Algorithms and Data Structures for Data Science
Graph Implementations 3

CS 277
Brad Solomon

April 1, 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

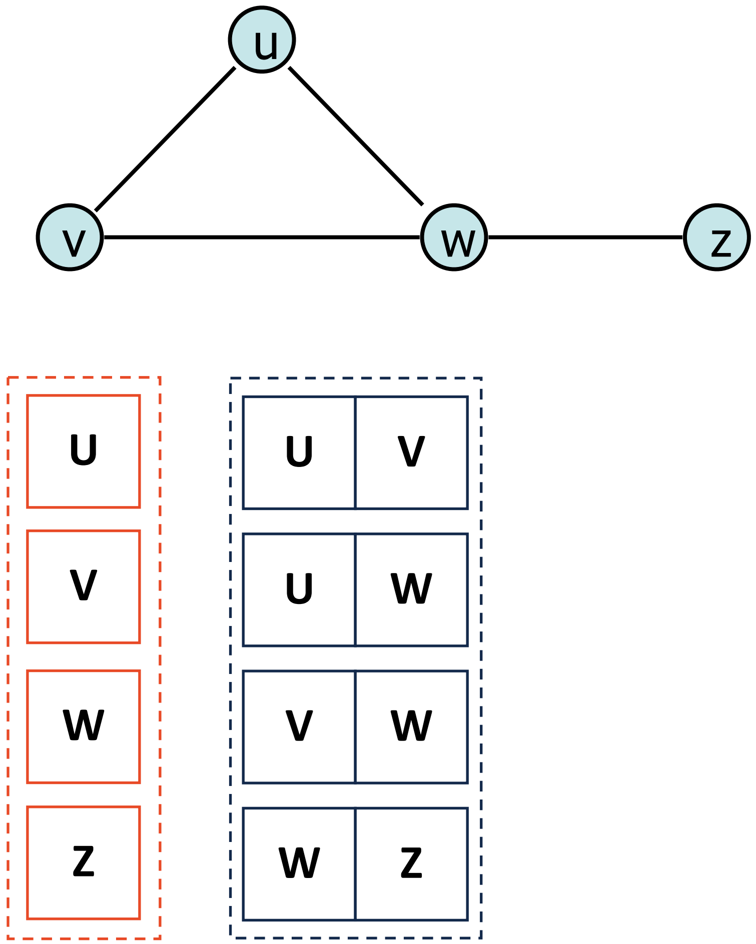
Remove

removeVertex(v) — removes a vertex from the graph

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

Graph Implementation: Edge List

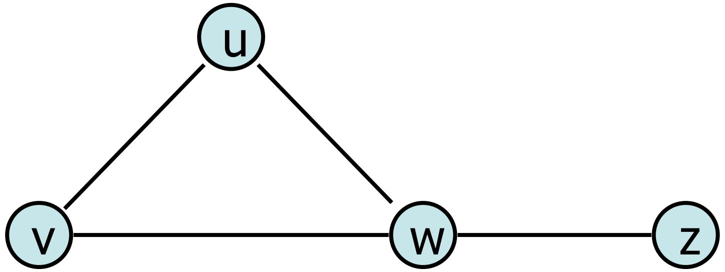
|V|= n, |E|= m



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

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



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

Expressed as O(f)	Adjacency Matrix
Space	n^2
insertVertex(v)	n*
removeVertex(v)	n*
insertEdge(u, v)	1
removeEdge(u, v)	1
getEdges(v)	n
areAdjacent(u, v)	1

