

Graph Data Structures

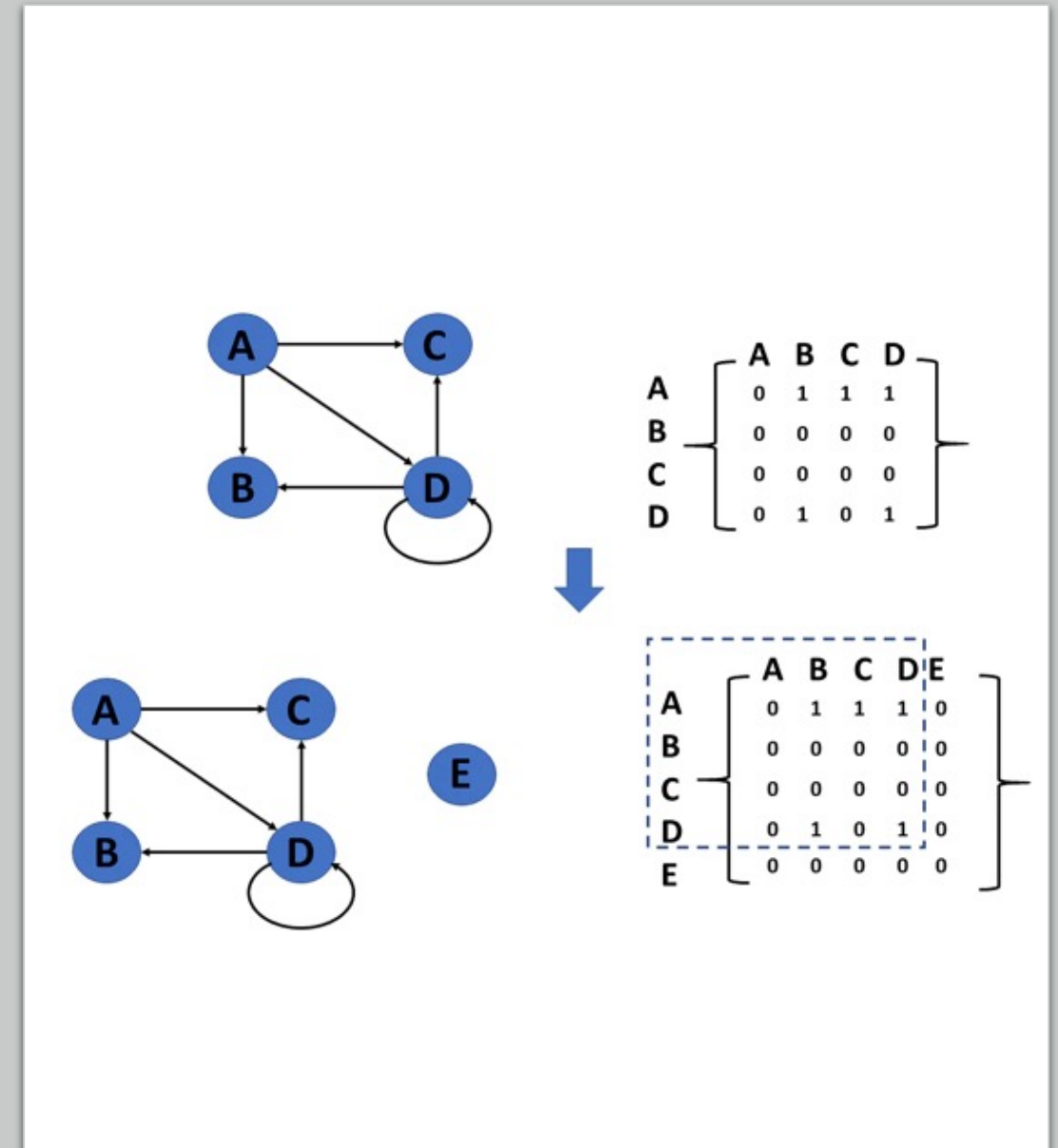
Instructor: Meng-Fen Chiang

COMPSCI: WEEK 9.2



OUTLINE

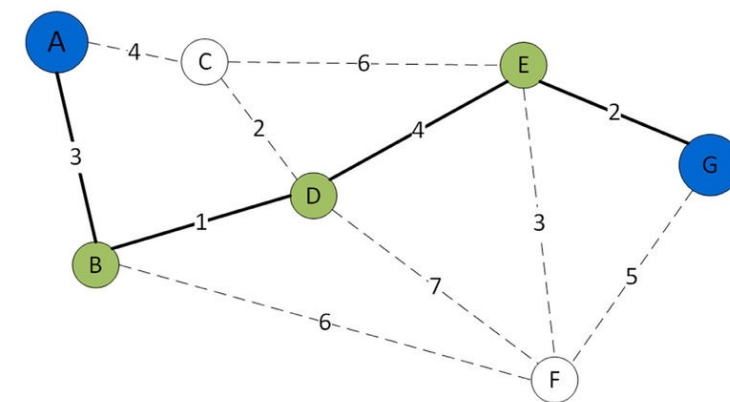
- Graph Data Structures
- Which to use?
- Graph Operations



Solving Graph Problems

There are lots of different problems defined on graphs we might like a computer to solve.

- List all nodes in order of their distance to a fixed node v .
- What is the nearest node to v with some property?
- Is there a path from u to v .
- What is the shortest path between u and v ?
- How many different paths are there between u and v .



