CompSci 214 - Data Structures and Algorithms

Project Report

Hayah Ubaid

Introduction

In my implementation of the Trip Planner project, I used the following ADTs: dictionaries, a graph, and priority queues. I also wrote various non trivial algorithms including dijkstra's algorithm and a helper function to help me initialize the Trip Planner class.

Design Choices

Dictionaries

- **Role:** I use two dictionaries in my code, my first dictionary is called AList and it uses indices as the key and points as the values and the second dictionary, “BList '' uses points as the keys and indices as the values. These dictionaries are used to establish a mapping between points and their indices, which is useful for efficient access and retrieval of point information during various operations in the TripPlanner class. For example, in the plan_route method, BList is used to retrieve the index of the starting point (src_lat, src_lon) and the endpoint with a specific name (dst_name). Similarly, AList is used to retrieve the point corresponding to an index in the prev list during the path finding process in the dijkstra method as well as to create my cons path list in the plan route method.
- **Data Structure:** Association Lists
- **Why this data structure:** I chose Association Lists because of its simplicity in implementing compared to a hashtable. Additionally, there was no need for hashing in my code and Hash tables rely on hashing functions to map keys to indices in an underlying array, which may take up more space. In contrast, Association Lists don't require hashing and can directly use the linked list structure to store key-value pairs. Hash tables also typically require more memory compared to Association Lists because in addition to storing the key-value pairs, hash tables need to allocate buckets to hold the entries.

Graph

- **Role:** The graph ADT plays an important role in solving the problem of finding the shortest path between two points. It is necessary to represent the road network as it allows for the efficient traversal and path finding that is needed for this project.
- **Data Structure:** Weighted-Undirected graph (WuGraph)
- **Why this data structure:** I chose the WuGraph data structure over other choices because it allows for efficient storage and retrieval of edges and distances and provides the necessary functionality for setting edges and their weights, as well as retrieving adjacent vertices and edge weights. It also supports the weighted nature of the road segments, which is essential for finding the shortest path using Dijkstra's algorithm. I chose the WuGraph data structure over a unweighted or directed graph because the weights are necessary to represent the distances between points and this project used road segments that were bi-directional rather than uni-directional, so an undirected graph is a better choice than a directed graph.

Priority Queue

- **Role:** The Priority Queue ADT is important because it resolves the issue of finding the shortest distance of a point in Djikstra's algorithm. Its remove min and find min feature makes it very efficient in finding the shortest distance and its sorting feature helps in the find_nearby() function as well.
- **Data Structure:** Binary Heap
- **Why this data structure:** I used a binary heap to implement a priority queue rather than using a linked list or an array to do so because the insertion, finding the minimum and removal operations are more efficient using a binary heap as the time complexity is O(logn) whereas linked lists and arrays would have a linear time complexity for these operations making them slower. Furthermore, After removing the minimum element from the priority queue, a binary heap efficiently reorganizes the heap to maintain the heap property. This ensures that the next minimum element can be easily and efficiently accessed. Contrastly, removing the minimum element from a linked list or an array would require shifting or rearranging the remaining elements, resulting in higher time complexity.

Dijkstra's Algorithm

- **Role:** Dijkstra's algorithm plays a crucial role in solving the problem of finding the shortest route between two points in the given code. It is used to calculate the shortest paths from a given starting point to all other points in the graph. In the context of the TripPlanner class, the dijkstra method is responsible for finding the shortest paths from the starting point to all other points of interest.
- **Why this algorithm:** I chose Dijkstra's algorithm because it efficiently explores the graph and updates the distances from a given starting point to all the other reachable points. In the context of trip planning, Dijkstra's algorithm is suitable because it can find the shortest route between locations, considering the distances associated with each road segment. I chose Dijkstra's algorithm over others because the endpoint in this code is not necessarily a single fixed node but rather a set of nodes belonging to a specific category (name, category, etc). Dijkstra's algorithm is used for finding the shortest path from a single starting point to multiple destinations, which matches the requirement of finding all interest points of a specific category. In this code, using A* would not have any significant advantages over using Dijkstra's because A* algorithm relies on a heuristic function to guide the search more efficiently, but this code has no heuristic information so I chose Dijkstra's over A*.

