

Data Structures

Graph Implementations 2

CS 225

October 25, 2024

Brad Solomon



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

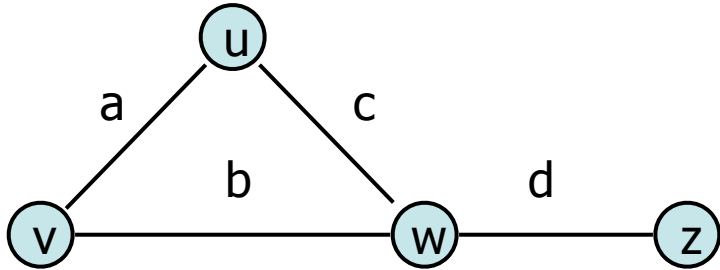
Department of Computer Science

Learning Objectives

Discuss graph implementation and storage strategies

Introduce graph traversals

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



$O(1)^*$

insertVertex(K key):

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

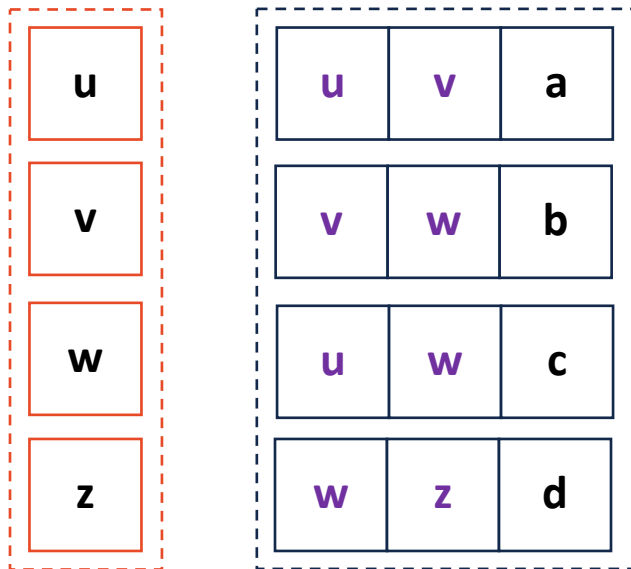
$O(m)$

removeVertex(Vertex v):

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

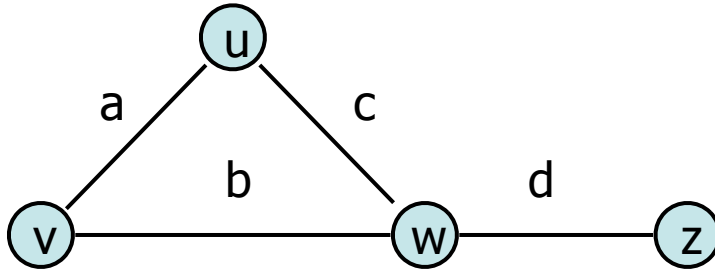
incidentEdges(Vertex v):

areAdjacent(Vertex v1, Vertex v2):



Graph Implementation: Adjacency Matrix

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



Vertex Storage:

A hash table of vertices

Implicitly or explicitly store index

Edge Storage:

A $|V| \times |V|$ matrix of edges

Weight is stored at position (u, v)

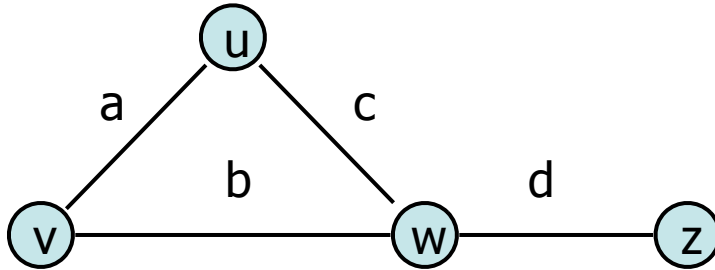
| | |
|---|---|
| u | 0 |
| v | 1 |
| w | 2 |
| z | 3 |

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | - | a | c | 0 |
| 1 | | - | b | 0 |
| 2 | | | - | d |
| 3 | | | | - |

Graph Implementation: Adjacency Matrix

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

removeVertex(Vertex v):



| | |
|---|---|
| u | 0 |
| v | 1 |
| w | 2 |
| z | 3 |

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | - | a | c | 0 |
| 1 | | - | b | 0 |
| 2 | | | - | d |
| 3 | | | | - |

Amortized Removal with Tombstoning

Remove an item by replacing its value or flipping a flag indicating 'deletion'

| | | | | | | | | | |
|---|---|---|---|---|----|---|---|---|---|
| 2 | 7 | 5 | 9 | 7 | 14 | 1 | 0 | 8 | 3 |
|---|---|---|---|---|----|---|---|---|---|

When there are enough deleted elements to merit resize, do it all at once!

| | |
|---|---|
| u | 0 |
| v | 1 |
| w | 2 |
| z | 3 |

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | - | a | c | 0 |
| 1 | | - | b | 0 |
| 2 | | | - | d |
| 3 | | | | - |

Adjacency Matrix:

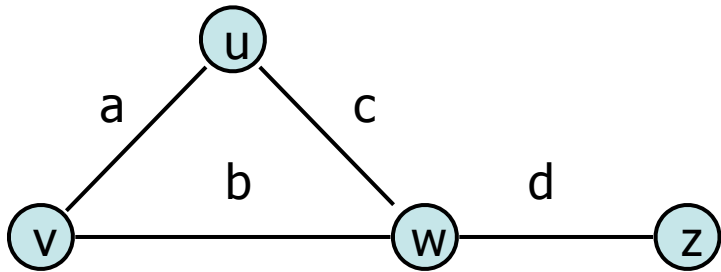
removeVertex() by tombstoning $|V|$ values

Resize when needed or by request

Graph Implementation: Adjacency Matrix



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



$O(1)$

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

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

areAdjacent(Vertex v1, Vertex v2):

$O(n)$

incidentEdges(Vertex v):

$O(n) \text{---} O(n^2)$

insertVertex(K key):

removeVertex(Vertex v):

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

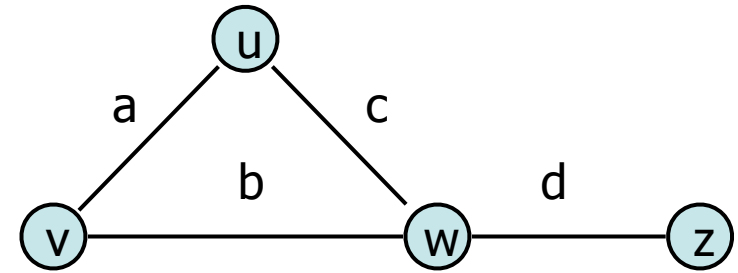
Graph Implementation Brainstorming

We want something...

Faster than an edge list

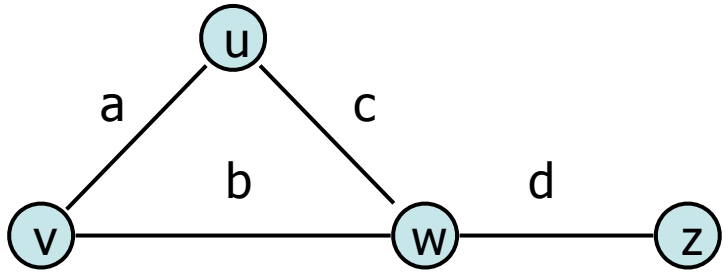
Less space than an adjacency matrix

Particularly good at **finding adjacent elements**



Graph Implementation: Edge List + ?

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

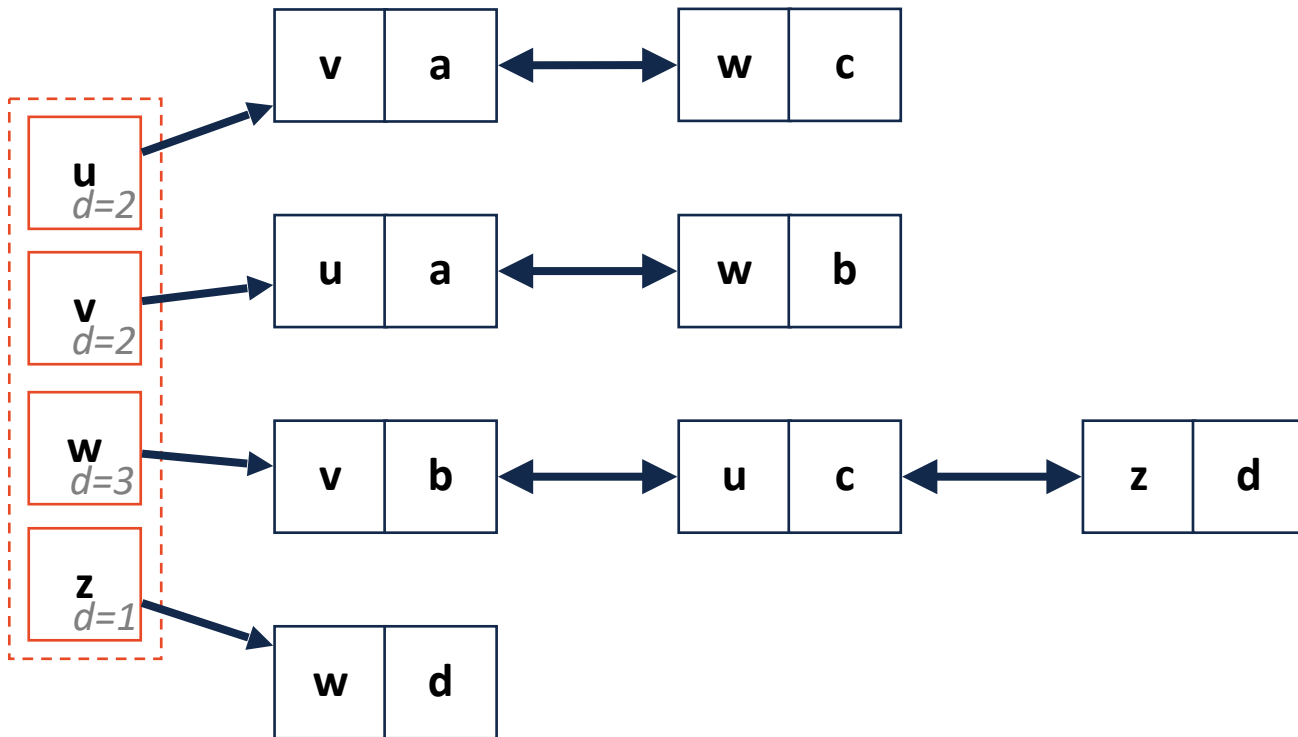
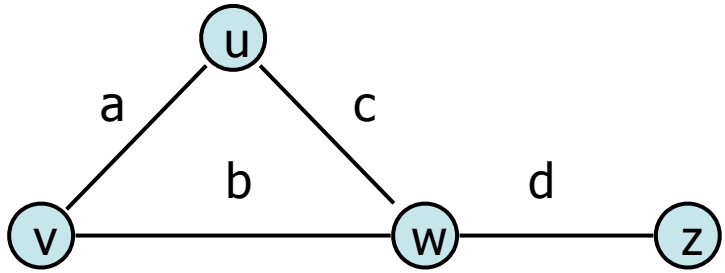


