Data Structures

Adjacency Matrix Graph Implementation

COMP128 Data Structures

Graph Implementation

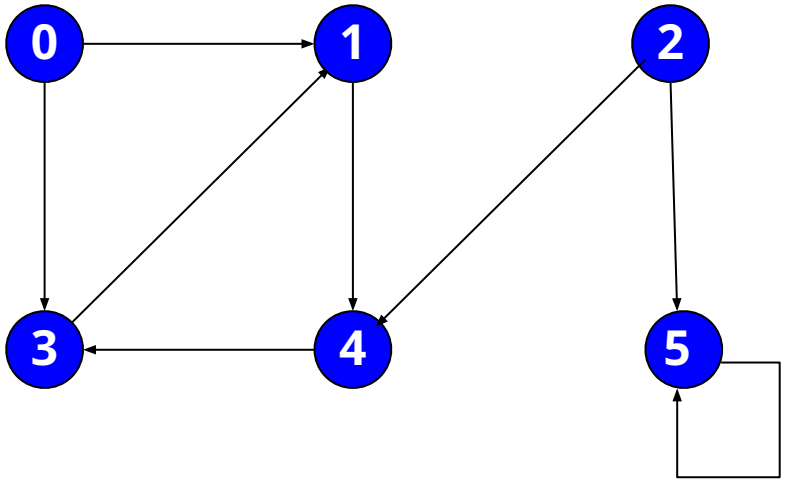
We have the following two different approaches to implement a graph data structure:

- Adjacency List
- Adjacency Matrix

In each representation, we maintain a collection to store the vertices and edges of a graph. However, the representations differ greatly in the way they organize the edges.

Using Adjacency Matrix: Directed Graph

An adjacency matrix uses a two disarray to store vertices and edges.

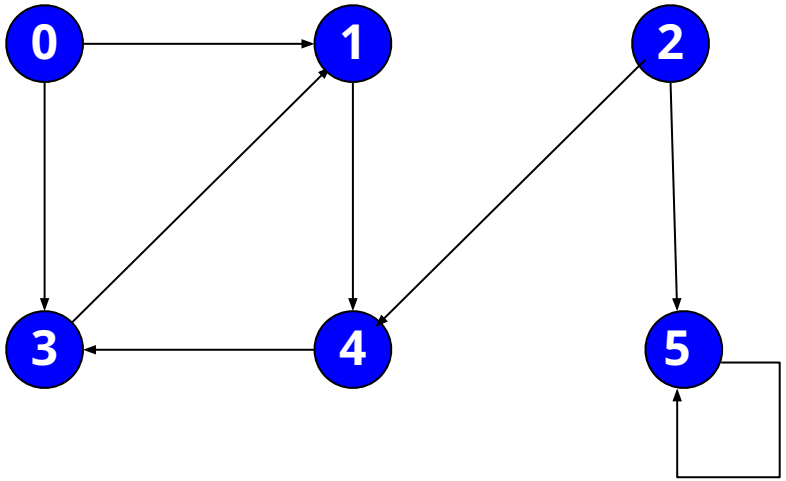


Column

Row	[0]	[1]	[2]	[3]	[4]	[5]
[0]		1.0		1.0		
[1]					1.0	
[2]					1.0	1.0
[3]		1.0				
[4]				1.0		
[5]						1.0

Using Adjacency Matrix: Directed Graph

For a weighted graph, you can store the weights rather than 1.0.

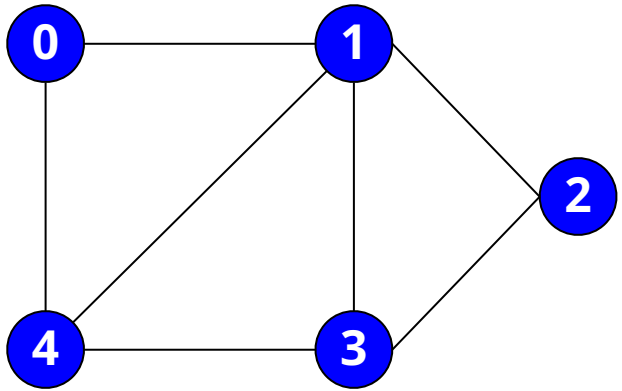


Column

Row	[0]	[1]	[2]	[3]	[4]	[5]
[0]		1.0		1.0		
[1]					1.0	
[2]					1.0	1.0
[3]		1.0				
[4]				1.0		
[5]						1.0

Using Adjacency List: Undirected Graph

If the graph is undirected, you only need to store the bottom half of the matrix.