Storing a (Di)graph

- Fundamentally, computers don't know what graphs are.
- All computers really know is a list of numbers (e.g. array).
- How would you write a graph algorithm in your favourite programming language?
- We need some ways to **represent** a graph which computers can make sense of.
- There are basically two types of representations.

Digraphs: Computer Representation

CONVENTION: For a digraph G of order n we label nodes $0, 1, \dots, n - 1$:

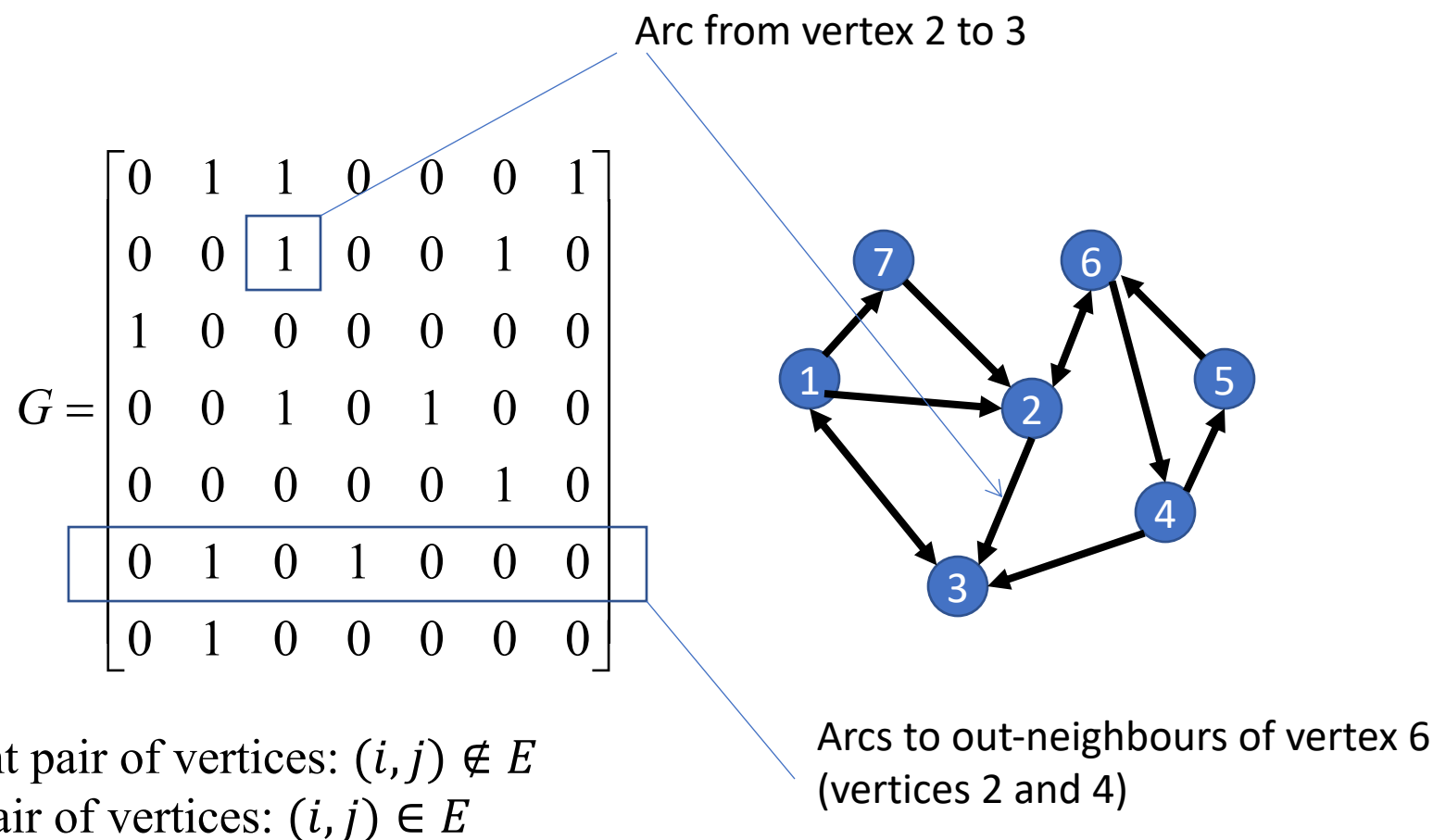
The **adjacency matrix** of G :

The $n \times n$ boolean matrix (often encoded with 0's and 1's) such that its entry (i, j) is true if and only if there is an arc (i, j) from the node i to node j .

