League of Patience Writeup

(a) Describe an algorithm solution to this problem. Feel free to talk about how you would adapt an algorithm we covered in class.

- A modified Dijkstra's algorithm would be used.
- Start with the source node and check it’s adjacent values according to Dijkstra's. Since we have a start time we would add the time to complete the quest to our start time (represented as the weight from u to v).
- Next, we would find out when we are able to start the next quest. We would find the difference of the next quest time plus the previous step. In other words, It would take us the quest time (weight u to v) plus the difference to start the next quest. From that point on we follow Dijkstra's algorithm.

(b) What is the complexity of your proposed solution in (a)?

O(V^2)

(c) See the file LeagueOfPatience.java, the method "genericShortest". Note you can run the LeagueOfPatience.java file and the method will output the solution from that method. Which algorithm is this genericShortest method implementing?

Dijkstra's algorithm. O(E+V^2)

(d) In the file LeagueOfPatience.java, how would you use the existing code to help you implement your algorithm? The existing code only handles one piece of data per edge, so describe some modifications. Note the helper methods available to you, including one that simulates the game's API that returns the next quest time.

- First add "T" as a parameter to genericShortestPath, in order to find S - T.
- Find node u with the method findNextToProcess(). Go through the for loop that checks all adjacent vertices to u. When an adjacent vertex found check that it's total time as described on part a (third bullet point) is less than previous accounted times to reach vertex v. if it's less update it to the new calculated value.

(e) What's the current complexity of "genericShortest" given V vertices and E edges? How would you make the "genericShortest" implementation faster? Describe any algorithm changes or data structure changes. What's the complexity of the optimal implementation?

- Current "genericShortest" complexity O(E+V^2).
- Instead of using an adjacency matrix we should use and adjacency list.
- Instead of using findNextToProcess() to obtain next u, we should use a priority queue our next vertex to use would be easily found by using the poll method. Which would be the vertex with the smallest value.
- The complexity should be: O(E + V logV)