Graph Traversal

- Depth-First Search
  - Goes deep first
  - Stack / Recursion
  - Pathfinding, topological sort
- Breadth-First Search
  - Explores level by level
  - Queue
  - Shortest path in unweighted graphs

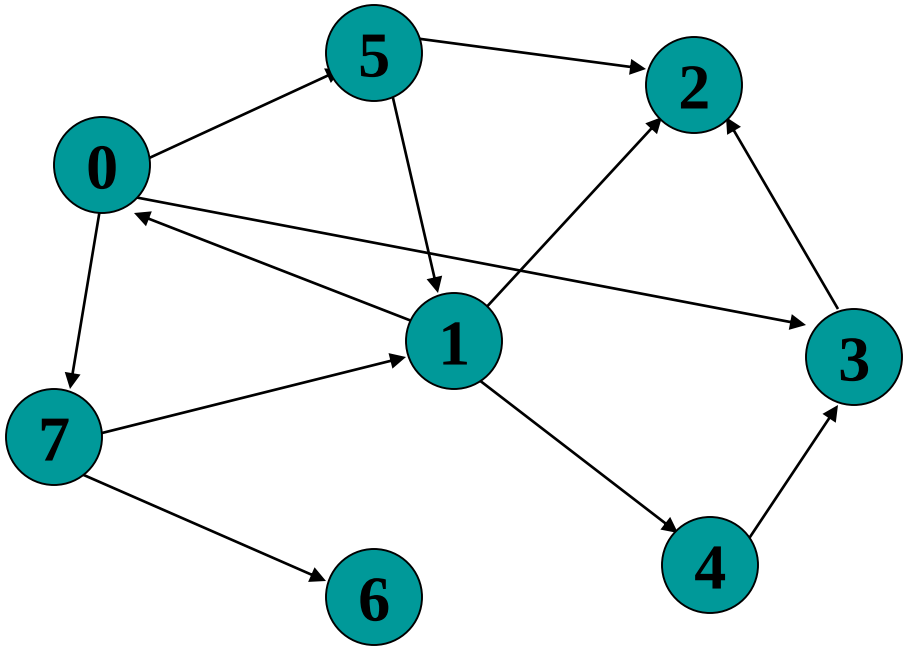
Overview

- Goal
  - To systematically visit the nodes of a graph
- A tree is a directed, acyclic, graph (DAG)
- If the graph is a tree,
  - DFS is exhibited by preorder, postorder, and (for binary trees) inorder traversals
  - BFS is exhibited by level-order traversal

Depth-First Search

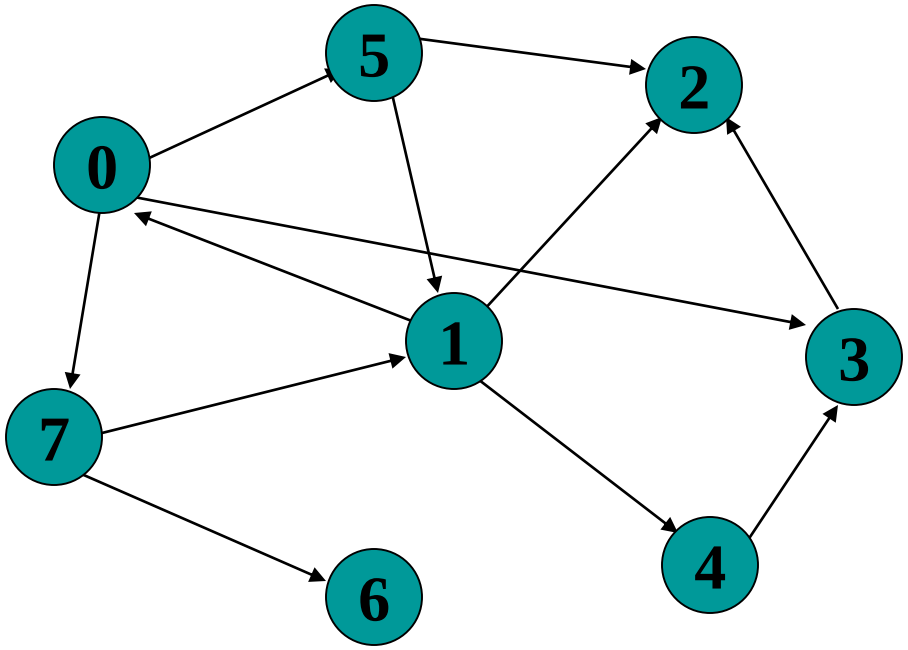
- **Initialize** all vertices in the graph as *unvisited*.
- **Start** from the given source vertex v.
- **Mark** the current vertex as *visited*.
- **Process** the current vertex (e.g., print it).
- **For each adjacent vertex** u of the current vertex v:
- If u is *unvisited*, then **recursively apply DFS** on u.
- **Repeat** until all vertices reachable from the source have been visited.
- If the graph is **disconnected**, repeat DFS for the remaining unvisited vertices.

Example



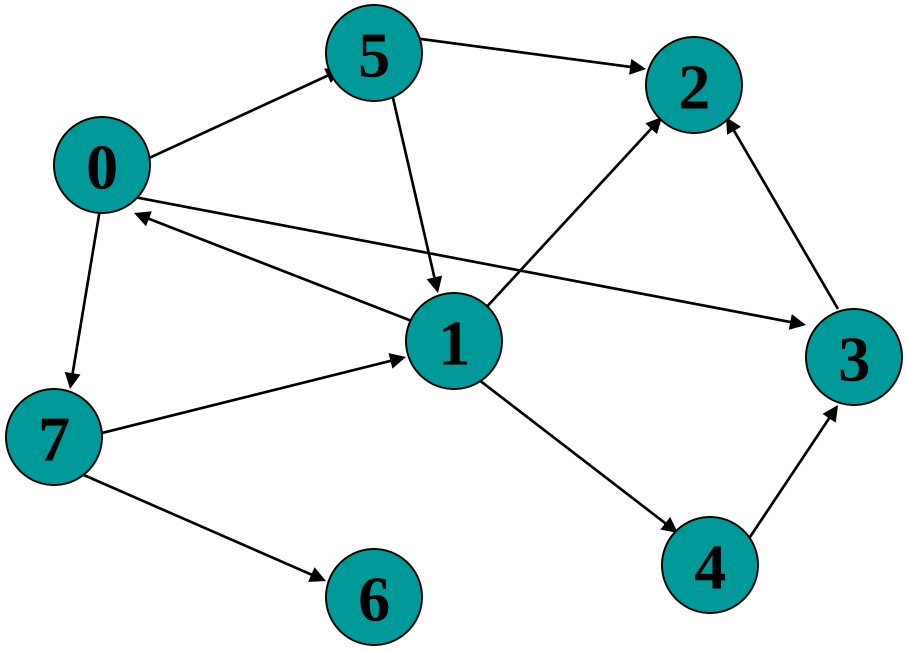
Policy: Visit adjacent nodes in increasing index order

