Representing Graphs

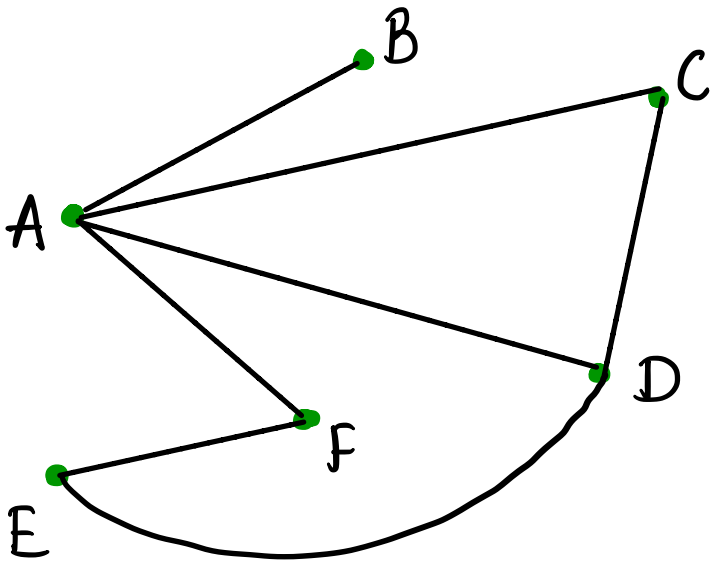
* Adjacency Matrix Representation.

A graph $G = \{V, E\}$ such that $|V| = n$ can be represented by a matrix A.

$$A[i, j] = \begin{cases} 1, & \text{if } (i, j) \in E \\ 0, & \text{otherwise.} \end{cases}$$

for undirected Graph : Adjacency Matrix is a symmetric matrix.

Example :- Undirected Graph.



Adjacency Matrix.

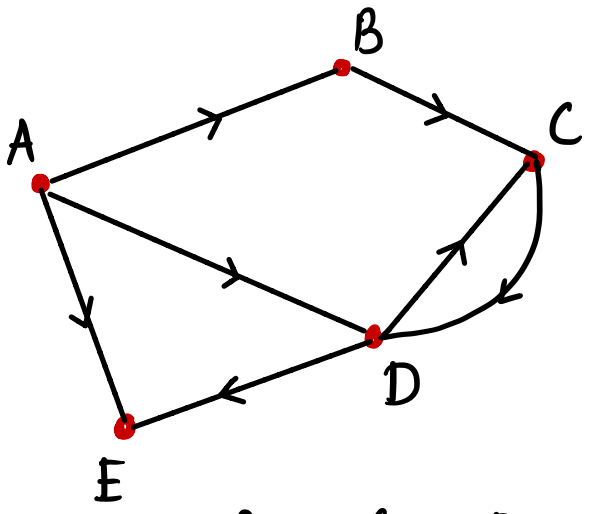
A :-

	A	B	C	D	E	F
A	0	1	1	1	0	1
B	1	0	0	0	0	0
C	1	0	0	1	0	0
D	1	0	1	0	1	0
E	0	0	0	1	0	1
F	1	0	0	0	1	0

Space complexity = $\Theta(n^2)$

If you store half matrix still $\Theta(n^2)$

Directed Graph.



(v_i, v_j)
Edge from v_i to v_j

(Outdegrees)

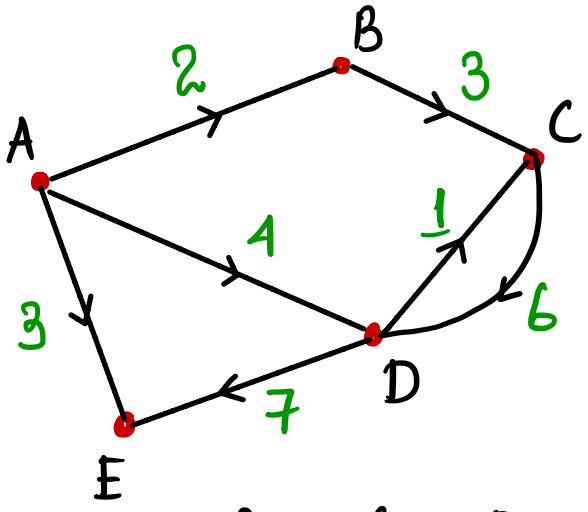
$A =$

	A	B	C	D	E	
A	0	1	0	1	1	3
B	0	0	1	0	0	1
C	0	0	0	1	0	1
D	0	0	1	0	1	2
E	0	0	0	0	0	0

(Indegrees) 0 1 2 2 2

Space Complexity = $\Theta(|V|^2)$

Weighted Undirected Graph.

