ADVANCED DATA STRUCTURES LABORATORY

PROJECT

SUPPLY CHAIN MANAGEMENT

USING

DIJKSTRA'S ALGORITHM

KARTHIKA P

2021115049

INTRODUCTION

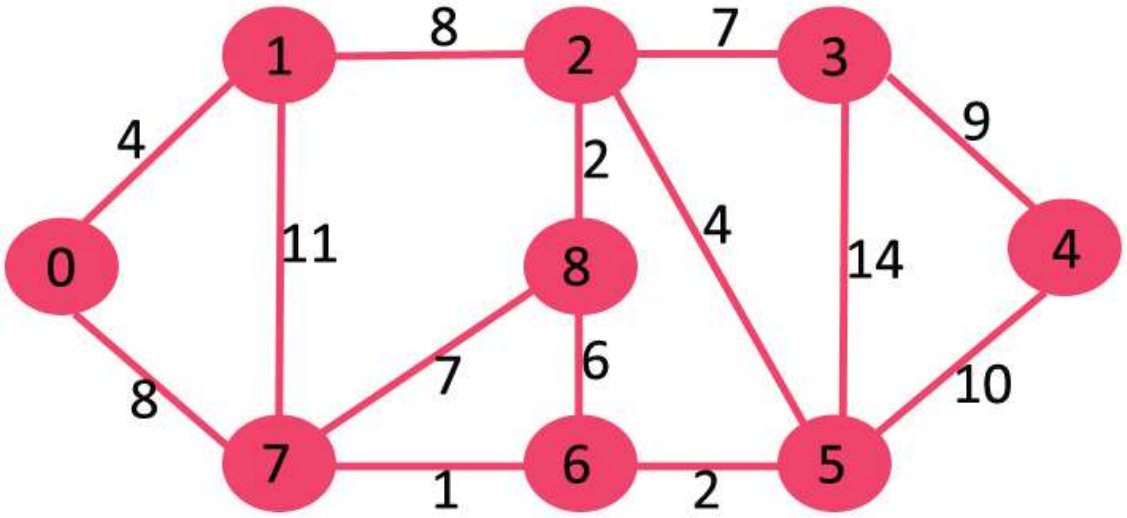
- Supply chain management involves managing the flow of goods and services from the source to the end consumer.
- To represent supply chain networks using a graph in C++, a graph is created where nodes represent various entities such as suppliers, manufacturers, distributors, and retailers, and edges represent the flow of goods between them.
- Each edge have transportation cost as an attribute.
- This code represents a simplified supply chain network and demonstrates finding the minimum cost path from a supplier to a retailer using Dijkstra's algorithm.

DIJKSTRA'S ALGORITHM

- Dijikstra's algorithm is a popular algorithms for solving many single-source shortest path problems having non-negative edge weight edge weighin the graphs
- i.e., it is to find the shortest distance between two vertices on a graph.
- It was conceived by Dutch computer scientist Egsger W.Dijkstra in 1956.

EXAMPLE

Input: src = 0



Output:

0 4 12 19 21 11 9 8 14

Explanation:

The distance from 0 to 1 = 4.
The minimum distance from 0 to 2 = 12. 0->1->2
The minimum distance from 0 to 3 = 19. 0->1->2->3
The minimum distance from 0 to 4 = 21. 0->7->6->5->4
The minimum distance from 0 to 5 = 11. 0->7->6->5
The minimum distance from 0 to 6 = 9. 0->7->6
The minimum distance from 0 to 7 = 8. 0->7
The minimum distance from 0 to 8 = 14. 0->1->2->8

NEED FOR DIJKSTRA'S ALGORITHM

- The need for Dijkstra's Algorithm arises in many application where finding the shortest path between two points is crucial.

CHARACTERISTICS

- Dijkstra's algorithm starts at the node source node we choose and then it analyzes the graph condition and its paths to find the optimal shortest distance between the given node and all other nodes in the graph.
- Dijkstra's algorithm keeps track of the currently known shortest distance from each node to the source node and updates the value after it finds the optimal path once the algorithm finds the shortest path between the source node and destination node then the specific node is marked as visited.

