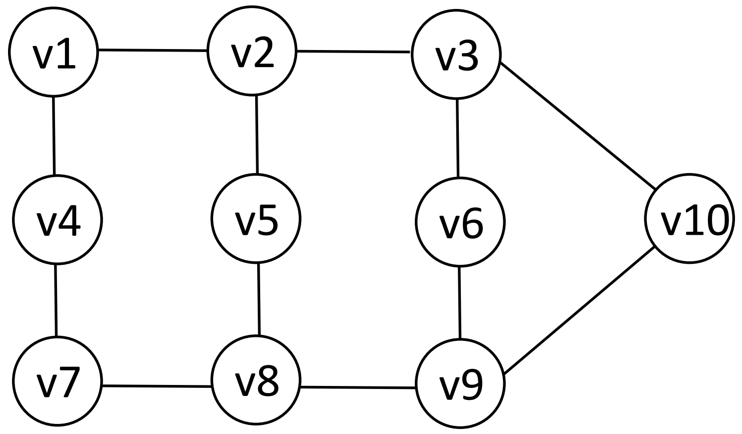
DS: **Graph**

Liwei

What is Graph?

- Graph is a pair of sets (V, E), where V is the set of vertices and E is the set of edges, connecting the pairs of vertices.
- How does a typical graph look like?

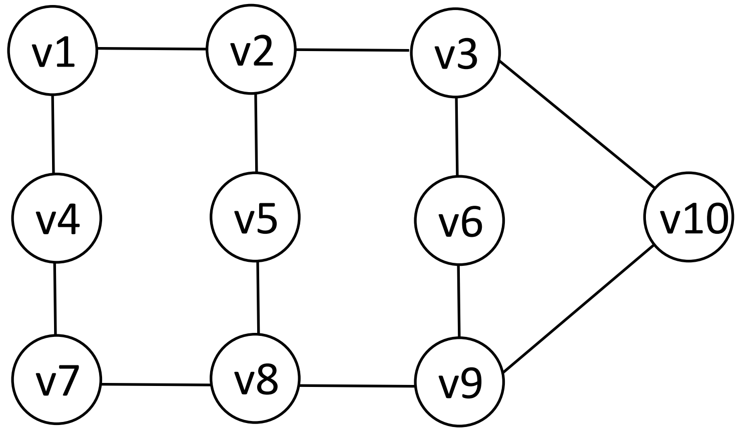


- V = {v1, v2, v3, v4, v5, v6, v7, v8, v9, v10}
- E = {v1v2, v2v3, v1v4, v4v7, v7v8, v2v5, v5v8, v3v6, v6v9, v8v9, v3v10, v9v10}

Why should we learn graph?

