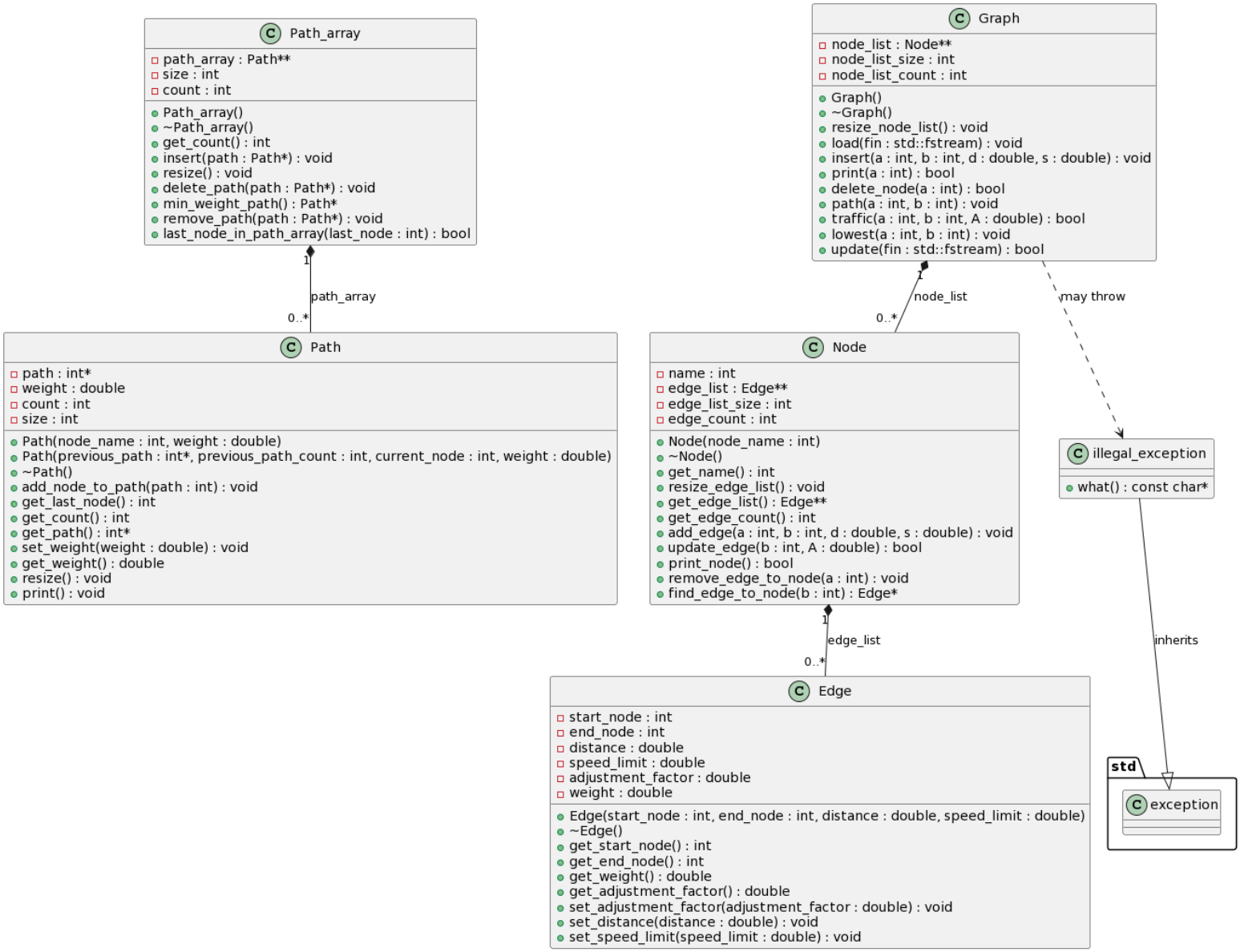
ECE250 LAB4 Design Document

Author: Bowen Zheng, student #: 20949303, student ID: b57zheng

Class UMLs



Edge Class:

- Class Design:

The design of the Edge class is pivotal for representing connections within a graph structure. A major design decision is the incorporation of both directional attributes and dynamic factors like traffic. The class also provides methods to retrieve and update edge attributes, ensuring the graph can be efficiently navigated and modified in response to changing conditions.

- Constructor & Destructor:

The overloaded constructor was designed to support the creation of edges. The destructor was chosen to be a default destructor since there is no dynamic allocation of memories to be released.

Node Class:

- Class Designs:

The Node class in the graph structure is designed for efficiency and adaptability. It employs dynamic allocation for edges, allowing for flexible edge management as the graph changes. The class uses lazy initialization and automatic resizing to handle varying graph sizes efficiently.

- Constructor & Destructor:

The Node class constructor supports the creation of nodes, initializes an edge pointer array, setting a default size and using nullptr to ensure safe memory access. This choice balances memory allocation efficiency with the need for potential edge expansions. The destructor method deallocates the edge pointer array, ensuring clean memory management upon node deletion.

- Function: add_edge

This function checks for an existing edge between two nodes before creating a new one, then update speed and distance for the edge, demonstrating efficiency by preventing duplicate edges. It is central to building the graph's connectivity.