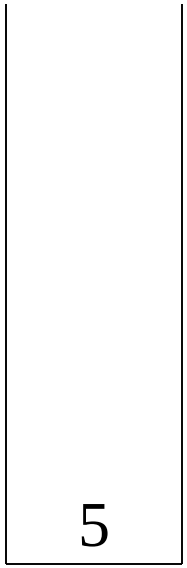
Preorder DFS: Start with Node 5



5 1 0 3 2 7 6 4

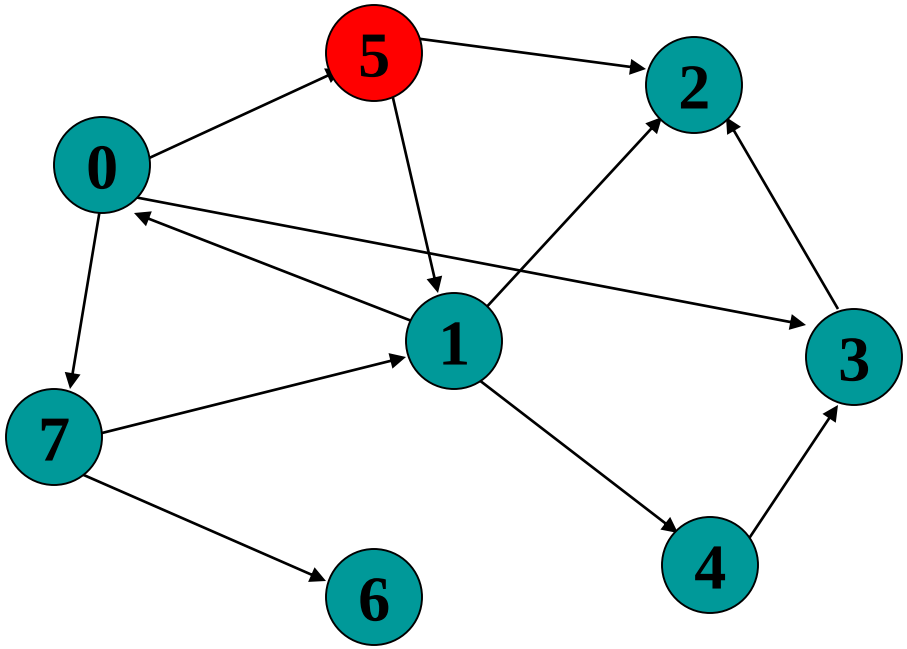
Preorder DFS: Start with Node 5





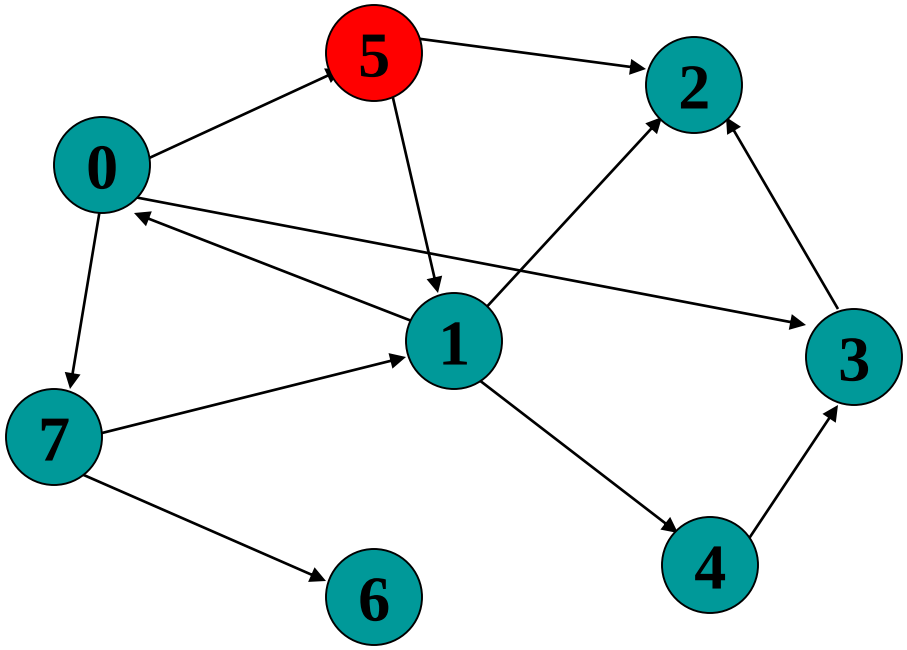
Push 5

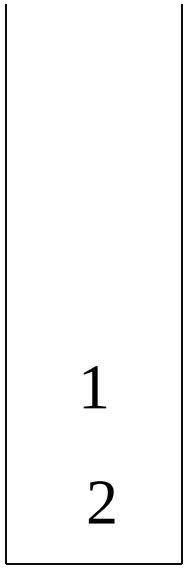
Preorder DFS: Start with Node 5



Pop/Visit/Mark 5

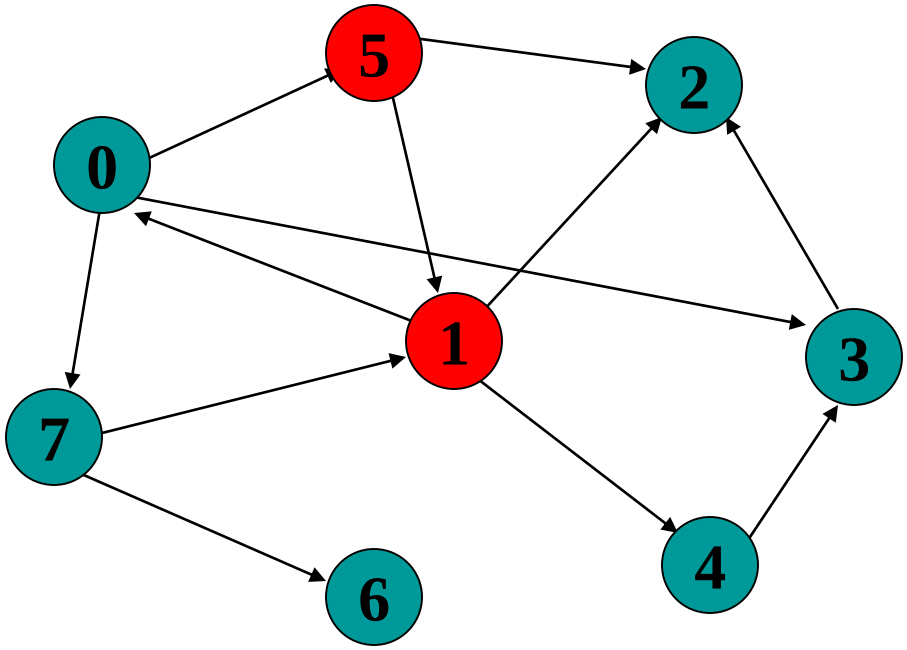
Preorder DFS: Start with Node 5

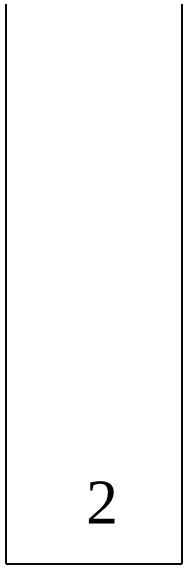




Push 2, Push 1

Preorder DFS: Start with Node 5

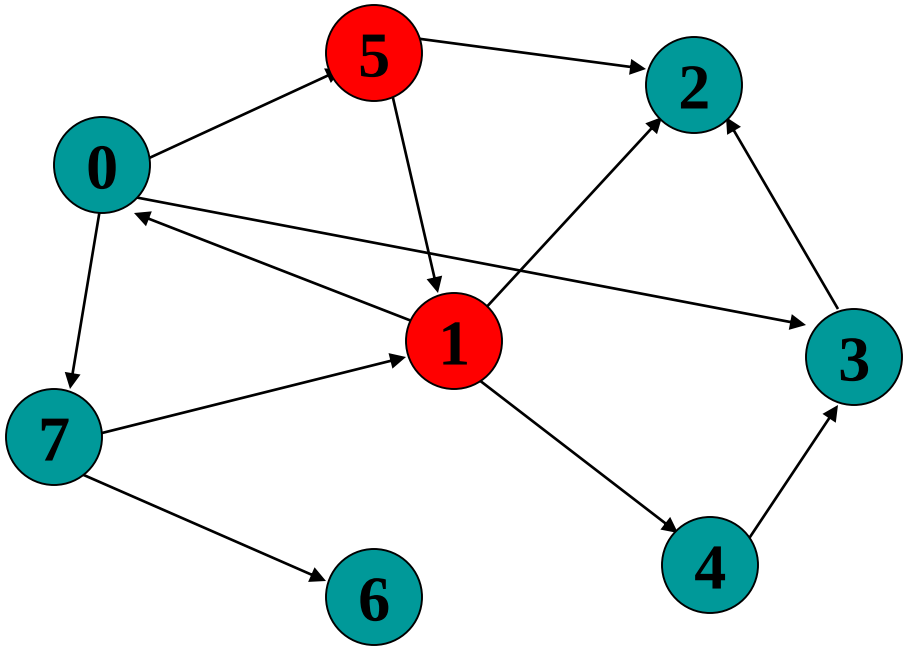


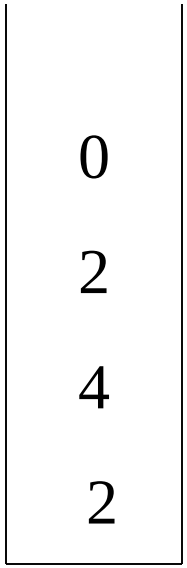


Pop/Visit/Mark 1

5 1

Preorder DFS: Start with Node 5

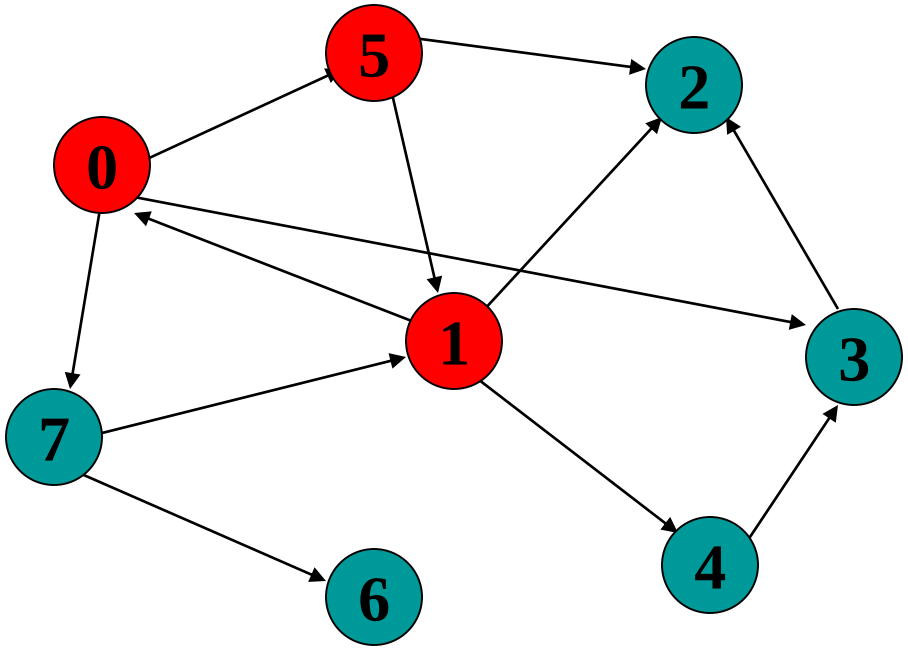




Push 4, Push 2, Push 0

5 1

Preorder DFS: Start with Node 5

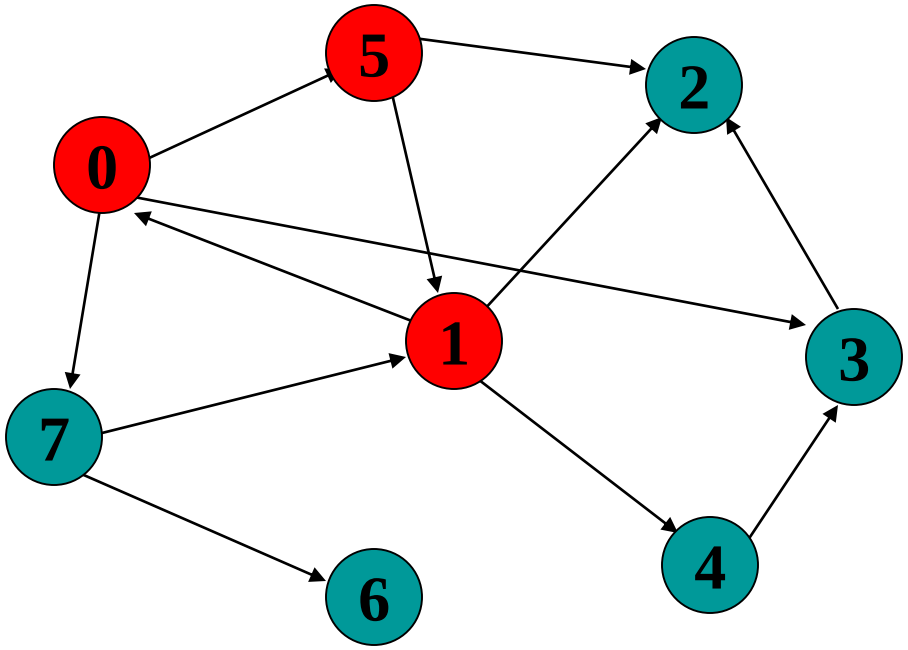


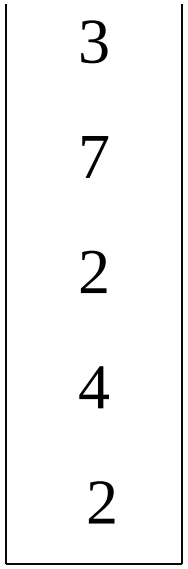
2
4
2

Pop/Visit/Mark 0

5 1 0

Preorder DFS: Start with Node 5

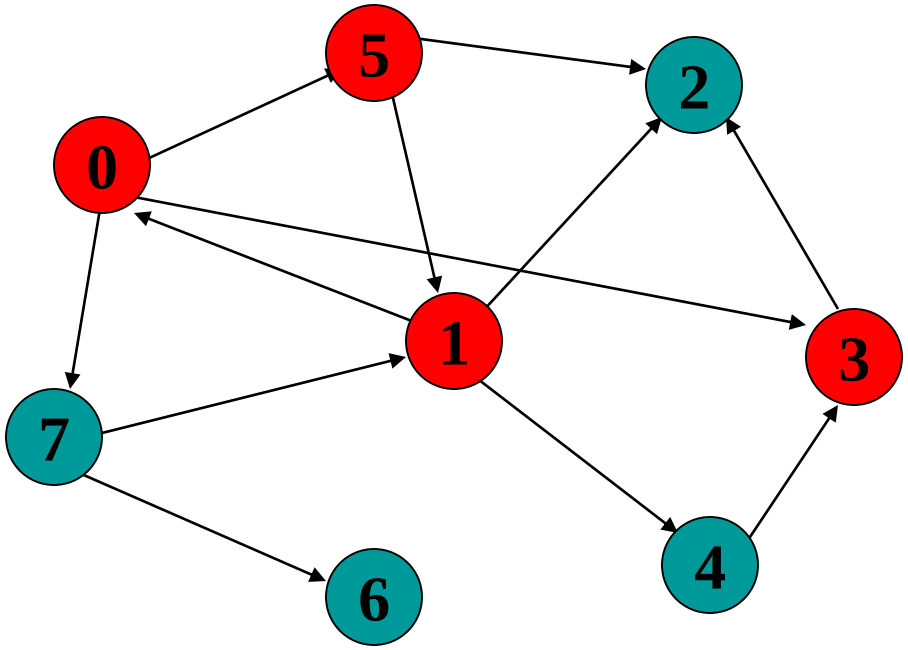




Push 7, Push 3

5 1 0

Preorder DFS: Start with Node 5

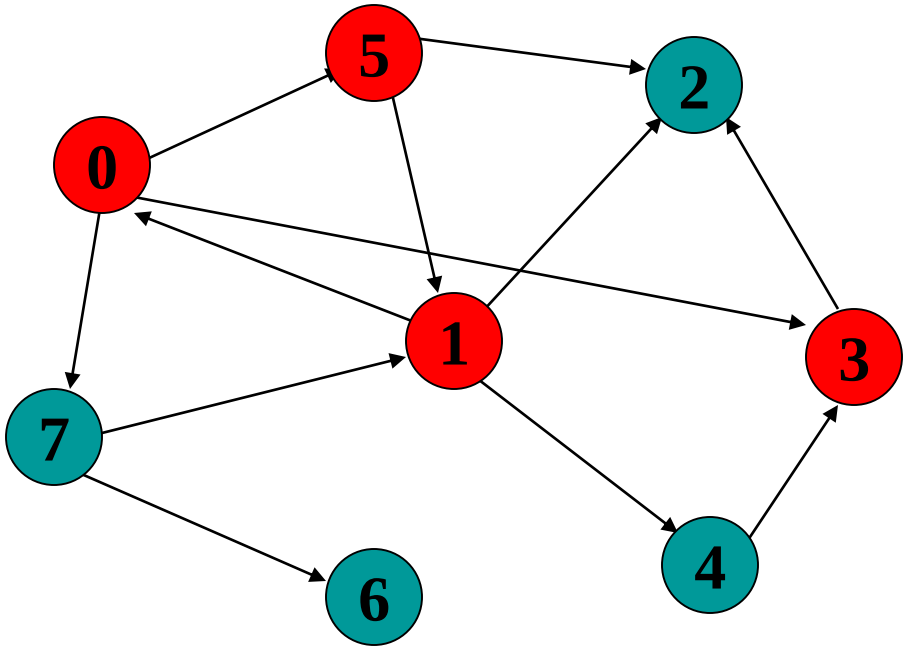


7
2
4
2