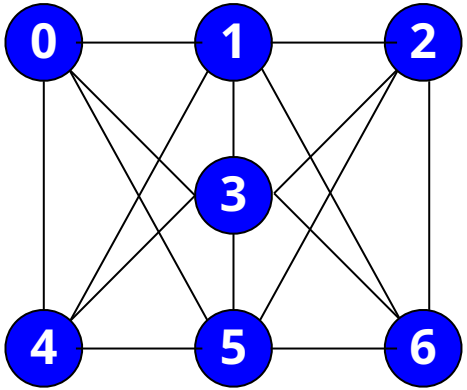


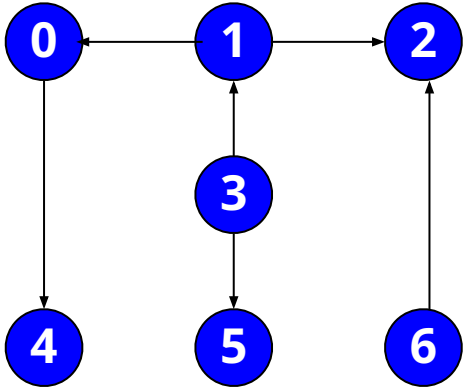
Column

Row	[0]	[1]	[2]	[3]	[4]
[0]		1.0			1.0
[1]	1.0		1.0	1.0	1.0
[2]		1.0		1.0	1.0
[3]		1.0	1.0		
[4]	1.0	1.0		1.0	

Quick Practice

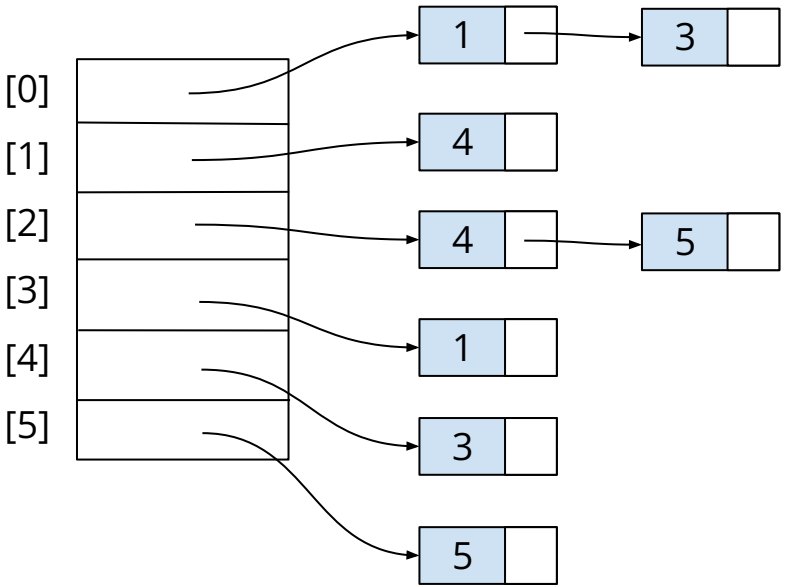
Draw on a piece of paper, the adjacency matrix structure for the following graphs. Then check with the person next to you.





Implementation Efficiency

Discuss with the person next to you:
When would you use an adjacency list vs. an adjacency matrix?



Column

Row	[0]	[1]	[2]	[3]	[4]	[5]
[0]		1.0		1.0		
[1]					1.0	
[2]					1.0	1.0
[3]		1.0				
[4]				1.0		
[5]						1.0

Implementation Efficiency

When would you use an adjacency list vs. an adjacency matrix?

This depends on whether the graph is sparse (few edges compared to the number of vertices) or dense.

For a sparse graph the adjacency list has better storage efficiency.

Row \ Column	[0]	[1]	[2]	[3]	[4]	[5]
[0]		1.0		1.0		
[1]					1.0	
[2]					1.0	1.0
[3]		1.0				
[4]				1.0		
[5]						1.0

Implementation Efficiency

For sparse graphs, the adjacency list will also have better timing performance for most algorithms. Why?

Implementation Efficiency

For sparse graphs, the adjacency list will also have better timing performance for most algorithms. Why?

Give a graph, G = (V, E):

Many algorithms are in the form:

1. For each vertex u in the graph:
    2. For each vertex v adjacent to u:
        3. Do something with the edge (u, v)

For an adjacency list this is O(|E|) because step 1 is O(|V|) and step 2 is O($|E_u|$).
For an adjacency matrix this is O($|V|^2$) because step 2 is O(|V|).
In a sparse graph, |E| is much less than $|V|^2$

In-class Activity

Adjacency Matrix Graph Activity