Graph Implementations

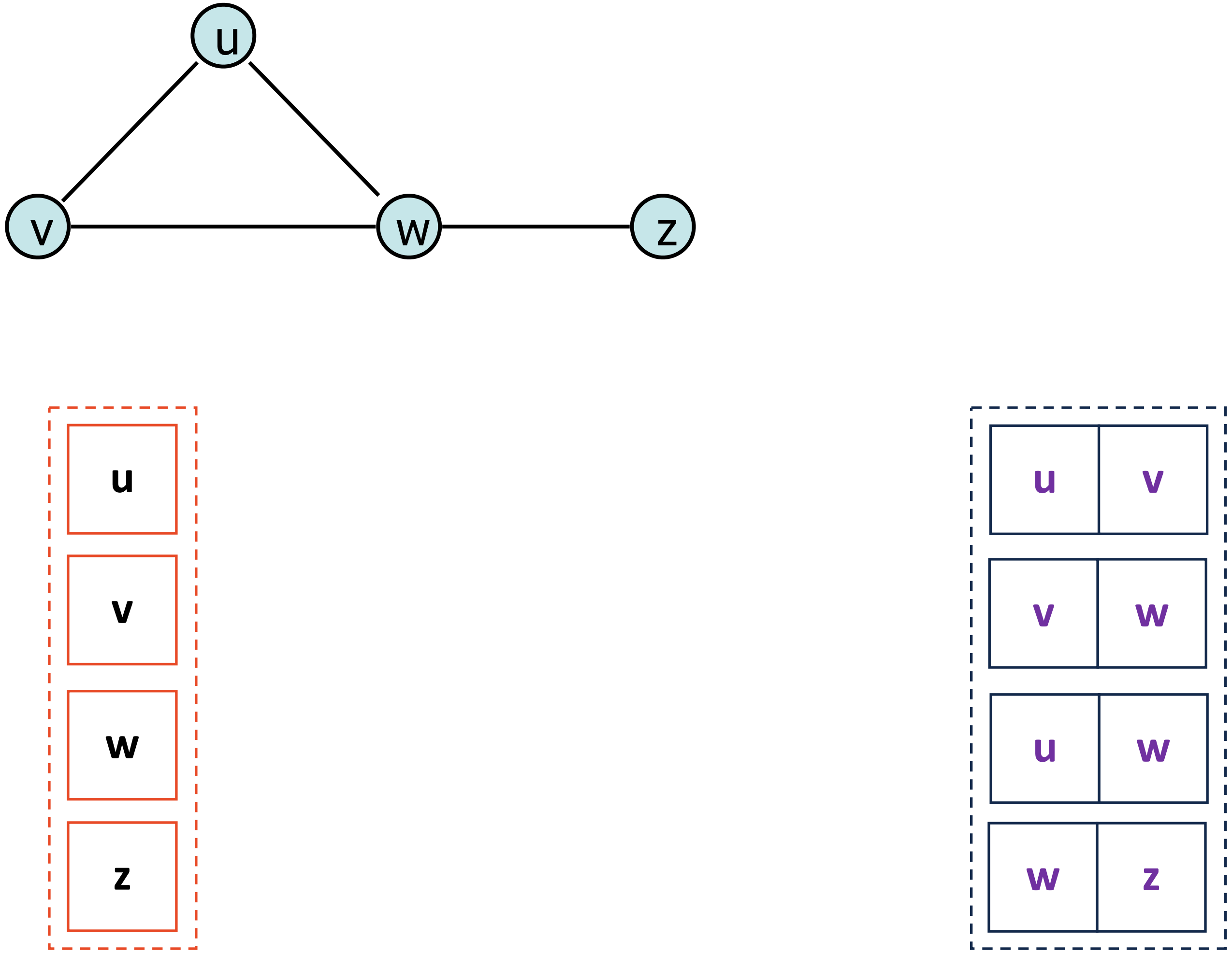
I want a graph that uses the least amount of memory possible

I want a graph that has the fastest lookup for specific edges

I want a graph that is efficient for a sparse dataset

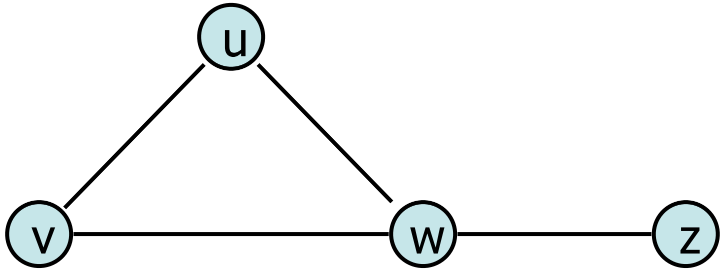
Graph Implementation: Edge List + ?

