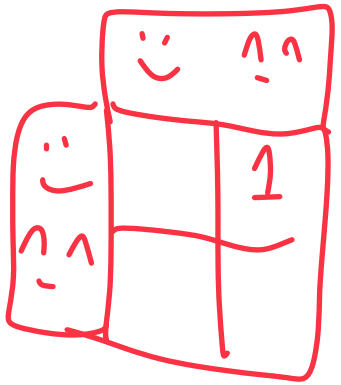
Graph Implementations



CS 225

November 13, 2023

Brad Solomon & G Carl Evans



UNIVERSITY OF
ILLINOIS
URBANA-CHAMPAIGN

Department of Computer Science

MP_Sketching Releases Today

A brand new assignment

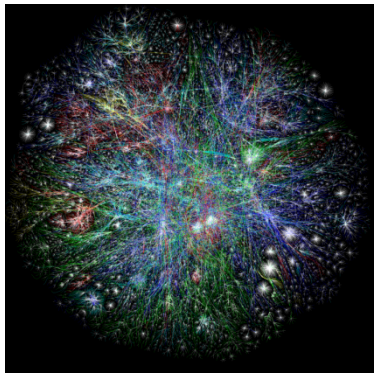
If instructions are unclear ask*!

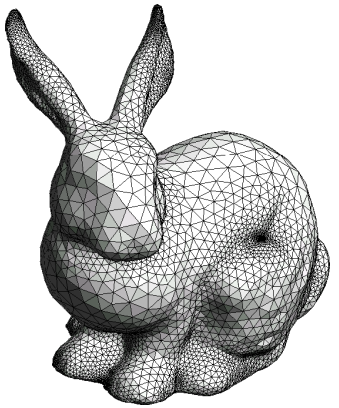
* Most staff probably won't know this assignment until Wednesday

Learning Objectives

Discuss graph implementation and storage strategies

Graphs





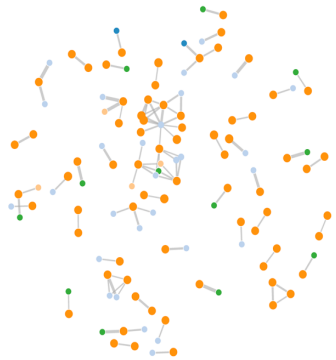
To study all of these structures:

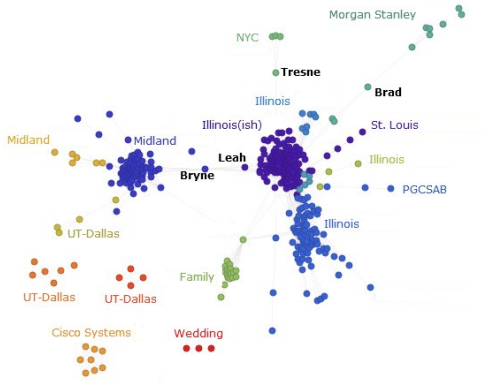
1. A common vocabulary
2. Graph implementations
3. Graph traversals
4. Graph algorithms

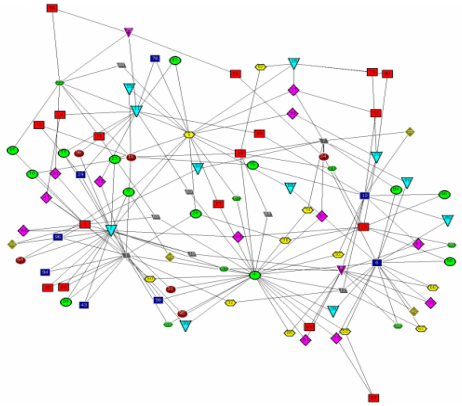
Find

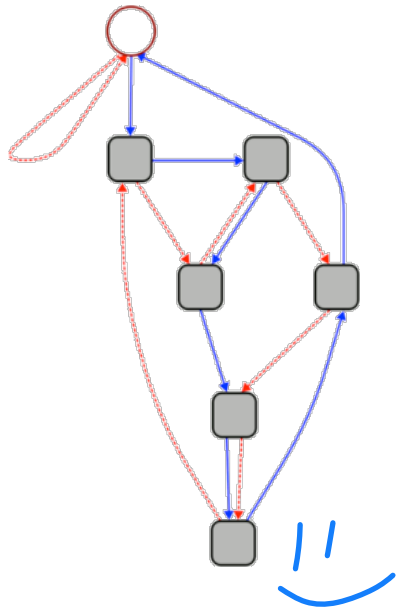
HAMLET

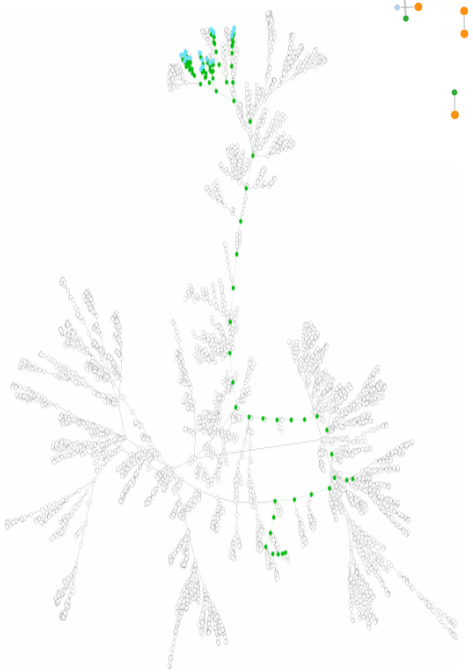
TROILUS AND CRESSIDA







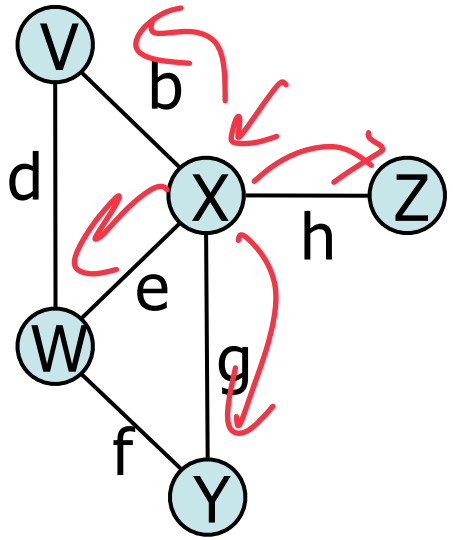




Graph ADT

Data:

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



Functions:

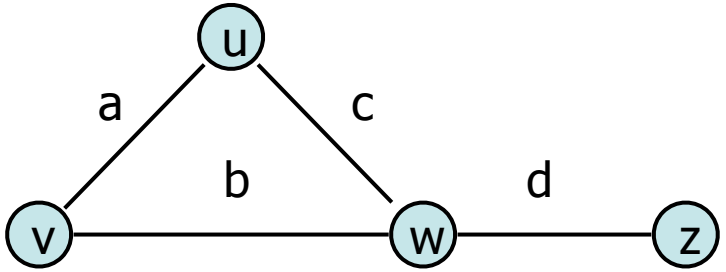
- insertVertex(K key);
- insertEdge(Vertex v1, Vertex v2, K key);

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

- incidentEdges(Vertex v);
- areAdjacent(Vertex v1, Vertex v2);
- origin(Edge e);
- destination(Edge e);

Graph Implementation Ideas → ADT

Insert / Remove / Access

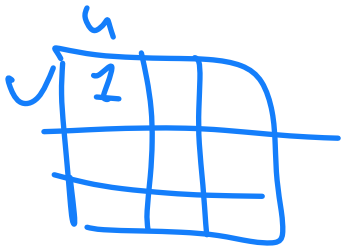


Vertices

Map [key=vertex, → adj list]

