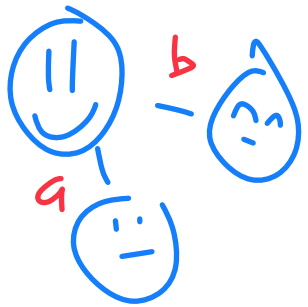
Data Structures

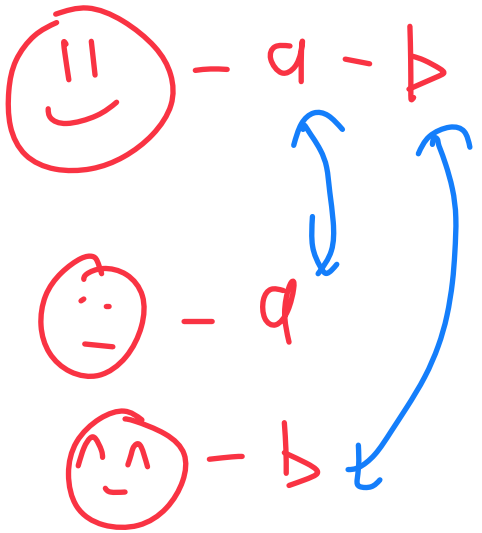
Graph Implementations 2



CS 225

November 15, 2023

Brad Solomon & G Carl Evans



UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Announcements

Exam 5 topic list (correctly) posted

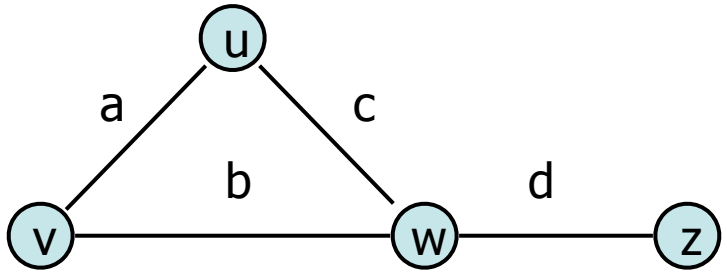
Practice exam released (additional questions will be added)

Extra Credit Project check-in deadline modified

Learning Objectives

- Discuss graph implementation and storage strategies
- Introduce graph traversals

Graph Implementation: Edge List

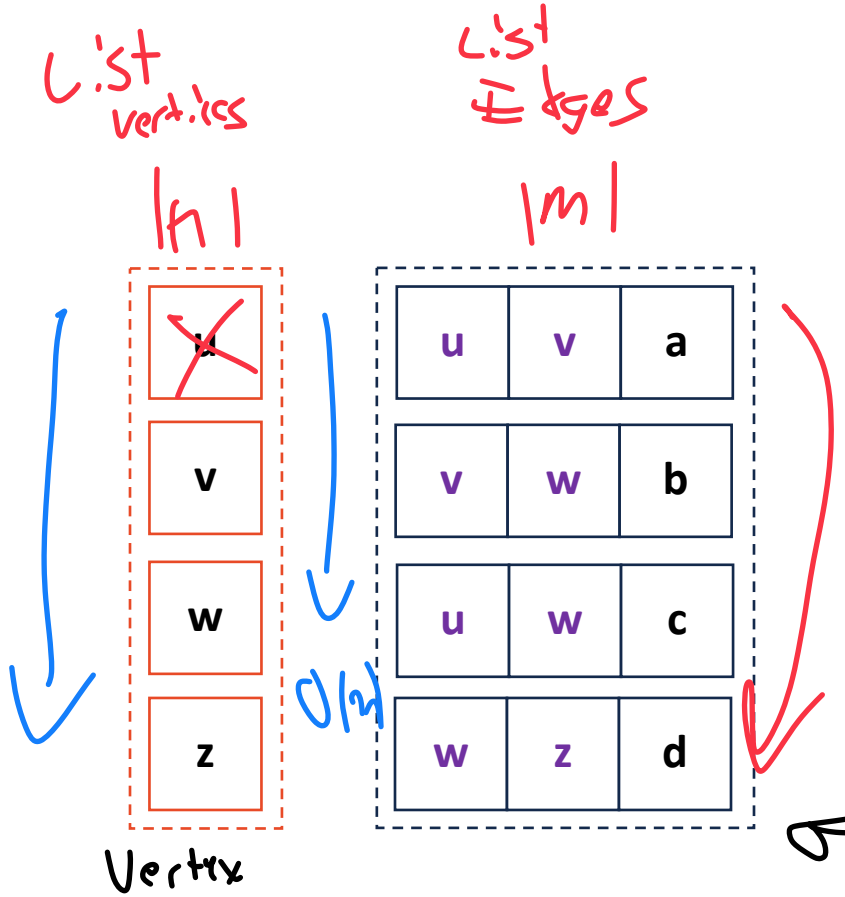


Big O

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

insertVertex(K key): $O(n)$* list

insertEdge(Vertex v1, Vertex v2, K key): $O(1)$* array almost LL



removeVertex(Vertex v): $O(n+m)$

removeEdge(Vertex v1, Vertex v2, K key): $O(m)$

↳ find & remove

find all edges

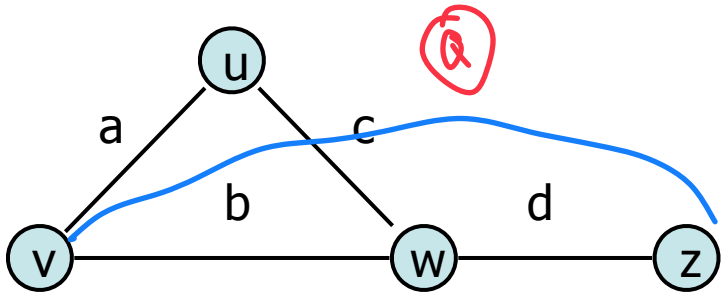
incidentEdges(Vertex v): $O(m)$

areAdjacent(Vertex v1, Vertex v2):

↳ $O(m)$

Graph Implementation: Adjacency Matrix

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



vertex list H.T.

Edges 2D matrix $O(n^2)$

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

insertVertex(K key): $O(n)$

insertEdge(Vertex v1, Vertex v2, ~~K key~~): $O(1)$ (weight)

removeVertex(Vertex v): $O(n)$

removeEdge(Vertex v1, Vertex v2, ~~K key~~): $O(1)$

incidentEdges(Vertex v): $O(n)$

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

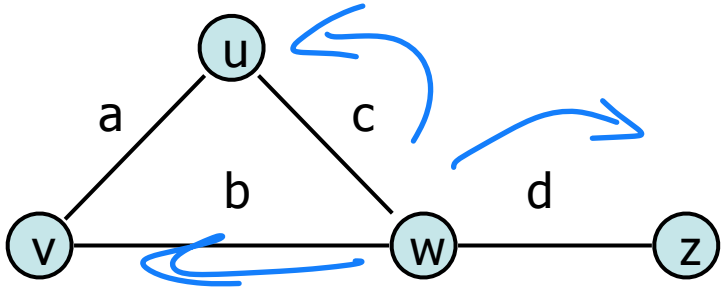
Great for dense graphs!

Graph Implementations

We want something…

Faster than an edge list

good for storage

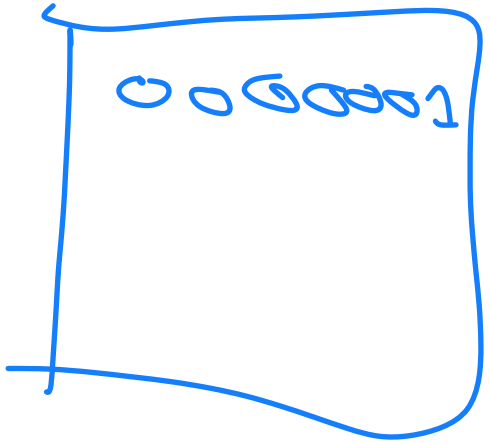


Less space than an adjacency matrix

good for dense graphs
→ direct edge queries

Particularly good at finding adjacent elements

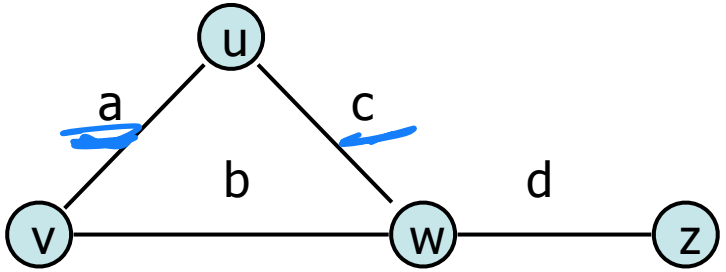
Adj List (vertex)