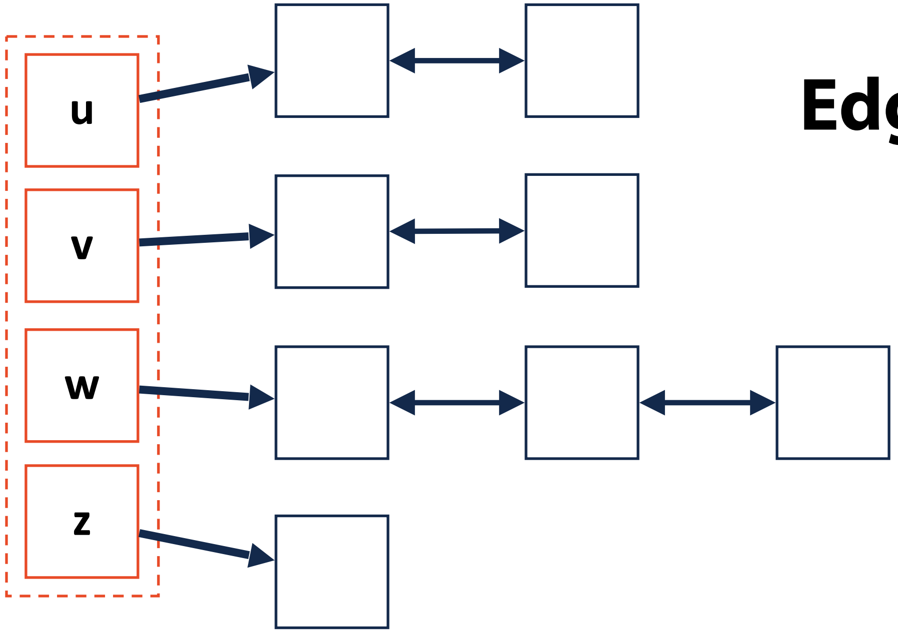
$|V| = n, |E| = m$



Adjacency List

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



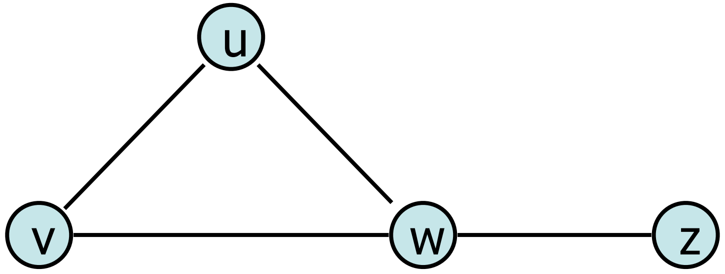


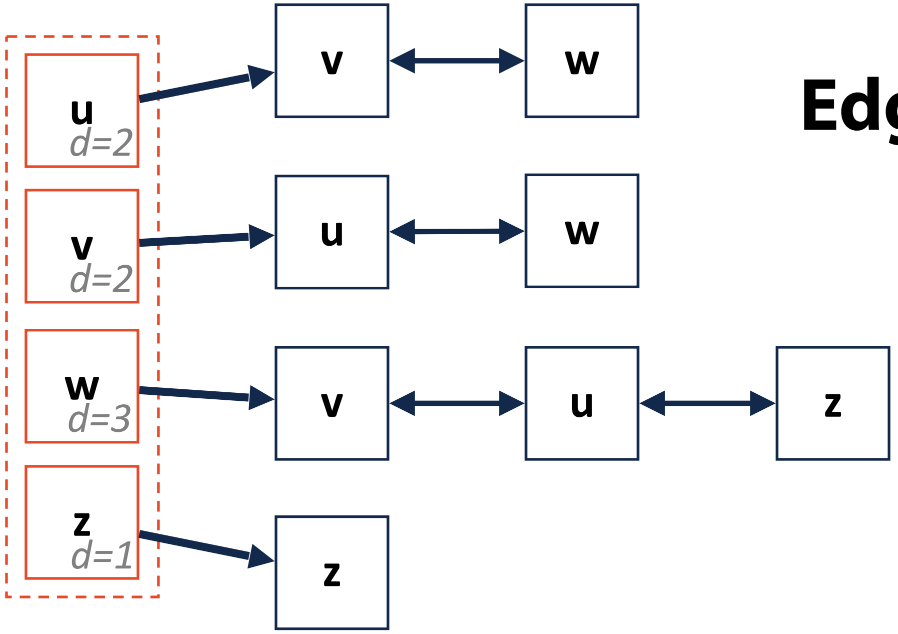
Vertex Storage:

Edge Storage:

Adjacency List

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



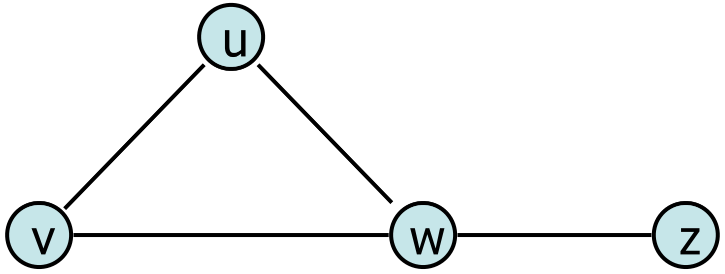


Vertex Storage:

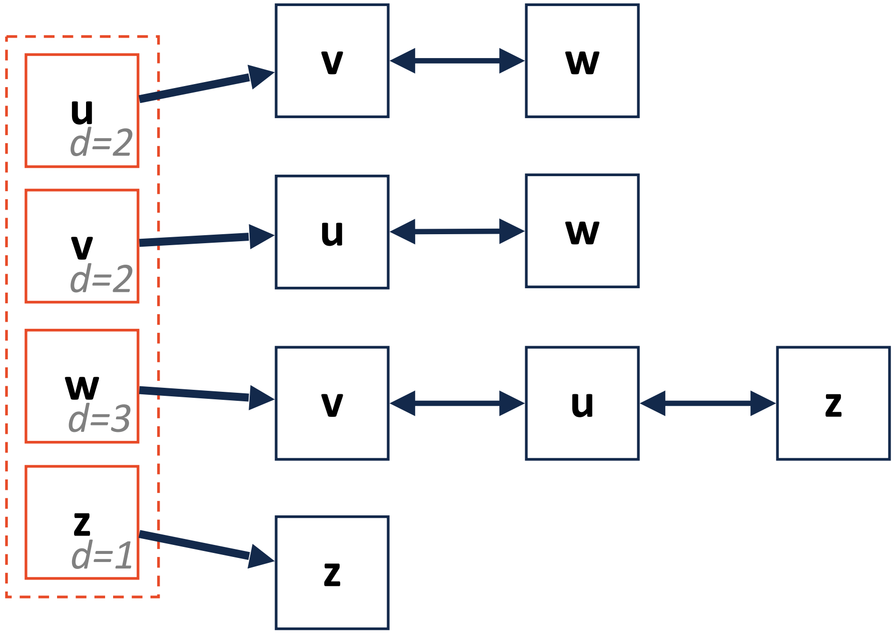
Edge Storage:

Adjacency List

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

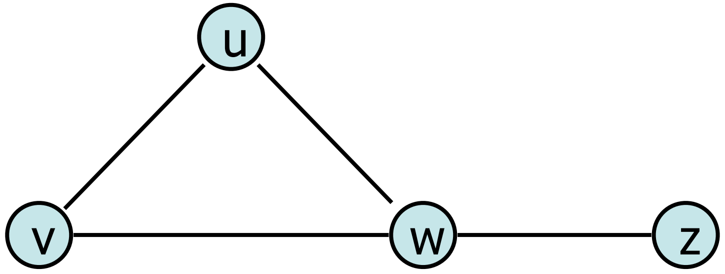


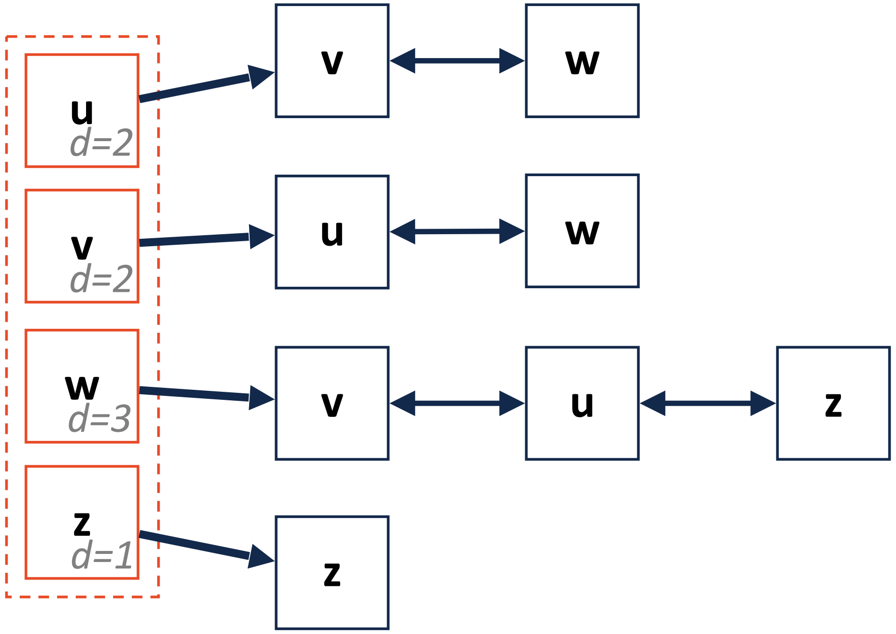
getVertices():



Adjacency List

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

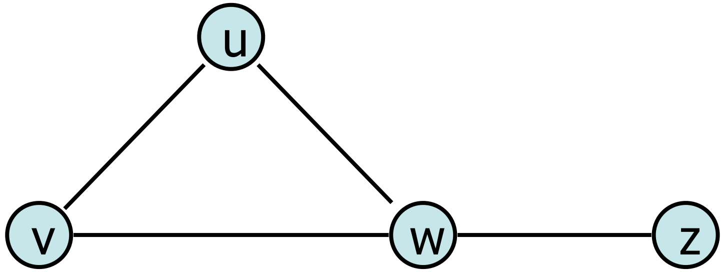




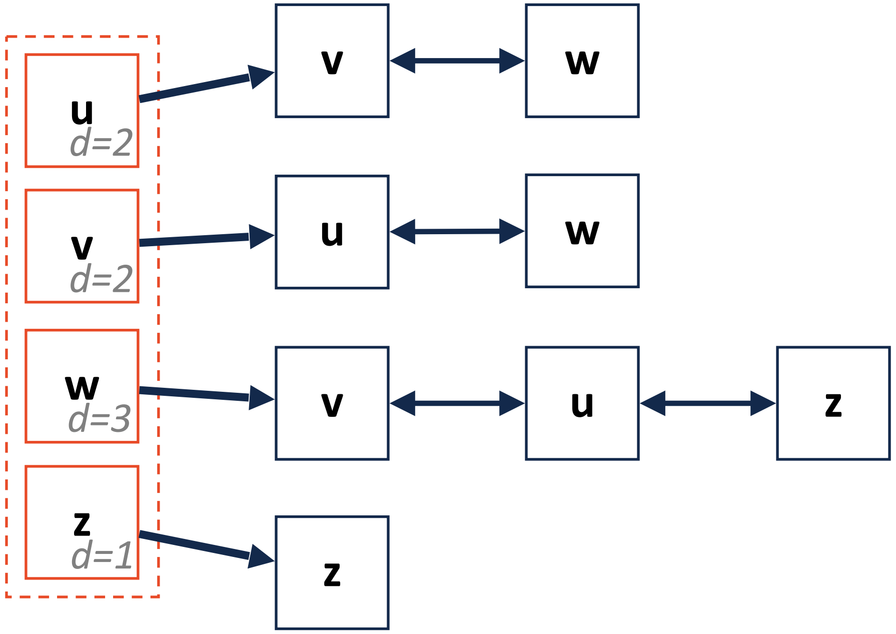
getEdges(v):

