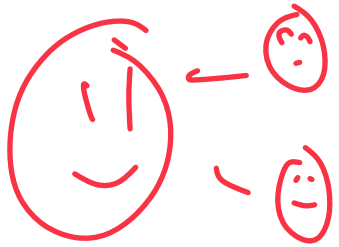
Algorithms and Data Structures for Data Science

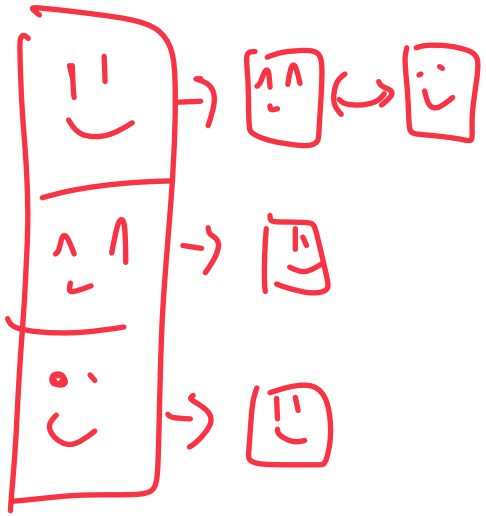
Graph Implementations 2



CS 277

Brad Solomon

March 27, 2024



UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Learning Objectives

Practice implementing complex data structures (graphs)

Compare and contrast different implementations

Review Big O concepts in the context of graphs

Graph ADT

Find

getVertices() — return the list of vertices in a graph

getEdges(v) — return the list of edges that touch the vertex v

areAdjacent(u, v) — returns a bool based on if an edge from u to v exists

Insert

insertVertex(v) — adds a vertex to the graph

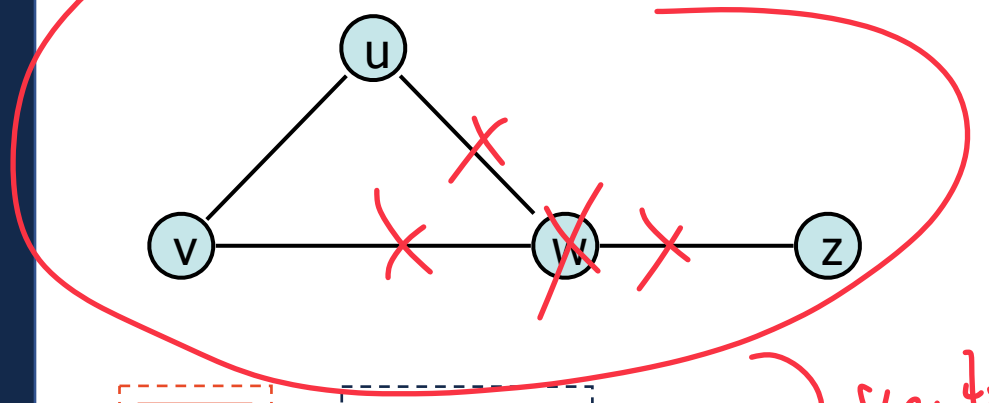
insertEdge(u, v) — adds an edge to the graph

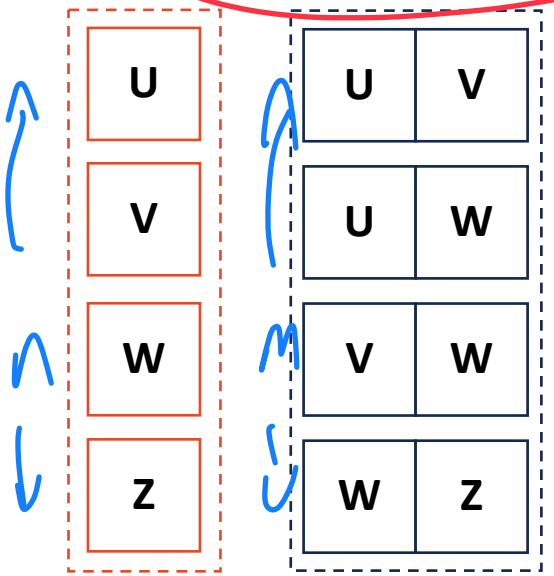
Remove

removeVertex(v) — removes a vertex from the graph

removeEdge(u, v) — removes an edge from the graph

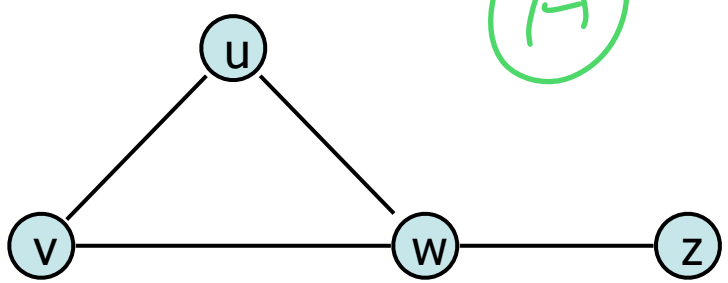
Graph Implementation: Edge List

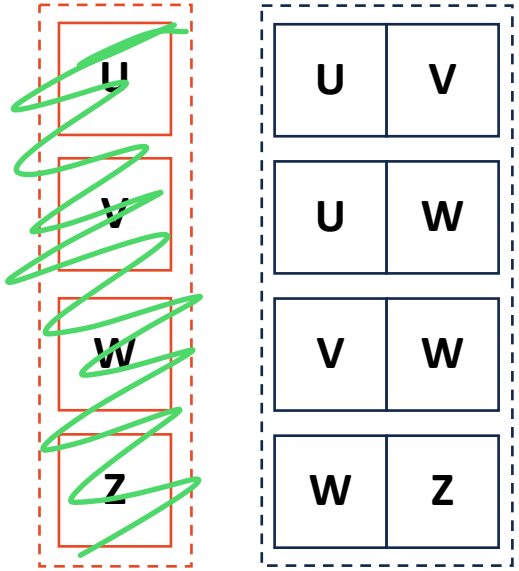




Expressed as O(f)	Edge List
Space	n+m
insertVertex(v)	O(1)*
removeVertex(v)	O(n+m)
insertEdge(u, v)	O(1)*
removeEdge(u, v)	O(m)
getEdges(v)	O(m)
areAdjacent(u, v)	O(m)