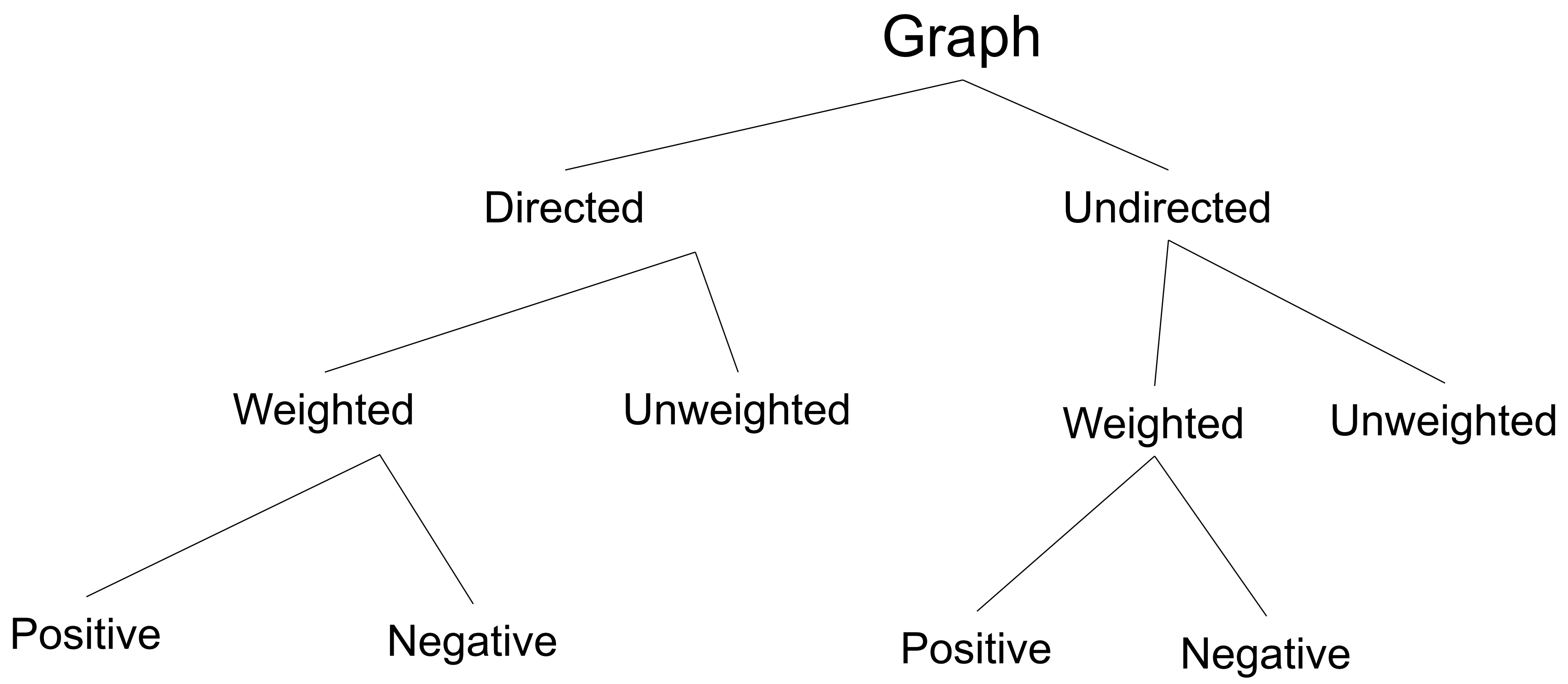
- Shorted path between cities / stations

Terminologies



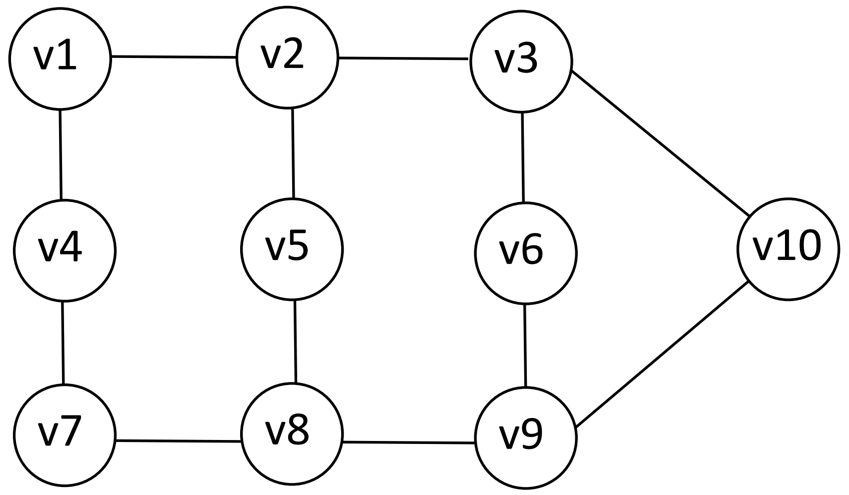
- **Vertices**: vertices are the nodes of the graph
- **Edges**: edges are the arcs that connect paris of vertices
- **Unweighted Graph**: a graph not having a weight associated with any edge
- **Weighted Graph**: a graph having a weight associated with each edge
- **Undirected Graph**: it is a graph that is a set of vertices connected by edges, where the edges don't have a direction associated with them
- **Directed Graph**: it is a graph that is a set of vertices connected by edges, where the edges have a direction associated with them
- **Cylick Graph**: a cyclic graph is a graph having at least one loop
- **Acyclic Graph**: an Acyclic graph is a graph having no loop.
- **Tree**: tree is a Special case of Directed Acyclic Graph (DAG)

