Experiment 3: Neighbour Table Determination

Aim: To create neighbor table for a given network topology

Objective: After carrying out this experiment, students will be able to:

- Generate neighbor table for all the nodes in a given topology.
- Analyse how this is useful in the process of routing data

Problem statement: You are required to write a program that calculates neighbor table for all the nodes in a given network. Consider a network with 10 nodes that is deployed in an area of 500 m^2. Your program should initially determine the distance between each node and all other nodes. Then the range of the nodes is given as input to the user. Using this range information, determine the neighbors of all the nodes.

Analysis: While analyzing your program, you are required to address the following points:

- How this is useful in the process of routing data?
- For a 3D topology, how would your program need to be changed?

MARKS DISTRIBUTION

Component	Maximum Marks	Marks Obtained
Preparation of Document	7	
Results	7	
Viva	6	
Total	**20**	

Submitted by: Satyajit Ghana

Register No: 17ETCS002159

1. Algorithm/Flowchart

generate_random_nodes(length, breadth):

```
1. nodes_pos.push_back({random_int(0, length), random_int(0, breadth)});
```

calculate_neighbour_table():

```
1. for i = 0 to nnodes - 1
2.     for j = 0 to nnodes - 1
3.             if dist = calc_dist(node_pos.at(i), node_pos.at(j) <
node_range.at(i)
4.                     neighbor_table.add_edge(i, j, dist)
```

calc_dist(pos1, pos2):

```
1. (post1.x - pos2.x)^2 + (pos1.y - pos2.y)^2
```

2. Program

main.cpp

```cpp
#include <iostream>

#include "network.hpp"

int main(int, char**) {
    using namespace std;

    Network mynet(10);

    // Generate the Random Nodes
    mynet.generate_random_nodes_with_dimensions(10, 50);

    mynet.print_nodes_pos();

    vector<int> ranges(10);
    cout << "Enter the Ranges for the 10 Nodes : ";
    for (int i = 0; i < 10; i++)
        cin >> ranges[i];

    // Set the Node Ranges given by the user and calculate the neighbour table
    mynet.set_node_ranges(ranges);

    mynet.print_neighbour_table();
}
```

network.cpp

```
#include "network.hpp"

#include <iostream>
#include <random>

Network::Network(int n_nodes) : n_nodes(n_nodes), neighbour_table(n_nodes), node_ranges(n
_nodes), nodes_pos(n_nodes) {}

void Network::generate_random_nodes_with_dimensions(int length, int breadth) {
    std::random_device random_device;
    std::mt19937 random_engine(random_device());
    std::uniform_int_distribution<int> l_dist(0, length);
    std::uniform_int_distribution<int> b_dist(0, breadth);

    this->nodes_pos.clear();

    for (int i = 0; i < this->n_nodes; i++) {
        nodes_pos.push_back({l_dist(random_engine), b_dist(random_engine)});
    }

    std::cout << this->n_nodes << " Random Nodes Generated" << std::endl;
}

void Network::print_nodes_pos() {
    int i = 0;
    for (auto& node : this->nodes_pos) {
        std::cout << "NODE " << i << " : (" << node.first << ", " << node.second << ")" <
< std::endl;
        i++;
    }
}

void Network::set_node_ranges(std::vector<int>& ranges) {
    if (ranges.size() != this->nodes_pos.size()) {
        throw "ranges length does not match nodes_pos length";
    }
    this->node_ranges.clear();

    this->node_ranges.assign(ranges.begin(), ranges.end());

    this->gen_fully_connected_network(this->node_ranges);
}

template <typename T>
double Network::calc_dist(std::pair<T, T> pos1, std::pair<T, T> pos2) {
    return std::sqrt(std::abs(
        ((pos1.first - pos2.first) * (pos1.first - pos2.first) + (pos1.second - pos2.seco
nd) * (pos1.second - pos2.second))));
}

// Generates the network from the given ranges
void Network::gen_fully_connected_network(std::vector<int>& ranges) {
    if (this->node_ranges.size() != this->nodes_pos.size()) {
        throw "WTF did you do ? and HTF did you do that ? i'm not even public ";
    }

    this->neighbour_table.clear_graph();

    for (int node_idx = 0; node_idx < this->nodes_pos.size(); node_idx++) {
        for (int node_jdx = 0; node_jdx < this->nodes_pos.size(); node_jdx++) {
            double dist = Network::calc_dist(this->nodes_pos.at(node_idx), this-
>nodes_pos.at(node_jdx));
            if (dist <= node_ranges.at(node_idx)) {
                this->neighbour_table.add_edge(node_idx, node_jdx, dist);
            }
        }
    }
}

void Network::print_neighbour_table() {
    if (this->node_ranges.empty()) {
        std::cout << "EMPTY" << std::endl;
        return;
    }

    this->neighbour_table.print_graph();
}
```

