Q1
1. a)

See file TransitionDiagram.png

1.b)

See the files "output_0.1.png" , "output_0.5.png", "output_0.8.png" and "output_1.png" for the plots

There are four possible optimal paths, all of length 5, and the algorithm outputs any one of these during the various trial runs

- RRRR -> URRR -> URFR -> UUFR -> UUFF
- RRRR -> URRR -> UURR -> UUFR -> UUFF
- RRRR -> RRFR -> URFR -> URFF -> UUFF
- RRRR -> RRFR -> RRFF -> URFF -> UUFF

In my trial runs the best path was of length 5 and the agent either returns a path of length 5 or a path of length 51 (which means the agent didn't learn or it failed as 50 was the cut-off length set in the code)

<u>Effect of number of episodes on the learning</u>
As is evident from all the plots, the number of failures (points in the graph with values 51) is very less or none after training episodes 100. This is because greater the number of episodes, greater is the updates to the Q-matrix and the agent has learned more information.

So greater the number of training episodes, greater is the learning and the agents converges to the optimal solution.

<u>A problem of the current training method</u>

An interesting question that I wanted to find answer was why the agent who had succeeded earlier with fewer number of episodes, later fails with more number of episodes. That is in the plots you can see the points 51 come up once in a while for sometime. The same agent who succeeded and report the optimal path in 30 episodes, fails with 82 episodes. Why?

This is because of the current way it is trained. As can be seen from the code, the learn_Q method exits when it reaches the numEpisodes passed as argument and each episode is a single update of the Q matrix. But the reward matrix is sparse and has a non-zero positive reward only for the final step. That means the Q-matrix will be updated with non-zero values only if the goal state is reached at least once in a training episode. Since actions are selected randomly there is good chance that we do not pick up the goal state during the learnQ method execution, and the method will exit with a sparse Q-matrix.

Not only the agent has to see the goal state during training, It should also see all the states on the optimal path (including the starting state) enough number of times. That is during a training (one execution of learnQ) all the states (from start to goal) on the optimal path has to picked up by the action selection logic enough number of times. Only then agent will have a dense enough Q-matrix that will help it decide or choose at each state what is the best next state. The reward of the final step has to be back propagated across the states in the optimal path back to the startState during a learnQ execution

To support this, below are data collected for one execution of assign3QL.py

- For path length 51 (when agent failed), goalState was picked up 2.48 times on average and startState was picked up 5.74 times on average by the random action selection logic

- For path length 5 (when agent succeeded), goalState was picked up 10.44 times on average and startState was picked up 11.71 times on average by the random action selection logic

As you can see from above two points, when agent succeeded it had seen start and goal states much more times during the training than when it failed.

A suggestion for improvement is to change the definition of an episode to make sure that all states are adequately visited during the Q-learning. One such definition of episode would be “any sequence from startState to goalState”

Effect of α on the learning

Three of the metrics I am using for defining the impact of α are :

- ***FIRST*** – which is the first occurrence (at what number of episodes) of the best path (length 5). It indicates how quickly it found the best path. Lower the value of ***FIRST***, the more quickly it found the best path
- ***LAST*** - which is the last occurrence (at what number of episodes) of the worst path (length = 51). It indicates how quickly it stabilized and remained at the best path. Lower the value of ***LAST***, the more quickly it stabilized
- ***DIFF*** – the number of occurrences of 5 – the number of occurrences of 51. Indicates how much agent was able to succeed rather than fail over the course of a trail run (up to 200 episodes in the code)

So a good value of α, should have lower values for both ***FIRST*** and ***LAST*** and a higher value for ***DIFF***

The average value of these metrics for 50 trial runs (where each trail run is one execution of assign3QL.py). Each table represents the average over 50 trials.

