Programming implementation:
I firstly modified the Handle class so that each handle has a key, a undefined-type value and a position to represent the handle's position in the heap.

Then, I modified the PriorityQueue class. I created a swap and a heapify function in the bottom as helpers, which implements the idea of priority queue we learned in class. Each priority queue has an ArrayLis<Handle<T>> variable, which is the minimum priority queue we are using. For the constructor, I add a null handle to the heap so that all real handles would start from index 1, which is more convenient for processing binary heaps. Then I set isEmpty() to return true if the size of the queue is large than 1, because it is size 1, it means there is only a dummy handle in that queue, which is meaningless.

I implemented insert(), min(), extractMin(), decreaseKey(), handleGetKey(), handleGetValue(), toString() in this class, all using the methods and structures we learned in class. One thing to notice is that when I swap two handles, I would update their positions to reflect their new location in the queue.

Finally, I modified the ShortestPath class, for which I created a ProrityQueue<Integer> variable to use a the queue for finding minimum keys, a local Multigraph G and an ArrayList of handles of the type Integer. Then, I implement the Dijkstra's Algorithm in the constructor, for which I set an if statement for startTime to determine if there is layover effect. If there is, then the program would use the functions I put in to find a realistic shortest path. Otherwise, it would just act like a normal Dijkstra.

I modified the Vertex class to include two integers for each vertex: its parent vertex id and its parent edge id. I did this to keep tract of both because it is essential when we try to find the returnPath. I set the parents as -1 if the vertex is the starting point.

Then, in the returnPath function, I simply run a while loop on calling the parent of the last vertex on shortest path, and store the edge ids we find on the way. Then I just convert those ids into an integer array in the correct order and return the array. If we were to find the shortest path for the starting point, the array would be at length 0.

Running Time:
Because I implemented PriorityQueue by the binary heap structure and ShortestPaths as Dijkstra's Algorithm, the running time constraints are satisfied.