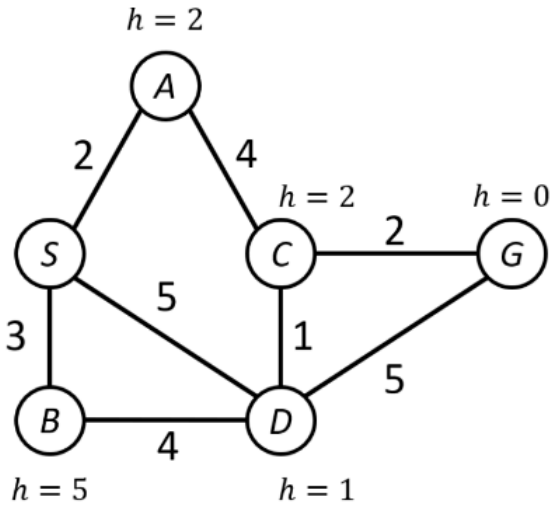
Problem 2 (24 points)

In the state space graph below, S is the start state and G is the goal state. Costs are shown along edges and heuristic values are shown adjacent to each node. All edges are undirected (or bidirectional). Assume that node expansion returns nodes in **reverse alphabetical order** of the corresponding states, and that search algorithms expand states in **reverse alphabetical order** when ties are present.

The *early* goal test refers to checking that a state is the goal right *before* insertion into the frontier. The *late* goal test refers to checking it right *after* popping from the frontier. Usage of a *reached* table allows for multiple path and cycle pruning.



1. **We use DFS (Depth-First Search) along the undirected (bidirectional) edges.**
    a. Early goal test and reached table: Our sequence by construction will check that a state is the goal right before insertion into the frontier.
        i. [(S, 0)] starting point of the sequence which is popped trivially
        ii. [(D, 5), (B, 3), (A, 2)] pop (D, 5) and perform goal test on (G,10)
        iii. Sequence Solution reached $[S \rightarrow D \rightarrow G]$ with a cost of 10.
    b. Late goal test and reached table: Our sequence by construction will check that a state is the goal after the node has been popped from the frontier.
        i. [(S, 0)] starting point of the sequence which is popped trivially
        ii. [(D, 5), (B, 3), (A, 2)] pop (D, 5) move to (G, 10)
        iii. [(G,10), (B, 3), (A, 2)] pop (G, 10) perform goal test on (G,10)
        iv. Sequence Solution reached $[S \rightarrow D \rightarrow G]$ with a cost of 10.
    c. Late goal test and no reached table.
        i. [(S, 0)] starting point of the sequence which is popped trivially
        ii. [(D, 5), (B, 3), (A, 2)] pop (D, 5), but due to the fact that there is no *reached table*, we don't actively track the previously visited node, and due to the reverse-alphabetical order we are stuck in an infinite loop between *S* and *D* due to the bidirectional nature of the tree sequence.

2. **We use BFS (Breadth-First Search) along the undirected (bidirectional) edges.**
    a. Early goal test and reached table:
        i. [(S, 0)] starting point of the sequence which is popped trivially
        ii. [(D, 5), (B, 3), (A, 2)] pop (D, 5) and perform goal test on (G,10)
        iii. Sequence Solution reached $[S \rightarrow D \rightarrow G]$ with a cost of 10.
    b. Late goal test and reached table:
        i. [(S, 0)] starting point of the sequence which is popped trivially
        ii. [(D, 5), (B, 3), (A, 2)] pop (D, 5) move to (G, 10)
        iii. [(G,10), (B, 3), (A, 2)] pop (B, 3) move to (D, 7)
        iv. [(G, 10), (D, 7), (A, 2)] pop (A, 2) move to (C, 6)
        v. [(G, 10), (D, 7), (C, 6)] perform a goal test on (G, 10).
        vi. Sequence Solution reached $[S \rightarrow D \rightarrow G]$ with a cost of 10.
    c. Late goal test and no reached table.
        i. [(S, 0)] starting point of the sequence which is popped trivially
        ii. [(D, 5), (B, 3), (A, 2)] pop (D, 5) move to (G, 10)
        iii. [(G,10), (B, 3), (A, 2)] pop (B, 3) move to (D, 7)
        iv. [(G, 10), (D, 7), (A, 2)] pop (A, 2) move to (C, 6)
        v. [(G, 10), (D, 7), (C, 6)] perform a goal test on (G, 10).
        vi. Sequence Solution reached $[S \rightarrow D \rightarrow G]$ with a cost of 10.