Graph Implementation: Edge List

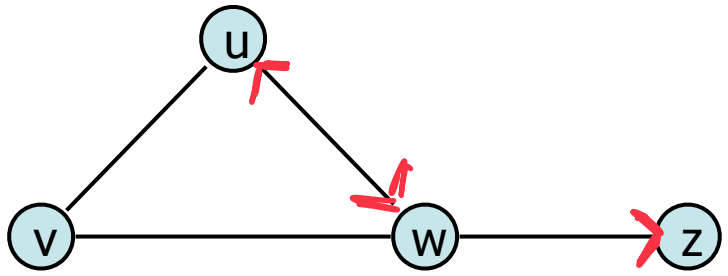




All vertices are in edge list

Expressed as O(f)	Edge List
Space	n+m
insertVertex(v)	1*
removeVertex(v)	n+m
insertEdge(u, v)	1
removeEdge(u, v)	m
getEdges(v)	m
areAdjacent(u, v)	m

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	0	0

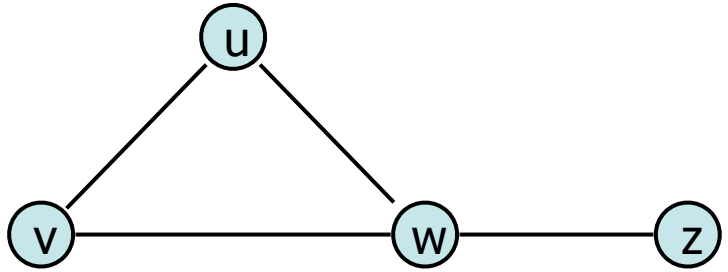
Vertex Storage:

A dictionary (key = label, value = index)

Edge Storage:

A 2D matrix storing edge existence

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

getVertices():

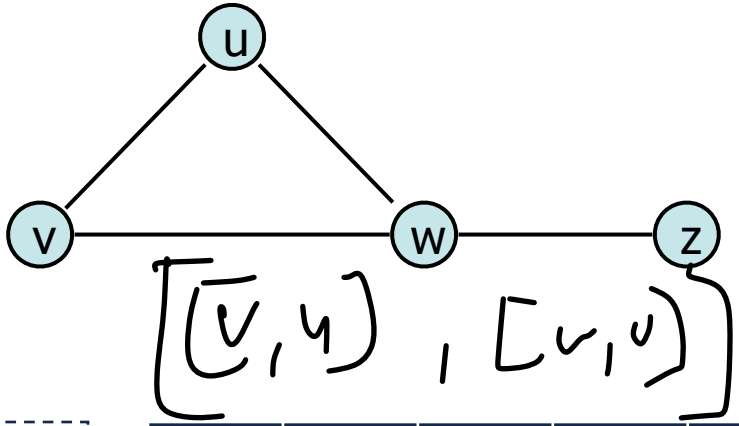
```
return self.vertices.keys()
```

Dictionary.keys() gets list of keys

```
if x in Dictionary.keys()
[ ]
```

Graph Implementation: Adjacency Matrix

O(n)



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

getEdges(v):

return list of edges

[v1, v4]

[(v1, v4), (v3, v4), ...]

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

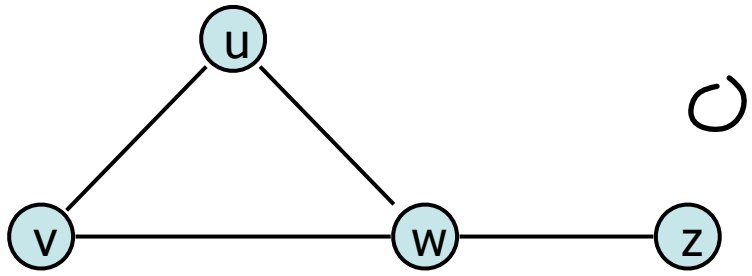
u	0
v	1
w	2
z	3

1) Look up label 'v' in vertex dictionary
   ↳ self.vertices[v] O(1)*

2) Look up row or col in matrix O(1)*
   ↳ self.edges[i]

3) Loop through all keys, looking up index, call it v2
   ↳ The edge is [v, v2]

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

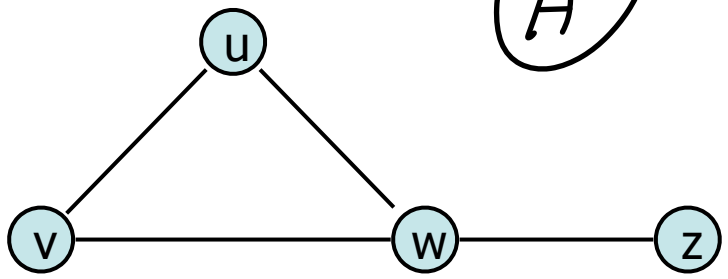
O(1)

