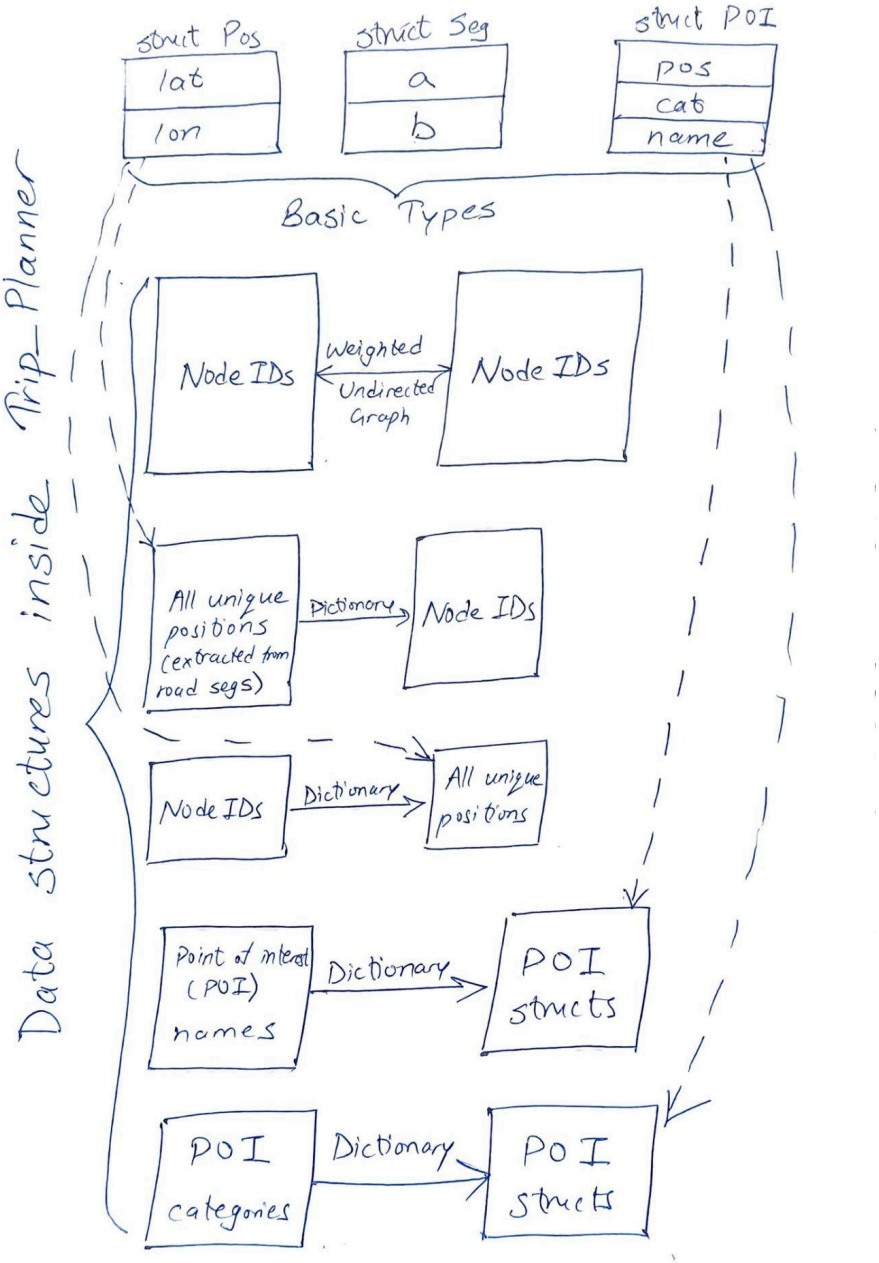
CS 214 Final Project Design Doc 1

Entity-relation diagram



Building blocks and ADTs

Purpose	**Lines in code submission**	**Building block/ADT**
Representing position	52-54	`struct` called `Pos` containing two fields: ● `lat` (a `num` representing latitude), and ● `lon` (a `num` representing longitude).
Representing road segment	56-58	`struct` called `Seg` containing two fields: ● `a` (a `Pos` representing one endpoint), and ● `b` (a `Pos` representing another endpoint).
Representing point-of-interest (POI)	60-63	`struct` called `POI` containing three fields: ● `pos` (a `Pos` representing its position), ● `cat` (a `string` representing its category), ● and `name` (a `string` representing its name).
Representing map	95, 118	A weighted undirected graph where each node is a unique endpoint of a road segment. The weight is the Euclidean distance if two nodes are connected, or `None` otherwise.
Representing position-to-node mappings	96, 107	A dictionary mapping every unique `Pos` determined from the vector of `Seg`s used to initialize the `TripPlanner`, to a natural number representing a node ID in the graph.
Representing node-to-position mappings	97, 108	A dictionary mapping every node ID to a unique `Pos`. Together, this and the dictionary above describe a bidirectional mapping.
Representing name-to-POI mappings	98, 121	A dictionary mapping POI names (keys) to POI `struct`s.
Representing category-to-POI mappings	99, 122	A dictionary mapping POI category names (keys) to a linked list containing POI `struct`s of that category.

Function implementations

`locate_all`

Look up the POI associated with the inputted category using the dictionary containing the category-to-POI mappings. From the linked list of `POI`s associated with the category, extract the positions and return them as a linked list.

`plan_route`

Look up the position of the inputted destination POI using the dictionary containing the name-to-POI mappings. Then, use a single-source shortest path-finding algorithm (ideally Dijkstra's) to find the shortest route between the starting and ending positions.

`find_nearby`

From the starting position, compute the distance to all POIs in the inputted category using `locate_all` and `plan_route`. After a POI's distance is computed, input it into a min-heap. After all the POIs have been entered into the heap, remove `n` POIs, chain them using a linked list, then return the linked list.