Types of Graph



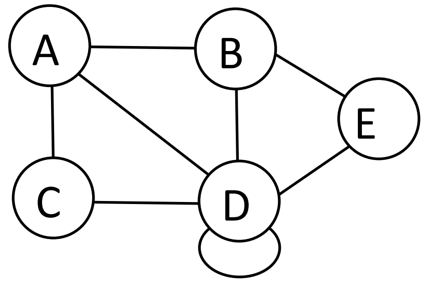
Types of Graph

- Un-Weighted-Undirected
- Un-Weighted-Directed
- Postive-Weighted-Undirected
- Positive-Weighted-Directed
- Negative-Weighted-Undirected
- Negative-Weighted-Directed

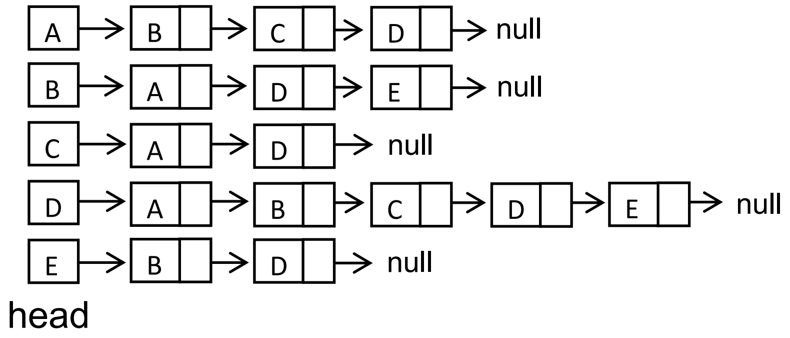


How is Graph represented

- Adjacency Matrix: in graph theory, an adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether paris of vertices are adjacent or not in the graph
- Adjacency List: in graph theory, an adjacency list is a collection of unordered lists used to represent a finite graph. Each list describes the set of neighbors of a vertex in the graph.



$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 1 & 1 & 1 & 0 \\ B & 1 & 0 & 0 & 1 & 1 \\ C & 1 & 0 & 0 & 1 & 0 \\ D & 1 & 1 & 1 & 1 & 1 \\ E & 0 & 1 & 0 & 1 & 0 \end{array}$$



When to use which representation?

- If the graph is a complete or near to complete graph, we should use adjacency matrix
- If the number of edges are few, we should use adjacency list

What is graph traversal

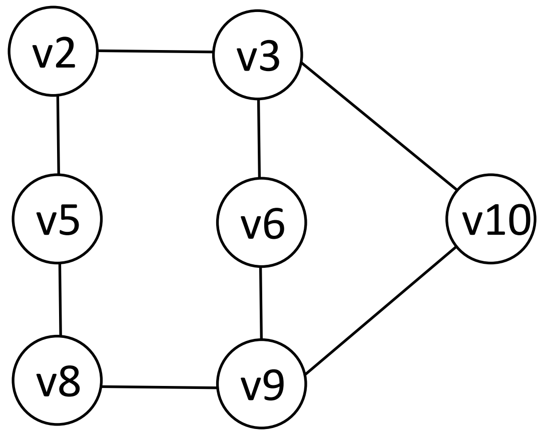
- Graph traversal refers to the process of visiting each vertex in a graph

Types of graph traversal

- Breadth First Search (BFS)
- Depth First Search (DFS)

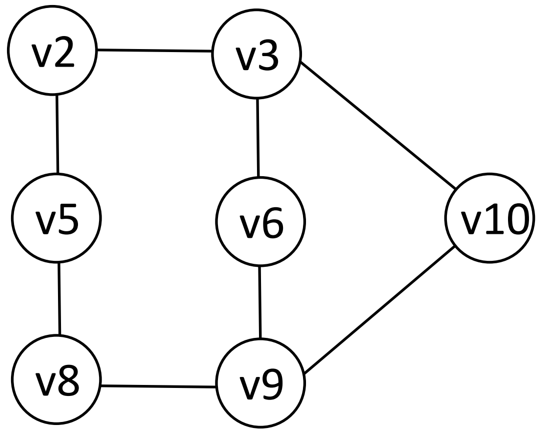
Breadth First Search

- BFS is an algorithm for traversing Graph data structures. It starts at some arbitrary node of a graph and explores the neightboring nodes (which are at current level) first, before moving to the next level neighbors.

