Mapping method: In each consumer we have a set of mapped nodes. Whenever new nodes are added to the graph, we do not change the mapping of the old set of nodes, but we give 1/10th of these new nodes to each consumer. If the initial graph has n1 nodes and each consumer process is mapped to 1/10th of these n1 nodes. Now the producer adds some nodes to the graph so that it has n1+n2 nodes. Then only the new n2 nodes are mapped to the consumers and the mapping of the original n1 nodes does not change.

For non-optimized version: In case the -optimize flag is not mentioned, multi-source dijkstra is run by each consumer and then the resulting paths are written/appended to the file. In each iteration all the known paths are recomputed and then they are appended/written to the file.

For the optimized version: If -optimize flag is mentioned then in the 1st iteration the consumer has to run multi-source dijkstra as in the case of non-optimized version. However for the subsequent iterations, the consumer will run multisource breadth first search in place of multisource dijkstra the reason being that all the weights in the graph are 1, so we can use bfs for shortest path computation. The worst case running time of multisource dijkstra is O(ElogV) whereas that for multisource bfs is O(V+E) for a graph with V nodes and E edges. Thus multisource bfs is asymptotically faster than multisource dijkstra algorithm.

Also a node x will be traversed (added to the bfs queue) only if its shortest path has been updated. To ensure this we maintain a dist array where dist[i] denotes the shortest distance of i from any of the source nodes in the current mapped set. The mapping method is such that for any consumer process always source nodes are added to the mapped set but they are never removed from the mapped set. This means once a source node is mapped to a consumer, it always belongs to this consumer irrespective of the number of iterations. This means that as more source nodes are added to the mapped set of the consumer process and as new nodes and edges are added to the graph, the value of dist[i] will either decrease or stay the same for any node i. So we can re-use the dist array from previous iteration to avoid traversing nodes for which the shortest path does not change.

To reconstruct the path we maintain a parent array par, where
par[i] = node traversed immediately before node i traversing on the shortest path starting from some source node

par[i] changes if and only if dist[i] changes. So we can reuse the par array from the previous iteration to avoid re-computing paths.

Now for those nodes for which the shortest path does indeed change, the dist[i] and par[i] values are updated by the multisource bfs algorithm appropriately.

For optimized/non-optimized version, once the par array is properly constructed by the appropriate algorithm, we can print the paths for each node in the graph from the nearest source node using the par array.