3. **Suppose we run UCS with a reached table.**
    a. Early goal test
        i. [(S, 0)] starting point of the sequence which is popped trivially
        ii. [(A, 2), (B, 3), (D, 5)] pop (A, 2) move to (C, 6)
        iii. [(C, 6), (B, 3), (D, 5)] pop (B, 3) move to (D, 7)
        iv. [(C, 6), (D, 7), (D, 5)] pop (D, 5) move to (G, 10) perform Goal test
        v. Sequence Solution reached $[S \rightarrow D \rightarrow G]$ with a cost of 10
    b. Late goal test
        i. [(S, 0)] starting point of the sequence which is popped trivially
        ii. [(A, 2), (B, 3), (D, 5)] pop (A, 2) move to (C, 6)
        iii. [(C, 6), (B, 3), (D, 5)] pop (B, 3) move to (D, 7)
        iv. [(C, 6), (D, 7), (D, 5)] pop (D, 5) move to (G, 10)
        v. [(C, 6), (D, 7), (G, 10)] pop (C, 6) move to (G, 8)
        vi. [(G, 8), (D, 7), (G, 10)] pop (G, 8) and perform goal test on (G, 8)
        vii. Sequence Solution reached $[S \rightarrow A \rightarrow C \rightarrow G]$ with a cost of 8. This is our optimal solution, with the lowest cost.

4. **Repeating the above using A* search**
    a. Early goal test
        i. [(S, 0)] starting point of the sequence which is popped trivially
        ii. [(A, 4), (D, 6), (B, 8)] pop (A, 4) move to (C, 8)
        iii. [(C, 8), (D, 6), (B, 8)] pop (D, 6) move to (G, 10) perform a goal test.
        iv. Sequence Solution reached $[S \rightarrow D \rightarrow G]$ with a cost of 10.
    b. Late goal test
        i. [(S, 0)] starting point of the sequence which is popped trivially
        ii. [(A, 4), (D, 6), (B, 8)] pop (A, 4) move to (C, 8)
        iii. [(C, 8), (D, 6), (B, 8)] pop (D, 6) move to (G, 10)
        iv. [(C, 8), (G, 10), (B, 8)] pop (C, 8) move to (G, 8) due to tie with (B, 8)
        v. [(G, 8), (D, 8), (G, 10)] pop (G, 8) and perform goal test
        vi. Sequence Solution reached $[S \rightarrow A \rightarrow C \rightarrow G]$ with a cost of 8. This is our optimal solution, with the lowest cost.

5. We find the range of nonnegative heuristic values that would cause A* to produce a suboptimal solution for the following nodes:
    a. $h(a)$ **does not** impact the optimality of the solution as the search would still favor the optimal route outlined in 2.4.b given., given the influence of the realized route path versus the heuristic.
    b. $h(b)$ **does not** impact the optimality of the solution as the search would still favor the optimal route outlined in 2.4.b given., given the influence of the realized route path versus the heuristic.
    c. $h(c)$ **does** impact the optimality of the solution as if we have a large *h(c)* value, namely with a value greater than or equal to 4 we will compare the routes between (C, 10) and (G, 10) and due to the reverse alphabetical order will opt to pop (G, 10), evaluating the goal test and return a sub-optimal sequence of $[S \rightarrow D \rightarrow G]$
    d. $h(d)$ **does not** impact the optimality of the solution as the search would still favor the optimal route outlined in 2.4.b given., given the influence of the realized route path versus the heuristic.

Problem 3: Local Search (14 points)

We will use local search to solve the 4-queens problem. To simplify representation, a state will be represented by a set of 4 coordinate tuples, one for each queen. Following NumPy indexing, the top left cell has coordinates $(0, 0)$ and the bottom right cell has coordinates $(3, 3)$. So the example state shown below is $\{(0, 1), (1, 3), (2, 0), (3, 2)\}$.

1. If all queens are centered in the left column {(0, 0), (1, 0), (2, 0), (3, 0)}, then by the "min conflicts" approach we should have an *h* value of 6, whereby the only conflicts present are vertical to each queen's current position with the other queens. If we were to shift the queens horizontally, we encounter the same, if not a smaller *h* value. As a result, we take this arrangement to be a **global maxima**, rather than a local maxima since we receive no improvement from neighboring states.

