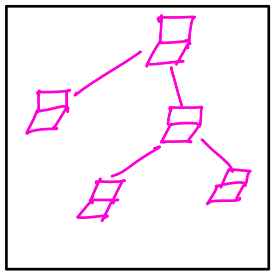
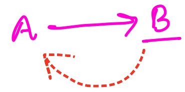
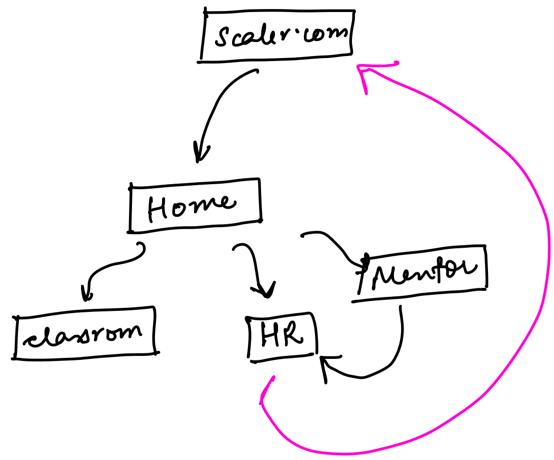
Graphs :- network

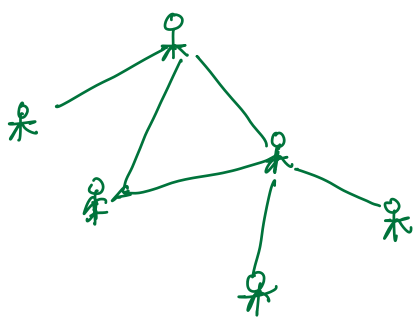
collection of nodes and edges



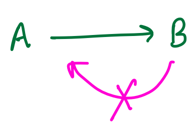




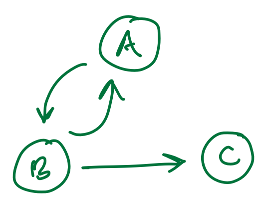
social media

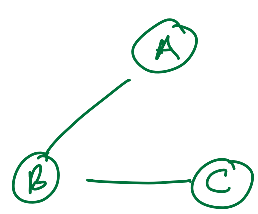


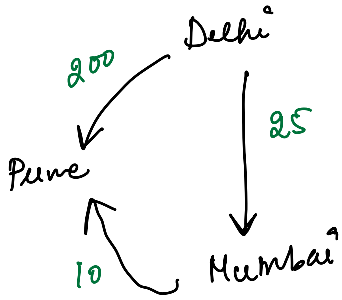
Tree – graph ?

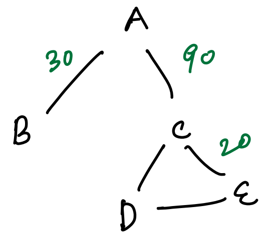


undirected / directed



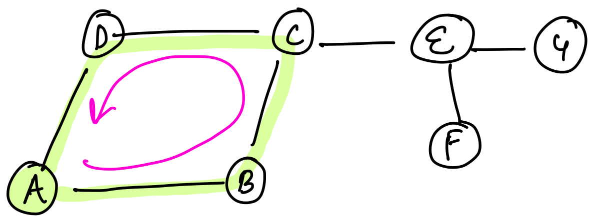






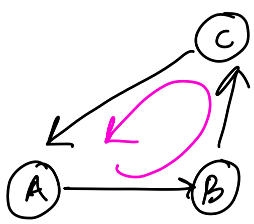
single graph
unweighted
weighted

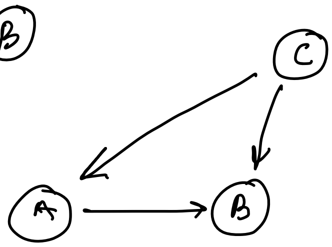
weighted/unweighted



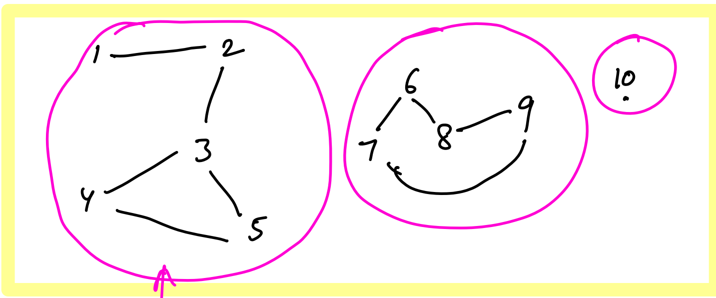
cyclic graph

If you can start from a node and come back to the same node without covering any edge twice ≡ cycle





acyclic graph

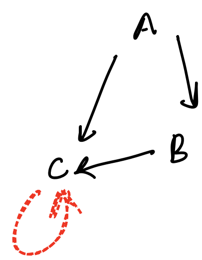


3!

