Mapping of nodes to consumers:

Assignment of nodes to i-th consumer:
If it is the first iteration, **all** the nodes in the graph are divided into 10 equal segments and the i-th segment is assigned to the i-th consumer.
Else, divide the set of **newly added nodes** into 10 equal segments and assign to the i-th consumer the i-th segment **and** the nodes that were assigned to it in the immediate previous iteration

Data structures maintained by each consumer:

1. a matrix *dist_mat* of dimensions *MAX_NODES * MAX_NODES* : *dist_mat[i][j]* contains the minimum distance between nodes *i* and *j* calculated in the previous iteration
2. a matrix *path_mat* of dimension *MAX_NODES * MAX_NODES* : *path_mat[i][j]* contains an array of nodes describing the shortest path from *i* to *j* calculate in the previous iteration

note that the contents of *dist_mat[i][j]* and *path_mat[i][j]* are meaningful if and only if any one of *i* or *j* (or both) was mapped to that particular consumer that is maintaining these matrices

Optimized algorithm:
(*key idea: Dijkstra is expensive, don't perform Dijkstra for all mapped nodes)*

```
for each consumer c in [0,9]:
        assign nodes to c following the rule as described before
        A = {nodes assigned to c}
        if (count of new nodes == 0):
                continue
        if (first iteration):
                perform Dijkstra algorithm with nodes of A as source and populate the dist_mat
        and path_mat accordingly
        else:
                S = set of all nodes that were added to the graph after the last iteration of the
        consumer
                perform Dijkstra with nodes of S only as source
                for each node i in A - (S^A):
                        for each node j in the graph:
                                for each node k in S:
                                        dist_mat[i][j] = min(dist_mat[i][j],dist_mat[i][k]+dist_mat[j][k])
                                        if (dist_mat[i][j] is updated):
                                                path_mat[i][j] = path_mat[i][k] append
        rev(path_mat[j][k])
        print the shortest paths to the output file
```

Complexity and improvement:

Reason for improvement: the size of the set S is bounded (max 30) [consumer executes at least once between two iterations of producer]

if n is the total number of nodes in the graph, for each iteration of a consumer time complexity is:

(in first iteration) $O(n/10 * n^2)$ - for performing Dijkstra for all n/10 nodes
(in subsequent iterations) $O(30 * n^2)$ - for performing Dijkstra for |S| nodes and |S| <= 30
+ $O(n * n/10 * 30)$ - updating for n * n/10 pairs of nodes

(overall, excluding the first iteration) $O(n^2)$

In the unoptimized setup every iteration is like the first iteration of the optimized consumer, with time complexity $O(n^3)$ required for performing Dijkstra for all n/10 nodes assigned to the consumer.

The speedup is achieved at the cost of increased space complexity required to maintain the shortest distance(dist_mat) and paths(path_mat) calculated during each iteration.

The rule followed for assignment/mapping of nodes to consumers ensures that the values accessed by the consumer from its distance and path matrices are correct and meaningful.

Time complexity analysis

For each consumer process,running time of that process was calculated using the sum of utime+stime from the file "/proc/<pid>/stat".
Utime: Amount of time that this process has been scheduled in user mode, measured in clock ticks (divide by sysconf(_SC_CLK_TCK)).
Stime: Amount of time that this process has been scheduled in kernel mode, measured in clock ticks (divide by sysconf(_SC_CLK_TCK))
(Reference: GNU manual for proc, https://man7.org/linux/man-pages/man5/proc.5.html)

Average time over all consumers

	Time in sec for first 5 iterations	Time in sec for first 10 iterations
Unoptimized	5.7	23.4
Optimized	1.6	4.7