Edges

2D Matrix



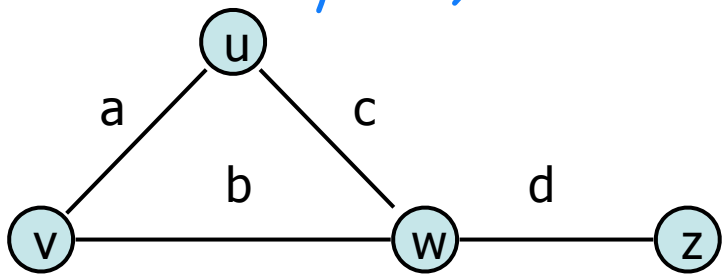
array

Linked list

Graph Implementation: Edge List

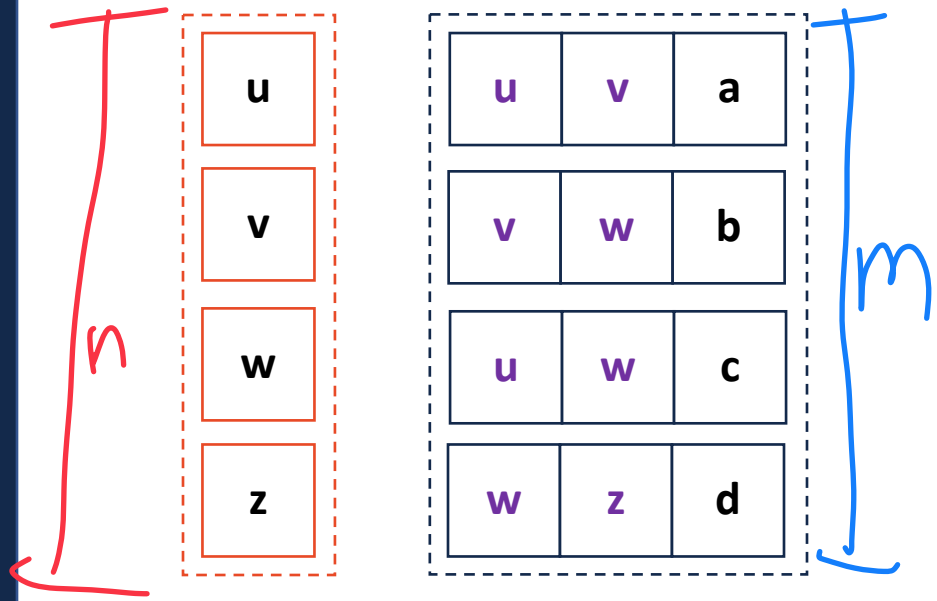
$|V| = n, |E| = m$

$G = (V, E)$



Vertex Storage: List

↳ 1D array list



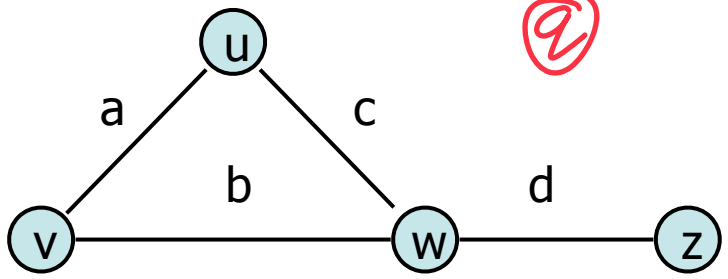
Edge Storage: List

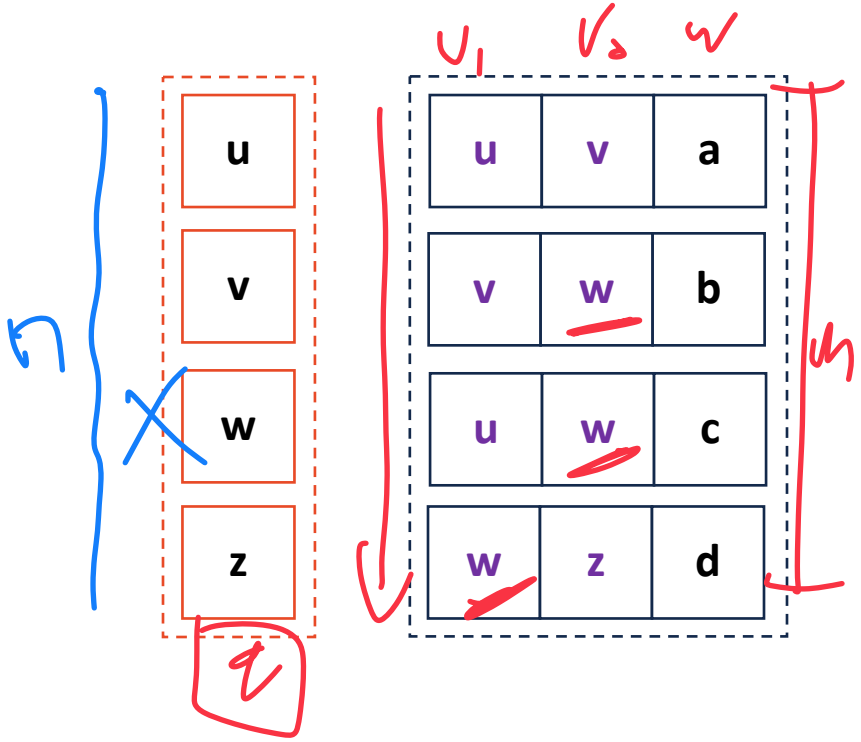
↳ 1D array list

Seems bad - but used a lot.

↳ Think about why?

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





insertVertex(K key):

↳ Array list insert $O(1)^*$

↳ Linked list $O(1)$

removeVertex(Vertex v):

↳ Array list remove(w) $O(n)$

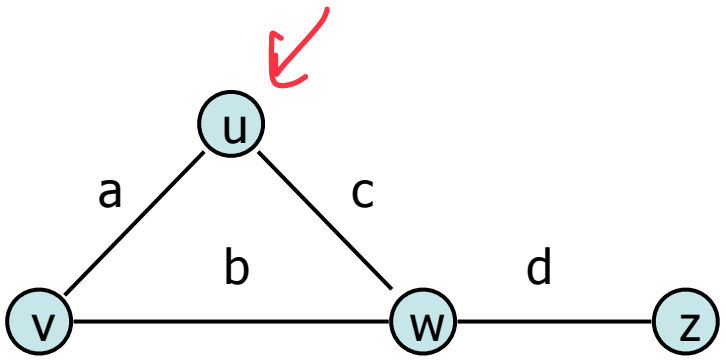
1) Remove from vertex list

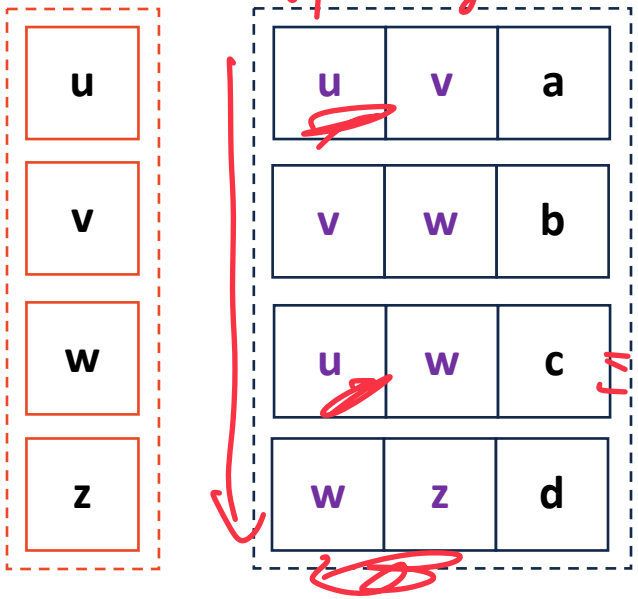
2) Remove from edge list $\rightarrow O(m)$

↳ check for w in v_1 or v_2

$O(n+m)$

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





incidentEdges(Vertex v):

Look through edge list for v in either v_1 or v_2

$O(m)$

areAdjacent(Vertex v1, Vertex v2):

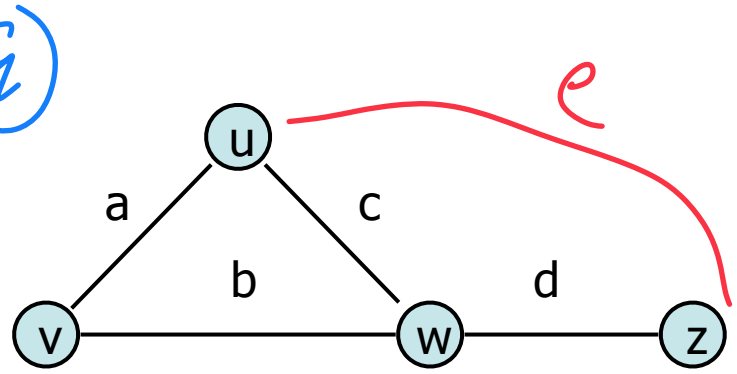
Look through list for (v_1, v_2) (v_2, v_1)

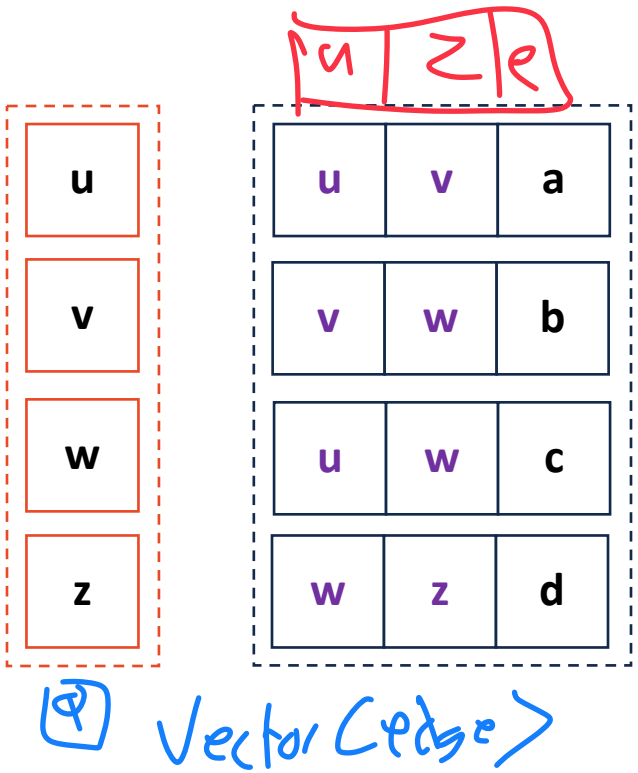
$= w \cap c$

↳ return True

$O(m)$

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





Vector<Edge>

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

1) Look if edge exists already — O(n)
   - If simple graph (no multiedges)
   - Replace?
2) Insert as array insert — O(1)*

