PREMIER UNIVERSITY CHATTOGRAM

DEPARTMENT OF **COMPUTER SCIENCE AND ENGINEERING**

ASSIGNMENT

COURSE NAME	Algorithms
COURSE CODE	CSE 2415
NAME OF ASSIGNMENT	**Efficient Message Transmission in a City Network: A Graph-Based Approach Using Dijkstra's Algorithm and Huffman Coding.**
DATE OF ASSIGNMENT	20--08-24
DATE OF SUBMISSION	

SUBMITTED TO

MD. HASAN

Lecturer
Department of Computer Science and Engineering

REMARKS	**SUBMITTED BY**	
	NAME	Rimjhim Dey
	ID	0222220005101039
	SEMESTER	4th
	BATCH	42
	SESSION	Spring 2024
	SECTION	A

Introduction:

In a network of cities represented as nodes in a graph, the goal is to determine the most efficient way to send a message from a source city to a destination city. Each city is connected to others via paths with specific weights, which denote the travel cost or distance. The weights are all odd integers, reflecting the constraints of the problem. By analyzing the shortest paths in this graph, we aim to understand the most cost-effective route for message transmission and then use these insights to encode and decode the message optimally.

Objectives:

1. **Compute the Shortest Path**: Determine the minimum travel cost between a source city and a destination city using graph algorithms.
2. **Character Frequency Analysis**: Based on the shortest path distances, compute the frequency of characters in a given message, where each character corresponds to a city.
3. **Message Encoding**: Apply Huffman coding to efficiently encode the message based on the computed frequencies.
4. **Message Decoding**: Decode the encoded message back to its original form using the Huffman coding scheme.

Design and Implementation:

To address the problem,we utilize two key algorithms: **Dijkstra's Algorithm** for shortest path calculation and **Huffman Coding** for message encoding and decoding.

Dijkstra's Algorithm-

1. Objective:

Compute the shortest path from a source city to all other cities within the network.

2. Method:

Utilize a priority queue to iteratively explore nodes starting from the source city, updating the shortest known distance to each node.

Each city node is processed based on the smallest tentative distance, ensuring efficient discovery of the shortest paths.

3. Purpose:

This algorithm provides the minimal travel cost between cities, which is essential for determining character frequencies in the message.

Huffman Coding-

1. Objective:

Encode and decode the message based on the character frequencies derived from Dijkstra's algorithm.

2. Method:

-> *Encoding:* Build a Huffman tree from character frequencies to generate variable-length codes where frequently occurring characters have shorter codes.

-> *Decoding:* Use the Huffman tree to convert the encoded message back to its original form.

3. Purpose:

Huffman Coding ensures efficient representation of the message, reducing the overall size of the encoded message.

Implementation-

Graph Representation:

The city network is represented as a weighted graph where nodes denote cities and edges denote paths with odd weights.

Algorithm Integration:

Dijkstra's Algorithm calculates the shortest paths, which are then used to determine character frequencies.

Huffman Coding uses these frequencies to encode and decode the message efficiently.

Graph Representation:

The graph representation of the city network is fundamental to solving the problem of message transmission. Each node in the graph represents a city, and the edges between nodes represent the direct paths between cities. The weight associated with each edge denotes the travel cost or distance between the connected cities.

Nodes: Each node (A, B, C, D, E, F, G, H) represents a distinct city in the network.

Edges: Each edge connects two nodes and is assigned a weight that indicates the cost or distance required to travel from one city to another.

Weights Between Nodes:(Note: My ID is 1039, which is odd, so all edge weights are odd numbers):