| |
|---|
| u |
| v |
| w |
| z |

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

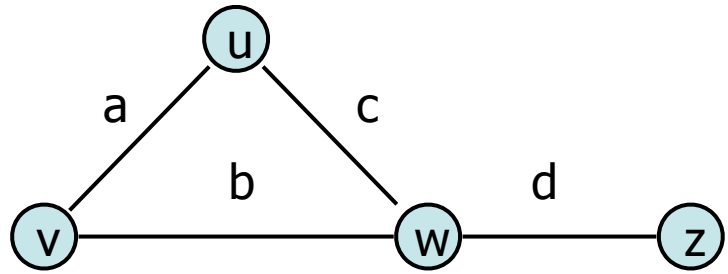
Naive Adjacency List

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



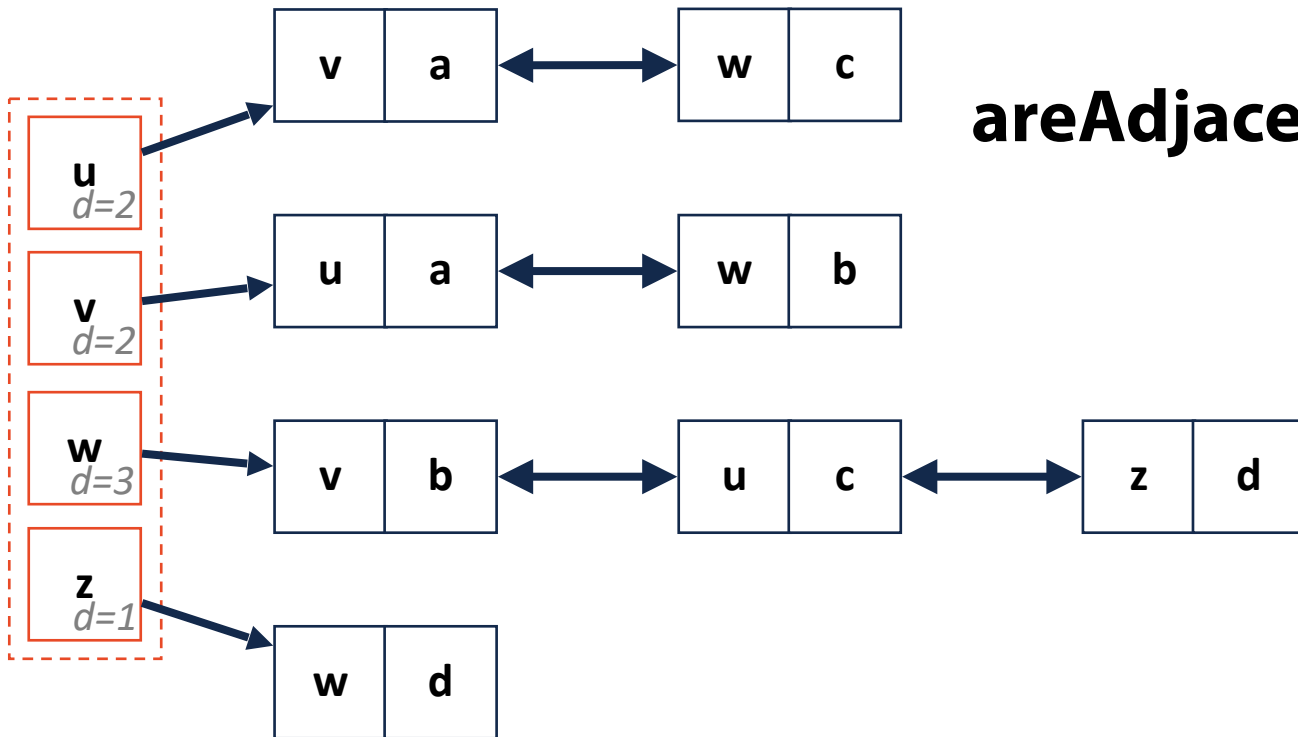
Naive Adjacency List

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



incidentEdges(Vertex v):

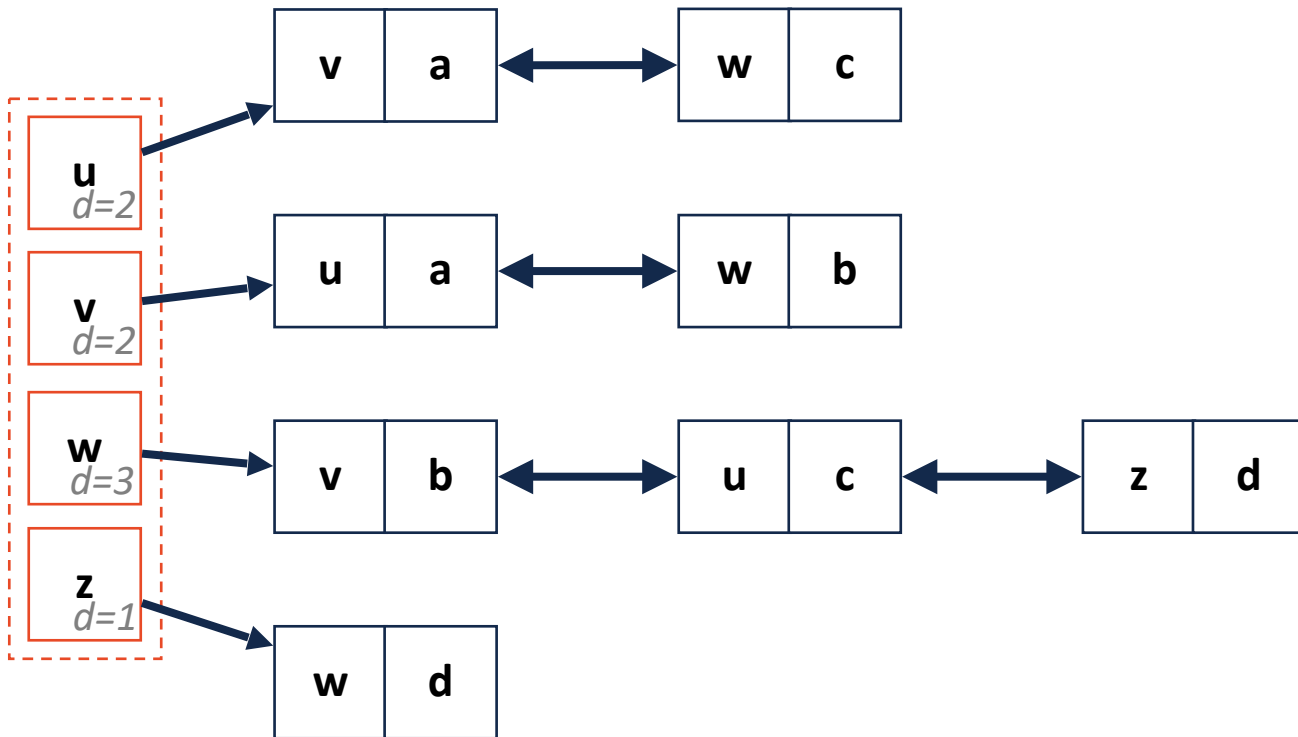
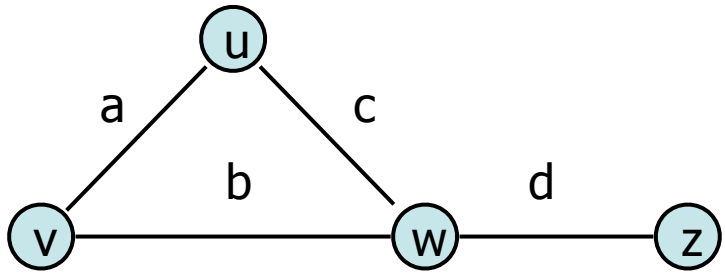
areAdjacent(Vertex v1, Vertex v2):



Naive Adjacency List

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

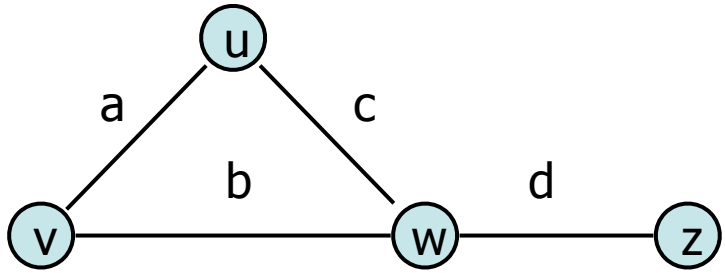
removeVertex(Vertex v):



