មហាវិទ្យាល័យវិស្វកម្ម
FACULTY OF ENGINEERING

Data Structure & Algorithm II

Lecture 7
Graph – Matrix – Weighted - BFS

Chhoeum Vantha, Ph.D.
Telecom & Electronic Engineering

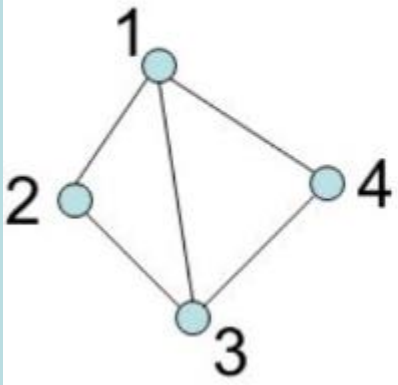
Content

- Graph Display in Matrix
- Removing Edges from Graph
- Weighted Graph
- Graph searching
  - Breadth-First-Search (BFS)
  - Depth-First-Search (DFS)
- Single-source shortest-path problem

Graph Display in Matrix

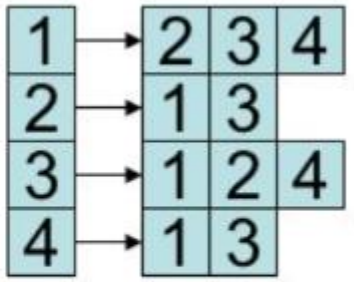
Weighted Graph Representation in Data Structure:

- Adjacency List representation
- Adjacency matrix representation



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

Adjacency matrix



Adjacency list

Graph Display in Matrix

```
1   #include<iostream>
2   using namespace std;
3   //the adjacency matrix initially 0
4   // with maximum size of 10
5   int vertArr[10][10];
6   int count = 0;
7   void displayMatrix(int v)
8   {
9       int i, j;
10      for(i = 0; i < v; i++) {
11          for(j = 0; j < v; j++) {
12              cout << vertArr[i][j] << "\t";
13          }
14          cout << endl;
15      }
16  }
```

Graph Display in Matrix

```
17   //function to add edge into the matrix
18   void add_edge(int u, int v)
19   {
20       vertArr[u][v] = 1;
21       vertArr[v][u] = 1;
22   }
```

- => check how it work?

Graph – Remove edge

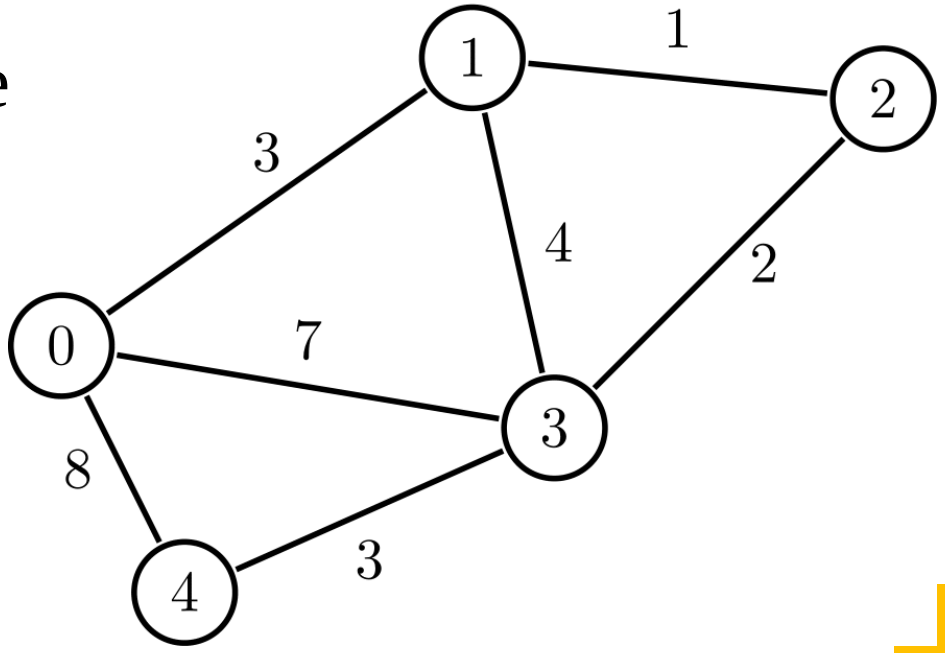
```cpp
// Adjacency Matrix representation in C++
#include <iostream>
using namespace std;
class Graph
{
  private:
  bool** adjMatrix;
  int numVertices;
 public:
  // Initialize the matrix to zero
  Graph(int numVertices)
  {
    this->numVertices = numVertices;
    adjMatrix = new bool*[numVertices];
    for (int i = 0; i < numVertices; i++)
    {
      adjMatrix[i] = new bool[numVertices];
      for (int j = 0; j < numVertices; j++)
      {
        adjMatrix[i][j] = false;
      }
    }
  }
```