network.hpp

```
#pragma once

#include <math.h>
#include <vector>

#include "graph.hpp"

class Network {
   public:
    Network(int nodes);
    int n_nodes;
    std::vector<std::pair<int, int>> nodes_pos;
    std::vector<int> node_ranges;
    Graph neighbour_table;

    void set_node_ranges(std::vector<int>& ranges);
    void generate_random_nodes_with_dimensions(int length, int breadth);
    void print_nodes_pos();
    void print_neighbour_table();

    template <typename T>
    static double calc_dist(std::pair<T, T>, std::pair<T, T>);

   private:
    void gen_fully_connected_network(std::vector<int>& ranges);
};
```

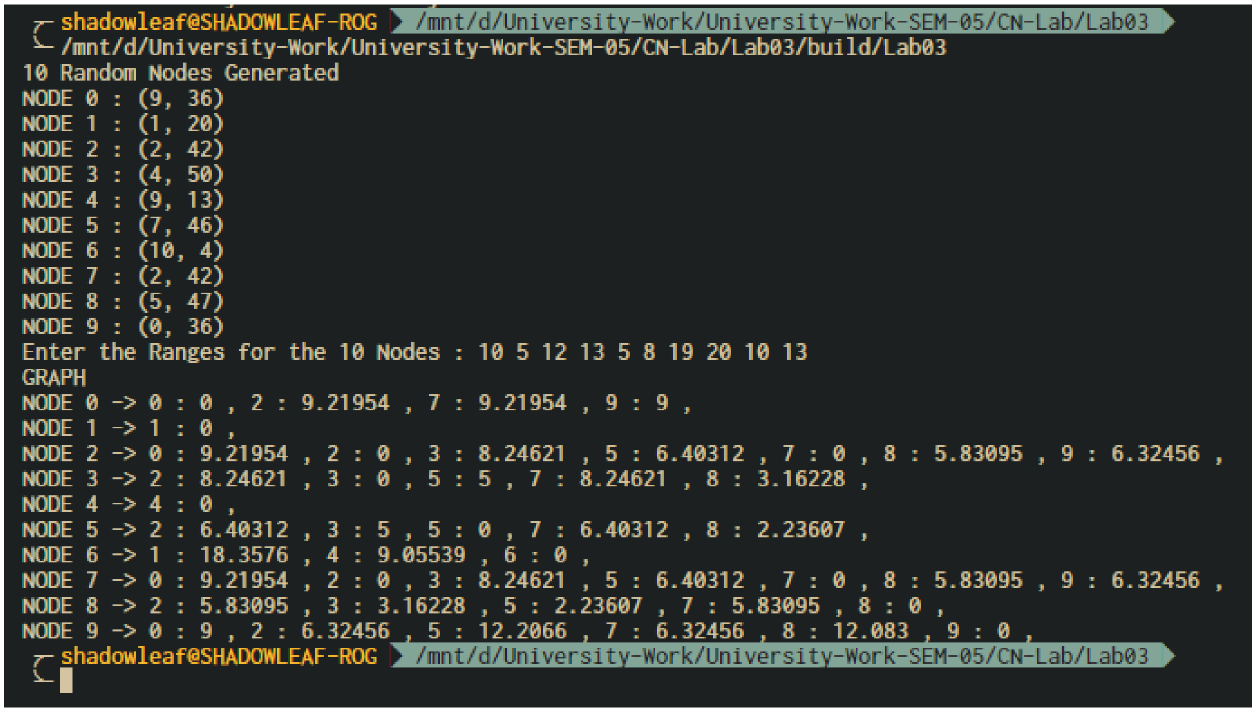
graph.cpp

```
#include "graph.hpp"

#include <iostream>

Graph::Graph(int nodes) : adj_list(nodes), order(nodes) {}

void Graph::add_edge(int src, int dest, double weight) {
    if (src >= adj_list.size() || dest >= adj_list.size()) {
        throw "Tried to Add Edge for Vertex that does not exist";
    }

    adj_list.at(src).push_back({dest, weight});
}

void Graph::print_graph() {
    std::cout << "GRAPH" << std::endl;
    int i = 0;
    for (auto& row : adj_list) {
        std::cout << "NODE " << i << " -> ";
        for (auto& ele : row) {
            std::cout << ele.first << " : " << ele.second << " , ";
        }
        std::cout << std::endl;
        i++;
    }
}
```

graph.hpp

```
#pragma once

#include <vector>

// Weighted DiGraph using Adjacency List
class Graph {
   public:
    Graph(int);
    std::vector<std::vector<std::pair<int, double>>> adj_list;

    const int order;

    void clear_graph() {
        this->adj_list.clear();
        this->adj_list.resize(order);
    };
    void add_edge(int src, int dest, double weight);

    bool is_empty() { return adj_list.empty(); };

    void print_graph();
};
```

