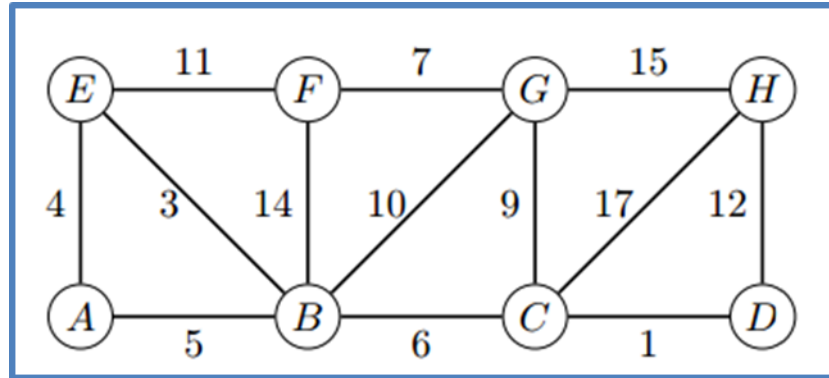
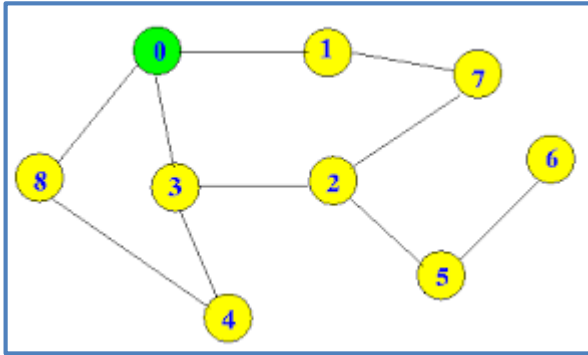


# TRAVERSAL OF UNDIRECTED GRAPHS



**Partha P Chakrabarti**  
Indian Institute of Technology Kharagpur

# Undirected Graph

An Undirected Graph  $G = (V, E)$  consists of the following:

- A set of Vertices or Nodes  $V$
- A set of Edges  $E$  where each edge connects two vertices of  $V$

**Example: Figure 1**

$V = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$

$E = \{(0,1), (0,8), (0,3), (1,7), (2,3), (2,5), (2,7), (3,4), (4,8), (5,6)\}$

**Successor Function:**  $\text{succ}(i) = \{\text{set of nodes to which node } i \text{ is connected}\}$

**Example:  $\text{Succ}(2) = \{3, 5, 7\}$**

**Weighted Undirected Graphs:** Such Graphs may have weights on edges (Figure 2)

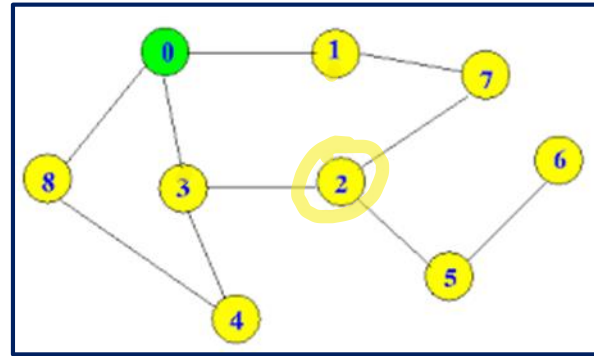


Figure 1

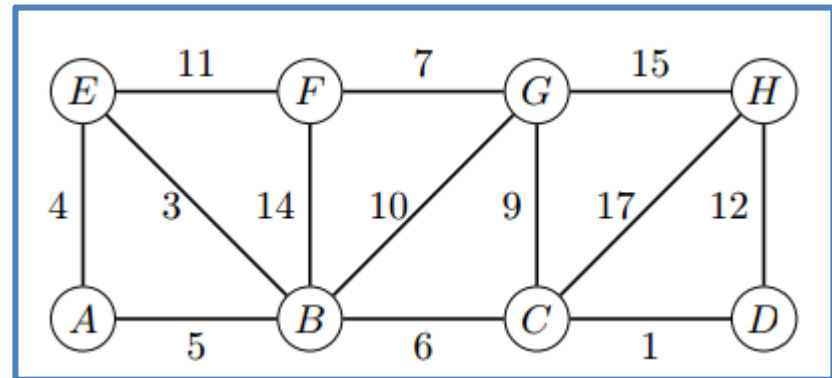


Figure 2

# Problems on Undirected Graphs

Reachability, Path, Cycle / Tree Detection, Connected Components, Bi-Connected Components, Spanning Tree, Shortest Path, Maximum Flow, Vertex Cover, Edge Cover, Travelling Salesperson,