Graph – Remove edge

```
24    // Add edges
25    void addEdge(int i, int j)
26    {
27        adjMatrix[i][j] = true;
28        adjMatrix[j][i] = true;
29    }
30    // Remove edges
31    void removeEdge(int i, int j)
32    {
33        adjMatrix[i][j] = false;
34        adjMatrix[j][i] = false;
35    }
```

Weighted Graph

- A special type of graph in which the edges are assigned some weights which represent
  - Cost
  - Distance
  - Many other relative measuring units



Weighted Graph

```cpp
// C++ program to represent undirected and weighted graph
// The program basically prints adjacency list
// representation of graph
//#include <bits/stdc++.h>
#include<iostream>
#include<vector>
using namespace std;

// To add an edge
void addEdge(vector <pair<int, int> > adj[], int u, int v, int wt)
{
    adj[u].push_back(make_pair(v, wt));
    adj[v].push_back(make_pair(u, wt));
}
```

Weighted Graph

```
15  // Print adjacency list representation of graph
16  void printGraph_list(vector<pair<int,int> > adj[], int V)
17  {
18      int v, w;
19      for (int u = 0; u < V; u++)
20      {
21          cout << u;
22          for (auto it = adj[u].begin(); it!=adj[u].end(); it++)
23          {
24              v = it->first;
25              w = it->second;
26              cout << "\t-> \t" << v << "\twighted: "
27                  << w << "\n";
28          }
29          cout << "\n";
30      }
31  }
```

Graph searching

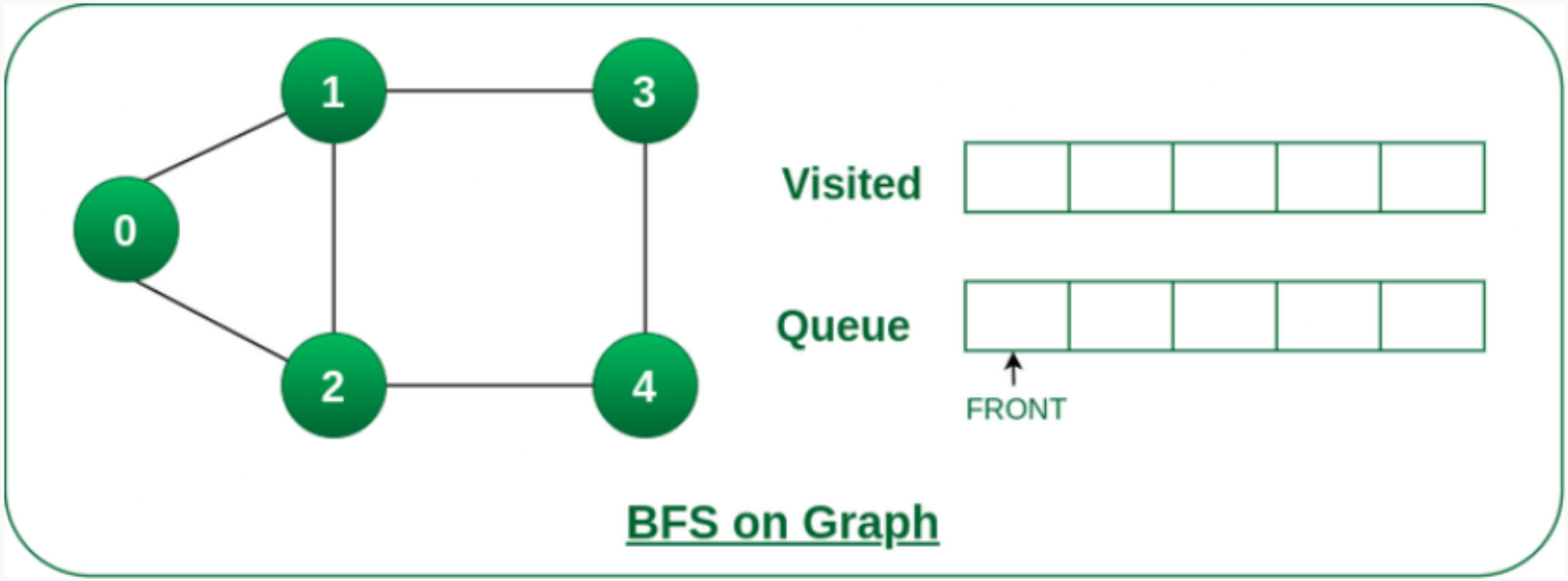
- *Problem*: find a path between two nodes of the graph (e.g., Austin and Washington)
- *Methods*:
  - Depth-First-Search (DFS)
  - Breadth-First-Search (BFS)

Breadth-First-Searching (BFS)

- To search a graph data structure for a node that meets a set of criteria.
- It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level
- To avoid processing a node more than once, we divide the vertices into two categories:
  - Visited
  - Not visited