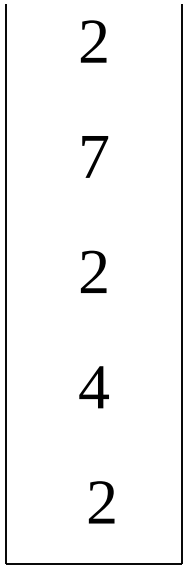
Pop/Visit/Mark 3

5 1 0 3

Preorder DFS: Start with Node 5

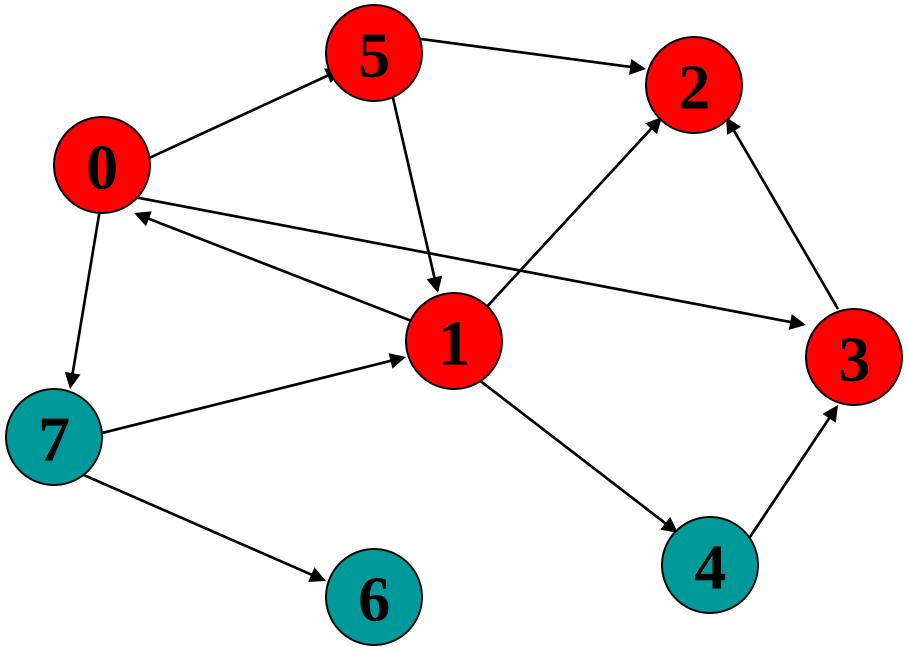


5 1 0 3



Push 2

Preorder DFS: Start with Node 5

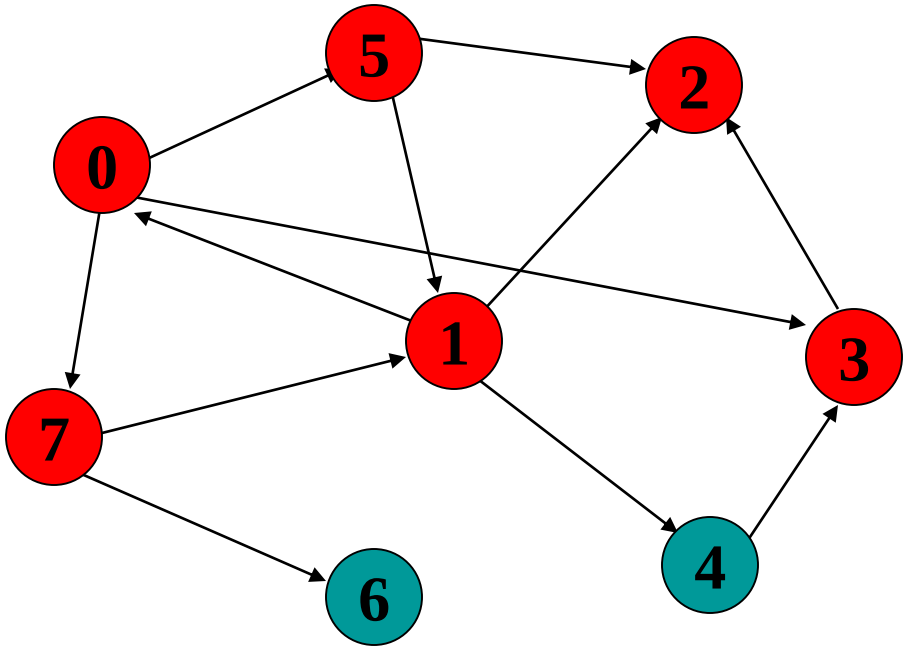


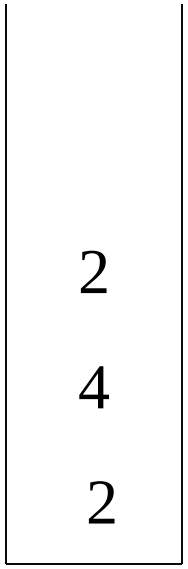
7
2
4
2

Pop/Mark/Visit 2

5 1 0 3 2

Preorder DFS: Start with Node 5

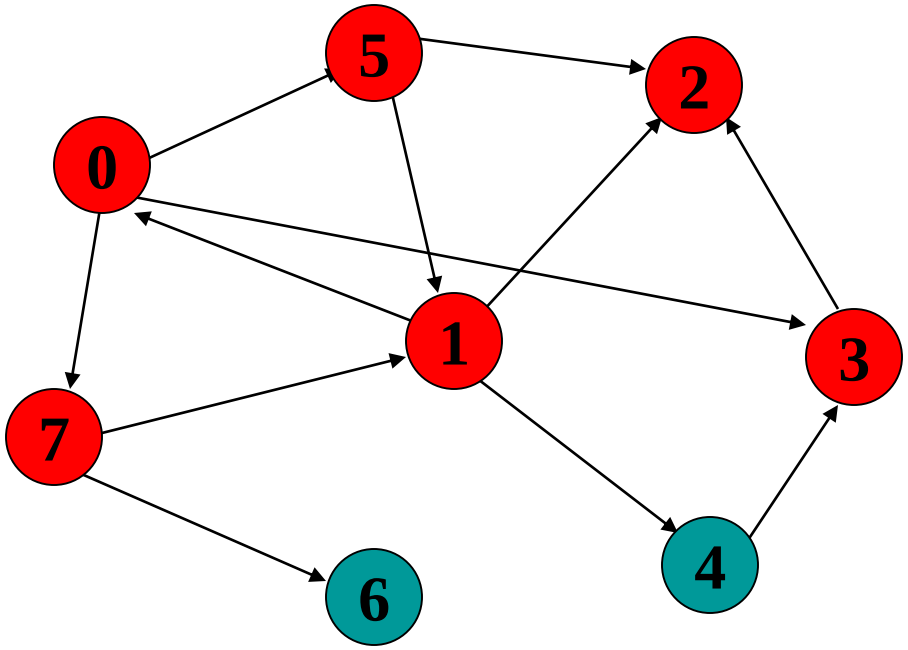


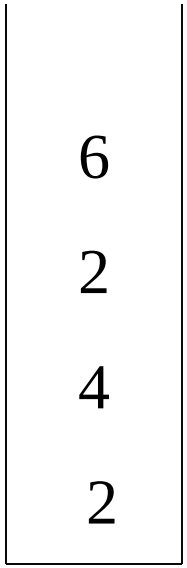


Pop/Mark/Visit 7

5 1 0 3 2 7

Preorder DFS: Start with Node 5

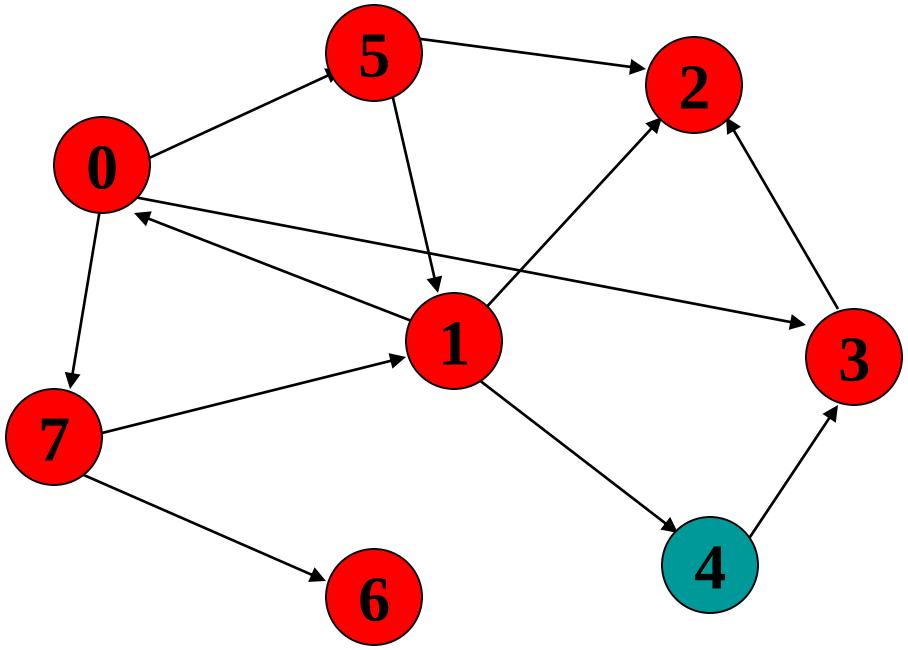


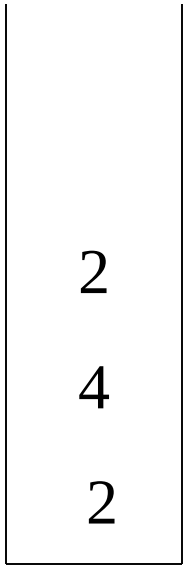


Push 6

5 1 0 3 2 7

Preorder DFS: Start with Node 5

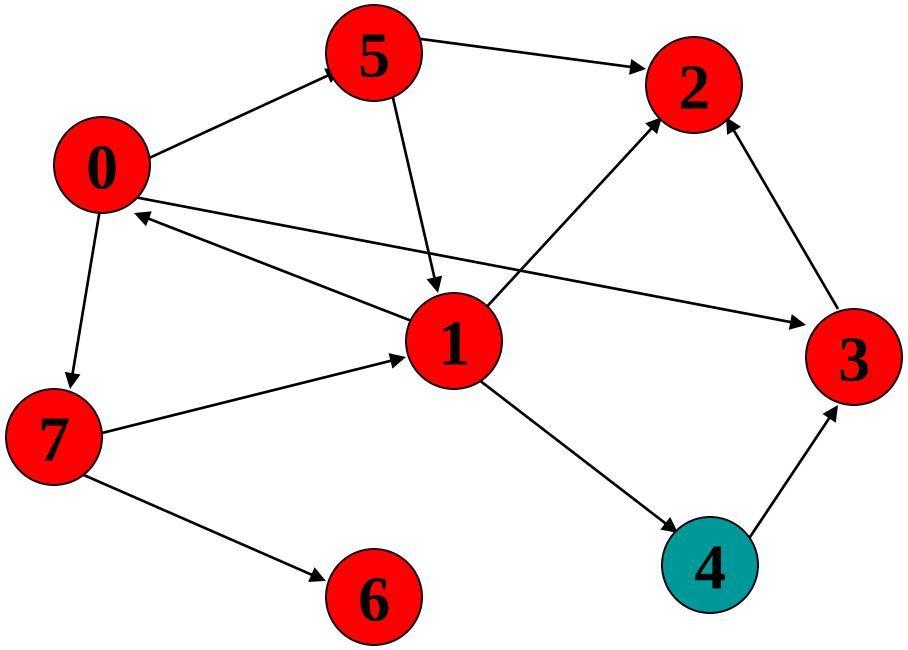


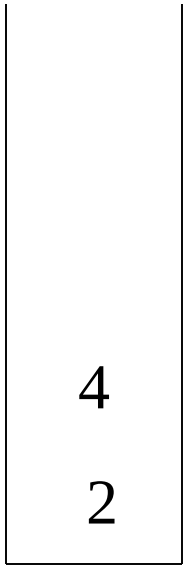


Pop/Mark/Visit 6

5 1 0 3 2 7 6

Preorder DFS: Start with Node 5

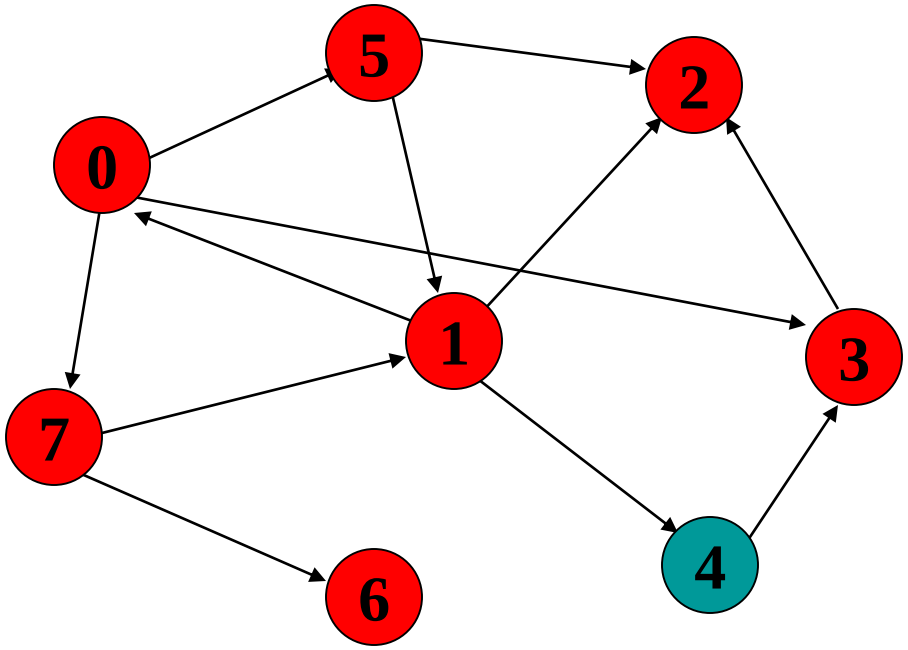


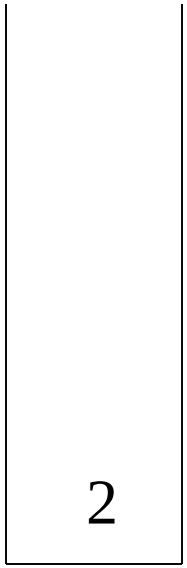


Pop (don't visit) 2

5 1 0 3 2 7 6

Preorder DFS: Start with Node 5

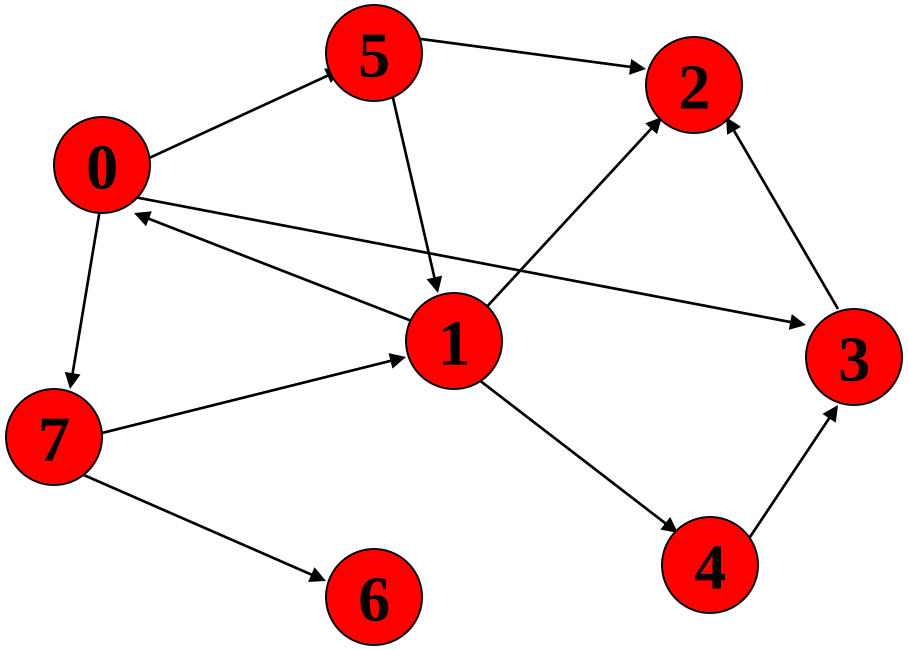


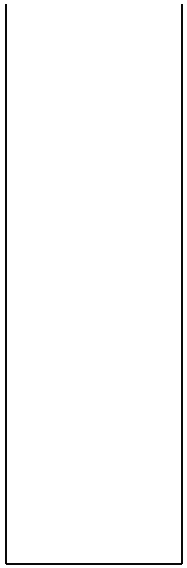


Pop/Mark/Visit 4