- Function: update_edge

This function updates the traffic adjustment factor of an edge between given two nodes if it exists. This is crucial for dynamic graph scenarios where edge weights (representing traffic conditions) need to reflect real-time data.

- Function: print_node

This helper function prints all edges connected to a node to support the Print method in the Graph class.

- Function: remove_edge_to_node

This function removes an edge connecting to a specific node. This function is a helper function to support the delete_node method in the Graph class.

- Function: find_edge_to_node

This function locates a specific edge connected to a node. This function is a helper function to support the update method in the Graph class, where the adjustment factor of a specific edge is needed.

- Function: Resize_edge_list

Expands the array's capacity, doubling its size. This is a critical design for handling increasing paths without running out of space, crucial for large graphs.

Path Class:

- Class Designs:

The Path class was designed to support Dijkstra's algorithm implementation. It efficiently represents a path in the graph, containing a sequence of nodes and the associated path weight. The Path class's primary role is to facilitate the tracking of the shortest paths during the execution of Dijkstra's algorithm.

- Constructors & Destructor:

The class has two constructors: one for initializing a new path with a single node and another for extending an existing path with additional nodes, supporting the progressive nature of Dijkstra's algorithms. The destructor deallocates memories allocated to path array, avoiding leaks.

- Function: add_node_to_path

This function adds node name to the end of the path array. This function was designed as a helper function for the Dijkstra's algorithm implemented for lowest and path method in the Graph class.

Path_array Class:

- Class Designs:

The Path_array class is designed to manage and manipulate a collection of Path objects in Dijkstra's graph algorithms. The significant design decision is to use a dynamically allocated array of pointers to Path objects, enabling the class to handle an evolving set of paths during graph traversal.

- Constructor & Destructor:

The constructor initializes the array. The destructor deallocates any Path objects stored in the array and then frees the array itself, preventing memory leaks.

- Function: insert

Adds a new path to the array. It checks if the array needs resizing before insertion, ensuring dynamic adaptability to the number of paths.

- Function: resize

Expands the array's capacity, doubling its size. This is a critical design for handling increasing paths without running out of space, crucial for large graphs.

- Function: delete_path

Removes a specified path from the array. It ensures memory is freed properly for the removed path, maintaining efficient memory usage.

- Function: Min_weight_path

Finds and returns the path with the minimum weight in the array of pointers pointing to the paths. This function is central to implementing Dijkstra's algorithm for shortest paths.

- Function: Remove_path

Deletes a path from the array but doesn't deallocate it, used for rearranging paths during graph traversal.

- Function: Last_node_in_path_array

Checks if a given node is the last node in any of the paths. This function is key in preventing revisits to nodes, enhancing efficiency in pathfinding tasks.

Graph Class:

- Class Design:

The Graph class is designed as the central structure for managing a network of nodes and edges, crucial for representing and manipulating graph data. Key functionalities include graph traversal and pathfinding, enabled by integrating auxiliary classes like Path and Path_array. The class also incorporates error handling through the illegal_exception class ensures stability against invalid inputs.

- Constructor & Destructor:

constructors initialize class attributes and allocate necessary memory, ensuring objects begin in a consistent state. Destructors, conversely, are key for memory management, safely deallocating resources to prevent leaks.

- Function: insert

This function creates an edge between two nodes (a and b) with distance 'd' and speed limit 's'. It's essential for constructing the graph, as it builds the connections that define the network. If a node doesn't exist, it will be created.

- Function: print

This function prints the edges connected to node 'a', providing a visual representation of a node's connections within the graph.

- Function: delete_node

This function removes a node and all its connected edges from the graph.

- Function: path

This function uses Dijkstra's algorithm to find the shortest path between nodes 'a' and 'b'. It's vital for route finding within the graph.

- Function: traffic

This function updates the adjustment factor 'A' on the edge between nodes 'a' and 'b'. It allows the graph to reflect dynamic changes, like traffic conditions, affecting edge traversal weight.

- Function: lowest

This function uses Dijkstra's algorithm to find the path with the lowest cost between nodes 'a' and 'b'. It's key for optimization tasks, like finding the most efficient route.

- Function: update

This function reads TRAFFIC commands data from a file to update the adjustment factor 'A'.

- Function: load

This function reads INSERT commands data from a file to construct the graph network.

- Function: resize_node_list

Expands the array's capacity, doubling its size. This is a critical design for handling increasing paths without running out of space, crucial for large graphs.

- Runtime Analysis: Dijkstra's algorithm
    - The algorithm initializes the first note in the path array with weight 0. This operation is O(1).
    - In each iteration, the algorithm extracts the node with min weight from the path array. In the worst case, every node is extracted once. Hence, the operation is O(|V|).
    - For each min weight path's end node, expand all edges. In the worst case, this operation repeats for every edge in the graph. Hence, the operation is O(|E|).
    - Total runtime for extraction is O(|log(V)|)
    - Total runtime for edge expansion is O(|E))
    - Hence, the total runtime is O(|E|log|V|).

illegal_exception Class:

- Class Design:

This class was designed as a custom exception for the graph implementation, inherits from the standard std::exception in C++. By overriding the what() method, it delivers pre-specified error information.