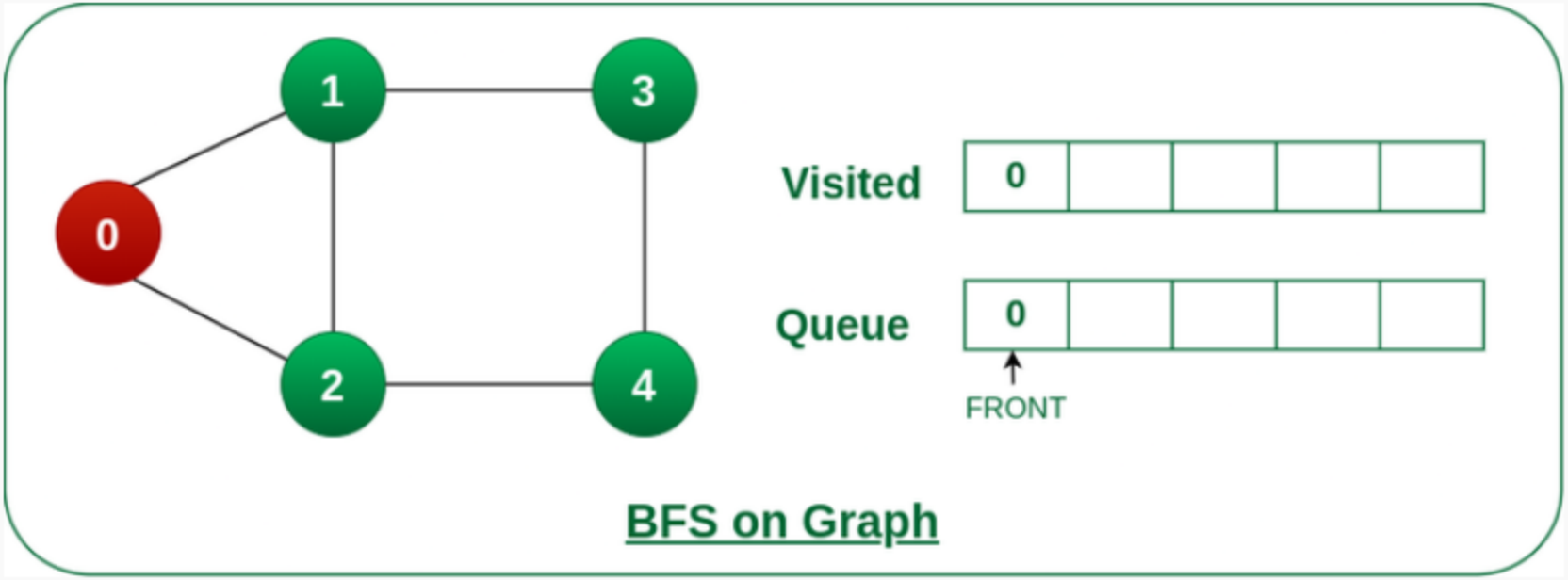
Breadth-First-Searching (BFS)

Step1: *Initially queue and visited arrays are empty.*



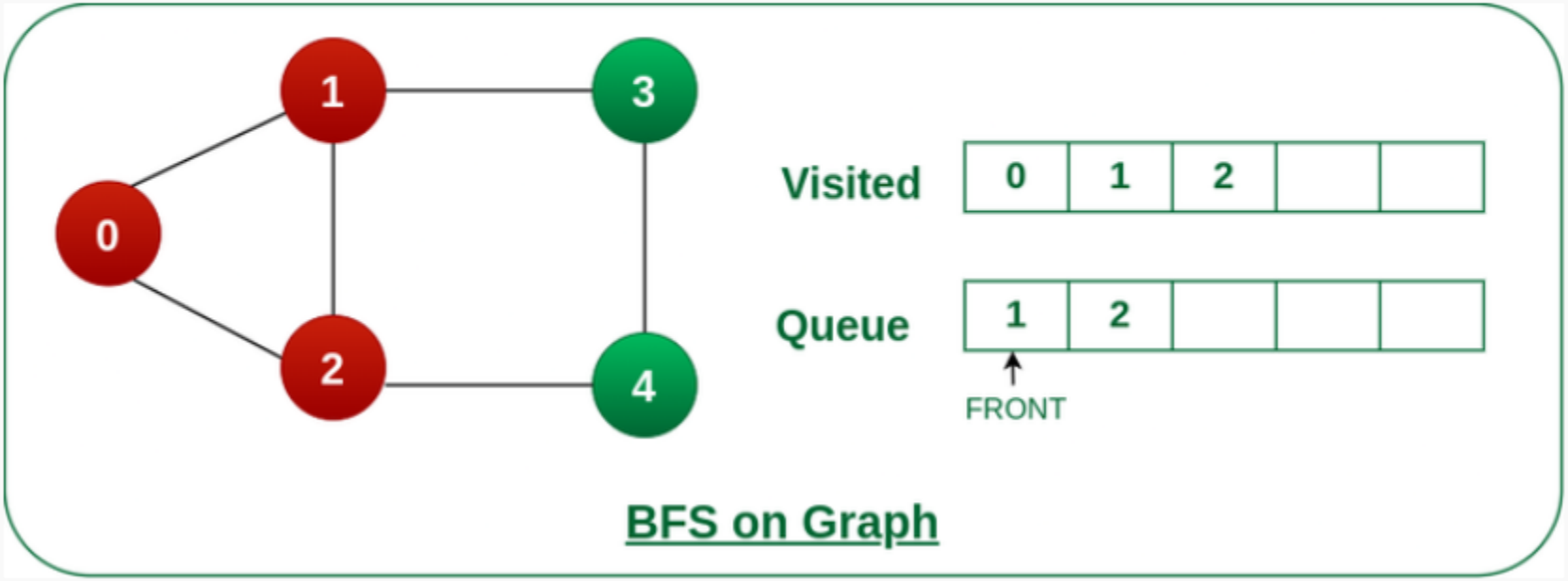
Breadth-First-Searching (BFS)