2. The hill descent algorithm produces the following sequence, see state and corresponding *h* value transcribed below moving to the lowest neighboring *h* value:
    a. Initial State: {(0, 0), (1, 0), (2, 0), (3, 0)}, *h*=6
    b. Iteration 1: {(0, 1), (1, 3), (2, 0), (3, 0)}, *h=3*
    c. Iteration 2: {(0, 1), (1, 0), (2, 0), (3, 0)}, *h*=1
    d. Iteration 3: {(0, 1), (1, 3), (2, 0), (3, 2)}, *h*=0

    When considering the following {(0, 1), (1, 3), (2, 2), (3, 0)}, we note a heuristic value of 1 given the conflict along the diagonal (i.e. (2, 2) and (1, 3)). Since, we have found another state space yielding the same heuristic value, namely the sequence {(0, 1), (1, 0), (2, 0), (3, 0)} from our base configuration we assume this to be the "**flat**" feature of the state space. It is possible, as shown above, to arrive at an even lower *h* value, namely {(0, 1), (1, 3), (2, 0), (3, 2)}.

3. If we allow "sideways' moves to neighboring states with equal *h*-values our hill-descent algorithm now produces the following sequence:
    a. Initial State: {(0, 1), (1, 3), (2, 2), (3, 0), *h*=1
    b. Iteration 2: {(0, 1), (1, 3), (2, 0), (3, 0)}, *h*=1
    c. Iteration 3: {(0, 1), (1, 3), (2, 0), (3, 2)}, *h*=0

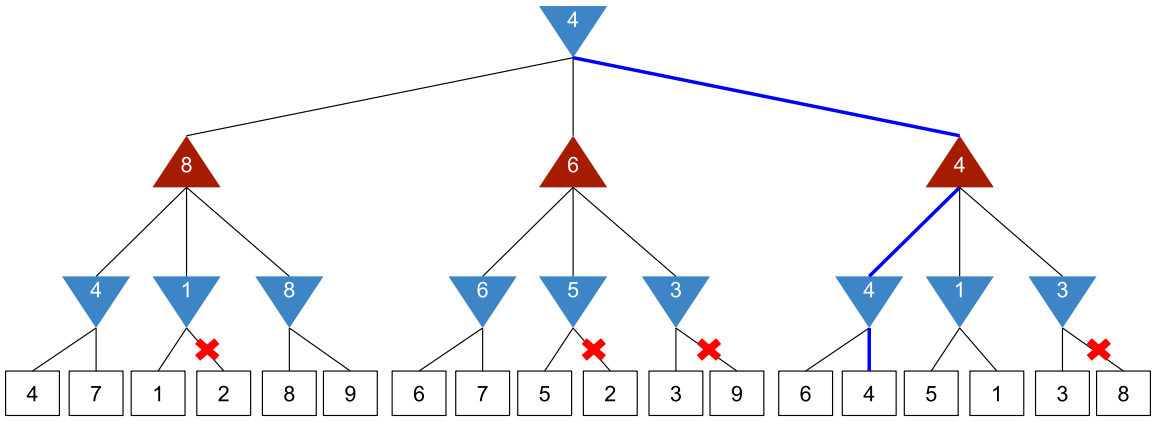
    It may still be possible to never find a solution to the above, given that by enabling sideways moves we run the risk of getting stuck on the flat local minimum (which isn't a shoulder) and forces an infinite loop between equally valued neighboring states. While a simple limit on the number of sideways moves helps avoid this infinite loop, we don't promise convergence to an optimal solution and thus may not change the path outcome.

Problem 1: Game Tree Search (24 points)

Two agents are playing a game using the following scoreboard. Player 1 (P1) first eliminates a column from the board. Player 2 (P2) next keeps a row from the current board. Finally, P1 selects one of the two values remaining as the game score. P1's objective is to minimize the score, while P2's objective is to maximize it.

6	4	7
5	1	2
3	8	9

MAX nodes (P2) are indicated with upward facing (red) triangles, MIN nodes (P1) are defined with downward facing (blue) triangles in the below diagram.



Leveraging an alpha-beta search on the above game to prune unnecessary routes, denoted by a red "x" on the path. We assign our variable space $\alpha = -\infty$, $\beta = \infty$ at inception.