Naive Adjacency List

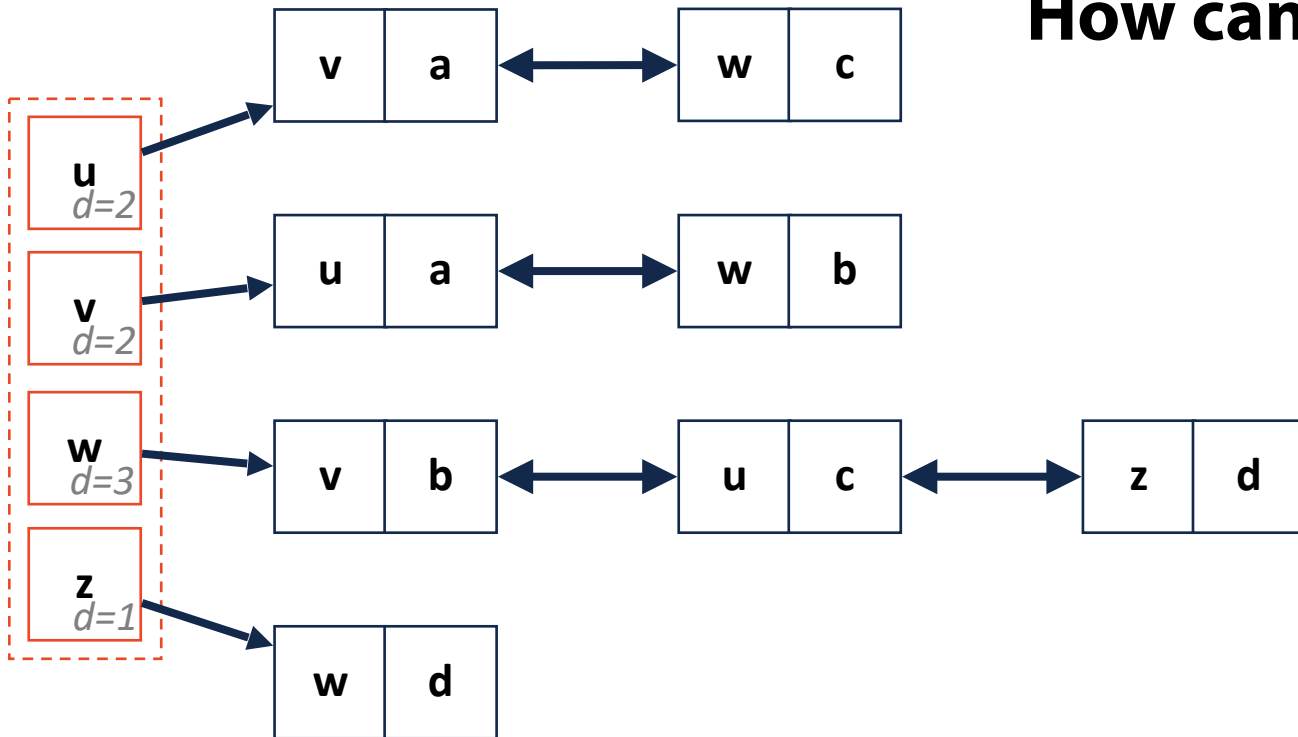


$$|V| = n, |E| = m$$



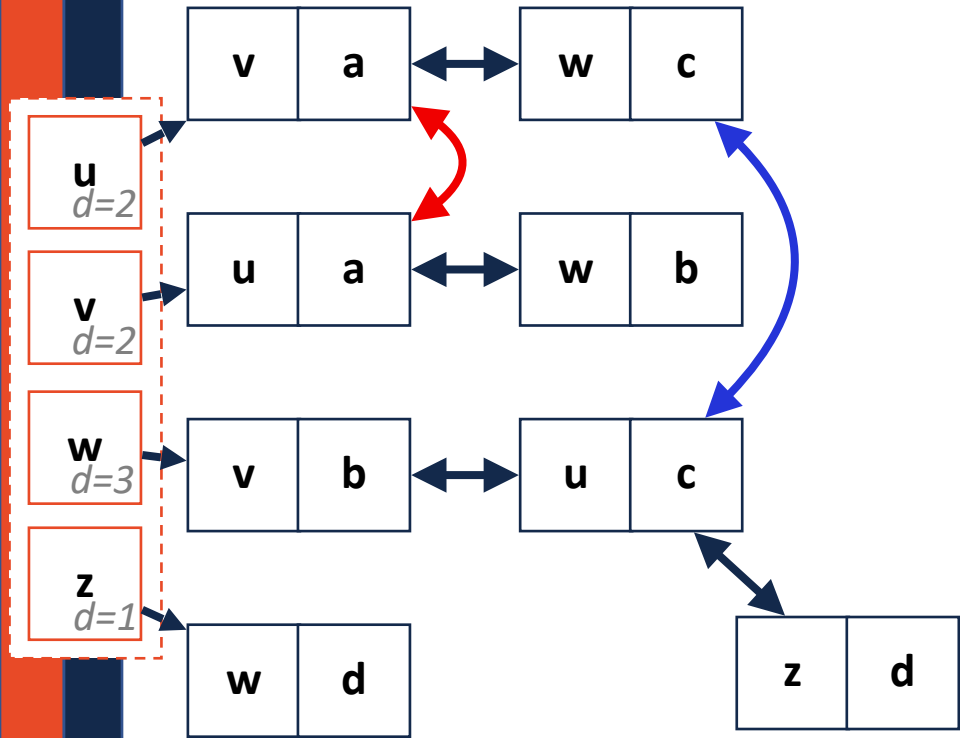
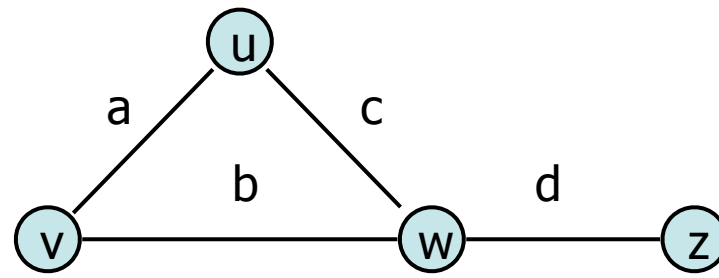
What's wrong with our implementation?

How can we fix it?



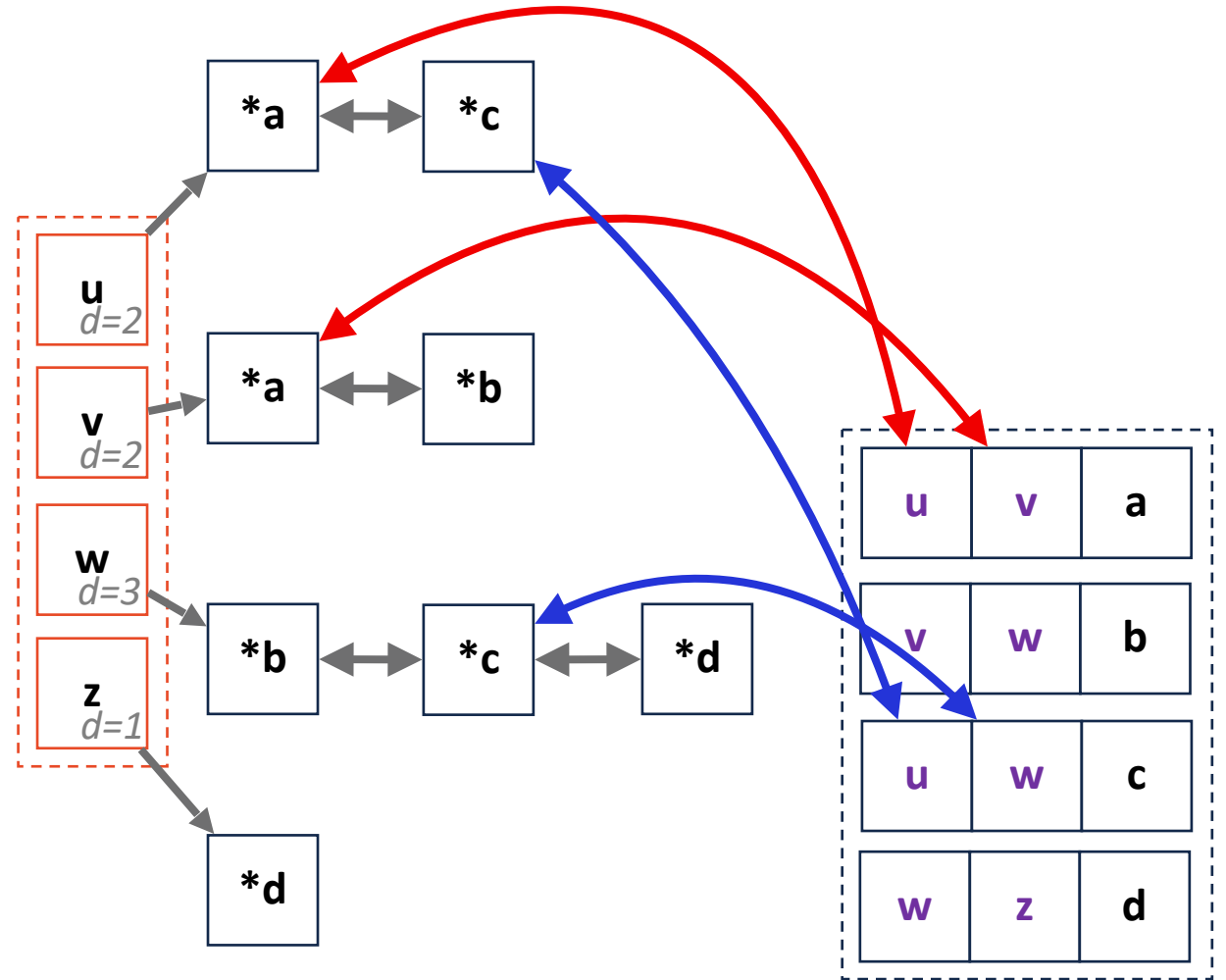
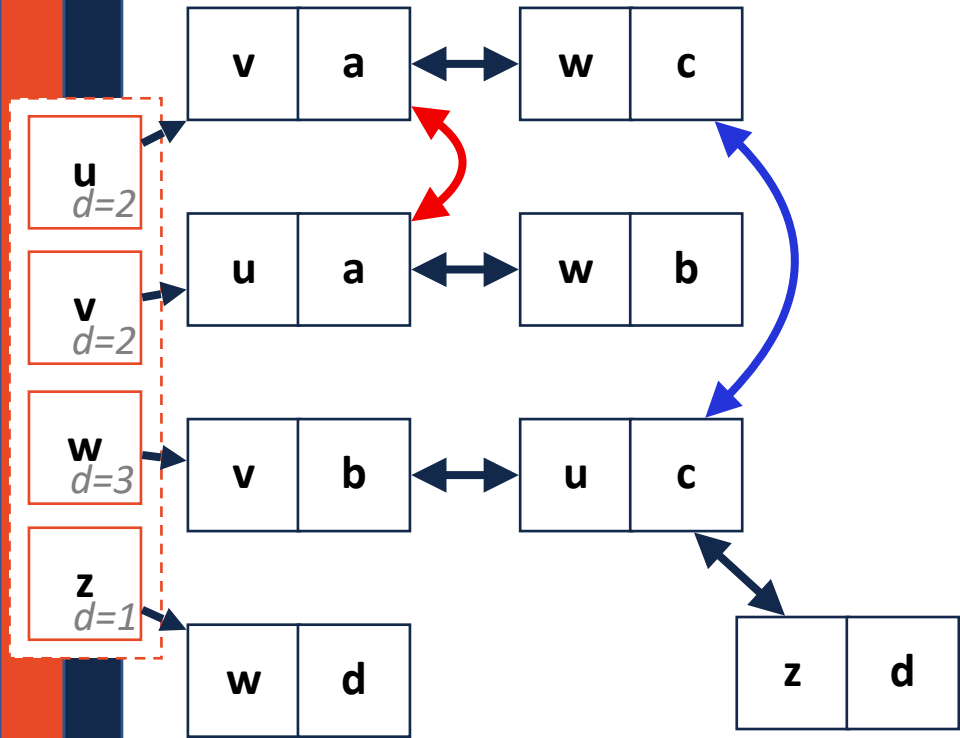
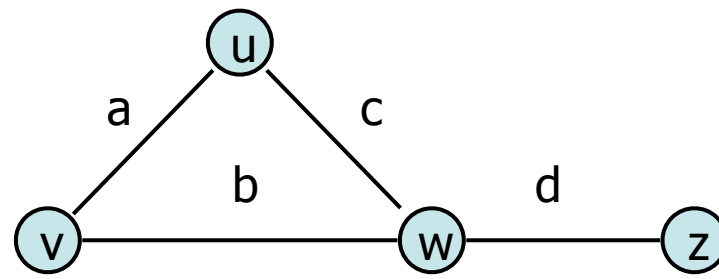
Adjacency List

