~~Get all the distance data between each node~~

~~Dijkstra – Generate the shortest path of the given ordered nodes~~

Events should be an object or something.

* Should Event be a reference type or value type?
* I THINK IT SHOULD BE A CLASS, since we are going to have a “life cycle” and events with similar values WON’T BE CONSIDERED the same thing

Event Class

class Event {

// All time-related variables should have at most TWO INTS at all times

// The first int should refer to the hour. The second int should refer to the minutes.

// This will be in military time. Can easily convert non-military time input at user input time.

var startTime = [Int]()

var endTime = [Int]()

let eventDuration = [Int]()

var eventName = “”

var eventAddress = “”

var weight = 0

// We may or may not need some variables/particular data structures

// to contain the data needed for us to use Google API.

// I say may or may not because I haven’t read through the Google API fully.

// But it seems reasonable to think so.

}

Will need a priority queue called orderedWeightedEvents. Each will be, naturally, ordered by their weights. (Maybe at user input, each event will have the weight of 0, so it wouldn’t matter?)

* Originally thought this would be fine as THE data structure holding all the events BUT what if you have lockable events? How much weight would you assign in order to get that event where you want to be?
* An easier way to get around this would perhaps be to just have a regular list/array of events
* The stuff in the priority queue will simply be popped and put into the list

Have a list of events called events. This is what any GUI function should use to display the schedule properly.

Making the Perfect Schedule Algorithm

Say you have A-E things, with movable events A,C,E; locked B, D.

If you have locked in events

* Ask user “when they want to start the day” (can leave blank)
* Ask user “when they want to end the day” (can leave blank)
* Ask if user has a starting event (e.g., apartment)
* Ask if user has an ending event (e.g., apartment)
* Get an ordered list of events from the user
* From there,
  + ASSIGN\_WEIGHTS\_TO\_EVENTS\_FUNCTION (have 2 lists, but here, one is the movable list, and one is the locked list)
  + Check the location of A (then C, then E) with the location of the next locked event, B (then D).
    1. Is A closest to B? (get distance)
       - If so, add 1 to A’s “weight”
       - If not, add nothing
    2. Is A closest to starting location? (get distance)
       - If so, add 1 to A’s “weight”
       - If not, add nothing
    3. Is the transportation time between START-A is within the time between STARTING\_TIME & B’s STARTING\_TIME?
       - If so, add 1 to A’s “weight”
       - If not, add nothing
  + ~~Compare the weights of all the movable events.~~ 
    - ~~Schedule the highest weighted event.~~
    - ~~If there is a tie, just pick the first “highest” event.~~
    - ~~If there are no weighted events, don’t schedule anything~~
  + Add event to the priority queue data structure, which should keep our order without us having to do anything
* REPEAT THE PROCESS TILL THERE ARE NO MORE MOVABLE EVENTS

If you don’t have ANY locked in events

* Ask user “when they want to start the day” (can leave blank)
* Ask user “when they want to end the day” (can leave blank)
* Ask if user has a starting event/location (e.g., apartment)
* Ask if user has an ending event/location (e.g., apartment)
* Take the ordered list
* Assign weights (same function as used in the above case, but this time the two lists passed in should be identical)
  + Check the location of first event, then…, in the list with next event…
  + If no next event, just add it to the priority queue
  + Repeat until all events have weights and are in priority queue
* ~~Compare the weights~~
* ~~Schedule from highest weights to lowest weights~~

This function should return the priority queue which can be used to systematically graphically created.

Assign Weights Function

1. Take in two lists
   1. If we KNOW we have lockable events, list\_1 should ONLY CONTAIN movable events, list\_2 should ONLY CONTAIN lockable events
   2. Otherwise, passed in list\_of\_events = list\_1’s contents = list\_2 contents
2. Iterate through list\_1 and list\_2
   1. If distance(list\_1\_item, list\_2\_item),