1. Representing a Graph:

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

void printGraph(const vector<vector<int>>& adjList) {
    for (int i = 0; i < adjList.size(); i++) {
        cout << "Node " << i << ": ";
        for (int neighbor : adjList[i]) {
            cout << neighbor << " ";
        }
        cout << endl;
    }
}

int main() {
    int nodes = 5; // Example with 5 nodes
    vector<vector<int>> adjList(nodes);

    // Adding edges
    adjList[0] = {1, 2};
    adjList[1] = {0, 3};
    adjList[2] = {0};
    adjList[3] = {1, 4};
    adjList[4] = {3};

    printGraph(adjList);
```

```
    return 0;
}
```

2. Depth-First Search (DFS):

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

void dfs(int node, vector<bool>& visited, const vector<vector<int>>& adjList)
{
    visited[node] = true;
    cout << node << " ";
    for (int neighbor : adjList[node]) {
        if (!visited[neighbor]) {
            dfs(neighbor, visited, adjList);
        }
    }
}

int main() {
    vector<vector<int>> adjList = {{1, 2}, {0, 3}, {0}, {1, 4}, {3}};
    vector<bool> visited(adjList.size(), false);

    cout << "DFS Traversal: ";
    dfs(0, visited, adjList);
```

```
    return 0;
}
```

3. Breadth-First Search (BFS):

How do you implement BFS traversal of a graph?

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

void bfs(int start, const vector<vector<int>>& adjList) {
    vector<bool> visited(adjList.size(), false);
    queue<int> q;
    q.push(start);
    visited[start] = true;

    while (!q.empty()) {
        int node = q.front();
        q.pop();
        cout << node << " ";
        for (int neighbor : adjList[node]) {
            if (!visited[neighbor]) {
                visited[neighbor] = true;
                q.push(neighbor);
            }
        }
    }
}

int main() {
    vector<vector<int>> adjList = {{1, 2}, {0, 3}, {0}, {1, 4}, {3}};

    cout << "BFS Traversal: ";
    bfs(0, adjList);
    return 0;
}
```

---

4. Detecting Cycles in an Undirected Graph:

How do you detect cycles in an undirected graph using DFS?

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

bool hasCycle(int node, int parent, vector<bool>& visited, const vector<vector<int>>& adjList) {
```

```
    visited[node] = true;
    for (int neighbor : adjList[node]) {
        if (!visited[neighbor]) {
            if (hasCycle(neighbor, node, visited, adjList)) return true;
        } else if (neighbor != parent) {
            return true;
        }
    }
    return false;
}

int main() {
    vector<vector<int>> adjList = {{1, 2}, {0, 3}, {0}, {1, 4}, {3}};
    vector<bool> visited(adjList.size(), false);

    cout << "Cycle detected: " << (hasCycle(0, -1, visited, adjList) ? "Yes" : "No") << endl;
    return 0;
}
```

---

5. Shortest Path in an Unweighted Graph:

How do you find the shortest path from a source to all other nodes in an unweighted graph?

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

void shortestPath(int start, const vector<vector<int>>& adjList) {
    vector<int> distance(adjList.size(), -1);
    queue<int> q;
    q.push(start);
    distance[start] = 0;

    while (!q.empty()) {
        int node = q.front();
        q.pop();
        for (int neighbor : adjList[node]) {
            if (distance[neighbor] == -1) {
                distance[neighbor] = distance[node] + 1;
                q.push(neighbor);
            }
        }
    }

    for (int i = 0; i < distance.size(); i++) {
        cout << "Shortest distance to node " << i << ": " << distance[i] << endl;
    }
}
```

```
int main() {
   vector<vector<int>> adjList = {{1, 2}, {0, 3}, {0}, {1, 4}, {3}};
   shortestPath(0, adjList);
   return 0;
}
```

---

6. Topological Sort (DAG):

How do you perform topological sorting on a directed acyclic graph?

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

void topologicalSort(int node, vector<bool>& visited, stack<int>& s, const vector<vector<int>>&
adjList) {
   visited[node] = true;
   for (int neighbor : adjList[node]) {
      if (!visited[neighbor]) {
         topologicalSort(neighbor, visited, s, adjList);
      }
   }
   s.push(node);
}

int main() {
   vector<vector<int>> adjList = {{1, 2}, {3}, {3}, {}};
   vector<bool> visited(adjList.size(), false);
   stack<int> s;

   for (int i = 0; i < adjList.size(); i++) {
      if (!visited[i]) {
         topologicalSort(i, visited, s, adjList);
      }
   }

   cout << "Topological Order: ";
   while (!s.empty()) {
      cout << s.top() << " ";
      s.pop();
   }
   return 0;
}
```