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

Walk through edge
find pair

→ O(m)

Graph Implementation: Edge List

Pros:

- using minimal storage
- Adding edges is nice? O(m) use O(1) / Adding vertices is nice O(1)*
- Simple data structure

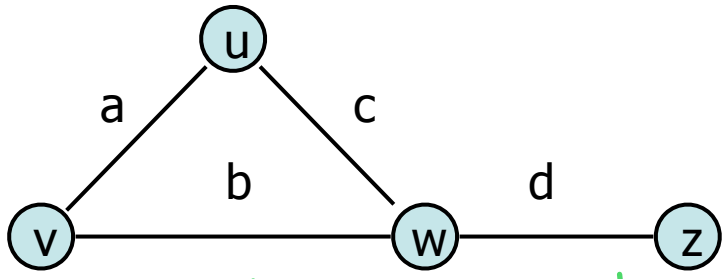
Cons:

- very hard to see graph structure
- Checking for specific edge is O(m)
- Most ops are slow

Improve graph by....

get Adjacent() fast!

Graph Implementation: Adjacency Matrix



Vertex list

Edge List

Big overlap!

Adjacency Matrix

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

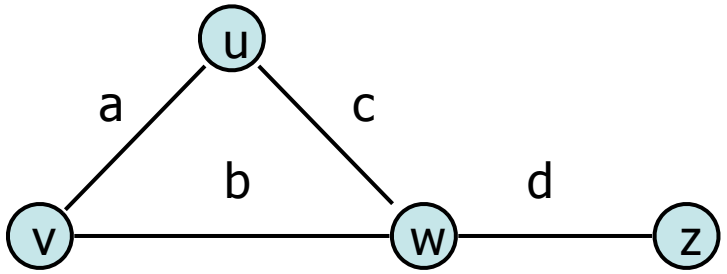
	u	v	w	z
u	F	T	T	F
v		F	T	F
w			F	T
z				F

edge list + something new

Graph Implementation: Adjacency Matrix

2D vector <int, int>

m[x][y] = w



map(pair<int, int>) = weight

map(str, str) = weight

Vertex → Hash table!

u	0
v	1
w	2
z	3

	0	1	2	3
0		9		
1				
2			w	X
3				

u | v | w/z

u

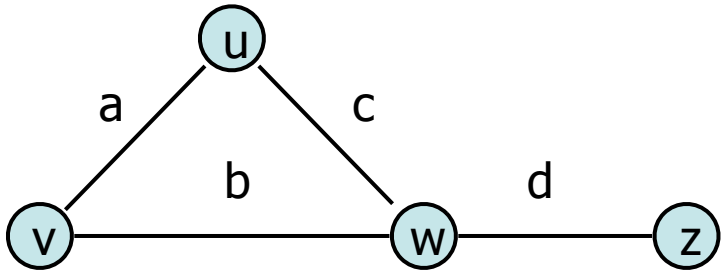
v

w

z

Graph Implementation: Adjacency Matrix

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



Vertex Storage: Hash table Key = vertex, Value = index into matrix

Unordered_map

u	0
v	1
w	2
z	3

Edge Storage: 2D Vector (vector<vector<int>>)

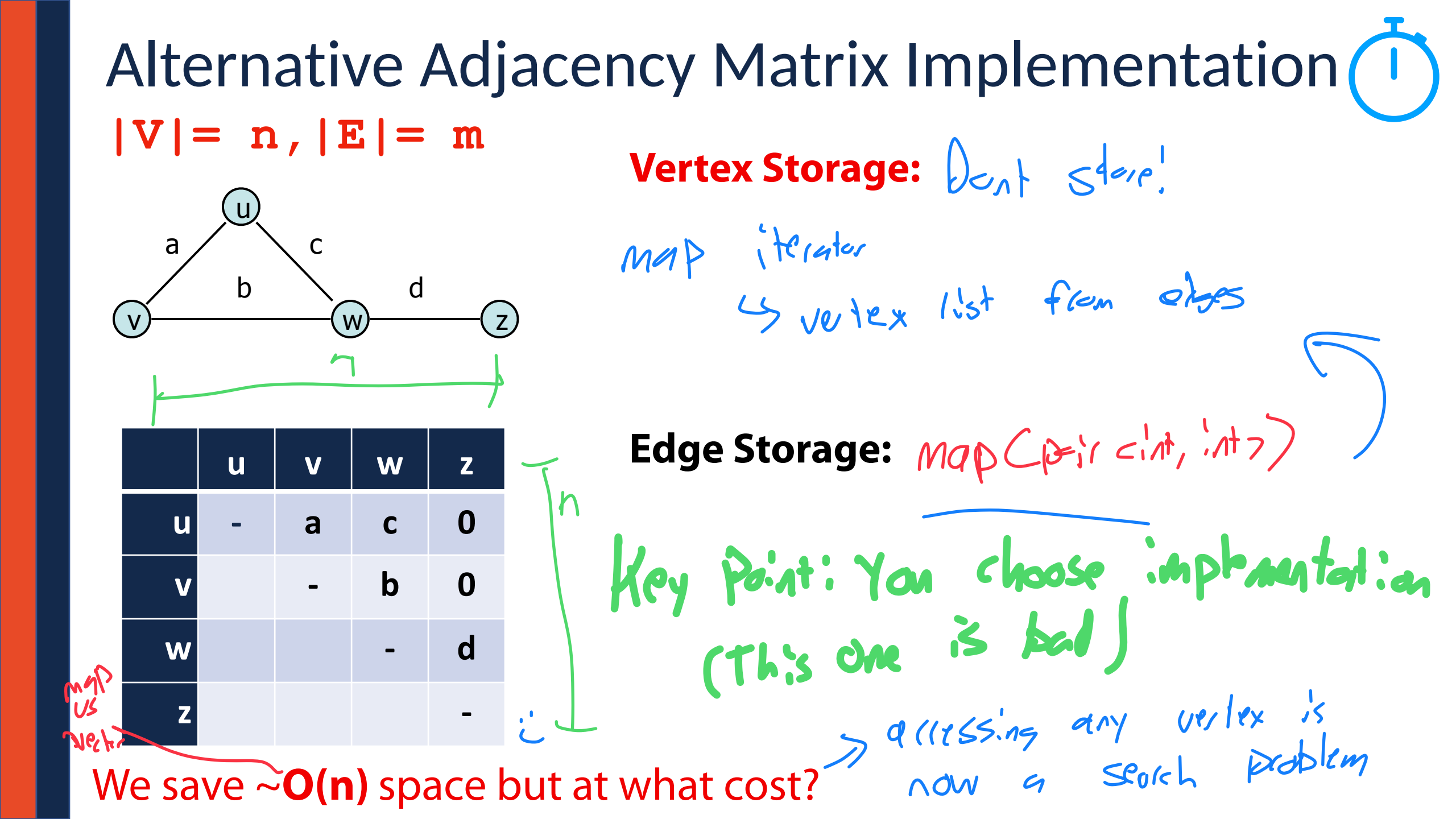
	0	1	2	3
0	-	a	c	0
1		-	b	0
2			-	d
3				-