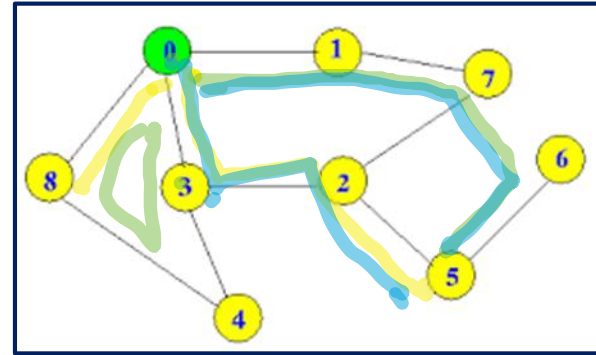


Figure 1

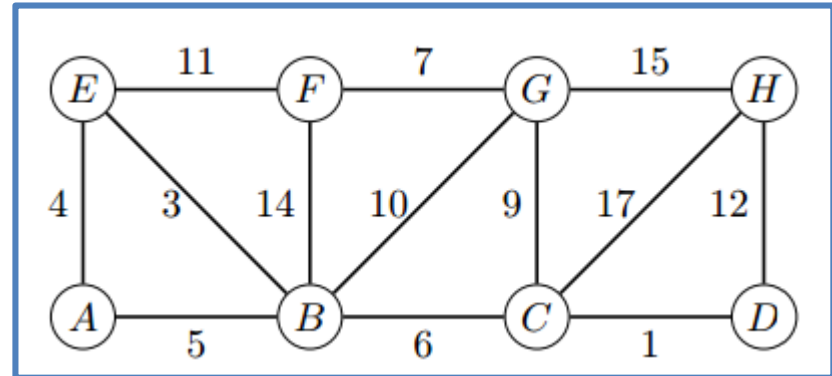
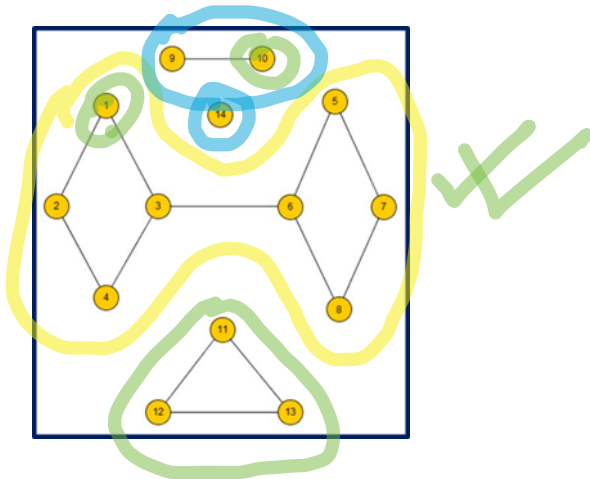