O(1)*

areAdjacent(u, v):

1) Look up u in vertex dictionary -> i1
2) Look up v in vertex dictionary -> i2
3) Look up matrix[i1][i2]
   - if 1, return True
   - if 0, return False

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

insertVertex(v):

1) Add vertex to vertex dictionary
   - key / value pair

table dictionary can get length of keys we know value

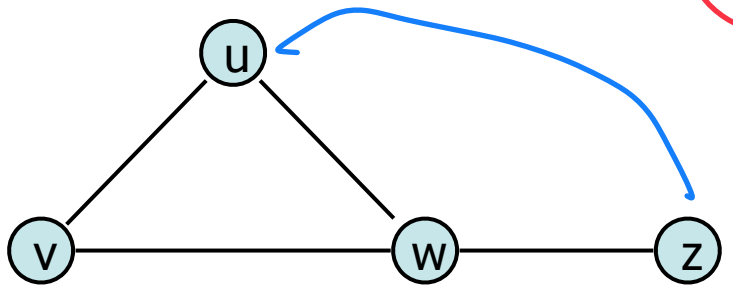
2) Increase matrix size
   - Append 0 to every list
   - Add a new n+1 list to matrix

n items + n items → list insert O(1)* → O(n)

O(n)

O(1)*

Graph Implementation: Adjacency Matrix



u	0
v	1
w	2
z	3

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

insertEdge(u, v):

1) Look up u (i1)
2) Look up v (i2)
3) change value M[i1][i2] = 1, M[i2][i1] = 1

O(1) O(1) O(1)

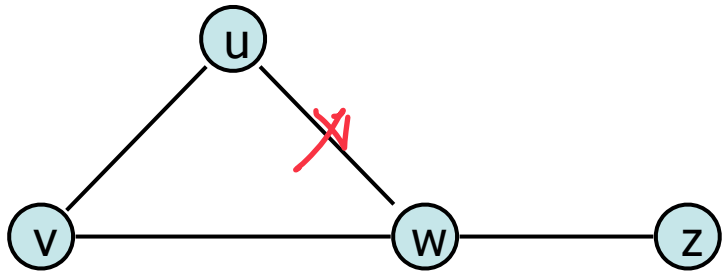
O(1) if no new vertex

if new vertex

1) insertVertex(u) O(n)
2) insertVertex(v) O(n)
3) insertEdge() O(1)

O(n)

Graph Implementation: Adjacency Matrix



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

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

removeVertex(v):

↳ Messy! List removal × N

removeEdge(u, v):

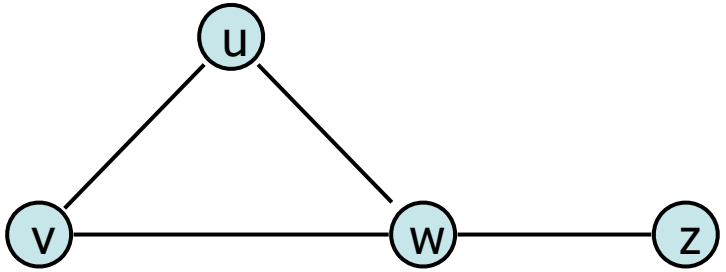
↳ set M[u][v] = 0

O(1)

M[v][u] = 0

1D version: X[0] = 3

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: 1) Lookup is easy ← $d(1)_2$

