A **Boolean Matrix** is a real matrix whose entries are either 0 or 1.

Note that the boolean entries 0 and 1 can be defined in several ways. In electrical switch to describe "on and off", in graph theory, the "adjacency matrix" etc , the boolean entries 0 and 1 are used. We consider the same type of Boolean matrices in our discussion.

The following two kinds of operations on the collection of all boolean matrices are defined.

Let $A = [a_{ij}]$ and $B = [b_{ij}]$ be any two boolean matrices of the **same type**. Then their **join** $\vee$ and **meet** $\wedge$ are defined as follows:

Definition : Join of A and B

$A \vee B = [a_{ij}] \vee [b_{ij}] = [a_{ij} \vee b_{ij}] = [c_{ij}]$

$$\text{where } c_{ij} = \begin{cases} 1, \text{ if either } a_{ij} = 1 \text{ or } b_{ij} = 1 \\ 0, \text{ if both } a_{ij} = 0 \text{ and } b_{ij} = 0 \end{cases}$$

Definition : Meet of A and B

$A \wedge B = [a_{ij}] \wedge [b_{ij}] = [a_{ij} \wedge b_{ij}] = [c_{ij}]$

$$\text{where } c_{ij} = \begin{cases} 1, \text{ if both } a_{ij} = 1 \text{ and } b_{ij} = 1 \\ 0, \text{ if either } a_{ij} = 0 \text{ or } b_{ij} = 0. \end{cases}$$

It is clear that $(a \vee b) = \max\{a, b\}$; $(a \wedge b) = \min\{a, b\}$, $a, b \in \{0, 1\}$.

Example 1

Let $A = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}, \ B = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ be any two boolean matrices of the same type. Find $A \vee B$ and $A \wedge B$.

Solution

$$\text{Then } A \vee B = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \vee \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0\vee 1 & 1\vee 1 \\ 1\vee 0 & 1\vee 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$A \wedge B = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \wedge \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0\wedge 1 & 1\wedge 1 \\ 1\wedge 0 & 1\wedge 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$$

Properties satisfied by join and meet

Let **B** be the set of all boolean matrices of the same type. We only state the properties of meet and join.

Closure property

$A, B \in$ **B**, $A \vee B = [\, a_{ij} \,] \vee [b_{ij} \,] = [\, a_{ij} \vee b_{ij} \,] \in$ **B**. (Because, $(a_{ij} \vee b_{ij})$ is either 0 or 1 $\forall i \,, j$. $\vee$ is a binary operation on **B**.

Associative property

A∨(B∨C) = (A ∨ B) ∨ C, ∀A,B,C ∈ **B**. ∨ is associative.

Existence of identity property

$\forall A \in$ **B**, $\exists$ the null matrix $0 \in$ **B** $\ni A \vee 0 = 0 \vee A = A$. The identity element for ∨ is the null matrix.

Existence of inverse property

For any matrix $A \in$ **B**, it is impossible to find a matrix

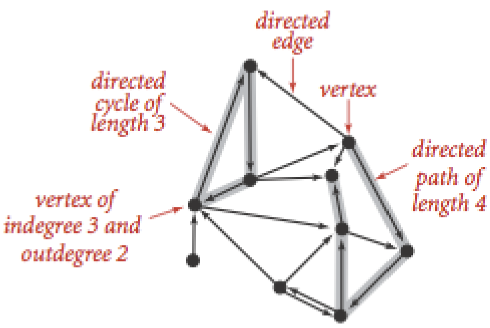
$B \in$ **B** $\ni A \vee B = B \vee A = 0$. So the inverse does not exist.

Similarly, it can be verified that the operation meet ∧ satisfies (i) closure property (ii) commutative property (iii) associative property (iv) the matrix $U = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ exists as the identity in B and (v) the existence of inverse is not assured.

Digraphs.

A *directed graph* (or *digraph*) is a set of *vertices* and a collection of *directed edges* that each connects an ordered pair of vertices. We say that a directed edge *points from* the first vertex in the pair and *points to* the second vertex in the pair. We use the names 0 through V-1 for the vertices in a V-vertex graph.

- A *self-loop* is an edge that connects a vertex to itself.
- Two edges are *parallel* if they connect the same ordered pair of vertices.
- The *outdegree* of a vertex is the number of edges pointing from it.
- The *indegree* of a vertex is the number of edges pointing to it.
- A *subgraph* is a subset of a digraph's edges (and associated vertices) that constitutes a digraph.
- A *directed path* in a digraph is a sequence of vertices in which there is a (directed) edge pointing from each vertex in the sequence to its successor in the sequence, with no repeated edges.
- A directed path is *simple* if it has no repeated vertices.
- A *directed cycle* is a directed path (with at least one edge) whose first and last vertices are the same.
- A directed cycle is *simple* if it has no repeated vertices (other than the requisite repetition of the first and last vertices).
- The *length* of a path or a cycle is its number of edges.
- We say that a vertex `w` is *reachable from* a vertex `v` if there exists a directed path from `v` to `w`.
- We say that two vertices `v` and `w` are *strongly connected* if they are mutually reachable: there is a directed path from `v` to `w` and a directed path from `w` to `v`.
- A digraph is *strongly connected* if there is a directed path from every vertex to every other vertex.
- A digraph that is not strongly connected consists of a set of *strongly connected components*, which are maximal strongly connected subgraphs.
- A *directed acyclic graph* (or DAG) is a digraph with no directed cycles.



