2. a. I would use Dijkstra's algorithm. Dijkstra's algorithm, most often, is used to find the shortest path to a target node from a source node. Dijkstra's algorithm, using the source node, builds a set of node with the least distance to the source node. This is reminiscient of greedy algorithm by virtue of finding the shortest path from a set node; producing, a globally optimized solution.
One advantage, although, a double edge sword, is a faster run time than Bellman-Ford's algorithm but an inability to deal with negatively weighed edges.

To implement Dijkstra's algorithm some changes are needed.
The waiting time at a station needs calculations and to produce a minimum cost journey.
The helper matrix is referenced to ensure minimum cost from source node to destination node.
To find the next train that can be caught to travel to the next station is given by this formula:
The time it takes to get between two adjacent stations u and v added to how frequently the train stops at u on its way to v multiplied by a arbitrary index subtracted by the start time.
I would initialize 2 arrays for shortest time and boolean array.
I would initialize the greatest value and false for these arrays
After calculating the number of vertices from given time in adjacency vertices, find the subsequent vertex to run calculations
Check its neighboring nodes and calculate the cost of those nodes. If minimum cost is undermined, then update min. cost, and repeat

b. It is $O(V^2)$.

c. It implements Dijkstra's algorithm.

d. Since the existing code handles one piece of data per edge, I could do this to help me implement my algorithm: Take into account the waiting time at station to get a connecting train and the time of the connecting train.

e. Current complexity is $O(v^3)$. It can be optimized to a complexity of $O(E+V \log V)$ by implementation of minimum priority queue.