My algorithm models this problem as a graph in which each city is a vertex and each highway is a weighted edge. Cities with diners are stored in a hashset. My Solution contains four parts after setting up the graph and hashset.

1. Classic Dijsktra’s shortest path algorithm from the goal city (ElogV).
2. A modified version of Dijkstra’s SPT with extra work being done every time it removes a vertex from the IndexMinPQ. For each edge incident to v, when the algorithm iterates through g.adj(v), if it’s part of a shortest path, the algorithm maintains data on both the number of shortest paths to the new vertex that the edge leads to, as well as the number of shortest paths containing a diner that lead to this new vertex. Assuming that data is properly maintained (how will be illustrated later- I didn’t want to interrupt the flow of the solution), this SPT algorithm will result in data on the number of paths with a diner that lead to the closest vertex to the destination of every unique shortest path, as well as the distance to each vertex from the source city. This SPT is (ELogV+E) because there is extra constant time work done for each edge.
3. A loop through each city with a diner, using the previously calculated values to find the lowest distance of a path containing a diner(source->vertex+vertex->destination). This is O(m).
4. One final use of Dijskra’s SPT algorithm from the destination. At each vertex popped of the IndexMinPQ, the algorithm checks if the distance the shortest path to this vertex from the destinaton+the distance of the shortest path from this vertex to the source= the min identified in step 3. If it does, the algorithm adds the number of paths with a diner that pass through this vertex to the result. This is O(ELogV+V).

All that is left to do is to explain how the information about paths is properly maintained to achieve that assumed result. The following chart illustrates the intended result.

In the diagram, below, the shortest path to 6 will be length six 1->2->6, which will not contain a diner. The goal of step 2 of my algorithm would be to store 1 non-diner and ) diner shortest paths to 6, one diner shortest path to 4 (of length 7), which will be used in the result when added to the distance from 4 to the destination. To ensure that the info is properly maintained, the info stored at an old vertex must be deleted once a new vertex is added to a shortest path and the info is transferred. Thus, when 4 is reached, the information at 3 is deleted, so as not to run into double counting issues. One more fact must be ensured: that information is only transferred if the next vertex is closer to the destination or the current path does not contain a diner, because otherwise it will be moved from a shortest path containing a diner to a longer one, thus the info at 4 will never be transferred to 5, so the shortest diner path won’t be shifted to 5 and deleted from 4, corrupting it.

3

3

3

diner

4

diner

5

3

2

When a vertex is removed from the PQ:

For each edge in g.adj(vertex):

V= old vertex, W=new vertex this edge leads to

If(distance from source to w>distance from source to v+ edge weight):

Do regular Djikstra stuff

#non-diner paths to w=#non-diner paths to v

If(w has diner && (w is closer to dest than v||diner paths to v=0):

diner paths to w=non diner paths to w

Else if(w is closer to dest than v||diner paths to v=0):

diner paths to w=diner paths to v

Else if(distance from source to w=distance from source to v+ edge weight):

Non-diner paths to w+= non-diner paths to v

If(w has diner && (w is closer to dest than v||diner paths to v=0):

diner paths to w+=non-diner paths to v

Else if(w is closer to dest than v||diner paths to v=0):

diner paths to w+=diner paths to v

With the above steps, the invariant is that every vertex when an edge to it is relaxed will contain the number of unique non-diner shortest paths that lead to it, as well as the number of unique diner shortest paths that lead to it, unless that information was already there and transferred to a vertex further on at an earlier stage in the algorithm, or that vertex is further way from the destination than an already existing shortest diner path.