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



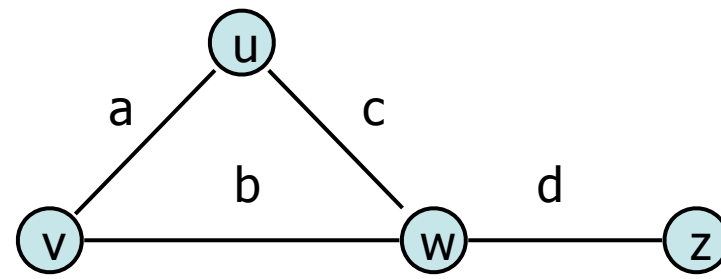
Adjacency List

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

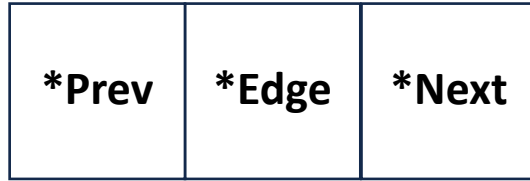


Adjacency List

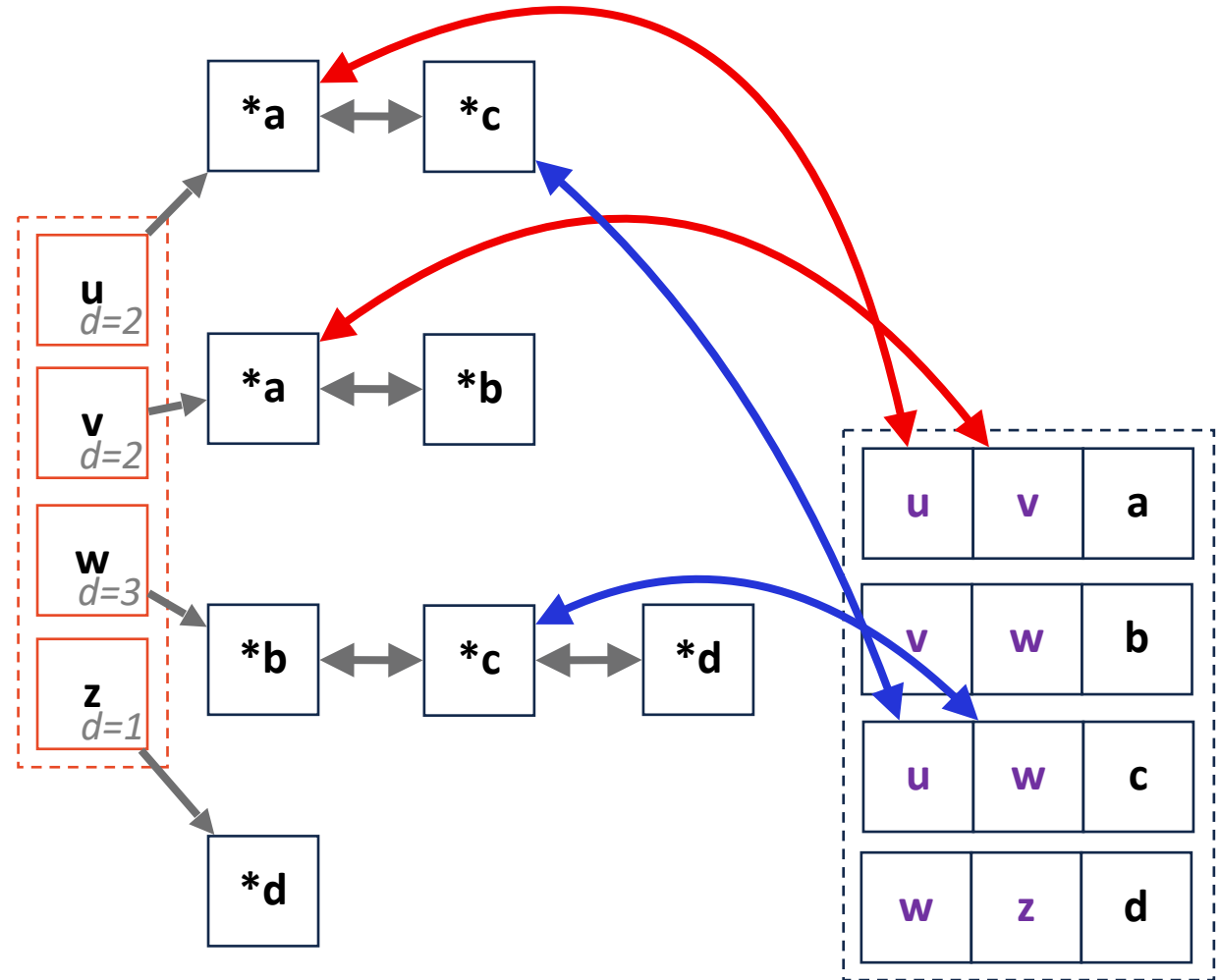
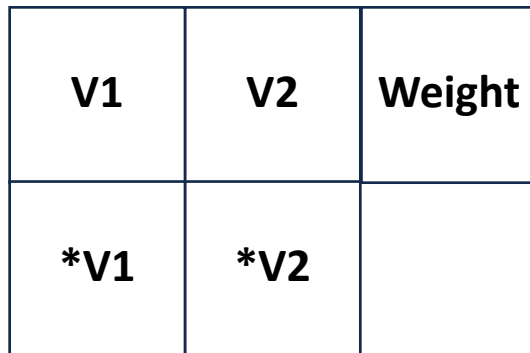
$$|V| = n, |E| = m$$



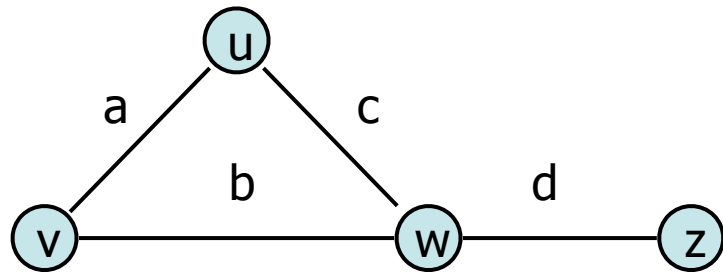
Adj List Node:



Edge List:

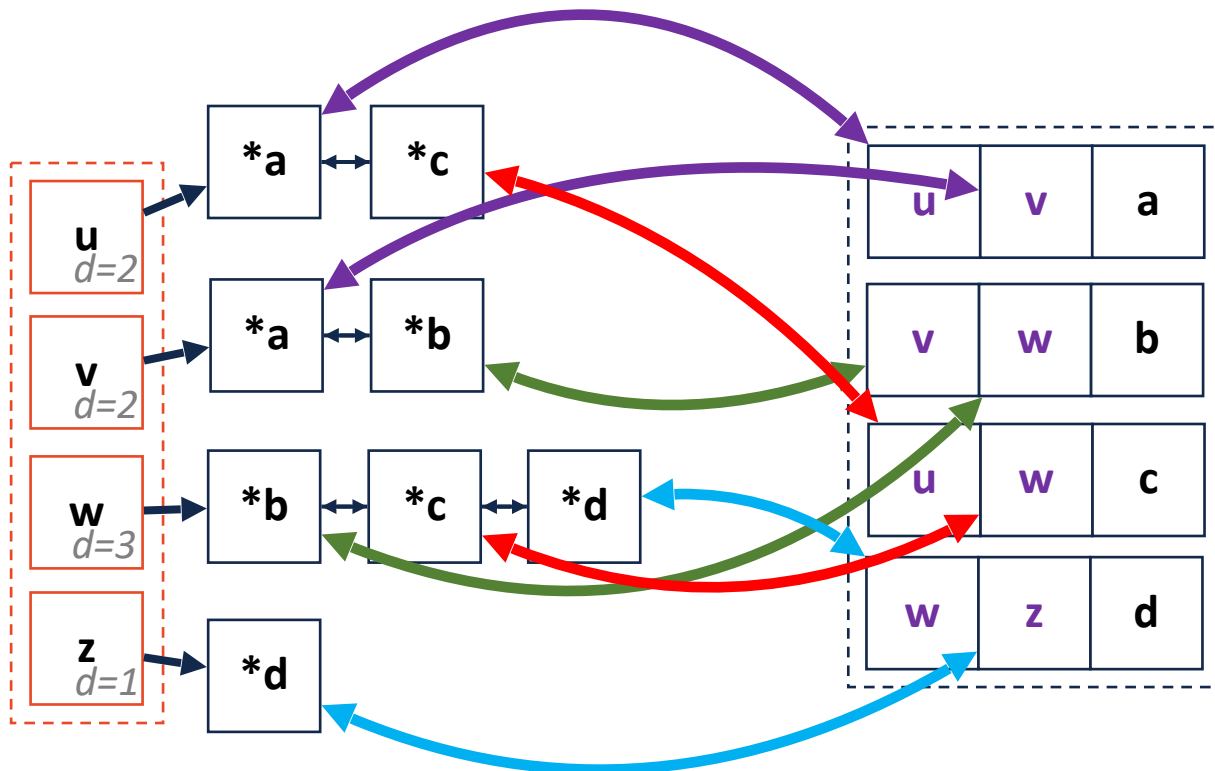


Adjacency List



Vertex Storage:

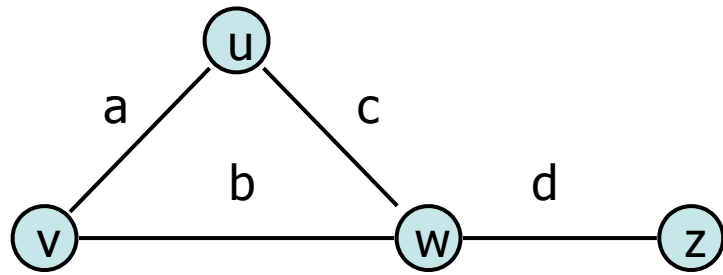
A bidirectional linked list with size variable
Each node is a pointer to edge in edge list



Edge Storage:

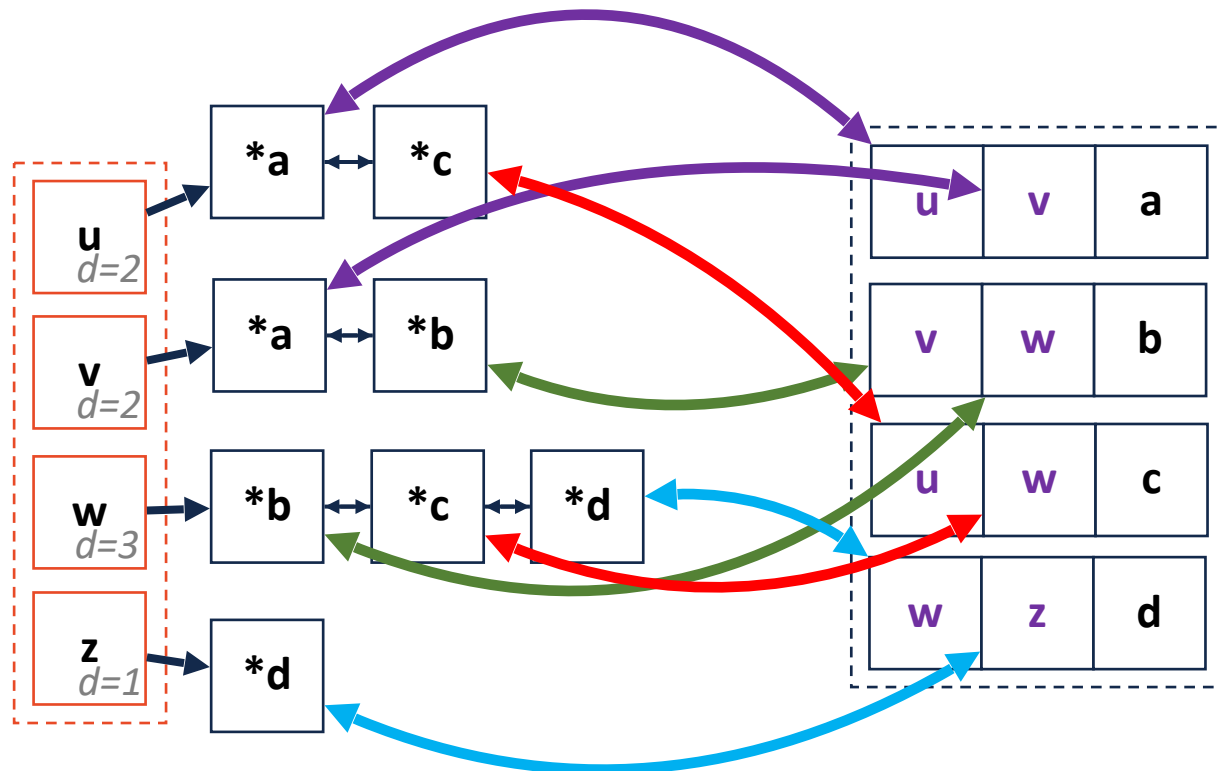
A list of (v1, v2, weight) edges
Also store pointers back to nodes

Adjacency List

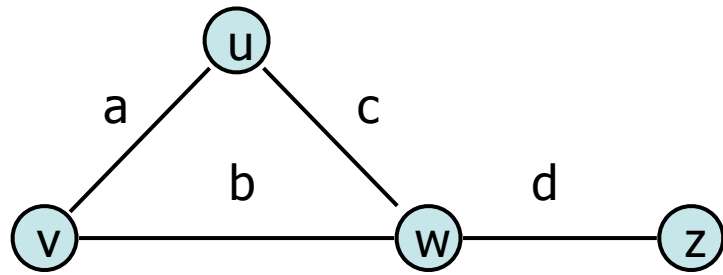


incidentEdges(Vertex v):

Look up vertex list (and walk across it)



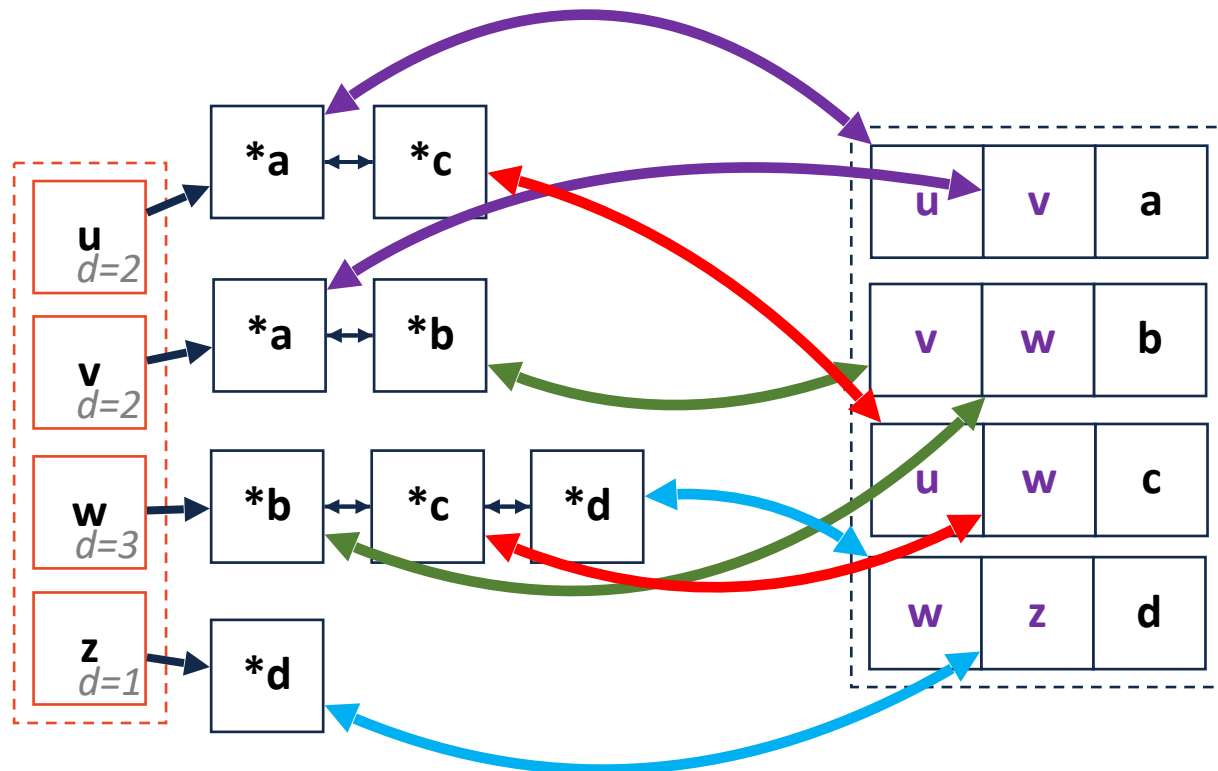
Adjacency List



areAdjacent(Vertex v1, Vertex v2):

