|  |  |
| --- | --- |
| **Depth First Search in C++** | |
| #include <bits/stdc++.h>  using namespace std;  class Solution {    public:      // Function to return Breadth First Traversal of given graph.      vector<int> bfsOfGraph(int V, vector<int> adj[]) {          int vis[V] = {0};          vis[0] = 1;          queue<int> q;          // push the initial starting node          q.push(0);          vector<int> bfs;          // iterate till the queue is empty          while(!q.empty()) {             // get the topmost element in the queue              int node = q.front();              q.pop();              bfs.push\_back(node);              // traverse for all its neighbours              for(auto it : adj[node]) {                  // if the neighbour has previously not been visited,                  // store in Q and mark as visited                  if(!vis[it]) {                      vis[it] = 1;                      q.push(it);                  }              }          }          return bfs;      }  };  void addEdge(vector<int> adj[], int u, int v) {      adj[u].push\_back(v);      adj[v].push\_back(u);  }  void printAns(vector <int> &ans) {      for (int i = 0; i < ans.size(); i++) {          cout << ans[i] << " ";      }  }  int main()  {      vector<int> adj[6];        addEdge(adj, 0, 1);      addEdge(adj, 1, 2);      addEdge(adj, 1, 3);      addEdge(adj, 0, 4);      Solution obj;      vector <int> ans = obj.bfsOfGraph(5, adj);      printAns(ans);      return 0;  } | Graph looks like: -  0  / \  1 4  / \  2 3  Adjacency list looks like:- adj[0] = {1, 4}  adj[1] = {0, 2, 3}  adj[2] = {1}  adj[3] = {1}  adj[4] = {0}  **Step-by-Step Execution**   1. **Start BFS from Node 0**:    * Mark 0 as visited: vis[0] = 1.    * Enqueue 0: q = {0}. 2. **Process Node 0**:    * Dequeue 0: q = {}.    * Add 0 to BFS result: bfs = {0}.    * Neighbors of 0: {1, 4}.      + 1 is unvisited, mark as visited and enqueue: vis[1] = 1, q = {1}.      + 4 is unvisited, mark as visited and enqueue: vis[4] = 1, q = {1, 4}. 3. **Process Node 1**:    * Dequeue 1: q = {4}.    * Add 1 to BFS result: bfs = {0, 1}.    * Neighbors of 1: {0, 2, 3}.      + 0 is already visited, skip.      + 2 is unvisited, mark as visited and enqueue: vis[2] = 1, q = {4, 2}.      + 3 is unvisited, mark as visited and enqueue: vis[3] = 1, q = {4, 2, 3}. 4. **Process Node 4**:    * Dequeue 4: q = {2, 3}.    * Add 4 to BFS result: bfs = {0, 1, 4}.    * Neighbors of 4: {0}.      + 0 is already visited, skip. 5. **Process Node 2**:    * Dequeue 2: q = {3}.    * Add 2 to BFS result: bfs = {0, 1, 4, 2}.    * Neighbors of 2: {1}.      + 1 is already visited, skip. 6. **Process Node 3**:    * Dequeue 3: q = {}.    * Add 3 to BFS result: bfs = {0, 1, 4, 2, 3}.    * Neighbors of 3: {1}.      + 1 is already visited, skip. 7. **Queue is Empty**:    * End BFS traversal. |
| **Output:-**  **0 1 4 2 3** | |