Adjacency List

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

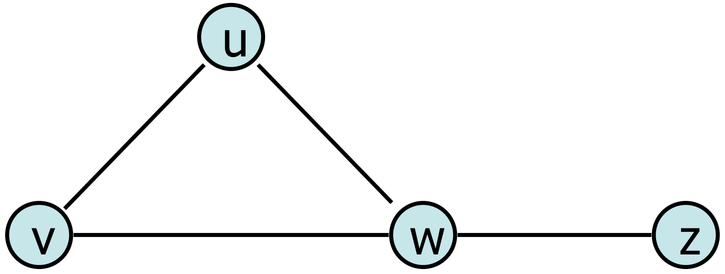


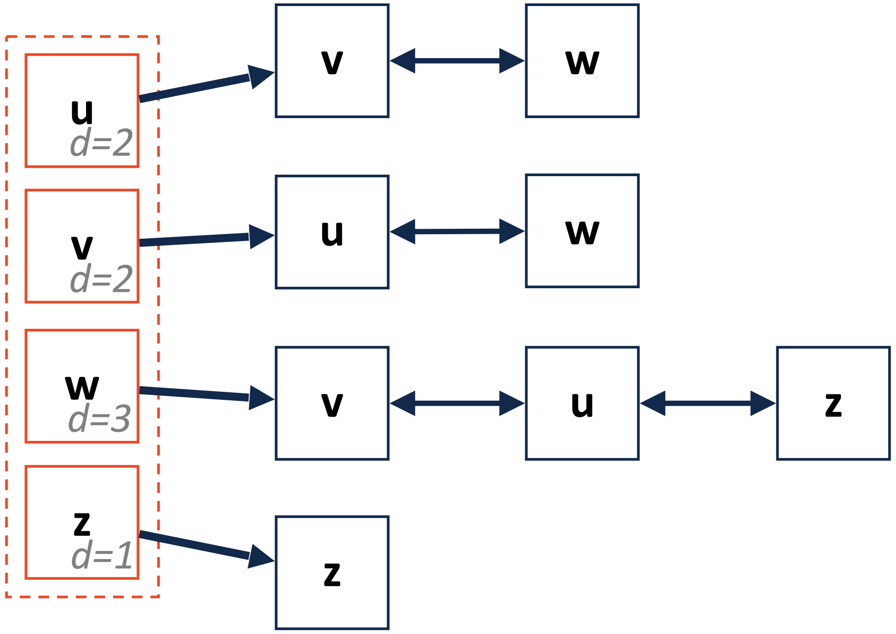
areAdjacent(u, v):



Adjacency List

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

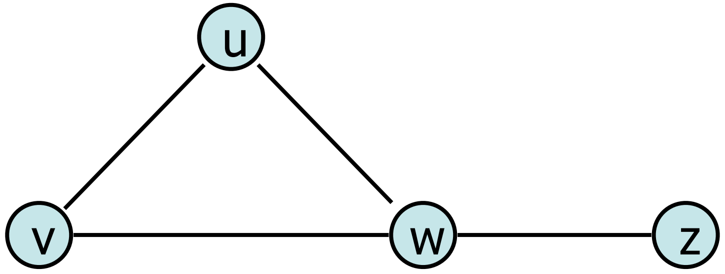


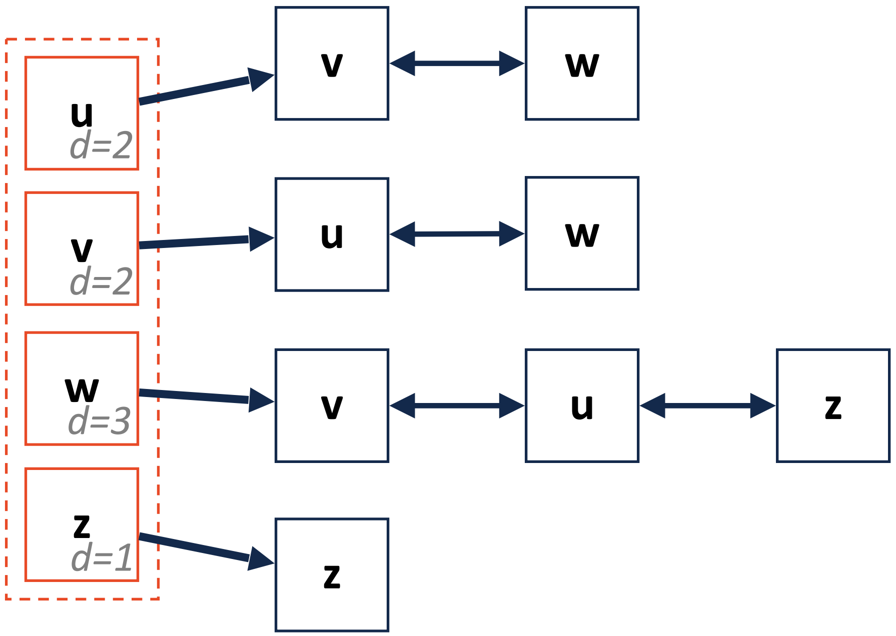


removeVertex(v):

Simple Adjacency List

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



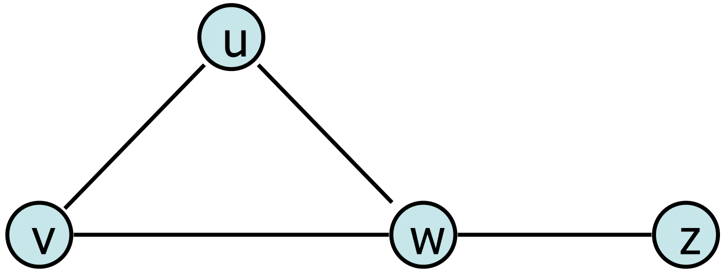


What's wrong with our implementation?

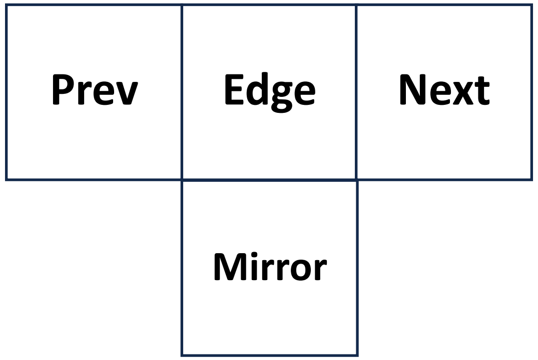
How can we fix it?

