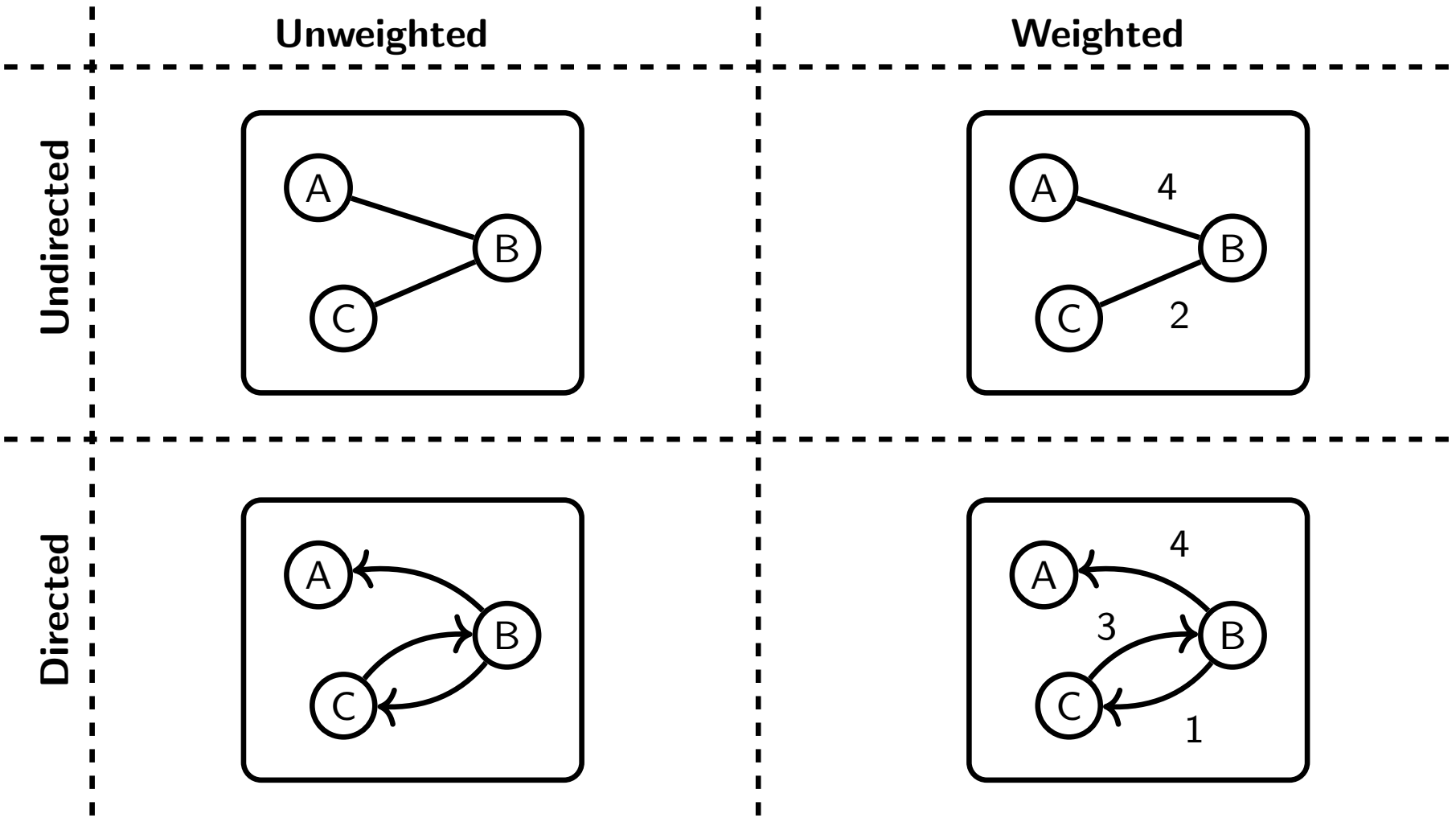
Graphs and graph algorithms

Graph Types

A **graph** is formed of a (finite) set of vertices/nodes and a set of edges between them. We distinguish four types of graphs:



Examples

Undirected unweighted graph:

- vertices = registered people on Facebook
- edges = friendships between people (it is mutual!)