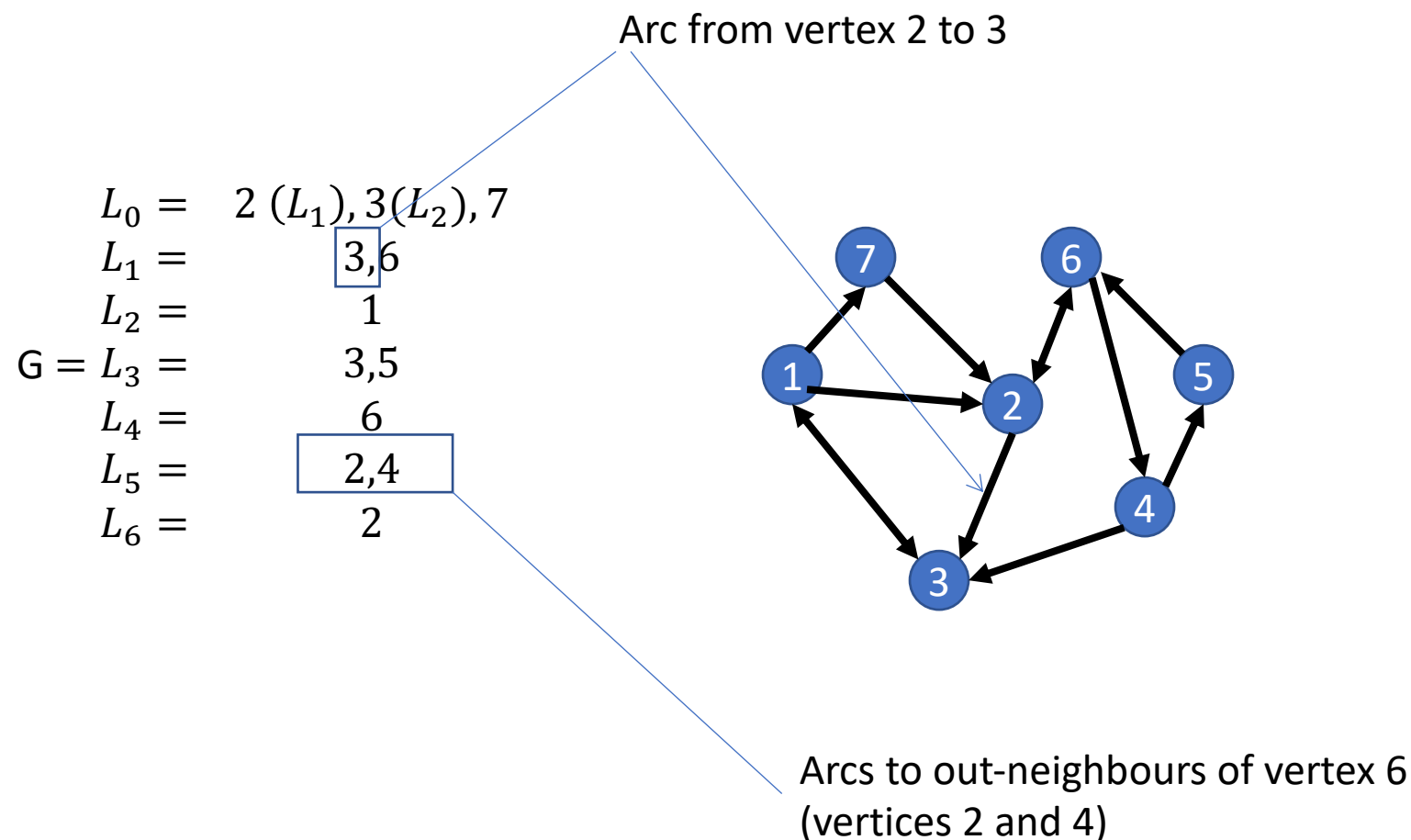
An **adjacency list** of G :

A list of n lists, L_0, \dots, L_{n-1} , such that the list L_i contains all nodes of G that are adjacent to the node i .

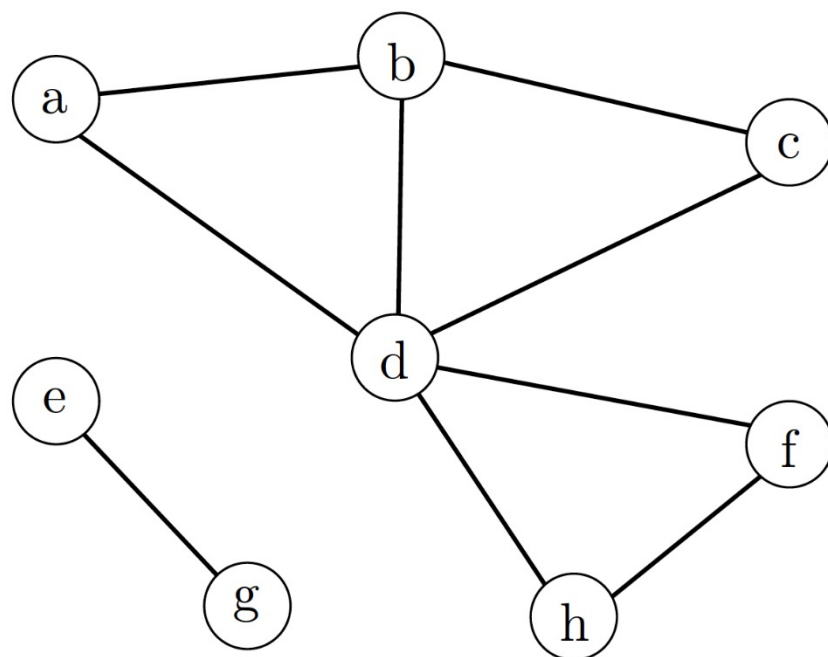
Example: Adjacency Matrix of a Digraph



Example: Adjacency List of a Digraph



Example: Adjacency List of a Graph



Graph $G = (V, E)$

symbolic	numeric
$0 = a: b d$	1 3
$1 = b: a c d$	0 2 3
$2 = c: b d$	1 3
$3 = d: a b c f h$	0 1 2 5 7
$4 = e: g$	6
$5 = f: d h$	3 7
$6 = g: e$	4
$7 = h: d f$	3 5

Numeric node labels in adjacency lists can omitted.