connected components

undirected

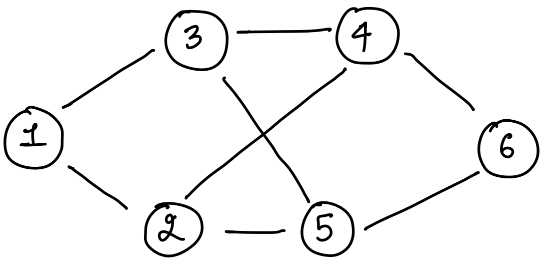
every node is reachable from other node



strongly connected components

Representation of graph

{ 1 → n
0 → n-1 }



$n = 6$
edges = 8

1–3	2–5
1–2	2–4
3–4	5–6
3–5	4–6

weighted ⇒ weight in replacement of 1

1) Adjacency Matrix

1 → n

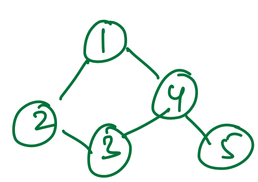
$(n+1) * (n+1)$

	0	1	2	3	4	5	6
0							
1	0	0	1	1		0	0
2		1				1	
3		1			1	1	
4				1			1
5			1	1			1
6					1	1	

mat[i][j] → 0 ~~i – j~~
mat[i][j] → 1 i – j

Advantage
- access – O(1)
- update – O(1) of edge
- wastage of space

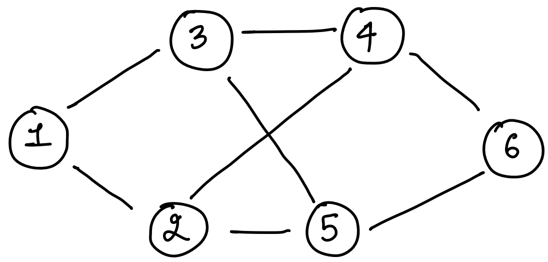
dense   sparse



5
5
1 2
2 3
3 4
4 5
1 4
} 6

```
int n; input(n)
int m; input(m)
for( int i=0; i<m; i++)
{
    int u, v; input(u,v);
    mat[u][v] = 1;
    mat[v][u] = 1;
}
```

Adjacency list



list

1	2, 3
2	1, 4, 5
3	1, 4, 5
4	3, 2, 6
5	2, 3, 6
6	4, 5

```
hashmap < int , list > graph;
list <int> graph[n+1];
```

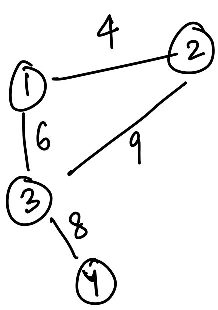
```
Hashset<int> graph[n+1];
```

when you want to have info about the edge.

```
int n;
int m;
list<int> graph[n+1];

for(i=0; i<m; i++)
{
        int u,v;
    graph[u].insert(v);
    graph[v].insert(u);
}
```

```
graph[u].insert(v);
```