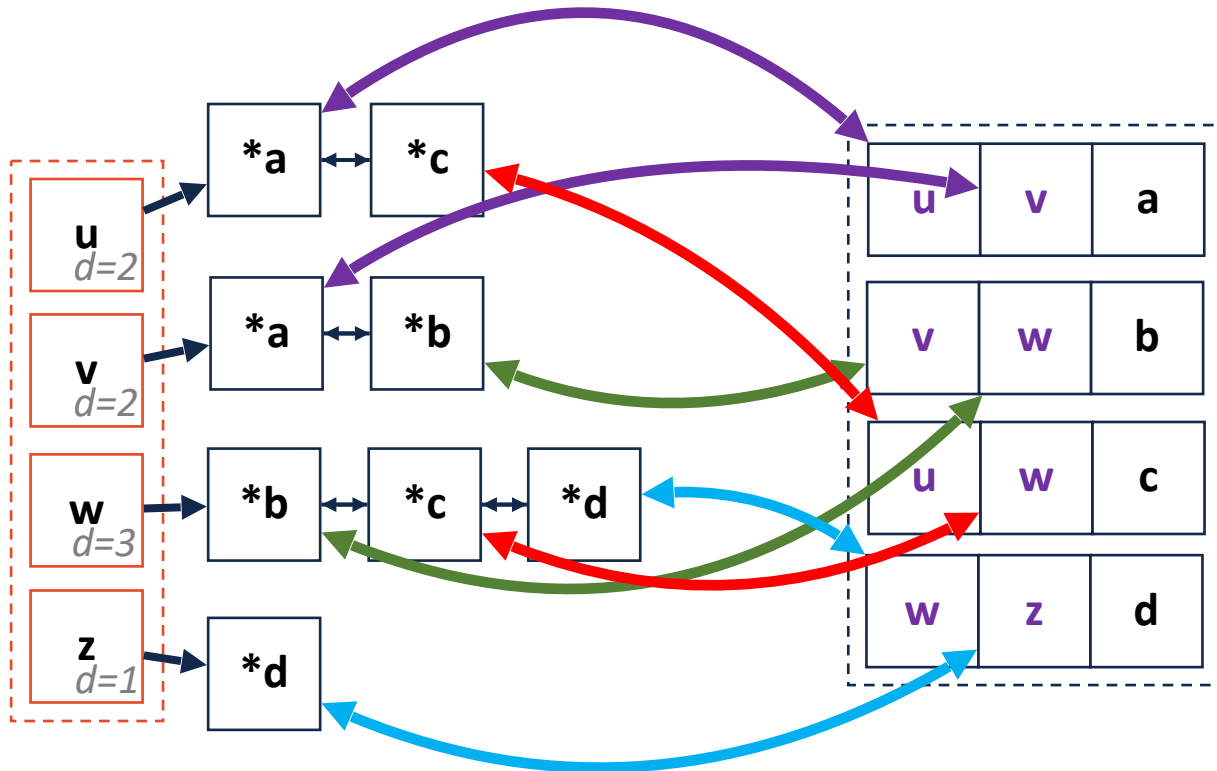
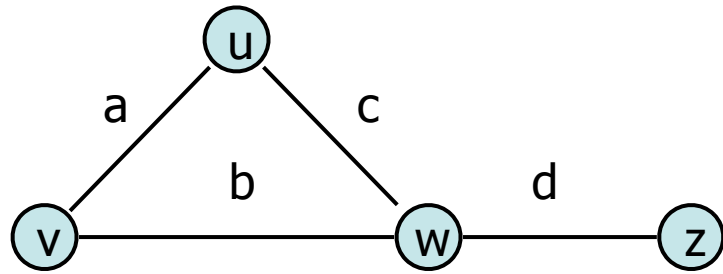
Look up min-degree vertex list

Search for other vertex across list

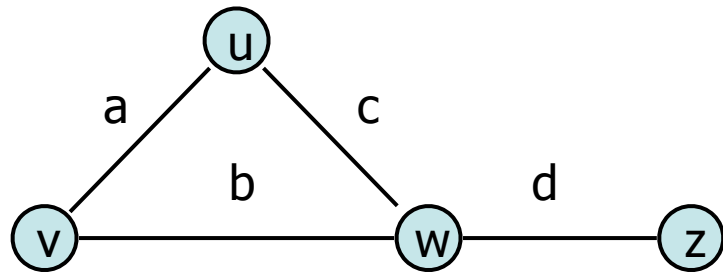


Adjacency List

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



Adjacency List

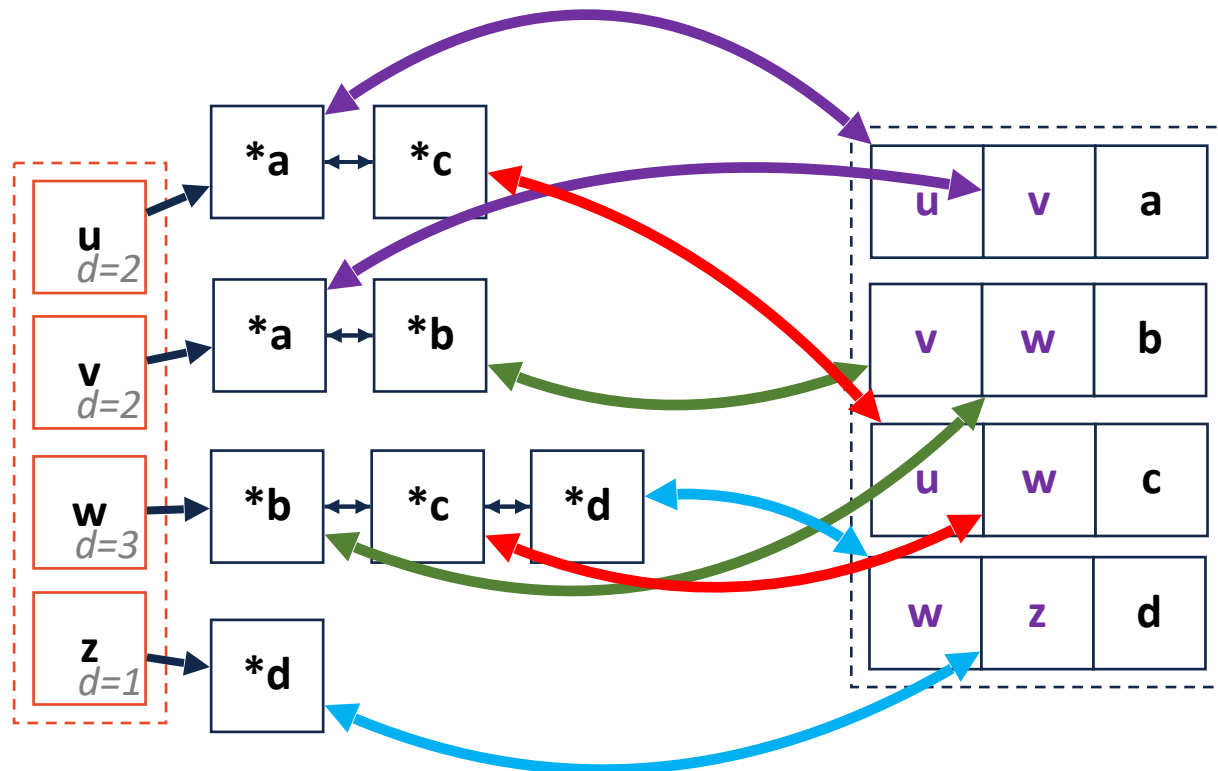


insertEdge(Vertex v1, Vertex v2, K key):

Add edge to edge list

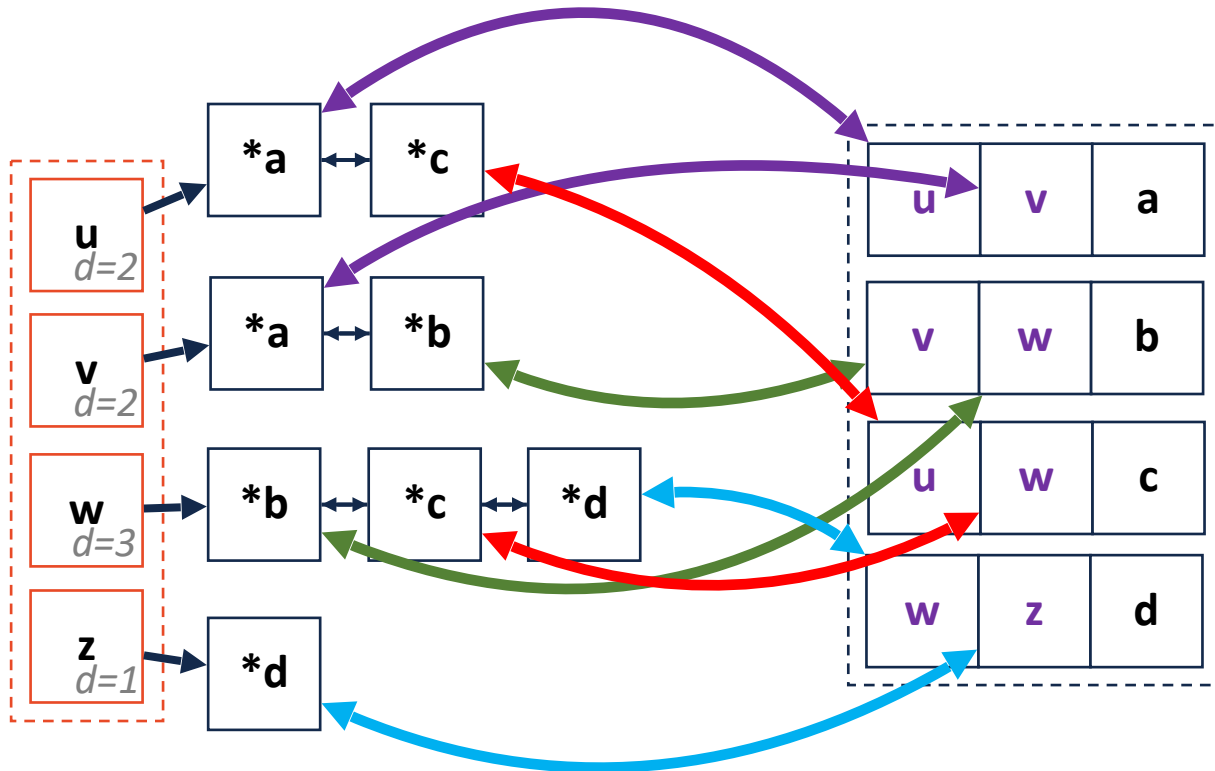
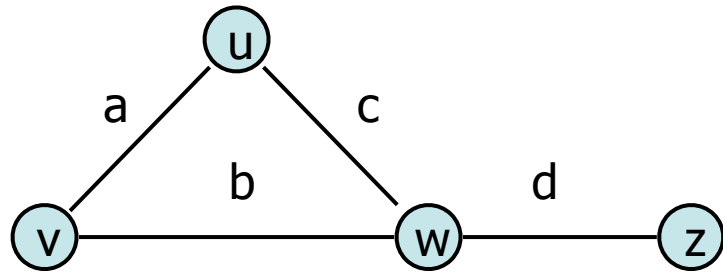
Add node to each vertex list

