1. Which algorithm is used to find the shortest paths from a single source vertex to all other vertices in a weighted graph?

**a) Dijkstra's algorithm**

b) Bellman-Ford algorithm

c) Kruskal's algorithm

d) Prim's algorithm

2. #include <iostream>

#include <vector>

#include <queue>

using namespace std;

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

int n = graph.size();

vector<int> dist(n, INT\_MAX);

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;

dist[start] = 0;

pq.push({0, start});

while (!pq.empty()) {

int u = pq.top().second;

pq.pop();

for (auto edge : graph[u]) {

int v = edge.first;

int weight = edge.second;

if (dist[u] + weight < dist[v]) {

dist[v] = dist[u] + weight;

pq.push({dist[v], v});

}

}

}

for (int i = 1; i < n; ++i) {

cout << dist[i] << " ";

}

}

int main() {

int n = 5;

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

graph[1].push\_back({2, 10});

graph[2].push\_back({3, 5});

graph[1].push\_back({3, 3});

graph[3].push\_back({4, 2});

graph[2].push\_back({4, 7});

graph[3].push\_back({5, 15});

graph[4].push\_back({5, 10});

Dijkstra(graph, 1);

return 0;

}

What will be the output of the program?

**a) 0 10 3 5 15**

b) 0 10 5 7 15

c) 0 5 3 2 10

d) 0 10 3 2 10

3. Which graph algorithm is used to find the shortest paths between all pairs of vertices in a graph?

a) Dijkstra's algorithm

b) Bellman-Ford algorithm

**c) Floyd-Warshall algorithm**

d) Kruskal's algorithm

4. #include <iostream>

#include <vector>

#include <climits>

using namespace std;

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

int n = graph.size();

vector<int> dist(n, INT\_MAX);

dist[start] = 0;

for (int i = 0; i < n - 1; ++i) {

for (int u = 0; u < n; ++u) {

for (auto edge : graph[u]) {

int v = edge.first;

int weight = edge.second;

if (dist[u] != INT\_MAX && dist[u] + weight < dist[v]) {

dist[v] = dist[u] + weight;

}

}

}

}

for (int i = 1; i < n; ++i) {

cout << dist[i] << " ";

}

}

int main() {

int n = 5;

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

graph[1].push\_back({2, 6});

graph[1].push\_back({3, 5});

graph[2].push\_back({4, -2});

graph[3].push\_back({2, -2});

graph[2].push\_back({3, 7});

graph[4].push\_back({5, 2});

graph[3].push\_back({5, 3});

BellmanFord(graph, 1);

return 0;

}

What will be the output of the program?

a) 0 1 5 -2 0

b) 0 6 5 -2 0

**c) 0 3 5 1 3**

d) 0 6 7 2 5

5. In Dijkstra's algorithm, what is the primary difference between using a priority queue and a simple queue?

a) A priority queue guarantees the shortest paths.

**b) A simple queue ensures that vertices are explored in increasing order of their distances.**

c) A priority queue explores vertices in the order they were inserted.

d) A simple queue is more efficient for large graphs.

6. #include <iostream>

#include <vector>

using namespace std;

void FloydWarshall(vector<vector<int>>& graph) {

int n = graph.size();

for (int k = 0; k < n; ++k) {

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

if (graph[i][k] != INT\_MAX && graph[k][j] != INT\_MAX && graph[i][k] + graph[k][j] < graph[i][j]) {

graph[i][j] = graph[i][k] + graph[k][j];

}

}

}

}

}

int main() {

int n = 4;

vector<vector<int>> graph(n, vector<int>(n, INT\_MAX));

for (int i = 0; i < n; ++i) {

graph[i][i] = 0;

}

graph[0][1] = 3;

graph[1][2] = -2;

graph[2][0] = 7;

graph[2][3] = 1;

graph[3][0] = 2;

FloydWarshall(graph);

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

cout << graph[i][j] << " ";

}

cout << endl;

}

return 0;

}

What will be the output of the program?

**a) 0 3 1 2**

**1 0 -2 -1**

**3 6 0 1**

**2 5 3 0**

b) 0 3 -2 1

5 0 -4 1

7 10 0 1

2 5 3 0

c) 0 3 1 1

7 0 -2 1

5 8 0 1

2 5 3 0

d) 0 3 -2 1

7 0 1 1

5 8 0 1

2 5 3 0

7.Which algorithm can handle negative weight edges in a graph?

a) Dijkstra's algorithm

**b) Bellman-Ford algorithm**

c) Kruskal's algorithm

d) Prim's algorithm

8. #include <iostream>

#include <vector>

using namespace std;

void FloydWarshall(vector<vector<int>>& graph) {

int n = graph.size();

for (int k = 0; k < n; ++k) {

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

if (graph[i][k] != INT\_MAX && graph[k][j] != INT\_MAX && graph[i][k] + graph[k][j] < graph[i][j]) {

graph[i][j] = graph[i][k] + graph[k][j];

}

}

}

}

}

int main() {

int n = 4;

vector<vector<int>> graph(n, vector<int>(n, INT\_MAX));

for (int i = 0; i < n; ++i) {

graph[i][i] = 0;