2) Very good at chasing edges

Cons: Adding or removing vertices is hard

is very inefficient

Its also very large

N^2

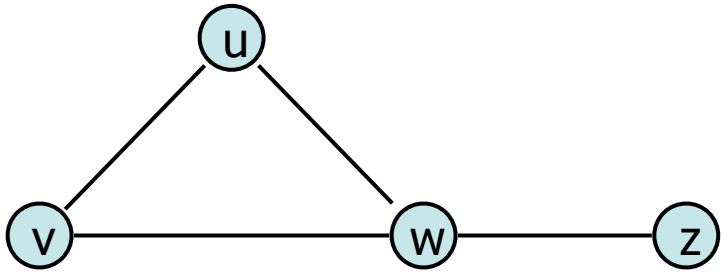
Graph Implementations

We want something…

Faster than an edge list

Less space than an adjacency matrix

Particularly good at finding adjacent elements



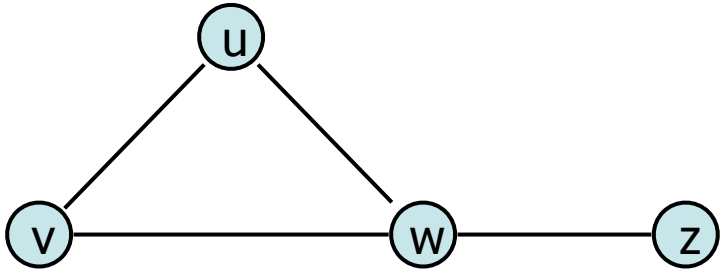
all

+

better adding vertices to ~~any~~ graph

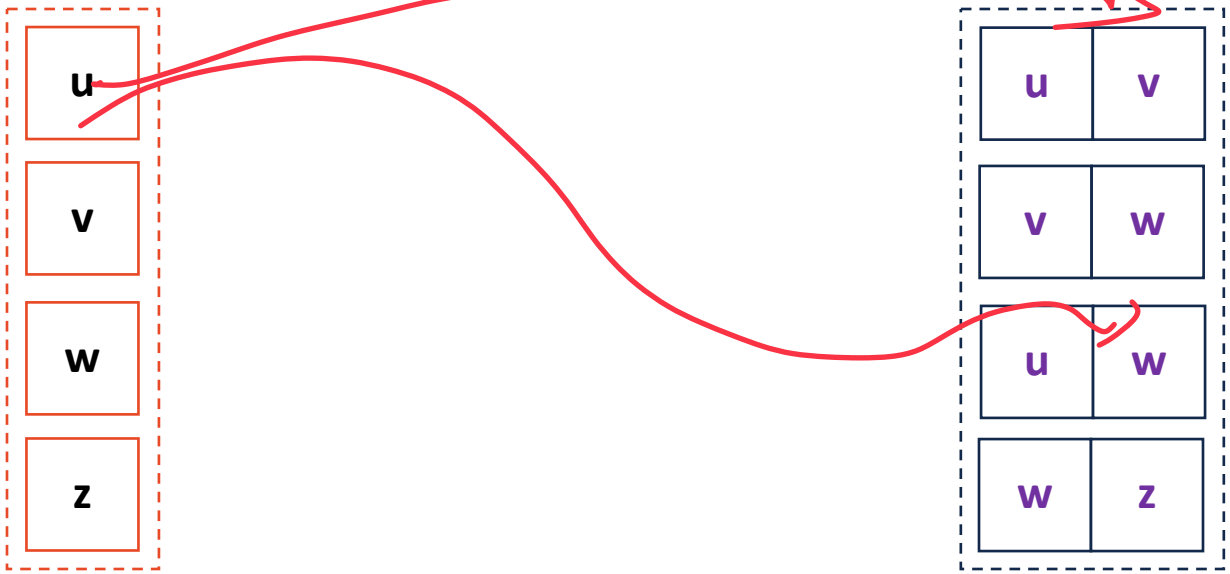
Graph Implementation: Edge List + ?

|V|= n, |E|= m



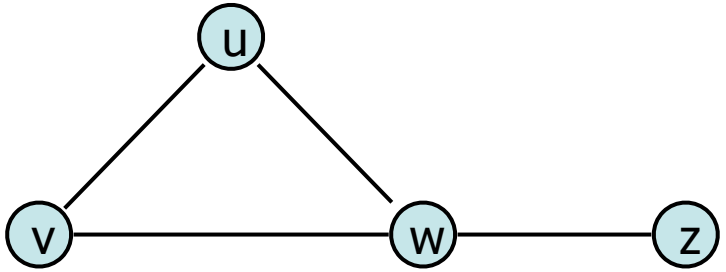
Edge List

vertex

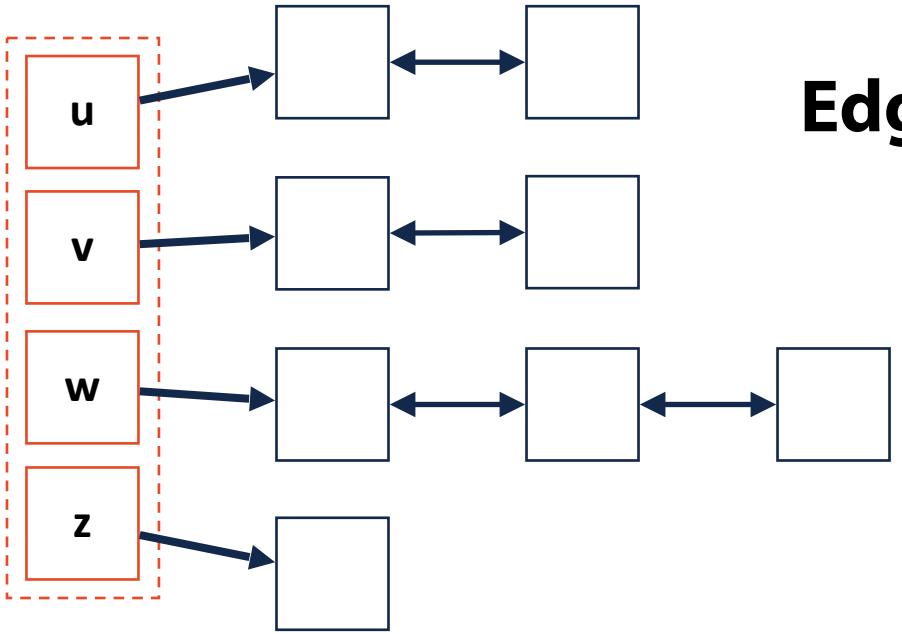


why cant vertex point to edges it has?

Adjacency List

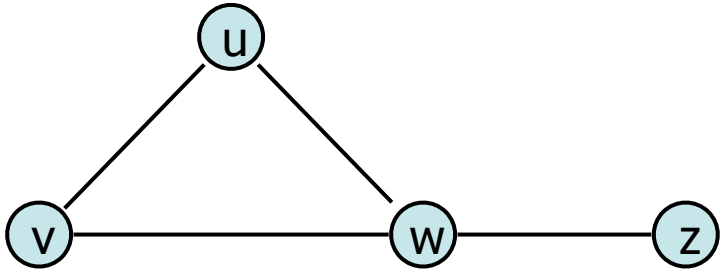


Vertex Storage:

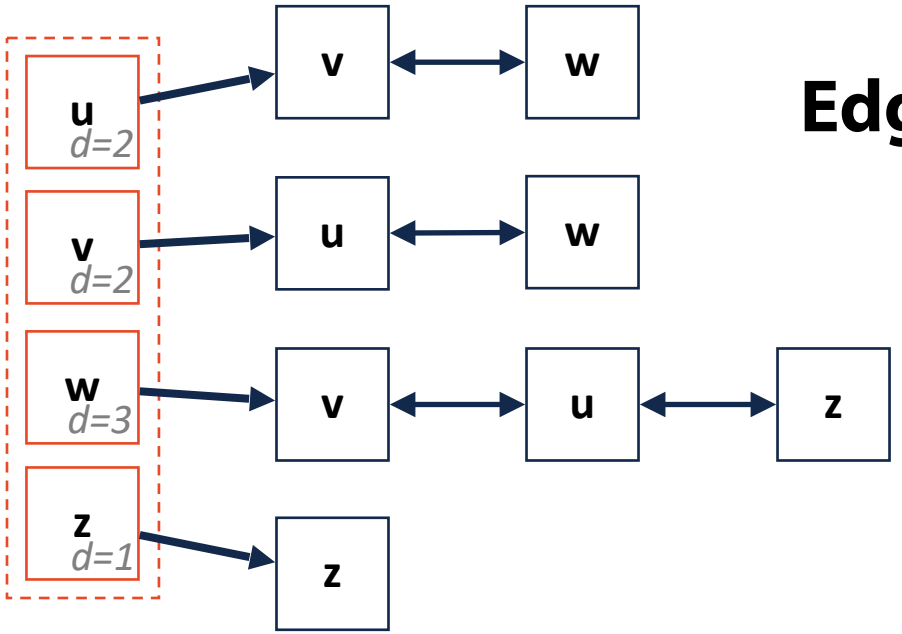


Edge Storage:

Adjacency List

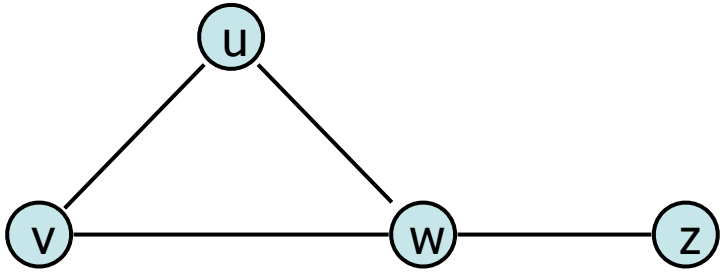


Vertex Storage:

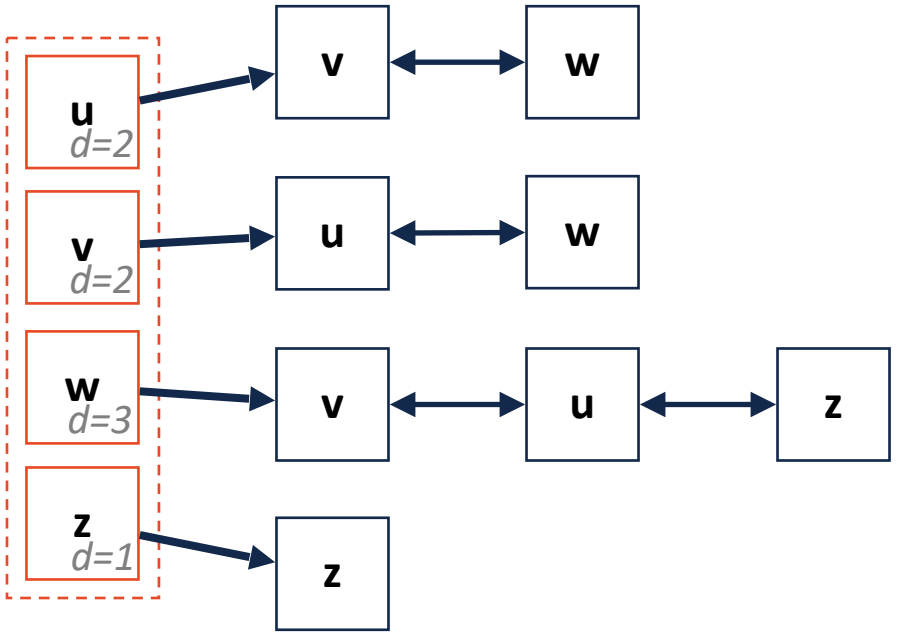


Edge Storage:

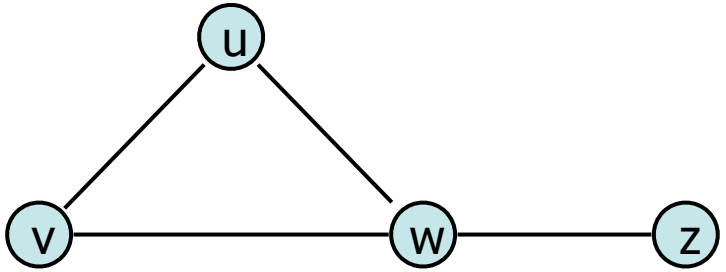
Adjacency List



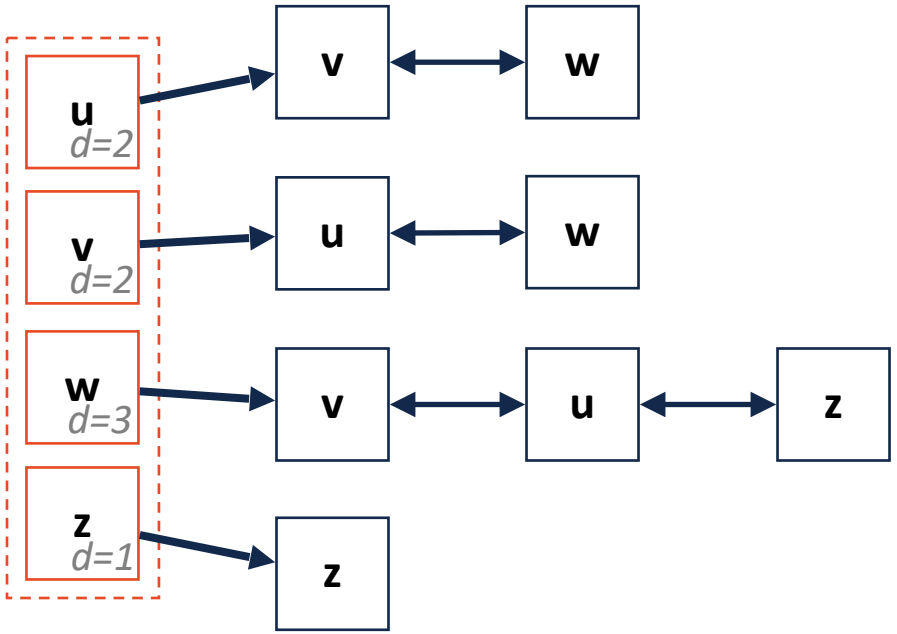
getVertices():



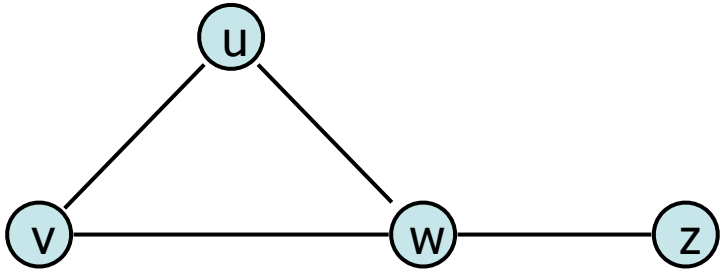
Adjacency List



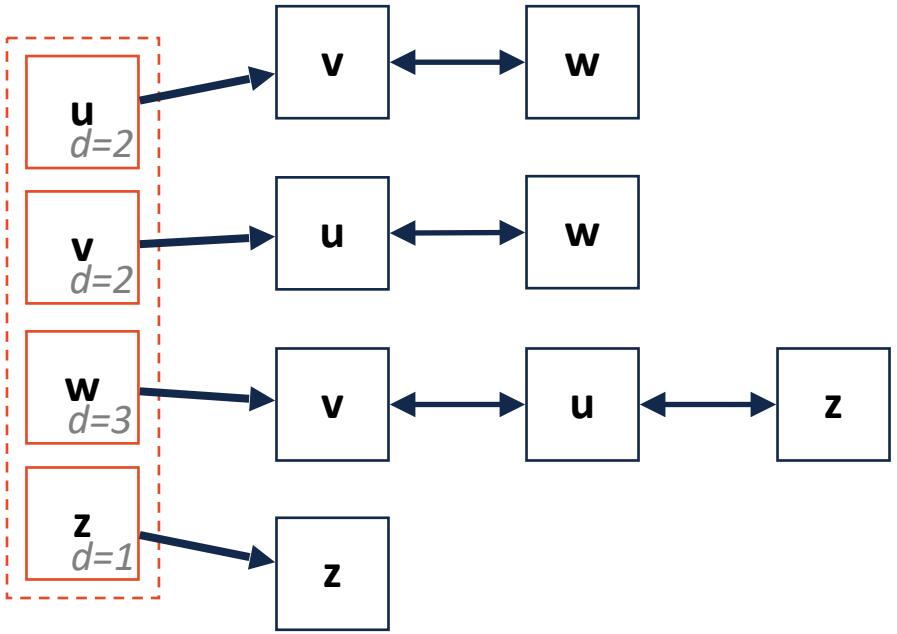
getEdges(v):



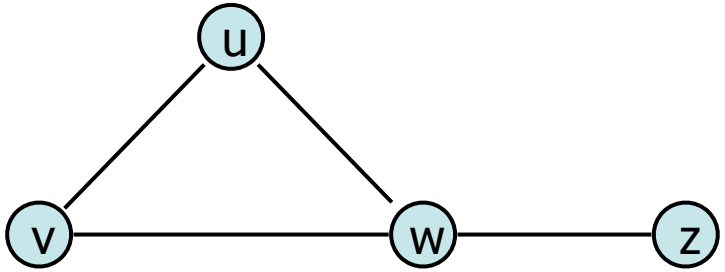
Adjacency List



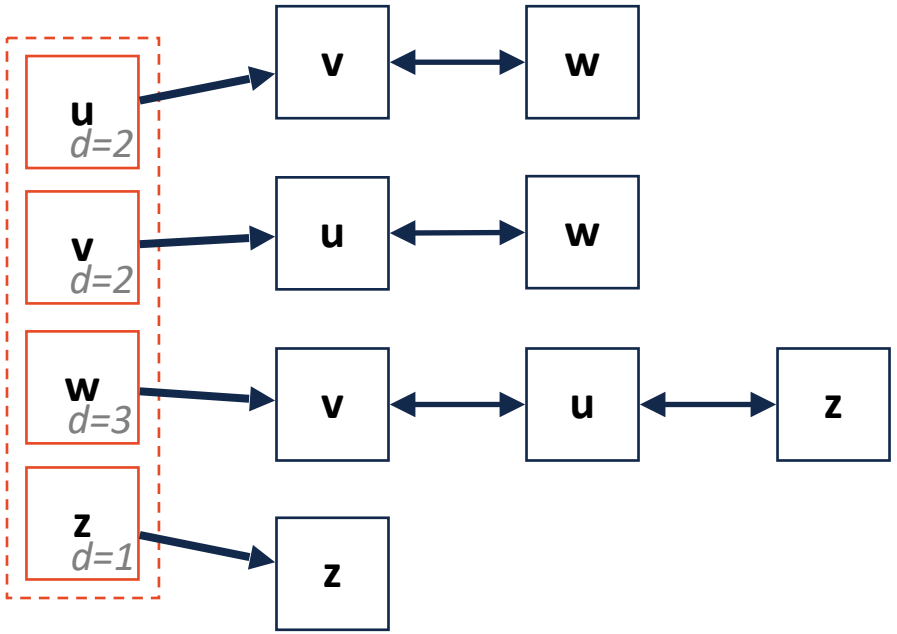
areAdjacent(u, v):



