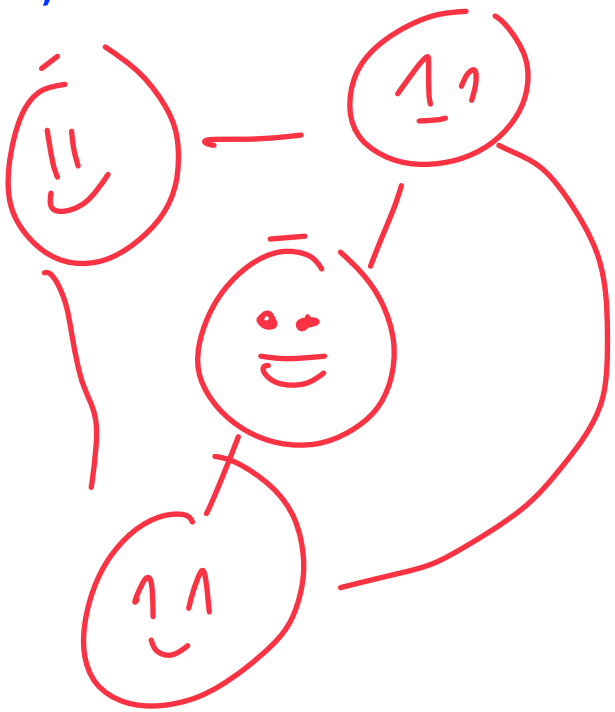
Algorithms and Data Structures for Data Science

Graph Implementations

CS 277

Brad Solomon

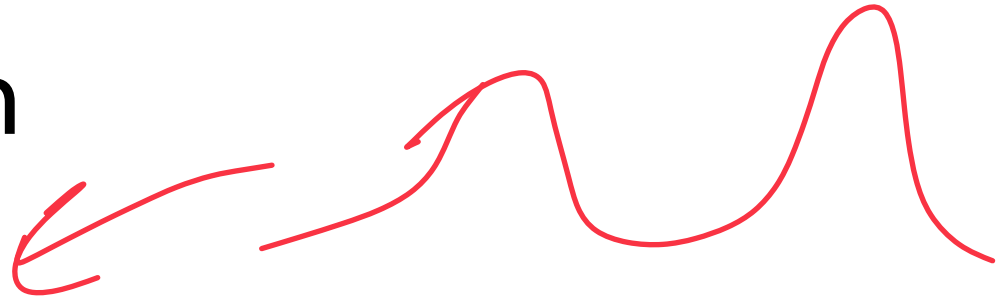
March 25, 2024



UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Exam 2 Reflection



Average: 72%

The programming question was the most difficult

I also consider the programming question to be extremely fair

Consider going to office hours to go over exam

retake exam

Learning Objectives

Practice implementing complex data structures (graphs)

Compare and contrast different implementations

Review Big O concepts in the context of graphs

Graph ADT

Constructor

Find: Need to be able to search for vertices, edges, and adjacency.

Insert: Need both a vertex and an edge insertion function

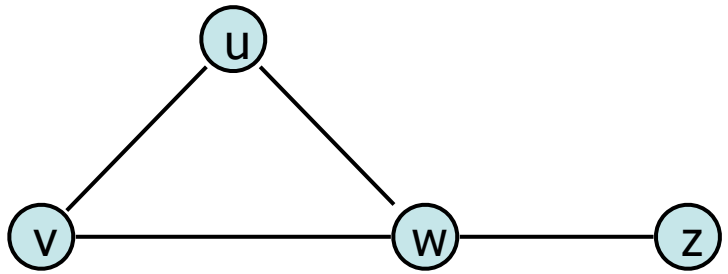
Remove: Need both a vertex and an edge removal function

Traversal: Need to be able to traversal a graph efficiently

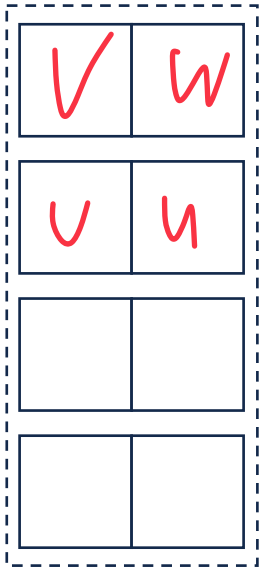
Graph Implementation: Edge List

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

The equivalent of an 'unordered' data structure



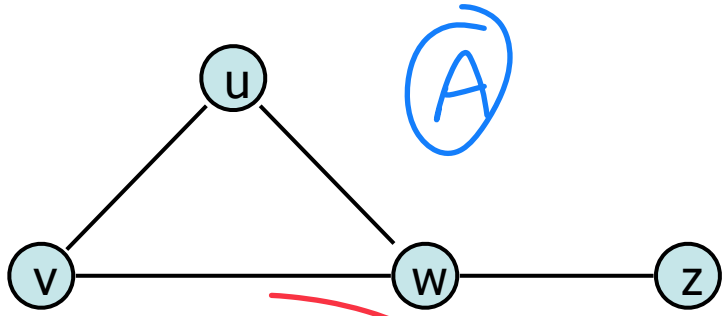
Vertex Storage:

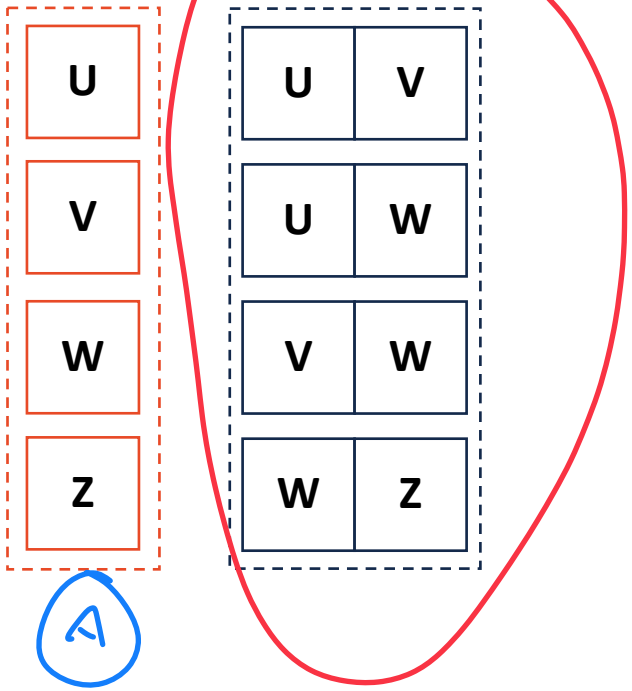


Edge Storage: (List of pairs of vertices)

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

The equivalent of an 'unordered' data structure





Vertex Storage:

None

A list of vertex labels

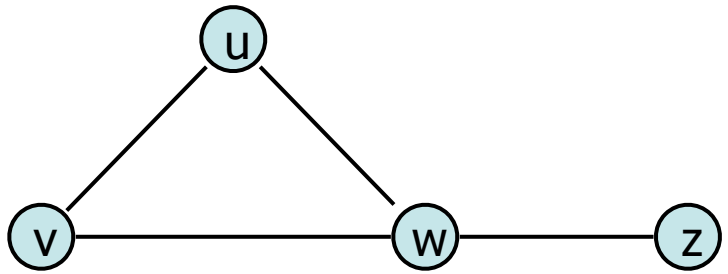
Edge Storage:

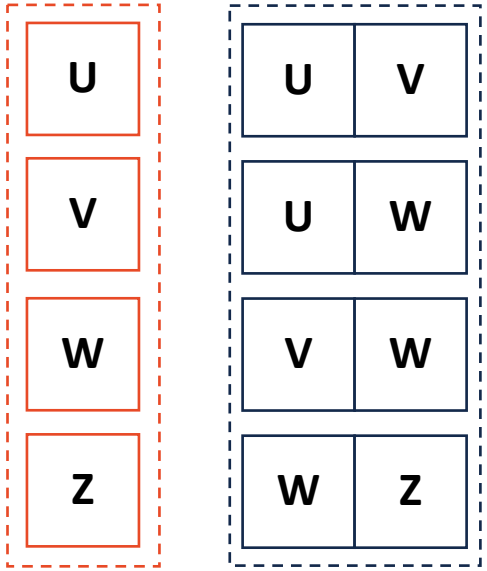
A list of paired vertex labels

Graph Implementation: Edge List

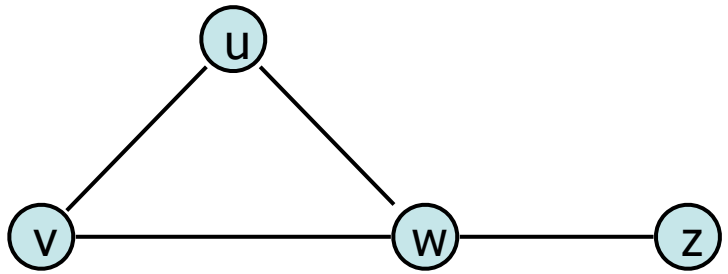
getVertices()

return vertex list





Graph Implementation: Edge List



U
V
W
Z

U	V
U	W
V	W
W	Z

getVertices()

Return vertex list

~~Loop through edge list and build list of vertices~~

getEdges(v)

self.Edge(w)

↳ loop through edge list

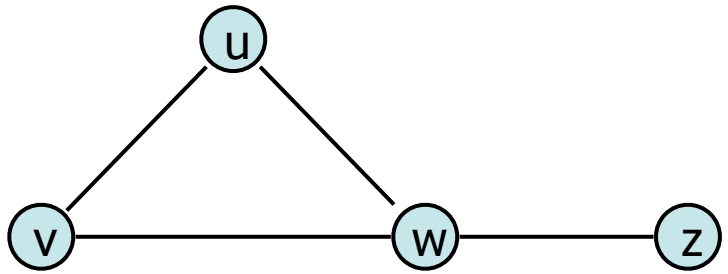
x = [(u,v), (u,w), (v,w), (w,z)]

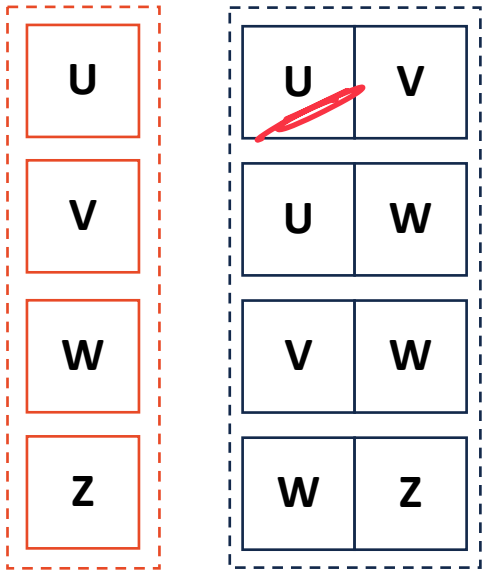
x[0] → (u,v)

↳ x[0][0] ≠ 'w' & x[0][1] ≠ 'w'

Out.append([u,w]) b/c x[1][1] == w

Graph Implementation: Edge List





getVertices()

Return vertex list

~~Loop through edge list and build list of vertices~~

getEdges(v)

Loop through edge list and build list of edges

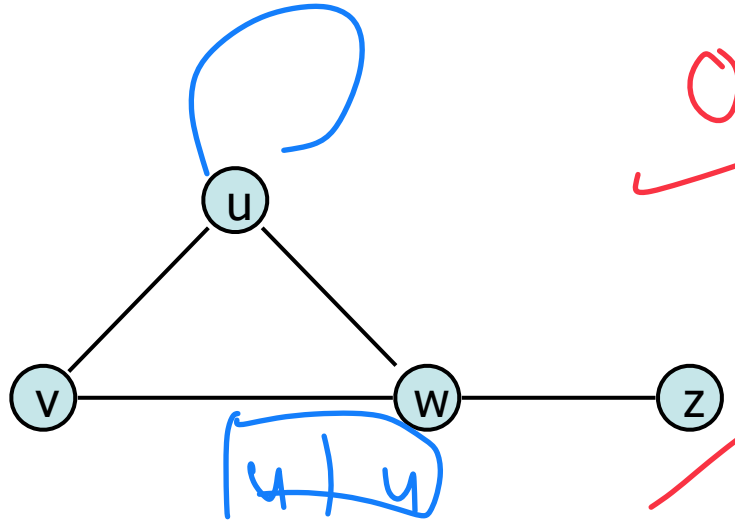
areAdjacent(u, v)

↳ getEdges(u) & getEdges(z)

if exists in both

Loop through EL and search for (u,z) or (z,u)

Graph Implementation: Edge List



U
V
W
Z

U	V
U	W
V	W
W	Z

getVertices()

Return vertex list

Loop through edge list and build list of vertices

getEdges(v)

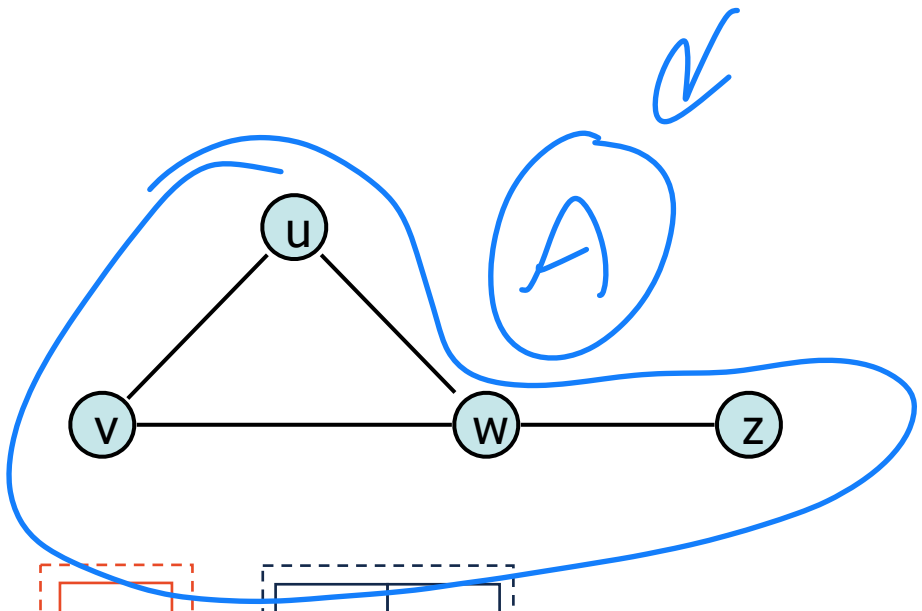
Loop through edge list and build list of edges

areAdjacent(u, v)

Loop through edge list and find (u, v) or (v, u)

Lets code this up!