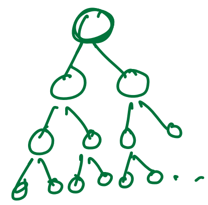


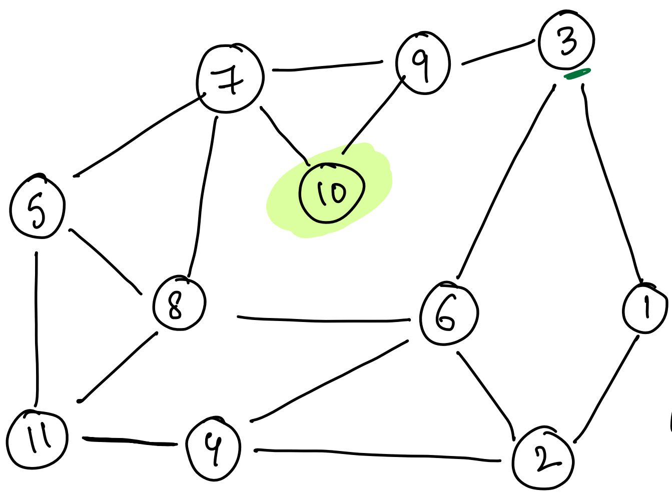
1	{2, 4}	{3, 6}
2	{1, 4}	{3, 9}
3	{1, 6}	{4, 8}
4	{3, 8}	

```
list<pair> graph[n+1];
```

Traversals → BFS
→ DFS

BFS: Breadth first search } level order





~~10~~ ~~7~~ ~~9~~ 5 8 10 9 7 10 3 6

bool visited[n+1];

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

10 9 7 3 5 8
6 1 11 4 2

queue

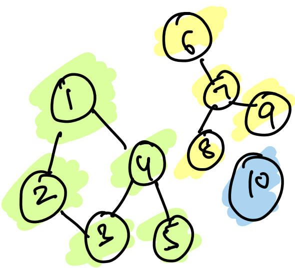
~~10~~ ~~9~~ ~~7~~ ~~3~~ ~~5~~ ~~8~~ ~~6~~ ~~1~~ ~~11~~ ~~4~~ ~~2~~

```
queue <int> q;
bool visited[n+1]; // initially false          1 -> n

for(int source=1; source<=n; source++)
{   if( visited[source] ) continue;
    q.push(source);
      visited[source] = true;

    while( ! q.empty())
    {
        int u = q.front();
             q.pop();
        print(u);

        for(int i=0; i< graph[u].size(); i++)
        {
            int v = graph[u][i];
            if( ! visited[v])
            {
                visited[v] = true;
                q.push(v);
            }
        }
    }
}
```



T.C: $O(n+m)$ — m: no of edges

S.C: $O(n)$

source → destination → valid

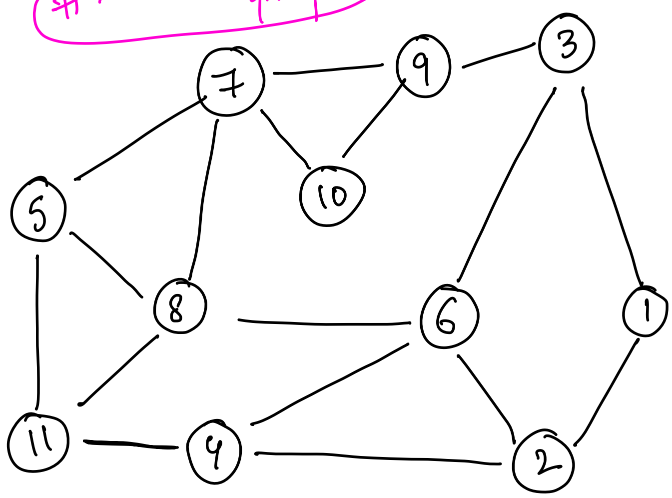
{ go from source to destination }

start doing BFS from source

if destination is already visited ⇒ $\boxed{\text{True}}$

Q source → min distance to all other nodes

unweighted graph

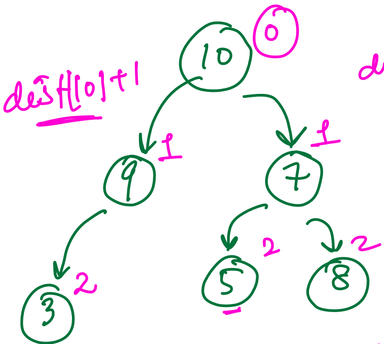


distance array

dist[10]+1

dist[9]=1

dist[7]=1



dist[3] = dist[9]+

```
queue <int> q;
bool visited [n+1];      // initially false
int dist [n+1];

dist [source] = 0;
q.push(source);
visited [source] = true;

while ( ! q.empty())
{
        int u = q.front();
        q.pop();
        print (u);

        for (int i=0; i< graph[u].size(); i++)
        {
                int v = grap[u][i];
                if ( ! visited [v])
                {
                        dist[v] = dist[u]+1;
                        visited [v] = true;
                        q.push (v);
                }
        }
}
```