Step2: *Push node 0 into queue and mark it visited.*



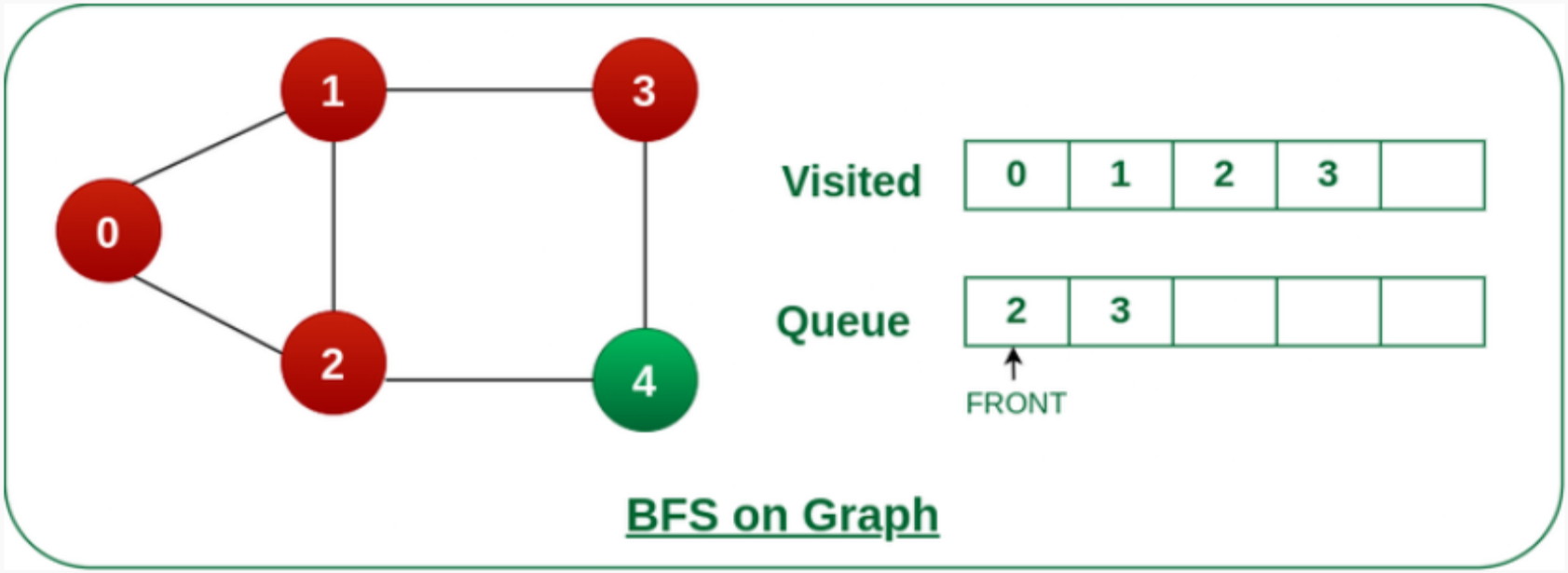
Breadth-First-Searching (BFS)

Step 3: *Remove node 0 from the front of queue and visit the unvisited neighbours and push them into queue.*