Directed unweighted graph:

- vertices = registered people on Twitter
- edges = who is following who

Undirected weighted graph:

- vertices = train stops/stations
- edges = rail lines connecting train stops with distance

Directed weighted graph:

- vertices = bank accounts
- edges = bank transfers with amounts transferred

Graph Concepts

- In maths, we tend to use the term **vertex**, in computer science, we use both **node** and **vertex** interchangeably
- If the graph is undirected, an edge between nodes `u` and `w` can be thought of as having two edges `u` $\rightarrow$ `w` and `w` $\rightarrow$ `u`.
- A directed graph is often called a **digraph**
- A graph is called **simple** if
  1. it has no self loops, i.e. edges connected at both ends to the same node, and
  2. it has no more than one edge between any pair of nodes
- A **path** is a sequence of vertices $v_1, v_2, \ldots, v_n$ such that v_i and v_{i+1} are connected by an edge for all $1 \leq i \leq n-1$.
- A **cycle** is a non-empty path whose first vertex is the same as its last vertex.
- A path is **simple** if no vertex appears on it twice (except for a cycle, where the first and last vertex may be the same)

Graph Concepts

- An undirected graph is **connected** if every pair of vertices has a path connecting them.
- A directed graphs is:
  - **weakly connected** if for every two vertices A and B there is either a path from A to B or a path from B to A.
  - **strongly connected** if there are paths leading both ways.
- A **tree** can be viewed as a simple connected graph with no cycles and one node identified as a root
- A graph, unlike a tree, does not have a **root** from which there is a unique path to each vertex, so it does not make sense to speak of parents and children in a graph.
- Two vertices A and B connected by an edge e are called **neighbours**, and e is said to be **incident** to A and B.
- Two edges with a vertex in common (e.g., one connecting A and B and one connecting B and C) are said to be **adjacent**.

Graph represented as an Adjacency Matrix

Assume that graph's vertices are numbered $V = \{0, 1, 2, ..., n-1\}$.

Adjacency matrix `G` is a two-dimensional array/matrix $n \times n$ where each cell `G[v][w]` contains information about the connection from vertex v to vertex w

Unweighted graphs:

- `G[v][w]` $= 1$ if there is an edge going from `v` to `w`
- `G[v][w]` $= 0$ if there is no such edge

Weighted graphs:

- `G[v][w]` $=$ weight of the edge going from `v` to `w`
- `G[v][w]` $= \infty$ if there is no such edge*
- `G[v][v] = 0` *

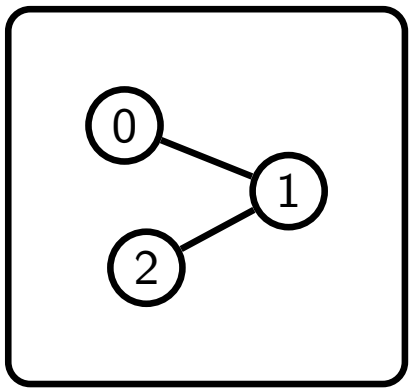
Remark: The graph is undirected if `G[v][w] = G[w][v]` for all vertices `v` and `w`.

---

*other values possible depending on application

Example: Adjacency matrix

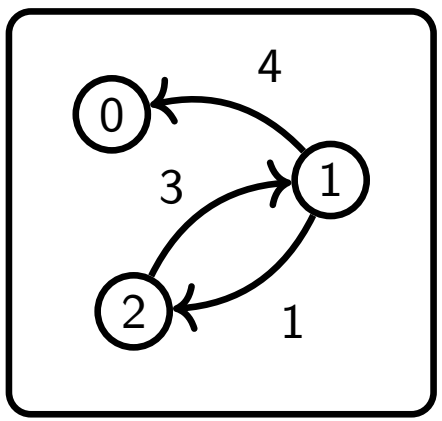
Unweighted undirected:



$$\texttt{G} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

For example: `G[2][0] = 0` and `G[2][1] = 1`.

Weighted directed:



$$\texttt{G} = \begin{pmatrix} 0 & \infty & \infty \\ 4 & 0 & 1 \\ \infty & 3 & 0 \end{pmatrix}$$

For example: `G[2][0] = infty` and `G[2][1] = 3`.

Graph represented as Adjacency Lists

To represent a graph on vertices $V = \{0, 1, 2, \ldots, n-1\}$ by *adjacency lists* we have an array `N` of n-many linked lists (one list for every vertex).

Unweighted:

- `N[v]` is the list of *neighbours* of `v`.

