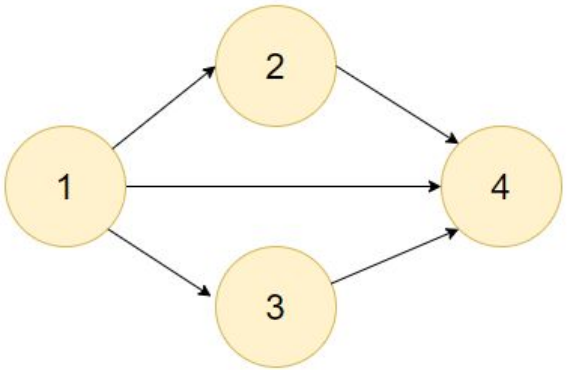
Algorithms: Graph Representation (Directed Graphs, Weighted Graphs)

Adjacency Matrix (Directed)

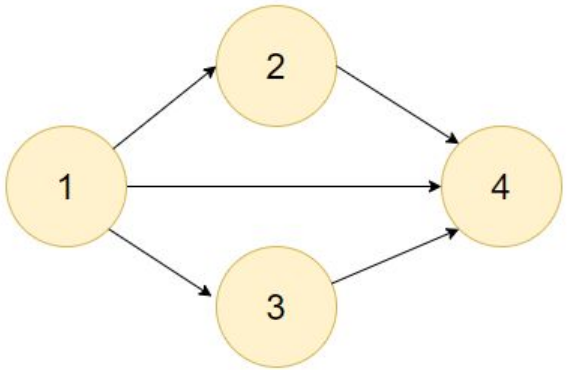


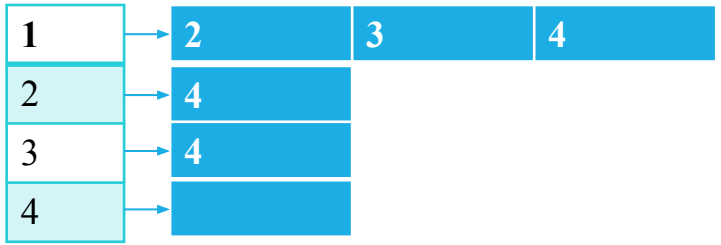
	1	2	3	4
1	0	**1**	**1**	**1**
2	0	0	0	**1**
3	0	0	0	**1**
4	0	0	0	0

Data Structure to Use: (n * n) Matrix.

Example: int adj [n][n];

Adjacency List (Directed)





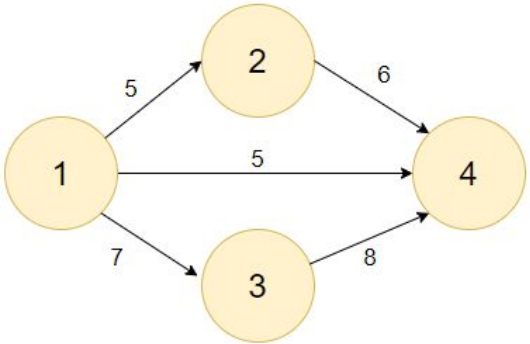
Data Structure to Use: 2D vector or array of linked list.

Example:

vector<int> adj [n];

vector< vector<int> > adj;

Adjacency Matrix (Directed and Weighted)

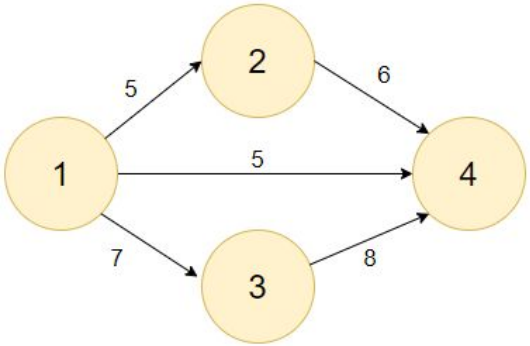


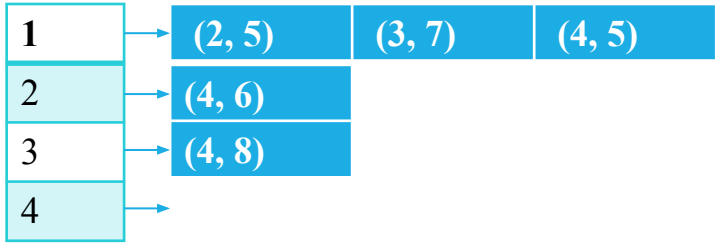
	1	2	3	4
1	0	**5**	**7**	**5**
2	0	0	0	**6**
3	0	0	0	**8**
4	0	0	0	0

Data Structure to Use: (n * n) Matrix.

Example: int adj [n][n];

Adjacency List (Directed and Weighted)





Data Structure to Use:
2D vector or array of linked list.

Example:

vector< pair<int, int> > adj [n];

Adjacency Matrix vs Adjacency List

- Adjacency Matrix:
  - Space Complexity: $O(n^2)$
  - Time complexity of Checking if an edge (u,v) exists: $O(1)$
- Adjacency List:
  - Space Complexity: $O(n+m)$
  - Time complexity of Checking if an edge (u, v) exists: $O(d)$

The benefit of using adjacency lists is that we can efficiently find the nodes to which we can move from a given node through an edge. For example, the following loop goes through all nodes to which we can move from node s:

```
for (auto u : adj[s]) {
    // process node u
}
```

- n : total number of nodes
- m: total number of edges
- d: degree (number edges connected to that node)