Anatomy of a digraph

Digraph graph data type.

We implement the following digraph API.

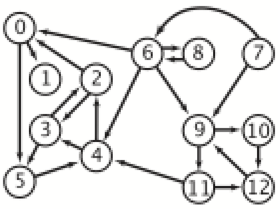
public class	Digraph	
	Digraph(int V)	*create a V-vertex digraph with no edges*
	Digraph(In in)	*read a digraph from input stream in*
int	V()	*number of vertices*
int	E()	*number of edges*
void	addEdge(int v, int w)	*add edge v->w to this digraph*
Iterable<Integer>	adj(int v)	*vertices connected to v by edges pointing from v*
Digraph	reverse()	*reverse of this digraph*
String	toString()	*string representation*

The key method `adj()` allows client code to iterate through the vertices adjacent from a given vertex.

We prepare the test data using the following input file format.

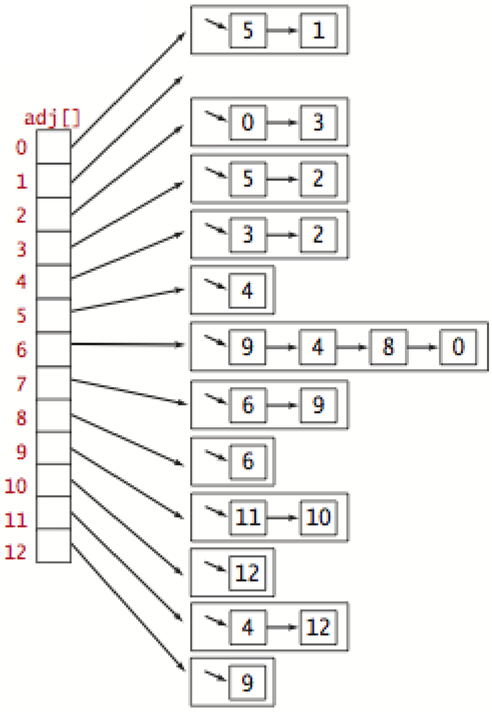
tinyDG.txt

```
V -> 13
     22 <- E
 4  2
 2  3
 3  2
 6  0
 0  1
 2  0
11 12
12  9
 9 10
 9 11
 7  9
10 12
11  4
 4  3
 3  5
 6  8
 8  6
 5  4
 0  5
 6  4
 6  9
 7  6
```



Graph representation.

We use the *adjacency-lists representation*, where we maintain a vertex-indexed array of lists of the vertices connected by an edge to each vertex.



Reachability in digraphs.

Depth-first search and breadth-first search are fundamentally digraph-processing algorithms.

Single-source reachability: Given a digraph and source `s`, is there a directed path from s to v? If so, find such a path.

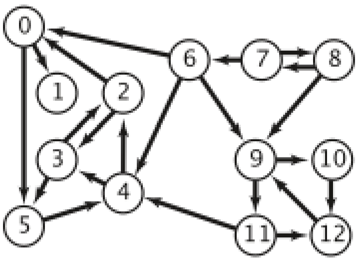
Multiple-source reachability: Given a digraph and a *set* of source vertices, is there a directed path from *any* vertex in the set to v?

Single-source directed paths: given a digraph and source `s`, is there a directed path from s to v? If so, find such a path.

Single-source shortest directed paths: given a digraph and source `s`, is there a directed path from s to v? If so, find a shortest such path.

Transitive closure.

The *transitive closure* of a digraph G is another digraph with the same set of vertices, but with an edge from v to w if and only if w is reachable from v in G.