Connect all three with pointers

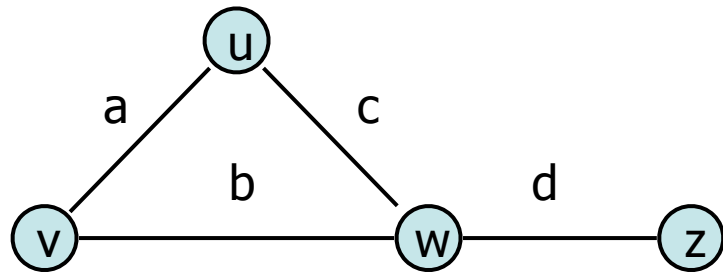


Adjacency List

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



Adjacency List



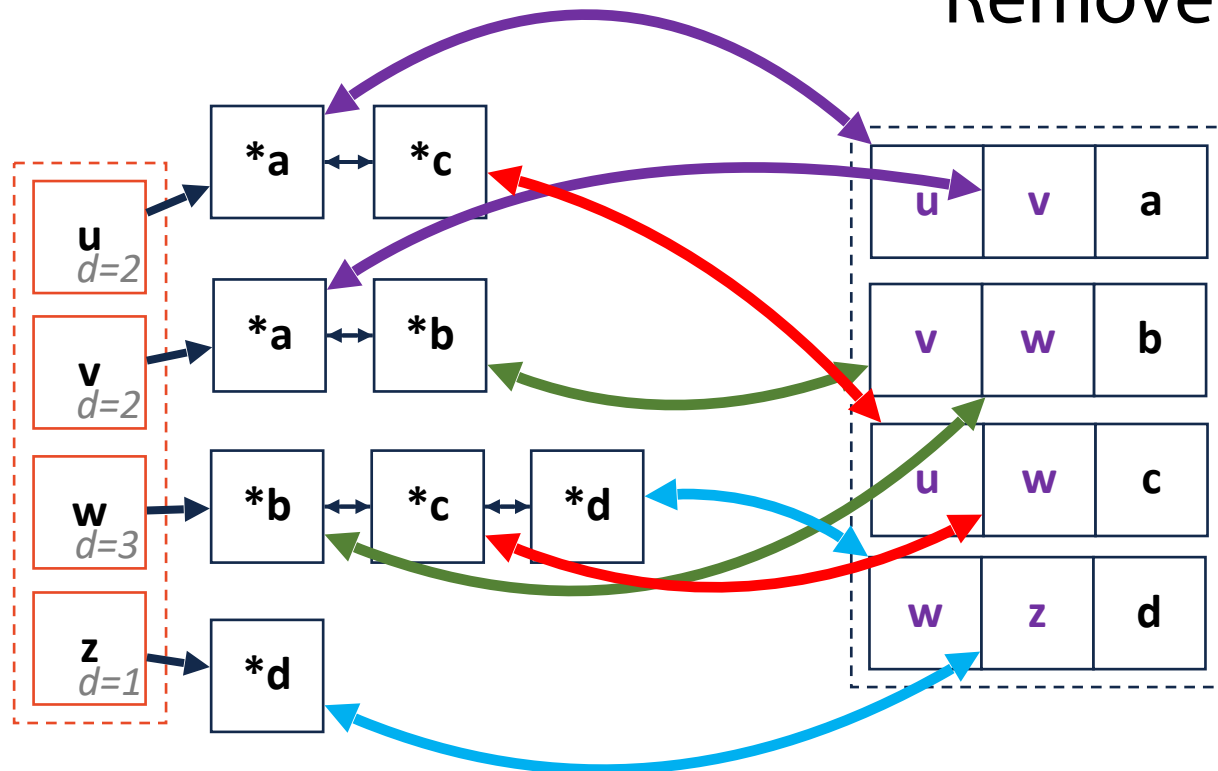
removeEdge(Vertex v1, Vertex v2, K key):

Search min-degree vertex list

Remove mirrored entry using pointers

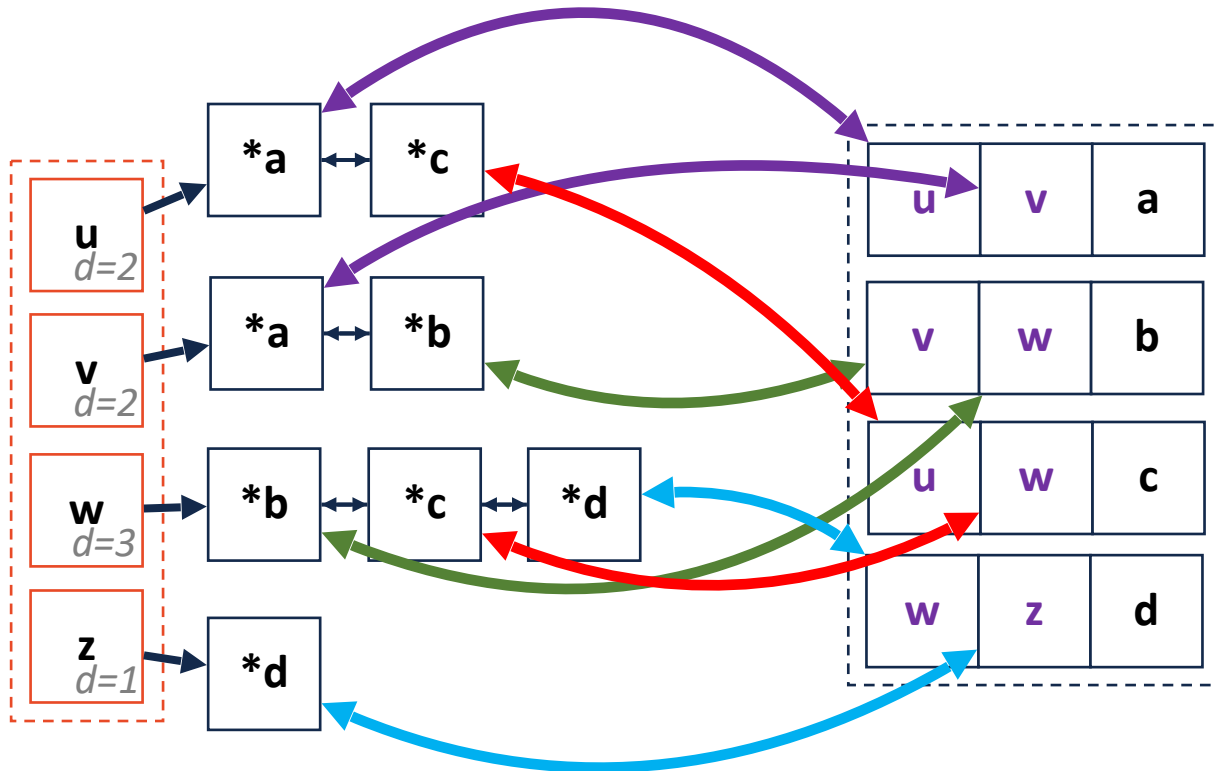
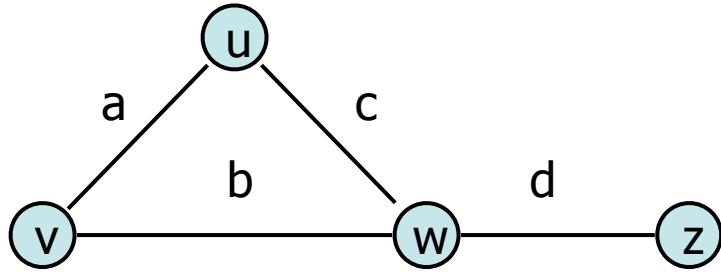
Remove edge from edge list

Remove entry from vertex list

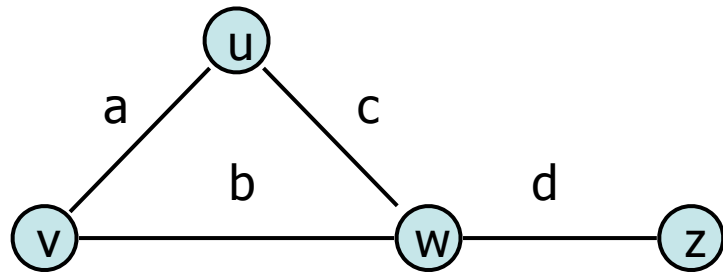


Adjacency List

insertVertex(K key):