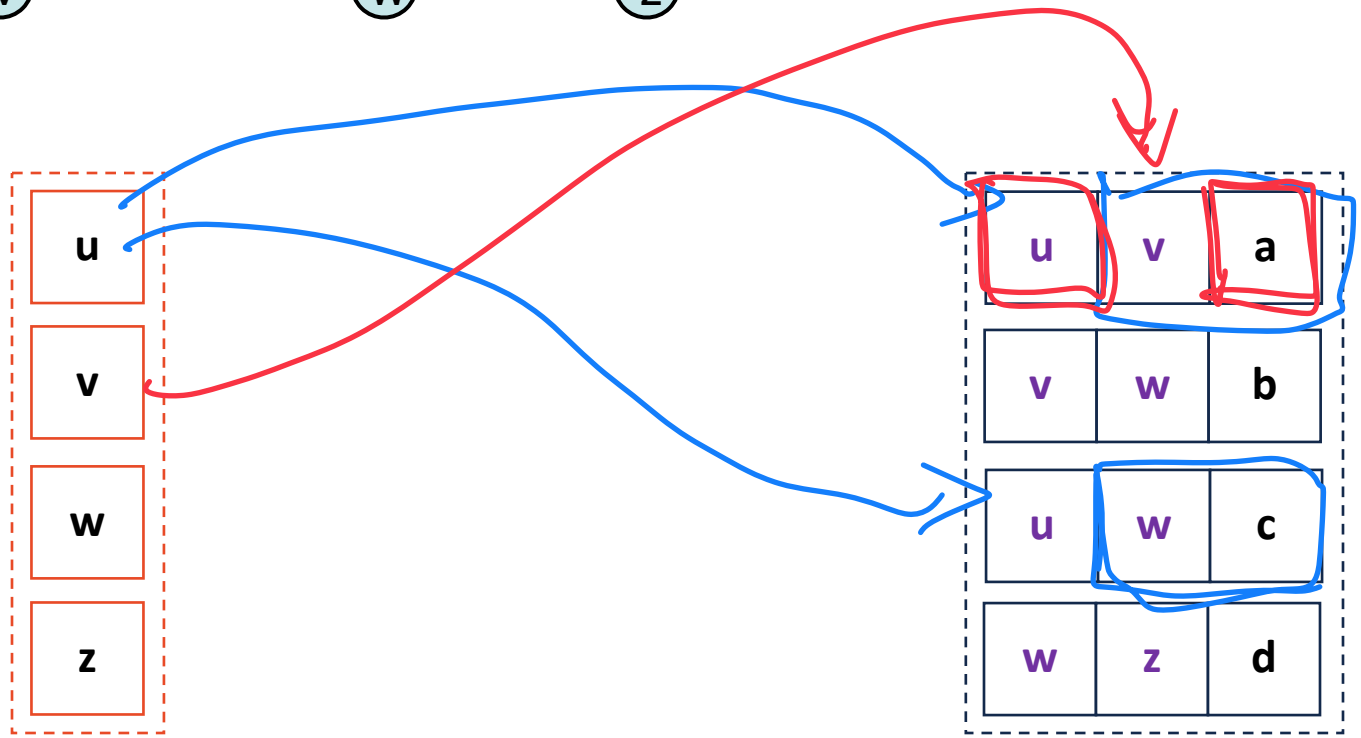


Graph Implementation: Edge List + ?