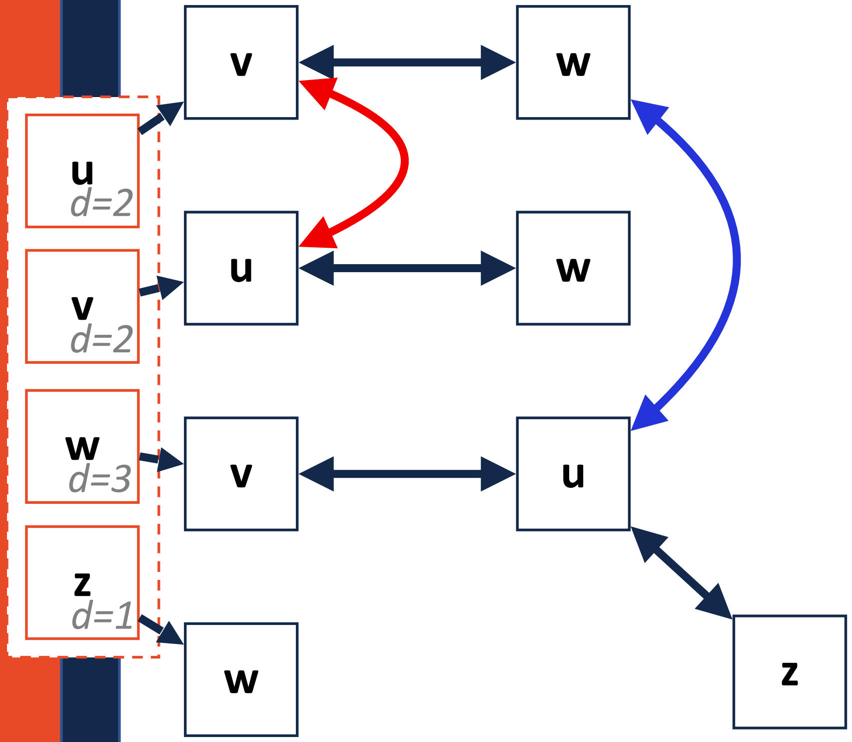
Adjacency List

|V|= n, |E|= m



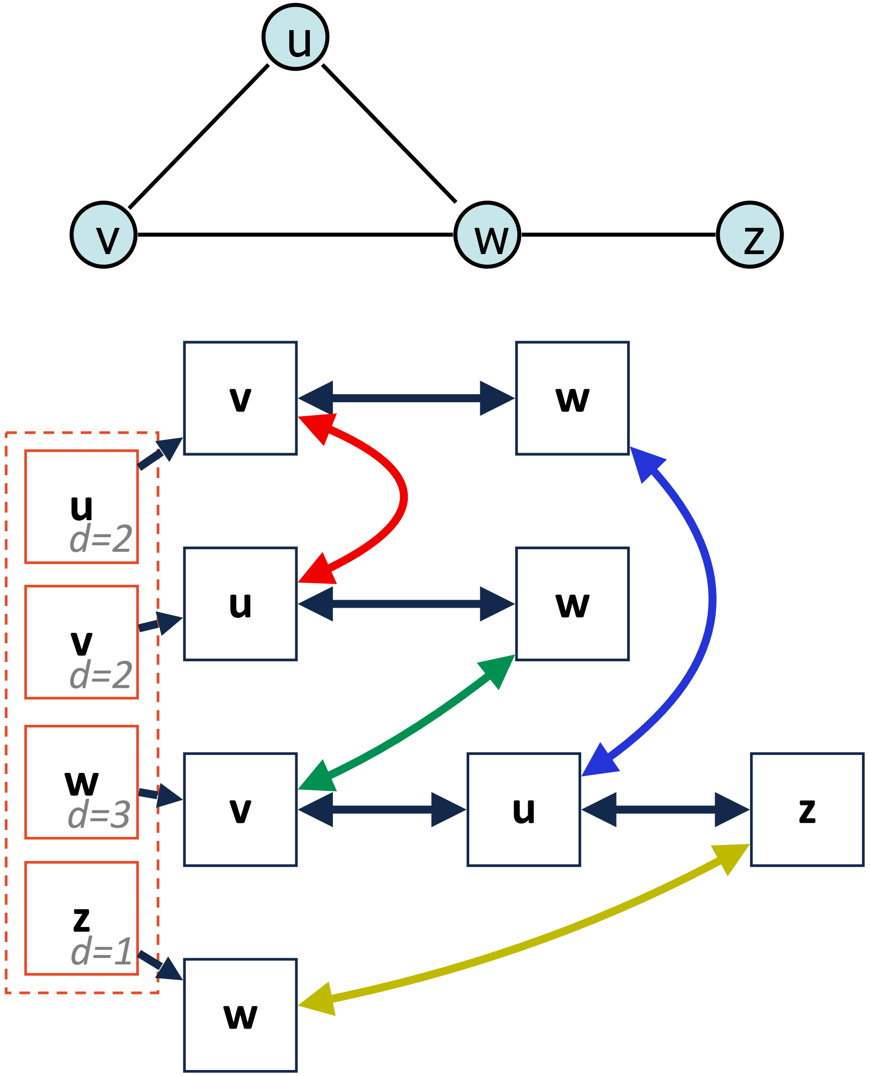
Adj List Node:





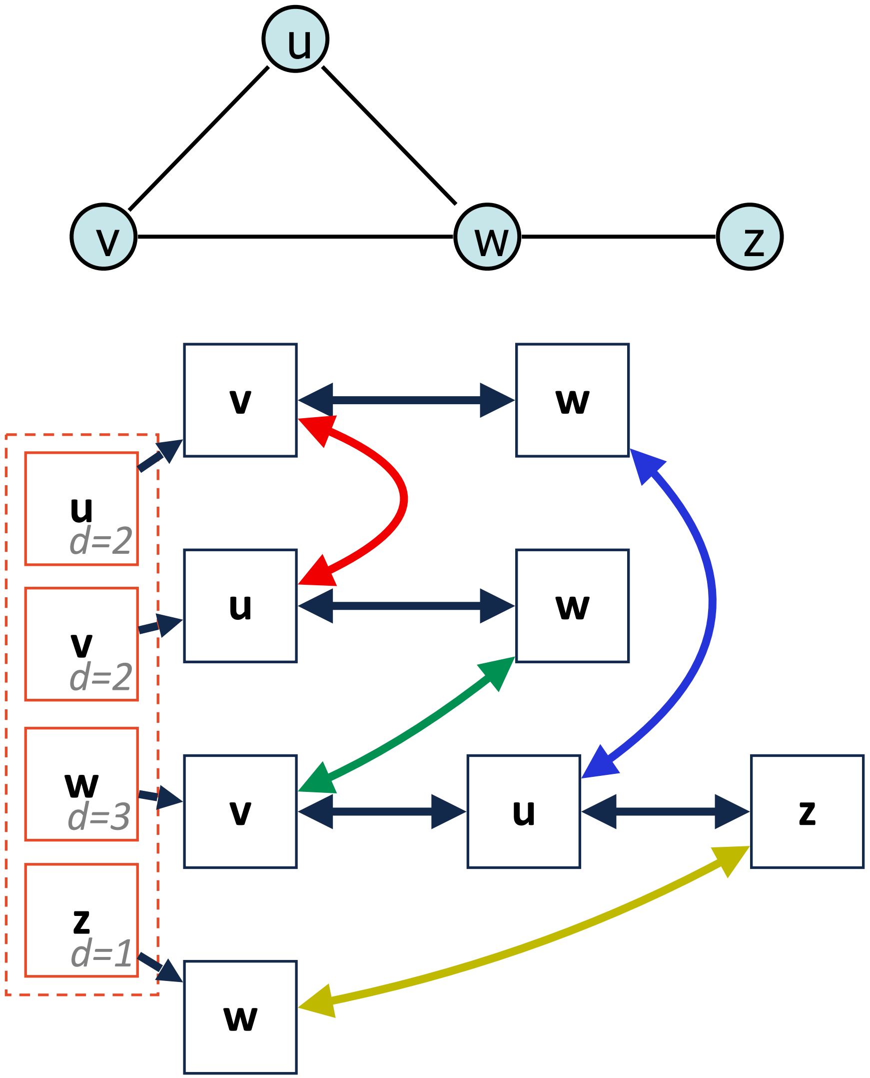
This is still a 'lie' but a more accurate one!

Adjacency List



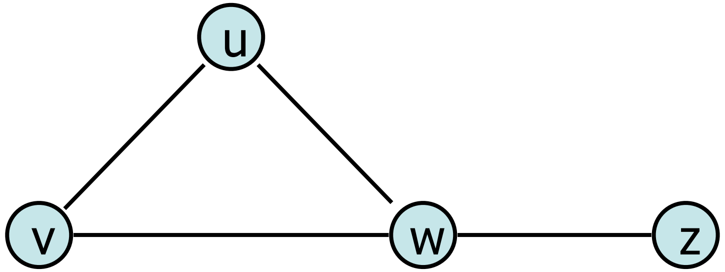
insertVertex(v):

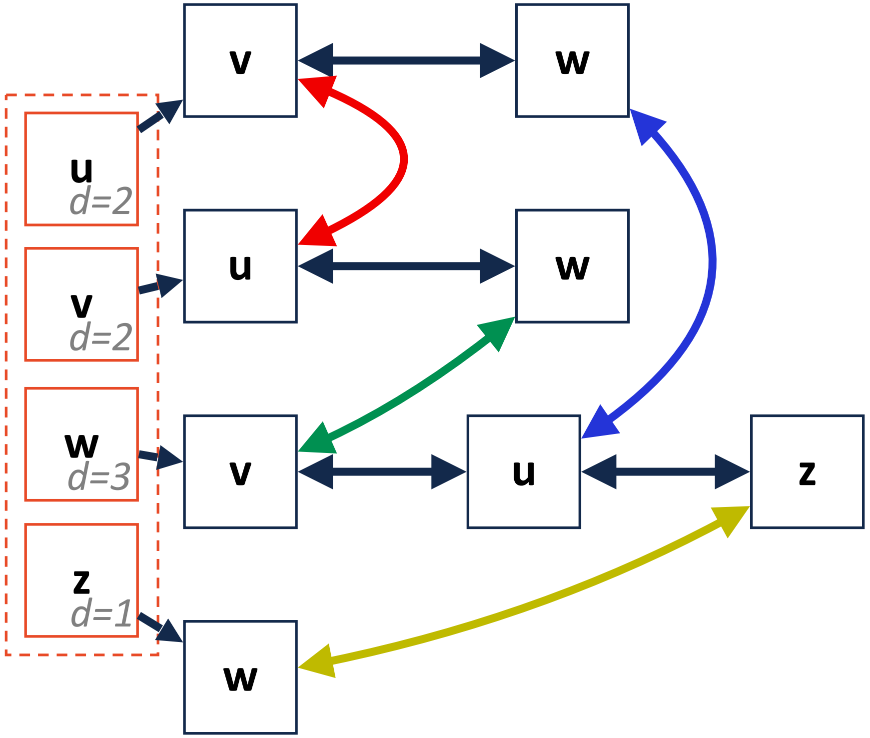
Adjacency List



insertEdge(u, v):