1. Prune Terminal Node 2 (i.e. number 4), given that the deciding MAX node has an α value of 4, which is greater than the adjacent node which returns a value of 1.
   Our new variables are: $\alpha = 4, \beta = 1, v = 1$.
2. Prune Terminal Node 2 (i.e., number 10) given that the deciding MAX node has an α value of 6, which is greater than the adjacent node which returns a value of 5.
   Our new variables are: $\alpha = 6, \beta = 8, v = 5$
3. Prune Terminal Node 9 (i.e., number 12) given that the deciding MAX node has an α value of 6, which is greater than the adjacent node which returns a value of 3.
   Our new variables are: $\alpha = 6, \beta = 8, v = 3$
4. Prune Terminal Node 8 (i.e., number 18) given that the deciding MAX node has an α value of 4, which is greater than the adjacent node which returns a value of 1.
   Our new variables are: $\alpha = 4, \beta = 6, v = 3$

Given that P2 now plays randomly rather than deterministically, we must weight the minmax nodes according to the likelihood that a particular route is chosen. We then follow P1's deterministic play of trying to minimize the score. This then follows as such:

P1 Removes Row 1 $= 0.25(4) + 0.5(1) + 0.25(8) = 3.50$

P1 Removes Row 2 $= 0.25(6) + 0.5(5) + 0.25(3) = 3.25$

P1 Removes Row 3 $= 0.25(4) + 0.5(1) + 0.25(3) = 2.25$

We now find that our expected score is now 2.25 (as opposed to 4), though P1 still opts to remove Row 3. Upon further examination, we note that P1 would never change their initial strategy of removing column 3 from the game, given that the sum of its cell values yields the highest score value amongst other columns. Since P1's objective is to minimize the score, they will avoid columns where there is a large concentration of high values.

If we opt to use an evaluation function in place of a full search, we estimate the forward looking game score based on which column P1 chooses to eliminate. This then follows:

a. Evaluation Function: Minimum of Remaining Matrix Values
   If using a minimum of the remaining matrix values, we find that if we remove either column 1 or 3, our minimum is the same in both cases. If moving from left to right, we'd pursue a strategy which removes the first column and ultimately returns a sub-optimal value of 8 for P1, assuming P2 acts deterministically.
b. Evaluation Function: Average of Remaining Matrix Values
   If using an average of the remaining matrix values, we find that we still prefer to remove column 3 which boasts the maximum average of the remaining values. As such, we would continue to follow our optimal strategy, with a game score of 4.

Problem 2: MDPs and Dynamic Programming (24 points)

A mobile robot is moving around on a rechargeable battery. There are three battery level states: *high*, *low*, and *off*. In the first two states, the robot may move *fast* or *slow*, while in *off*, the robot may only *recharge*. Transitions are stochastic; moving *fast* is guaranteed to lower the robot's battery level, while moving *slow* may sometimes do so. A transition and reward function are defined for this robot as follows.

s	a	s'	$T(s,a,s')$	$R(s,a,s')$
high	*fast*	*low*	1.0	+3
high	*slow*	*high*	0.5	+2
high	*slow*	*low*	0.5	+2
low	*fast*	*off*	1.0	+2
low	*slow*	*low*	0.75	+2
low	*slow*	*off*	0.25	+1
off	*recharge*	*high*	1.0	−2

1. Considering the policy π in which the robot goes *fast* in both the high and low states, we have the following recursive relationships, where:

   $V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$

   $V^{\pi}(high) = 1 \cdot [3 + \gamma V^{\pi}(low)]$

   $V^{\pi}(low) = 1 \cdot [2 + \gamma V^{\pi}(off)]$

   $V^{\pi}(off) = 1 \cdot [-2 + \gamma V^{\pi}(high)]$

   Solving the above system of linear equations for $\gamma = 0.5$ yields the following — for simplicity we will use x, y, z to represent high, low and off, respectively.

