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)

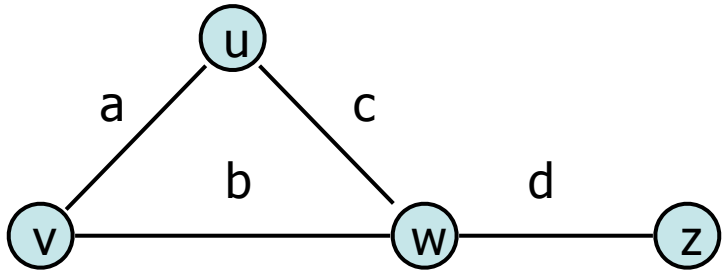
Extra Credit Project check-in deadline modified

Learning Objectives

Discuss graph implementation and storage strategies

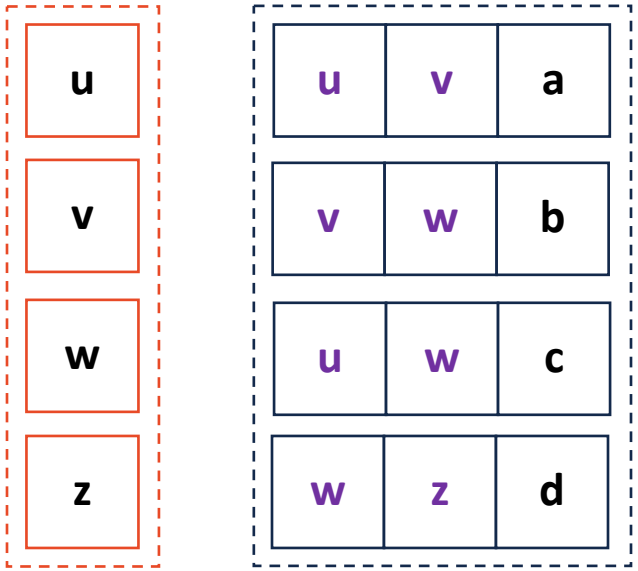
Introduce graph traversals

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



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

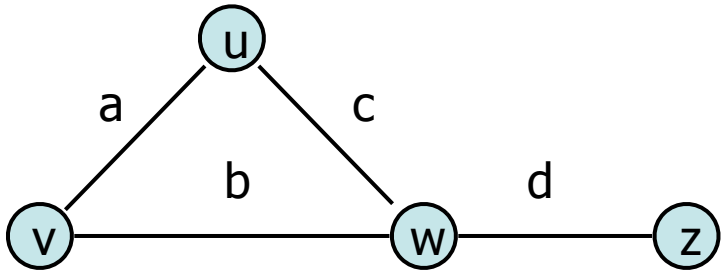
removeVertex(Vertex v):
removeEdge(Vertex v1, Vertex v2, K key):



incidentEdges(Vertex v):
areAdjacent(Vertex v1, Vertex v2):

Graph Implementation: Adjacency Matrix

|V|= n, |E|= m



insertVertex(K key):

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

removeVertex(Vertex v):

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

incidentEdges(Vertex v):

areAdjacent(Vertex v1, Vertex v2):

u
v
w
z

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

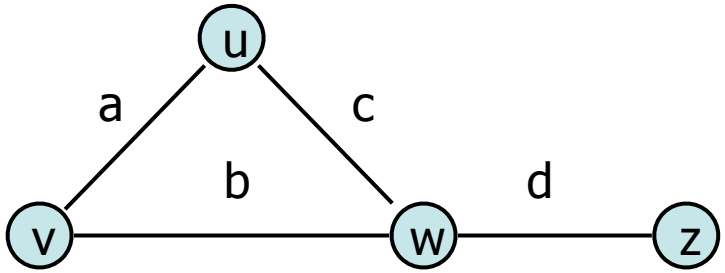
Graph Implementations

We want something…

Faster than an edge list

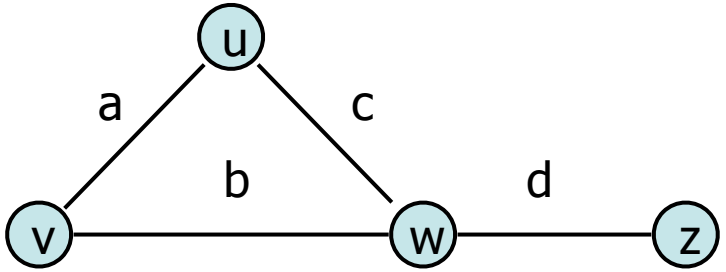
Less space than an adjacency matrix

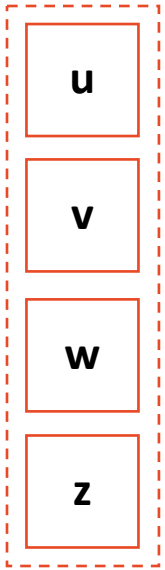
Particularly good at finding adjacent elements

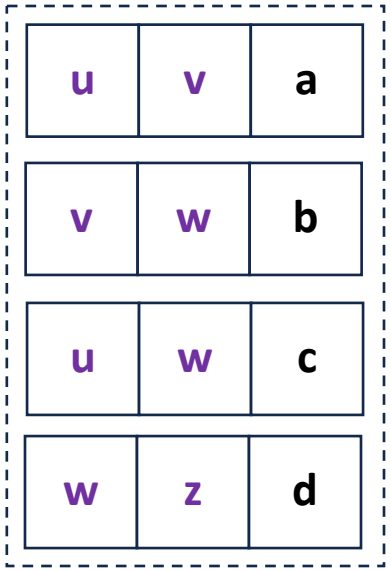


Graph Implementation: Edge List + ?