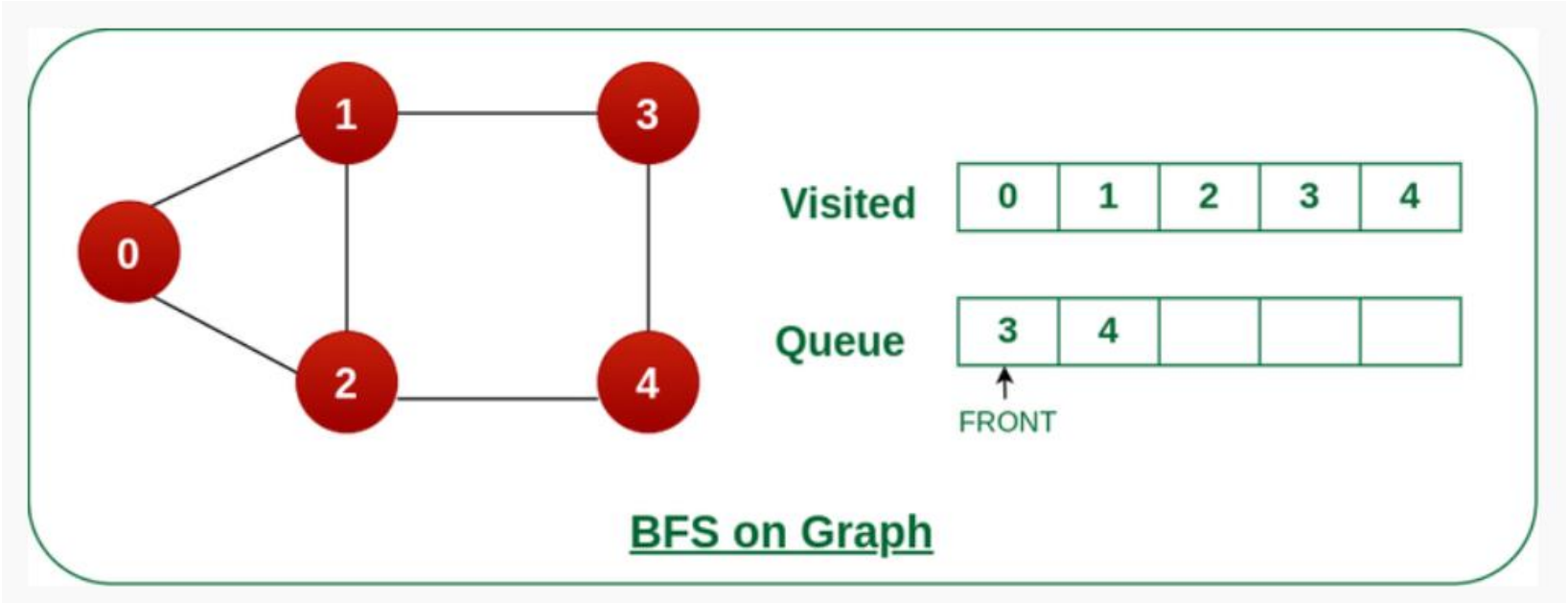
Breadth-First-Searching (BFS)

Step 4: *Remove node 1 from the front of queue and visit the unvisited neighbours and push them into queue.*



Breadth-First-Searching (BFS)

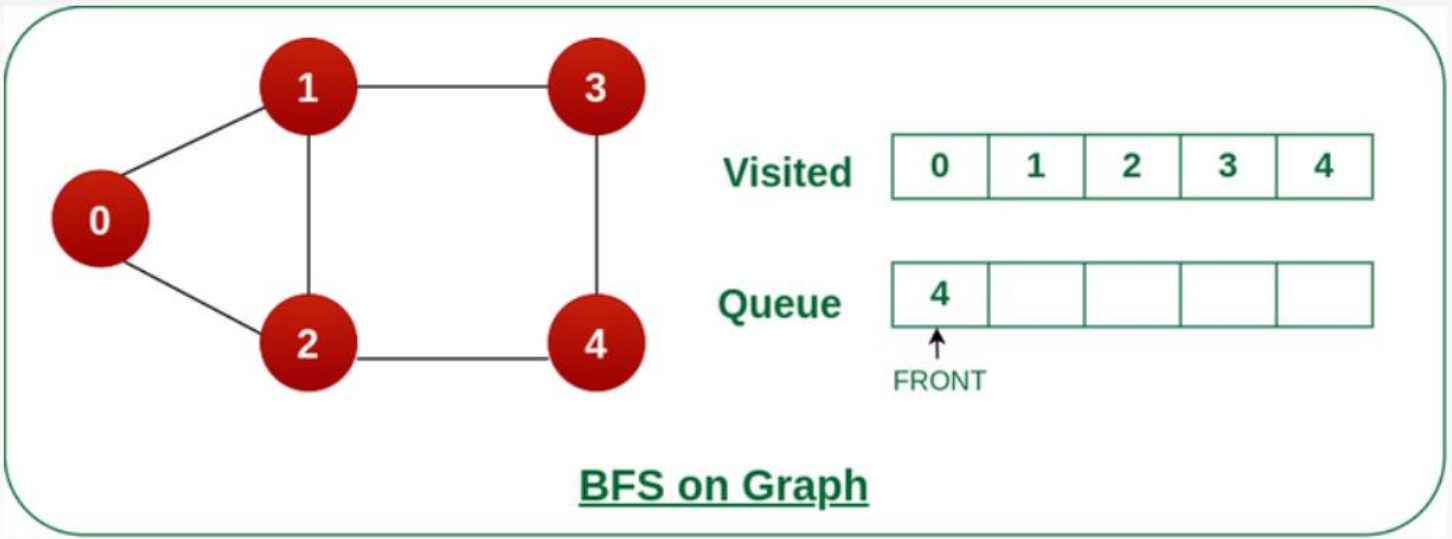
Step 5: *Remove node 2 from the front of queue and visit the unvisited neighbours and push them into queue.*



