|  |  |
| --- | --- |
| **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 Definition (Adjacency List)**  vector<int> adj[6];  addEdge(adj, 0, 1);  addEdge(adj, 1, 2);  addEdge(adj, 1, 3);  addEdge(adj, 0, 4);  Adjacency List:  0 → [1, 4]  1 → [0, 2, 3]  2 → [1]  3 → [1]  4 → [0]  **🧠 BFS Variables**   * vis[5] = {1, 0, 0, 0, 0} → Only node 0 marked visited initially * Queue: q = [0] * Result vector: bfs = []   **🔁 BFS Traversal Table**   | **Step** | **Queue** | **Node Popped** | **BFS List** | **Neighbors** | **Action** | | --- | --- | --- | --- | --- | --- | | 1 | [0] | 0 | [0] | [1, 4] | Visit 1 & 4 → mark visited, enqueue → Queue: [1, 4] | | 2 | [1, 4] | 1 | [0, 1] | [0, 2, 3] | 0 already visited; Visit 2 & 3 → mark visited, enqueue → Queue: [4, 2, 3] | | 3 | [4, 2, 3] | 4 | [0, 1, 4] | [0] | 0 already visited → nothing added | | 4 | [2, 3] | 2 | [0, 1, 4, 2] | [1] | 1 already visited | | 5 | [3] | 3 | [0, 1, 4, 2, 3] | [1] | 1 already visited | | 6 | [] | - | Done | - | Queue empty → BFS complete |   **✅ Final BFS Output**  [0, 1, 4, 2, 3]  **🧠 Summary Table**   | **Node** | **Visited** | **Enqueued** | **When** | | --- | --- | --- | --- | | 0 | ✅ | ✅ | Start | | 1 | ✅ | ✅ | From 0 | | 4 | ✅ | ✅ | From 0 | | 2 | ✅ | ✅ | From 1 | | 3 | ✅ | ✅ | From 1 |   **📌 Output on Console:**  0 1 4 2 3 |
| **Output:-**  **0 1 4 2 3** | |