No self edges

We will use this

Unfilled means its a mirror

Alternative Adjacency Matrix Implementation



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

	u	v	w	z
u	-	a	c	0
v		-	b	0
w			-	d
z				-

n

n

Vertex Storage: Don't store!

map iterator
↳ vertex list from edges

Edge Storage: map<pair<int, int>>

Key Point: You choose implementation
(This one is bad)

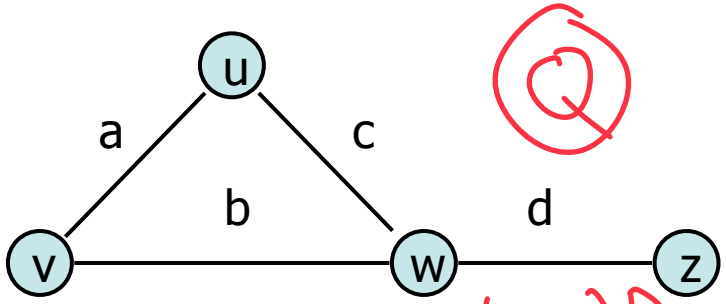
map us vert

We save ~**O(n)** space but at what cost?

accessing any vertex is now a search problem

Graph Implementation: Adjacency Matrix

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



H.T.

u	0
v	1
w	2
z	3

vertex info

Q | 4

vector 2D

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

O(n)

Q

insertVertex(K key):

map[Q] = 4

↳ Hash table

1) Add to vertex table/map/vector O(1)

2) Also add to matrix

vector 2D → O(n²) resizes O(n)

O(n)

Map → add only n items

removeVertex(Vertex v):

1) Remove vertex from H.T. O(1)

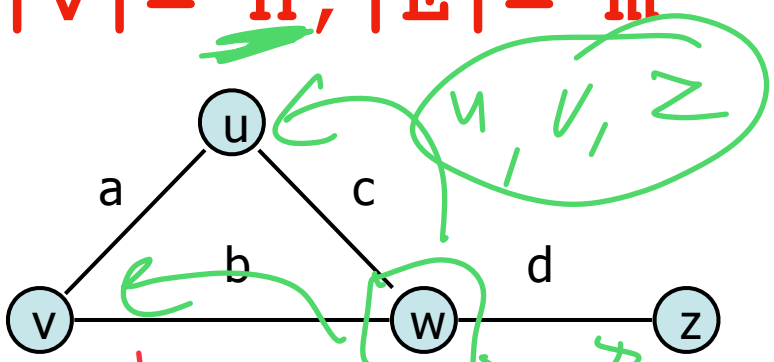
2) Remove row & column O(n)

vector 2D → tombstone

map → O(n)

Graph Implementation: Adjacency Matrix

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



incidentEdges(Vertex v):

1) Lookup vertex index — O(1)

2) Look at row/column in matrix

↳ O(n) + O(n) ≈ O(n)

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

u	0
v	1
w	2
z	3

1) If directed

why not return full row & column?

c b d 0 0 0 0

Look up Z

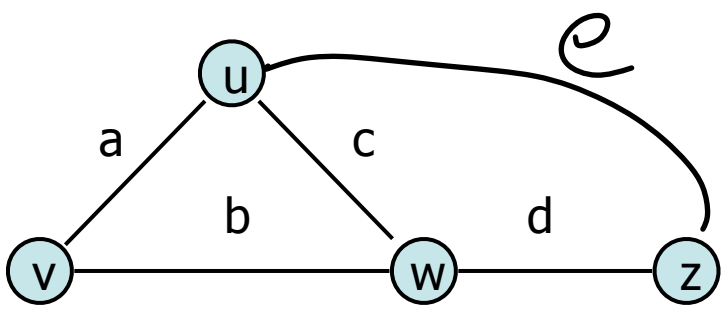
areAdjacent(Vertex v1, Vertex v2):

2 index values $v_1 \to int$, $v_2 \to int$

$M[v_1, v_2]$ — O(1)

Graph Implementation: Adjacency Matrix

|V|= n, |E|= m



u	0
v	1
w	2
z	3

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

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

↳ $M[v_1, v_2] = k$

:) $O(1)$

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

↳ $M[v_1, v_2] = 0$

:) $O(1)$

Graph Implementation: Adjacency Matrix

Pros:

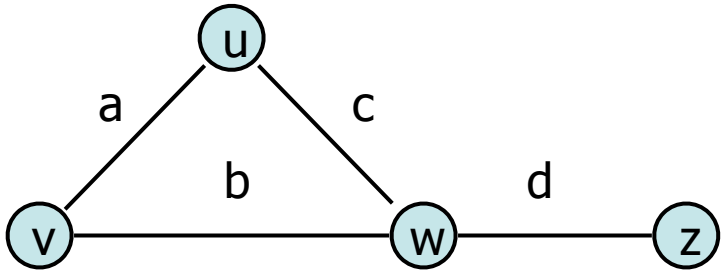
- very good edge lookup / modification

Note: only average at "get all neighbors" O(n)

Cons:

- Very big (large storage cost) O(n²)
- Adding / Removing vertices is slow

Graph Implementations



We want something…

Faster than an edge list

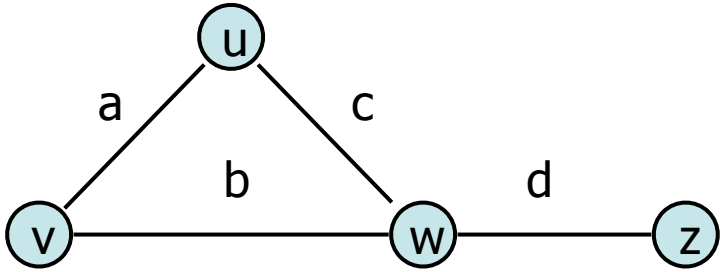
Less space than an adjacency matrix

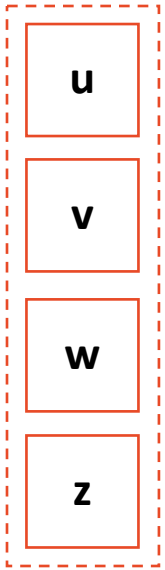
Particularly good at finding adjacent elements

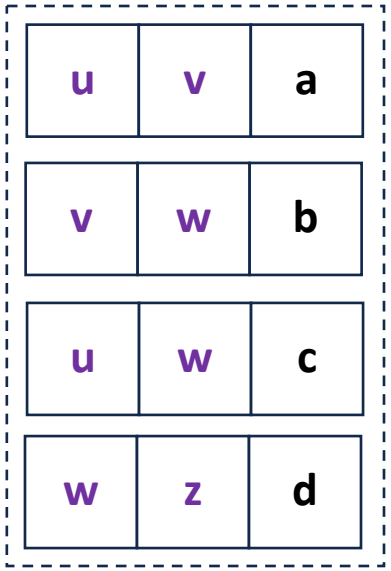
↳ why is neighbors not O(1)

Graph Implementation: Edge List + ?