   $x = 3 + 0.5y$

   $y = 2 + 0.5z$

   $z = (-2) + 0.5x$

   $x = 4,\ y = 2,\ z = 0$

   Python's numpy library was used to solve this linear system of equations, see code snippet:

   ```
   a = np.array([[1, -0.5, 0],[0, 1, -0.5], [-0.5, 0, 1]])
   b = np.array([3, 2, -2])
   print(np.linalg.solve(a,b))
   ```

2. If we assume that the values computed above are the time-limited values V_i we can compute the next iteration as follows: $V^{\pi}_{i+1}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}_{i}(s')]$ for each of the action states to determine the optimal policy π^* using our previously computed values
   a. $V^{\pi}_{i+1}(high) = max\{[3 + 0.5V^{\pi}_{i}(2)], 0.5[2 + 0.5V^{\pi}_{i}(4)] + 0.5[2 + 0.5V^{\pi}_{i}(2)]\}$
   b. $V^{\pi}_{i+1}(low) = max\{[2 + 0.5V^{\pi}_{i}(0)], 0.75[2 + 0.5V^{\pi}_{i}(2)] + 0.25[1 + 0.5V^{\pi}_{i}(0)]\}$
   c. $V^{\pi}_{i+1}(off) = max\{[-2 + 0.5V^{\pi}_{i}(4)]\}$

   Solving the above equations using our provided optimal values, yields the following values
   a. $V^{\pi}_{i+1}(high) = max\{4, 3.5\} = 4$
   b. $V^{\pi}_{i+1}(low) = max\{2.0, 2.5\} = 2.5$
   c. $V^{\pi}_{i+1}(off) = max\{0.0\} = 0$

3. Given the optimal values are given for the entire system state as $V^*(high) = 4.42$, $V^*(low) = 2.84$ and $V^*(off) = 0.21$, we simply compute another iteration in for our system of equations and back out the implied policy direction (argmax) that provides our solution:
   a. $V^{\pi}_{i+1}(high) = max\{[3 + 0.5V^{\pi}_{i}(low)], 0.5[2 + 0.5V^{\pi}_{i}(high)] + 0.5[2 + 0.5V^{\pi}_{i}(low)]\}$
   b. $V^{\pi}_{i+1}(low) = max\{[2 + 0.5V^{\pi}_{i}(off)], 0.75[2 + 0.5V^{\pi}_{i}(low)] + 0.25[1 + 0.5V^{\pi}_{i}(off)]\}$
   c. $V^{\pi}_{i+1}(off) = max\{[-2 + 0.5V^{\pi}_{i}(high)]\}$

   Solving the above equations using our provided optimal values, yields the following:
   d. $V^{\pi}_{i+1}(high) = max\{4.42, 3.815\}$
   e. $V^{\pi}_{i+1}(low) = max\{2.105, 2.84125\}$
   f. $V^{\pi}_{i+1}(off) = max\{0.21\}$

   As such, we have that our optimal policy (π^*) is {high:fast, low:slow, off:recharge}

4. If we were to change the prevailing gamma to 0, our optimal policy would be myopic and focus solely on the near term reward. As such, our optimal policy π would be to go *fast* in all states high/low and recharge during the off state. When comparing this to our optimal action in part 3, we note that while our recharge and high states don't change, we switch from a *slow* to a *high* action on account of the myopic nature. Since, we value short term gratification, there is no value in conserving movement to extend the life of the robot and avoid the recharge cost.

Problem 2: Reinforcement Learning (24 points)

The performance of the robot from HW2 has degraded, and the original model no longer appears to be valid. In particular, the *off* state is now a terminal state. Suppose we observe the following two episodes of state and reward sequences following the policy π of going *slow* in both states.

- Episode 1: $high, 2, high, 1, low, 1, low, 1, low, 0, off$
- Episode 2: $high, 2, high, 2, high, 1, low, 0, off$

1. Solving iteratively via first-visit Monte-Carlo we have the following state return values:
   Starting with episode 1 with a gamma (γ) value of 0.5 yields the following
   $V^{\pi}(high) \leftarrow G(high) = 2 + 0.5(1) + 0.5^{2}(1) + 0.5^{3}(1) = 2.875$
   $V^{\pi}(low) \leftarrow G(low) = 1 + 0.5(1) = 1.5$
   Continuing with episode 2 we leverage our previously computed values as follows:
   $\frac{1}{2}(V^{\pi}(high) + G(high)) = \frac{1}{2}([2 + 0.5(2) + 0.5^{2}(1)] + [2.875]) = 3.0625$
   $\frac{1}{2}(V^{\pi}(low) + G(low)) = \frac{1}{2}(1.5 + 0) = 0.75$
   Averaging the sequential sequence gives the following:
   $V^{\pi}(high) = 3.0625, V^{\pi}(low) = 0.75$