Graph Implementation: Edge List



U
V
W
Z

A

U	V
U	W
V	W
W	Z

insertVertex(v)

↳ list.append(v)

↑ vertex list

insertEdge(u, v)

edge list.append((u, v))

if u or v not in vertex list

↳ update (add) to vertex list

insert

O(1)

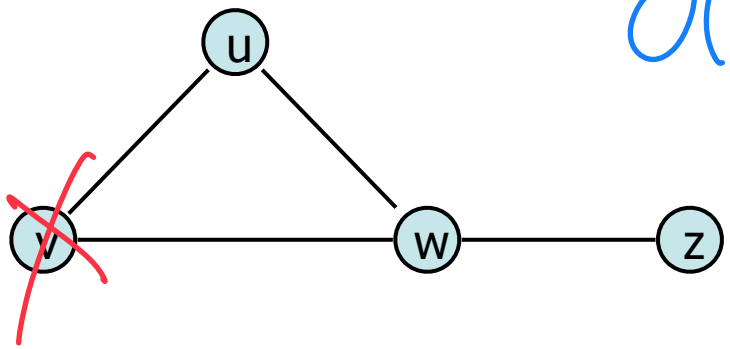
↑

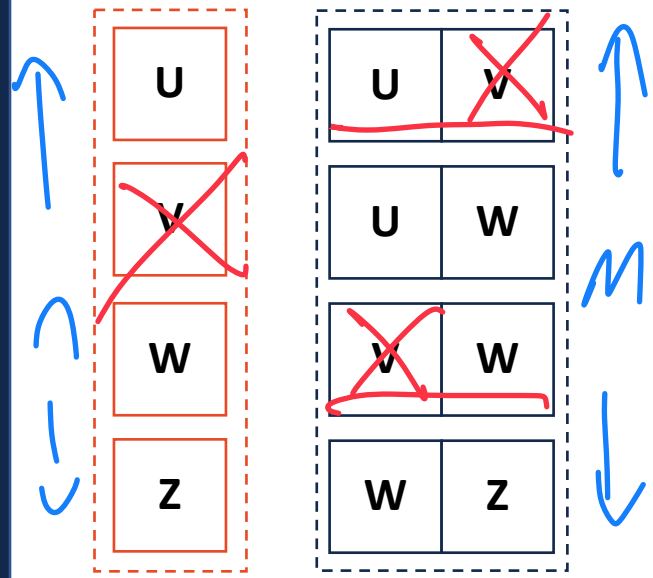
as good

O(1) + O(1)

O(1)

Graph Implementation: Edge List





removeVertex(v)

O(n + m)

↳ Search vertex list & remove O(n)
(List Remove)

↳ Search edge list & remove

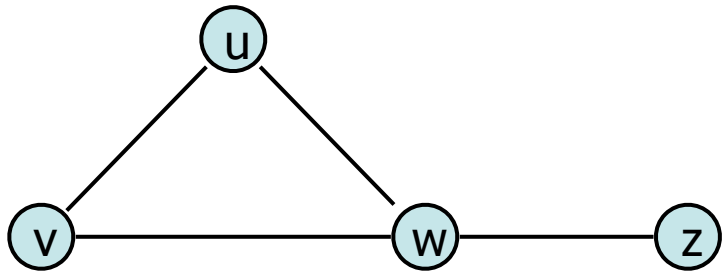
removeEdge(u, v)

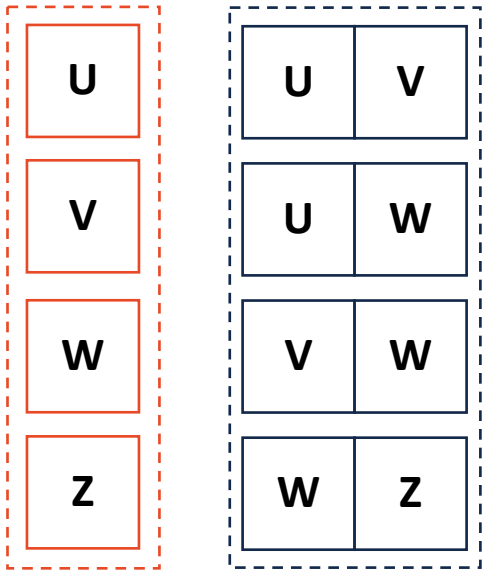
↳ Search for loop through edge List)

O(m) (List Remove)*

* maybe multiple times

Graph Implementation: Edge List





Pros: Insert very efficient

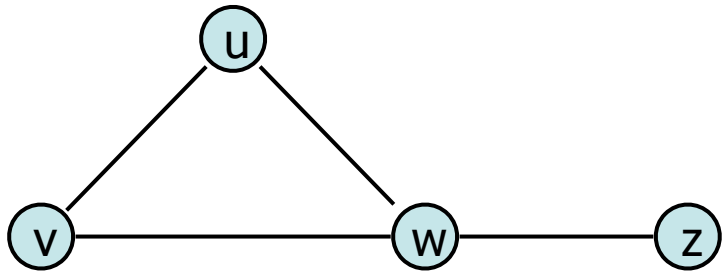
Easy to implement

Cons:

↳ Doesn't track duplicates (our version doesn't)

↳ Find / Remove is slow

Graph Implementation: Adjacency Matrix



Vertex Storage:

↳ as a dictionary

key – Vertex label

value

Draw arrow from label to row/col

Edge Storage:

↳ Store edges as a matrix (of bools)

or ints

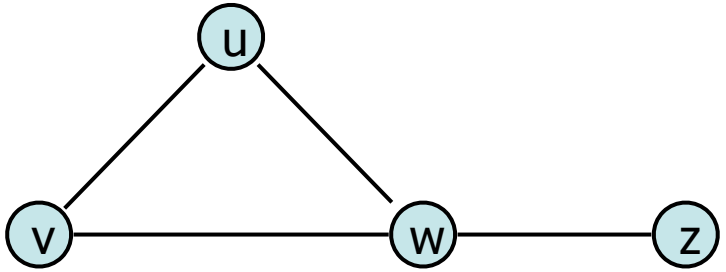
Vertex Dictionary

u	0
v	1
w	2
z	3

Keys

	u	v	w	z
u	0	T	T	F
v		0	T	F
w	T	T	0	T
z	F	F	T	0

Graph Implementation: Adjacency Matrix



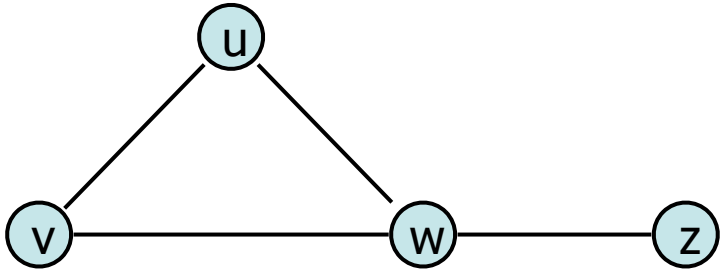
Vertex Storage:

U	0
V	1
W	2
Z	3

Edge Storage:

	u	v	w	z
u	0	1	1	0
v	1	0	1	0
w	1	1	0	1
z	0	0	1	0

Graph Implementation: Adjacency Matrix



getVertices():

↳ Dictionary.keys()

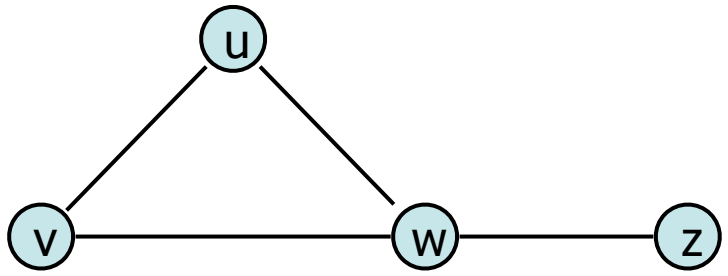
unordered

U	0
V	1
W	2
Z	3

	u	v	w	z
u	0	1	1	0
v	1	0	1	0
w	1	1	0	1
z	0	0	1	0

Graph Implementation: Adjacency Matrix

O(n)



u	0
v	1
w	2
z	3

	u	v	w	z
u	0	1	1	0
v	1	0	1	0
w	1	1	0	1
z	0	0	1	0

"v"

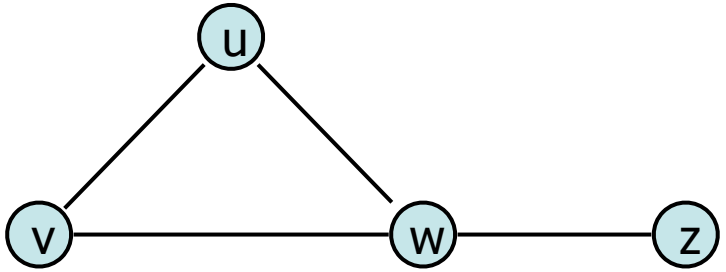
getEdges(v):

1) Look up value for "v" in dictionary
   (that value is index in matrix)
2) Look up row or col in matrix
3) Return list of edges where (u, v) = 1
   x[u][v] = 2

x[i] or x[i]

x[0][i], x[1][i], ...

Graph Implementation: Adjacency Matrix

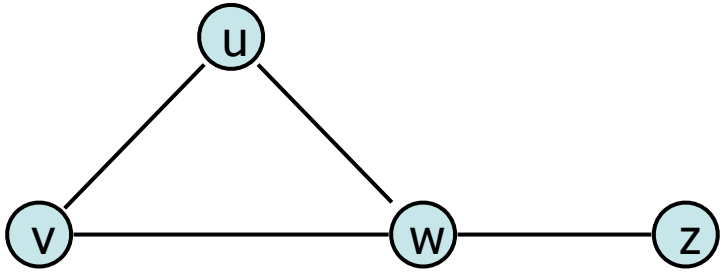


areAdjacent(u, v):

U	0
V	1
W	2
Z	3

	u	v	w	z
u	0	1	1	0
v	1	0	1	0
w	1	1	0	1
z	0	0	1	0

Graph Implementation: Adjacency Matrix

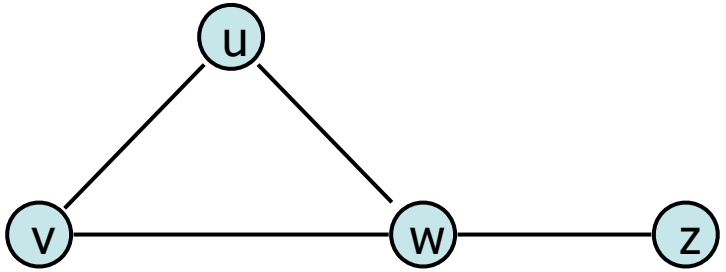


insertVertex(v):

U	0
V	1
W	2
Z	3

	u	v	w	z
u	0	1	1	0
v	1	0	1	0
w	1	1	0	1
z	0	0	1	0

Graph Implementation: Adjacency Matrix

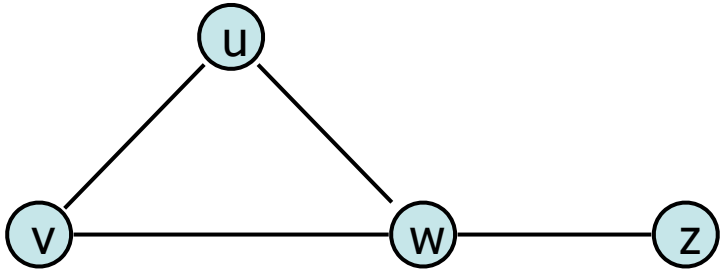


insertEdge(u, v):

U	0
V	1
W	2
Z	3

	u	v	w	z
u	0	1	1	0
v	1	0	1	0
w	1	1	0	1
z	0	0	1	0

Graph Implementation: Adjacency Matrix



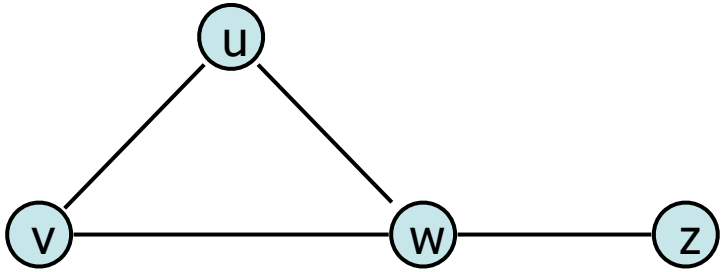
U	0
V	1
W	2
Z	3

	u	v	w	z
u	0	1	1	0
v	1	0	1	0
w	1	1	0	1
z	0	0	1	0

removeVertex(v):

removeEdge(u, v):

Graph Implementation: Adjacency Matrix



U	0
V	1
W	2
Z	3

	u	v	w	z
u	0	1	1	0
v	1	0	1	0
w	1	1	0	1
z	0	0	1	0

Pros:

Cons:

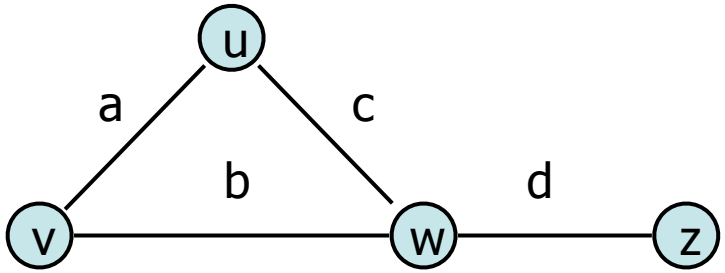
Graph Implementations

We want something…

Faster than an edge list

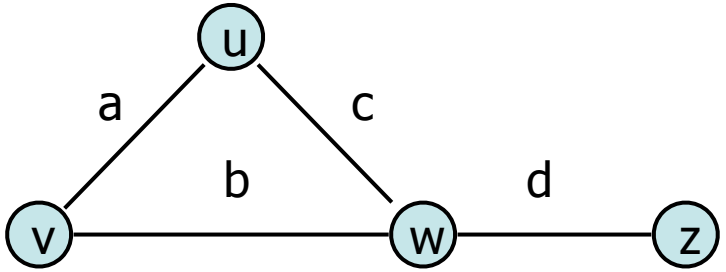
Less space than an adjacency matrix

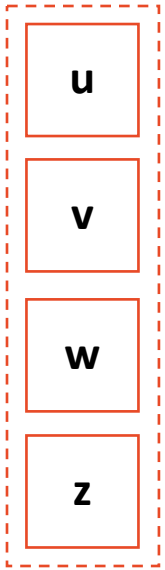
Particularly good at finding adjacent elements

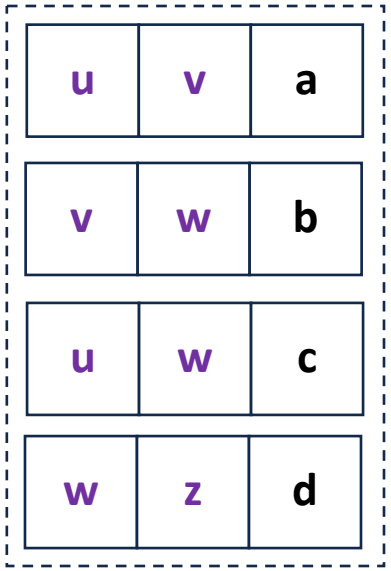


Graph Implementation: Edge List + ?

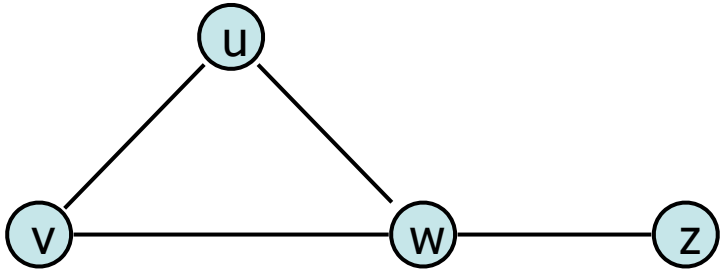
|V|= n,|E|= m



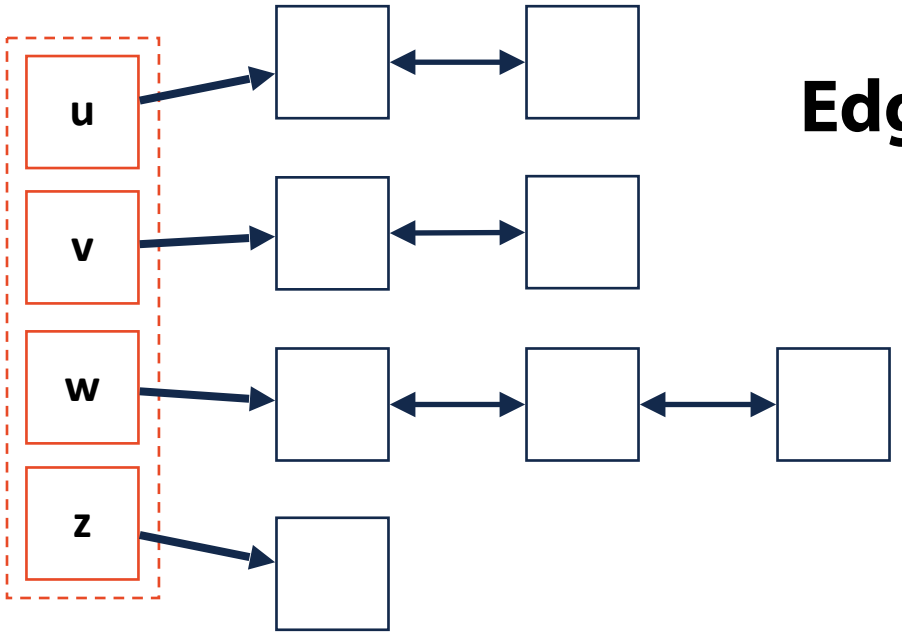




Adjacency List

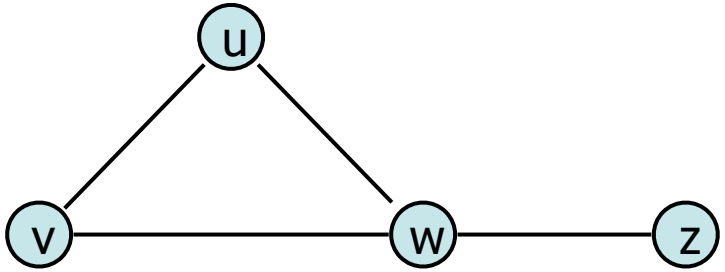


Vertex Storage:

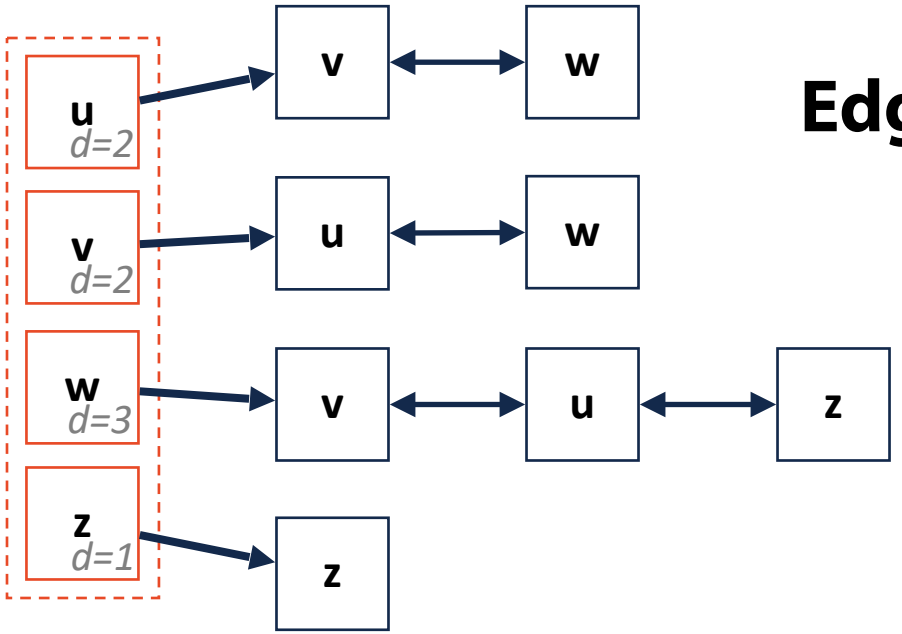


Edge Storage:

Adjacency List

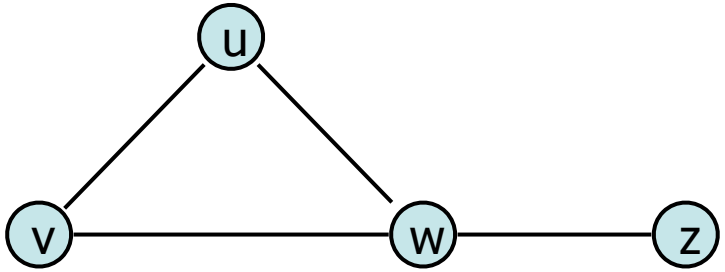


Vertex Storage:

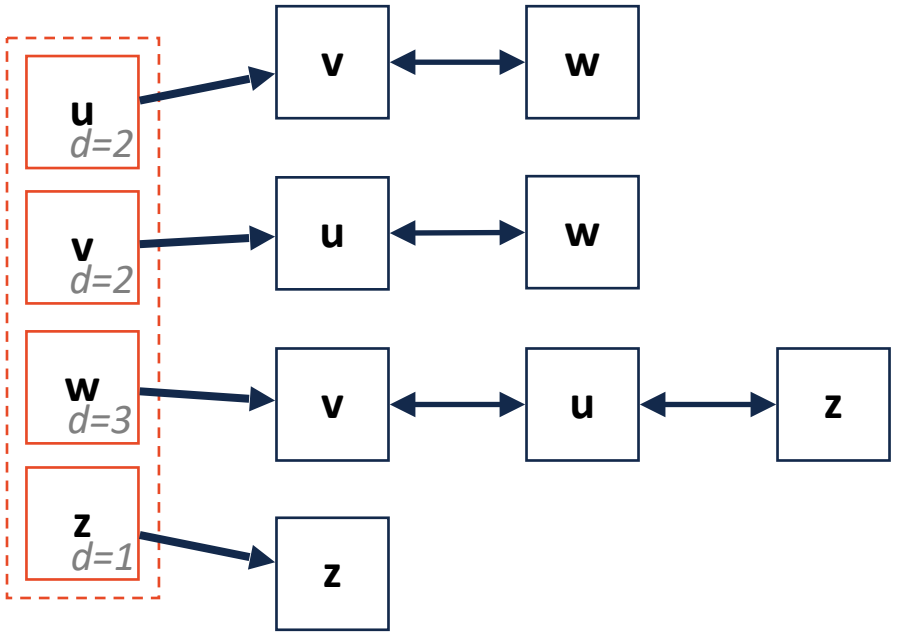


Edge Storage:

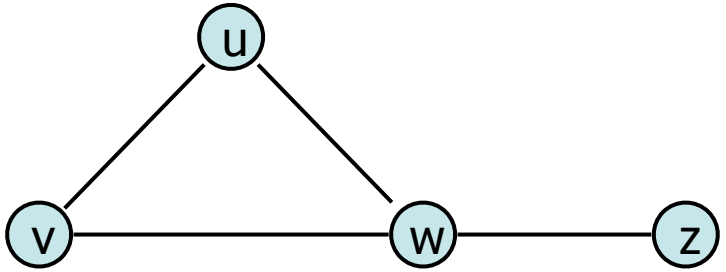
Adjacency List



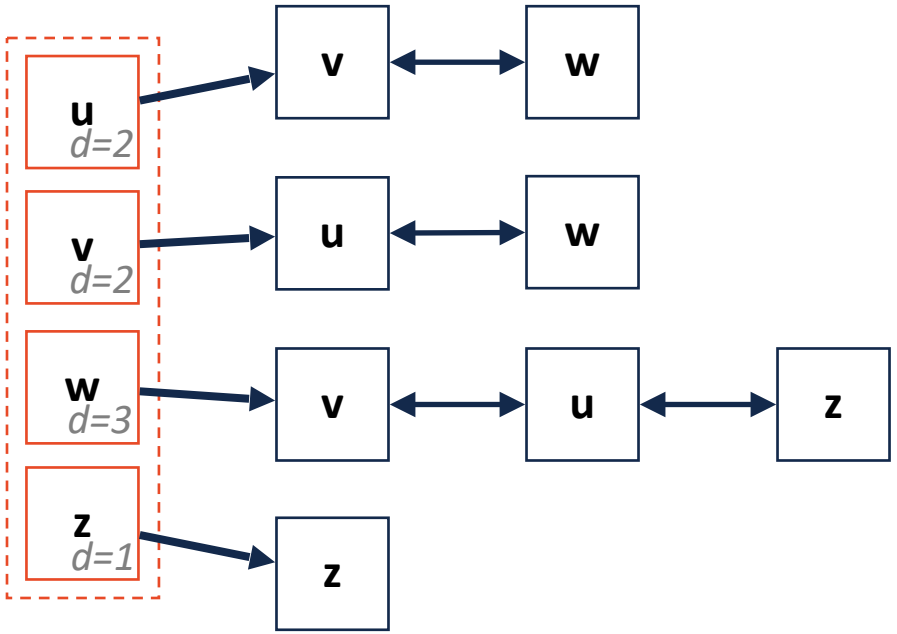
getVertices():



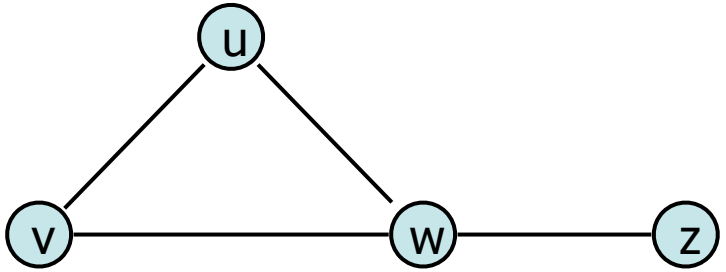
Adjacency List



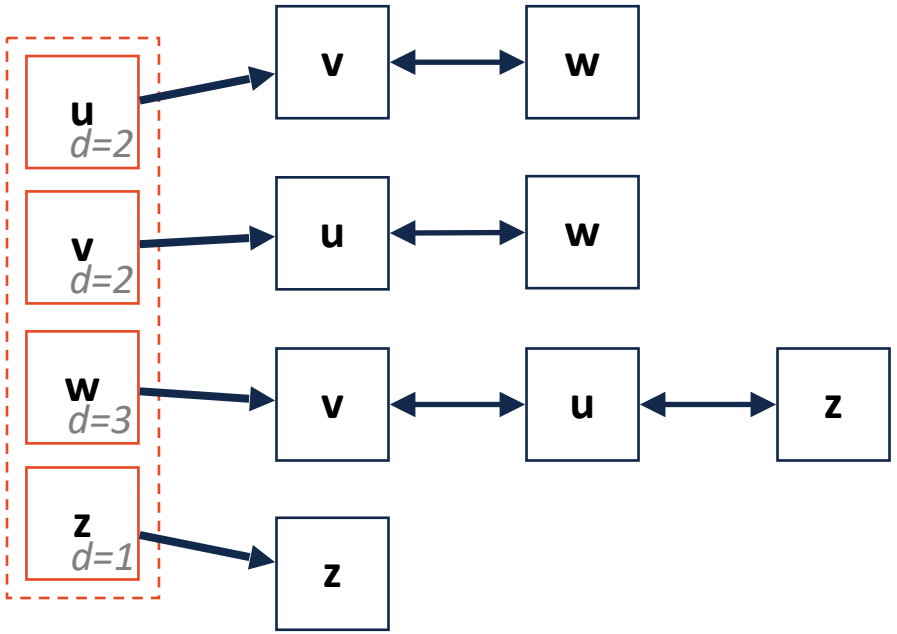
getEdges(v):



