CN LAB

Experiment – 8

IMPLEMENTATION OF SHORTEST PATH ALGORITHM (DIJKSTRA'S ALGORITHM)

Submitted To

Prof. Mangal Singh

Computer Networks

SIT, Pune

Submitted By

Roshan Kumar Yadav

21070126130

AIML B3

AIM

To understand and implement Shortest Path Algorithm (Dijkstra’s Algorithm)

THEORY

In computer networks, Dijkstra's algorithm is employed to determine the most efficient path for data transmission between network nodes. This algorithm calculates the shortest path from a source node to all other nodes, considering factors like link latency or cost, facilitating optimal routing, and minimizing delays in network communication.

Algorithm Steps

1. Initialize a distance array to store the minimum distance from the source node to all other nodes, marking them as "unvisited."
2. Set the source node's distance to 0 and all other nodes' distances to infinity.
3. While there are unvisited nodes:
   a. Select the unvisited node with the smallest distance as the current node.
   b. For each neighbouring node of the current node, calculate its tentative distance by adding the current node's distance and the link weight to it.
   c. If the tentative distance is less than the previously recorded distance, update the distance and mark the current node as the previous node.
4. Once all nodes are visited, backtrack from the destination node using the previous node information to find the shortest path.
5. The result includes the shortest path and associated distances, which can be used to construct routing tables for network traffic optimization.

OSERVATION

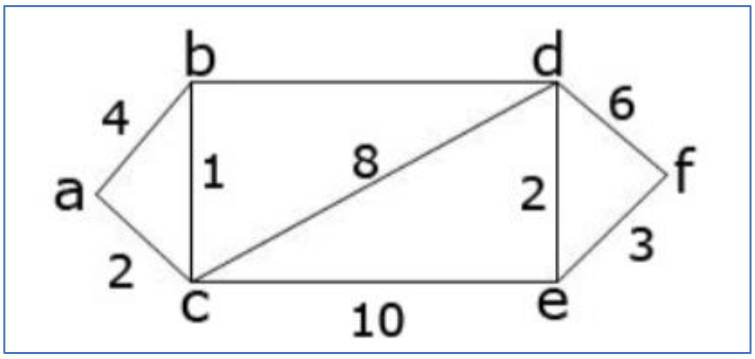
Implementation in C

```
#include <stdio.h>
#include <string.h>

#define INFINITY 9999
#define N 6
```