Figure 2

# Basic Traversal Algorithm (Depth First Search) I

Global Data:  $G = (V, E)$

**visited [i]** indicates if node i is visited. For all nodes j **visited [j]** is initialized to 0

**succ(i)** = {set of nodes to which node i is connected}

**Dfs(node)** {

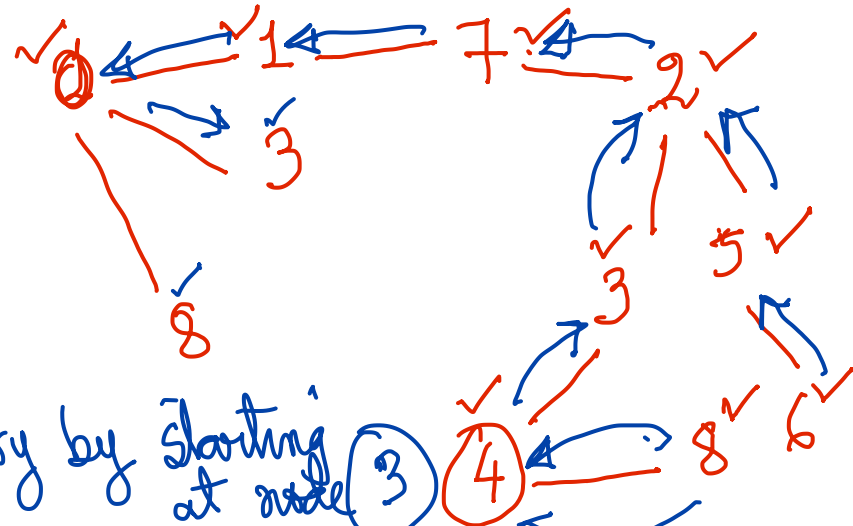
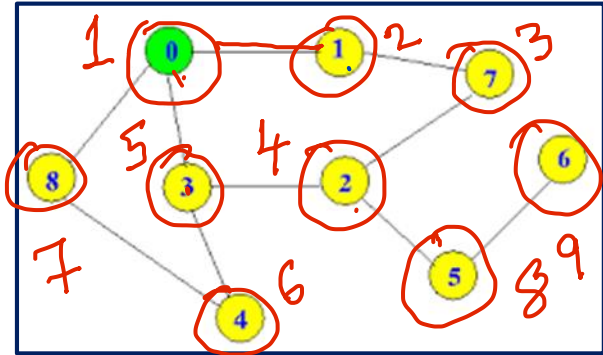
visited[node] = 1; ✓

for each j in succ(node) do {

if (visited [j] == 0) **Dfs(j)**

}

}



# Basic Traversal Algorithm (Depth First Search) II

Global Data:  $G = (V, E)$

visited [i] indicates if node i is visited. For all nodes j visited [j] is initialized to 0

succ(i) = {set of nodes to which node i is connected}

Dfs(node) {

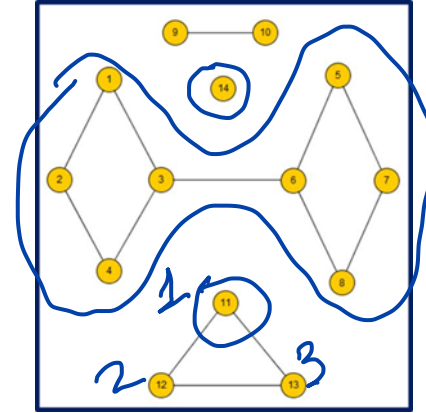
visited[node] = 1;

for each j in succ(node) do {

if (visited [j] == 0) Dfs(j)

}

}



# Basic Traversal Algorithm (Depth First Search) III

Global Data:  $G = (V, E)$

visited [i] indicates if node i is visited. For all nodes j visited [j] is initialized to 0

succ(i) = {set of nodes to which node i is connected}

Dfs(node) {

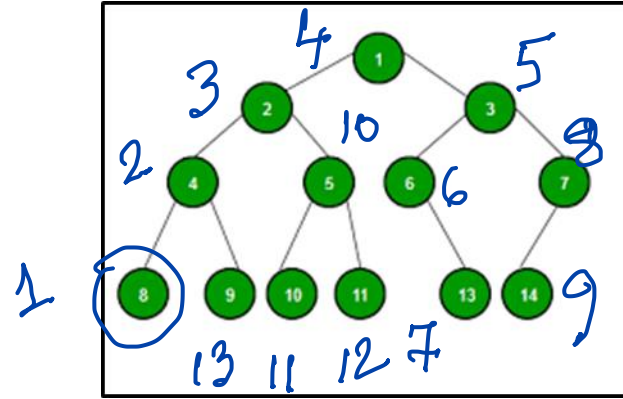
visited[node] = 1;

for each j in succ(node) do {

if (visited [j] == 0) Dfs(j)

}

}



# Cycle Detection

Global Data:  $G = (V, E)$

visited [i] indicates if node i is visited. For all nodes j visited [j] is initialized to 0