|V| = n, |E| = m

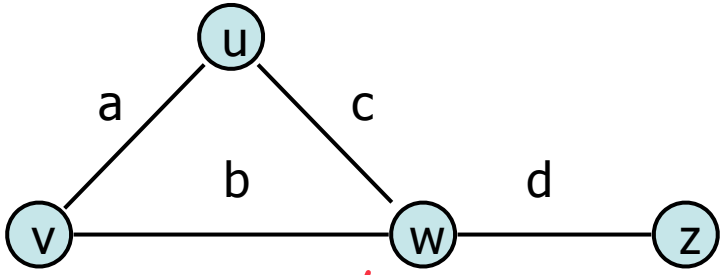
Each vertex to know its outgoing edges

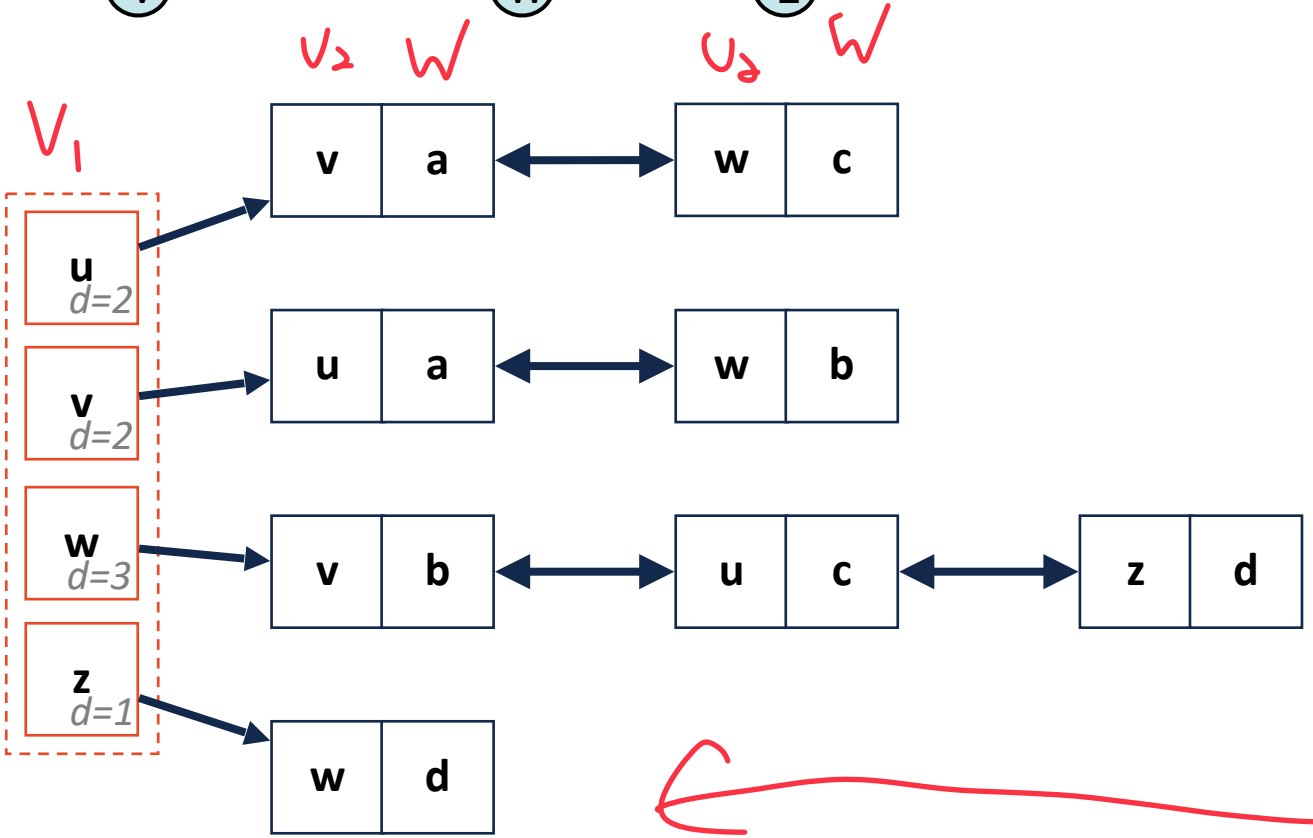




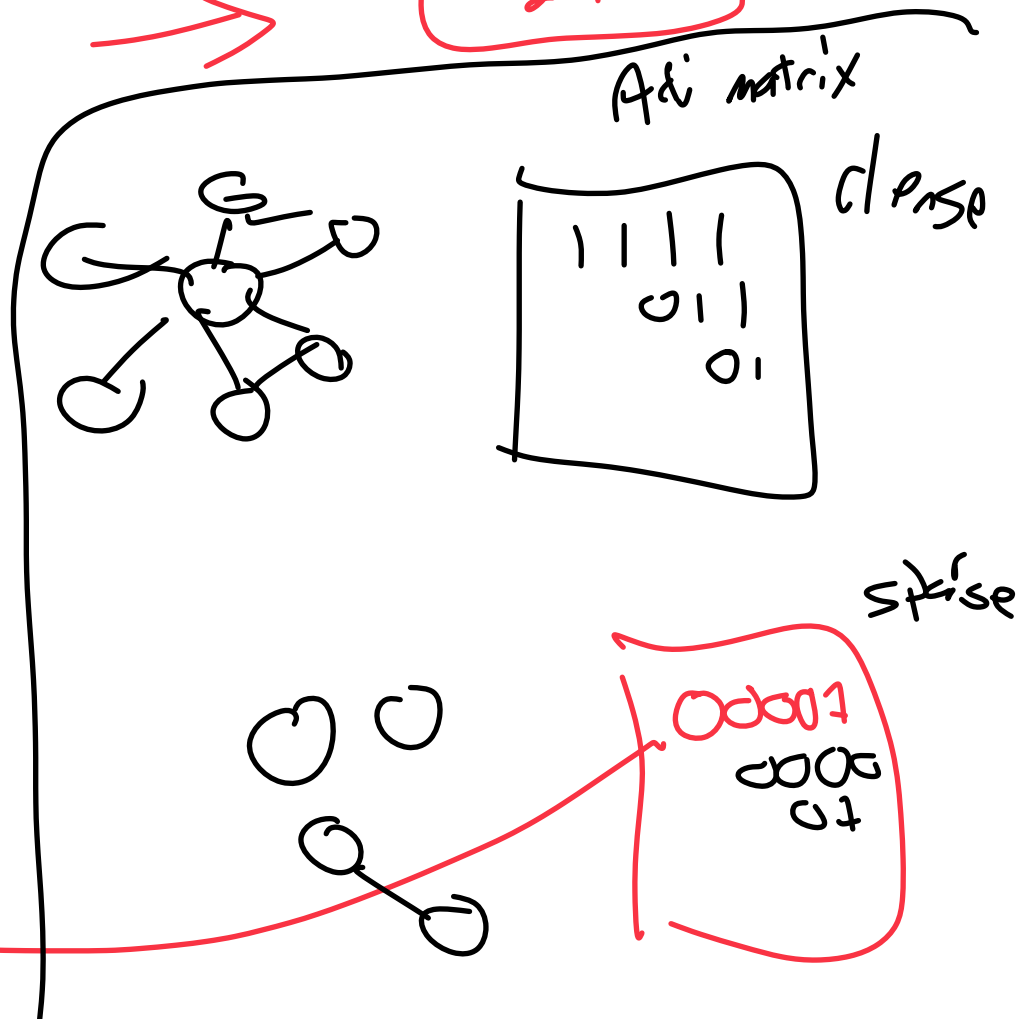
Simple Adjacency List

|V|= n, |E|= m



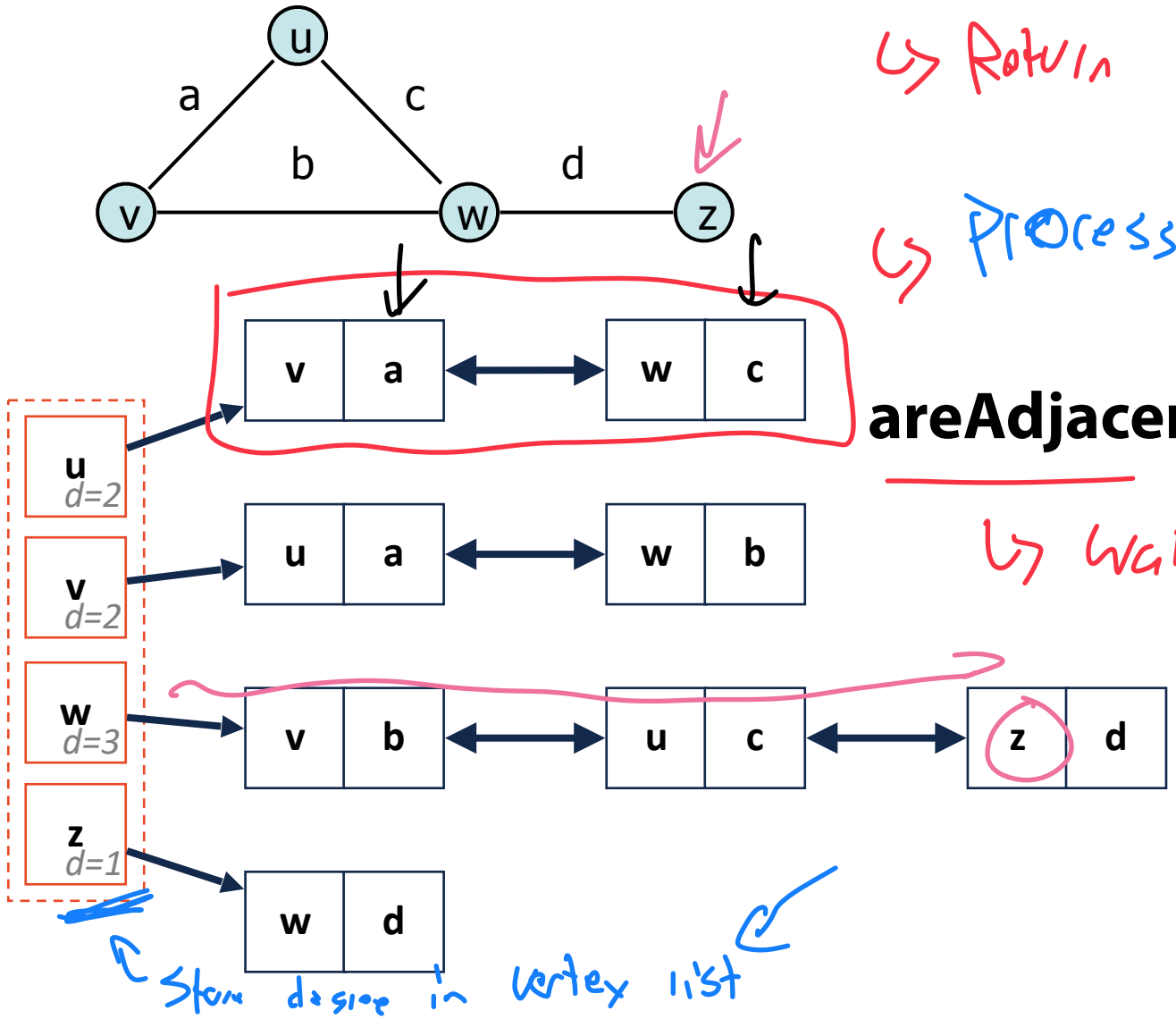


Vertex list H.T. → Linked list



Simple Adjacency List

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



Store degree in vertex list

incidentEdges(Vertex v):

↳ Return LL nodes $O(1)$

↳ Process → $[\deg(v)] \simeq O(n)$

Naive!

[a, c] edge weights

[(v,u), (v,w)]

n-1 edges

areAdjacent(Vertex v1, Vertex v2): (w, z)

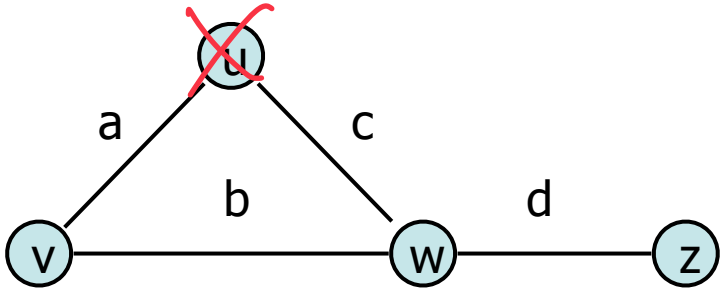
↳ walk through linked list w/ min degree

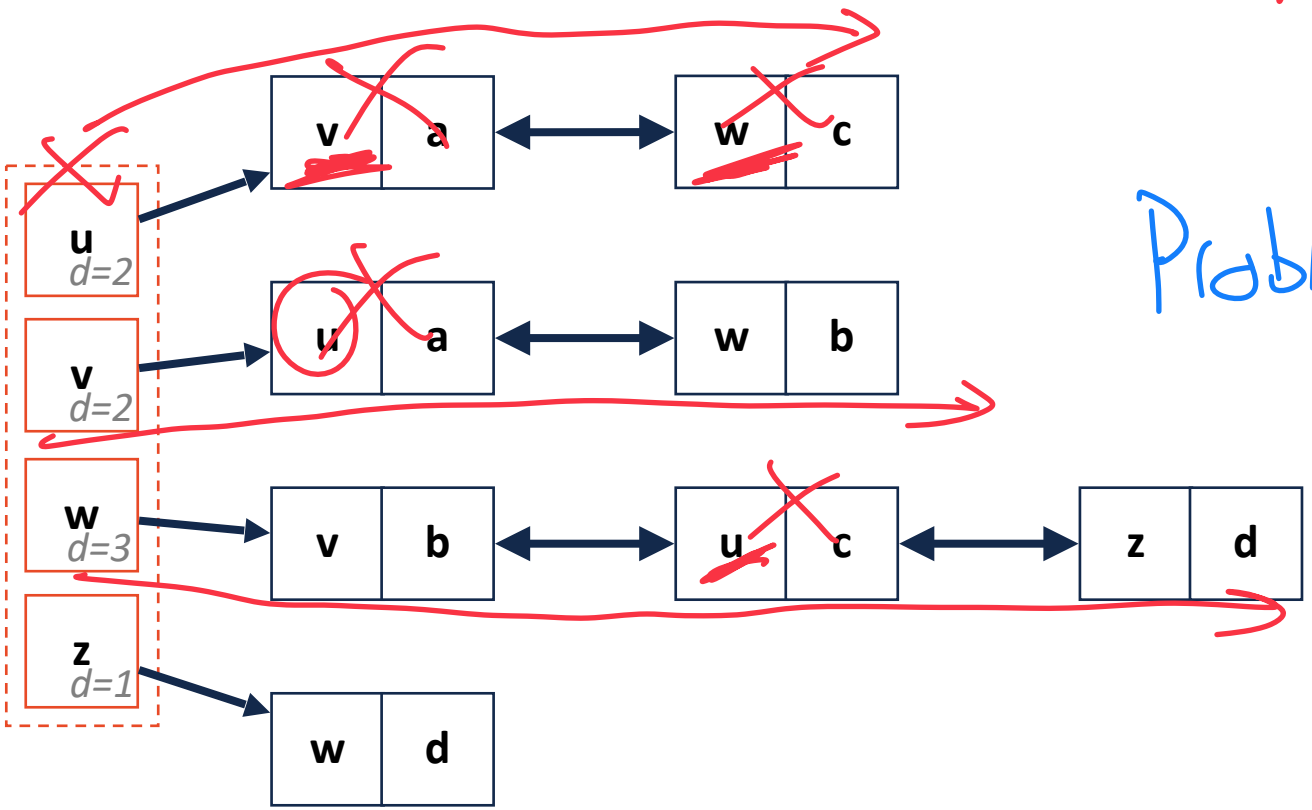
$\min[\deg(v_1), \deg(v_2)]$

$\leq O(n)$

Simple Adjacency List

|V|= n, |E|= m





removeVertex(Vertex v):

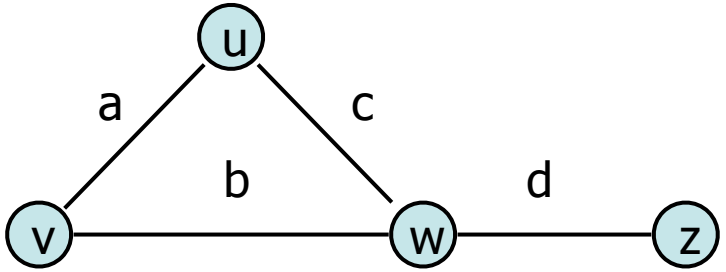
1) Walk down list Removing every Node
   ↳ walk down every node we see & look for v