|V|= n,|E|= m

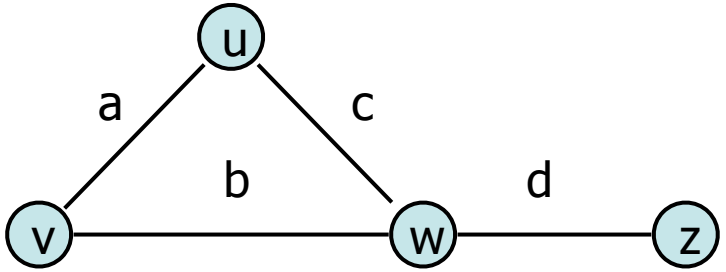


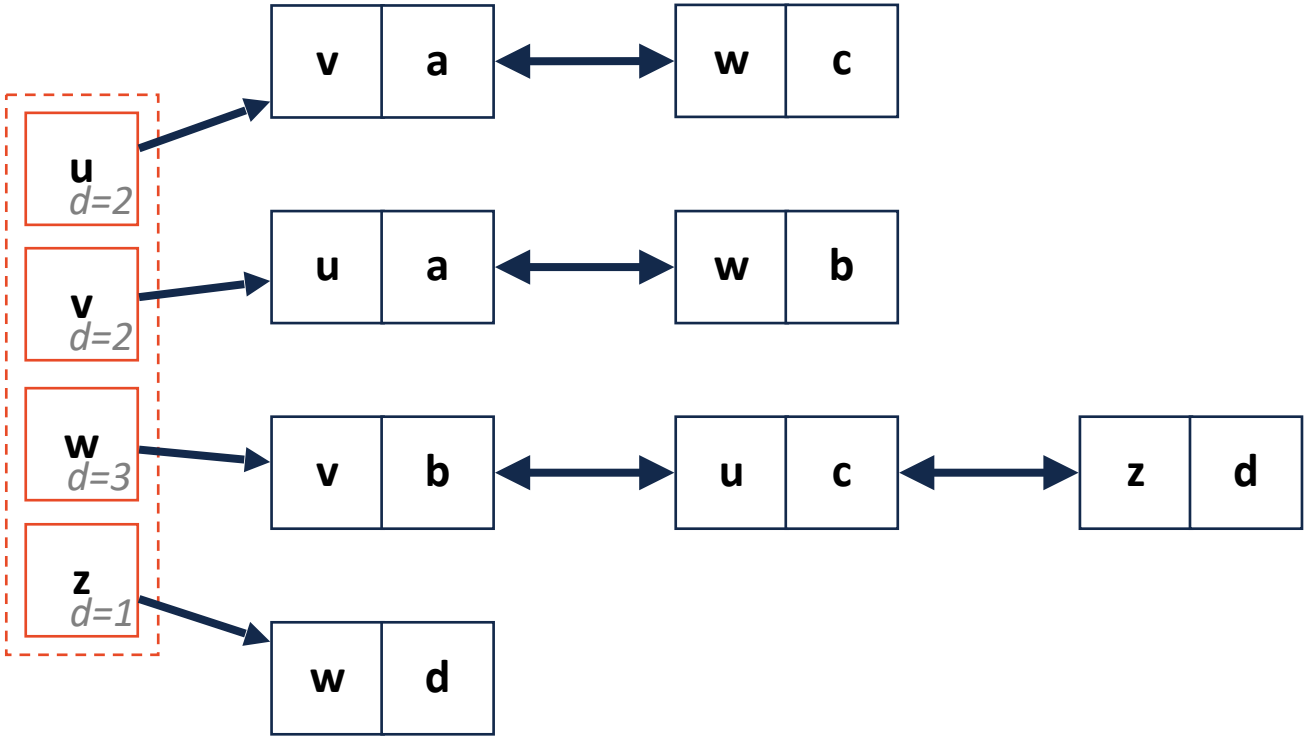




Simple Adjacency List

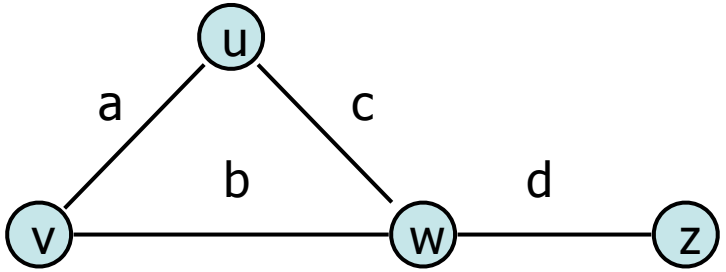
|V|= n,|E|= m



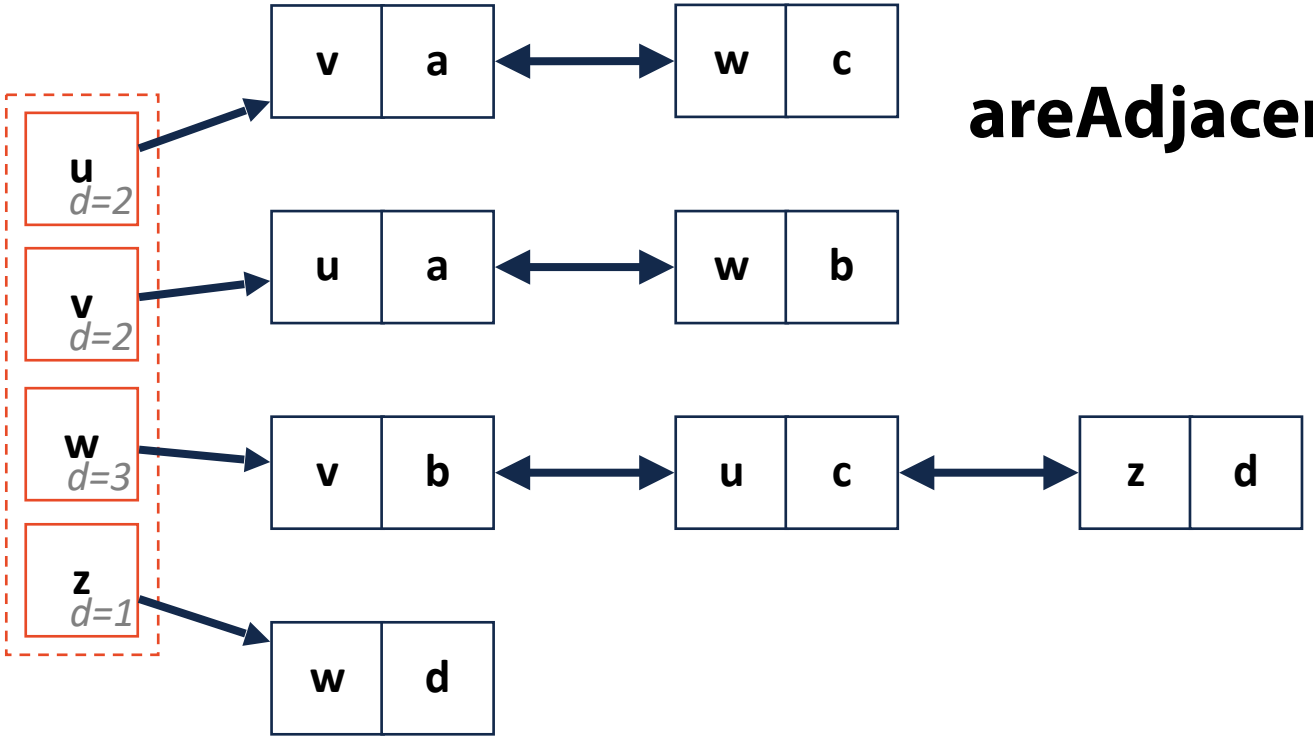


Simple Adjacency List

|V|= n, |E|= m



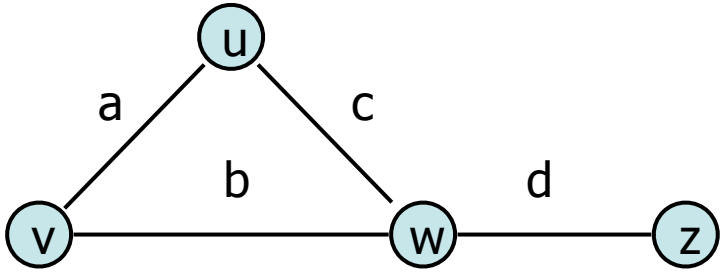
incidentEdges(Vertex v):