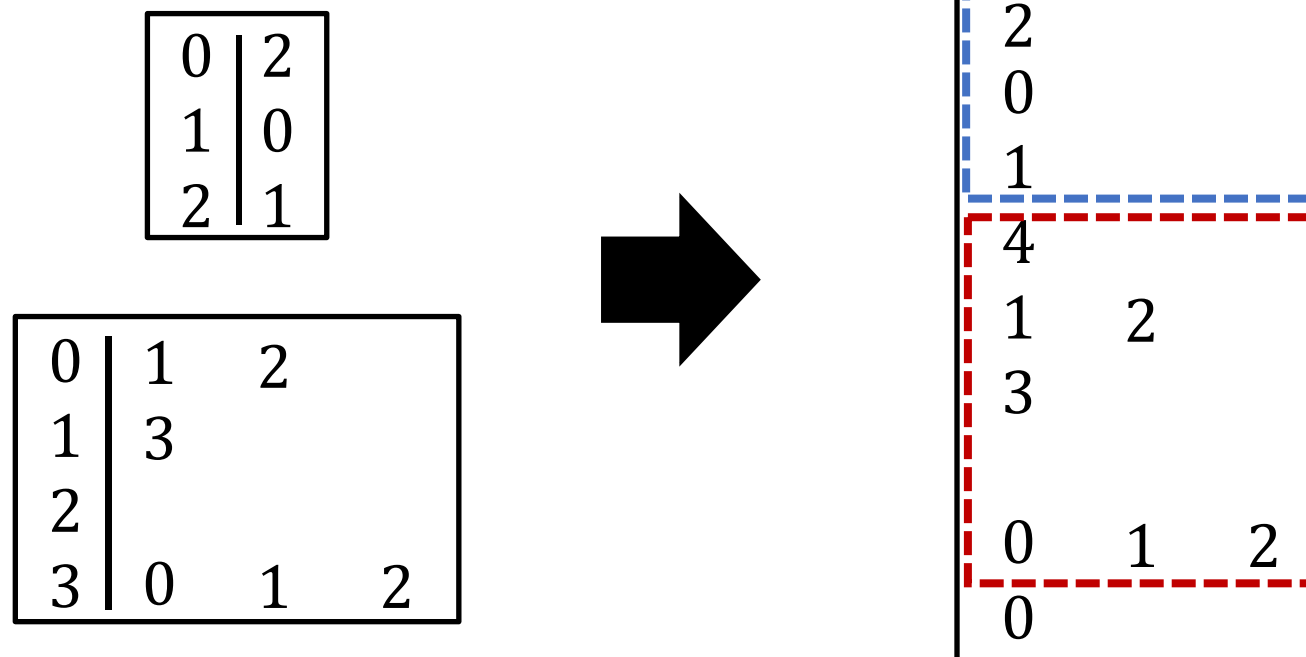
Representing multiple graphs in a single file

We can store several digraphs one after the other in a single file as follows:

- Use one line to store the **order** at the beginning of each digraph.
- If the order is n then the next n lines give the **adjacency matrix** or **adjacency lists** representation of the digraph. If we use numeric labels, node labels can be omitted.
- The end of the file is marked with a line denoting a digraph of order 0.

Example: Representing multiple graphs in a single file

- The two digraphs on the left could be put in a single file:



Other structures to represent graphs

We have already seen a way to represent a binary heap as the array. Here is a way of storing a general tree in an **array**.

- A general rooted tree of n nodes can be stored in array *pred* of size n .
- *pred*[i] is the parent of node i .
- The root has no parent, so assign it *null* or 1 if we number nodes from 0 to $n-1$ in the usual way.
- This is a form of adjacency lists, using in-neighbours instead of out-neighbours.

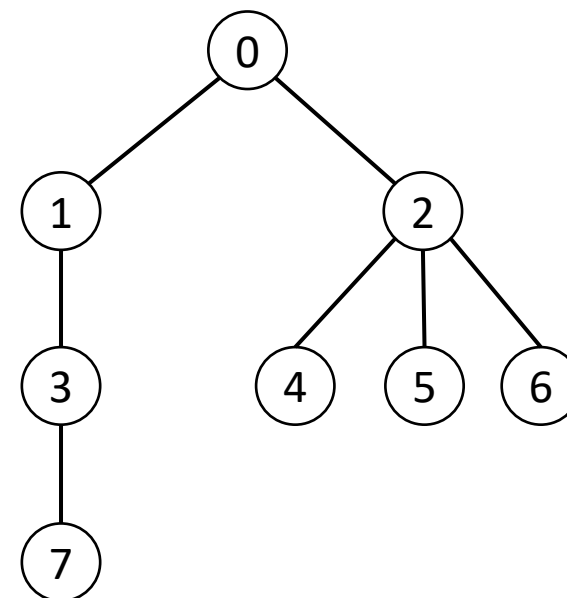
Example 21.8

- Draw the tree represented by the array

pred = [-1, 0, 0, 1, 2, 2, 2, 3]

0 1 2 3 4 5 6 7

The parent of 1



Implementation of digraph ADT

- An **adjacency matrix** is simply a matrix which is an **array of arrays**.
- **Adjacency lists** are a **list of lists**.
- There are several ways in which a list can be implemented, for example by an array, or singly- or doubly-linked lists using pointers.
- Depending on an implementation certain operations may have different running time.

