Solution1

a. For vertices i and j, if M2(i, j) = 1, it implies that there exists a path from vertex i to vertex j with a length of 2 or less. If M2(i, j) = 0, it means that there is no path from vertex i to vertex j with a length of 2 or less.

b. The entries of M4 signify the existence of paths from vertex i to vertex j with a length of 4 or less. If M4(i, j) = 1, there exists at least one path from vertex i to vertex j with a length of 4 or less. If M4(i, j) = 0, there is no path from vertex i to vertex j with a length of exactly 4.

The entries of M5 = M4 * M signify the existence of paths from vertex i to vertex j with a length of 5 or less. Similarly, the entries of M^p represent the existence of paths from vertex i to vertex j with a length of p or less.

c. In the case of a weighted graph, where M represents the adjacency matrix, M2(i, j) = k implies that the minimum cost to reach from vertex i to vertex j through a single intermediate vertex is k. The value of M2(i, j) represents the minimum weight of the path from vertex i to vertex j with a single intermediate vertex.

Solution 2

To find the maximum bandwidth of a path between two switching centers, a and b, in a telephone network represented by a graph G, we can use a variation of Dijkstra's algorithm. Here's the algorithm:

Initialize a variable, maxBandwidth, to negative infinity. This will store the maximum bandwidth encountered during the search.

Create a priority queue, PQ, to store vertices based on their bandwidth values.

Enqueue vertex a into PQ with a bandwidth of infinity.

While PQ is not empty, do the following steps: a. Dequeue the vertex, curr, with the highest bandwidth from PQ. b. If curr is vertex b, return maxBandwidth. c. For each neighbor, next, of curr:

Calculate the minimum bandwidth, minBandwidth, as the minimum of the current bandwidth of curr and the bandwidth of the edge between curr and next.

If minBandwidth is greater than maxBandwidth, update maxBandwidth with minBandwidth.

Enqueue next into PQ with the bandwidth minBandwidth.

If the algorithm reaches this point, it means there is no path from a to b. Return an appropriate error message.

This algorithm explores the graph using a priority queue, always selecting the vertex with the highest bandwidth. It updates the maxBandwidth whenever a higher bandwidth is encountered during the search. Once vertex b is dequeued, the algorithm returns the maximum bandwidth encountered.

The runtime complexity of this algorithm depends on the underlying data structure used for the priority queue. Using a binary heap or a Fibonacci heap, the algorithm has a runtime complexity of O((V + E) log V), where V is the number of vertices and E is the number of edges in the graph.

Solution3

To find the maximum bandwidth of a path between two switching centers, a and b, in a telephone network, a modified version of Dijkstra's algorithm can be used. Here's the algorithm:

Initialize an array, dist[], with negative infinity values for all vertices except vertex a, which is set to infinity. This array will store the maximum bandwidth found so far for each vertex.

Create a priority queue, PQ, to store vertices based on their maximum bandwidth.

Enqueue vertex a into PQ with a maximum bandwidth of infinity.

While PQ is not empty, do the following: a. Dequeue the vertex, curr, with the maximum bandwidth from PQ. b. If curr is vertex b, return the maximum bandwidth stored in dist[b]. c. For each neighbor, next, of curr:

Calculate the maximum bandwidth, bandwidth, as the minimum of the current maximum bandwidth of curr and the bandwidth of the edge between curr and next.

If bandwidth is greater than the maximum bandwidth stored in dist[next], update dist[next] with bandwidth.

Enqueue next into PQ with the updated maximum bandwidth.

If the algorithm reaches this point, it means there is no path from a to b. Return an appropriate error message.

This algorithm is a variation of Dijkstra's algorithm and performs a breadth-first search (BFS) while keeping track of the maximum bandwidth encountered so far. The priority queue ensures that vertices with higher maximum bandwidths are explored first.

The runtime complexity of the algorithm depends on the priority queue's underlying data structure. Using a binary heap or a Fibonacci heap, the algorithm has a runtime complexity of O((V + E) log V), where V is the number of vertices and E is the number of edges in the graph.