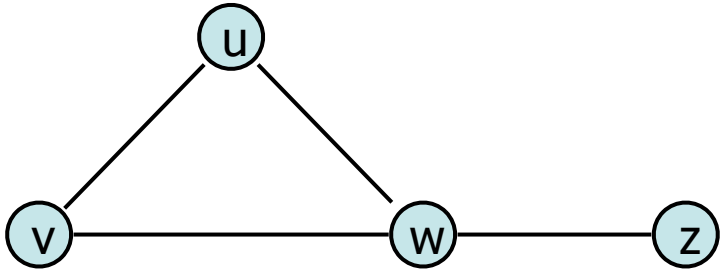
Adjacency List



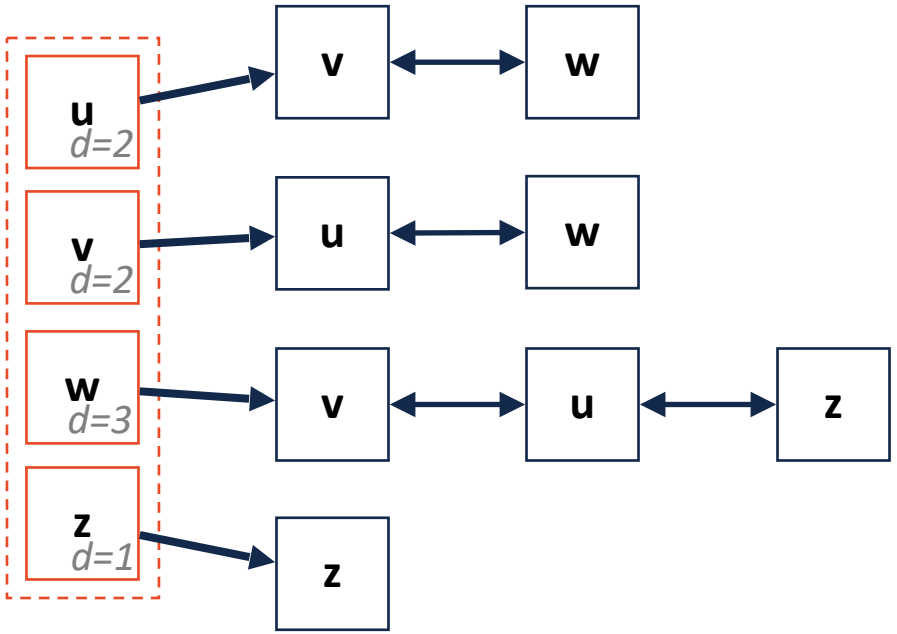
areAdjacent(u, v):



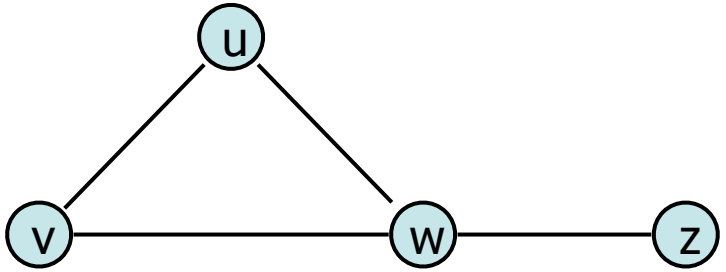
Adjacency List



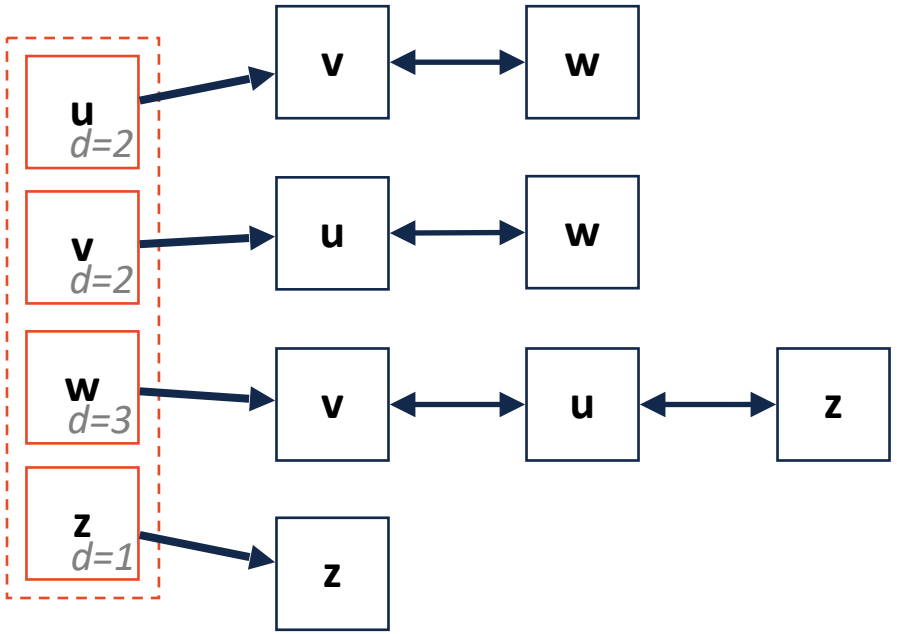
insertVertex(v):



