Graph Representations

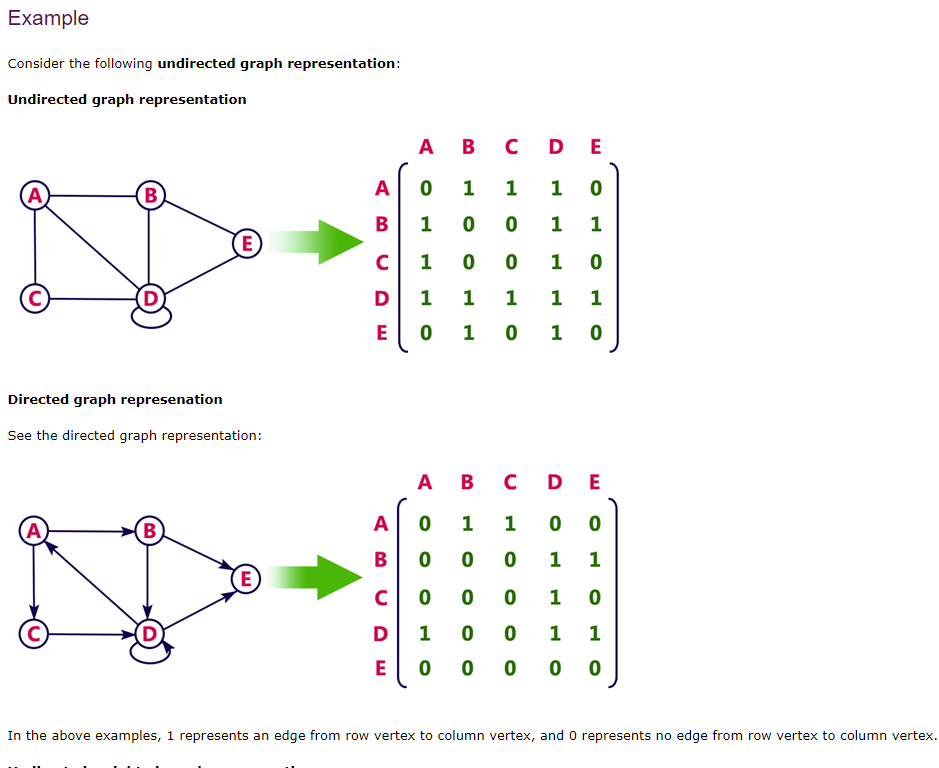
In graph theory, a graph representation is a technique to store graph into the memory of computer.

There are different ways to optimally represent a graph, depending on the density of its edges, type of operations to be performed and ease of use:-

* Adjacency Matrix
* Incidence Matrix
* Adjacency List

**Adjacency Matrix:**

* Adjacency matrix is a sequential representation.
* It is used to represent which nodes are adjacent to each other. i.e. is there any edge connecting nodes to a graph.
* In this representation, we have to construct a nXn matrix A. If there is any edge from a vertex i to vertex j, then the corresponding element of A, ai,j = 1, otherwise ai,j= 0.
* If there is any weighted graph then instead of 1s and 0s, we can store the weight of the edge.



## Incidence Matrix

In **Incidence matrix representation**, graph can be represented using a matrix of size:

Total number of vertices by total number of edges.

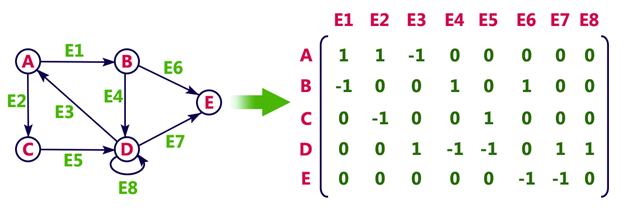
It means if a graph has 4 vertices and 6 edges, then it can be represented using a matrix of 4X6 class. In this matrix, columns represent edges and rows represent vertices.