Adjacency List or Matrix?

- Depends on three factors governing storage requirement and performance:
- The **order** of the (di)graph $|V(G)|$: the storage space we require for an adjacency matrix is $\Theta(n^2)$.
- The **size** of the (di)graph $|E(G)|$: the storage space we require for an adjacency list is $\Theta(n^2)$. However, in most practical cases, the graphs are sparse. E.g., if the in-/out-degree is limited, the space requirement is only $\Theta(n + e)$, where e is the number of edges.
- What we want to do with the graph (operations). This requires a bit more discussion.

Elementary Graph Operations

1. Does the arc (u,v) exist?
2. What is the outdegree of vertex u ?
3. What is the indegree of vertex u ?
4. Add an arc between two vertices u and v .
5. Delete the arc between vertices u and v .
6. Add a node to the (di)graph.
7. Delete a node from the (di)graph.

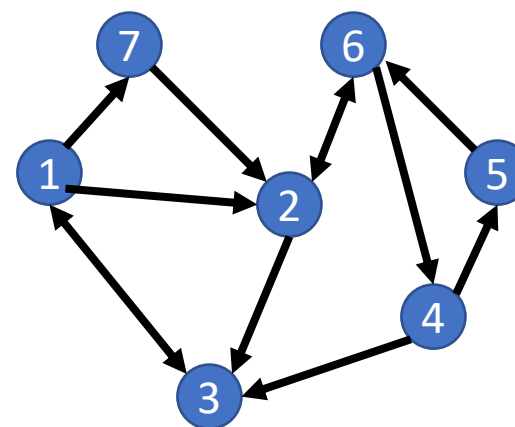
Q1: Does the arc (i, j) exist?

- In the adjacency matrix, we need to find the entry in row i , column j . This operation is equivalent to accessing an element in an array: $\Theta(1)$.
- In the adjacency list, the requires looking up the list ($\Theta(1)$) and then checking whether j is on that list. If d is the out-degree of vertex i , this will take $\Theta(d)$ in the worst case. The overall worst case scenario is thus $\Theta(d)$.
- The matrix wins!

Q1: Matrix Example

Arc from vertex 6 to 4

$$G = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

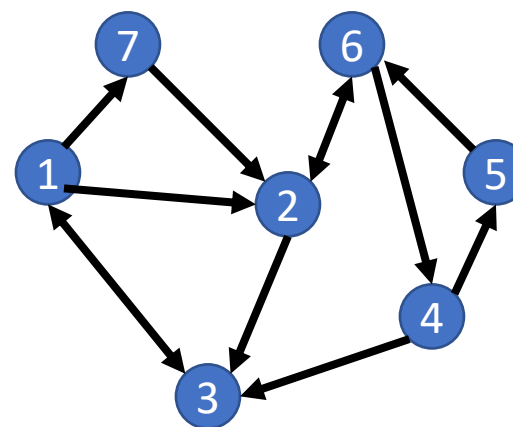


Arcs to out-neighbours of vertex 6
(vertices 2 and 4)

Q1: List Example

Arc from vertex 6 to 4

$L_0 = 2(L_1), 3(L_2), 7$
 $L_1 = 3, 6$
 $L_2 = 1$
 $G = L_3 = 3, 5$
 $L_4 = 6$
 $L_5 = \boxed{2, 4}$
 $L_6 = 2$



Arcs to out-neighbours of vertex 6
 (vertices 2 and 4)

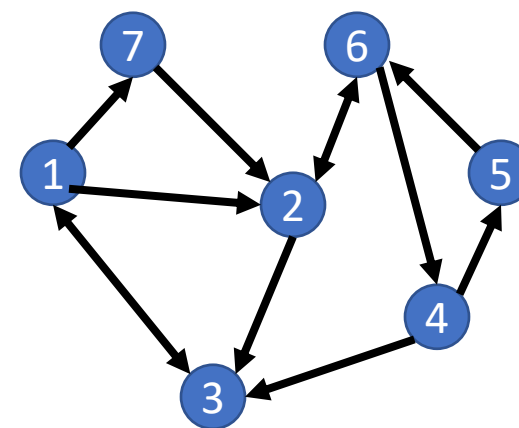
Q2: What is the out-degree of vertex i ?

- In the adjacency matrix, we need to scan row i , and count the number of 1's we find. This operation is $\Theta(n)$.
- In the adjacency list, this requires looking up the size of the list L_i , which takes $\Theta(1)$ (if the list is properly implemented).
- The list wins!
- Note: This problem is equivalent to finding all the out-neighbours of i .