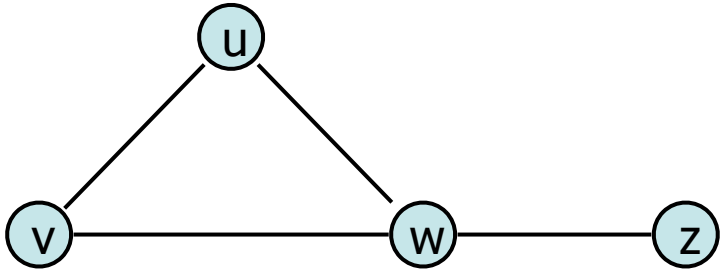
Adjacency List



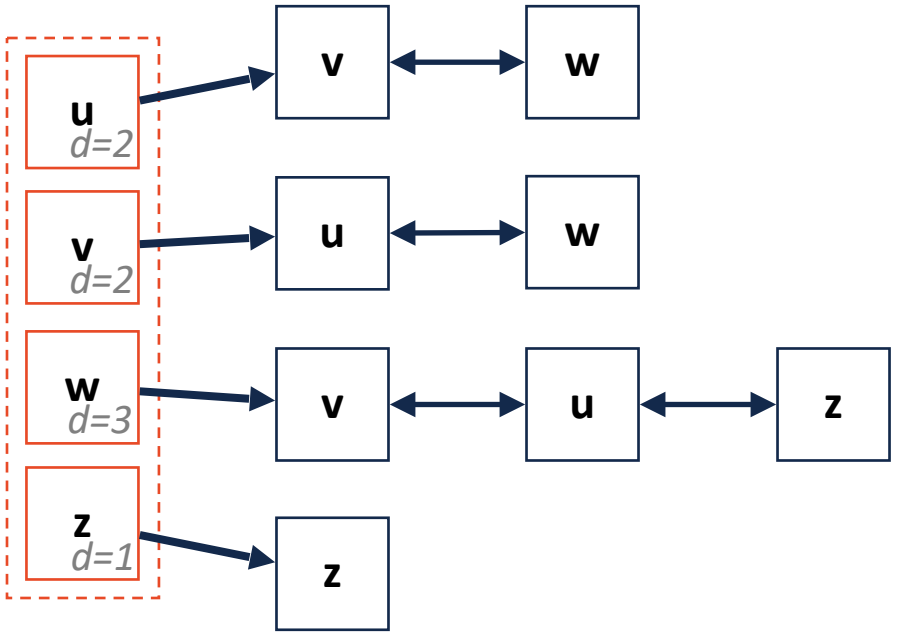
insertVertex(v):



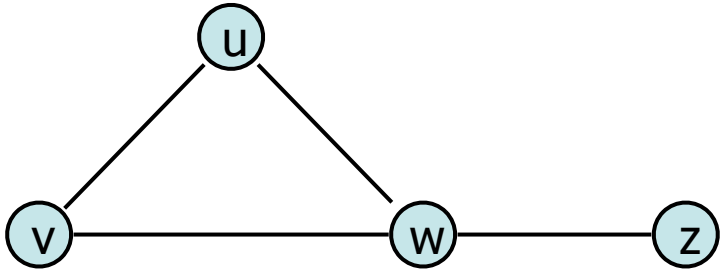
Adjacency List



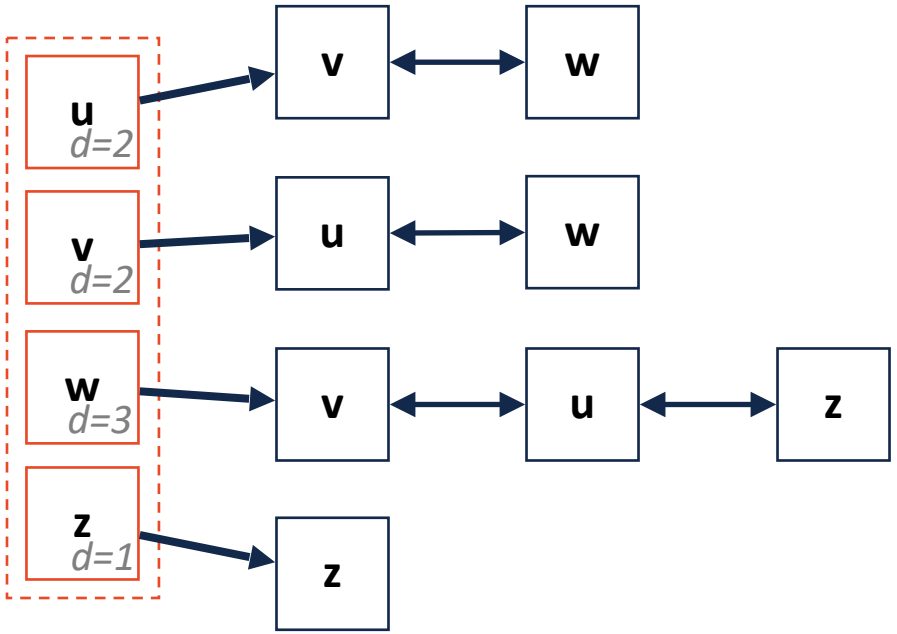
removeVertex(v):