	0	1	2	3	4	5	6	7	8	9	10	11	12
0	T	T	T	T	T	T							
1		T											
2	T	T	T	T	T	T							
3	T	T	T	T	T	T							
4	T	T	T	T	T	T							
5	T	T	T	T	T	T							
6	T	T	T	T	T	T	T			T	T	T	T
7	T	T	T	T	T	T	T	T	T	T	T	T	T
8	T	T	T	T	T	T	T	T	T	T	T	T	T
9	T	T	T	T	T	T				T	T	T	T
10	T	T	T	T	T	T				T	T	T	T
11	T	T	T	T	T	T				T	T	T	T
12	T	T	T	T	T	T				T	T	T	T

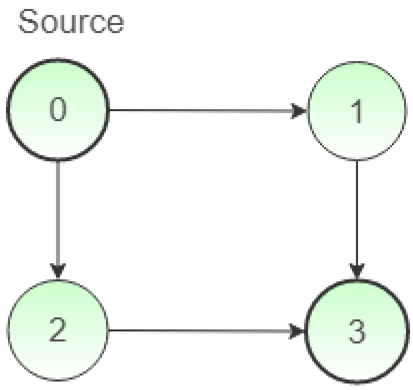
original edge (red)

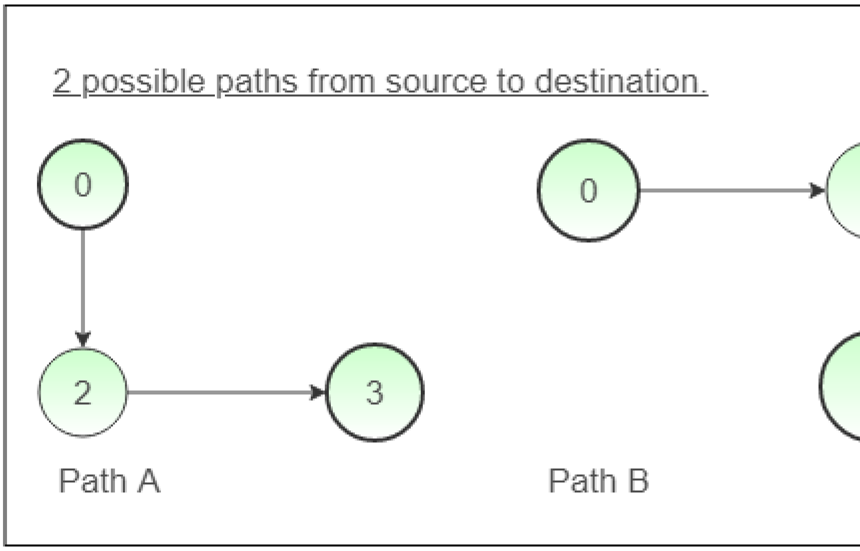
self-loop (gray)

12 is reachable from 6

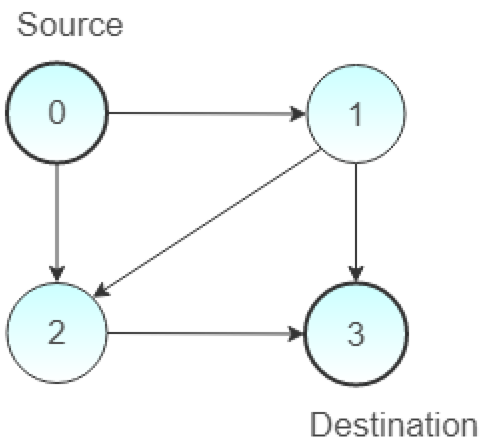
Paths in directed graph: All paths in a directed acyclic graph from a given source node to a given destination node can be found using **Depth-First-Search** traversal.

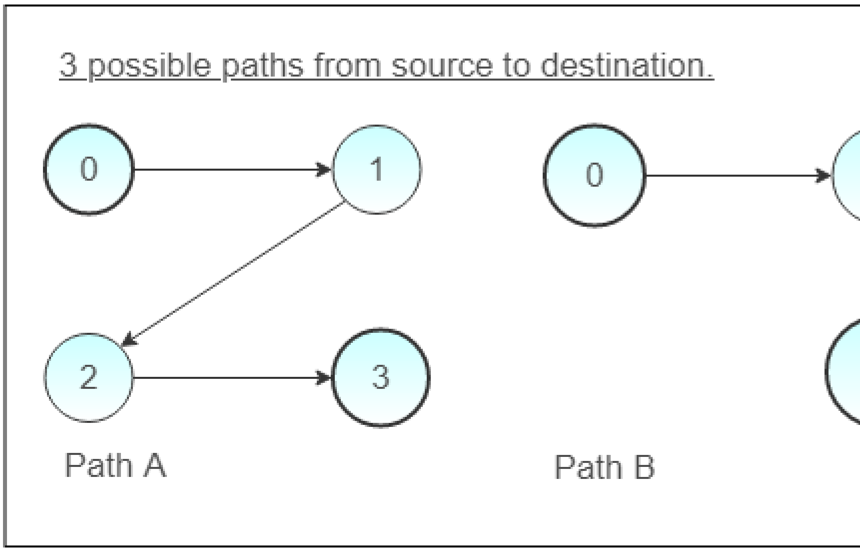
Dircted Acyclic Graph G1





Dircted Acyclic Graph G2





Dircted Acyclic Graph G3