3. Results



```
┌ shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab03
└ /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab03/build/Lab03
10 Random Nodes Generated
NODE 0 : (9, 36)
NODE 1 : (1, 20)
NODE 2 : (2, 42)
NODE 3 : (4, 50)
NODE 4 : (9, 13)
NODE 5 : (7, 46)
NODE 6 : (10, 4)
NODE 7 : (2, 42)
NODE 8 : (5, 47)
NODE 9 : (0, 36)
Enter the Ranges for the 10 Nodes : 10 5 12 13 5 8 19 20 10 13
GRAPH
NODE 0 -> 0 : 0 , 2 : 9.21954 , 7 : 9.21954 , 9 : 9 ,
NODE 1 -> 1 : 0 ,
NODE 2 -> 0 : 9.21954 , 2 : 0 , 3 : 8.24621 , 5 : 6.40312 , 7 : 0 , 8 : 5.83095 , 9 : 6.32456 ,
NODE 3 -> 2 : 8.24621 , 3 : 0 , 5 : 5 , 7 : 8.24621 , 8 : 3.16228 ,
NODE 4 -> 4 : 0 ,
NODE 5 -> 2 : 6.40312 , 3 : 5 , 5 : 0 , 7 : 6.40312 , 8 : 2.23607 ,
NODE 6 -> 1 : 18.3576 , 4 : 9.05539 , 6 : 0 ,
NODE 7 -> 0 : 9.21954 , 2 : 0 , 3 : 8.24621 , 5 : 6.40312 , 7 : 0 , 8 : 5.83095 , 9 : 6.32456 ,
NODE 8 -> 2 : 5.83095 , 3 : 3.16228 , 5 : 2.23607 , 7 : 5.83095 , 8 : 0 ,
NODE 9 -> 0 : 9 , 2 : 6.32456 , 5 : 12.2066 , 7 : 6.32456 , 8 : 12.083 , 9 : 0 ,
┌ shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab03
└ █
```

Figure 0-1 Output

In Figure 0-1 Output 10 random nodes are generated in an area of 500 square units using a standard normal distribution, then the range of each of those nodes is taken as input from the user.

Since this is a fully connected network, we traverse breadth first, across all connection, on each connection, the distance from the source to the destination is calculated, and if the range of the source node is greater than or equal to the range, then this connection is marked as valid, a matrix of such valid connection is made, this is the neighbor table for the network.

4. Analysis and Discussions

- How this is useful in the process of routing data?

Neighbour Table is useful since from the table, every node knows, which node can reach which other nodes in the network. Suppose a packet needs to be sent from a source to a destination node, the source node can calculate the shortest path the packet has to travel, it can also determine whether the packet will even reach the destination. This will optimize the bandwidth of the network and utilize the resources better. Since we know the neighbors and their destination, the route is predetermined, this is an added advantage for the network.

- For a 3D topology, how would your program need to be changed?

A 3D topology of network can be flattened to 2D network as well, it's just a matter of perception, the program would work even for the 3D network, the only change that has to be made is the dimensionality for the network, the position vectors of the nodes is now $\mathbb{R}^3$, the Euclidian distance formula will have to be changed, other than that everything is same, the Adjacency matrix is 2D and can very well store the connections for the 3D topology.

5. Conclusions

Neighbour table is a pretty simple mechanism by which the nodes can determine the paths in which the packet has to be travelled, the fact that the path is predetermined is the main advantage in this mechanism, one big disadvantage is that the neighbor table has to be determined again entirely if any node is damaged or goes offline. The complexity of the algorithm is $\mathrm{O}(\mathrm{n}^2)$. The time will increase polynomially.

6. Comments

   a. Limitations of the experiment

The complexity increases as the number of nodes increase.

   b. Limitations of the results obtained

The results are obtained for a very small set of nodes, which fails to depict the limitation of this experiment.

   c. Learning

We learnt a mechanism for generating a Neighbour table for a given topology of network.

d. Recommendations

The program needs to be simulated for a larger number of nodes, with 2D and 3D topology as well.