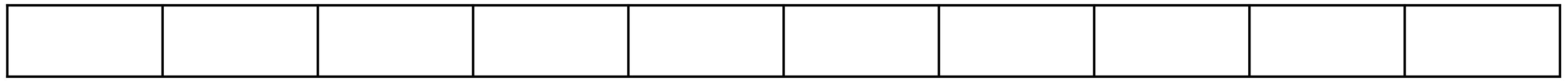


Algorithm: Breadth First Search

```
BFS(G)
    while all the vertices are not explored, do:
    enqueue (any vertex)
    while Q is not empty
        p = Dequeue()
        if p is unvisited
            print p and mark p as visited
            enqueue (all adjacent unvisited vertices of p)
```



QUEUE



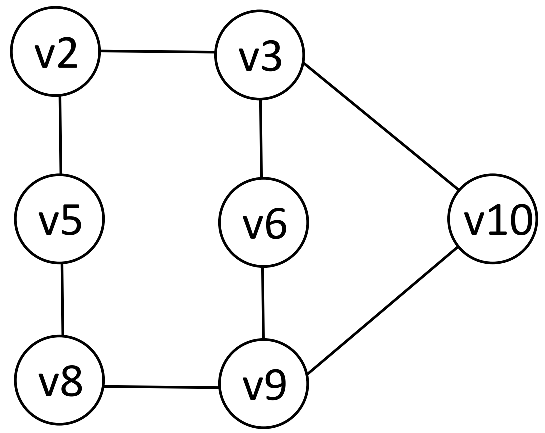
Algorithm: Breadth First Search

```
BFS(G)
    while all the vertices are not explored, do:              O(V)
    enqueue (any vertex)                                      O(1)
    while Q is not empty                                      O(V)         ┐
        p = Dequeue()                                         O(1)         │
        if p is unvisited                                     O(1)         │  O(E)
            print p and mark p as visited                     O(1)         │
            enqueue (all adjacent unvisited vertices of p)         O(Adj Vs) ┘
```

Time complexity: O(V+E)

Depth First Search

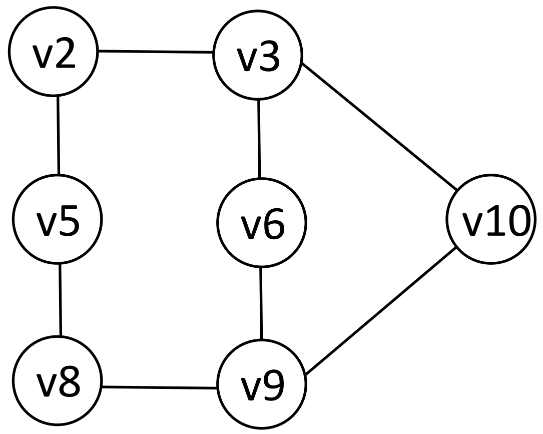
- DFS is an algorithm for traversing Graph data structures. It starts selecting some arbitrary node and explores as far as possible along each edge before backtracking.

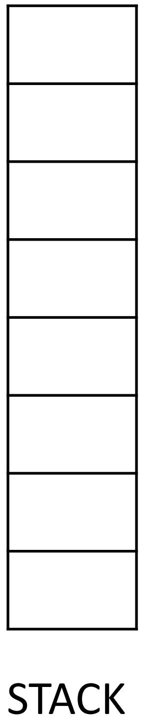


v2, v5, v8, v9, v6, v3, v10

Algorithm: Depth First Search

```
DFS(G)
        while all the vertices are not explored, do:
        push (any vertex)
        while Stack is not empty
                p = pop ()
                if p is unvisited
                        print p and mark p as visited
                        push (all unvisited adjacent vertices of p)
```