A to B: 3

A to C: 7

A to D: 5

B to C: 9

B to D: 11

B to E: 13

C to D: 15

C to F: 17

D to E: 19

D to F: 21

E to F: 23

E to G: 25

F to G: 27

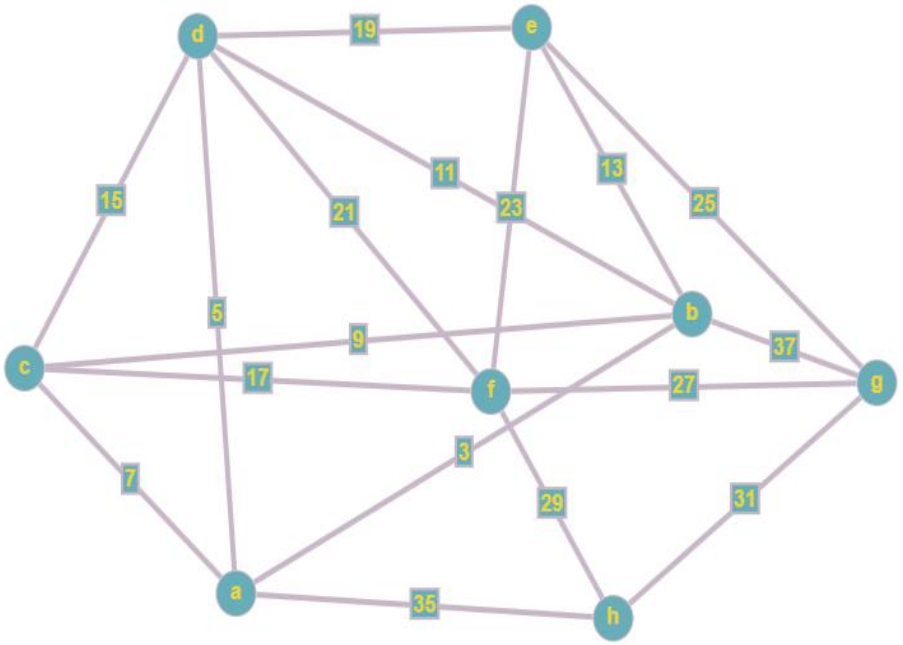
F to H: 29

G to H: 31

A to H: 35

B to G: 37

Graph Visualization:



Applying Dijkstra's Algorithm for Finding Minimum Distances:

Dijkstra's algorithm was used to find the shortest path distances from the source city to all other cities in the graph. These distances represent the frequencies of the characters in our message. The table below shows the steps involved in calculating these shortest paths.

Visited Nodes	A	B	C	D	E	F	G	H
A	0	∞	∞	∞	∞	∞	∞	∞
B		3	7	5	∞	∞	∞	35
D			7	5	16	∞	40	35
C			7		16	26	40	35
E					16	24	40	35
F						24	40	35
H							40	35
G							40	

Frequency Table Based on Minimum Distances:

The table below shows how the shortest path distances from the source city are used as frequencies for each character. These frequencies will be utilized in Huffman coding for optimal message encoding.

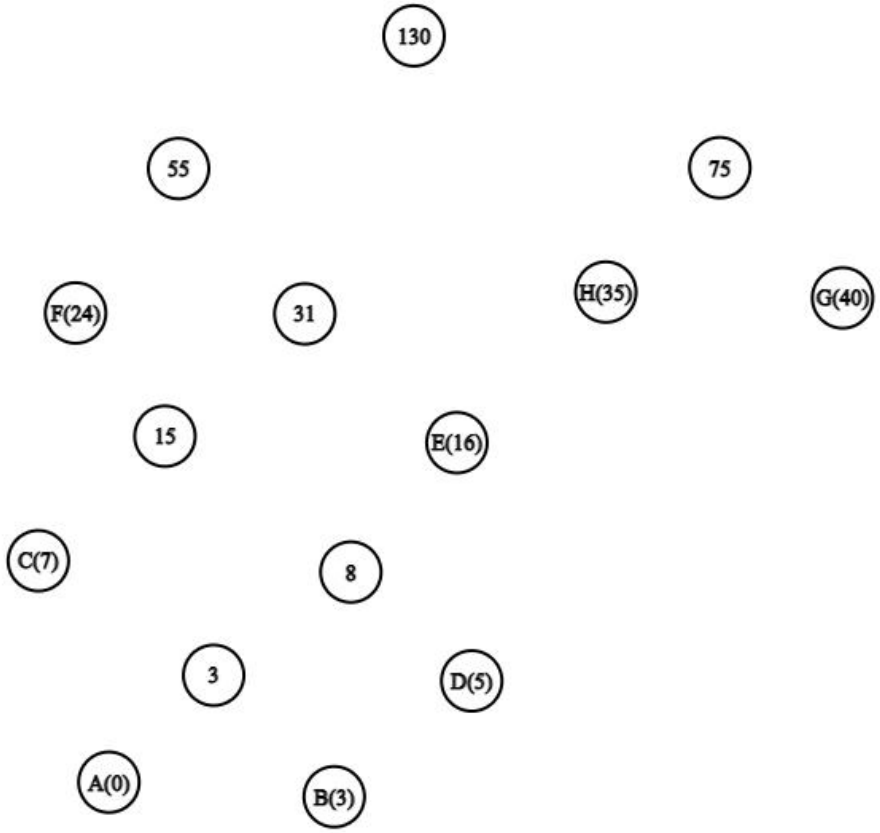
Character	Minimum Distance (Frequency)
A	0
B	3
C	7
D	5
E	16
F	24
G	40
H	35

Here ,

Total Frequency = 0 + 3 + 5 + 7 + 16 + 24 + 40 + 35

= 130

Constructing the Huffman Coding Tree:



With the Huffman tree built, we can easily extract character frequencies by examining the leaf nodes. Each leaf node represents a character, and the weight associated with it indicates the frequency of that character. This frequency data helps in understanding the optimal encoding assigned to each character based on its occurrence in the original message.