Breadth-First-Searching (BFS)

Step 6: *Remove node 3 from the front of queue and visit the unvisited neighbours and push them into queue.*

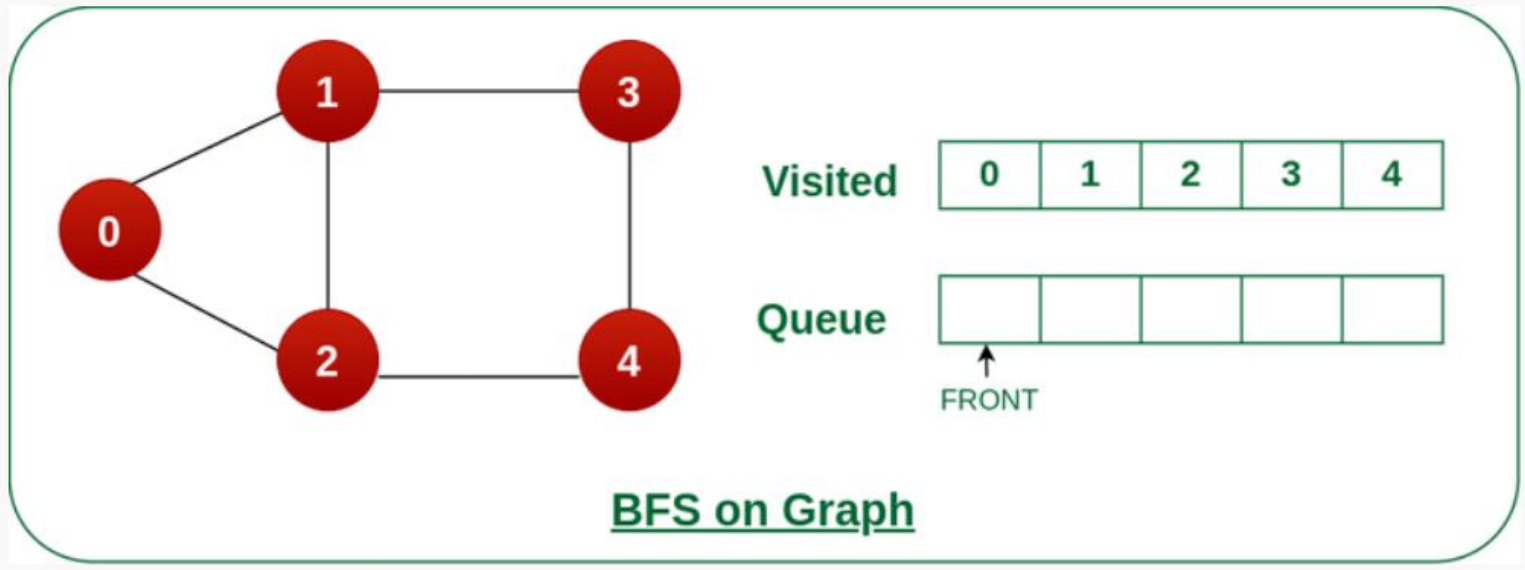
As we can see that every neighbours of node 3 is visited, so move to the next node that are in the front of the queue.



Breadth-First-Searching (BFS)

Steps 7: *Remove node 4 from the front of queue and visit the unvisited neighbours and push them into queue.*

As we can see that every neighbours of node 4 are visited, so move to the next node that is in the front of the queue.



Breadth-First-Searching (BFS)

```
1   // BFS algorithm in C++
2   #include <iostream>
3   #include <list>
4   using namespace std;
5   class Graph
6   {
7     int numVertices;
8     list<int>* adjLists;
9     bool* visited;
10  public:
11    Graph(int vertices);
12    void addEdge(int v, int w);
13    void BFS(int startVertex);
14    void printGraph(Graph const &graph, int n);
15  };
```

Breadth-First-Searching (BFS)

```
16  // Create a graph with given vertices,
17  // and maintain an adjacency list
18  Graph::Graph(int vertices)
19  {
20      numVertices = vertices;
21      adjLists = new list<int>[vertices];
22  }
23  // Add edges to the graph
24  void Graph::addEdge(int v, int w)
25  {
26      adjLists[v].push_back(w);
27      adjLists[w].push_back(v);
28  }
```

Breadth-First-Searching (BFS)

```
29  // BFS algorithm
30  void Graph::BFS(int startVertex)
31  {
32    visited = new bool[numVertices];
33    for (int i = 0; i < numVertices; i++)
34    {
35      visited[i] = false;
36    }
37    list<int> queue;
38    visited[startVertex] = true;
39    queue.push_back(startVertex);
40    list<int>::iterator i;
```