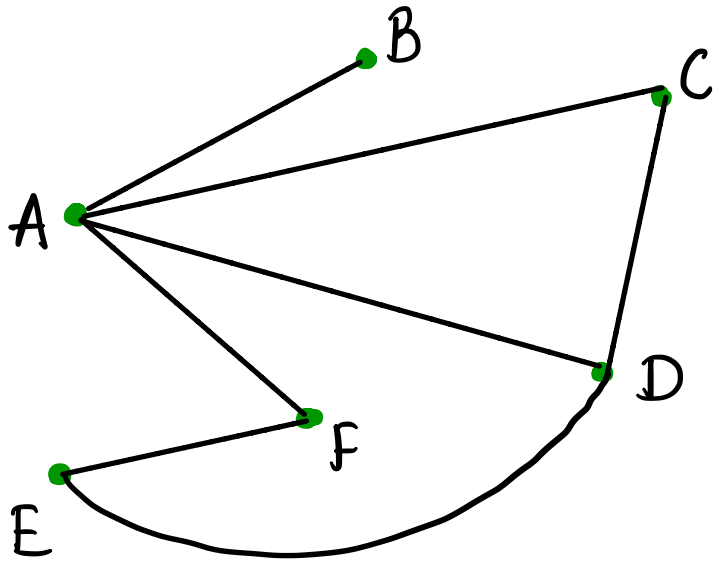


$A =$

	A	B	C	D	E
A	0	2	0	4	1
B	0	0	3	0	0
C	0	0	0	6	0
D	0	0	1	0	7
E	0	0	0	0	0

Most of the Cases Adjacency Matrix is Sparse (More No. of Zeros).

We can avoid storing 0s



Neighbors.

(A) → {B, C, D, E, F}

(B) → {A}

(C) → {A, D}

(D) → {A, C, E}

(E) → {D, F}

(F) → {A, E}

∴ Adjacency List Representation"

Space complexity :-

$\max(\Theta(|V|), \Theta(|E|))$

Adjacency List

Comparison Adjacency Matrix and Adjacency List Representation of Graph.

Adjacency List (AL) representation takes comparatively less space than Adjacency Matrix (AM) representation.

To Check if j is a neighbour of i

AM Representation : Check if A[i][j] is 1 or not.

AL Representation : Scan the complete adjacency list. (costly)

To find all neighbours of i^{th} vertex

AM : Scan all the columns of i^{th} row.

AL : Already Available.

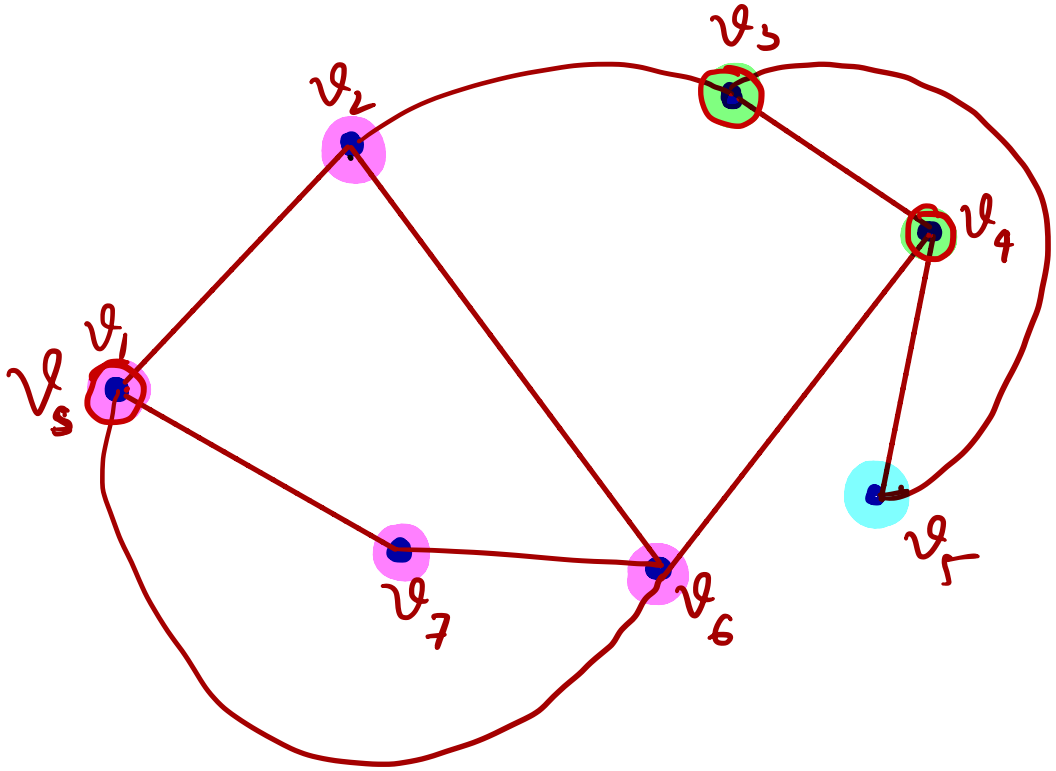
Breadth First Search. (BFS)

An algorithm for traversing a graph.