5 1 0 3 2 7 6 4

Preorder DFS: Start with Node 5

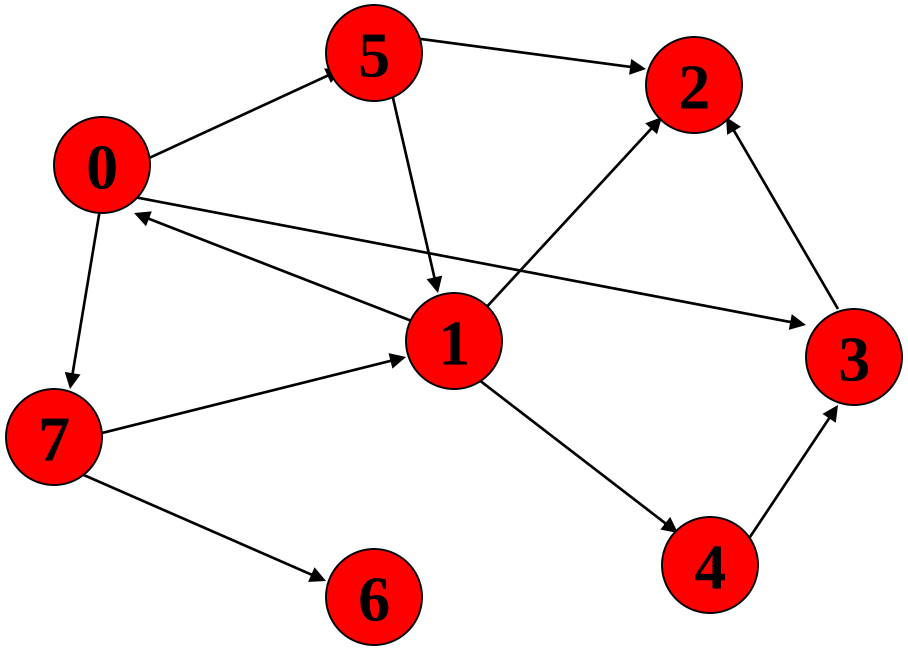


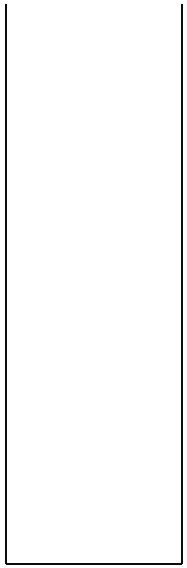


Pop (don’t visit) 2

5 1 0 3 2 7 6 4

Preorder DFS: Start with Node 5

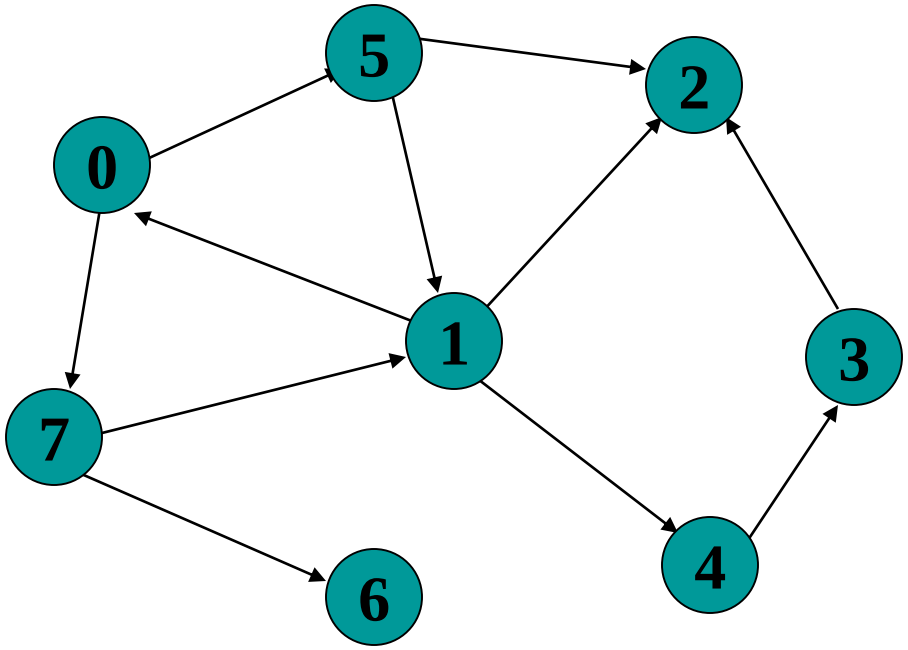




Done

5 1 0 3 2 7 6 4

Note: edge (0,3) removed



5 1 0 7 6 2 4 3

Depth-First Search

Graph.h

```
#ifndef GRAPH_H
#define GRAPH_H

// Define maximum number of vertices
#define MAX_V 20

// Function declarations
void addEdge(int graph[MAX_V][MAX_V], int src, int dest);
void DFS(int graph[MAX_V][MAX_V], int visited[MAX_V], int
vertex, int vertices);

#endif
```

Depth-First Search

Graph.c

```
#include <stdio.h>
#include "graph.h"
/*
 * Function: addEdge
 * -----------------
 * Adds a directed edge from src to dest.
 * In adjacency matrix representation:
 *     graph[src][dest] = 1
 * (No reverse edge is added because it is a directed graph)
 */
void addEdge(int graph[MAX_V][MAX_V], int src, int dest)
{
    graph[src][dest] = 1;   // Directed edge from src → dest
}
```

Adjacency Matrix

```
   0 1 2 3 4
0 : 0 1 1 0 0
1 : 0 0 0 1 0
2 : 0 0 0 1 0
3 : 0 0 0 0 1
4 : 0 0 1 0 0
```

Depth-First Search

Graph.c

```
/*
 * Function: DFS
 * -------------
 * Performs Depth First Search starting from a given vertex.
 *
 * Parameters:
 * - graph   : adjacency matrix representation of the graph
 * - visited : keeps track of visited vertices
 * - vertex  : the current vertex being explored
 * - vertices: total number of vertices in the graph
 *
 * Working:
 * 1. Mark current vertex as visited.
 * 2. Print current vertex.
 * 3. Recursively visit all unvisited adjacent vertices.
 */
```

Depth-First Search

Graph.c

```
void DFS(int graph[MAX_V][MAX_V], int visited[MAX_V], int
vertex, int vertices)
{
    printf("%d ", vertex);      // Print current vertex
    visited[vertex] = 1;        // Mark as visited

    // Explore all adjacent vertices
    for (int i = 0; i < vertices; i++) {
        if (graph[vertex][i] == 1 && !visited[i]) {
            DFS(graph, visited, i, vertices);
        }
    }
}
```