Graph – Implementation

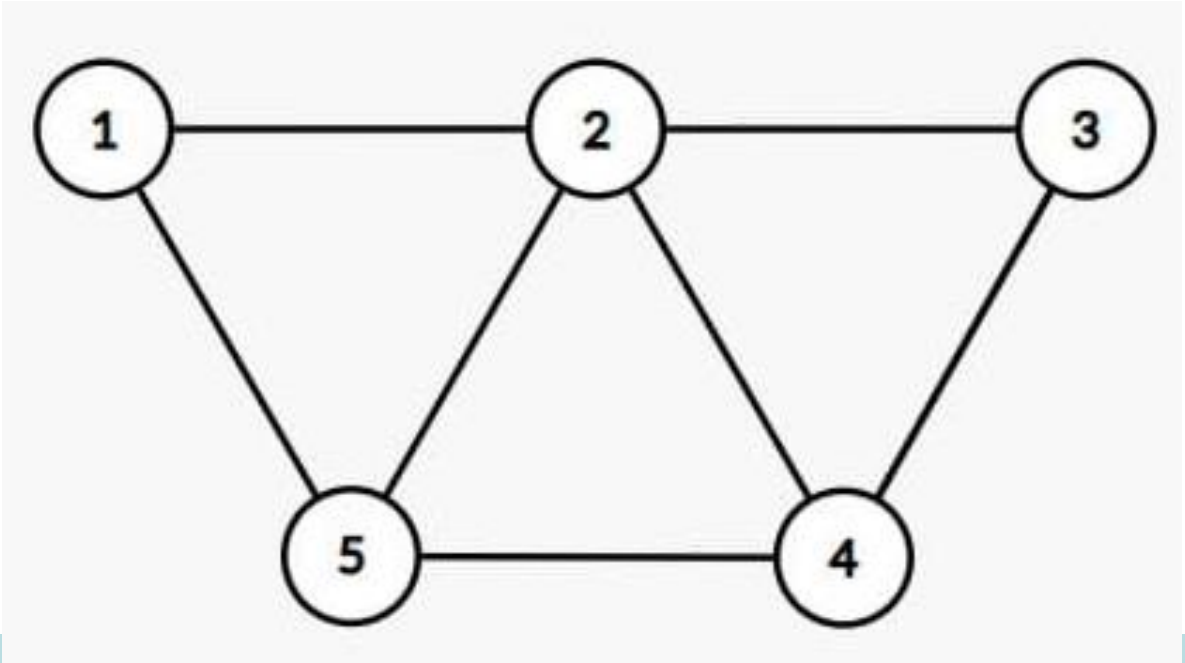
- Networks
  - The networks may include paths in a city or telephone network or circuit network.
- Social networks like linkedIn, Facebook.
  - Facebook, each person is represented with a vertex(or node)
  - Each node is a structure and contains information like person id, name, gender, locale etc.

W7 – Lab

Ex. 1 – Graph Display in Matrix

a) Draws an adjacency Matrix representation of the graph below

b) Find the complete execution of the above code with graph below:

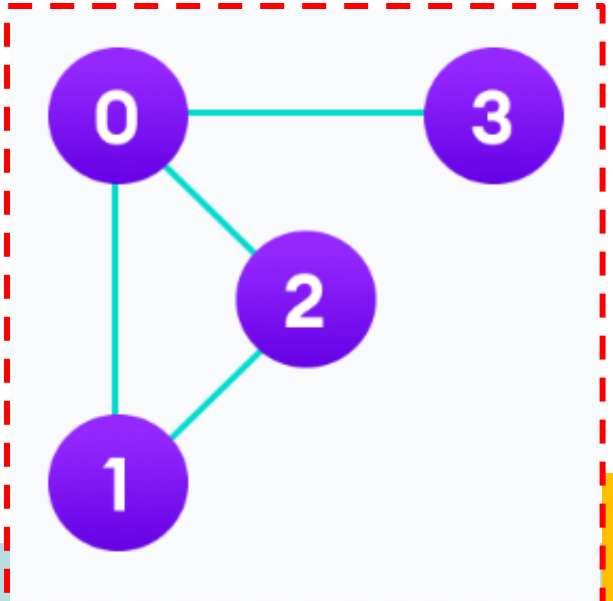


Ex. 2 – Graph Display in Matrix – Remove edge

a) Draws an adjacency Matrix representation of the original graph below

b) Draws an adjacency Matrix representation of the graph after remove edge (1, 2)