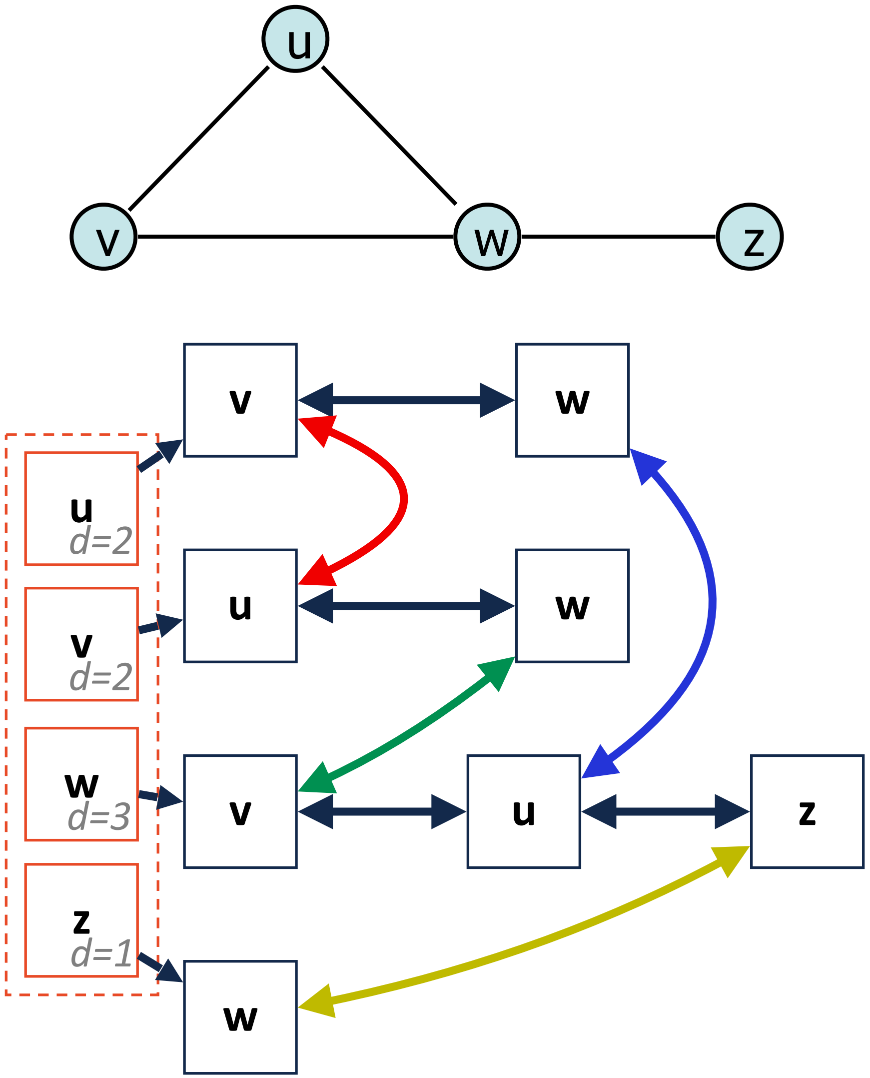
Adjacency List





removeEdge(u, v):

Adjacency List



Pros:

Cons:

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

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

Where do we go from here?

Consider: How does our implementation change for weights?

for directed edges?

Consider: How can we implement traversal on graphs?

Consider: What are some common graph algorithms / uses?

Graphs in Python: NetworkX Package

NetworkX uses concepts from all three implementation methods

A graph can be built from an edge list, adjacency matrix, or adjacency list

A graph can be saved or output as any of the three formats

Many algorithms (and traversals) are built-in.

Creating a NetworkX graph

```
G = nx.Graph()
```

```
G = nx.Graph(edgeList)
```

```
G.add_node(label, **kwargs)
```

```
G.add_edge(v1, v2, **kwargs)
```

NetworkX Example

```
import networkx as nx

G = nx.Graph()

G.add_edge("A", “B")

G.add_edge("B", "C", weight=5)

G.add_edge("A", "C", anything="Bob", I="was", want="here")


print(G.nodes())


print(G.edges())


print(G.edges(data=True))
```

NetworkX Example

```
import networkx as nx

G = nx.Graph()

G.add_node(“A")

G.add_node("B", name=“Bob")

G.add_node("C", anything="Bob", I="was", want="here")

print(G.nodes())

print(G.nodes(data=True))
```

NetworkX Example

```
G=nx.random_regular_graph(3, 6)

nx.draw(G, edge_color='k', width=2, with_labels=True)

plt.show()

m = nx.adjacency_matrix(G)
print(m.todense())







adjL = nx.generate_adjlist(G)
for line in adjL:
    print(line)
```

Graphs in Python: NetworkX Package

Networkx (and Python packages in general) can do a lot for you!

But they can sometimes make design decisions that don't work for you.

Ex: An adjacency list in NetworkX doesn't duplicate edges!