Helper Function

- **Role:** The helper function's role is initializing the points and calculating the length of the road segments in the TripPlanner class. It is called in the initialization to set up the necessary data structures for the class. For example, it is used to set up self.length which is used in many parts of the code.
- **Why this algorithm:** I wrote this helper function because it efficiently determines unique points from the road segments. It iterates through the road segments and checks if each endpoint is already in the points list. If not, it adds the endpoint to the list. By using flags and checking conditions, it ensures that each point is added only once, even if it appears multiple times in the road segments. The algorithm then returns the points list and its length, which are used in other parts of the code.

Open Ended Questions

1. Suppose your boss asked you, the implementer, to make businesses that provide discounts to trip planner users rank higher in find_nearby. How would you change your design and implementation to do this? What positive effects could adding this feature have? What negative effects could adding this feature have? Would you agree to do it? Why?

Answer: To implement this feature, I would create a new helper function that takes in the discount factor which would be from 0 to 1 (representing the discount percentage that business is offering) and then uses Dijskstra's algorithm to get the initial distance list, copies that list onto a new list (so that the self.distance_list variable holding all real distances is not changed) and updates the distances in the new list for those businesses only to 1 minus their discount factor, keeping all other distances to businesses not offering a discount the same. This way businesses offering a higher discount will show as having the lowest distance and will be given priority (If a business offers a 75% discount, their distance would be shown as 0.25). I will change my find_nearby function to take in the new distance list created by the new helper function rather than the self.distance_list variable that holds the real distances. This way none of the other functions or methods of the code are changed. The positive effects of doing this is that more businesses may start offering trip planner users discounts which will in turn incentivize more users to start using trip planner in order to get the discounts. And for the users, the positive effects will be that due to the discounts they will get they may end up paying less for a service that they traveled further for. However, there are numerous negative effects as well. Firstly, it compromises the integrity and it would be unethical if we are not transparent about doing this because users expect the ranking to be purely based on distance and if they notice it is not then the reliability of Trip Planner is called into question. Furthermore, in emergency situations if someone is trying to get to for example, the nearest hospital, the find nearby function will show a hospital that may be much further away and this could have terrible consequences. I would not agree to add this feature unless I can add a flag of some sort as well saying something like "sponsored" or "discounts available" so that users are aware that that business may not actually be the closest but those businesses still get exposure and therefore continue offering discounts to trip planner users.

2. Suppose your boss asked you, the implementer, to add a feature to "shadow ban" some businesses from the trip planner: the businesses will see them- selves in the trip planner query results, but they would not appear in query results for trip planner users. What positive effects could this feature have? What negative effects could this feature have? Would you agree to do it? Why?

Answer: Positive effects of shadow-banning businesses from the trip planner could be that if a business engages in harmful, unethical or illegal practices, shadowbanning them could protect users. Additionally, if the business is known to scam users or is consistently providing badly rated services then shadowbanning them may be good for the users and will ensure they have a pleasant experience using trip planner. However, there are negative effects to consider. The shadow ban feature may unfairly filter out businesses if the shadow banning criteria is not defined and transparent to users and businesses. Some businesses may also feel unfairly targeted by this feature. For the users, negative effects include businesses that may be relevant to their query may end up being hidden and they will not find what they are looking for and this may push them to stop using trip planner altogether. I would not implement this feature because I think it is unethical as it hides information from users and businesses will not know they are shadowbanned. Furthermore, the negative effects outweigh the potential positive effects. Alternatively, I would recommend flagging those companies by displaying warning signs so that users are informed of a potential threat but can still choose to go and use their services.