Q2: Matrix Example

Out-degree vertex 2

$$G = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

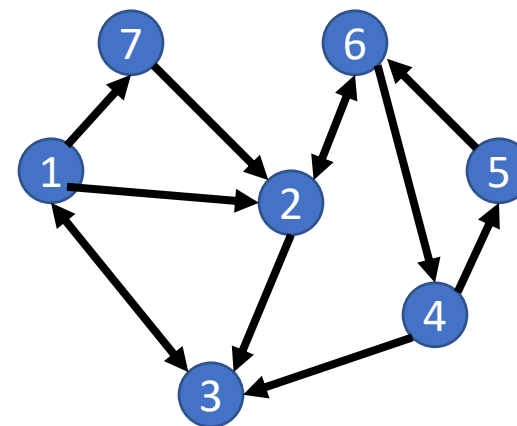


Scan array in row 2 and count "1"

Q2: List Example

Out-degree vertex 2

$L_0 = 2(L_1), 3(L_2), 7$
 $L_1 = 3, 6$
 $L_2 = 1$
 $G = L_3 = 3, 5$
 $L_4 = 6$
 $L_5 = 2, 4$
 $L_6 = 2$



Get the size of these sequence

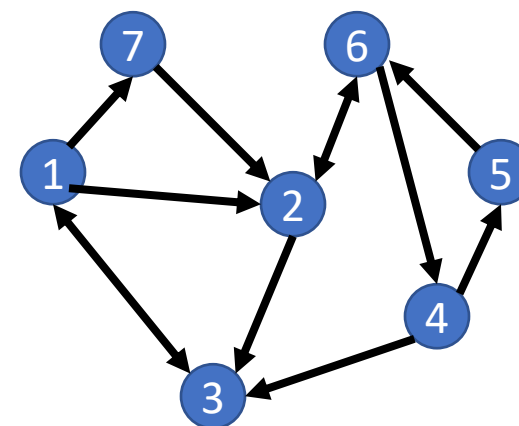
Q3: What is the in-degree of vertex i ?

- In the adjacency matrix: Essentially the same operation as for the out-degree, except that we scan a column for u rather than a row: $\Theta(n)$.
- In the adjacency list, this requires checking for all $j \neq i$ whether i is in the sequence: L_j . which takes $\Theta(n + e)$. Note: need to check each L_j even if the sequence is empty.
- The matrix wins – at least for dense (di)graphs!

Q3: Matrix Example

In-degree vertex 2

$$G = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

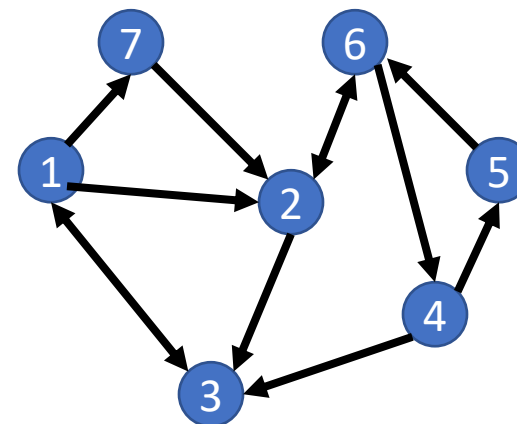


Scan array for column 2 and count "1"

Q3: List Example

In-degree vertex 2

$L_0 = 2 (L_1), 3(L_2), 7$
 $L_1 = 3, 6$
 $L_2 = 1$
 $G = L_3 = 3, 5$
 $L_4 = 6$
 $L_5 = 2, 4$
 $L_6 = 2$



Scan all the sequences and count the occurrence of “2”

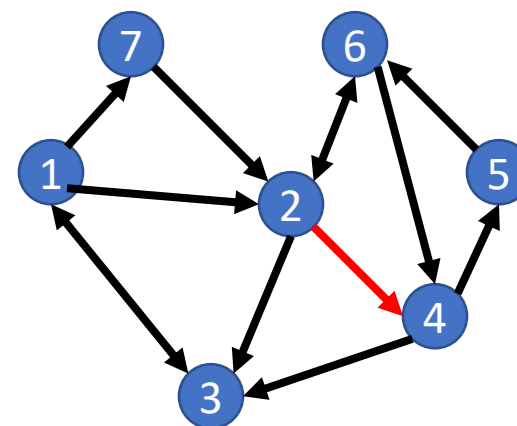
Q4: Adding an arc from i to j

- For the adjacency matrix: change value of matrix element (i, j) . This is $\Theta(1)$.
- For the adjacency list: Insert j into list i . This is also $\Theta(1)$.
- Matrix and list perform equally well here.

Q4: Matrix Example

Add arc (2,4)

$$G = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

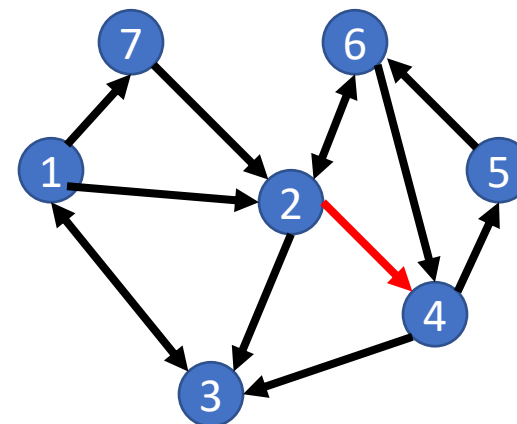


Put "1" in the second array in position 3

Q4: List Example

Adding an arc (2,4)