|V|= n,|E|= m

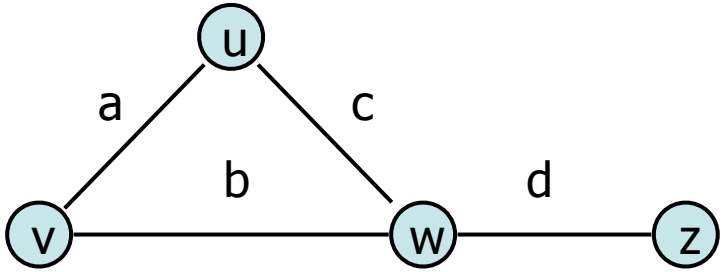


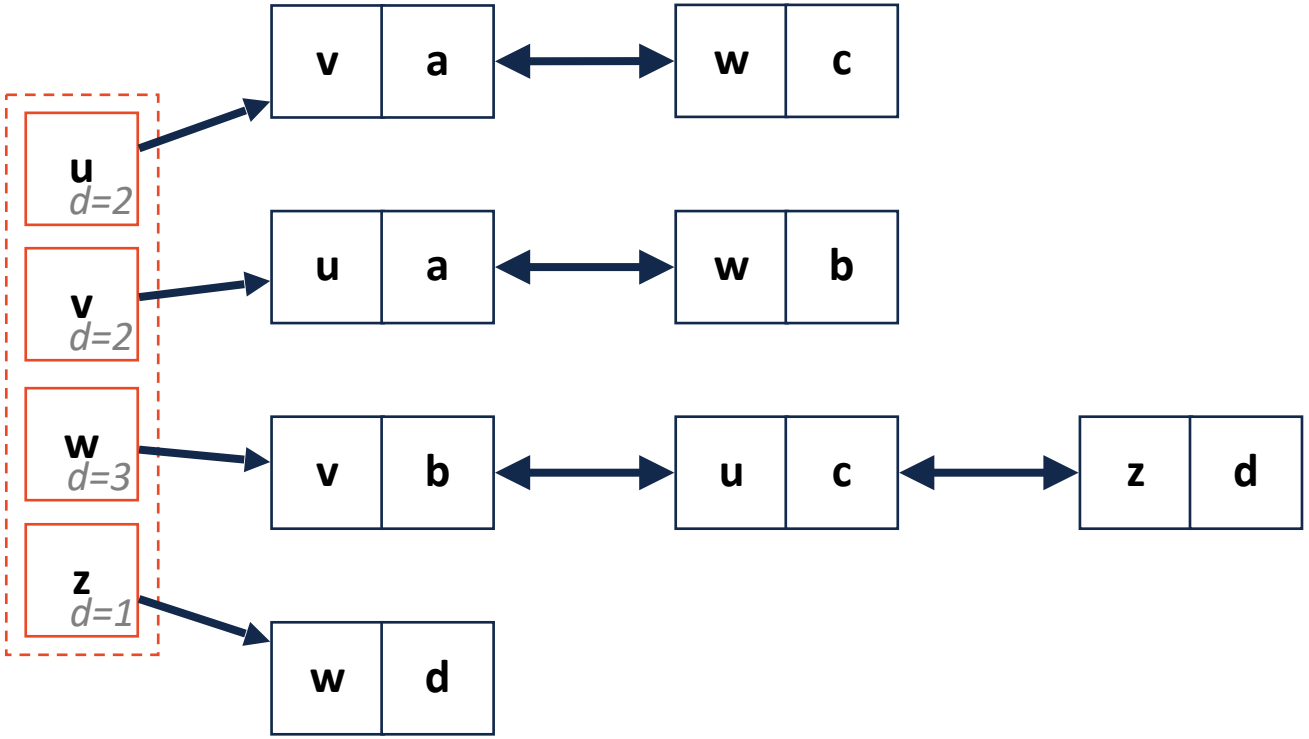




Graph Implementation: Adjacency List

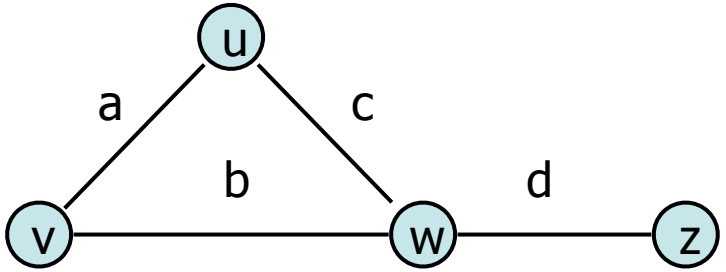
|V|= n, |E|= m



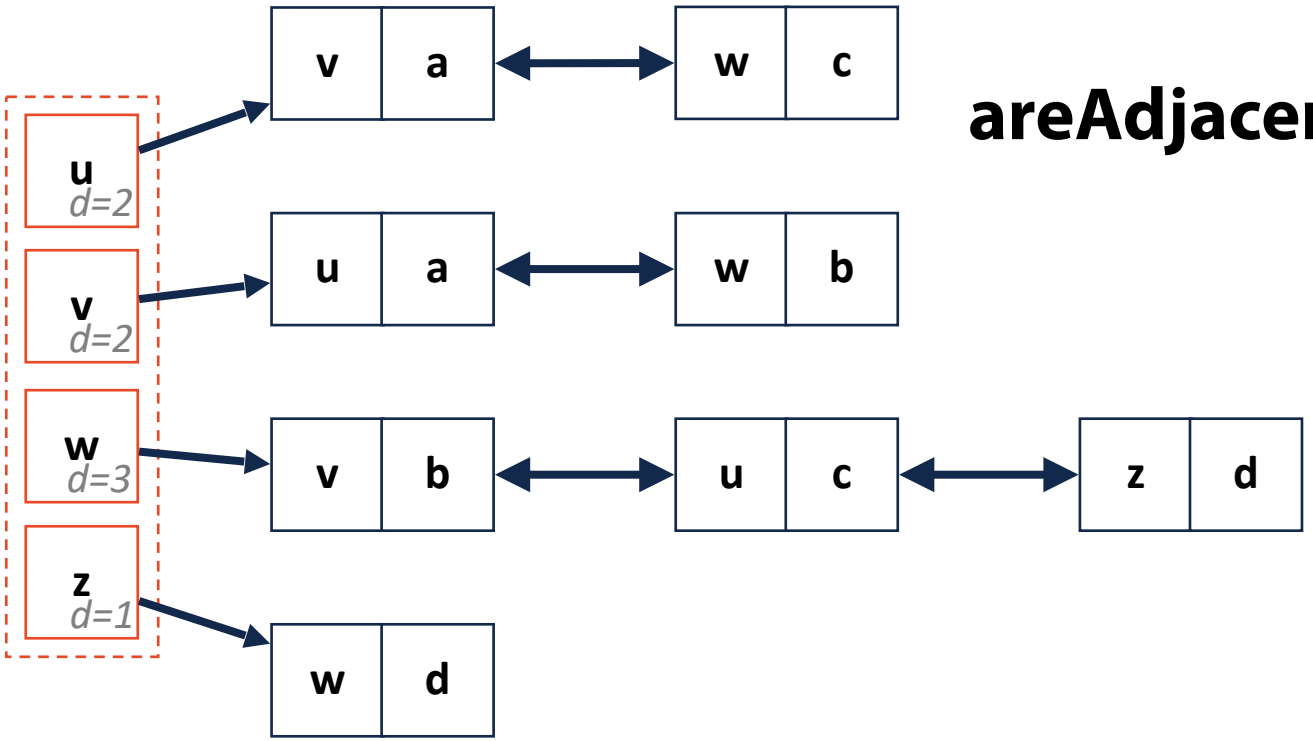


Graph Implementation: Adjacency List

|V|= n, |E|= m



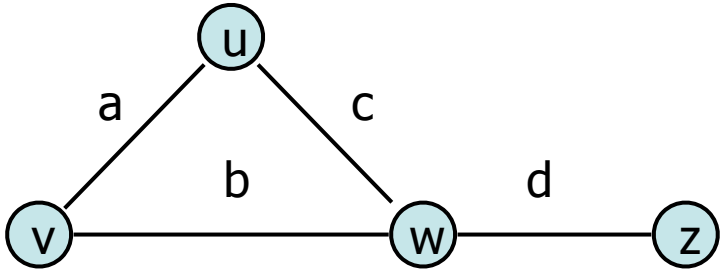
incidentEdges(Vertex v):



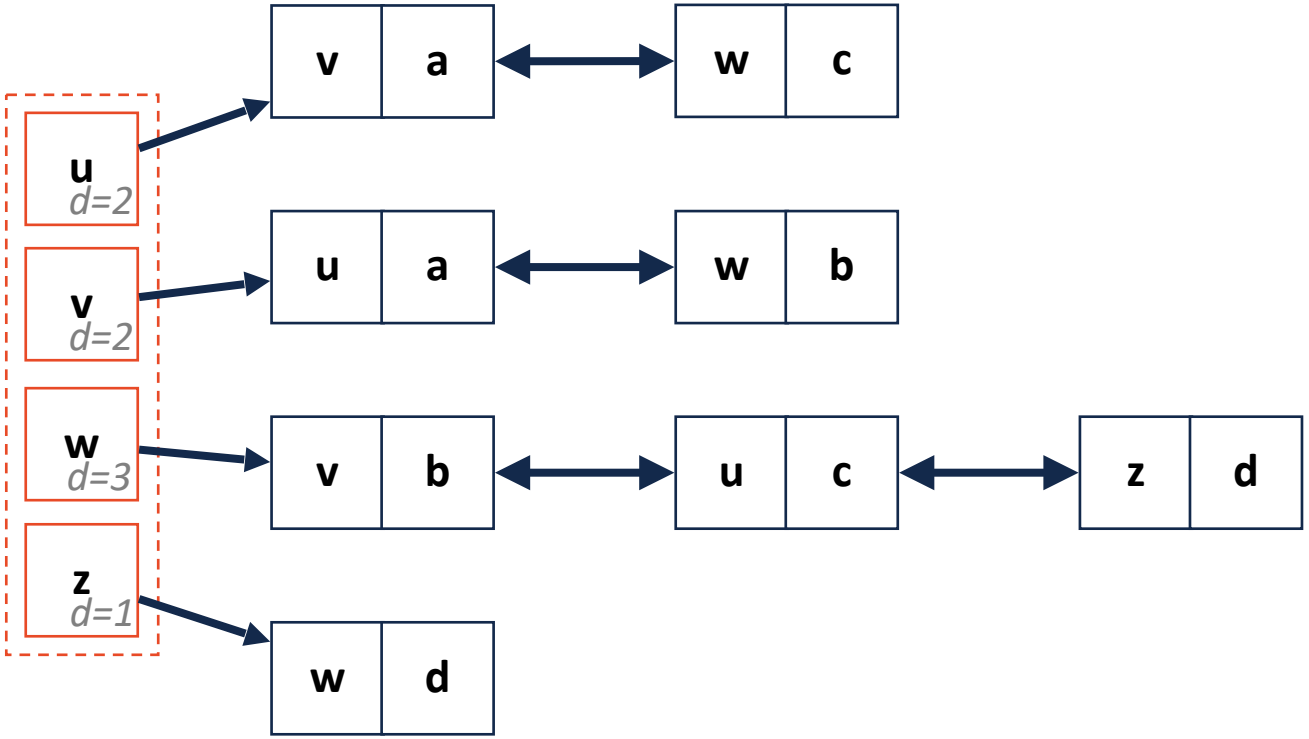
areAdjacent(Vertex v1, Vertex v2):