Depth-First Search

main.c

```
int main() {
    int vertices, edges, src, dest, startVertex;
    int graph[MAX_V][MAX_V] = {0};  // Initialize adjacency matrix
    int visited[MAX_V] = {0};       // Track visited vertices

    printf("Enter number of vertices: ");
    scanf("%d", &vertices);

    printf("Enter number of edges: ");
    scanf("%d", &edges);

    // Input edges
    for (int i = 0; i < edges; i++) {
        printf("Enter edge (src dest): ");
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);  // Directed edge
    }
```

Example Directed Graph

Number of vertices: **5 (0–4)**

Number of edges: **6**

Edges

0 → 1

0 → 2

1 → 3

2 → 3

3 → 4

4 → 2

Depth-First Search

main.c

```
printf("Enter starting vertex for DFS: ");
    scanf("%d", &startVertex);

    printf("DFS Traversal starting from vertex %d:\n", startVertex);
    DFS(graph, visited, startVertex, vertices);

    return 0;
}
```

DFS Traversal Starting from Vertex 0

We call DFS(graph, visited, 0, 5)

Depth-First Search

Edges

0 → 1
0 → 2
1 → 3
2 → 3
3 → 4
4 → 2

Adjacency Matrix

	0	1	2	3	4
0 :	0	1	1	0	0
1 :	0	0	0	1	0
2 :	0	0	0	1	0
3 :	0	0	0	0	1
4 :	0	0	1	0	0

DFS Traversal Starting from Vertex 0

We call DFS(graph, visited, 0, 5)

Step 1:
- Visit **0** → print 0
- Mark visited[0] = 1
- Adjacent to 0 → {1, 2}

Step 2 (explore 1):
- Call DFS(1)
- Visit **1** → print 1
- Mark visited[1] = 1
- Adjacent to 1 → {3}

Step 3 (explore 3 from 1):
- Call DFS(3)
- Visit **3** → print 3
- Mark visited[3] = 1
- Adjacent to 3 → {4}

Step 4 (explore 4 from 3):
- Call DFS(4)
- Visit **4** → print 4
- Mark visited[4] = 1
- Adjacent to 4 → {2}

Step 5 (explore 2 from 4):
- Call DFS(2)
- Visit **2** → print 2
- Mark visited[2] = 1
- Adjacent to 2 → {3}
- but 3 is already visited → stop.

DFS Traversal Order

0 1 3 4 2

Breadth-first Search

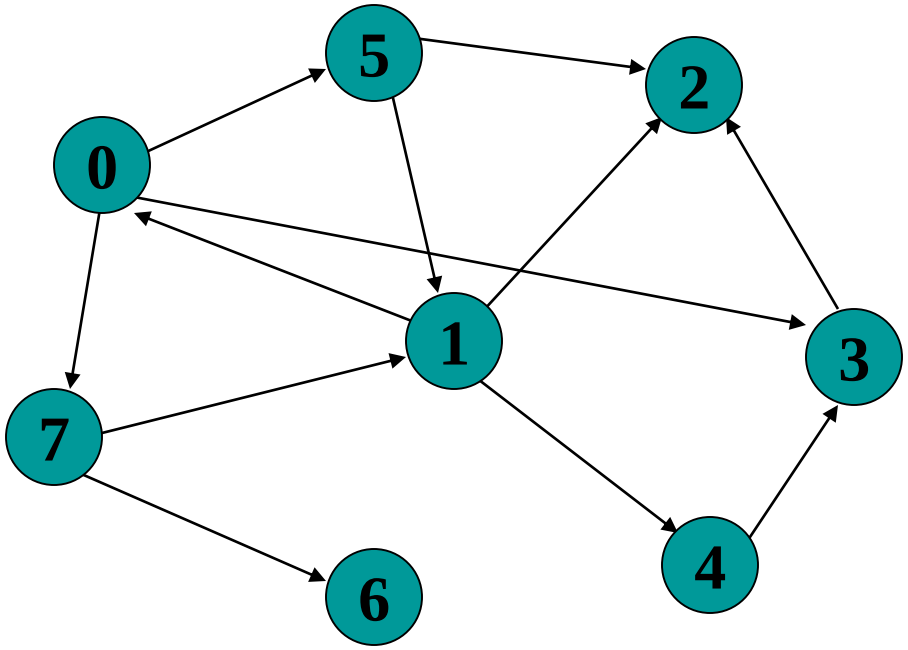
- Ripples in a pond
- Visit designated node
- Then visited unvisited nodes a distance *i* away, where *i* = 1, 2, 3, etc.
- For nodes the same distance away, visit nodes in systematic manner (eg. increasing index order)

Breadth-First Search

Input: Graph G=(V,E)G = (V, E)G=(V,E), starting vertex s
Output: BFS traversal of the graph
Steps:

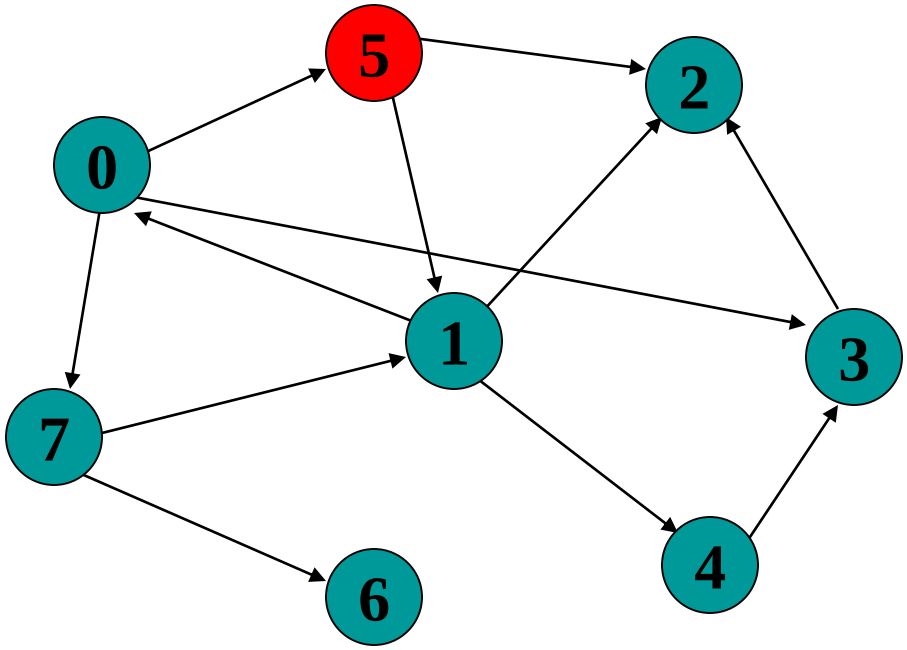
1. Initialize all vertices as *unvisited*.
2. Enqueue the starting vertex s and mark it visited.
3. While the queue is not empty:
    - Dequeue a vertex v.
    - Process v (print it).
    - For each adjacent vertex u of v:
        - If u is unvisited, mark it visited and enqueue it.