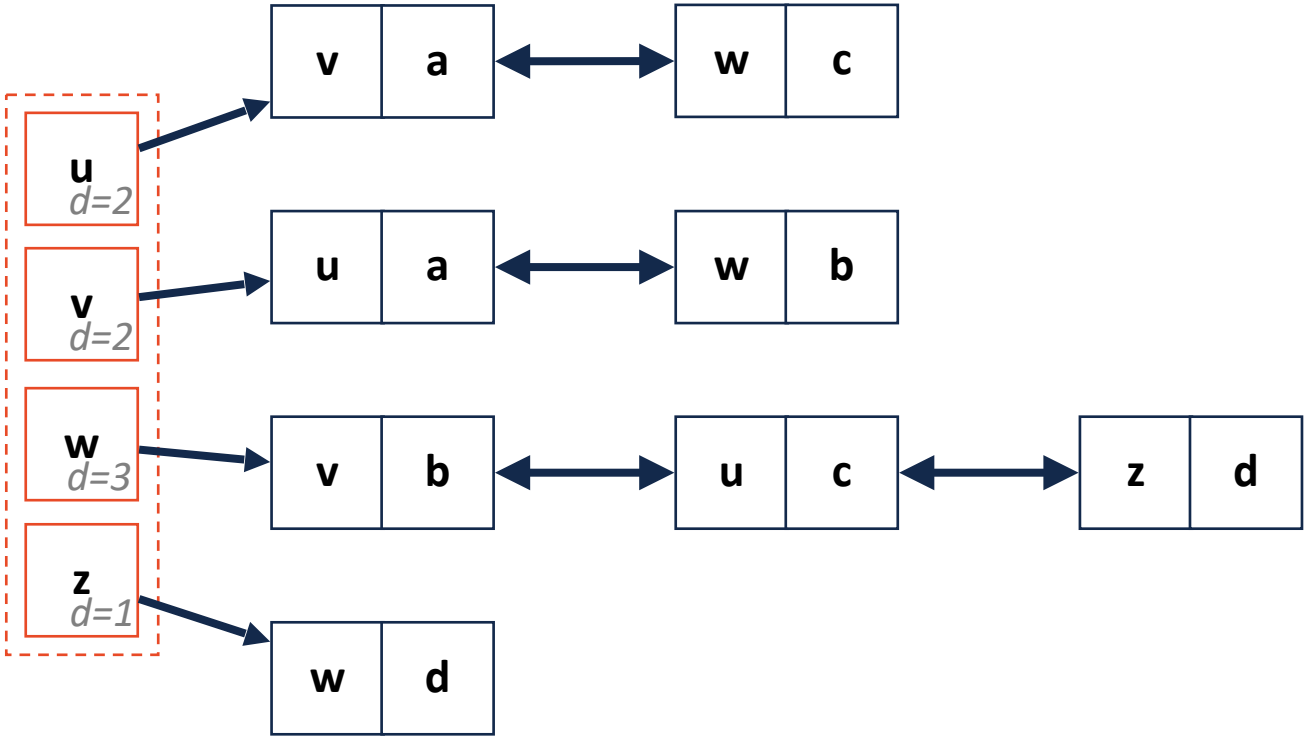
areAdjacent(Vertex v1, Vertex v2):

Simple Adjacency List

|V|= n, |E|= m

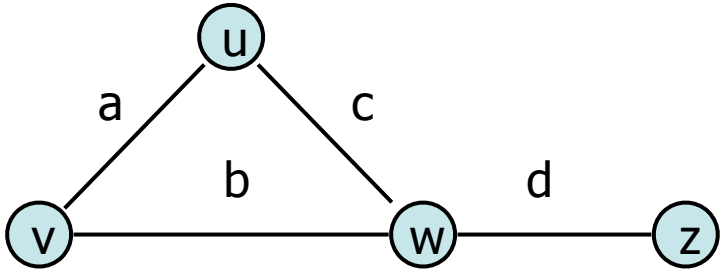


removeVertex(Vertex v):



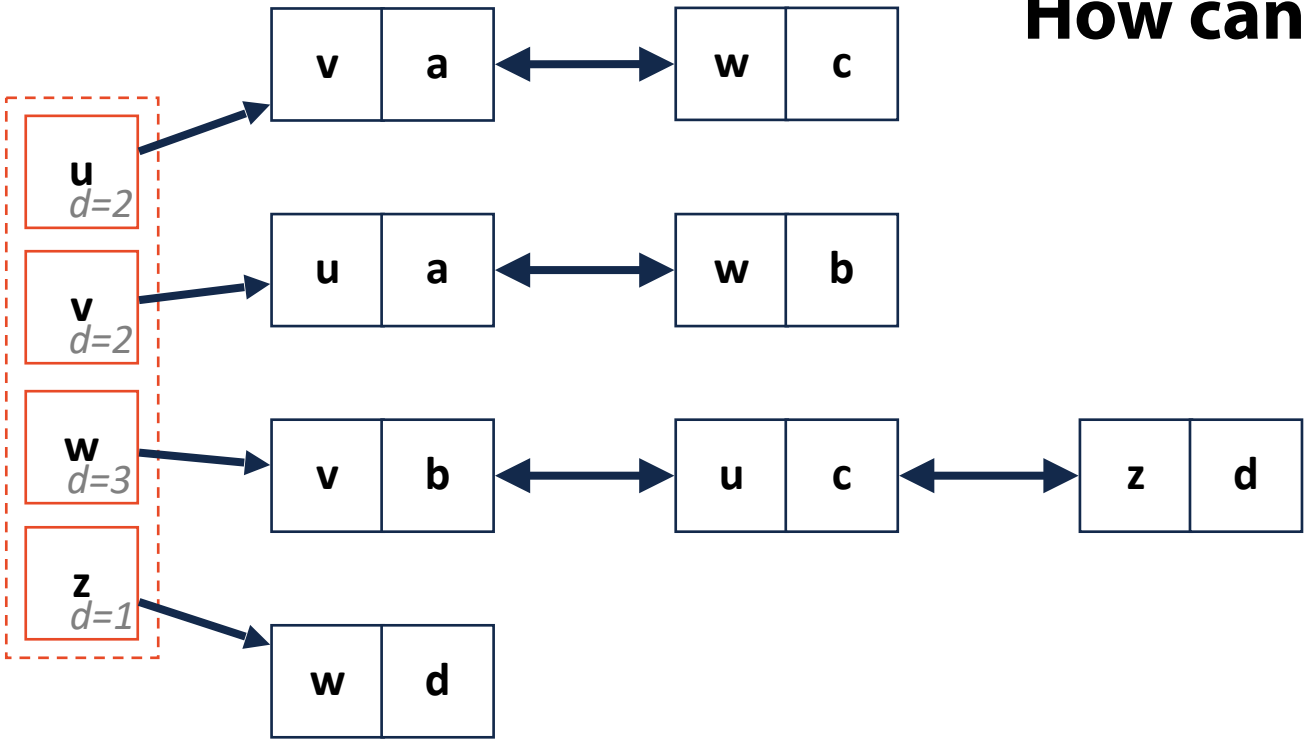
Simple Adjacency List

|V|= n, |E|= m



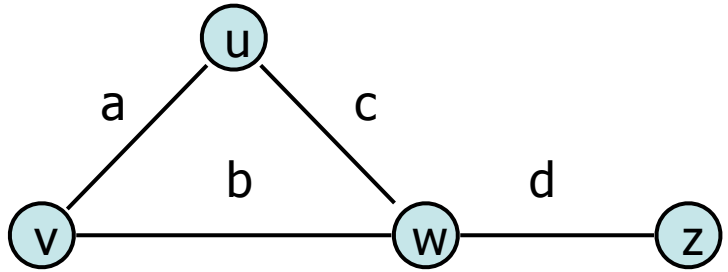
What's wrong with our implementation?

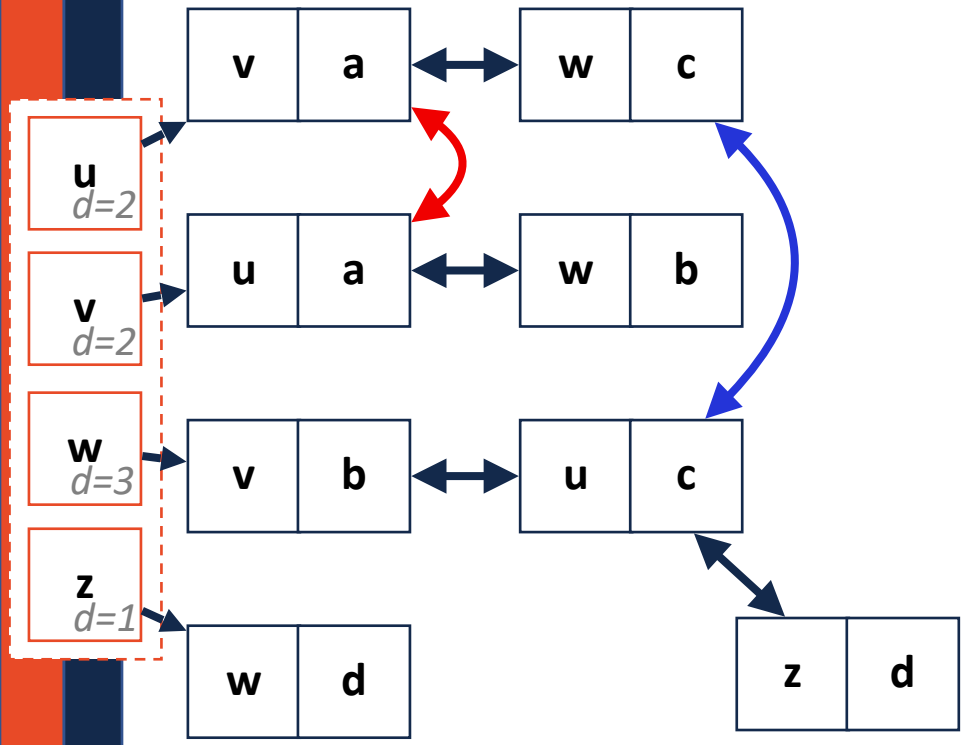
How can we fix it?