This matrix is filled with either **0 or 1** or -1. Where,

* 0 is used to represent row edge which is not connected to column vertex.
* 1 is used to represent row edge which is connected as outgoing edge to column vertex.
* -1 is used to represent row edge which is connected as incoming edge to column vertex.

### **Example**

Consider the following directed graph representation.

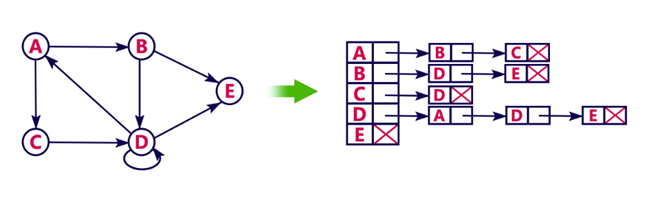


## Adjacency List

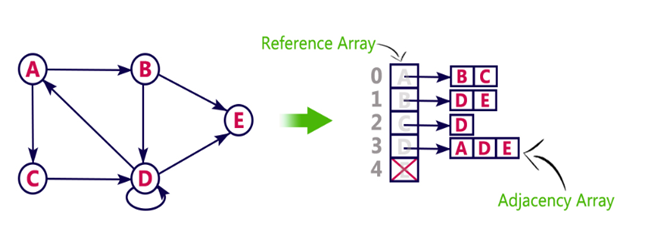
* Adjacency list is a linked representation.
* In this representation, for each vertex in the graph, we maintain the list of its neighbors. It means, every vertex of the graph contains list of its adjacent vertices.
* We have an array of vertices which is indexed by the vertex number and for each vertex v, the corresponding array element points to a **singly linked list** of neighbors of v.

### **Example**

Let's see the following directed graph representation implemented using linked list:



We can also implement this representation using array as follows:



**Pros:**

* Adjacency list saves lot of space.
* We can easily insert or delete as we use linked list.
* Such kind of representation is easy to follow and clearly shows the adjacent nodes of node.

**Cons:**

* The adjacency list allows testing whether two vertices are adjacent to each other but it is slower to support this operation.

# [**When are adjacency lists or matrices the better choice?**](https://cs.stackexchange.com/questions/79322/when-are-adjacency-lists-or-matrices-the-better-choice)

First of all note that [*sparse*](https://en.wikipedia.org/wiki/Dense_graph#Sparse_and_tight_graphs) means that you have very few edges, and [*dense*](https://en.wikipedia.org/wiki/Dense_graph) means many edges, or almost complete graph. In a complete graph you have **n(n−1)/2n(n−1)/2 edges**, where **n** is the number of nodes.

Now, when we use matrix representation we allocate **n×n matrix** to store node-connectivity information, e.g., **M[i][j]= 1** if there is edge between nodes i and j, otherwise **M[i][j]=0.**

But if we use adjacency list then we have an array of nodes and each node points to its adjacency list **containing ONLY its neighboring nodes**.

Now if a graph is sparse and we use matrix representation then most of the matrix cells **remain unused which leads to the waste of memory**. **Thus, we usually don't use matrix representation for sparse graphs. We prefer adjacency list.**

But if the graph is dense then the number of edges is close to (the complete) **n(n−1)/2,** or to **n^2** if the graph is directed with self-loops. Then there is no advantage of using adjacency list over matrix.

**In terms of space complexity**

Adjacency matrix: O(n^2)  
Adjacency list: O(n+m)

where **n** is the number nodes,  **m** is the number of edges.

When the graph is undirected tree then  
Adjacency matrix: O(n^2)  
Adjacency list: O(n+n) is O(n) (better than n^2)

When the graph is directed, complete, with self-loops then  
Adjacency matrix: O(n^2)  
Adjacency list: O(n+n^2) is O(n^2) (no difference)

And finally, when you implement using matrix, **checking if there is an edge between two nodes takes O(1) times, while with an adjacency list, it may take linear time in n.**