1 Definition of a graph

A graph is a set of nodes, or vertices, connected by edges

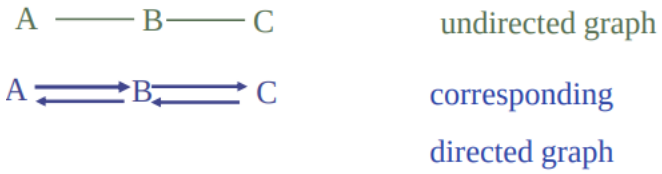
2 Applications

- Networks (e.g., of computers or roads)
- Flow charts
- Tasks in some project (some of which should be completed before others), so edges correspond to prerequisites
- States of an automaton / program

3 Directed and Undirected

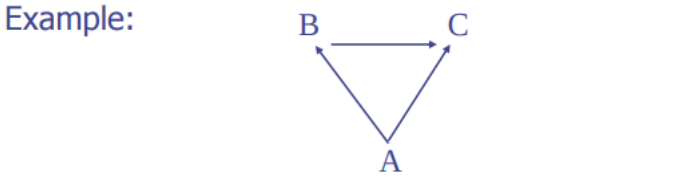
- undirected edges dont have direction (relationship goes both ways)
- directed (digraph) edges have direction (relationship flow has to be defined)

Undirected graphs can be represented as directed graphs where for each edge (X,Y) there is a corresponding edge (Y,X).



4 Notation

- Set V of vertices (nodes)
- Set E of edges $E \subseteq V \times V$



- Node B is *adjacent* to A if there is an edge from A to B. ($A \rightarrow B$)
- A *path* from A to B is a sequence of vertices where each vertex points to the second one
- A vertex B is *reachable* from A if there is a path from A to B
- A *cycle* is a path from a vertex to itself
- Graph is *acyclic* if it does not have cycles
- An **undirected** graph is *connected* if there is a path between every pair of vertices
- For a directed graph:
  - It is *weakly connected* if the corresponding *undirected graph* is **connected** (i.e. if we permit traversal of edges in any direction)
  - It is *strongly connected* if for every ordered pair (v1,v2) of vertices, there is a path that respects the directions of the edges, and that goes from v1 to v2. Note need paths from v1 to v2, and also v2 to v1.

5 Common graph problems

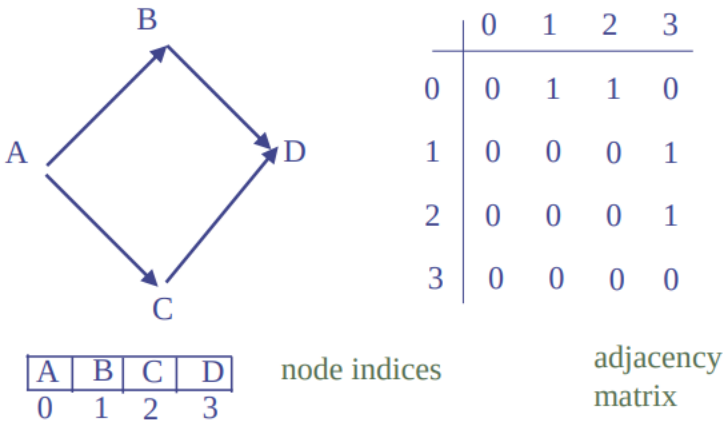
- Searching a graph for a vertex
- Searching a graph for an edge
- Finding a path in the graph (from one vertex to another)
- Finding the shortest path between two vertices
- Cycle detection

Implementation

- **static** indexed data structure: *Adjacency Matrix*
- **dynamic** data structure *Adjacency Lists*

6 Static Implementation: Adjacency Matrix

- Store node in an array: each node is associated with an integer (array index inside 2D grid)
- Represent information about the edges using a two dimensional array, where `array[i][j] == 1` iff there is an edge from node with index `i` to the node with index `j`.



For weighted graphs, place weights in the matrix if there is no edge we use a value which cant be confused with a weight, e.g., `-1` or `Integer.MAX_VALUE`

6.1 Disadvantages

- Sparse graphs with few edges for number that are possible result in many zero entries in adjacency matrix
- This **wastes space** and makes many algorithms **less efficient** e.g., to find nodes adjacent to a given node, we have to iterate through the whole row even if there are few 1s there.
- Also, if the number of nodes in the graph may change, matrix representation is too b especially if we **dont know** the maximal size of the graph.

7 Adjacency List

- For every vertex, keep a **list** of adjacent vertices.
- Keep a list of vertices, or keep vertices in a Map (e.g. HashMap) as keys and lists of adjacent vertices as values. (The best choice depends on what the graph algorithm needs to do.)
- Example: A - key, "B, C" - value(list of nodes that A points to) $A \rightarrow B, C$

Reference section

placeholder