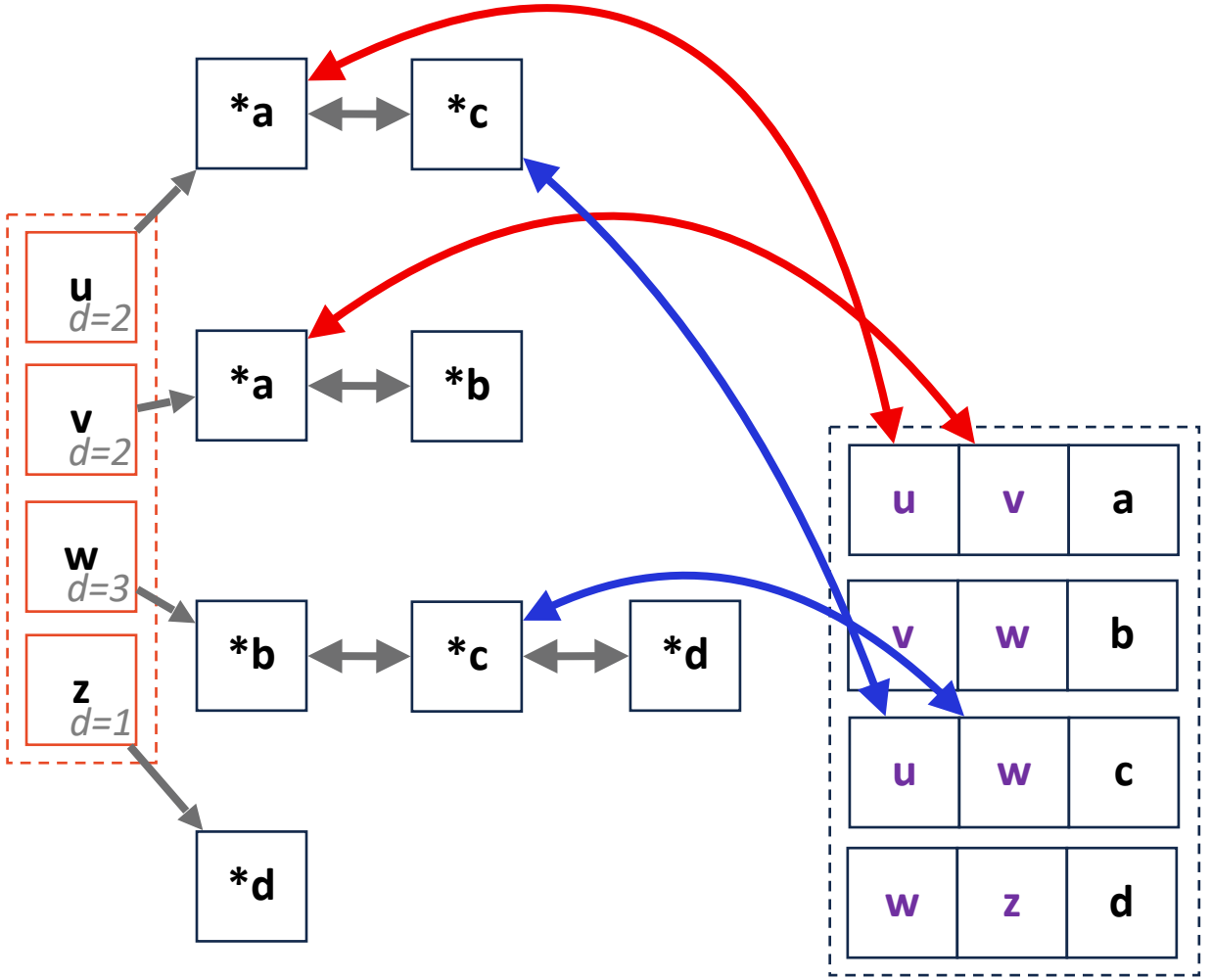


Adjacency List

|V|= n, |E|= m

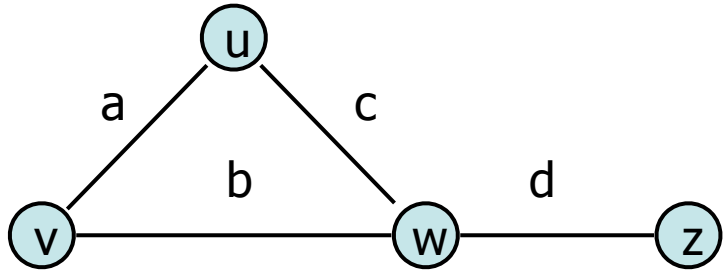






Adjacency List

|V|= n, |E|= m

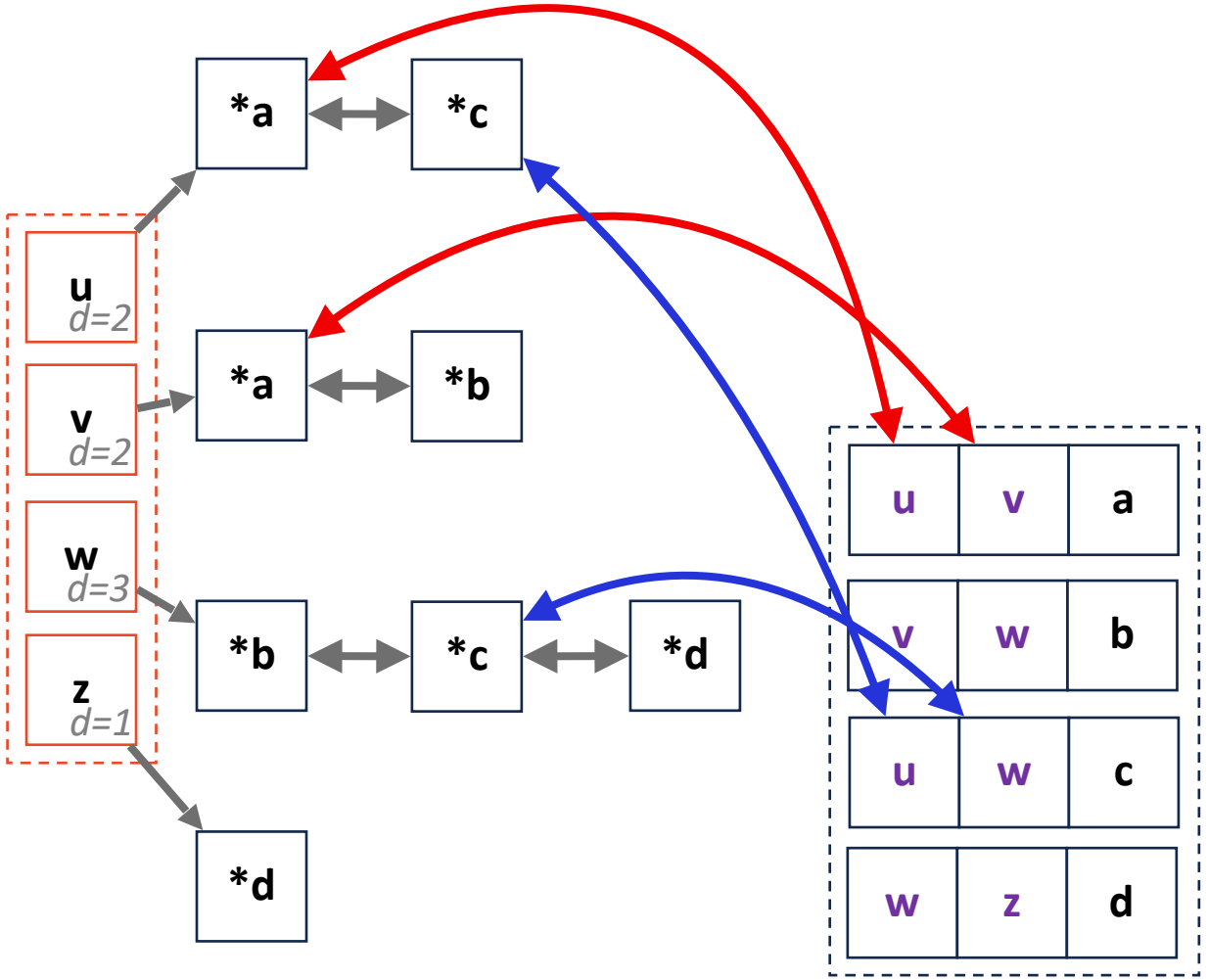


Adj List Node:

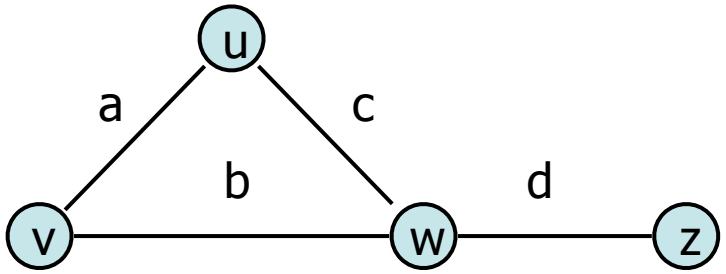
Prev	Edge	Next

Edge List:

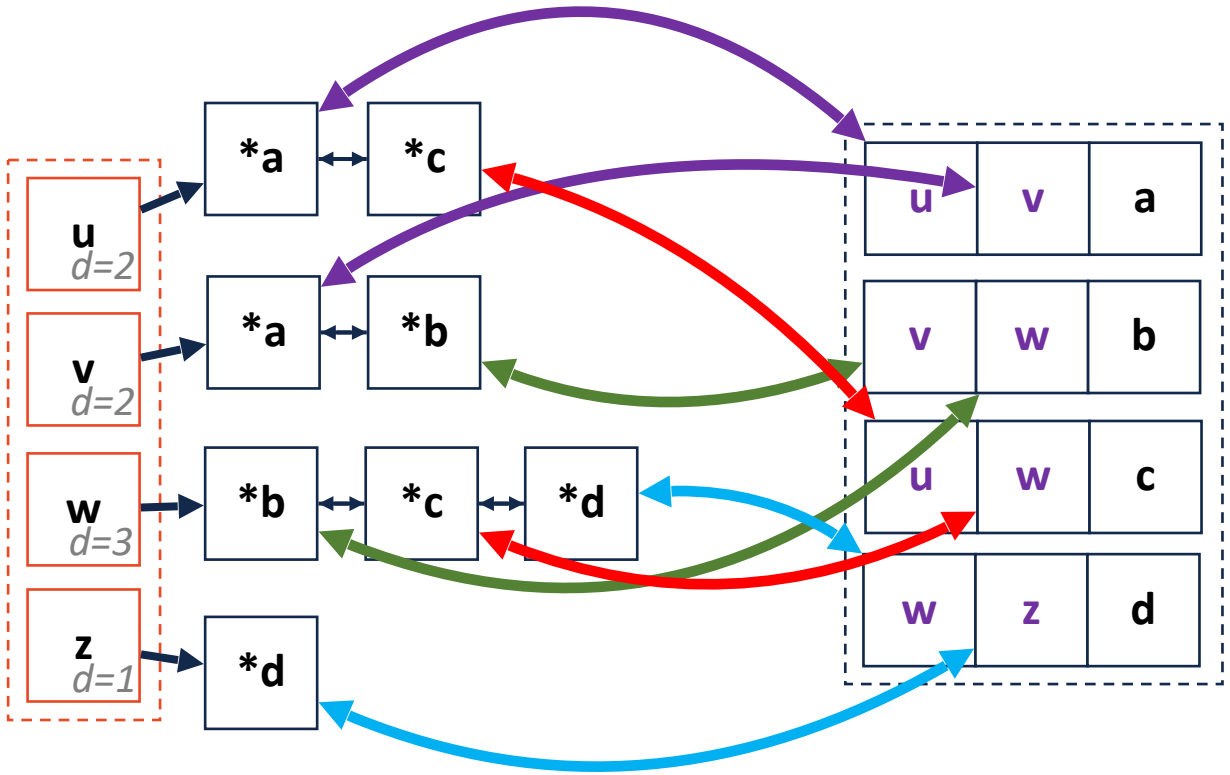
V1	V2	Weight
*V1	*V2	



Adjacency List



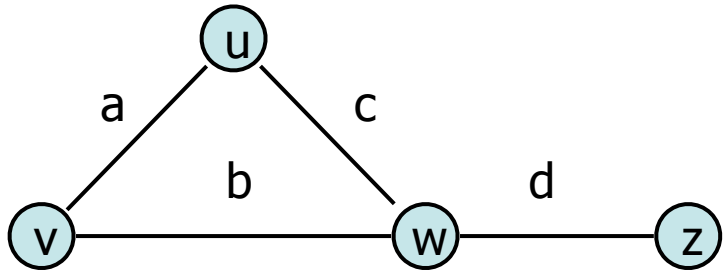
Vertex Storage:

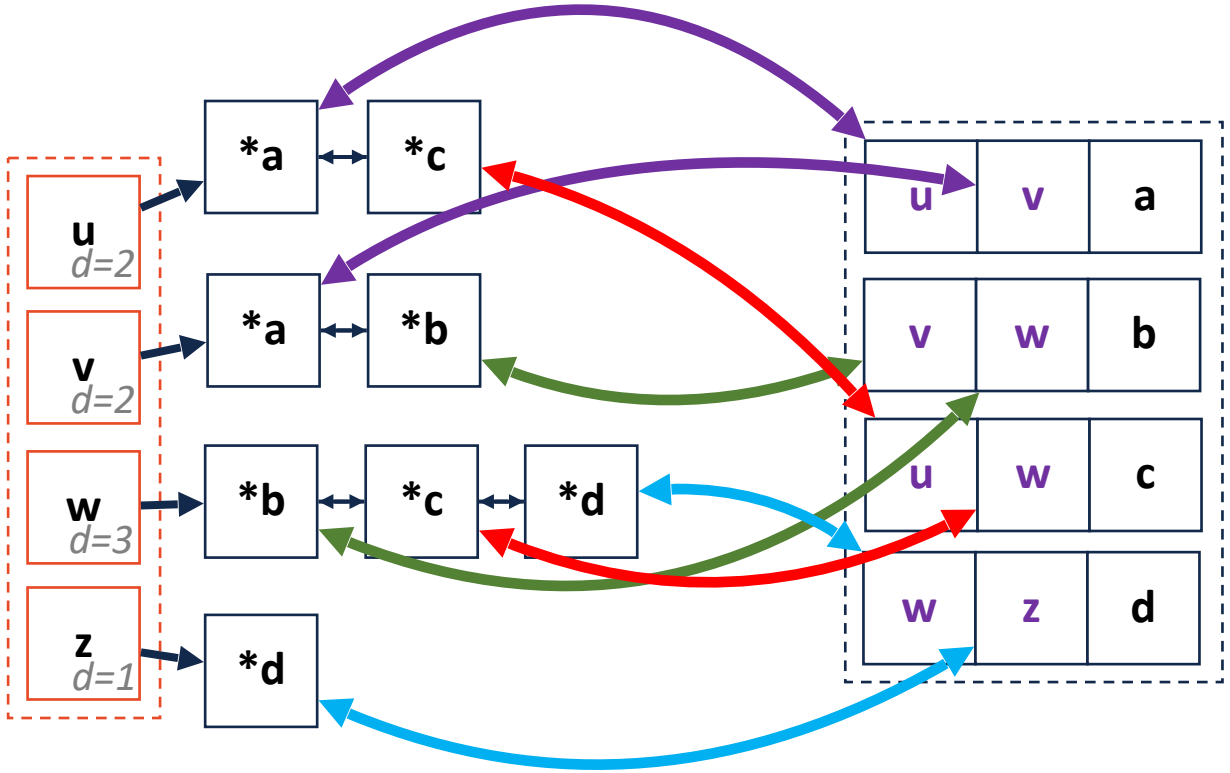


Edge Storage:

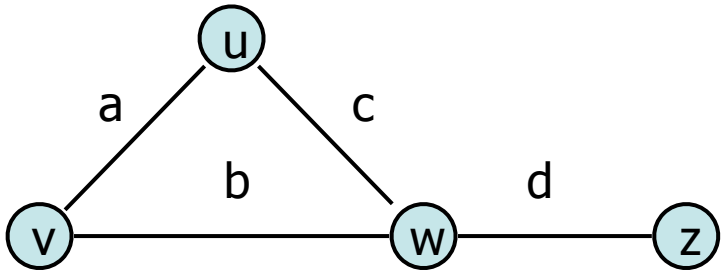
Adjacency List

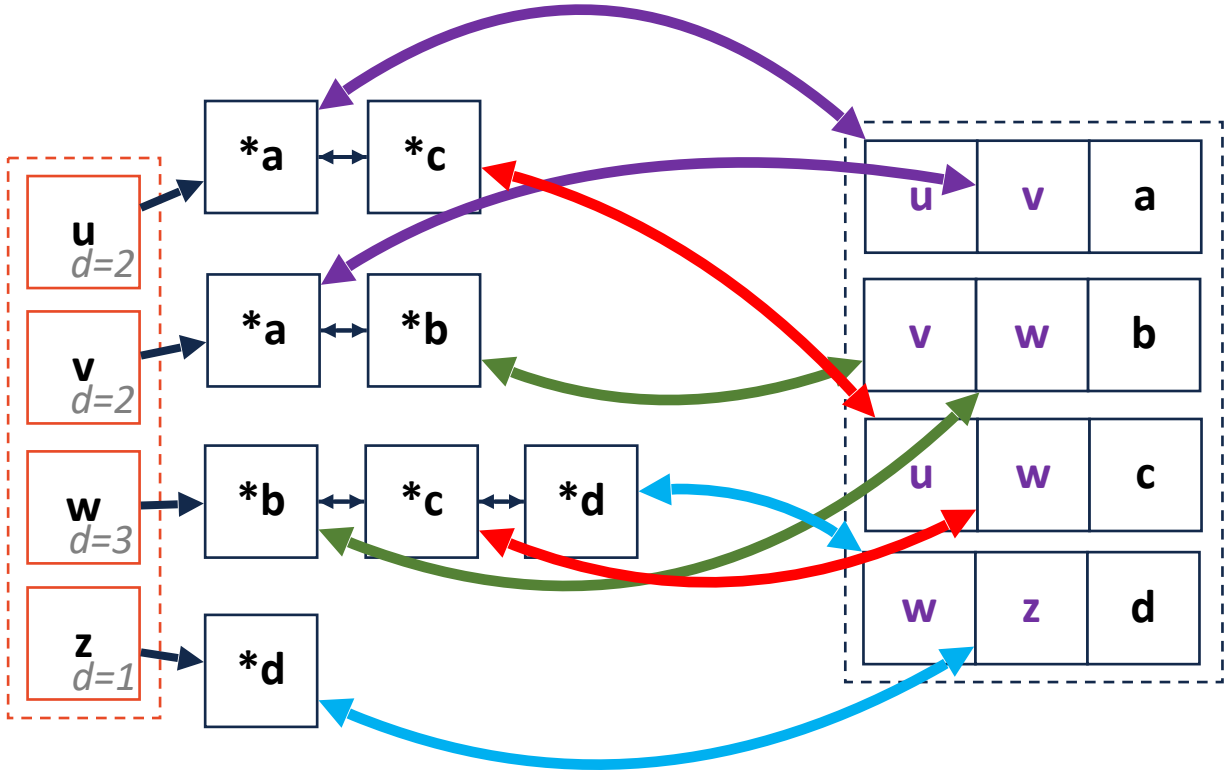
insertVertex(K key):





