Operating Systems Laboratory
(CS39002)

Assignment: 3
Optimization Report for Dijkstra's Algorithm
Date: 27th February, 2023

Submitted by:

Group - 12 :
Syam Manoj Kumar Paila (20CS10041)
Anirudh Aitipamula (20CS30002)
Ganjikunta Vijay Kumar (20CS30018)
Vineet Amol Pippal (20CS30058)

Problem Statement:

The producer only adds a limited number of nodes and/or edges to the (much larger) network, which might not change many shortest paths. Design and implement a strategy to optimize the process of re-computing shortest paths. Write a report with the design strategy. This version will run with a “-optimize” flag to your code.

Shared Memory and Processes:

The shared memory is created by the main process, which stores the initial graph, as an array of struct graphedge, indicating the edges in the graph. It also stores an array {NUM_NODES, NUM_EDGES} which indicates the number of nodes and edges in the graph.

The producer process wakes up and adds new edges to the shared memory, indicating connections between the newly added nodes. The consumer process wakes up and runs the dijkstra’s algorithm

Optimization:

Each consumer process is allocated a set of graph nodes based on its index, given to it as a command line argument. If the consumer has index i, it initially picks nodes in range 10 * i% to (10 * i + 10)% (example: the first consumer picks the first 10%, the second consumer picks the next 10% and so on). This initial allocation is done when the consumer process is first invoked.

Next, each consumer process computes the shortest paths from each of the nodes from its set to all the other nodes and stores the shortest paths locally using Single-Source Dijkstra’s Algorithm, while printing

them to a file. Suppose, p new nodes were added by the producer the next time a consumer woke up. It would automatically know that nodes in range

[(NUM_NODES + p)*i/10, (NUM_NODES + p)*(i+10)/10]

belong to it. (where NUM_NODES is the number of nodes before changes were made to the graph). This would repeat over and over in an unoptimized setting.

When consumers are invoked with the “-optimize” flag given as the fourth argument on the command line (after shmkey1, shmkey2 and index), they will start to keep track of the newly added nodes.

The consumer first runs the dijkstra’s algorithm on the nodes that were lost to it when the update happened i.e. for consumer-i, dijkstra’s algorithm is first run on the nodes which got transferred to the consumer-(i-1) in this iteration, taking all the other nodes corresponding to other customers as destinations.

Next, the consumer runs the algorithm taking the newly added nodes as sources and all the other nodes as destination nodes. This way, the shortest paths to all the nodes changed for a consumer get updated.

Now, the consumer sees if for a node belonging to it, there is an alternate shorter path after the addition of the newer nodes to it i.e. if for sum node in the graph N, the path length from a customer node C to N is larger than the path length from that N to some newly added node Q + path length from Q to C, update the shortest path.

This way, the consumer only runs the dijsktra’s shortest path algorithm on the newly added / lost nodes and is able to efficiently update the shortest paths to all the nodes, taking its nodes as sources. This optimization hence is better than running the algorithm on all the customer nodes over all the iterations.