succ(i) = {set of nodes to which node i is connected}

Dfs(node) {

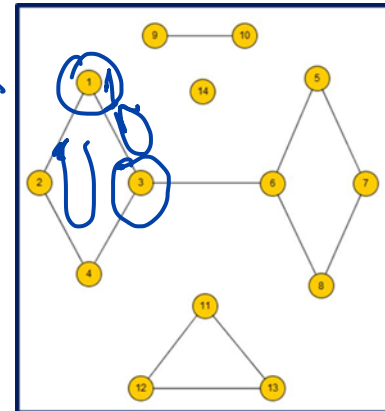
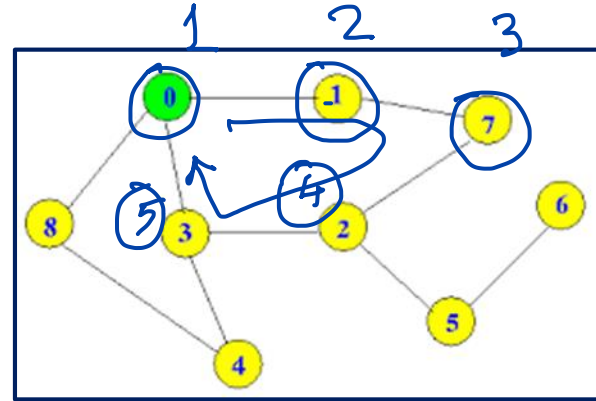
visited[node] = 1;

for each j in succ(node) do {

if (visited [j] == 0) Dfs(j)

}

// Cycle Detection //



if visited[j] == 1 & ~~not~~  
"cycle"

there is no edge between





# Connected Components

Global Data:  $G = (V, E)$

Visited[i], comp[i] all initialized to 0

count = 0; ✓✓

Algorithm components() {

for each node k do {

if visited [k] == 0 { count = count + 1; ✓

DfComp(k) }

DfComp(node) {

visited[node] = 1; comp[node] = count; ✓

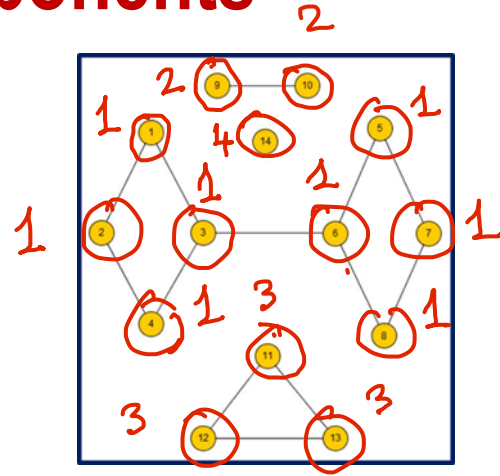
for each j in succ(node) do {

if (visited [j] == 0) DfComp(j)

}

}

$O(|V| + |E|)$



4 connected components  
and the count value  
is the component  
number

# Depth-First Numbering & Time Stamping

Global Data:  $G = (V, E)$

Visited[i], comp[i] all initialized to 0  
count = 0;

Algorithm components() {

for each node k do {

if visited [k] == 0 { count = count + 1;  
DfComp\_S(k) }

DfComp(node) {

visited[node] = 1; comp[node] = count;  $t_1$

for each j in succ(node) do {  
if (visited [j] == 0) {

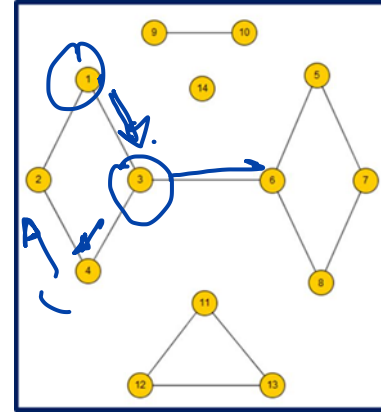
DfComp(j)

$t_2$

}

}

}



Depth-first numbering  
and  
Entry-Exit Time Stamps  
Bi-Connected Components

# Breadth-First Search

Global Data:  $G = (V, E)$

Visited[i] all initialized to 0

Queue Q initially {}

BFS(k) {

visited[k] = 1; Q = {k};

While Q != {} {

j = DeQueue(Q);

if visited[j] == 0 {

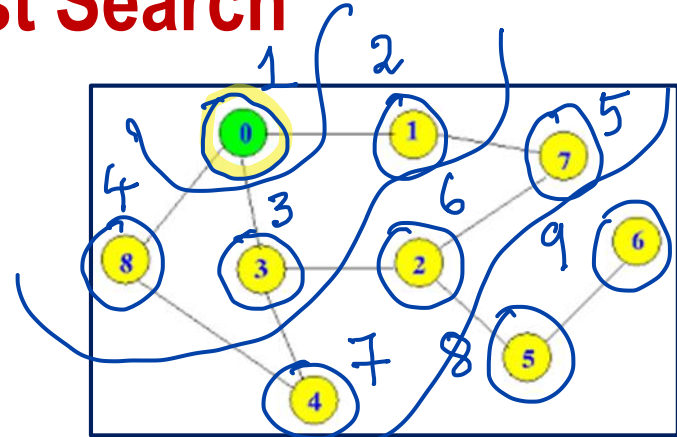
visited[j] = 1;