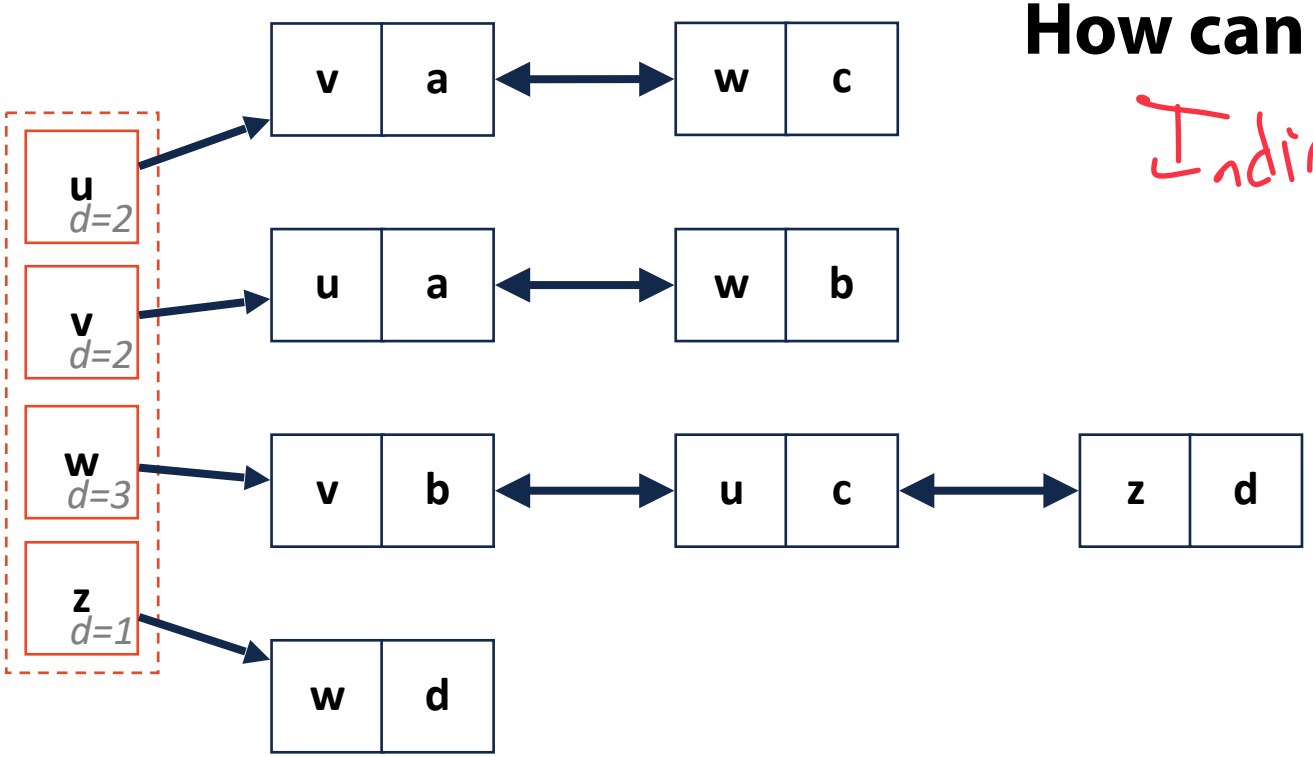
Problem; I have to walk through deg(v) LL. Each LL is deg(v)

↳ $O(n^2)$

Simple Adjacency List

|V|= n, |E|= m





What's wrong with our implementation?

↳ No connections between vertices

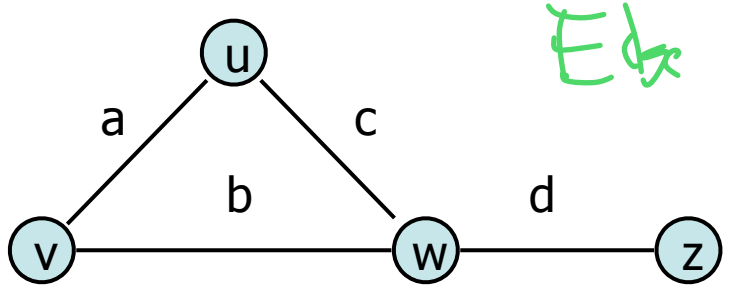
↳ Everything is stored twice

How can we fix it?

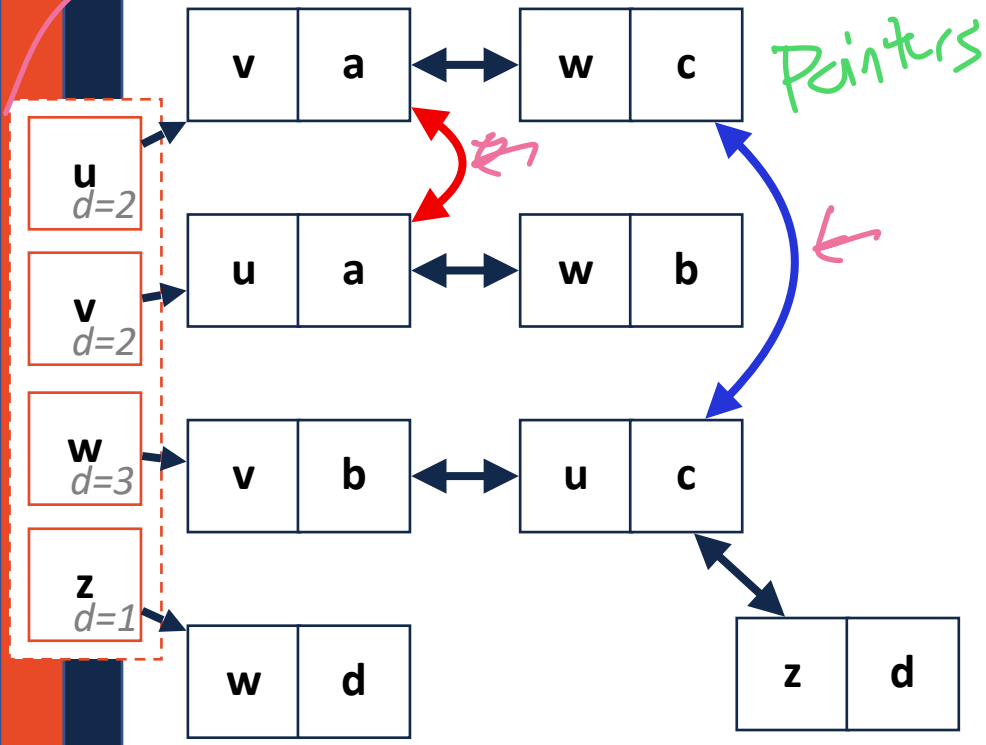
Indirection! (Pointers)