Graph Implementation: Adjacency List

|V|= n, |E|= m



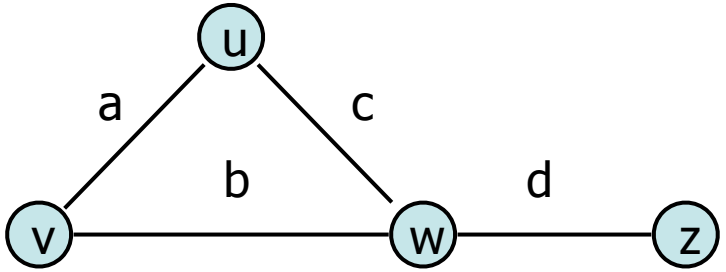
removeEdge(Vertex v1, Vertex v2, K key):

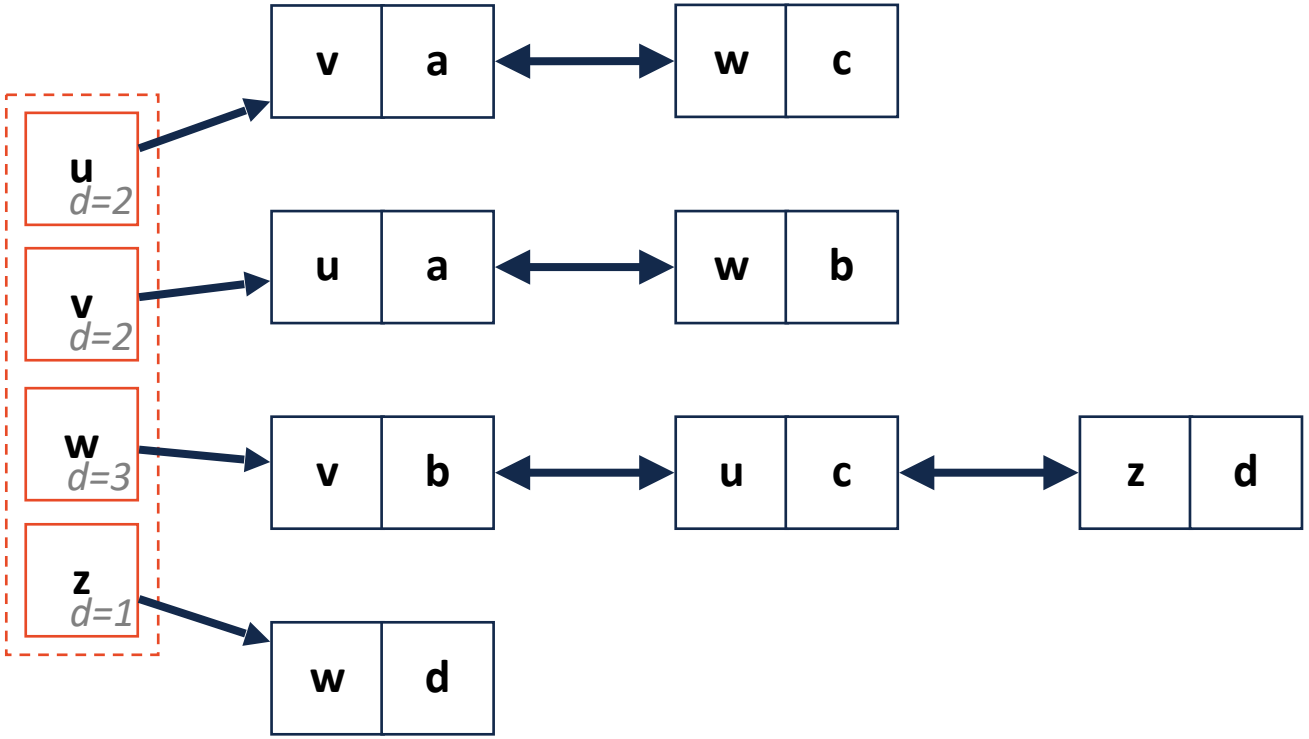


Graph Implementation: Adjacency List

|V|= n, |E|= m

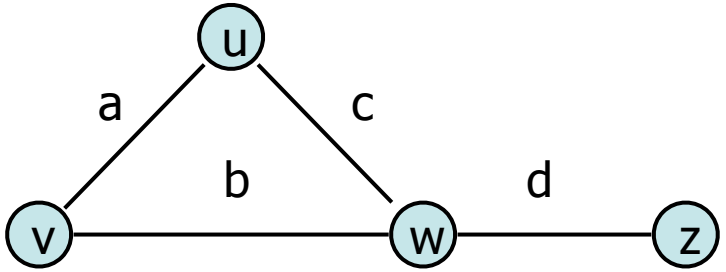
removeVertex(Vertex v):





Graph Implementation: Adjacency List

|V|= n, |E|= m



What's wrong with our implementation?

How can we fix it?