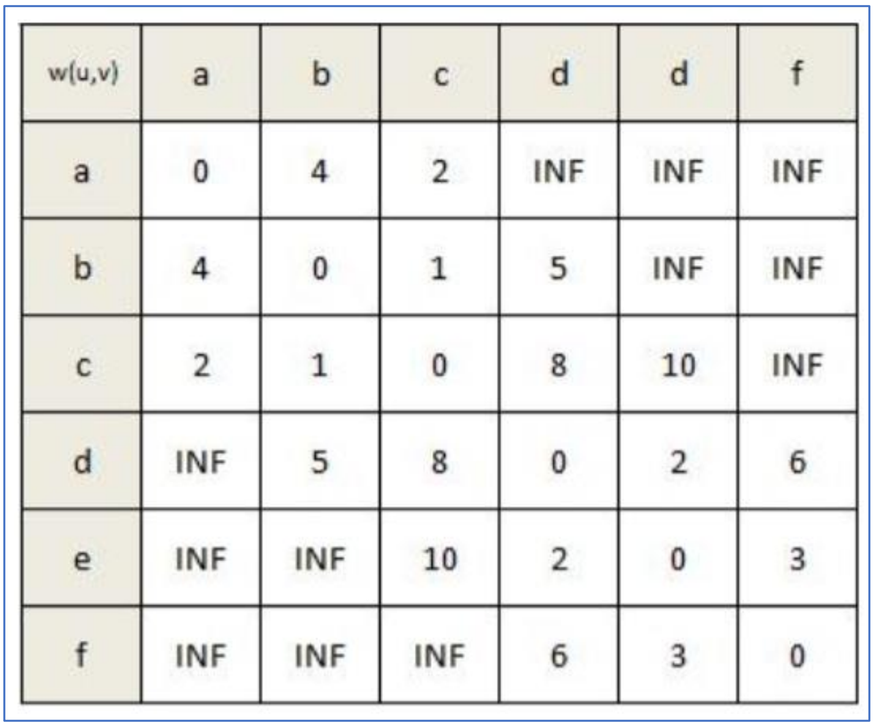
```c
// Function prototype for Dijkstra's algorithm
int dijkstra(int cost[][N], int source, int target);

int main() {
    int cost[N][N], i, j, w, co;
    int source, target, x, y;

    // Print a header for the program
    printf("\tThe Shortest Path Algorithm (DIJKSTRA'S ALGORITHM in C)\n\n");

    // Initialize the cost matrix
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            cost[i][j] = (i == j) ? 0 : INFINITY;
        }
    }

    // Get path weights from the user
    for (x = 0; x < N; x++) {
        for (y = x + 1; y < N; y++) {
            printf("Enter the weight of the path between nodes %d and %d: ", x + 1, y + 1);
            scanf("%d", &w);
            cost[x][y] = cost[y][x] = w;
        }
        printf("\n");
    }

    // Get source and target nodes from the user
    printf("Enter the source: ");
    scanf("%d", &source);
    printf("Enter the target: ");
    scanf("%d", &target);

    // Call Dijkstra's algorithm and store the result
    co = dijkstra(cost, source, target);

    // Print the shortest path distance
    printf("The Shortest Path: %d\n", co);

    return 0;
}

// Dijkstra's algorithm to find the shortest path
int dijkstra(int cost[][N], int source, int target) {
    int dist[N], prev[N], selected[N] = {0}, i, m, min, start, d, j;
    char path[N];

    // Initialize distance and previous node arrays
    for (i = 0; i < N; i++) {
        dist[i] = INFINITY;
        prev[i] = -1;
    }

    // Set the source node as the current node
    start = source - 1;
    selected[start] = 1;
    dist[start] = 0;

    // Continue until the target node is selected
    while (!selected[target - 1]) {
        min = INFINITY;
        m = 0;

        // Update distances and previous nodes for neighboring nodes
        for (i = 0; i < N; i++) {
            d = dist[start] + cost[start][i];
            if (d < dist[i] && !selected[i]) {
                dist[i] = d;
                prev[i] = start;
            }
            if (min > dist[i] && !selected[i]) {
                min = dist[i];
                m = i;
            }
        }

        // Set the next node as the current node
        start = m;
        selected[start] = 1;
    }

    // Initialize variables for constructing the path
    start = target - 1;
    j = 0;

    // Backtrack from the target to the source to construct the path
    while (start != -1) {
        path[j++] = start + 'A';  // Convert node index to character ('A', 'B', ...)
        start = prev[start];
    }

    path[j] = '\0';  // Null-terminate the path

    // Print the path in reverse order
    for (i = j - 1; i >= 0; i--) {
        printf("%c", path[i]);
    }

    // Return the shortest path distance
    return dist[target - 1];
}
```

<u>Output of the Code Implementation</u>

Graph Used for Output



Adjacency Matrix of the graph used



w(u,v)	a	b	c	d	d	f
a	0	4	2	INF	INF	INF
b	4	0	1	5	INF	INF
c	2	1	0	8	10	INF
d	INF	5	8	0	2	6
e	INF	INF	10	2	0	3
f	INF	INF	INF	6	3	0

```
The Shortest Path Algorithm (DIJKSTRA'S ALGORITHM in C)

Enter the weight of the path between nodes 1 and 2: 4
Enter the weight of the path between nodes 1 and 3: 2
Enter the weight of the path between nodes 1 and 4: 99
Enter the weight of the path between nodes 1 and 5: 99
Enter the weight of the path between nodes 1 and 6: 99
Enter the weight of the path between nodes 2 and 3: 1
Enter the weight of the path between nodes 2 and 4: 5
Enter the weight of the path between nodes 2 and 5: 99
Enter the weight of the path between nodes 2 and 6: 99
Enter the weight of the path between nodes 3 and 4: 8
Enter the weight of the path between nodes 3 and 5: 10
Enter the weight of the path between nodes 3 and 6: 99
Enter the weight of the path between nodes 4 and 5: 2
Enter the weight of the path between nodes 4 and 6: 6
Enter the weight of the path between nodes 5 and 6: 3
Enter the source: 1
Enter the target: 5
ACBDEThe Shortest Path: 10
```