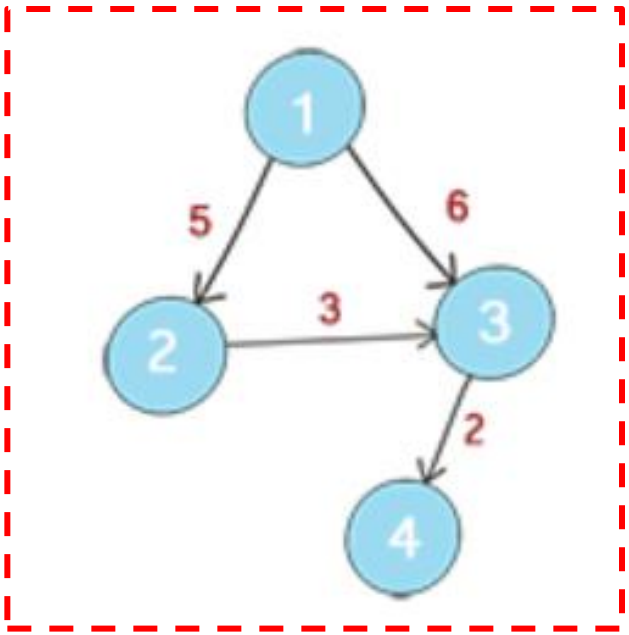
c) Find the complete execution of the above code with graph below:

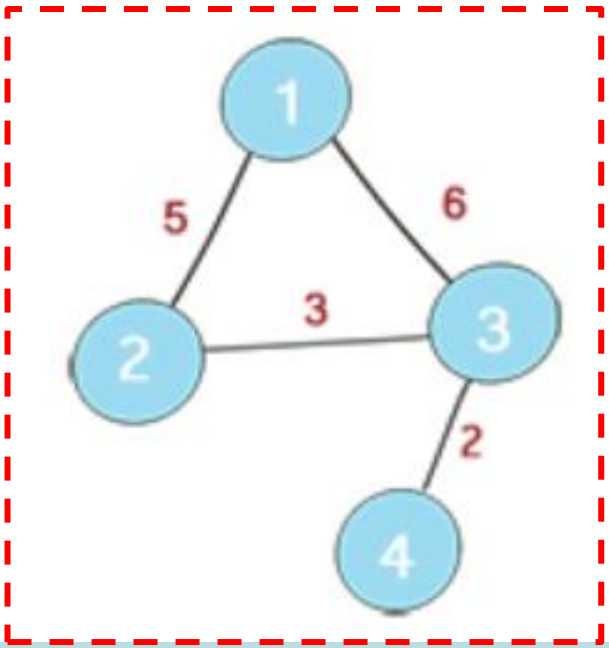


Ex. 3 – Weighted Graph

a) Draws an adjacency list representation of the graphs below

b) Find the complete execution of the above code with the graphs below:



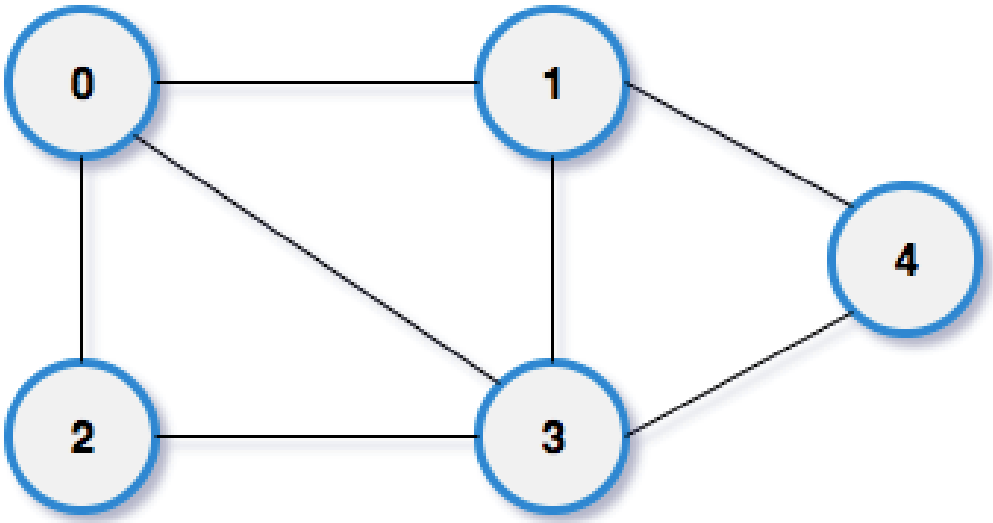


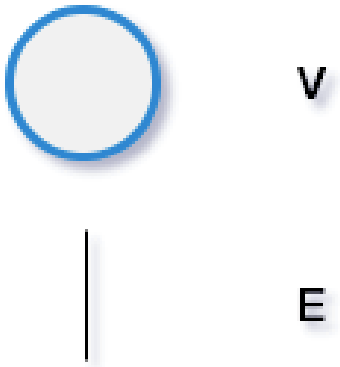
Ex. 4 – BFS

a) Draws an adjacency list representation of the graphs below

b) Estimate the output of BFS starting with 3

c) Find the complete execution of the above code with the graphs below:





Thanks!