Breadth First Search in C++

```
#include <iostream>
#include <vector>
#include <queue>
#include <deque>
using namespace std;
// Function to add an edge between two vertices u and v
void addEdge(vector<vector<int>>& adj, int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);
}
// Function to perform BFS traversal
void bfs(vector<vector<int>>& adj, int v, int s) {
    deque<int> q;
    vector<bool> visited(v, false);
    q.push_back(s);
    visited[s] = true;
    while (!q.empty()) {
        int rem = q.front();
        q.pop_front();
        cout << rem << " ";
        for (int nbr : adj[rem]) {
            if (!visited[nbr]) {
                visited[nbr] = true;
                q.push_back(nbr);
            }
        }
    }
    cout << endl; // Print newline after traversal
}
int main() {
    int V = 7;
    vector<vector<int>> adj(V);
    // Adding edges to the graph
    addEdge(adj, 0, 1);
    addEdge(adj, 0, 2);
    addEdge(adj, 2, 3);
    addEdge(adj, 1, 3);
    addEdge(adj, 1, 4);
    addEdge(adj, 3, 4);
    cout << "Following is Breadth First Traversal: \n";
    bfs(adj, V, 0);
    return 0;
}
```

Graph looks like:-

```
0 - 1
|   |
2 - 3 – 4
```

Adjacency list looks like:-

0 -> 1, 2
1 -> 0, 3, 4
2 -> 0, 3
3 -> 2, 1, 4
4 -> 1, 3
5 -> (no neighbors)
6 -> (no neighbors)

Dry Run of BFS (Start Vertex = 0):

Initialization:

- deque<int> q: Initially contains 0 (q = {0}).
- vector<bool> visited: All elements are false, except visited[0] = true.

Steps:

1. **Process Vertex 0:**
    - rem = q.front() → rem = 0.
    - Print 0.
    - Add neighbors of 0 (1 and 2) to q:
        - Mark visited[1] = true and visited[2] = true.
        - q = {1, 2}.
2. **Process Vertex 1:**
    - rem = q.front() → rem = 1.
    - Print 1.
    - Add unvisited neighbors of 1 (3 and 4) to q:
        - Mark visited[3] = true and visited[4] = true.
        - q = {2, 3, 4}.
3. **Process Vertex 2:**
    - rem = q.front() → rem = 2.
    - Print 2.
    - Add unvisited neighbors of 2 (none, as 3 is already visited).
        - q = {3, 4}.
4. **Process Vertex 3:**
    - rem = q.front() → rem = 3.
    - Print 3.
    - Add unvisited neighbors of 3 (none, as 4 is already visited).
        - q = {4}.
5. **Process Vertex 4:**
    - rem = q.front() → rem = 4.
    - Print 4.
    - Add unvisited neighbors of 4 (none).
        - q = {} (empty).

Output:-

```
0 1 2 3 4
```