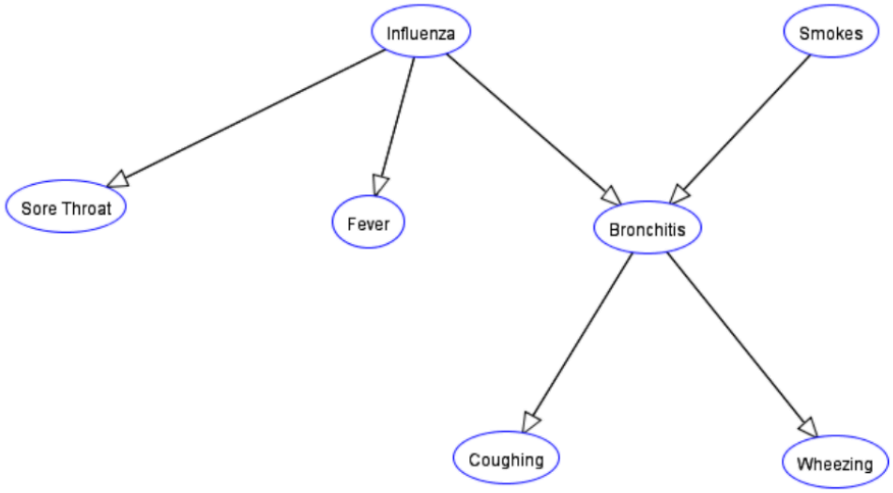
2. Considering alternative weights to each return episode we adjust our first-visit Monte Carlo approach and modify our G values from episode 1 with the 0.2 :
   $V^{\pi}(high) \leftarrow \alpha \cdot G(high) = 0.2 \cdot (2 + 0.5(1) + 0.5^{2}(1) + 0.5^{3}(1)) = 0.575$
   $V^{\pi}(low) \leftarrow \alpha \cdot G(low) = 0.2 \cdot (1 + 0.5(1)) = 0.3$
   Continuing with episode 2 we leverage our previously computed values as follows:
   $\frac{1}{2}(V^{\pi}(high) + \alpha \cdot G(high)) = (0.575 + 0.8 \cdot [2 + 0.5(2) + 0.5^{2}(1)]) = 3.175$
   $\frac{1}{2}(V^{\pi}(low) + \alpha \cdot G(low)) = (0.8 \cdot 0 + 0.3) = 0.3$
   Since our alpha values dictate the amount of updating we perform through each iteration, our choice of alphas between episode 1 & 2 segment our training preference to weight recent events higher than previous events. Better way to compute values in a nonstationary problem.

3. Performing the Q-learning algorithm with an initial *high* state, yields the following relationship $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s'a') - Q(s,a))$, where we first start off in the *high* state and take the greedy action (i.e., highest Q-value action) which lands us in the *low* state where we then take the exploratory action (i.e. the lowest Q-value action). However, by Q-learning, we will act on the target policy, which implies we take the max operation. The Q-values so far: Q(high, f ast) = 2, Q(high, slow) = 0, Q(low, f ast) = −1, Q(low, slow) = 1.
$Q(high, fast) \leftarrow Q(high, fast) + 0.8(2 + 0.5Q(low, slow) - Q(high, fast))$
$Q(high, fast) \leftarrow 2 + 0.8(2 + 0.5(1) - 2) = 2.4$
We take the same case structure when engaging in the second exploratory action, i.e. the state with a lower Q-value (sub-optimal) pathing.
$Q(low, fast) \leftarrow Q(low, fast) + 0.8(2 + 0.5Q(low, slow) - Q(low, fast))$
$Q(low, fast) \leftarrow -1 + 0.8(1 + 0.5(-1) - (-1)) = 1$

4. Using SARSA as opposed to Q-learning changes our expected Q-value updates, as we no longer maximize our target policy, but rather we follow our behavioral policy which corresponds to the action we've chosen for our action sequence. As such, instead of our second transition of Q(low, slow) we will look to explore with action Q(low, high) which yields a lower Q-value.
$Q(high, fast) \leftarrow Q(high, fast) + 0.8(2 + 0.5Q(low, fast) - Q(high, fast))$
$Q(high, fast) \leftarrow 2 + 0.8(2 + 0.5(-1) - 2) = 1.6$

Problem 2: Diagnostic Bayes Net (28 points)

The following Bayes net is the "Simple Diagnostic Example" from the Sample Problems of the Belief and Decision Networks tool on AIspace. All variables are binary.