// Question 7: Dijkstra's Algorithm

```
#include <iostream>
#include <vector>
#include <queue>
```

```
#include <climits>
using namespace std;

typedef pair<int, int> pii; // {distance, node}

void dijkstra(int start, const vector<vector<pii>>& adjList) {
   vector<int> dist(adjList.size(), INT_MAX);
   priority_queue<pii, vector<pii>, greater<pii>> pq;

   dist[start] = 0;
   pq.push({0, start});

   while (!pq.empty()) {
      int currDist = pq.top().first;
      int node = pq.top().second;
      pq.pop();

      if (currDist > dist[node]) continue;

      for (const auto& neighbor : adjList[node]) {
         int nextNode = neighbor.first;
         int edgeWeight = neighbor.second;

         if (dist[node] + edgeWeight < dist[nextNode]) {
            dist[nextNode] = dist[node] + edgeWeight;
            pq.push({dist[nextNode], nextNode});
         }
      }
   }

   for (int i = 0; i < dist.size(); i++) {
      cout << "Shortest distance to node " << i << ": " << dist[i] << endl;
   }
}

int main() {
   vector<vector<pii>> adjList = {
      {{1, 2}, {2, 4}},
      {{2, 1}, {3, 7}},
      {{3, 3}},
      {}
   };
   dijkstra(0, adjList);
   return 0;
}

// Question 8: Prim's Algorithm

#include <iostream>
#include <vector>
#include <queue>
```

```
#include <climits>
using namespace std;

typedef pair<int, int> pii;

void prim(const vector<vector<pii>>& adjList) {
   vector<bool> inMST(adjList.size(), false);
   priority_queue<pii, vector<pii>, greater<pii>> pq;
   vector<int> parent(adjList.size(), -1);
   vector<int> key(adjList.size(), INT_MAX);

   key[0] = 0;
   pq.push({0, 0}); // {key, node}

   while (!pq.empty()) {
      int node = pq.top().second;
      pq.pop();

      inMST[node] = true;

      for (const auto& neighbor : adjList[node]) {
         int nextNode = neighbor.first;
         int weight = neighbor.second;

         if (!inMST[nextNode] && weight < key[nextNode]) {
            key[nextNode] = weight;
            pq.push({key[nextNode], nextNode});
            parent[nextNode] = node;
         }
      }
   }

   cout << "Edges in MST:" << endl;
   for (int i = 1; i < adjList.size(); i++) {
      cout << parent[i] << " - " << i << endl;
   }
}

int main() {
   vector<vector<pii>> adjList = {
      {{1, 2}, {3, 6}},
      {{0, 2}, {2, 3}, {3, 8}, {4, 5}},
      {{1, 3}, {4, 7}},
      {{0, 6}, {1, 8}},
      {{1, 5}, {2, 7}}
   };
   prim(adjList);
   return 0;
}

// Question 9: Checking Bipartiteness
```

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

bool isBipartite(const vector<vector<int>>& adjList) {
   vector<int> color(adjList.size(), -1);

   for (int i = 0; i < adjList.size(); i++) {
      if (color[i] == -1) {
         queue<int> q;
         q.push(i);
         color[i] = 0;

         while (!q.empty()) {
            int node = q.front();
            q.pop();

            for (int neighbor : adjList[node]) {
               if (color[neighbor] == -1) {
                  color[neighbor] = 1 - color[node];
                  q.push(neighbor);
               } else if (color[neighbor] == color[node]) {
                  return false;
               }
            }
         }
      }
   }
   return true;
}

int main() {
   vector<vector<int>> adjList = {
      {1, 3},
      {0, 2},
      {1, 3},
      {0, 2}
   };

   cout << "Graph is " << (isBipartite(adjList) ? "Bipartite" : "Not Bipartite") << endl;
   return 0;
}
```