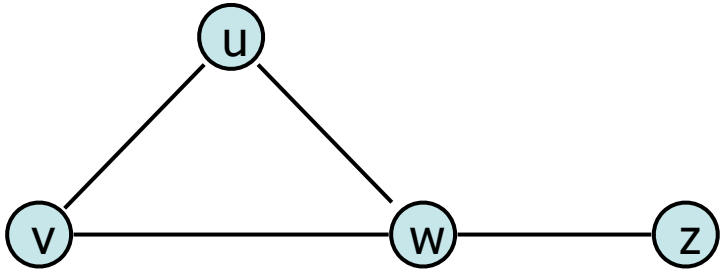
Adjacency List



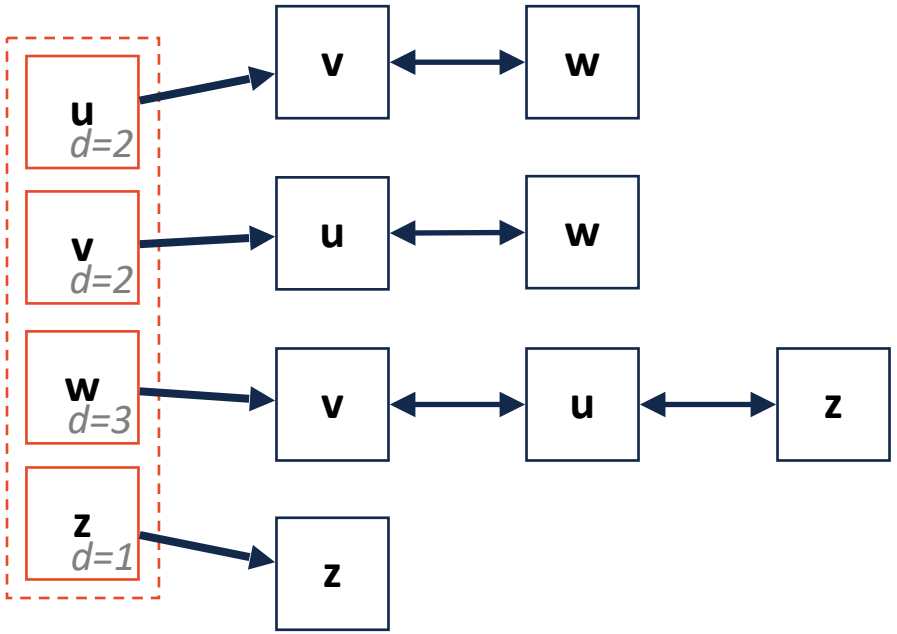
removeVertex(v):



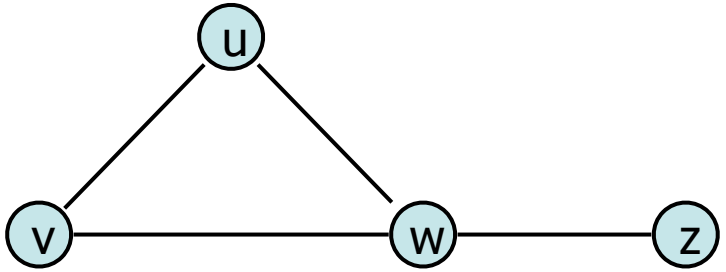
Adjacency List



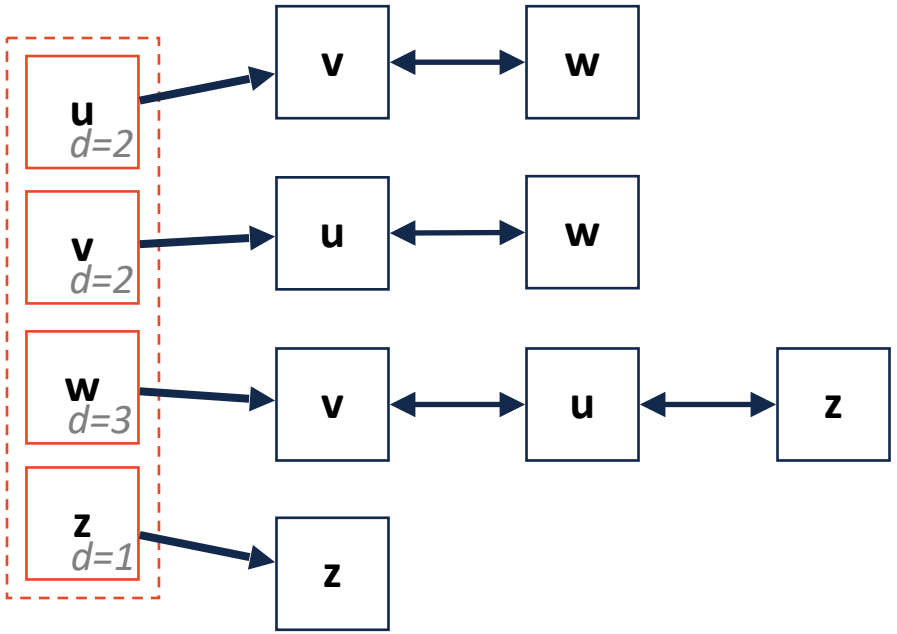
insertEdge(u, v):



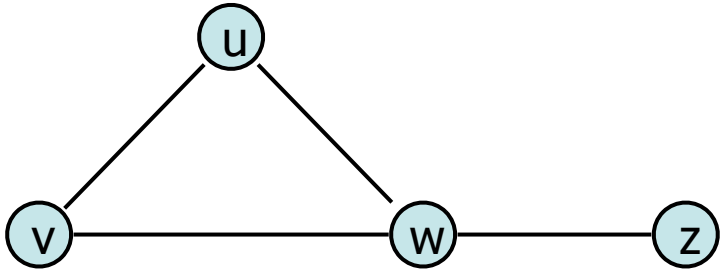
Adjacency List

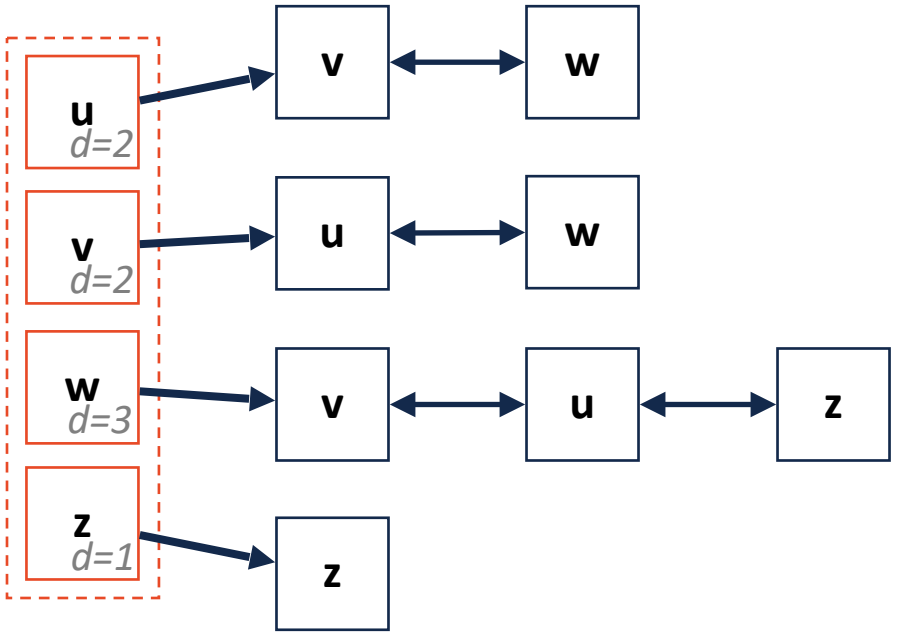


removeEdge(u, v):



Adjacency List





Pros:

Cons:

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

Expressed as O(f)	Edge List	Adjacency Matrix	Adjacency List
Space			
insertVertex(v)			
removeVertex(v)			
insertEdge(u, v)			
removeEdge(u, v)			
getEdges(v)			
areAdjacent(u, v)			