END-TERM EVALUATION

HUFFMAN HIGHWAY ACA

Project Mentors

SIDDHANT LALPURIA

SHIVAM SHARMA

GAUTAM CHANDAK

HARSH SHAH

Presentation by

RUDRANSH VERMA (230881)

Overview of the Project

First Half of the Project

The first half of the project include the basic C++ coding, some useful tips and some puzzling problems. It also includes the theory of Object Oriented Programming (OOP) and some practical questions related to real life scenarios. Also we were introduced with stacks, queues, linked lists, trees and binary search. Finally we were given Mid Evaluation Task of Huffman Coding for file (text and image separately) compression and decompression.

Second Half of the Project

The second half of the project starts off with introduction to bits and their manipulations. Essentially after that we were introduced to Graphs and solved some really good, and again practice real life problems. Then we were introduced with the dynamic programming and solved questions related to the same. Finally, it came to search algorithms Like Dijkstra and A* search algorithms for searching nodes in the graphs.

Assignment-wise brief analaysis of the Project.

Assignment 1

Assignment 1 was all about the basic C++ coding, specially targeted for the beginner coders or those who were unfamiliar with C++.

Assignment 2

For this assignment we had to go through Object Oriented Programming (OOPS), and its uses such as inheritance, operators etc. The question included real life problems such as building a account management system using OOPS.

Assignment 3

This assignment included introduction and using trees for different applications like Making record management systems.

Assignment 4

This assignment introduced Linked Lists and Doubly Linked Lists, and continued more on operations and types of trees and also given some questions for basic concept clarity of the C++ language.

Assignment 5

For this, we were introduced more elements from the Standard Template Library such as queues and stacks and were given some good practical questions to tackle.

Assignment 6

This assignment introduced us with graphs and some problems on this topic which helped us understand and implement graphs in the code with better clarity.

Assignment 7

In this, were introduced with concepts of dynamic programming (DP), and again were given some valuable questions for improving our understanding such as the commonly known Knapsack Problem but with some alterations to make it more challenging.

Assignment 8 (and the final End-Evaluation Task)

Finally, in this assignment we were asked to utilise all the knowledge that we gained throughout the project and convert a map of IIT Kanpur into a graph with edges as routes between the nodes (locations).
After that we were introduced with some common search algorithms such as Dijkstra and A* search algorithms for searching nodes in graphs and hence were tasked to find shortest route between any two locations at IIT Kanpur using the above graph.

Here's the main code for the Assignment 8

```
class Graph {
private:
    unordered_map<string, vector<Edge>> directConnection;

public:
    void addNode(const string& node) {
        //initialise node in the map
        directConnection[node] = vector<Edge>();
    }

    void addEdge(const string& first, const string& second, int distance) {
        directConnection[first].push_back ({ second, distance });
        directConnection[second].push_back ({ first, distance });
    }

    // prints each connection from every source
    void printGraph() const {
        for (const auto& pair : directConnection) {
            cout << pair.first << ": ";
            for (const auto& edge : pair.second) {
                cout << "(" << edge.destination << ", " << edge.distance << ") ";
            }
            cout << endl;
        }
    }

    vector<string> dijkstraSearch(const string& start, const string& end) const {
        unordered_map<string, int> distances;
        unordered_map<string, string> previous;
        auto compare = [&distances](const string& lhs, const string& rhs) {
            return distances.at(lhs) > distances.at(rhs);
        };
        priority_queue<string, vector<string>, decltype(compare)> queue(compare);

        for (const auto& pair : directConnection) {
            distances[pair.first] = INT_MAX;
            previous[pair.first] = "";
        }
        distances[start] = 0;
        queue.push(start);

        while (!queue.empty()) {
            string current = queue.top();
            queue.pop();

            if (current == end) break;

            for (const auto& edge : directConnection.at(current)) {
                int alt = distances[current] + edge.distance;
                if (alt < distances[edge.destination]) {
                    distances[edge.destination] = alt;
                    previous[edge.destination] = current;
                    queue.push(edge.destination);
                }
            }
        }

        vector<string> path;
        for (string at = end; !at.empty(); at = previous[at]) {
            path.push_back(at);
        }
        reverse(path.begin(), path.end());
        return path;
    }
};
```

THANK YOU