BFS: Start with Node 5

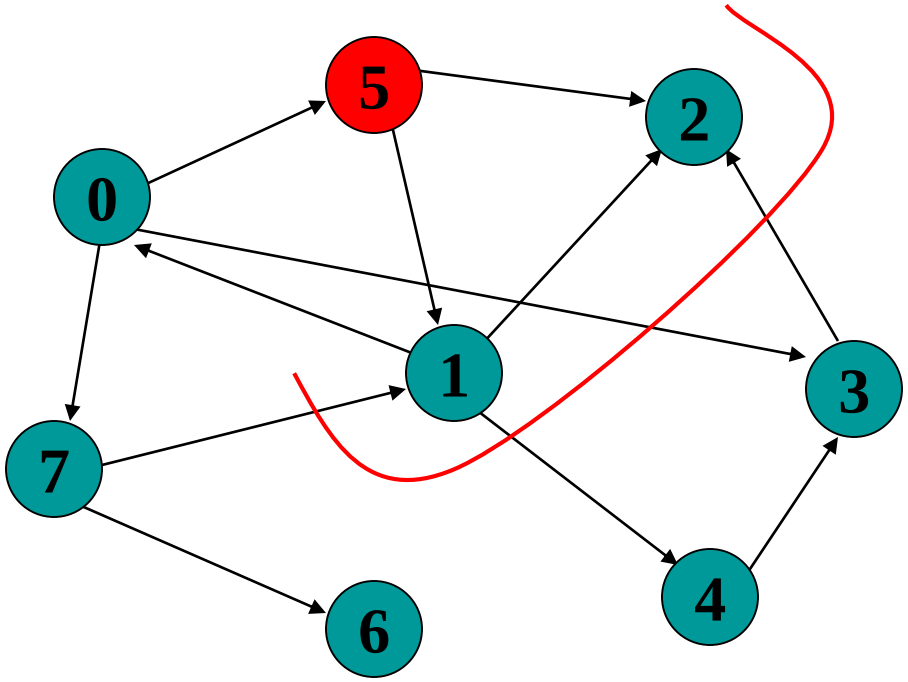


5 1 2 0 4 3 7 6

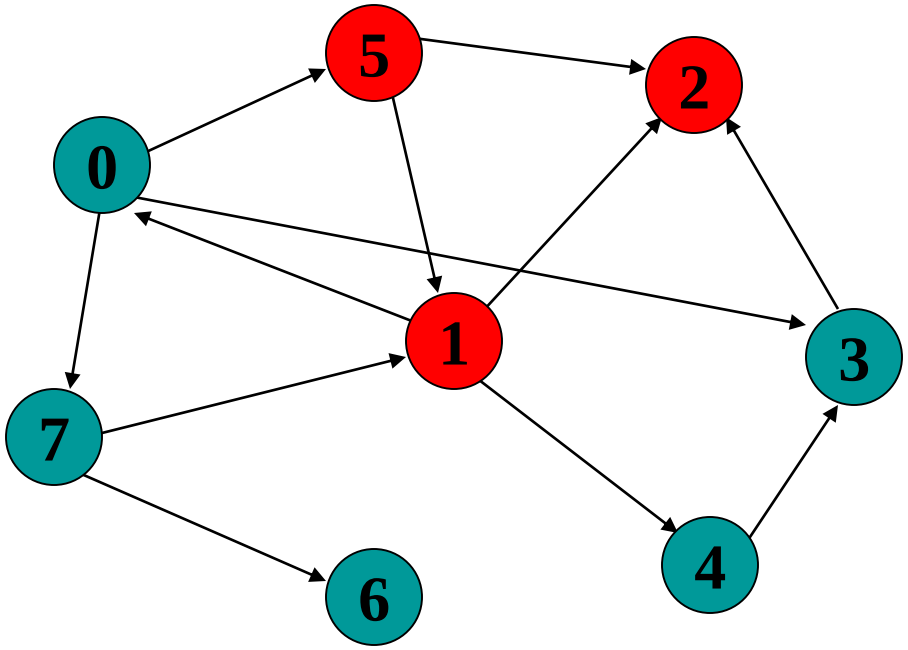
BFS: Start with Node 5



BFS: Node one-away

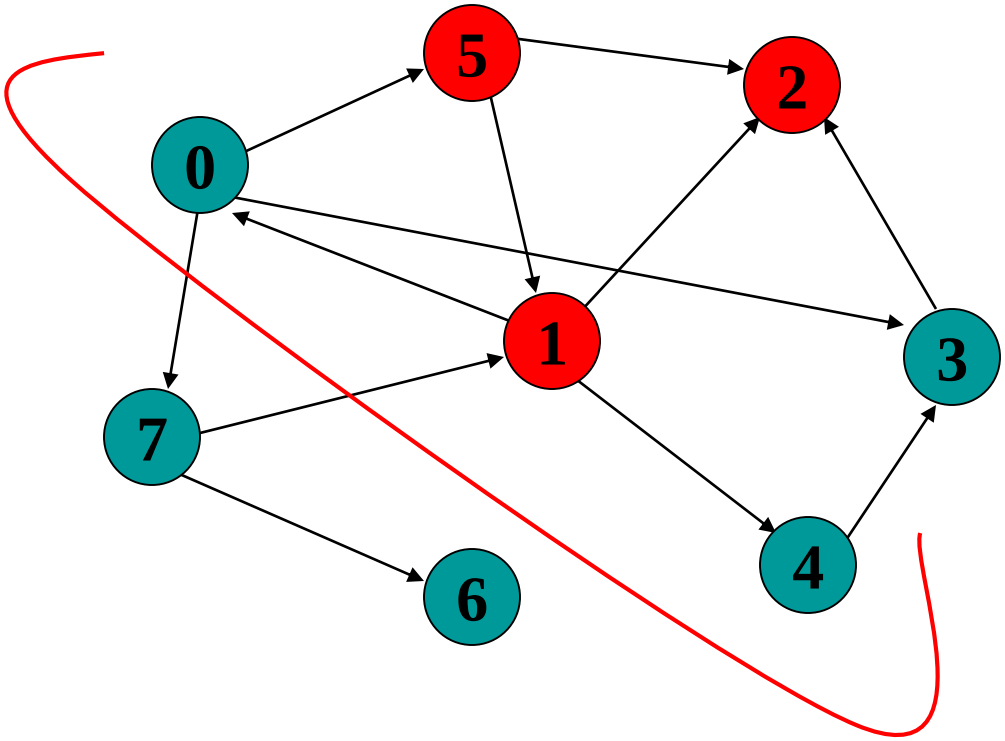


BFS: Visit 1 and 2



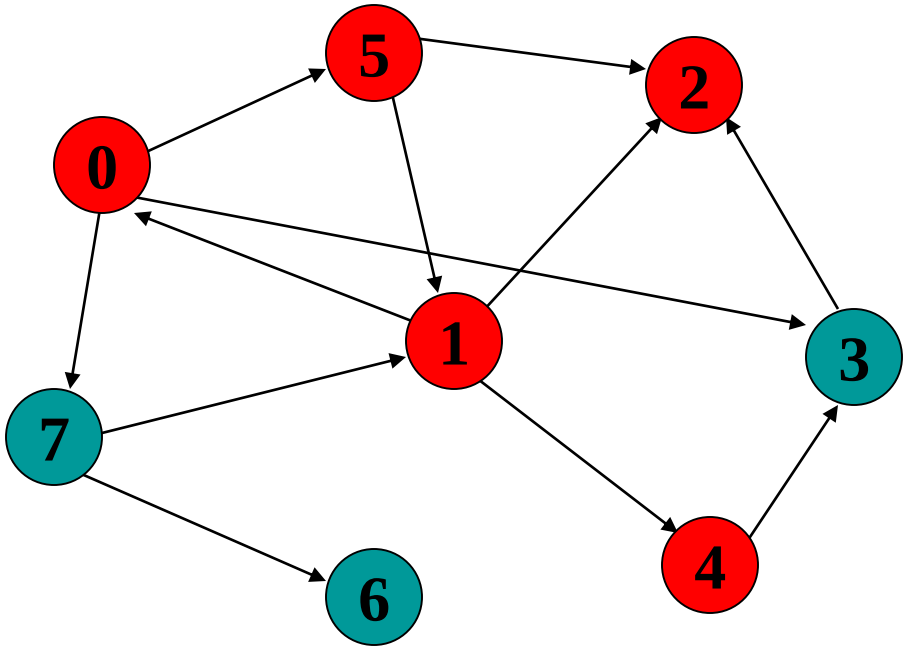
5 1 2

BFS: Nodes two-away



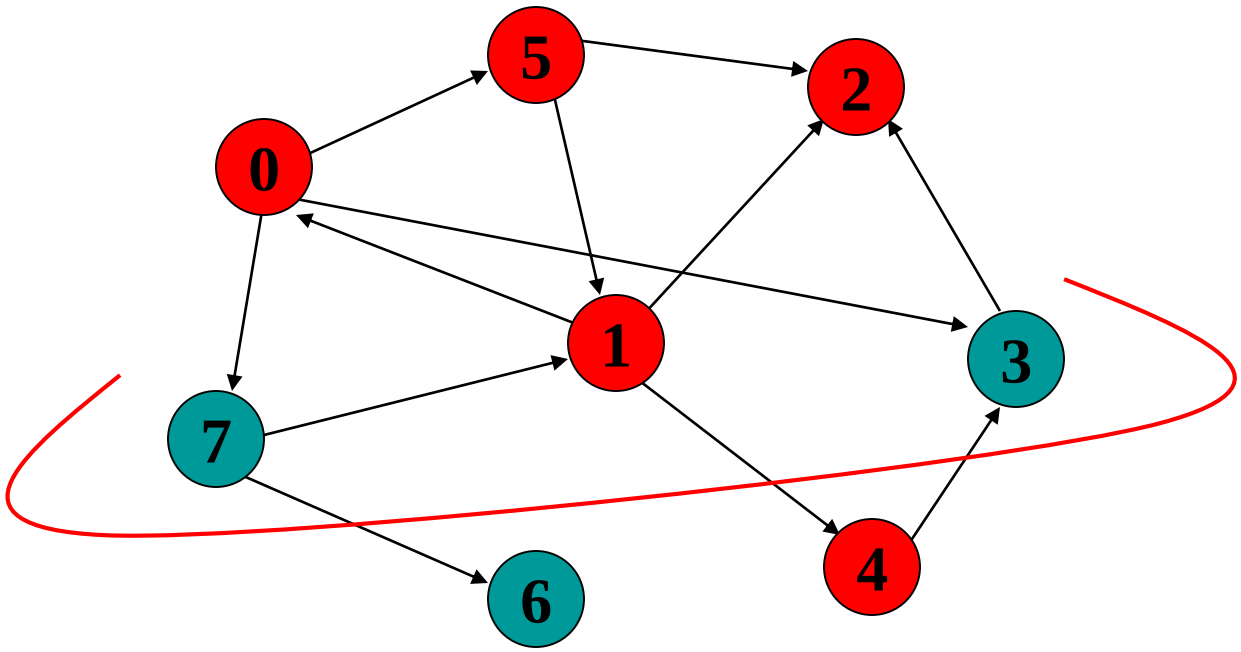
5 1 2

BFS: Visit 0 and 4



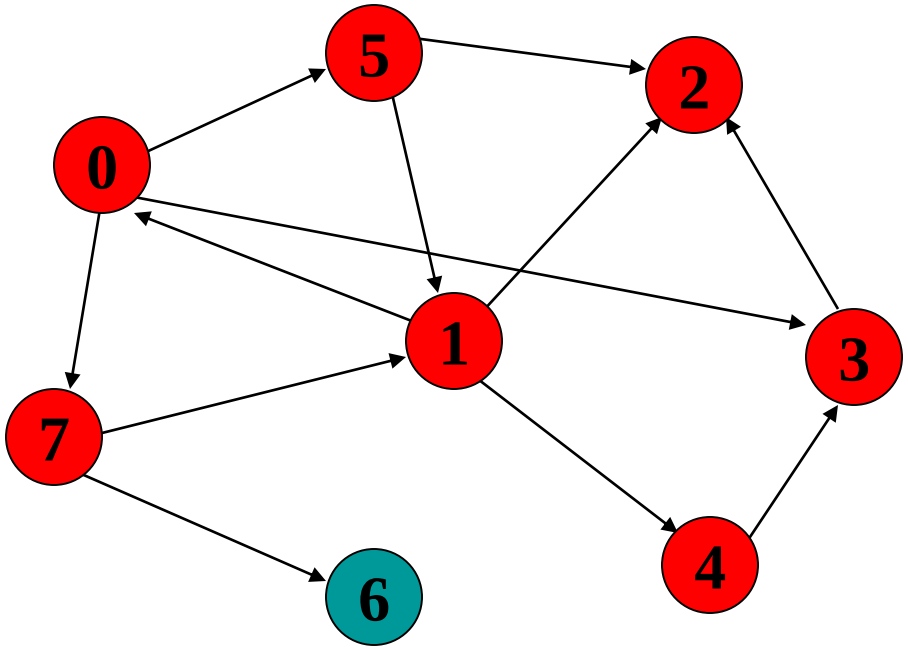
5 1 2 0 4

BFS: Nodes three-away



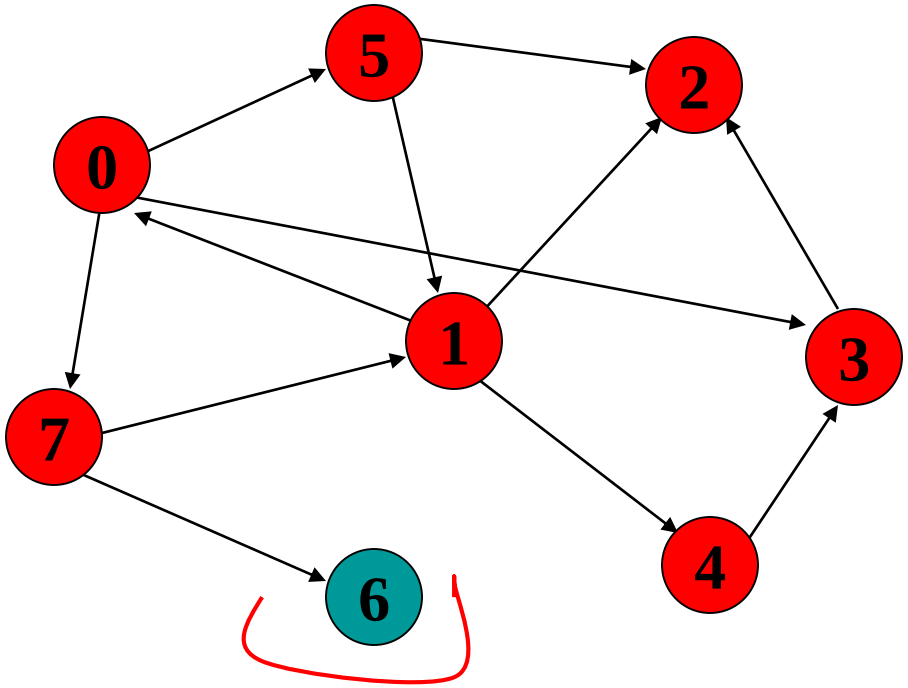
5 1 2 0 4

BFS: Visit nodes 3 and 7



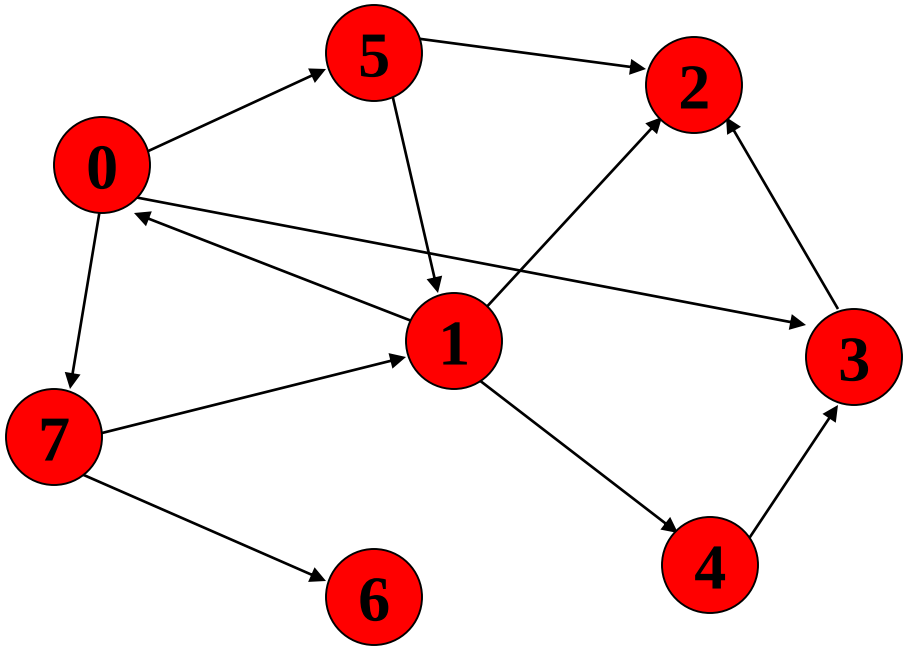
5 1 2 0 4 3 7

BFS: Node four-away



5 1 2 0 4 3 7

BFS: Visit 6



5 1 2 0 4 3 7 6

Breadth-First Search

Graph.h

```
#ifndef GRAPH_H
#define GRAPH_H

#define MAX_V 20   // Maximum number of vertices

// Function declarations
void addEdge(int graph[MAX_V][MAX_V], int src, int dest);
void BFS(int graph[MAX_V][MAX_V], int visited[MAX_V], int
startVertex, int vertices);

#endif
```

Breadth-First Search

Graph.c

```
#include <stdio.h>
#include "graph.h"

/*
 * Function: addEdge
 * -----------------
 * Adds a directed edge from src to dest.
 * For an undirected graph, also add graph[dest][src] = 1.
 */
void addEdge(int graph[MAX_V][MAX_V], int src, int dest) {
    graph[src][dest] = 1;   // Directed edge src → dest
    // For undirected graph, uncomment the next line:
    // graph[dest][src] = 1;
}
```

Adjacency Matrix

```
   0 1 2 3 4
0 : 0 1 1 0 0
1 : 0 0 0 1 0
2 : 0 0 0 1 0
3 : 0 0 0 0 1
4 : 0 0 1 0 0
```

Breadth-First Search

Graph.c

```
/*
 * Function: BFS
 * -------------
 * Performs Breadth First Search starting from startVertex.
 *
 * Parameters:
 * - graph   : adjacency matrix of the graph
 * - visited : array to track visited vertices
 * - startVertex : vertex to start BFS from
 * - vertices: total number of vertices
 *
 * Working:
 * 1. Create a queue.
 * 2. Enqueue startVertex and mark it visited.
 * 3. While queue is not empty:
 *    - Dequeue a vertex and print it.
 *    - Enqueue all its unvisited adjacent vertices.
 */
```

Breadth-First Search

Graph.c

```
void BFS(int graph[MAX_V][MAX_V], int visited[MAX_V], int
startVertex, int vertices) {
    int queue[MAX_V], front = 0, rear = 0;

    // Enqueue start vertex
    queue[rear++] = startVertex;