SELF-ASSESSMENT

1. **Enlist types of routing algorithms in Computer network?**
   - ✓ Static Routing
   - ✓ Dynamic Routing
   - ✓ Distance-Vector Routing
   - ✓ Link-State Routing
   - ✓ Adaptive Routing
   - ✓ Shortest Path Routing
   - ✓ Flooding
   - ✓ Broadcast Routing
   - ✓ Unicast Routing
   - ✓ Multicast Routing
2. **Explain shortest path routing in Computer network?**

   Shortest Path Routing is a routing algorithm used in computer networks to find the most efficient path for data to travel from a source node to a destination node. It calculates the path with the minimum cost, often based on factors like link latency or cost, minimizing delays and optimizing network communication. This method is fundamental in ensuring that data packets take the quickest route through a network, which is essential for efficient data transmission and the overall performance of the network. One well-known algorithm for implementing shortest path routing is Dijkstra's algorithm.
3. **What is static routing and dynamic routing?**

   Static Routing:
   - ✓ Static routing is a routing method where network administrators manually configure routing tables on routers.
   - ✓ Routing paths are explicitly defined and do not adapt to changes in network topology or traffic patterns.
   - ✓ Simple and predictable, making it suitable for small, stable networks.
   - ✓ Inflexible and not well-suited for large, dynamic networks.

Dynamic Routing:

- ✓ Dynamic routing is a routing method where routers use routing protocols to exchange information and adapt to network changes.
- ✓ Routers share information about network topology and use algorithms to calculate the best routes.
- ✓ Well-suited for large and dynamic networks where routing needs to adapt to changes in link status and traffic loads.
- ✓ Common dynamic routing protocols include OSPF (Open Shortest Path First) and RIP (Routing Information Protocol).

4. **What is spanning tree algorithm?**

A spanning tree algorithm, such as the Minimum Spanning Tree (MST) algorithm, is used to find the smallest set of edges that connect all nodes in a connected, undirected graph, forming a tree with no cycles. Here are steps for finding a minimum spanning tree:

- ➢ Start with Any Node: Begin with any node in the graph.
- ➢ Select the Smallest Edge: Choose the edge with the smallest weight connected to the current set of selected nodes.
- ➢ Add to MST: Add this edge to the Minimum Spanning Tree (MST).
- ➢ Mark the Node: Mark the node at the other end of the edge as part of the MST.
- ➢ Repeat: Continue selecting the smallest edge connected to any marked node and adding it to the MST until all nodes are marked.
- ➢ Finish: Once all nodes are marked, the resulting tree is a minimum spanning tree.

Minimum Spanning Trees are vital for optimizing network design, such as in computer networks or infrastructure projects, as they help connect all points while minimizing the total cost or weight of the edges.

5. **What is need of different routing algorithms in Computer network?**

Here are points explaining the need for different routing algorithms:

- ✓ Network Size and Complexity: Different routing algorithms are required to accommodate various network sizes and complexities, from small LANs to large WANs.
- ✓ Topology Changes: Dynamic routing adapts to frequent changes in network topology, whereas static routing suits stable networks.
- ✓ Optimization Goals: Routing algorithms are chosen based on the specific optimization goals, such as minimizing latency, cost, or congestion.

- ✓ Security and QoS: Certain networks prioritize security and Quality of Service (QoS), necessitating routing algorithms that meet these requirements.
- ✓ Load Balancing: Networks with high traffic loads benefit from load-balancing routing algorithms.
- ✓ Reliability and Redundancy: Reliable networks may use specialized routing to ensure backup routes during failures.
- ✓ Compatibility: Routing algorithms are chosen based on compatibility with specific protocols and devices.