$L_0 = 2(L_1), 3(L_2), 7$
 $L_1 = 3, 6, 4$
 $L_2 = 1$
 $G = L_3 = 3, 5$
 $L_4 = 6$
 $L_5 = 2, 4$
 $L_6 = 2$



Add 4 at the end of the sequence for node 2

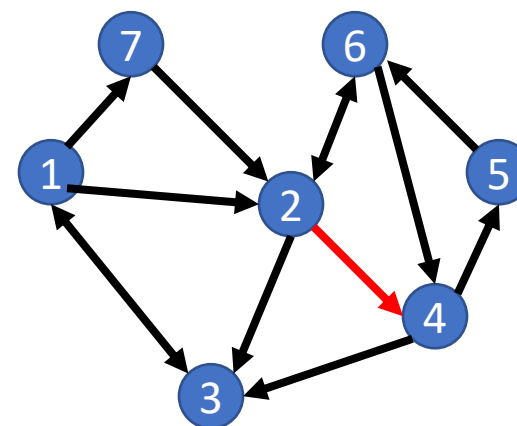
Q5: Deleting an arc from i to j

- This problem is pretty much identical to the that of answering the question whether the arc exists
- In the adjacency matrix, rather than returning the value of (i, j) , we set it to 0 (false). So we have $\Theta(1)$.
- In the adjacency list, we delete the appropriate list entry – constant time to locate the node in the list and up to d searches in the sub-list containing the arcs. Therefore, $\Theta(d)$.
- Matrix wins!

Q5: Matrix Example

Delete arc (2,4)

$$G = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

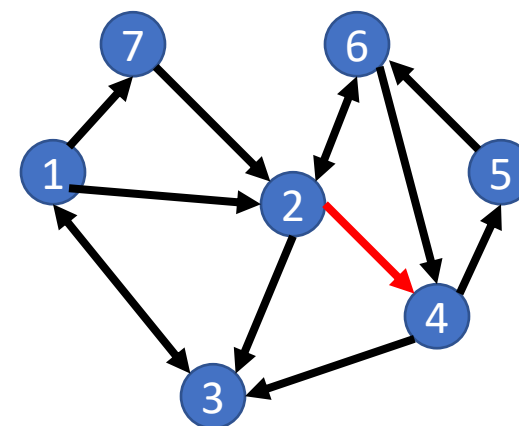


Put "0" in the second array in position 3

Q5: List Example

Delete arc (2,4)

$L_0 = 2(L_1), 3(L_2), 7$
 $L_1 = 3, 6, 4$
 $L_2 = 1$
 $G = L_3 = 3, 5$
 $L_4 = 6$
 $L_5 = 2, 4$
 $L_6 = 2$



Scan the sequence for node 2 until you find node 4 and then delete it

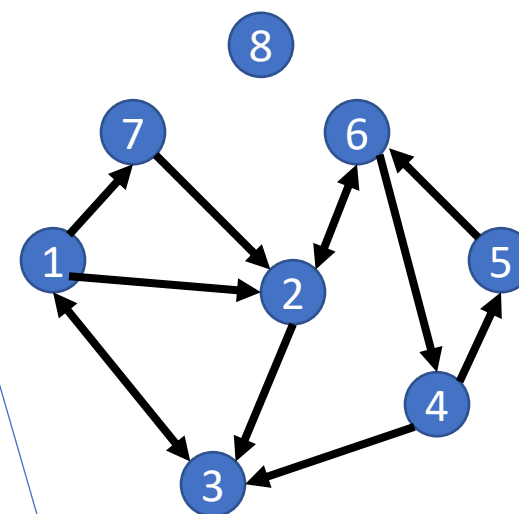
Q6: Adding a vertex to the (di)graph

- In the adjacency matrix: Need to add $2n + 1$ entries (one row with n entries at the bottom and then one column with $n + 1$ entries at the end): $\Theta(n)$
- Think of a C/C++ implementation of this; and see if you can achieve $\Theta(n)$.
- In the adjacency list: Add one entry: $\Theta(1)$
- List wins!

Q6: Matrix Example

Adding vertex 8

$$G = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$



Add 7 entries for row 8

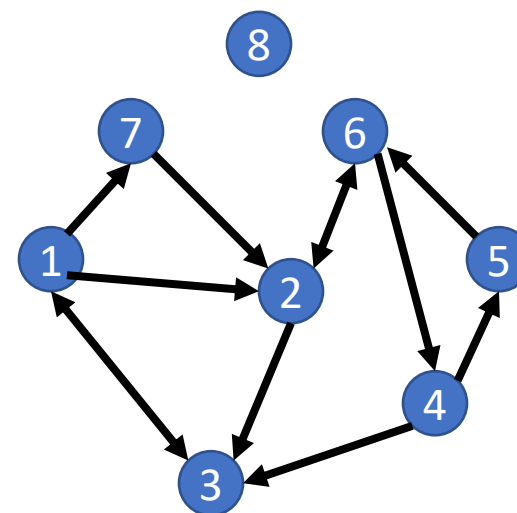
Add 8 entries for column 8

Q6: List Example

Adding vertex 8

$L_0 = 2(L_1), 3(L_2), 7$
 $L_1 = 3, 6, 4$
 $L_2 = 1$
 $G = L_3 = 3, 5$
 $L_4 = 6$
 $L_5 = 2, 4$
 $L_6 = 2$

$L_7 =$



Create a new sequence for node 8

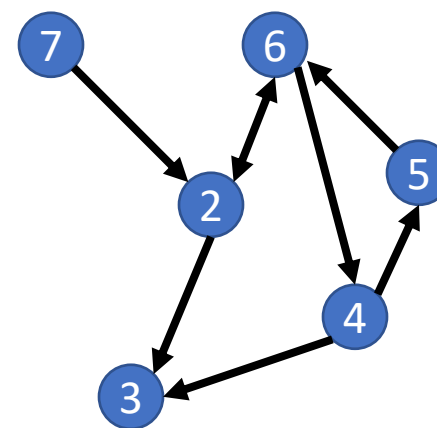
Q7: Deleting a vertex from the (di)graph