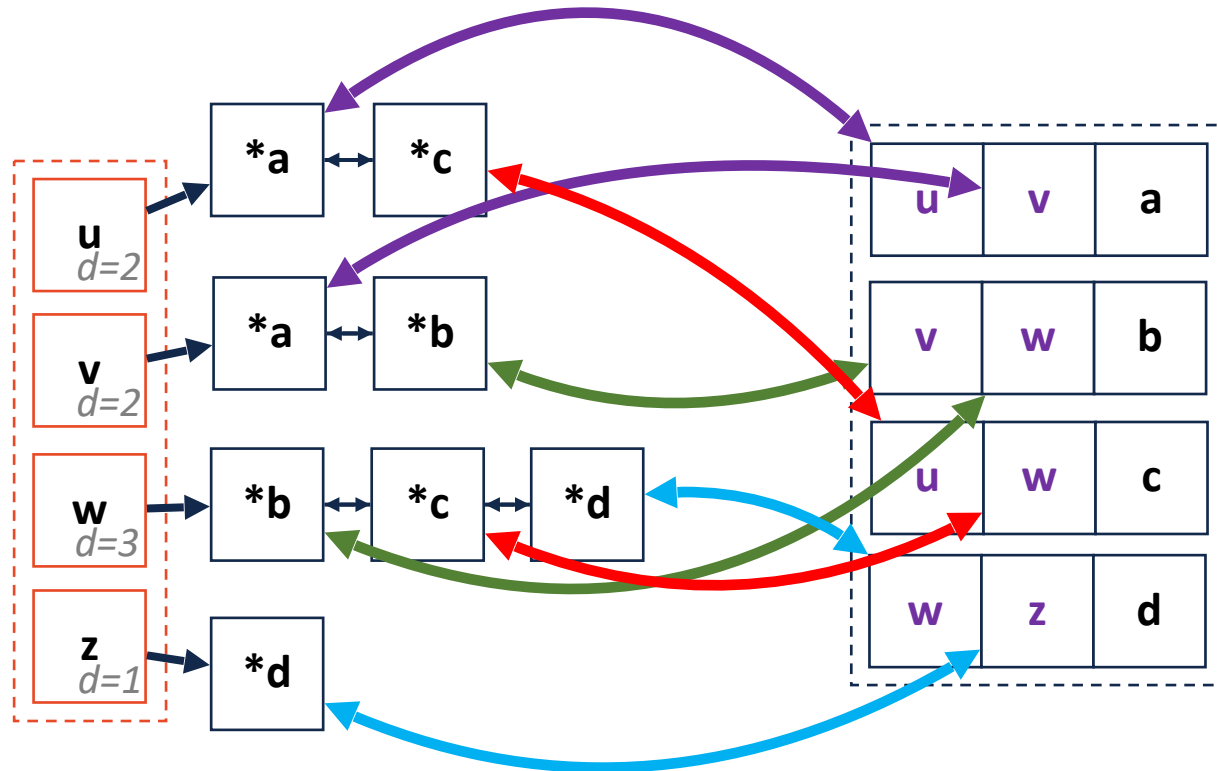


Adjacency List



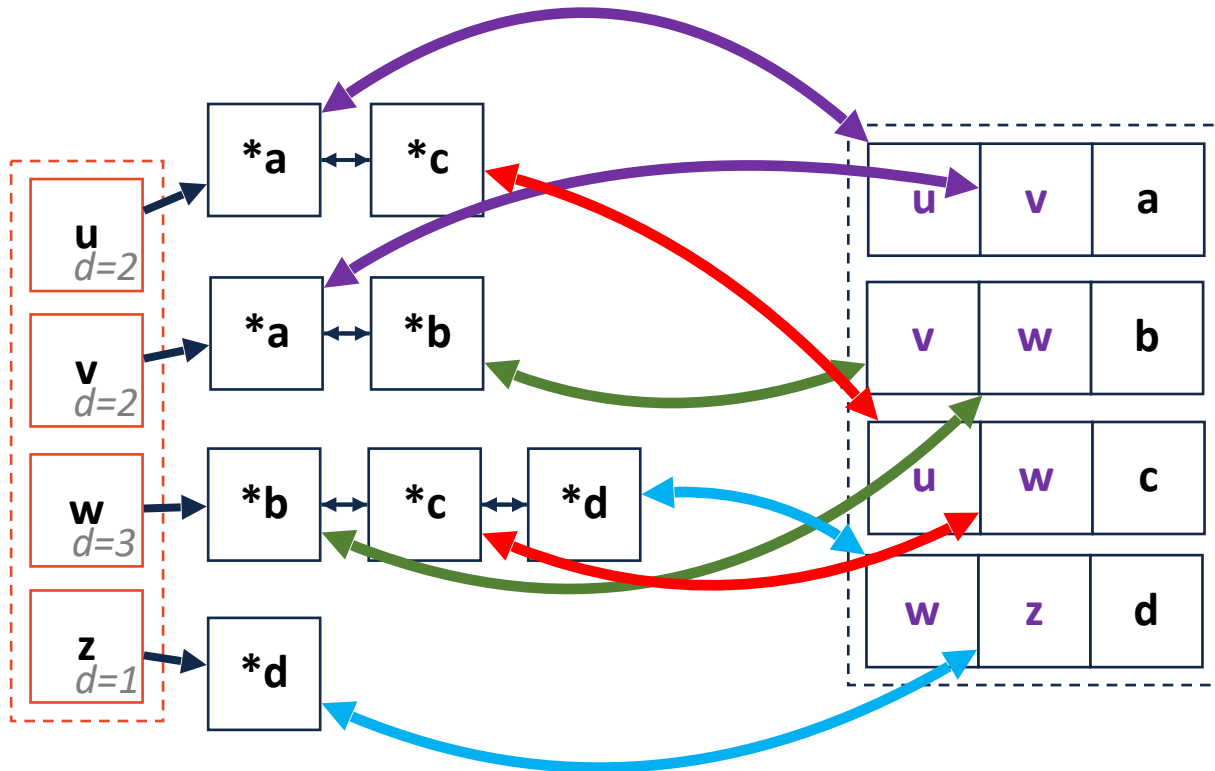
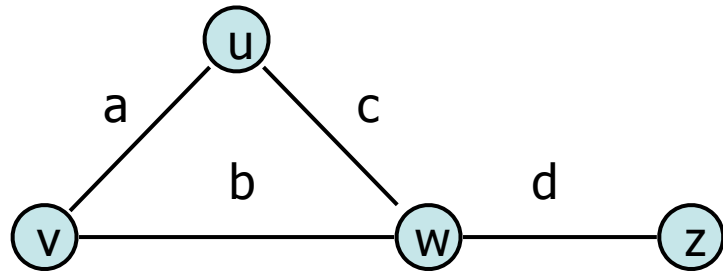
insertVertex(K key):

Add new empty list to vertex list.

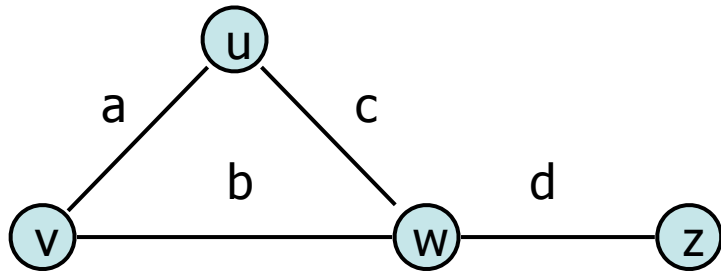


Adjacency List

`removeVertex(Vertex v):`



Adjacency List



removeVertex(Vertex v):

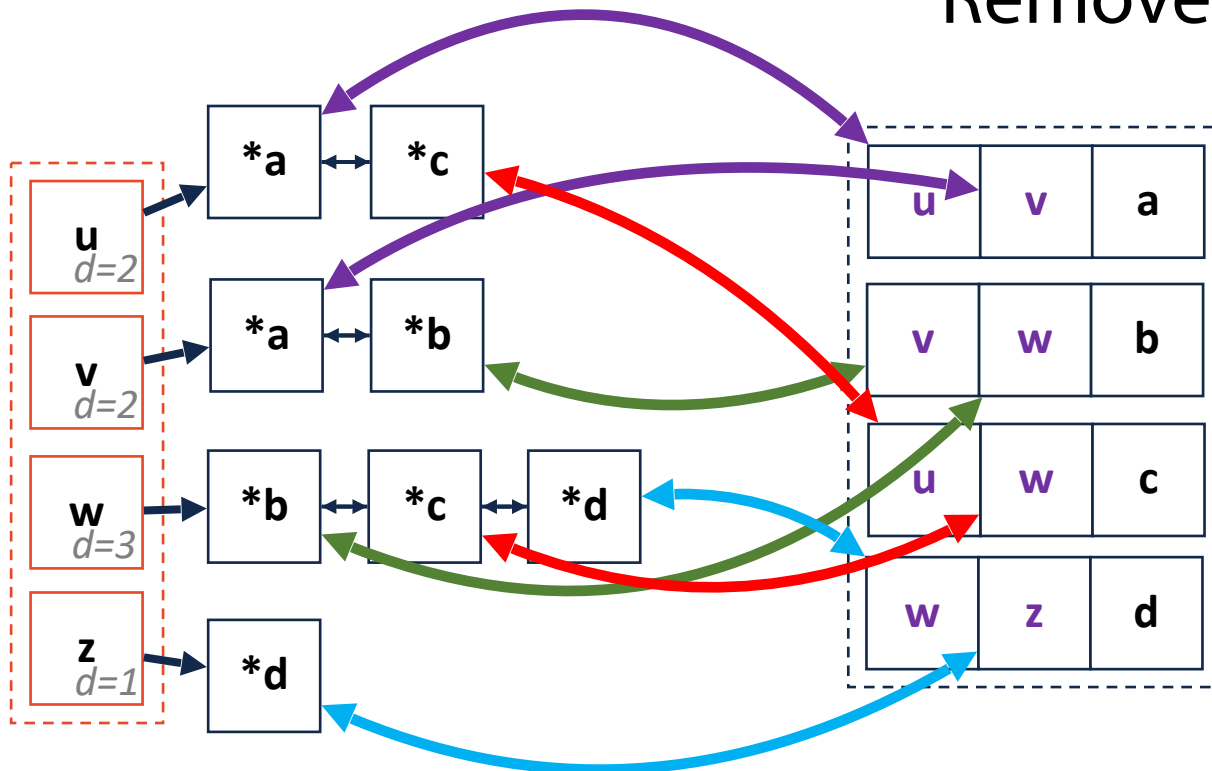
Look up vertex in vertex list

Walk across list

Remove mirrored entry using pointers

Remove edge from edge list

Remove entry from vertex list



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



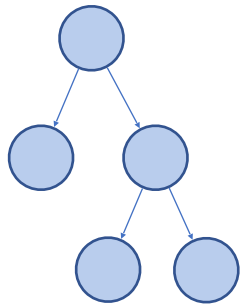
| Expressed as O(f) | Edge List | Adjacency Matrix | Adjacency List |
|-------------------|-----------|------------------|--------------------------------------|
| Space | $n+m$ | n^2 | $n+m$ |
| insertVertex(v) | 1^* | n^* | 1^* |
| removeVertex(v) | $n+m$ | n | $\text{deg}(v)$ |
| insertEdge(u, v)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

Graph Traversals

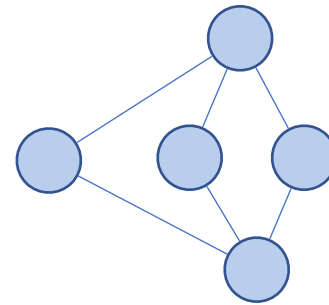
There is no clear order in a graph (even less than a tree!)

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

Tree traversals won't work — lets compare:

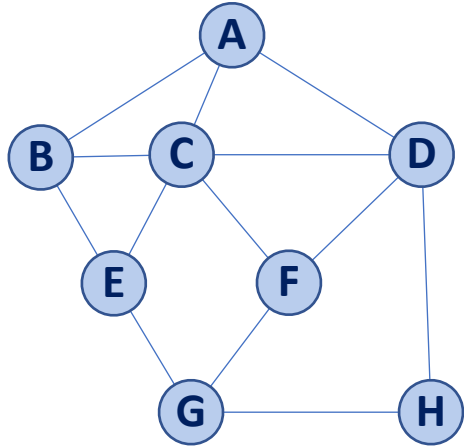


- Rooted
- Acyclic
-

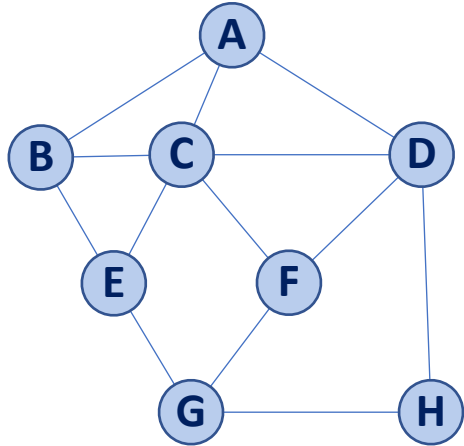


-
-
-

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 |
