Final Project: Trip Planner

For this particular assignment, even arms-length collaboration is discouraged. Most of the learning value of this assignment comes from figuring out the overall structure, moving pieces, and how they fit together, and less from the implementation of that plan. To get the most out of this assignment, we therefore recommend you do the former completely independently.

For this assignment you will implement a trip planning API that provides routing and searching services. Unlike previous homework assignments, the representation is not defined for you, nor have we specified which data structures and algorithms to use. Instead, you may choose from among the data structures you have implemented for previous assignments as well as any algorithms that we either implemented or saw in class. Selecting appropriate data structures and algorithms is up to you, and your grade will reflect these choices.

In `planner.rkt`, I've supplied a definition of the interface you need to implement, as well as descriptions of the input and output data.

1 Problem Overview

Your trip planner will store map data representing three kinds of entities and answer two kinds of queries about them. The following two subsections describe the entities and the queries in turn.

1.1 Entities

- A *position* has a latitude and a longitude, both numbers.
- A *road segment* has two endpoints, both positions.
- A *point-of-interest* (*POI*) has a position, a category (a string), and a name (a string). The name of a point-of-interest is unique across all points-of-interest, but a category may be shared by multiple points-of-interest.

See figure 1 for an example of a map containing the three kinds of entities. Note that each position can contain zero, one, or more POIs.

We will make three assumptions about the segments and their positions:

1. All roads are two-way roads.
2. The length of a road segment is the standard Euclidian distance between its endpoints.
3. Points-of-interest cannot be in the middle of a road segment; they are either at an endpoint, or completely isolated.

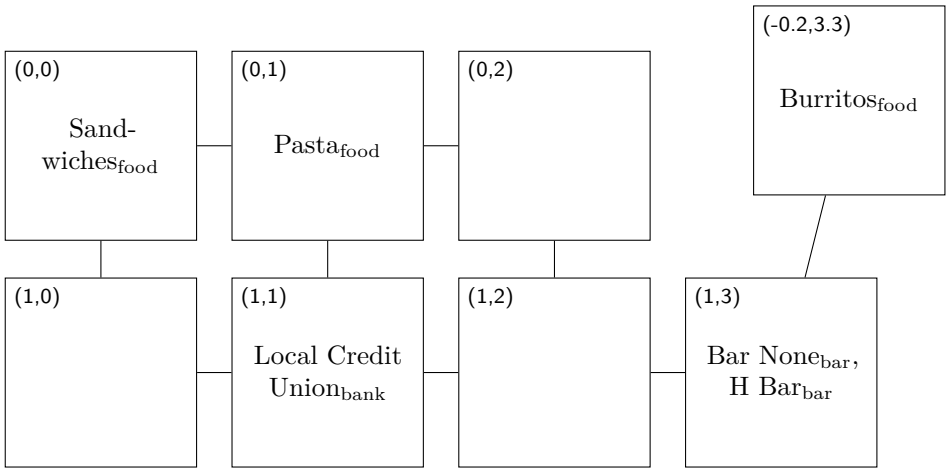

Figure 1: Example map with POIs labeled by latitude and longitude

Start	**Name**	**Result path**
(0,0)	Sandwiches	(0,0)
(0,1)	Sandwiches	(0,1)–(0,0)
(1,1)	Sandwiches	(1,1)–(1,0)–(0,0) *or* (1,1)–(0,1)–(0,0)
(1,1)	Burritos	(1,1)–(1,2)–(1,3)–(-0.2,3.3)
(1,1)	Sushi	*nothing*

Table 1: Example *find-route* queries

Start	**Category**	n	**Result POIs**
(1,3)	food	1	Burritos
(0,2)	food	1	Pasta
(0,2)	food	2	Pasta, Sandwiches
(0,2)	food	3	Pasta, Sandwiches, Burritos
(0,2)	food	4	Pasta, Sandwiches, Burritos
(0,2)	bar	1	Bar None *or* H Bar
(0,2)	bar	2	Bar None, H Bar *or* H Bar, Bar None
(0,2)	bar	3	Bar None, H Bar *or* H Bar, Bar None
(0,2)	school	5	*Nothing*

Table 2: Example *find-nearby* queries
(your implementation should return the full POIs, not just the names)

Representation	Alias	Purpose
`num?`	`Lat?`	latitude
`num?`	`Lon?`	longitude
`str?`	`Cat?`	POI category
`str?`	`Name?`	POI name
`[Lat?, Lon?]`		position
`[Lat?, Lon?, Lat?, Lon?]`		road segment
`[Lat?, Lon?, Cat?, Name?]`		POI
vector of road segments		input road segments
vector of POIs		input POIs
linked list of positions		*find-route* result
linked list of POIs		*find-nearby* result

Table 3: Summary of vocabulary types

1.2 Queries

Your trip planner must support two forms of queries:

find-route Takes a starting position (latitude and longitude) and the name of a point-of-interest; returns a shortest path from the starting position to the named point-of-interest. Returns the empty list if the destination does not exist or is unreachable.

find-nearby Takes a starting position (latitude and longitude), a point-of-interest category, and a limit n; returns the (up to) n points-of-interest in the given category nearest the starting position. Resolve ties however you want, and order points-of-interest within the list in any order you want.

For some example queries and results see tables 1 and 2. For specifics on the interface your code code should conform to, read on.

Advice: you may find it easier to do *find-route* first, then do *find-nearby*.

2 API Specification

The trip planner API is specified as a DSSL2 interface named `TRIP_PLANNER` (provided in the starter code), which you must implement as a class named `TripPlanner`.

