1. In a graph with 7 vertices and 9 edges, what is the maximum number of edges a vertex can have if it's connected to all other vertices?

**a) 6**

b) 7

c) 8

d) 9

2. #include <iostream>

#include <vector>

#include <queue>

using namespace std;

void BFS(vector<vector<int>>& graph, int start) {

queue<int> q;

q.push(start);

while (!q.empty()) {

int node = q.front();

q.pop();

cout << node << " ";

for (int neighbor : graph[node]) {

q.push(neighbor);

}

}

}

int main() {

int n = 5;

vector<vector<int>> graph(n + 1);

graph[1] = {2, 3};

graph[2] = {4, 5};

graph[3] = {};

graph[4] = {};

graph[5] = {};

BFS(graph, 1);

return 0;

}

What will be the output of the program?

**a) 1 2 3 4 5**

b) 1 2 4 5 3

c) 1 2 3

d) 1 2

3. Which traversal algorithm explores the graph layer by layer?

**a) BFS**

b) DFS

c) Both BFS and DFS

d) Neither BFS nor DFS

4. #include <iostream>

#include <vector>

#include <stack>

using namespace std;

void DFS(vector<vector<int>>& graph, int start) {

stack<int> s;

s.push(start);

while (!s.empty()) {

int node = s.top();

s.pop();

cout << node << " ";

for (int neighbor : graph[node]) {

s.push(neighbor);

}

}

}

int main() {

int n = 5;

vector<vector<int>> graph(n + 1);

graph[1] = {2, 3};

graph[2] = {4, 5};

graph[3] = {};

graph[4] = {};

graph[5] = {};

DFS(graph, 1);

return 0;

}

What will be the output of the program?

a) 1 2 3 4 5

**b) 1 3 2 5 4**

c) 1 2 3

d) 1 2

5. What data structure is typically used to implement BFS?

a) Stack

**b) Queue**

c) Priority Queue

d) Hash Table

6. #include <iostream>

#include <vector>

#include <queue>

using namespace std;

void BFS(vector<vector<int>>& graph, int start) {

queue<int> q;

q.push(start);

while (!q.empty()) {

int node = q.front();

q.pop();

cout << node << " ";

for (int neighbor : graph[node]) {

q.push(neighbor);

}

}

}

int main() {

int n = 5;

vector<vector<int>> graph(n + 1);

graph[1] = {3};

graph[2] = {5};

graph[3] = {4,2};

graph[4] = {};

graph[5] = {};

BFS(graph, 1);

return 0;

}

What will be the output of the program?

**a) 1 3 4 2 5**

b) 1 2 4 3 5

c) 1 2 3

d) 1 2

7. DFS is implemented using which data structure?

**a) Stack**

b) Queue

c) Priority Queue

d) Linked List

8. #include <iostream>

#include <vector>

#include <stack>

using namespace std;

void DFS(vector<vector<int>>& graph, int start) {

stack<int> s;

s.push(start);

while (!s.empty()) {

int node = s.top();

s.pop();

cout << node << " ";

for (int neighbor : graph[node]) {

s.push(neighbor);

}

}

}

int main() {

int n = 4;

vector<vector<int>> graph(n + 1);

graph[1] = {2, 3};

graph[2] = {4};

graph[3] = {};

graph[4] = {3};

DFS(graph, 1);

return 0;

}

What will be the output of the program?

a) 1 2 3 4

**b) 1 3 2 4**

c) 1 2 3

d) 1 2

9. In DFS, which vertex is explored first in each iteration?

a) The vertex with the lowest index.

b) The vertex with the highest index.

**c) Any unvisited vertex adjacent to the current one.**

d) The vertex with the maximum degree.

10.

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

void BFS(vector<vector<int>>& graph, int start) {

queue<int> q;

q.push(start);

while (!q.empty()) {

int node = q.front();

q.pop();

cout << node << " ";

for (int neighbor : graph[node]) {

q.push(neighbor);

}

}

}

int main() {

int n = 3;

vector<vector<int>> graph(n + 1);

graph[1] = {2, 3};

graph[2] = {3};

graph[3] = {};

BFS(graph, 1);

return 0;

}

What will be the output of the program?

**a) 1 2 3**

b) 1 3 2

c) 1 2

d) 1 3

11. Which of the following statements is true about BFS and DFS in terms of memory usage?

**a) BFS uses more memory than DFS.**

b) DFS uses more memory than BFS.

c) Both BFS and DFS use the same amount of memory.

d) Memory usage depends on the size of the graph.