→ Visiting each vertex of graph in a systematic way.

Logic :-

* Select a starting vertex v_s
* Visit Vertices which are just one step away from v_s
* Visit all non-visited vertices which are two step away v_s
* Visit all non-visted vertices which are three step away from v_s.
* and Go On till you have no vertices left to visit.



$\{v_1, v_2, v_7, v_6\}$ (One Step Away)

$\{v_3, v_4\}$ (Two Step Away)

$\{v_5\}$ (Three Step Away)

All nodes (vertices) visited.

Input to the Algorithm : Starting Vertex.

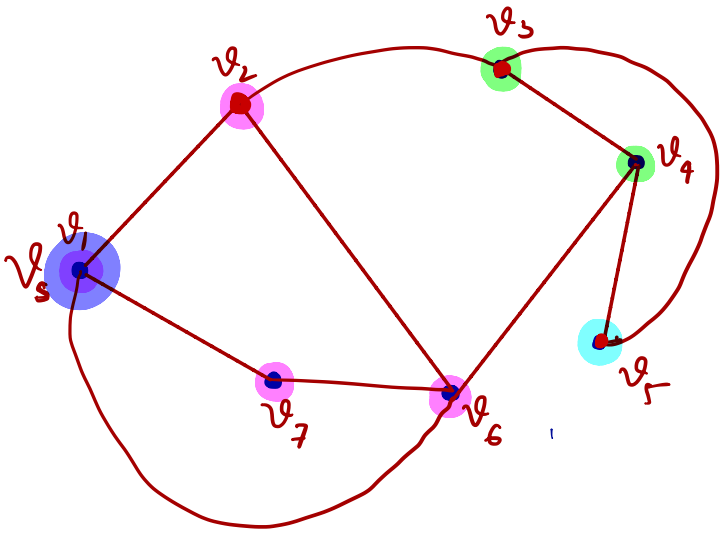
Output of the Algorithm : Visiting every node in a connected component of the graph.

```
BreadthFirstSearch ( v_s )
{
    for ( i = 1 to n)
        visited[i] ← 0 ;
        parent[i] ← -1 ;
        level[i] ← -1 ;
    endfor.
    Queue Q ; // Empty Queue.
    visited[s] = 1 ;  // Starting Vertex is visited
    Q.enqueue(s) ;
    level[s] = 0 ;
```

```
while(!Q.IsEmpty())
    j ← Q.dequeue();
    for each (j, k) ∈ E,
        if (visited[k] == 0)
            visited[k] = 1;
            Q.enqueue(k);
            parent[k] = j;
            level[k] = 1 + level[j];
        endif.
    end for each.
End while
```

Complexity :-

$O(m+n)$



Initialization

visited:	1	0	0	0	0	0	0
parent:	-1	-1	-1	-1	-1	-1	-1
level:	-1	-1	-1	-1	-1	-1	-1

Q: 1

Iteration 1 (j = 1)

Visited:	1	1	0	0	0	1	1
parent:	-1	1	-1	-1	-1	1	1
level:	0	1	-1	-1	-1	1	1

Q: 2 | 6 | 7

Iteration 2 (j = 2)

Visited:	1	1	1	0	0	1	1
Parent:	-1	1	2	-1	-1	1	1
level:	0	1	2	-1	-1	1	1

Q: 6 | 7 | 3

Iteration 3 (j = 6)

Visited:	1	1	1	1	0	1	1
Parent:	-1	1	2	6	-1	1	1
level:	0	1	2	2	-1	1	1

Q: 7 | 3 | 4

Iteration 4 (j = 7)

visited:	1	1	1	1	0	1	1
parent:	-1	1	2	6	-1	1	1
level:	0	1	2	2	-1	1	1

Q: 3 | 4

Iteration 5 (j = 3)

visited:	1	1	1	1	1	1	1
parent:	-1	1	2	6	3	1	1
level:	0	1	2	2	3	1	1

Q: 4 | 5

Iteration 6 (j = 4)

visited :

1	1	1	1	1	1	1

parent :

-1	1	2	6	3	1	1

level :

0	1	2	2	3	1	1

Q :

5						

Iteration 7 (j = 5)

visited :

1	1	1	1	1	1	1

parent :

-1	1	2	6	3	1	1

level :

0	1	2	2	3	1	1

Q :

Observations :-

1. Each vertex in a connected graph enters exactly once in Q.
2. Each vertex, except the source vertex v_s, has a parent.
3. The traversal visits all the vertices of G that are reachable from 's'.
4. BFS gives the shortest path from source to each nodes in terms of no. of edges or if the weights associated with each and every edge of graph is constant.