Adjacency List

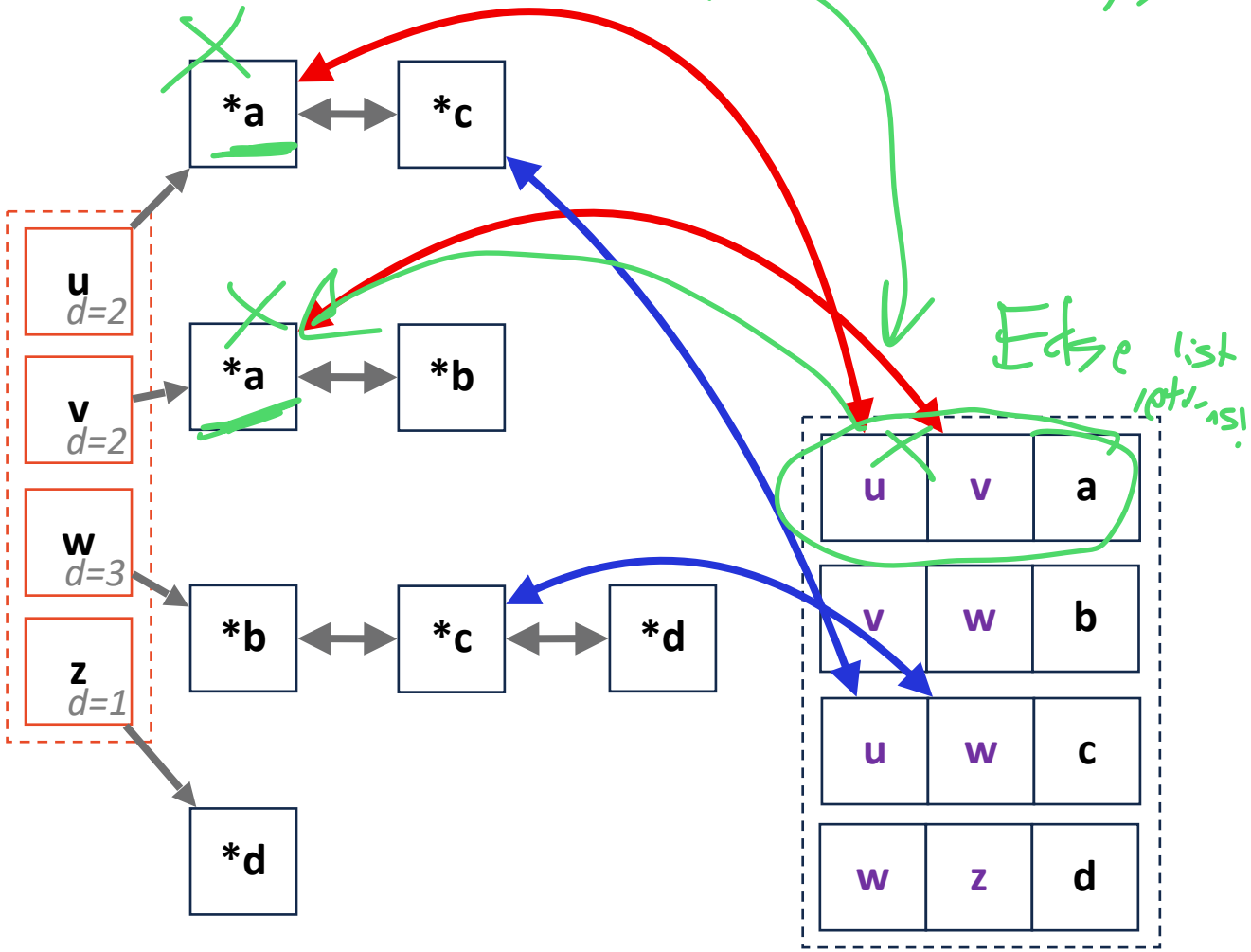
|V| = n, |E| = m



We had this

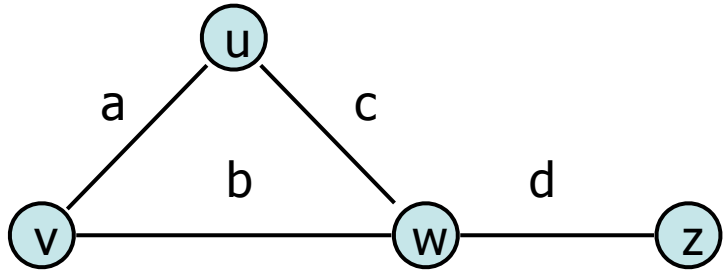


Edge list w/ Pointers to and from edges



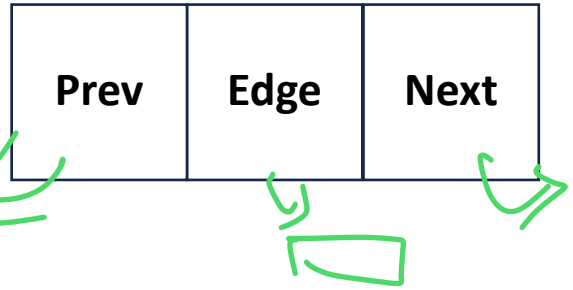
Adjacency List

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

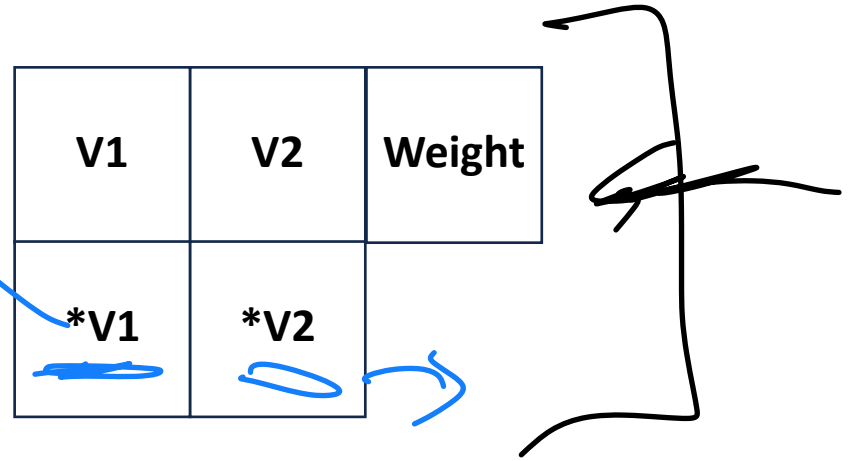


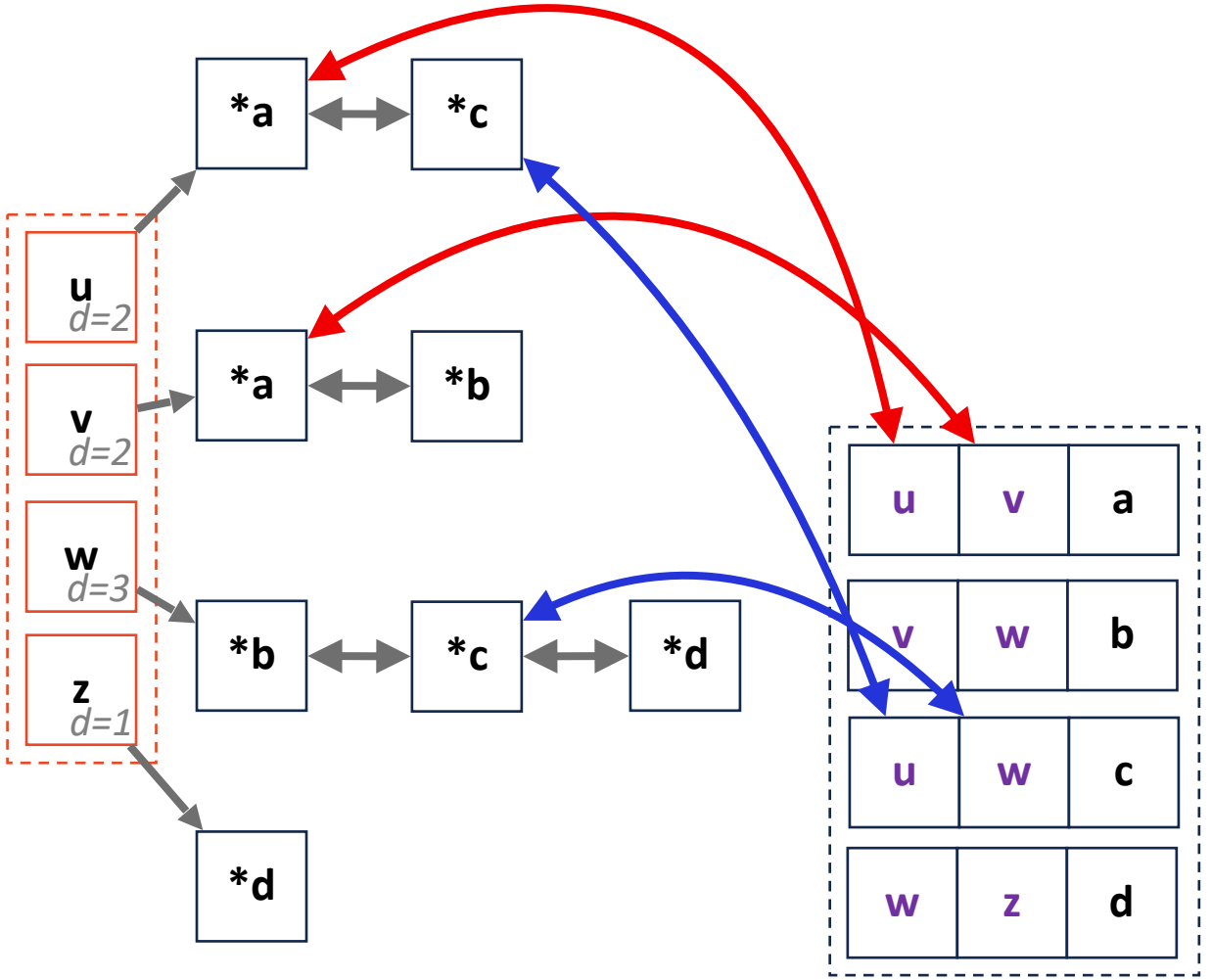
Adj List Node: Bidirectional LL (w/ tail pointer)

