Computer Science Basics: Data Structures

Ryan Mason

Data Structure Part 1

For part one, I chose the approach of using a hashmap containing linked lists. I chose this structure because I am familiar with python lists and dictionaries and was easily able to visualize using this approach to solve the problem. You can create keys that are the first letter of the food type, and map the types to the keys using a linked list. An example of this using python built-in types would look something like this:

```
from collections import defaultdict

letter_to_type_dict = defaultdict(list)

for i in types:

        letter_to_type_dict[i[0]] = [j for j in types if j.startswith(i[0])]
```

This provides you with a dictionary of first letter of the food type mapped to a list of the restaurants that start with that letter.

The runtime of searching for a food type is O(n). For example, if you have two food types (African, American), worst case you would have to run through the loop twice, once when you search “a”, and again to narrow it down to your food type further by providing an extra letter to your original search criteria (“af” for “African”).

Data Structure Part 2

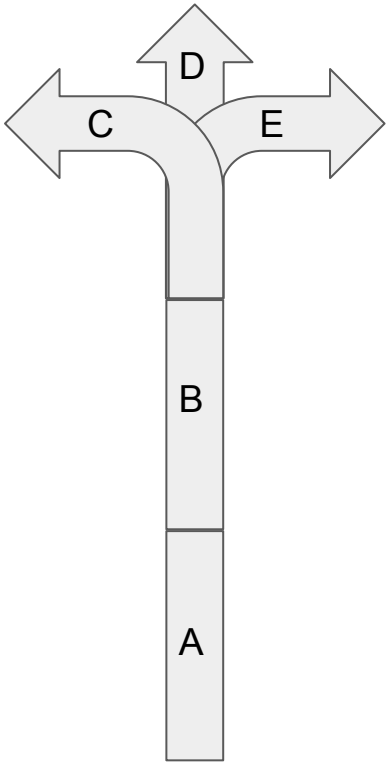
For part two, I also chose the hashmap to linked list approach because it solves the problem for part two the same way as it is solved for part one. I used a namedtuple to store the restaurant info rather than a hashmap. Defining a namedtuple allows you to group all of the restaurant data into one structure, and each structure can be a single value in the linked list so you can retrieve the values in a clear and organized fashion rather than nesting hashmaps inside of another hashmap.

The runtime of part 2 is also O(n), because when you choose a restaurant type it will have *n* number of restaurants in the linked list, and you must traverse the linked list to retrieve and print all of the data for *n* items.

Data Structure Utilization

One way to implement a data structure covered in this course would be to use a graph to create a map. For example, imagine you have to create a large map of a city. You could break up the map into smaller sections that contain road info. Each vertex would be a segment of road, and the vertices would share an edge with neighboring lanes. Each vertex could contain road geometry, neighboring roads, stop signs, traffic lights, etc. One example would be if you have a long stretch of straight road with no intersections. Each road segment could contain the road geometry and the current road (vertex) could share an edge with the previous road and the next road segment. At an intersection, a vertex could have a number of shared edges equal to the amount of different paths you could take (left, right, straight). A big advantage of using a graph for this application would allow you to store a large map, but be able to edit sections without having to touch other sections. If a lane is closed off at an intersection, you could easily remove the neighboring vertex without having to modify the entire map. If your map was a large continuous piece, you would have to modify the entire geometry. Another advantage of this data structure is that you could also add a weight to the neighboring vertices in order to determine the fastest route to your destination. I have provided a visualization of this on the following slide.

Example Illustration

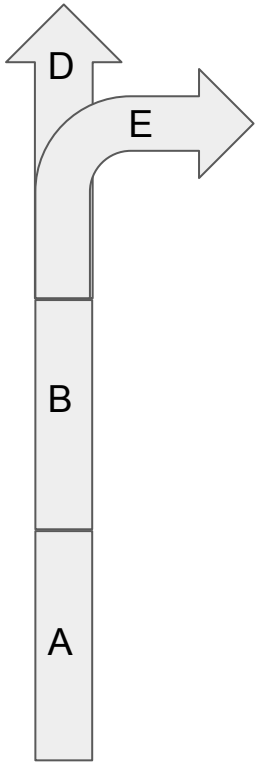



```
RoadSegment:

Name: B

previous_roads: [A]

next_roads: [C, D, E]

segment_polygon: [(0,0), (0,1), (1,1), (1,0)]

stop_signs: None

traffic_lights: [...]
```

In this example, if we are in road segment B, we could query the road segment and see its attributes. Using this data stored in the vertex, we can determine which segments of road are available to us as we continue on our route, as well as get the geometry of the road and determine if there are any traffic lights or stop signs within the road.

In the next slide, we can see how easy it is to modify this map if road segment C is closed off.

Example Illustration Continued




```
RoadSegment:

Name: B

previous_roads: [A]

next_roads: [D, E]

segment_polygon: [(0,0), (0,1), (1,1), (1,0)]

stop_signs: None

traffic_lights: [...]
```

Now, since road C has been closed off, all we have to do is remove that road segment from our list of next_roads associated with road segment B. None of the other road segments need to be modified with this approach.