1. Given that we are interested in computing the fever distribution given a positive wheezing symptom we can leverage the Bayes formula to switch the conditioning on the pathing of the network. We will simplify our binary True/False as +/- respectively and define our observed states as fever (F), wheezing (w) as well as hidden states influenza (i), bronchitis (b) and smokes (s) conditioned as follows:

   $$P(F|w) \propto P(F, i, b, s, w) = \sum_{i,b,s} P(F|i)P(w|b)P(b|i,s)P(i)P(s) = \sum_{i,b,s} f_I(i)f_F(F,i)f_S(s)f_B(b,i,s)f_w(b)$$

   Given the above analytical expression, our intermediate factor (assuming marginalization is done at the end) should be of size 16, i.e. 2^4 to reflect the joint combinations of i, b, s, F.
   Set cardinality product i.e. size of each list item and it's product grouping
2. We employ variable elimination with the following sequence below by splitting up the analytical expression and returning the maximal number of terms form each summation:
   a. **influenza**, **smokes**, sore throat, fever, **bronchitis**, coughing, wheezing

      $$\sum_b P(+\ w|b) \sum_s P(s) \sum_i P(b|i,s)P(F|i)P(i)$$

      Largest intermediate factor is 16, after considering the variable elimination.
   b. wheezing, coughing, **bronchitis**, fever, sore throat, **smokes**, **influenza**

      $$\sum_i P(F|i)P(i) \sum_s P(s) \sum_b P(b|i,s)P(+\ w|b)$$

      Largest intermediate factor is 8 after considering the variable elimination

```
prob_wGb = pd.DataFrame({'Bronchitis': ['T', 'F'], 'Proba': [0.6, 0.001]})
prob_bGis = pd.DataFrame({'Influenza': ['T', 'T', 'F', 'F', 'T', 'T', 'F', 'F'],
                          'Smokes': ['T', 'F', 'T', 'F', 'T', 'F', 'T', 'F'],
                          'Bronchitis': ['T', 'T', 'T', 'T', 'F', 'F', 'F', 'F'],
                          'Proba': [0.99, 0.9, 0.7, 1e-4, 0.01, 0.1, 0.3, 0.9999]})
prob_i = pd.DataFrame({'Influenza': ['T', 'F'], 'Proba': [0.05, 0.95]})
prob_s = pd.DataFrame({'Smokes': ['T', 'F'], 'Proba': [0.2, 0.8]})
prob_fGi = pd.DataFrame({'Influenza': ['T', 'T', 'F', 'F'], 'Fever': ['T', 'F', 'T', 'F'], 'Proba': [0.9, 0.1, 0.05, 0.95]})
```

```
# First intermediate product given merge
X1 = pd.merge(left=prob_bGis, right=prob_wGb, on='Bronchitis')
X1['Proba'] = X1['Proba_x'] * X1['Proba_y']
X1 = X1.groupby(['Influenza', 'Smokes'])['Proba'].sum().reset_index()
```

```
X1
```

	Influenza	Smokes	Proba
0	F	F	0.00106
1	F	T	0.42030
2	T	F	0.54010
3	T	T	0.59401

```
# Second intermediate product given merge
X2 = pd.merge(left=X1, right=prob_s, on='Smokes')
X2['Proba'] = X2['Proba_x'] * X2['Proba_y']
X2 = X2.groupby(['Influenza'])['Proba'].sum().reset_index()
```

```
X2
```

	Influenza	Proba
0	F	0.084908
1	T	0.550882

```
# Third intermediate product given merge
X3 = pd.merge(left=X2, right=prob_i, on='Influenza')
X3 ['Proba'] = X3 ['Proba_x'] * X3 ['Proba_y']
X3 = X3.groupby(['Influenza'])['Proba'].sum().reset_index()
```

```
X3
```