3 pointers

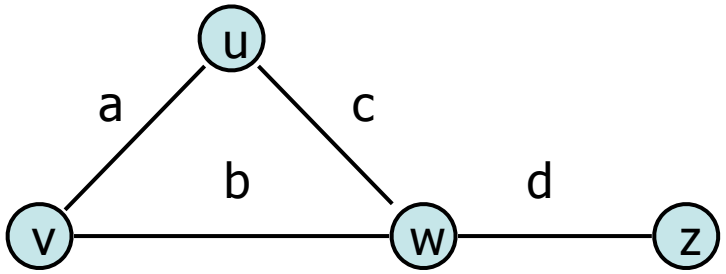


Edge List: Edge list + 2 pointers



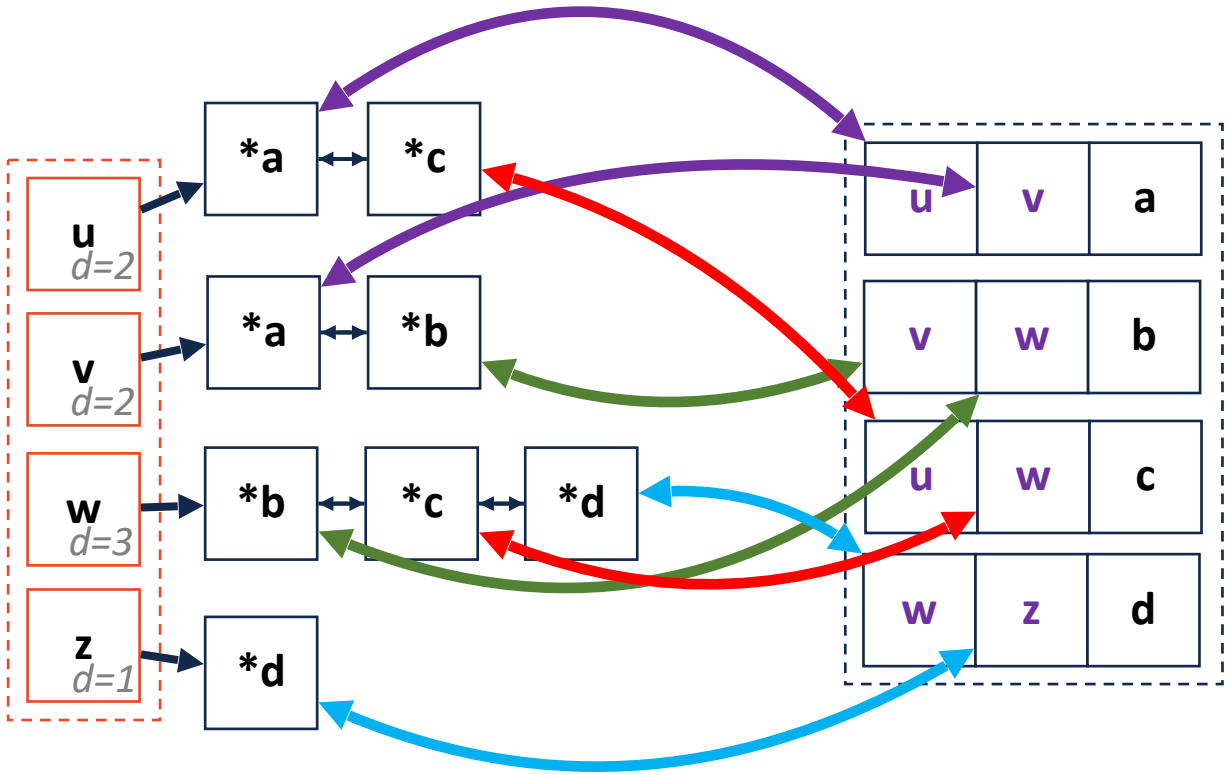


Adjacency List



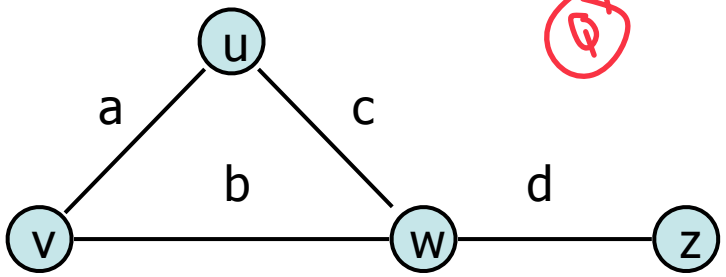
Vertex Storage: is Hash table (Key = Vertex label, Val = adj list)

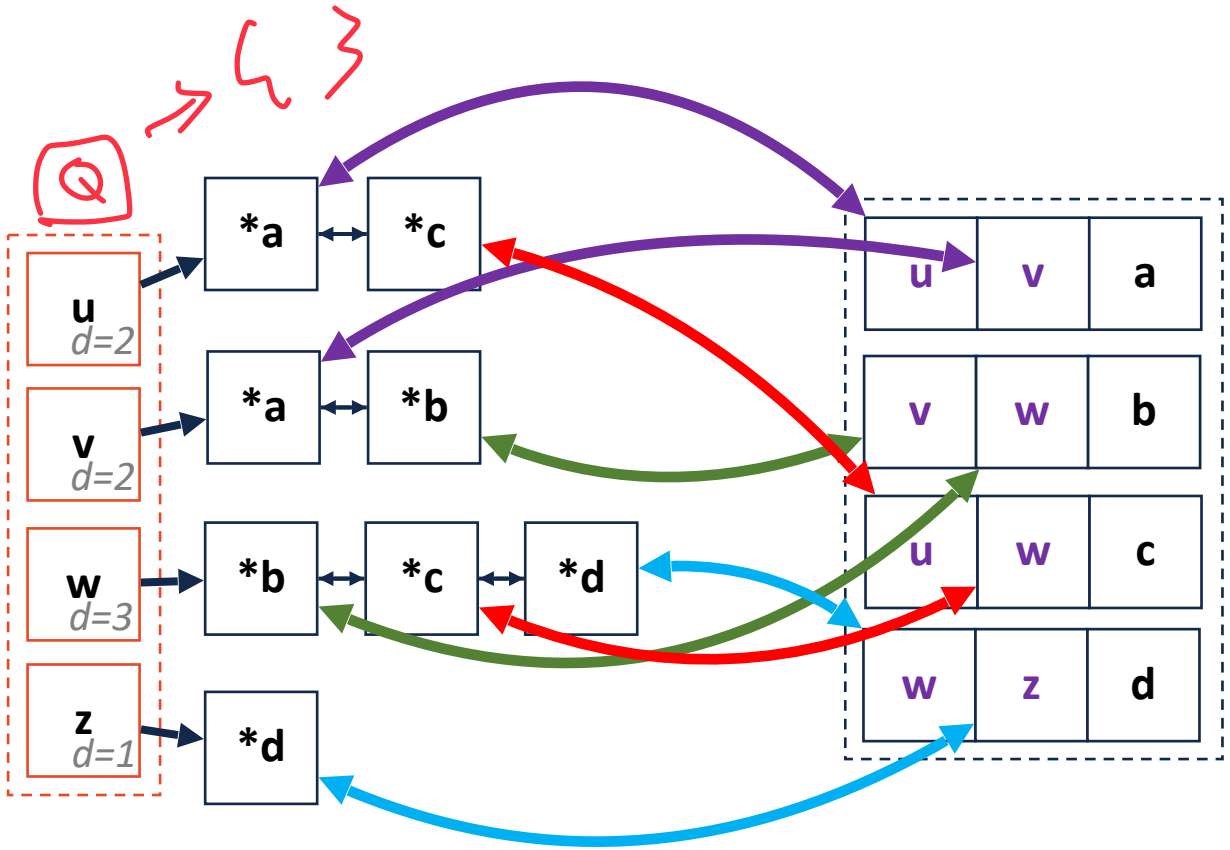
is knows size of linked list



Edge Storage: Edge list + 2 pointers

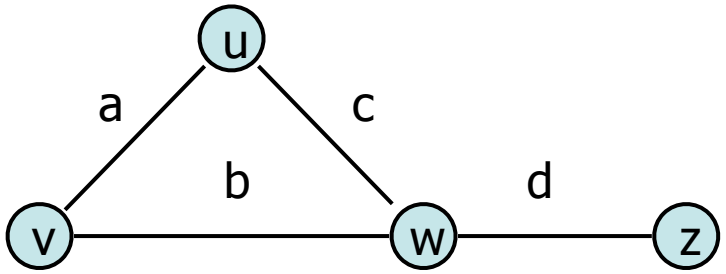
Adjacency List

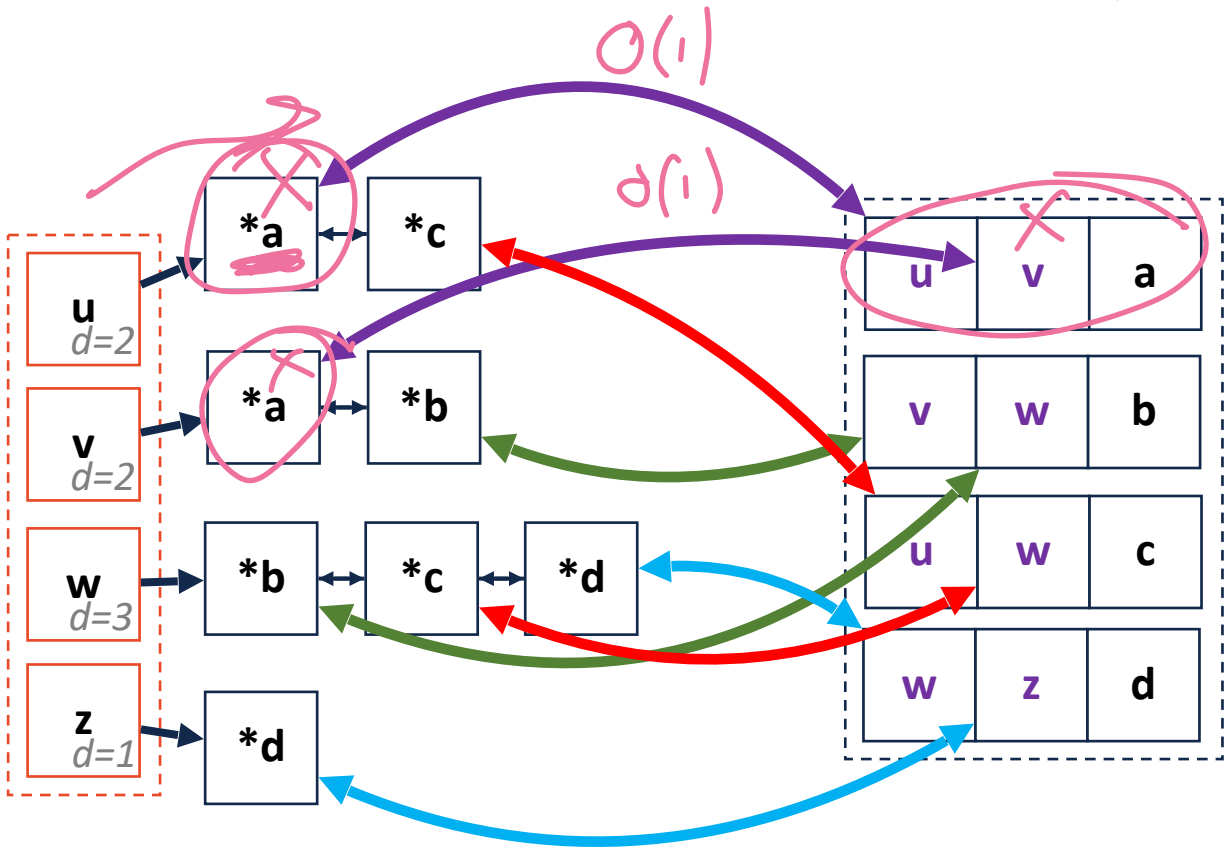




insertVertex(K key):

Adjacency List





removeVertex(Vertex v):

1) Remove every adj list node in V

2) Look at edge & find match in other vertex LL

3) Remove edge

Connections are all O(1)

≥ things at most

Each edge is O(1) to remove

deg(v) edges

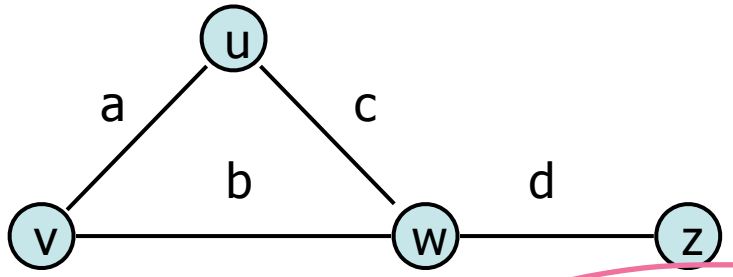
O(deg(v))