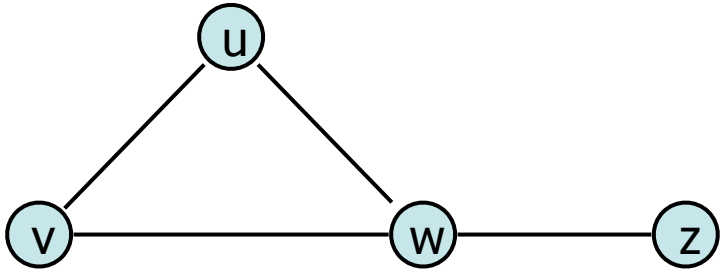
Adjacency List



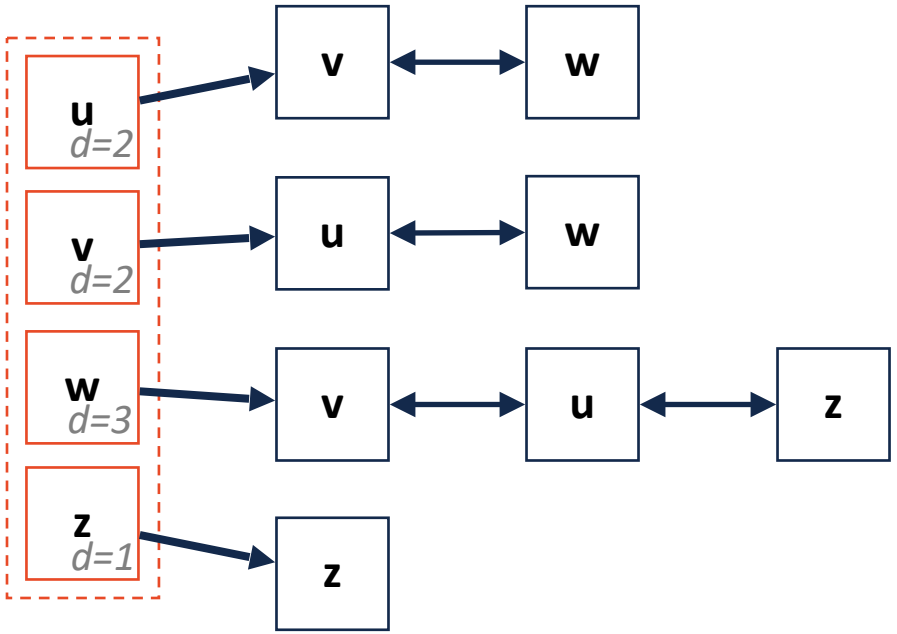
insertEdge(u, v):



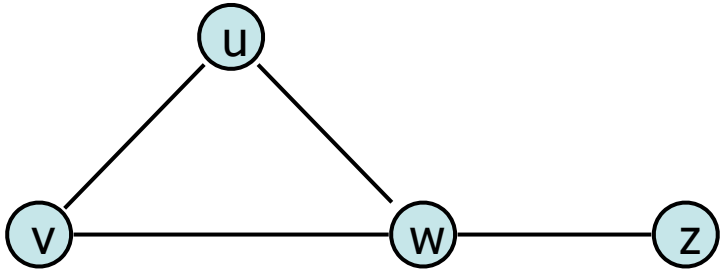
Adjacency List

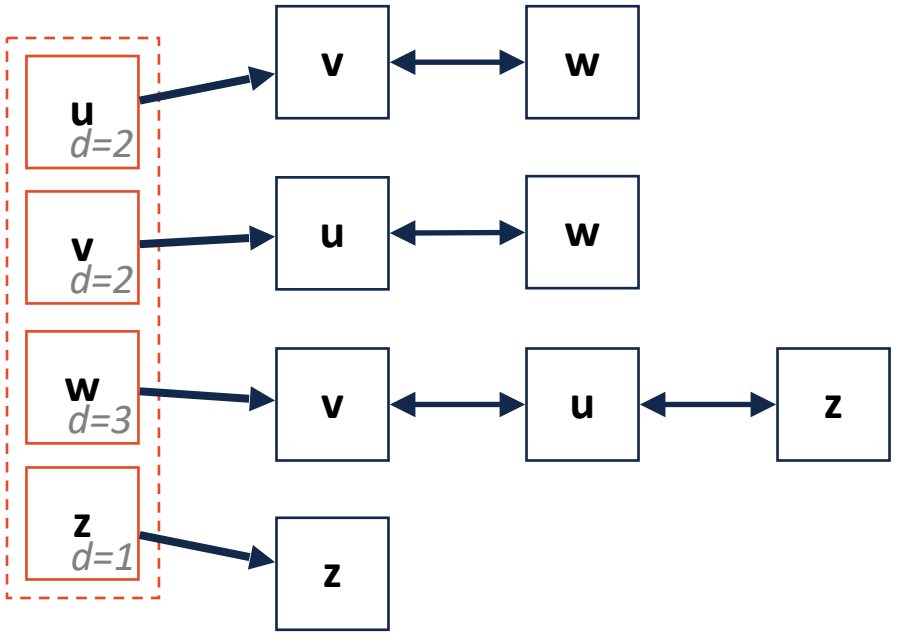


removeEdge(u, v):



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)			