Adjacency List

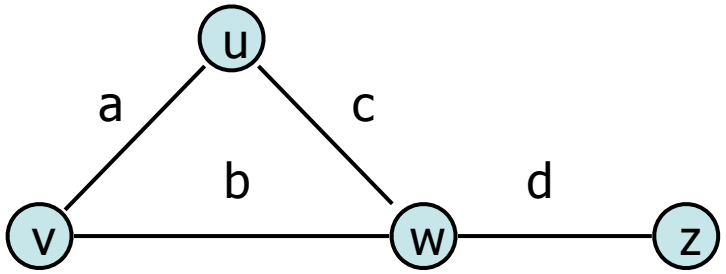


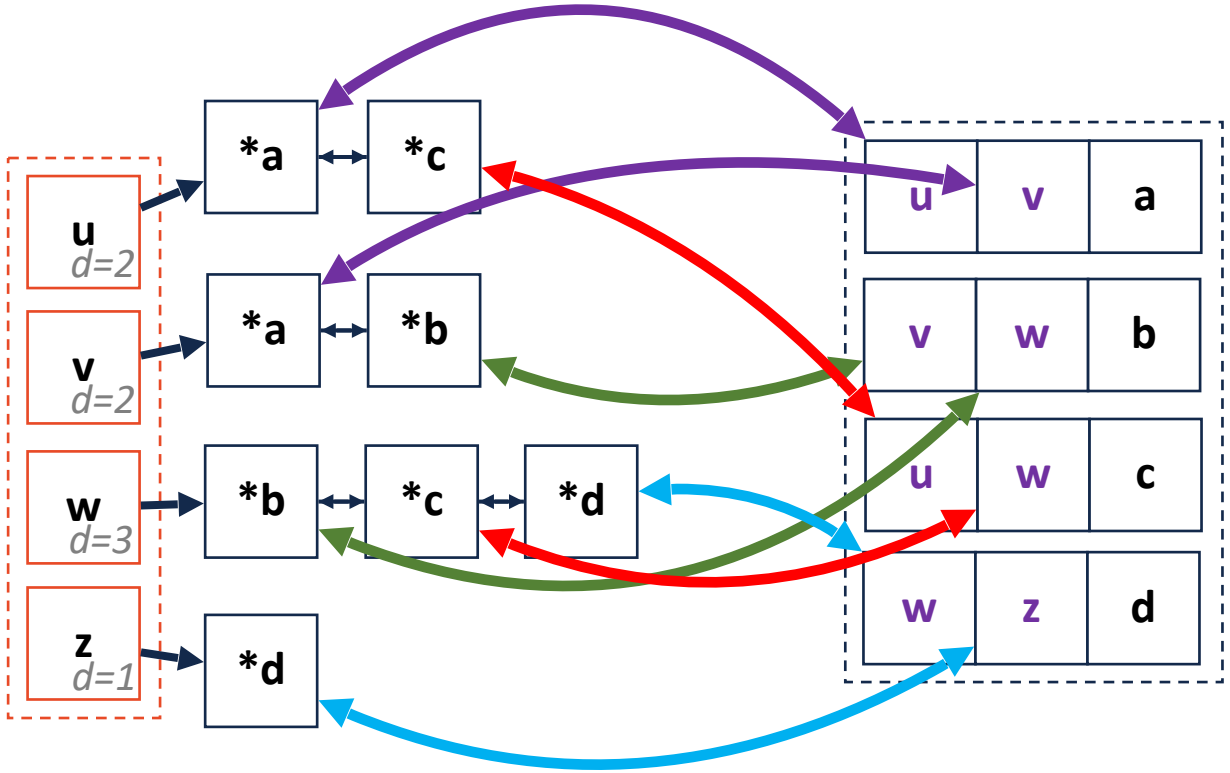


removeVertex(Vertex v):

Adjacency List

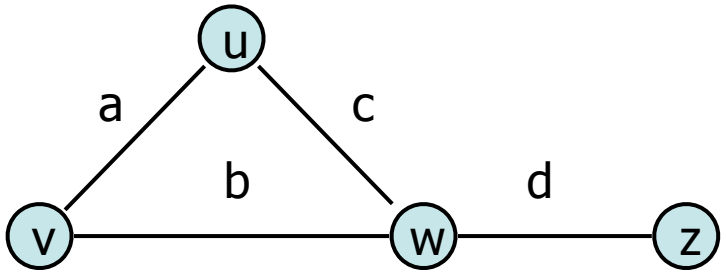
incidentEdges(Vertex v):

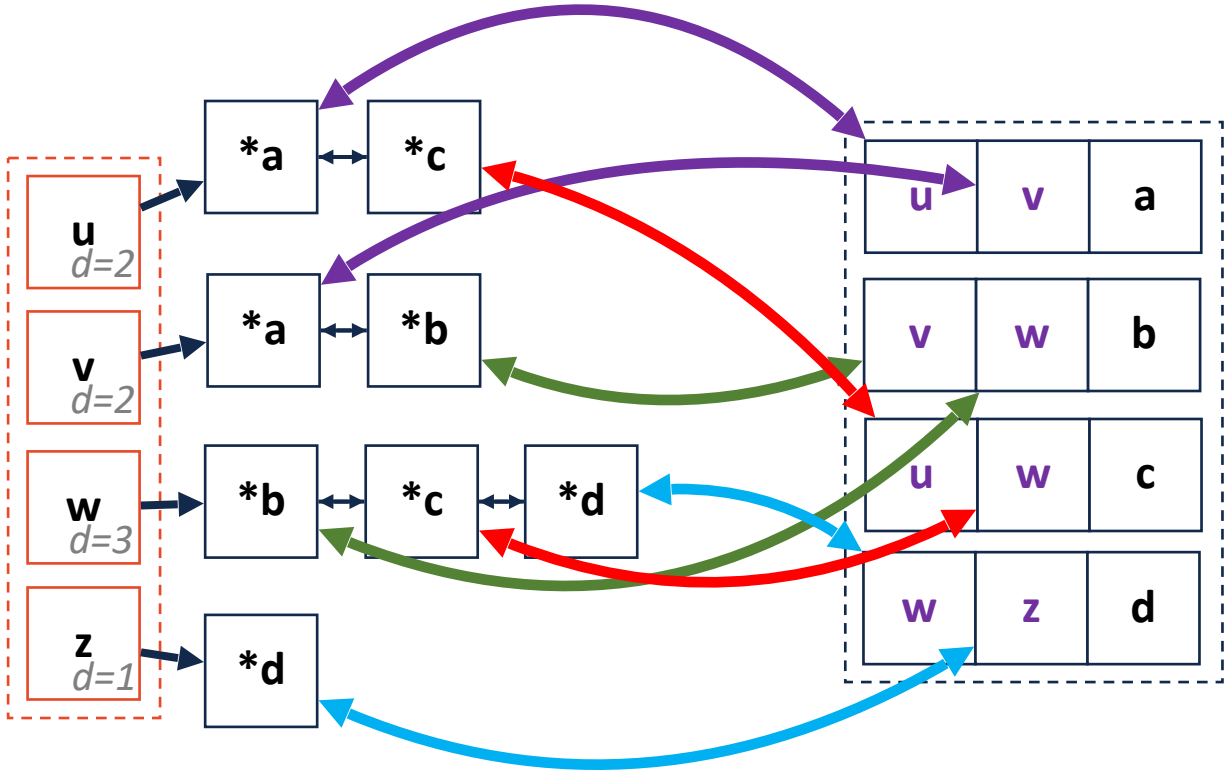




Adjacency List

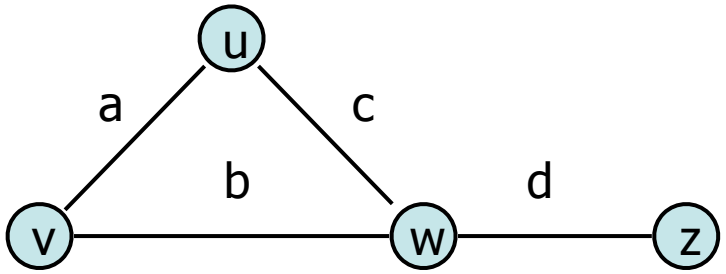
areAdjacent(Vertex v1, Vertex v2):

