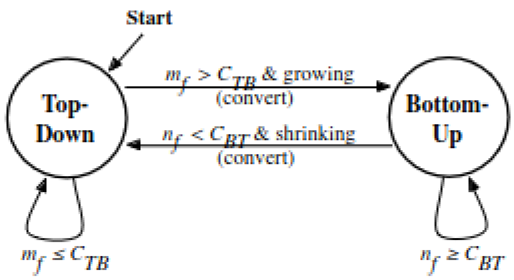
Assignment 2

1. I have submitted the grader for both 8 vCPU(local machine) and 16 vCPU(gcp) in the zip file.
2. Describe BFS
    a. We had to implement BFS in two ways:
        i. Top Down approach:
            1. For each node in the frontier, we traverse the outgoing edges and check if that node has been visited or not;if not, add it to the new_frontier.
            2. To parallelize the same, each thread is allotted a node in the frontier to process the neighbors.
            3. Race conditions can happen when the distance vector is checked for a node to be added in the new_frontier and then updated.
            4. For the same, I have made these parts of the code atomic and they are updated to a local new_frontier for every thread.
            5. Then the nodes in the local_new_frontier are added to the actual new frontier, again, atomically.
            6. A barrier is needed at this stage before the frontier and the new frontier are updated for the next iteration. #pragma omp parallel gives an barrier here by default before returning the vectors where they are updated.
        ii. Bottom Up:
            1. Similar to the above, the difference being, now the incoming edges were checked and one atomic operation was decreased.
        iii. I tried to switch between serial and parallel dynamically to decrease the synchronization overhead when the frontier is smaller but didn't couldn't figure out when I needed to switch.
    b. For the hybrid part,
        i. Both approaches have their advantages and disadvantages. When the frontier is large, it is better to use a bottom up approach else the top down approach. For the same I used the frontier size/total_nodes ratio. I used a trial and error method to find a ratio that gave best performance.
        ii. I also went through this paper https://parlab.eecs.berkeley.edu/sites/all/parlab/files/main.pdf and used the formula to see when to switch between the top_down and bottom_up approaches.



$$m_f > \frac{m_u}{\alpha} = C_{TB}$$

$$n_f < \frac{n}{\beta} = C_{BT}$$

where ,
m_f = edges adjacent to frontier
m_u = edges adjacent to
unvisited nodes
n_f = vertices in frontier
n = total vertices

iii. But the formula didn't give good results as the values of alpha and beta may not be accurate for these graphs.
iv. Hence, I submitted the solution with the trial and error method giving better results. The other method is commented.

c. The performance is varying in every run and I think it is mainly due to the synchronization overhead as I was not sure when to switch between serial/parallel or top_down and bottom_up approaches. It could also be the workload imbalance as the no.of neighbors for each node in the frontier, making each thread take different time.