For each k in succ(j) EnQueue(Q,k);

}

/Parent links, Shortest Length Path Finding in unweighted graphs/

$O(|V| + |E|)$



1. Q = {0}
2. Q = {1, 3, 8}
3. Q = {3, 8, 7}
4. Q = {8, 7, 2, 4}
5. Q = {7, 2, 4}
6. Q = {2, 4}
7. Q = {4, 5}
8. Q = {5}
9. Q = {6}
10. Q = {}

# Pathfinding in Weighted Undirected Graphs I

Global Data:  $G = (V, E)$

Visited[i] all initialized to 0,

Cost[j] all initialized to INFINITY

Ordered Queue Q initially {} ✓✓

BFSW(k) {

visited [k] = 1; cost [k] = 0; Q = {k};

While Q != {} {

  j = DeQueue (Q);

  if visited[j] == 0 {

    visited [j] = 1;

    For each k in succ (j) {

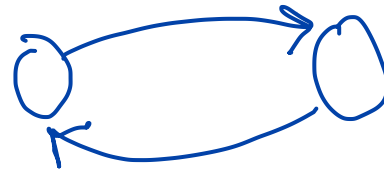
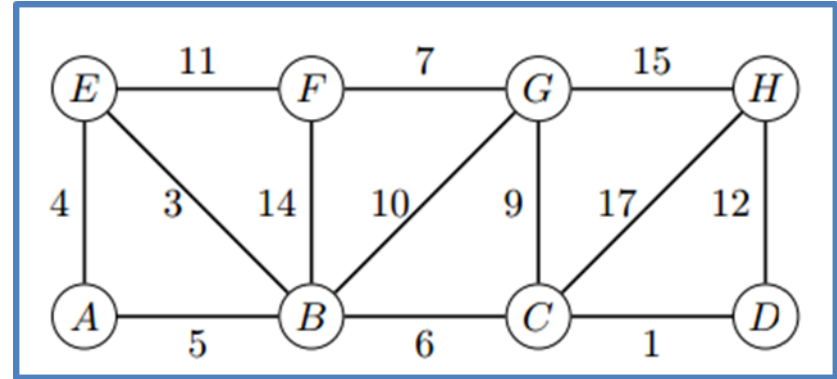
      if cost[k] > cost[j] + c[j,k]

        cost[k] = cost[j] + c[j,k];

      EnQueue(Q,k);

    }

  }



Dijkstra's Shortest Path Algorithm

# Pathfinding in Weighted Undirected Graphs II

Global Data:  $G = (V, E)$

Visited[i] all initialized to 0,

Cost[j] all initialized to INFINITY

Ordered Queue Q initially {}

BFSW(k) {

visited [k] = 1; cost [k] = 0; Q = {k};

While Q != {} {

    j = DeQueue (Q);

    if visited[j] == 0 {

        visited [j] = 1;

        For each k in succ (j) {

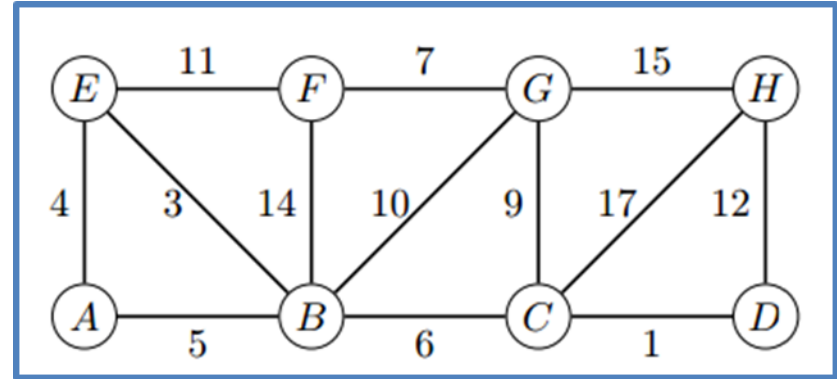
            if cost[k] > cost[j] + c[j,k]

                cost[k] = cost[j] + c[j,k];

            EnQueue(Q,k);

        }

    }



All pairs paths or  
Transitive Closure  
All pair shortest paths

**Thank you**