12. Which of the following represent the correct pseudo code for non recursive DFS algorithm?

**a)**

**procedure DFS-non\_recursive(G,v):**

**//let St be a stack**

**St.push(v)**

**while St is not empty**

**v = St.pop()**

**if v is not discovered:**

**label v as discovered**

**for all adjacent vertices of v do**

**St.push(a) //a being the adjacent vertex**

b)

procedure DFS-non\_recursive(G,v):

//let St be a stack

St.pop()

while St is not empty

v = St.push(v)

if v is not discovered:

label v as discovered

for all adjacent vertices of v do

St.push(a) //a being the adjacent vertex

c)

procedure DFS-non\_recursive(G,v):

//let St be a stack

St.push(v)

while St is not empty

v = St.pop()

if v is not discovered:

label v as discovered

for all adjacent vertices of v do

St.push(v)

d)

procedure DFS-non\_recursive(G,v):

//let St be a stack

St.pop(v)

while St is not empty

v = St.pop()

if v is not discovered:

label v as discovered

for all adjacent vertices of v do

St.push(a) //a being the adjacent vertex

13. DFS can be used to find which of the following in a graph?

A) Shortest path between two nodes

B) Minimum spanning tree

**C) Topological ordering**

D) All of the above

14. Which of the following functions correctly represent iterative DFS?

a)

void DFS(int s)

{

vector<bool> discovered(V, true);

stack<int> st;

st.push(s);

while (!st.empty())

{

s = st.top();

st.pop();

if (!discovered[s])

{

cout << s << " ";

discovered[s] = true;

}

for (auto i = adjacent[s].begin(); i != adjacent[s].end(); ++i)

if (!discovered[\*i])

st.push(\*i);

}

}

**b)**

**void DFS(int s)**

**{**

**vector<bool> discovered(V, false);**

**stack<int> st;**

**st.push(s);**

**while (!st.empty())**

**{**

**s = st.top();**

**st.pop();**

**if (!discovered[s])**

**{**

**cout << s << " ";**

**discovered[s] = true;**

**}**

**for (auto i = adjacent[s].begin(); i != adjacent[s].end(); ++i)**

**if (!discovered[\*i])**

**st.push(\*i);**

**}**

**}**

c)

void DFS(int s)

{

vector<bool> discovered(V, false);

stack<int> st;

st.push(s);

while (!st.empty())

{

st.pop();

s = st.top();

if (!discovered[s])

{

cout << s << " ";

discovered[s] = true;

}

for (auto i = adjacent[s].begin(); i != adjacent[s].end(); ++i)

if (!discovered[\*i])

st.push(\*i);

}

}

d)

void DFS(int s)

{

vector<bool> discovered(V, false);

stack<int> st;

st.push(s);

while (!st.empty())

{

s = st.top();

st.pop();

if (!discovered[s])

{

cout << s << " ";

discovered[s] = false;

}

for (auto i = adjacent[s].begin(); i != adjacent[s].end(); ++i)

if (discovered[\*i])

st.push(\*i);

}

}

15. Which traversal algorithm is commonly used to check for bipartite graphs?

**A) BFS**

B) DFS

C) Both BFS and DFS

D) Neither BFS nor DFS

16. Regarding implementation of Breadth First Search using queues, what is the maximum distance between two nodes present in the queue? (considering each edge length 1)

a) Can be anything

b) 0

**c) At most 1**

d) Insufficient Information

17. In which traversal algorithm is it easier to find the shortest path between two nodes?

**A) BFS**

B) DFS

C) Both BFS and DFS

D) Neither BFS nor DFS

18. Which traversal algorithm is typically used to detect cycles in a graph

A) BFS

**B) DFS**

C) Both BFS and DFS

D) Neither BFS nor DFS

19. What is the major missing in DFS snippet given below?

void Graph::DFS(int v)

{

cout << v << " ";

list<int>::iterator i;

for (i = adj[v].begin(); i != adj[v].end(); ++i)

if (!visited[\*i])

DFS(\*i);

}

a) Nothing

b) Recursive call

**c) Mark the visited vertex as true**

d) Check for unvisited vertex

20. What is the major missing in BFS snippet given below?

while (!queue.empty()) {

s = queue.front();

cout << s << " ";

for (auto adjacent : adj[s]) {

if (!visited[adjacent]) {

visited[adjacent] = true;

queue.push\_back(adjacent);

}

}

}

a) Adjacent vertex is not visited

**b) front from queue is not popped**

c) Nothing

d) Adjacent vertex is not pushed to queue