The `TRIP_PLANNER` interface refers to a variety of "vocabulary types" in order to represent the three kinds of entities discussed in section 1.1, as well as collections thereof. Thus, before describing the interface itself, we must define these types.

2.1 Vocabulary Types

The types described in this section are defined in three layers in the next three subsections: basic types, entities, and collections of entities. All the types are summarized in table 3.

2.1.1 Basic Types

The basic vocabulary types include latitude and longitude (represented as numbers) and POI categories and names (represented as strings).

2.1.2 Entity Types

The three entity types are represented as vectors (not structs) of basic types. In particular:

- A position is represented as a 2-element vector containing the latitude and longitude: `[Lat?, Lon?]`.
- A road segment is represented as a 4-element vector containing the latitude and longitude of one end position followed by the latitude and longitude of the other: `[Lat?, Lon?, Lat?, Lon?]`.
- A point-of-interest is represented as a 4-element vector containing the latitude and longitude of its position, then its category, and then its name: `[Lat?, Lon?, Cat?, Name?]`.

These types are intended for communication between the API and the client, but you will probably want to define richer representations for internal usage in your implementation.

2.1.3 Collection Types

The API uses collections of entities in two places:

- The constructor for your `TripPlanner` class needs to take a collection of road segments and a collection of points-of-interest from which to build its map. These will be passed as a vector of road segments and a vector of POIs, respectively.
- The queries defined by the `TRIP_PLANNER` interface (and thus implemented by your `TripPlanner` class) need to return sequences of positions (paths) and sets of nearby POIs. These results must be represented as linked lists (using `cons`) of positions and POIs, respectively.

2.2 The `TripPlanner` Class and `TRIP_PLANNER` Interface

2.2.1 Creating a `TripPlanner` Instance

Your `TripPlanner` constructor must take two arguments: a vector of road segments and a vector of points-of-interest, in that order. Each should be as described in section 2.1.3.

2.2.2 Using a `TRIP_PLANNER` Instance

Your `TripPlanner` class must implement the `TRIP_PLANNER` interface, which defines a method corresponding to each of the two trip planner query operations described in section 1.2.

3 Dependencies

Your solution is likely to rely on a number of different ADTs and data structures, including some which you've implemented as part of previous assignments.

As we did with homework 4, we're providing you with compiled versions of solutions, this time for for homeworks 2, 3, 4, and 5. That way, everyone can start on firm footing regardless of the state of their previous homeworks.

As before, extract the `project-lib.zip` archive in the same directory as your `planner.rkt` file and import the files you need. The file names and interfaces are identical to the ones used for these assignments.

For example, `import 'project-lib/dictionaries.rkt'` would import a dictionary library whose interface is the one from homework 3. For reference, the graph representation used by our `graph.rkt` is an adjacency matrix.

Alternatively, you can use your own homework solutions (e.g., if they are better suited to the task at hand). Because you will only turn in your `planner.rkt` file, you will need to copy over to it any homework code you wish to use.

You may also import modules from the DSSL2 standard library, such as the `cons` and `sbox_hash` libraries.

Any of the code I posted to Canvas alongside lectures are also fair game. You'll just have to copy their implementation to your `planner.rkt` file. And be sure to acknowledge your sources.

4 Report

We expect you to put care and thought into your choices of ADTs, data structures, and algorithms. To assess this, you will need to write a brief report explaining your decisions.

For each use of an ADT in your program, list the following:

- What role does it play in solving the problem?
- What data structure did you pick for it?
- Why did you pick that data structure over other choices?

Each of these should be no more than one or two sentences. For conciseness and legibility, please format as a series of bullet points:

- *ADT 1*
  - *role (1-2 sentences)*
  - *which data structure*
  - *why this one (1-2 sentences)*
- *ADT 2*
  - *role (1-2 sentences)*
  - *which data structure*
  - *why this one (1-2 sentences)*
- ...

You don't need to describe intermediate structs, or intermediate arrays that are not meaningful sequences in their own right. When in doubt, mention it.

Then, for each non-trivial algorithm in your program, explain the following:

- What role does it play in solving the problem?
- Why did you pick that algorithm over other choices?

By non-trivial algorithm, we mean an algorithm that is sophisticated enough to be worth discussing in its own right. For example, merge sort is non-trivial, but looping over an array to find the largest element is not. When in doubt, mention it.

Please follow the same format as for the ADT portion of the report.

Note: Unlike self-evaluations for previous assignments, your report should describe your solution as it stands in your *final* submission.

Note: Even if you cannot get your code working, submit your report! If your design is sound, you will get partial credit for it with your report.

5 Honor code

Every homework assignment you hand in must begin with the following definition (taken from the Provost's website[1]; see that for a more detailed explanation of these points):

```
let eight_principles = ["Know your rights.",
     "Acknowledge your sources.",
     "Protect your work.",
     "Avoid suspicion.",
     "Do your own work.",
     "Never falsify a record or permit another person to do so.",
     "Never fabricate data, citations, or experimental results.",
     "Always tell the truth when discussing your work with your instructor."]
```

If the definition is not present, you receive no credit for the assignment.

Note: Be careful about formatting the above in your source code! Depending on your pdf reader, directly copy-pasting may not yield valid DSSL2 formatting. To avoid surprises, be sure to test your code *after* copying the above definition.

6 Deliverables

Your completed `planner.rkt`, containing

- a complete definition of the `TripPlanner` class,
- tests to cover all cases and be confident of your code's correctness.
- the honor code.

and your report in pdf format.

Note that each should be submitted to a separate Canvas assignment.

Submissions which crash or otherwise fail to execute will receive a grade of zero for the code portion.

[1] `http://www.northwestern.edu/provost/students/integrity/rules.html`