}

graph[0][1] = 3;

graph[1][2] = -2;

graph[2][0] = 7;

graph[2][3] = 1;

graph[3][0] = 2;

FloydWarshall(graph);

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

cout << graph[i][j] << " ";

}

cout << endl;

}

return 0;

}

What will be the output of the program?

a) 0 3 -2 1

5 0 -4 1

7 10 0 1

2 5 3 0

b) 0 3 1 1

7 0 -2 1

5 8 0 1

2 5 3 0

c) 0 3 1 2

7 0 -2 1

5 8 0 1

2 5 3 0

**d) 0 3 1 2**

**1 0 -2 -1**

**3 6 0 1**

**2 5 3 0**

9. What does the Bellman-Ford algorithm guarantee in a graph with no negative-weight cycles?

**a) Shortest paths from the source to all vertices.**

b) Shortest paths from the source to some vertices.

c) Shortest paths from the source to exactly one vertex.

d) Shortest paths from the source to no vertex.

10.

#include <iostream>

#include <vector>

#include<climits>

using namespace std;

void FloydWarshall(vector<vector<int>>& graph) {

int n = graph.size();

for (int k = 0; k < n; ++k) {

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

if (graph[i][k] != INT\_MAX && graph[k][j] != INT\_MAX && graph[i][k] + graph[k][j] < graph[i][j]) {

graph[i][j] = graph[i][k] + graph[k][j];

}

}

}

}

}

int main() {

int n = 3;

vector<vector<int>> graph(n, vector<int>(n, INT\_MAX));

for (int i = 0; i < n; ++i) {

graph[i][i] = 0;

}

graph[0][1] = 3;

graph[1][2] = -2;

graph[2][0] = 7;

FloydWarshall(graph);

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

cout << graph[i][j] << " ";

}

cout << endl;

}

return 0;

}

What will be the output of the program?

a) 0 3 1

7 0 -2

5 8 0

b) 0 3 -2

5 0 -4

7 10 0

**c) 0 3 1**

**5 0 -2**

**7 10 0**

d) 0 3 1

7 0 -2

5 10 0

11. What is the main disadvantage of the Floyd-Warshall algorithm?

a) It doesn't work with negative weight edges.

**b) It's inefficient for sparse graphs.**

c) It requires a priority queue.

d) It doesn't guarantee correctness.

12. #include <iostream>

#include <vector>

#include <climits>

using namespace std;

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

int n = graph.size();

vector<int> dist(n, INT\_MAX);

dist[start] = 0;

for (int i = 0; i < n - 1; ++i) {

for (int u = 0; u < n; ++u) {

for (auto edge : graph[u]) {

int v = edge.first;

int weight = edge.second;

if (dist[u] != INT\_MAX && dist[u] + weight < dist[v]) {

dist[v] = dist[u] + weight;

}

}

}

}

for (int i = 1; i < n; ++i) {

cout << dist[i] << " ";

}

}

int main() {

int n = 5;

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

graph[1].push\_back({2, 6});

graph[1].push\_back({3, 5});

graph[2].push\_back({4, -2});

graph[3].push\_back({2, -2});

graph[2].push\_back({3, 7});

graph[3].push\_back({5, 3});

graph[4].push\_back({5, 1});

BellmanFord(graph, 1);

return 0;

}

What will be the output of the program?

a) 0 3 5 -2 0

**b) 0 3 5 1 2**

c) 0 5 3 2 5

d) 0 6 7 2 5

13. Which data structure is commonly used to implement the priority queue in Dijkstra's algorithm?

a) Queue

b) Stack

**c) Min-heap**

d) Max-heap

14. #include <iostream>

#include <vector>

#include <climits>

using namespace std;

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

int n = graph.size();

vector<int> dist(n, INT\_MAX);

dist[start] = 0;

for (int i = 0; i < n - 1; ++i) {

for (int u = 0; u < n; ++u) {

for (auto edge : graph[u]) {

int v = edge.first;

int weight = edge.second;

if (dist[u] != INT\_MAX && dist[u] + weight < dist[v]) {

dist[v] = dist[u] + weight;

}

}

}

}

for (int i = 1; i < n; ++i) {

cout << dist[i] << " ";

}

}

int main() {

int n = 5;

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

graph[1].push\_back({2, 6});

graph[1].push\_back({3, 5});

graph[2].push\_back({4, -2});

graph[3].push\_back({2, -2});

graph[2].push\_back({3, 7});

graph[3].push\_back({5, 3});

graph[4].push\_back({5, 1});

BellmanFord(graph, 1);

return 0;

}

What will be the output of the program?

a) 0 3 5 -2 0

b) 0 6 5 -2 0

**c) 0 3 5 1 2**

d) 0 6 7 2 5

15. In Dijkstra's algorithm, what does the "relaxation" step involve?

a) Finding the minimum distance vertex.

**b) Updating the distances of adjacent vertices.**

c) Removing the smallest element from the priority queue.

d) Adding the current vertex to the shortest path tree.

