Problems of Chapter 4

4.3 The adjacency matrix, when squared, stores the information about the paths of length 2 in G. The $(i, j)^{th}$ entry in M^2 indicates the number of length two paths from vertex i to vertex j. Similarly, the $(i, j)^{th}$ entry in M^3 indicates the number of length three paths from vertex i to vertex j. Note that $< 1, 5, 1, 2 >$ is considered a length 3 path. Knowing M^3 it is easy to check if there exists a cycle of length 3.

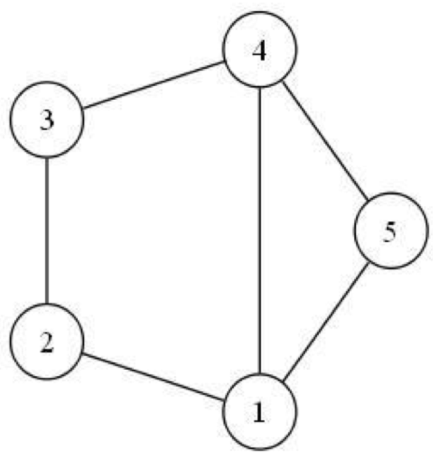

Figure 1: Graph $G = (V, E)$

0.	1.	0.	1.	1.
1.	0.	1.	0.	0.
0.	1.	0.	1.	0.
1.	0.	1.	0.	1.
1.	0.	0.	1.	0.

Figure 2: Adjacency matrix M of G

3.	0.	2.	1.	1.
0.	2.	0.	2.	1.
2.	0.	2.	0.	1.
1.	2.	0.	3.	1.
1.	1.	1.	1.	2.

Figure 3: M^2

2.	5.	1.	6.	4.
5.	0.	4.	1.	2.
1.	4.	0.	5.	2.
6.	1.	5.	2.	4.
4.	2.	2.	4.	2.

Figure 4: M^3

4.4 The proposed algorithm fails to identify the cycles that involve two back edges during the DFS.

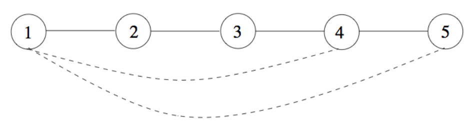

Figure 3: Counterexample for 4.4.

4.9 If there is a negative cycle involving s, clearly, there doesn't exist any shortest paths. Suppose it is the case that there is no such negative cycle, we can show that Dijkstra's algorithm can be applied. We first add a positive constant, say K, to each edge incident on s. Now all the edges in the modified graph G' have positive weights. The shortest path to any vertex v contains one edge incident to s. The true distance to v is the distance to v in G' minus K.

4.10 Apply Bellman-Ford (BF) algorithm k rounds. It can be shown that after the i^{th} round, BF algorithm will finish computing the shortest paths of vertices of length at most i.

4.11 Very similar to 4.4.

4.13 The valid path from s to t cannot contain any edge of length greater than L. Eliminate these edges before Dijkstra's algorithm is applied.

4.14 Consider a shortest path from u to v throughout v_0. We can find this path by precomputing the shortest path tree in G from v_0, and computing the shortest path tree from v_0 to the graph G^R where all edges of G are reversed.

4.19 There are two approaches. **Approach 1:** We make the following modifications to Dijkstra's algorithm to take into account node weights: In the initialization phase, dist(s) = w(s). In the update phase, we use dist(u) + l(u, v) + w(v) instead of dist(u) + l(u, v). Analysis of correctness and running time are exactly the same as in Dijkstras algorithm.

Approach 2: We make the following modification to each node of G. Dijkstra's algorithm is applied to the modified graph.