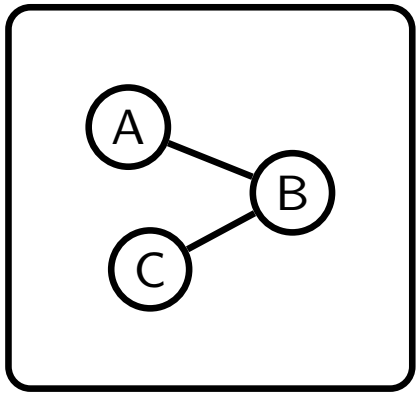
  (`w` is a neighbour of `v` if there is an edge `v` $\rightarrow$ `w`)

Weighted:

- `N[v]` is the list of *neighbours* of `v` together with the weight of the edge that connects them with `v`.

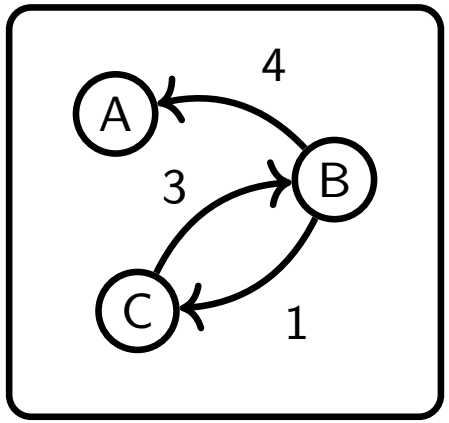
Example: adjacency lists

Unweighted undirected:



N[v]	neighbours
A	B
B	A, C
C	B

Weighted directed:

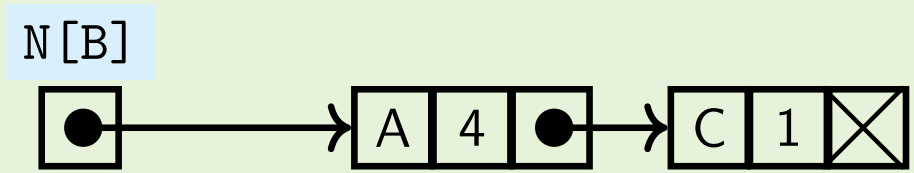


N[v]	neighbours & weights
A	
B	(A, 4), (C,1)
C	(B, 3)

We said that representing a graph by adjacency lists means that we will have an array `N` of n-many linked list (where n is the number of vertices). Then, for example, `N[2]` stores the address of the head of the linked list of all neighbours of the `2` nd vertex. If we name our vertices by letters `A`, `B`, `C`, for example, we need to find a way to assign indexes of the array `N` to the letters `A`, `B`, `C`. One way to do this is to use hash tables.

However, in the example given here, we don't care how this is done. We assume that we have lists of neighbours stored in `N[A]`, `N[B]`, `N[C]`.

In the weighted case, `N[B]` also stores the weights of the edges:



But instead of drawing this we just say that `N[B]` stores the list (A, 4), (C,1).

Comparison of those two methods

Set n = the number of vertices, m = the total number of edges.

	Adjacency matrix	**Adjacency lists**
Checking if there is an edge `v` $\rightarrow$ `w`:	Reading `G[v][w]` (which is in $O(1)$)	Checking if `w` is in in the list `N[v]`
Allocated space:	n arrays of size n $= O(n \times n)$ space	n linked lists storing m edges in total $= O(n + m)$ space
Traversing `v`'s neighbours:	Traversing all `G[v][0]`, `G[v][1]`, .., `G[v][n-1]`. $= O(n)$ time	Traversing only the linked list `N[v]`

In the third case (with adjacency lists) we only traverse the actual neighbours of `v`. This is better whenever the graph is **sparse** (= not dense), that is, if there are relatively few edges.

A graph is **sparse** if it has few edges relative to its number of vertices, e.g. $\frac{m}{n}$ is small. For simple graphs, some authors say it is sparse if m is $O(n)$ or less, rather than $O(n^2)$

An example of a sparse graph would be the graph of Facebook users with edges representing friendships. Facebook has hundreds of millions of users but each user has only a few hundreds of friends. In other words, every vertex of the graph has only a few hundreds of neighbours.

From the table we see that checking whether an edge exists is much faster for adjacency matrices than for adjacency lists. On the other hand, if our graph is sparse, then the allocated space of adjacency lists is much smaller than adjacency matrices and also traversing neighbours is faster for adjacency lists than adjacency matrices.