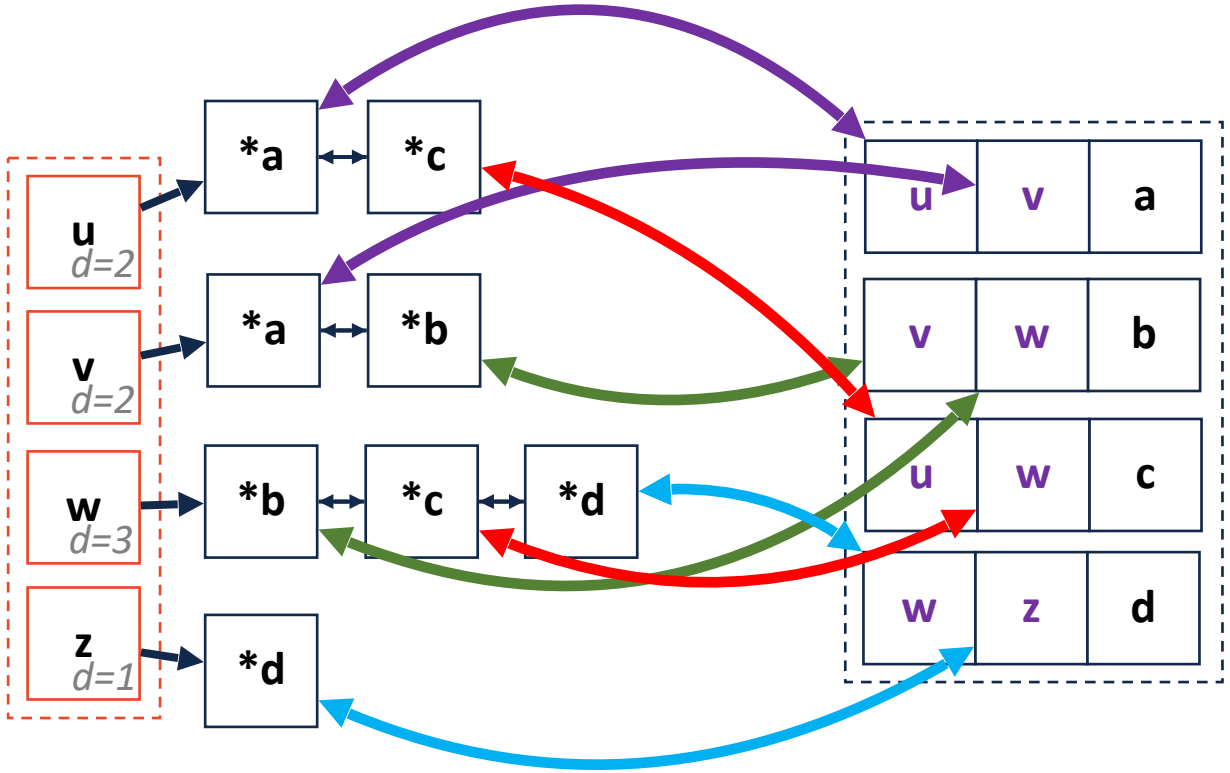




Adjacency List

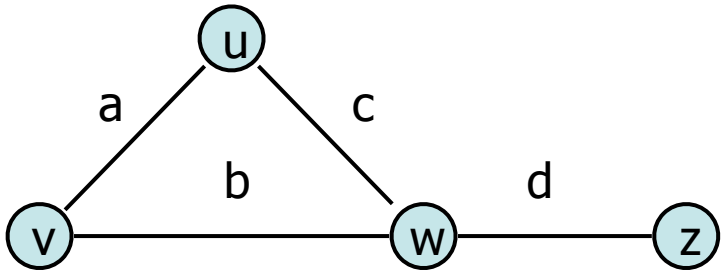
insertEdge(Vertex v1, Vertex v2, K key):

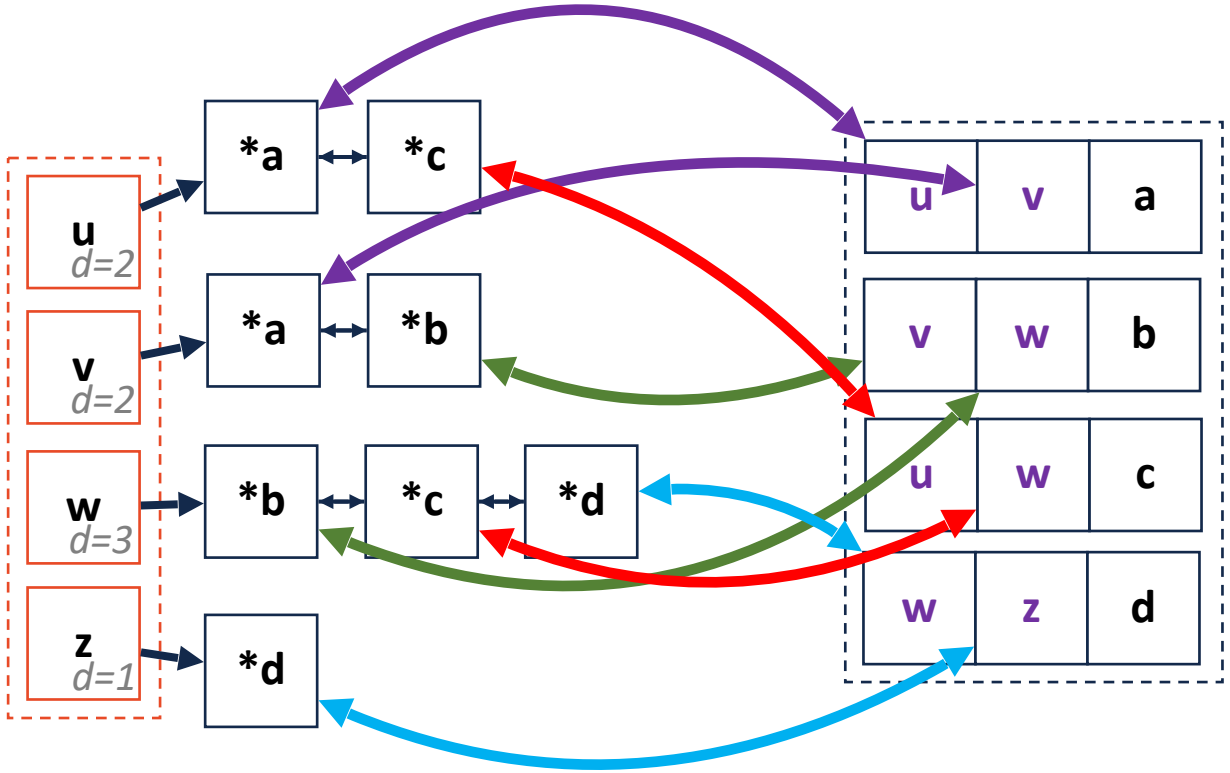




Adjacency List

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





|V|= n, |E|= m

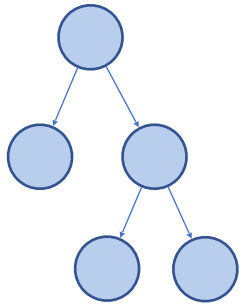
Expressed as O(f)	Edge List	Adjacency Matrix	Adjacency List
Space	n+m	n^2	n+m
insertVertex(v)	1*	n*	1*
removeVertex(v)	m**	n	deg(v)***
insertEdge(u, v)	1	1	1*
removeEdge(u, v)	m	1	min(deg(u), deg(v))
incidentEdges(v)	m	n	deg(v)
areAdjacent(u, v)	m	1	min(deg(u), deg(v))

Graph Traversals

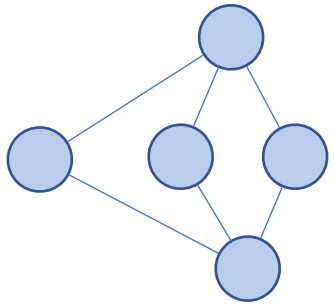
There is no clear order in a graph (even less than a tree!)

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

Tree traversals won't work — lets compare:

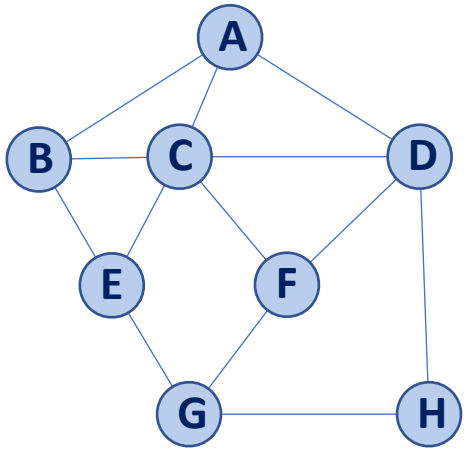


- Rooted
- Acyclic
-

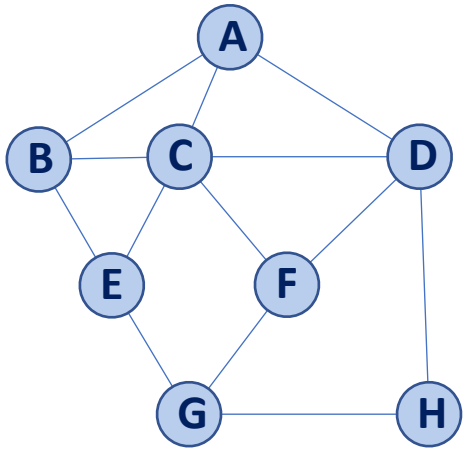


-
-
-

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