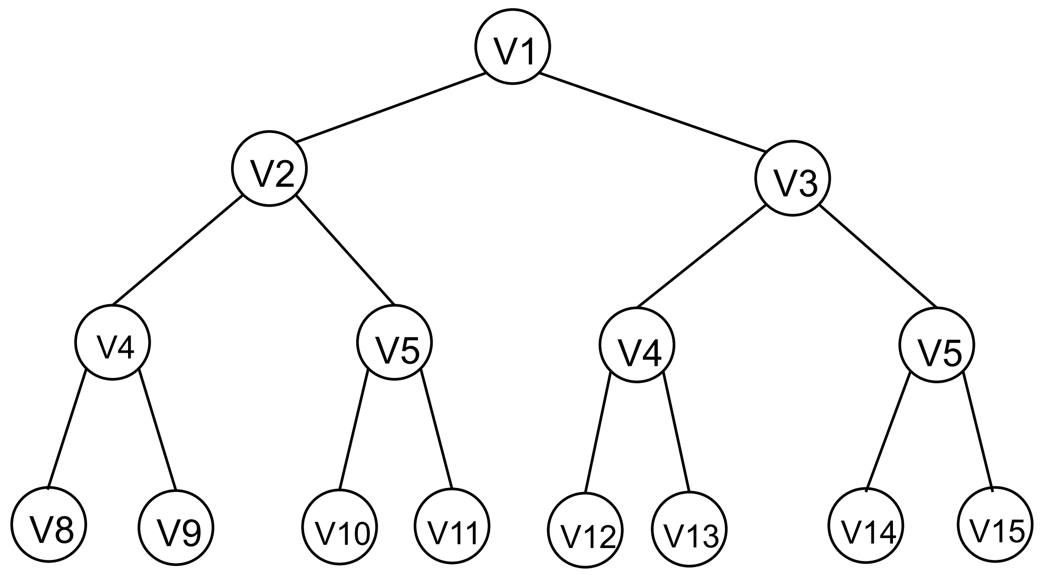


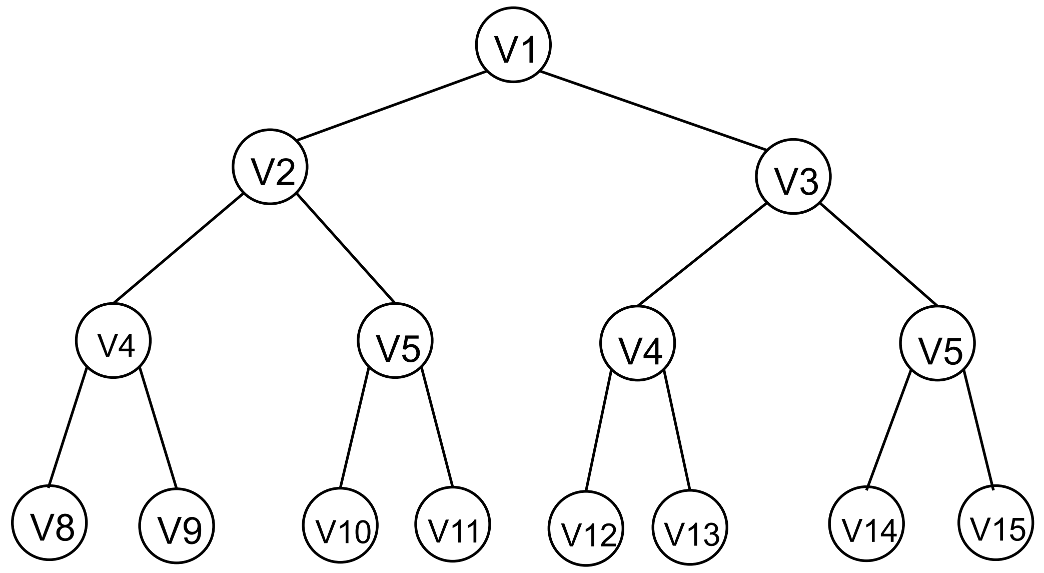
Algorithm: Depth First Search

```
DFS(G)
        while all the vertices are not explored, do:      O(V)
        push (any vertex)                                 O(1)
        while Stack is not empty                          O(V)   ⎤
                p = pop ()                                O(1)   ⎥
                if p is unvisited                         O(1)   ⎥  O(E)
                        print p and mark p as visited     O(1)   ⎥
                        push (all unvisited adjacent vertices of p)   O(Adj Vs)   ⎦
```

Time complexity: O(V+E)

BFS vs. DFS


V1
V2
V3
V4
V5
V4
V5
V8
V9
V10
V11
V12
V13
V14
V15


V1
V2
V3
V4
V5
V4
V5
V8
V9
V10
V11
V12
V13
V14
V15

BFS vs. DFS

	BFS	DFS
How it works internally	It goes in breadth first	It goes in depth first
Internally uses which DS	Queue	Stack
Time Complexity	O(V+E)	O(V+E)
When to use which	If we know that the target vertex is close to starting point	If we already know that the target vertex is burided very deep