α	***FIRST***	***LAST***	***DIFF***
0.1	40.60	102.00	42.00
0.5	41.36	104.72	39.98
0.8	40.80	105.20	38.60
1	39.76	113.36	38.28

α	***FIRST***	***LAST***	***DIFF***
0.1	40.36	101.24	41.44
0.5	42.88	107.96	40.20
0.8	39.52	105.72	39.38
1	39.64	110.32	38.00

α	***FIRST***	***LAST***	***DIFF***
0.1	40.52	104.12	40.52
0.5	40.44	109.48	39.10
0.8	42.60	112.44	38.86
1	39.76	110.16	38.40

α is the learning rate and decides the extend by which the agent acquires or incorporates new information.
As is evident from the 3 tables above, the smaller values of **α,** stabilizes quickly (i.e. lower values of ***LAST***) and has higher success (larger values of ***DIFF***), and higher values of **α,** finds the best path quicker (i.e. lower values of ***FIRST***).
From this we can say that a good approach for α would be to have a variable value of α a fixed value. It is best to have α start with value 1 and then decrease slowly as time progresses. This causes the agent to acquire new information quickly at the beginning but after enough learning has occurred then a lower value of α would help to converge or stabilize faster

Q2

Coded in python. There are 2 files "factor.py" and "factor_operations.py"

Q3

Note: For question 2 and 3, the initial CPTs (from the Bayesian Network) are stored in files inside the folder "Initial_CPTs". Please edit the file paths appropriately for testing.

The math for how the factor list was obtained for questions 3b to 3e can be seen in the file "equations.pdf"

3.a)

For transition diagram, see file "Bayesian Network.png"
For CPTs see file "CPTs.pdf"

3.b)

Query = prior probability that Fido will howl = P(FH = True)

factorList =

[CPT_FH_given_FS_FM_NDG, CPT_FS, CPT_FM, CPT_NDG_given_NA_FM, CPT_NA]
queryVariables = ['FH']
orderedListOfHiddenVariables = ['NA', 'FS', 'FM', 'NDG']
evidenceList = []

Answer = prior probability that Fido will howl (i.e. Pr(FH = True)) = 0.073

For initial factors, intermediate factors and final probability distribution generated during variable elimination, see file "3b.pdf" in results folder

3.c)

Query = P(FS = True | FM = True, FH = True)

factorList = [CPT_FH_given_FS_FM_NDG, CPT_NDG_given_NA_FM, CPT_FS, CPT_FM, CPT_NA]
queryVariables = ['FS']
orderedListOfHiddenVariables = ['NA', 'NDG']
evidenceList = [('FM', 'True'), ('FH', 'True')]

Answer = Probability that Fido is sick given that it is full moon and Fido is howling = P(FS = True | FM = True, FH = True) = 0.086

For initial factors, intermediate factors and final probability distribution generated during variable elimination, see file "3c.pdf" in results folder

3.d)

Query = P(FS = True | FB = True, FM = True, FH = True)

factorList = [CPT_FH_given_FS_FM_NDG, CPT_NDG_given_NA_FM, CPT_FB_given_FS, CPT_FS, CPT_FM, CPT_NA]
queryVariables = ['FS']
orderedListOfHiddenVariables = ['NA', 'NDG']
evidenceList = [('FM', 'True'), ('FH', 'True'), ('FB', 'True')]

Answer = Probability that Fido is sick given bowl is full, it is full moon and Fido is howling = 0.361

For initial factors, intermediate factors and final probability distribution generated during variable elimination, see file “3d.pdf” in results folder

3.e)

Query = P(FS = True| FB = True, FM = True, FH = True, NA = True)

factorList = [CPT_FH_given_FS_FM_NDG, CPT_NDG_given_NA_FM, CPT_FB_given_FS, CPT_FS, CPT_FM, CPT_NA]
queryVariables = ['FS']
orderedListOfHiddenVariables = ['NDG']
evidenceList = [('FM', 'True'), ('FH', 'True'), ('FB', 'True'), ('NA', 'True')]

For initial factors, intermediate factors and final probability distribution generated during variable elimination, see file “3e.pdf” in results folder

Answer = Probability that Fido is sick given bowl is full, it is full moon, Fido is howling and neighbour is away = 0.338