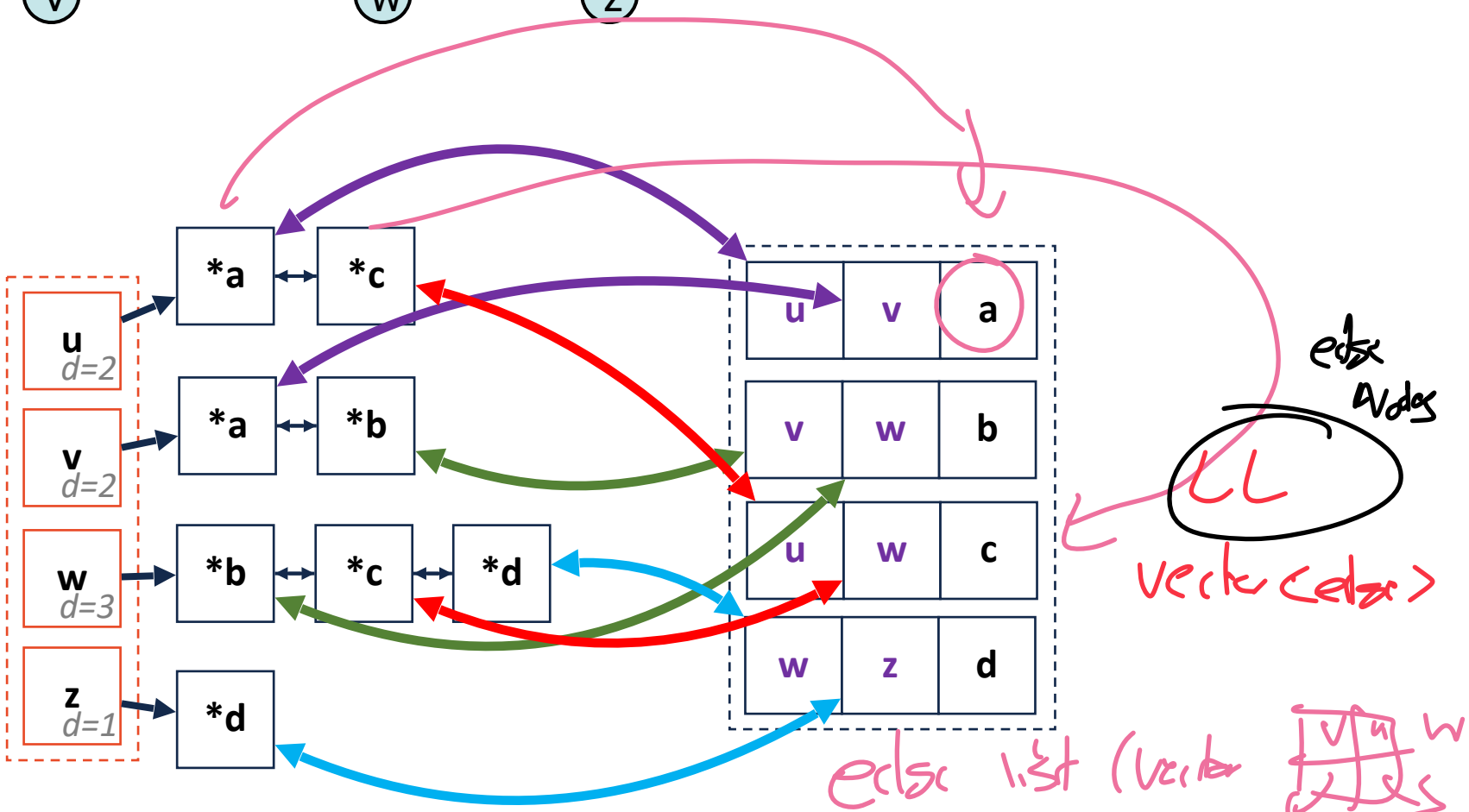
Adjacency List

incidentEdges(Vertex v):

↳ unchanged btwn simple

↳ deg(v)





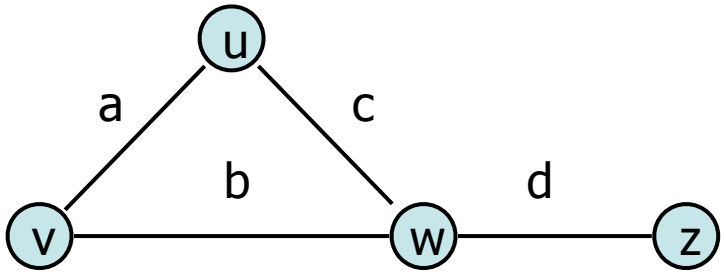
O[deg(v)]

∑∑

O(n)

edge list (vertex

Adjacency List

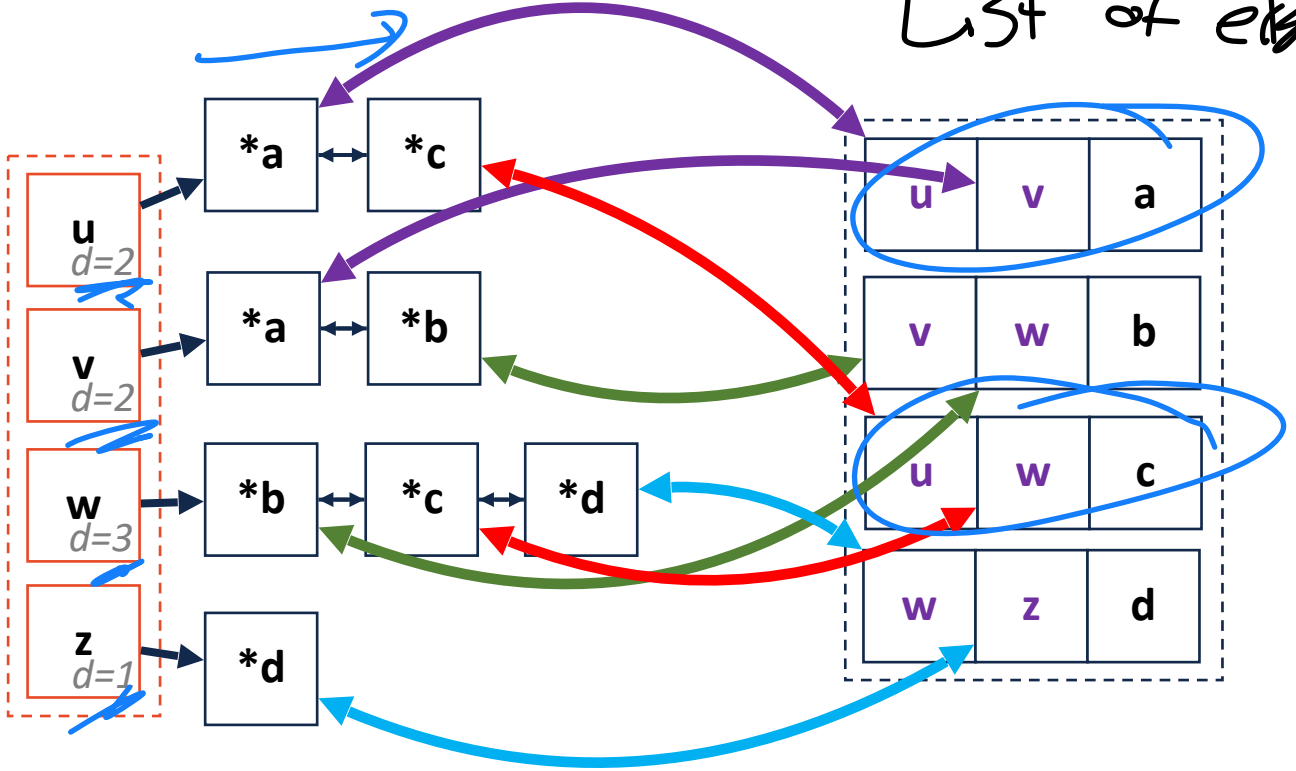


areAdjacent(Vertex v1, Vertex v2):

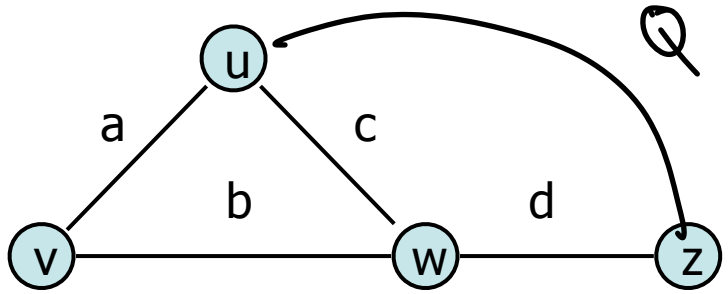
↳ deg (v)

