Graph Traversal Techniques

Graph traversal means visiting all the vertices and edges of a graph. Two main traversal techniques are:

- **Breadth-First Search (BFS)**
- **Depth-First Search (DFS)**

These are fundamental for many graph-related problems like path finding, connectivity checks, cycle detection, and topological sorting.

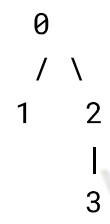
Breadth-First Search (BFS)

Definition:

BFS explores the graph level by level starting from the source node, visiting all neighbors before moving to the next level.

Traversal Order Example:

Graph:



BFS from node 0:

```
Traversal: 0 → 1 → 2 → 3
```

BFS Algorithm (Using Queue):

1. Mark the starting node as visited.
2. Enqueue the start node.
3. While the queue is not empty:
    - Dequeue a node and process it.
    - Enqueue all its unvisited neighbors.

C++ Implementation:

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

void BFS(vector<int> adj[], int start, int V) {
    vector<bool> visited(V, false);
    queue<int> q;

    visited[start] = true;
    q.push(start);

    while (!q.empty()) {
        int curr = q.front();
        q.pop();
        cout << curr << " ";

        for (int neighbor : adj[curr]) {
            if (!visited[neighbor]) {
                visited[neighbor] = true;
                q.push(neighbor);
```

```
            }
        }
    }
}

int main() {
    int V = 4;
    vector<int> adj[V];

    // Creating the graph
    adj[0].push_back(1);
    adj[0].push_back(2);
    adj[2].push_back(3);

    cout << "BFS starting from node 0:\n";
    BFS(adj, 0, V);
    return 0;
}
```

Applications of BFS:

- Shortest Path in an unweighted graph
- Web Crawlers
- Social Network Analysis
- Level-order traversal in Trees
- Connectivity in Graph

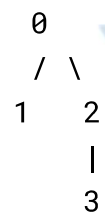
Depth-First Search (DFS)

Definition:

DFS explores as far as possible along one branch before backtracking.

Traversal Order Example:

Graph:



DFS from node 0:

```
Traversal: 0 → 1 → 2 → 3
```

DFS Algorithm (Recursive):

1. Visit the current node and mark it as visited.
2. Recursively visit all unvisited neighbors.

C++ Implementation (Recursive):

```
#include <iostream>
#include <vector>
using namespace std;

void DFS(vector<int> adj[], int curr, vector<bool>& visited) {
    visited[curr] = true;
    cout << curr << " ";

    for (int neighbor : adj[curr]) {
        if (!visited[neighbor])
            DFS(adj, neighbor, visited);
    }
}

int main() {
    int V = 4;
    vector<int> adj[V];

    // Creating the graph
    adj[0].push_back(1);
    adj[0].push_back(2);
    adj[2].push_back(3);

    vector<bool> visited(V, false);
    cout << "DFS starting from node 0:\n";
    DFS(adj, 0, visited);

    return 0;
}
```

C++ Implementation (Iterative using Stack):

```
#include <iostream>
#include <vector>
#include <stack>
using namespace std;

void DFS_iterative(vector<int> adj[], int V, int start) {
    vector<bool> visited(V, false);
    stack<int> st;
    st.push(start);

    while (!st.empty()) {
        int curr = st.top();
        st.pop();

        if (!visited[curr]) {
            visited[curr] = true;
            cout << curr << " ";

            for (auto it = adj[curr].rbegin(); it != adj[curr].rend();
++it) {
                if (!visited[*it])
                    st.push(*it);
            }
        }
    }
}
```

Applications of DFS:

- Cycle Detection in Graph
- Topological Sorting
- Connected Components
- Solving puzzles or games
- Path Finding