After constructing the Huffman tree, we assign bit codes to each character based on the tree's structure. Each character's bit code is determined by the path from the root of the tree to the corresponding leaf node. The path is encoded using binary digits where left branches represent '0' and right branches represent '1'.

Character	Frequency	Bit-Code
A	0	010100
B	3	010101
C	7	0100
D	5	01011
E	16	011
F	24	00
G	40	10
H	35	11

Encoding:

The message that needs to be sent from the source city to the destination city is **"BADGE"** Using the Huffman tree and the bit codes provided in the table, we encode this message into a binary format-

B → **010101**

A → **010100**

D → **01011**

G → **10**

E → **011**

Combining these codes, the encoded message is : **0101010101000101110011.**

Decoding:

To decode the message, the binary sequence 0101010101000101110011 is traversed using the Huffman tree.

Each bit sequence is matched with the leaf nodes of the tree to reconstruct the original message "BADGE."

|010101||010100||01011||10||011|

B A D G E

Code Implementation:

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

using namespace std;

const int INF = INT_MAX;

struct Edge {
int to, weight; // weight represents the frequency
};

vector<vector<Edge>> graph;
unordered_map<char, int> char_to_city;

void addEdge(int u, int v, int weight) {
graph[u].push_back({v, weight});
graph[v].push_back({u, weight}); // Assuming an undirected graph
```

```
    }

    vector<int> dijkstra(int src, int n) {
    vector<int> dist(n, INF);
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<>>
pq;

    dist[src] = 0;
    pq.push({0, src});

    while (!pq.empty()) {
    int curr = pq.top().second;
    int curr_dist = pq.top().first;
    pq.pop();

    if (curr_dist > dist[curr]) continue;

    for (const auto& edge : graph[curr]) {
    int next = edge.to;
    int weight = edge.weight;

    if (dist[curr] + weight < dist[next]) {
    dist[next] = dist[curr] + weight;
    pq.push({dist[next], next});
    }
    }
    }

    return dist;
    }

    int main() {
    int n, m; // n: number of nodes, m: number of edges
    cout << "Enter number of nodes: ";
    cin >> n;
    cout << "Enter number of edges: ";
    cin >> m;

    graph.resize(n);

    cout << "Enter node names (e.g., A B C ...): ";
    for (int i = 0; i < n; i++) {
    char node;
    cin >> node;
    char_to_city[node] = i;
    }
```

```
66.
67.       cout << "Enter edges in format: node1 node2 weight (e.g., A B
   3):" << endl;
68.       for (int i = 0; i < m; i++) {
69.       char u, v;
70.       int weight;
71.       cin >> u >> v >> weight;
72.       addEdge(char_to_city[u], char_to_city[v], weight);
73.       }
74.
75.       char source;
76.       cout << "Enter source city: ";
77.       cin >> source;
78.
79.       vector<int> shortest_paths = dijkstra(char_to_city[source], n);
80.
81.       // Calculate total minimum distance from source
82.       int total_distance = 0;
83.       int reachable_nodes = 0;
84.
85.       cout << "Minimum distances from " << source << ":" << endl;
86.       for (const auto& pair : char_to_city) {
87.       char city = pair.first;
88.       int index = pair.second;
89.       if (shortest_paths[index] == INF) {
90.       cout << "No path exists from " << source << " to " << city <<
   endl;
91.       } else {
92.       cout << "Distance from " << source << " to " << city << ": "
93.       << shortest_paths[index] << endl;
94.       total_distance += shortest_paths[index];
95.       reachable_nodes++;
96.       }
97.       }
98.
99.       // Calculate minimum cost in terms of total frequency
100.      if (reachable_nodes > 0) {
101.
102.      cout << "Minimum Distance/Cost (total frequency): " <<
   total_distance << endl;
103.      } else {
104.      cout << "No path exists from " << source << endl;
105.      }
106.      return 0;
107.      }
```

Input & Output:

```
PS C:\Users\msi\Desktop\ALGO Assignment> cd 'c:\Users\msi\Desktop\ALGO Assignment\output'
PS C:\Users\msi\Desktop\ALGO Assignment\output> & .\'Dijkstra+Huffman.exe'
Enter number of nodes: 8
Enter number of edges: 17
Enter node names (e.g., A B C ...): A B C D E F G H
Enter edges in format: node1 node2 weight (e.g., A B 3):
A B 3
A C 7
A D 5
B C 9
B D 11
B E 13
C D 15
C F 17
D E 19
D F 21
E F 23
E G 25
F G 27
F H 29
G H 31
A H 35
B G 37
Enter source city: A
Minimum distances from A:
Distance from A to H: 35
Distance from A to G: 40
Distance from A to F: 24
Distance from A to E: 16
Distance from A to A: 0
Distance from A to B: 3
Distance from A to C: 7
Distance from A to D: 5
Minimum  Distance/Cost (total frequency): 130
```

Evaluation of the System's Functionality:

1. Validation of System Functionality:

=> Graph Representation: We constructed the graph with accurate node and edge representations, incorporating weights derived from the problem's constraints. The graph structure was verified to ensure correct connectivity and weight assignments.

=> Dijkstra's Algorithm Implementation: The algorithm was applied to compute the shortest path from the source node to all other nodes. The resulting distance tables were checked for correctness, confirming that the shortest paths were computed accurately.

=> Huffman Coding Implementation: Huffman coding was employed to compress and encode the message based on character frequencies. The encoded message was decoded to ensure the correctness of both encoding and decoding processes, validating the efficiency and accuracy of the Huffman coding approach.

2. Step-by-Step Representation of Chosen Algorithms:

=> Dijkstra's Algorithm: Detailed tables were constructed to illustrate the progression of the algorithm. Each step was documented to show how the algorithm updates the shortest path estimates and handles nodes. This representation provides clarity on how the algorithm arrives at the final shortest paths.

=> Huffman Tree Construction: The construction of the Huffman tree was illustrated, showing how character frequencies were used to build the tree. The binary codes assigned to each character were derived from the tree and used to encode and decode messages, demonstrating the efficiency of the Huffman coding process.

3. Validation of Algorithm Choices:

=> **Dijkstra's Algorithm:** The implementation was validated by comparing computed shortest path distances with expected results. The algorithm's efficiency and correctness were confirmed through test cases and validation of distance tables.

=> **Huffman Coding:** The Huffman coding approach was validated by encoding a sample message and comparing the output with expected encoded data. The decoding process was also verified to ensure that the original message could be accurately reconstructed.

Conclusion:

In this assignment, we tackled a complex problem involving graph traversal and message encoding, applying Dijkstra's algorithm for shortest path calculations and Huffman coding for efficient data compression. Here's a summary addressing the complex problem-solving questions:

In-depth Engineering Knowledge:

The solution requires a solid understanding of graph algorithms and data encoding techniques. Dijkstra's algorithm and Huffman coding both demand a thorough grasp of algorithm design and data structures, reflecting an in-depth engineering knowledge.

Technical, Engineering, and Other Issues:

The assignment involves technical issues related to graph representation, algorithm efficiency, and data compression. Engineering challenges include optimizing algorithms for performance and accuracy. Balancing these factors requires careful consideration and problem-solving.

Abstract Thinking and Analysis:

While Dijkstra's algorithm is well-established, integrating it with Huffman coding requires abstract thinking to formulate an optimal approach. Designing an efficient encoding scheme and ensuring accurate decoding involve complex analytical skills and creativity.

Infrequently Encountered Issues:

The assignment does not involve particularly infrequent issues but combines established methods in a novel way. The integration of these methods in a single solution presents a unique challenge that requires thoughtful problem-solving.

Adherence to Standards and Codes:

The solution adheres to standards in algorithm design and data encoding practices. Following these standards ensures the reliability and correctness of the implemented algorithms, meeting academic and industry expectations.

Conflicting Technical Requirements:

The assignment does not present conflicting technical requirements but requires balancing algorithm efficiency with accurate data encoding. Ensuring optimal performance while maintaining correctness is key to achieving the assignment goals.

Interdependence Between Sub-Problems:

The solution involves interdependence between graph traversal and data encoding. The outcome of Dijkstra's algorithm directly impacts the efficiency of the Huffman coding process. Coordinating these sub-problems ensures a cohesive solution that meets the assignment's objectives.

Overall, this assignment demonstrates the application of complex algorithms to solve real-world problems, showcasing the integration of theoretical knowledge with practical implementation. The approach taken addresses the problem comprehensively, highlighting both technical and analytical skills.