16. #include <iostream>

#include <vector>

#include <queue>

using namespace std;

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

int n = graph.size();

vector<int> dist(n, INT\_MAX);

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;

dist[start] = 0;

pq.push({0, start});

while (!pq.empty()) {

int u = pq.top().second;

pq.pop();

for (auto edge : graph[u]) {

int v = edge.first;

int weight = edge.second;

if (dist[u] + weight < dist[v]) {

dist[v] = dist[u] + weight;

pq.push({dist[v], v});

}

}

}

for (int i = 1; i < n; ++i) {

cout << dist[i] << " ";

}

}

int main() {

int n = 4;

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

graph[1].push\_back({2, 1});

graph[1].push\_back({3, 4});

graph[2].push\_back({3, 2});

graph[2].push\_back({4, 3});

graph[3].push\_back({4, 1});

Dijkstra(graph, 1);

return 0;

}

What will be the output of the program?

**a) 0 1 3 4**

b) 0 1 2 3

c) 0 3 4 5

d) 0 1 4 5

17. Which of the following graph algorithms can be used to solve the single-source shortest path problem on a directed acyclic graph (DAG)?

a) Dijkstra's algorithm

b) Bellman-Ford algorithm

**c) Topological sorting**

d) Floyd-Warshall algorithm

18. #include <iostream>

#include <vector>

#include <climits>

using namespace std;

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

int n = graph.size();

vector<int> dist(n, INT\_MAX);

dist[start] = 0;

for (int i = 0; i < n - 1; ++i) {

for (int u = 0; u < n; ++u) {

for (auto edge : graph[u]) {

int v = edge.first;

int weight = edge.second;

if (dist[u] != INT\_MAX && dist[u] + weight < dist[v]) {

dist[v] = dist[u] + weight;

}

}

}

}

for (int i = 1; i < n; ++i) {

cout << dist[i] << " ";

}

}

int main() {

int n = 4;

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

graph[1].push\_back({2, 3});

graph[1].push\_back({3, 2});

graph[2].push\_back({3, -5});

graph[3].push\_back({2, 1});

graph[3].push\_back({4, 2});

graph[4].push\_back({2, 2});

BellmanFord(graph, 1);

return 0;

}

What will be the output of the program?

a) 0 3 -3 1

b) 0 3 -5 1

c) 0 3 -3 2

**d) 0 -13 -14 -12**

19. What is the time complexity of Dijkstra's algorithm when implemented using a binary heap priority queue?

a) O(V + E)

b) O(V log V)

**c) O(E log V)**

d) O(V^2)

20. What will be the output if we pass { {0 2 0},{0 0 -1},{-3 0 0 }} as input to following function?

void FloydWarshall(vector<vector<int>>& graph) {

int n = graph.size();

for (int k = 0; k < n; ++k) {

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

if (graph[i][k] != INT\_MAX && graph[k][j] != INT\_MAX && graph[i][k] + graph[k][j] < graph[i][j]) {

graph[i][j] = graph[i][k] + graph[k][j];

}

}

}

}

}

What will be the output of the program?

a) 0 2 -1

2 0 -1

-3 -1 0

**b) -2 0 -1**

**-4 -2 -3**

**-5 -3 -4**

c) 0 2 -1

2 0 -3

-3 -1 0

d) 0 2 -3

2 0 -3

-3 -1 0

21. What type of graph does Dijkstra's algorithm work on?

a) Directed Acyclic Graph (DAG)

b) Undirected Graph

c) Directed Graph with Negative Weights

**d) Weighted Graph (with non-negative weights)**

22. Dijkstra's algorithm guarantees the shortest path in a graph with:

a) Negative weights

**b) Positive weights**

c) No weights

d) Cyclic paths

23. Which data structure is commonly used to implement the priority queue in Dijkstra's algorithm?

a) Stack

b) Queue

**c) Min-Heap**

d) Max-Heap

24. Dijkstra's algorithm is based on the principle of:

**a) Greedy Strategy**

b) Divide and Conquer

c) Depth-First Search

d) Breadth-First Search

25. What is the main limitation of Dijkstra's algorithm when working with graphs containing negative weight edges?

a) It doesn't work with negative weights.

**b) It might enter an infinite loop.**

c) It will give incorrect results.

d) It becomes slower.

26. The Bellman-Ford algorithm works on graphs with:

**a) Negative weights**

b) Positive weights

c) No weights

d) Cycles

27. Which property does the Bellman-Ford algorithm use to find the shortest paths?

a) Greedy property

**b) Dynamic programming**

c) Divide and conquer

d) Backtracking

28. Bellman-Ford algorithm detects negative weight cycles by:

a) Terminating with a negative distance value

b) Terminating with a positive distance value

c) Detecting negative weight edges

**d) Performing one extra iteration**

29. In the worst case, how many iterations does the Bellman-Ford algorithm perform on a graph with V vertices and E edges?

a) V - 1

**b) E**

**c) V**

d) E - 1

30. Which algorithm can handle graphs with negative weight edges without entering an infinite loop?

a) Dijkstra's algorithm

**b) Bellman-Ford algorithm**

c) Kruskal's algorithm

d) Prim's algorithm