CODE

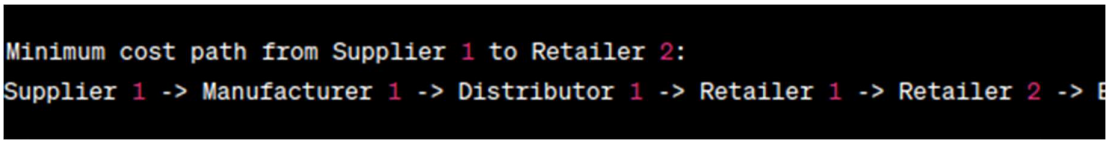
//Scm.cpp

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

using namespace std;

// Structure representing a node in the supply chain graph
struct Node {
    string name;
    vector<pair<Node*, int>> neighbors;

    Node(string name) : name(name) {}
};

// Function to find the minimum cost path using Dijkstra's algorithm
vector<Node*> findMinCostPath(Node* source, Node* destination) {
    // Priority queue to store nodes based on their cost
    priority_queue<pair<int, Node*>, vector<pair<int, Node*>>,
greater<pair<int, Node*>>> pq;

    // Map to store the minimum cost to reach each node
    unordered_map<Node*, int> costs;

    // Map to store the previous node in the minimum cost path
    unordered_map<Node*, Node*> prev;

    // Initialize costs to infinity except the source node (0 cost)
    for (auto& neighbor : source->neighbors) {
        Node* neighborNode = neighbor.first;
        int cost = neighbor.second;
        costs[neighborNode] = cost;
        prev[neighborNode] = source;
        pq.push({cost, neighborNode});
    }
```

```
    while (!pq.empty()) {
        Node* currNode = pq.top().second;
        pq.pop();

        if (currNode == destination)
            break;

        for (auto& neighbor : currNode->neighbors) {
            Node* neighborNode = neighbor.first;
            int edgeCost = neighbor.second;

            int newCost = costs[currNode] + edgeCost;

            if (newCost < costs[neighborNode]) {
                costs[neighborNode] = newCost;
                prev[neighborNode] = currNode;
                pq.push({newCost, neighborNode});
            }
        }
    }

    // Reconstruct the minimum cost path

    vector<Node*> minCostPath;
    Node* curr = destination;
    while (curr != nullptr) {
        minCostPath.push_back(curr);
        curr = prev[curr];
    }
    reverse(minCostPath.begin(), minCostPath.end());

    return minCostPath;
}

int main() {

    // Create nodes representing entities in the supply chain

    Node* supplier1 = new Node("Supplier 1");
    Node* supplier2 = new Node("Supplier 2");
    Node* manufacturer1 = new Node("Manufacturer 1");
    Node* manufacturer2 = new Node("Manufacturer 2");
```

```
    Node* distributor1 = new Node("Distributor 1");
    Node* distributor2 = new Node("Distributor 2");
    Node* retailer1 = new Node("Retailer 1");
    Node* retailer2 = new Node("Retailer 2");

    // Set up the connections and transportation costs

    supplier1->neighbors = {{manufacturer1, 4}, {manufacturer2, 5}};
    supplier2->neighbors = {{manufacturer2, 3}};
    manufacturer1->neighbors = {{distributor1, 2}, {distributor2, 3}};
    manufacturer2->neighbors = {{distributor2, 2}};
    distributor1->neighbors = {{retailer1, 2}};
    distributor2->neighbors = {{retailer1, 1}, {retailer2, 3}};
    retailer1->neighbors = {};
    retailer2->neighbors = {};

    // Find the minimum cost path from Supplier 1 to Retailer 2

    vector<Node*> minCostPath = findMinCostPath(supplier1, retailer2);

    // Print the minimum cost path

    cout << "Minimum cost path from Supplier 1 to Retailer 2:" << endl;

    for (auto node : minCostPath) {
        cout << node->name << " -> ";
    }
    cout << "End" << endl;

    return 0;
}
```

OUTPUT



SOME OTHER APPLICATIONS OF DIJKSTRA'S ALGORITHM

- Digital Mapping Services in Google Maps
- Social Networking Applications
- Telephone Network
- IP routing to find Open shortest Path First
- Flighting Agenda
- Designate file server
- Robotic Path.

****