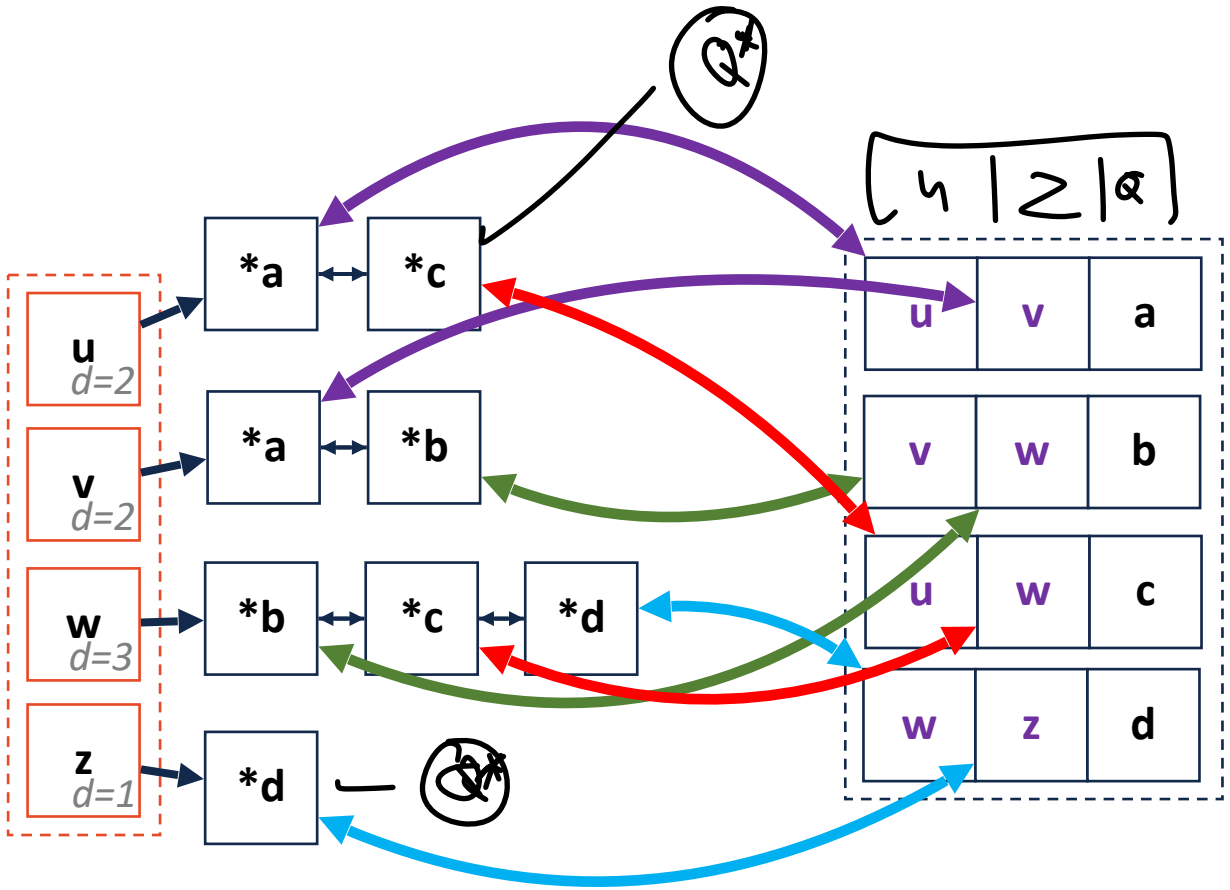
min [deg(v1), deg(v2)]

List of edges



Adjacency List





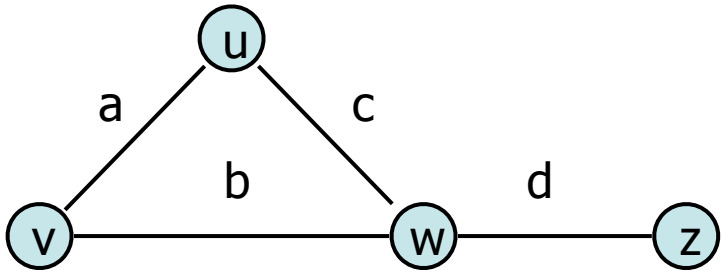
insertEdge(Vertex v1, Vertex v2, K key):

2 hil points insert O(1)

+

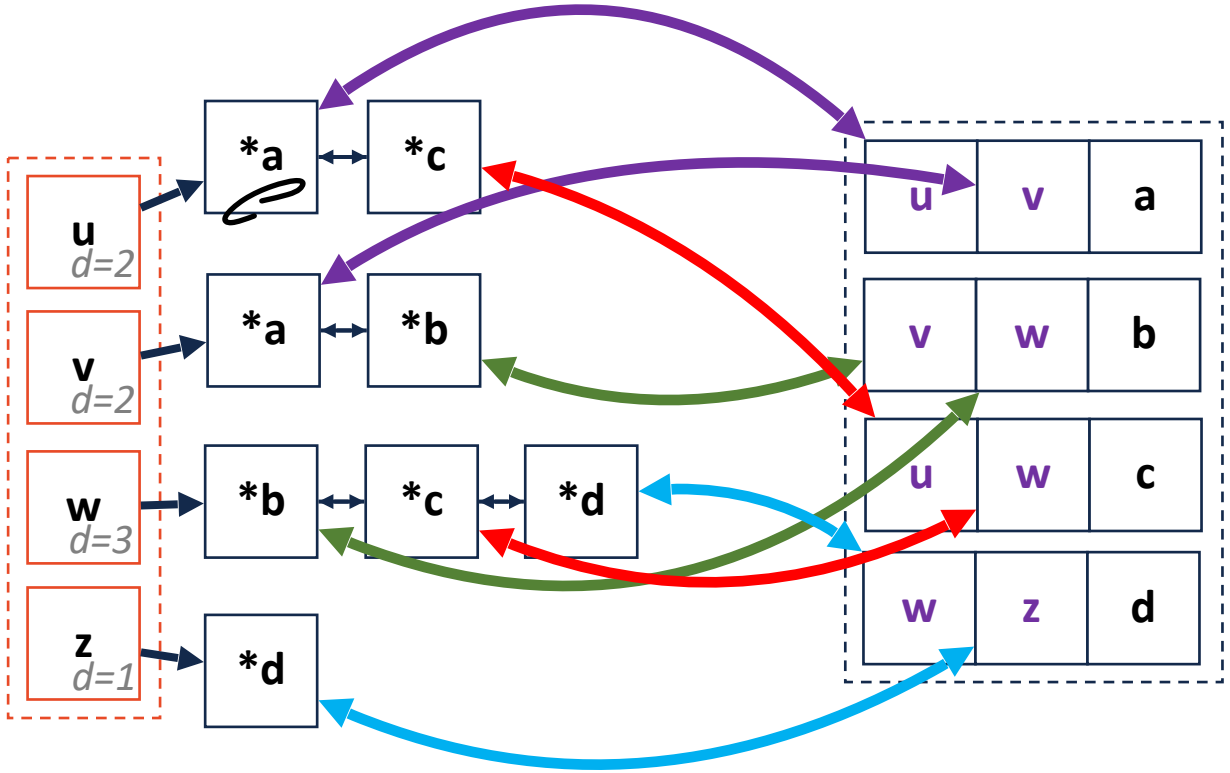
1 edge insert O(1)*

Adjacency List



removeEdge(Vertex v1, Vertex v2, ~~Kkey~~):

min (deg(v1), deg(v2))



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

we will review on friday

Expressed as O(f)	Edge List	Adjacency Matrix	Adjacency List
Space	n+m	n^2	n+m
insertVertex(v)	1* ← tail pointer or amortized	n*	1*
removeVertex(v)	m** ← hash table for vertex; Min otherwise	n	deg(v)***
insertEdge(u, v)	1	1	1*
removeEdge(u, v)	m	1	min(deg(u), deg(v))
incidentEdges(v)	m	n	deg(v)
areAdjacent(u, v)	m	1	min(deg(u), deg(v))

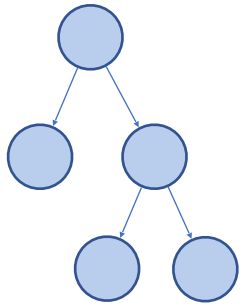
sparse

Graph Traversals

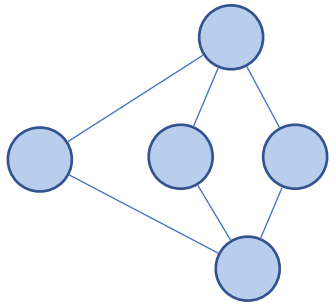
There is no clear order in a graph (even less than a tree!)

How can we systematically go through a complex graph in the fewest steps?

Tree traversals won't work — lets compare:

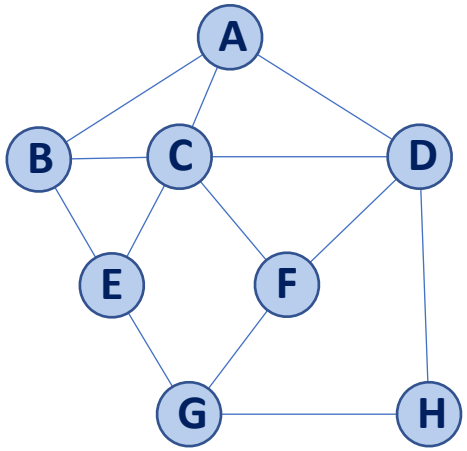


- Rooted
- Acyclic
-

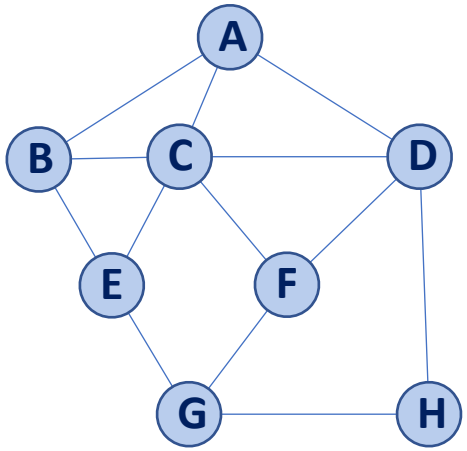


-
-
-

Traversal: BFS



Traversal: BFS



v	d	P	Adjacent Edges
A			B C D
B			A C E
C			A B D E F
D			A C F H
E			B C G
F			C D G
G			E F H
H			D G