- Note that the vertex may have arcs attached!
- Matrix: Remove a row and a column, shifting up to $(n - 1)$ elements up and, up to $(n - 1)$ elements to the left: $\Theta(n^2)$. A simpler way to do this is to copy each required element of the original matrix to a new matrix of size $(n - 1)^2$.
- List: Need to look at all n entries and need to check their sequences for the presence of the node that needs to be removed. The total number of sequence entries is the number of arcs, e , so the total time complexity here is $\Theta(n + e)$.
- In sufficiently sparse graphs, the list wins, but as $e \leq n(n - 1)$ for digraphs and $e \leq n(n - 1)/2$ for graphs, this isn't necessarily always so!

Q7: Matrix Example

Delete vertex 1 (worst case)

$$G = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

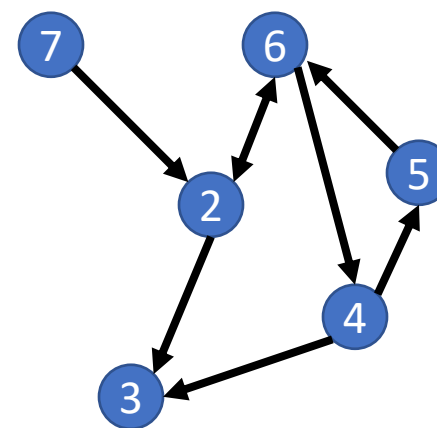


Need to shift 6 rows up (n-1)

Q7: Matrix Example

Delete vertex 1 (worst case)

$$G = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

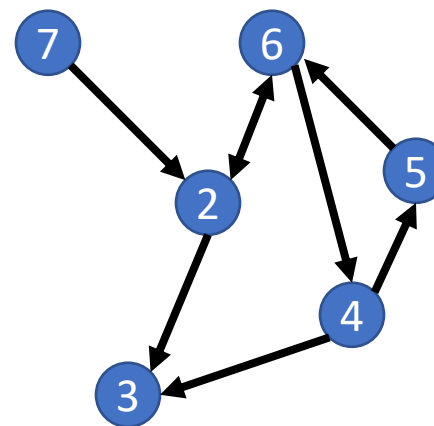


Need to shift 6 columns left (n-1)

Q7: List Example

Delete vertex 1

$L_0 =$	2 (L_1), 3(L_2), 7
$L_1 =$	3, 6, 4
$L_2 =$	1
$G = L_3 =$	3, 5
$L_4 =$	6
$L_5 =$	2, 4
$L_6 =$	2



Drop sequence for node 1; 3 no longer
has an edge to 1

Digraph Operations w.r.t. Data Structures

Operation	Adjacency Matrix	Adjacency Lists
arc (i, j) exists?	is entry (i, j) 0 or 1	find j in list i
out-degree of i	scan row and sum 1's	size of list i
in-degree of i	scan column and sum 1's	for $j \neq i$, find i in list j
add arc (i, j)	change entry (i, j)	insert j in list i
delete arc (i, j)	change entry (i, j)	delete j from list i
add node	create new row/column	add new list at end
delete node i	Delete row/column i and shuffle other entries	delete list i and for $j \neq i$, delete i from list j

Adjacency Lists / Matrices: Comparative Performance

Operation	array/array	list/list
arc (i, j) exists?	$\Theta(1)$	$\Theta(d)^*$
out-degree of i	$\Theta(n)$	$\Theta(1)$
in-degree of i	$\Theta(n)$	$\Theta(n + e)$
add arc (i, j)	$\Theta(1)$	$\Theta(1)$
delete (i, j)	$\Theta(1)$	$\Theta(d)$
add node	$\Theta(n)$	$\Theta(1)$
delete node i	$\Theta(n^2)$	$\Theta(n + e)$

Space Requirements

- The adjacency matrix representation requires $\Theta(n^2)$ storage as we simply need a matrix of n^2 bits.
- The adjacency list space requirement is $\Theta(n + e \log n)$.
 - we need n lists and the combined length of all the lists is e .
 - Node numbers require more than one bit of storage each; the number k uses about $\Theta(\log k)$ bits.

Rule of Thumb

- Use adjacency **matrix** for **small, dense** (di)graphs for which we wish to test for the **existence of arcs, find the in-degree of vertices**, and/or delete arcs.
- Use adjacency **list** for **large, sparse** (di)graphs for which we need to **compute out-degree** and **add** and/or **delete vertices**.

SUMMARY

- Graph Data Structures
 - Adjacency matrix
 - Adjacency list
- 7 Elementary Graph Operations
 - Does the arc (u,v) exist?
 - What is the outdegree of vertex u?
 - What is the indegree of vertex u?
 - Add an arc between two vertices u and v.
 - Delete the arc between vertices u and v.
 - Add a node to the (di)graph.
 - Delete a node from the (di)graph
- Performance Comparison