    visited[startVertex] = 1;
```

Breadth-First Search

Graph.c

```
while (front < rear) {
        int currentVertex = queue[front++]; // Dequeue
        printf("%d ", currentVertex);

        // Check all adjacent vertices
        for (int i = 0; i < vertices; i++) {
            if (graph[currentVertex][i] == 1 && !visited[i]) {
                queue[rear++] = i;   // Enqueue
                visited[i] = 1;      // Mark visited
            }
        }
    }
}
```

Breadth-First Search

main.c

```
int main() {
    int vertices, edges, src, dest, startVertex;
    int graph[MAX_V][MAX_V] = {0};  // Initialize adjacency matrix
    int visited[MAX_V] = {0};       // Track visited vertices

    printf("Enter number of vertices: ");
    scanf("%d", &vertices);

    printf("Enter number of edges: ");
    scanf("%d", &edges);

    // Input edges
    for (int i = 0; i < edges; i++) {
        printf("Enter edge (src dest): ");
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
    }
```

Example Directed Graph

Number of vertices: **5 (0–4)**

Number of edges: **6**

Edges

0 → 1

0 → 2

1 → 3

2 → 3

3 → 4

4 → 2

Breadth-First Search

main.c

```
printf("Enter starting vertex for BFS: ");
    scanf("%d", &startVertex);

    printf("BFS Traversal starting from vertex %d:\n", startVertex);
    BFS(graph, visited, startVertex, vertices);

    return 0;
}
```

Breadth-First Search

BFS(graph, visited, 0, 5)

Step 1: Initialization
- StartVertex = 0
- Mark visited[0] = 1
- Enqueue 0

Queue: [0]
Output: —

Step 2: Process 0
- Dequeue 0 → print 0
- Adjacent to 0 → {1, 2}
- Enqueue 1, Enqueue 2, mark visited

Queue: [1, 2]
Output: 0

Step 3: Process 1
- Dequeue 1 → print 1
- Adjacent to 1 → {3}
- Enqueue 3, mark visited

Queue: [2, 3]
Output: 0 1

Step 4: Process 2
- Dequeue 2 → print 2
- Adjacent to 2 → {3} but already visited → skip

Queue: [3]
Output: 0 1 2

Step 5: Process 3
- Dequeue 3 → print 3
- Adjacent to 3 → {4}
- Enqueue 4, mark visited

Queue: [4]
Output: 0 1 2 3

Step 6: Process 4
- Dequeue 4 → print 4
- Adjacent to 4 → {} (none)

Queue: [] (empty)
Output: 0 1 2 3 4

Final BFS Traversal Order:

0 1 2 3 4

Queue Evolution at Each Step

Start: [0]
After 0: [1, 2]
After 1: [2, 3]
After 2: [3]
After 3: [4]
After 4: []