	Influenza	Proba
0	F	0.080663
1	T	0.027544

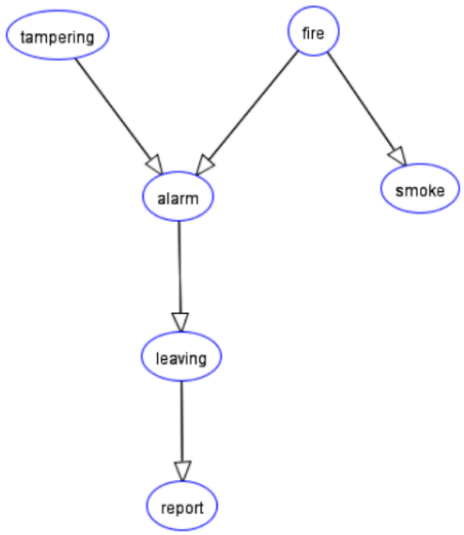
```
# Final intermediate product given merge
X4 = pd.merge(left=X3, right=prob_fGi, on='Influenza')
X4 ['Proba'] = X4 ['Proba_x'] * X4 ['Proba_y']
X4 = X4.groupby(['Fever'])['Proba'].sum().reset_index()
X4 = X4.set_index('Fever')
```

```
X4 / X4.sum()
```

Fever	Proba
F	0.733632
T	0.266368

Problem 1: Fire Alarm Bayes Net (16 points)

The following Bayes net is the "Fire Alarm Belief Network" from the Sample Problems of the Belief and Decision Networks tool on AIspace. All variables are binary.



1. Leveraging likelihood weighting to construct a posterior distribution with the following evidence:
    a. Tampering and Fire: Given that our conditioned evidence is *upstream* with both tampering and fire serving as root nodes as representing the prior distribution. Given this, we note that our posterior distribution is similar to these priors, and will provide a reasonable estimate for the posterior.
    b. Smoke and Report: As our conditioning is now *downstream* with both smoke and report serving as terminal nodes, our generated sample evidence will now reflect the posterior distribution. However, given this focus on later states in the Bayes net, we note that only a few samples will dominate our estimates for the posterior distribution, leaving many samples as irrelevant to the query and potentially add greater variance to our estimates.
2. After observing the CPTs we can reconcile a list of evidence which won't occur. For breadth we will be using t, f, a, s, l, r to represent out labeled nodes which follow:
    a. $\{-\ t, -\ f, +\ a\}$ i.e. can't have a positive alarm if tampering and fire are false
    b. $\{-\ a, +\ l\}$ i.e. if the alarm is false we can't have a positive leaving state – note this includes all preceding states prior to an alarm (tampering, fire)

    If we were to use these events as observed in rejection sampling we would run the risk of rejecting all other samples that don't match our provided evidence value, which in turn will result in no posterior distribution. Having said that, if we were to use likelihood estimation, while we won't reject our items given that the events aren't feasible we run the risk of broadcasting this zero likelihood through our posterior distribution.
3. By performing Gibbs sampling with the goal of resampling the Alarm (A) variable conditioned on the current sample defined as $(+\ t, -\ f, -\ s, -\ l, -\ r)$, we can reduce the evidence space to

the Markov Blanket which includes the parents and children node of Alarm. We can then define an analytical expression as follows for the Bayes net CPTs using the applet parameters:

$P(A|\ +t, -f, -s, -l, -r) \rightarrow P(A|mb(+t, -f, -l)) \propto P(A, +t, -f, -l)$

$P(A, +t, -f, -l) = P(+t)P(-f)P(A|\ +t, -f)P(-l|A)$

$P(A, +t, -f, -l) = 0.02 \cdot 0.99 \cdot [0.85,\ 0.15] \cdot [0.12,\ 1.0]$

Solving the above then produces the following distribution for Alarm after normalizing:

	$P(A\|\ +t, -f, -l)$
True	0.40476
False	0.59524

4. If we lose the value of Leaving i.e. $(-\ l)$ from before, we must modify our analytically expression slightly to consider both values of Leaving (i.e. True and False) which follows:

$P(A|\ +t, -f, -s, -r) \rightarrow P(A|\ +t, -f, -r) \propto P(A, +t, -f, -r)$

$P(A, +t, -f, -r) = P(+t)P(-f)P(A|\ +t, -f)P(L|A)P(-r|L)$

$P(A, +t, -f, -r) = 0.02 \cdot 0.99 \cdot [0.85,\ 0.15] \cdot [[0.88,\ 0.12], [0,\ 1.0]] \cdot [0.25,\ 0.99]$

Solving the above produces the following distribution for Alarm after normalizing:

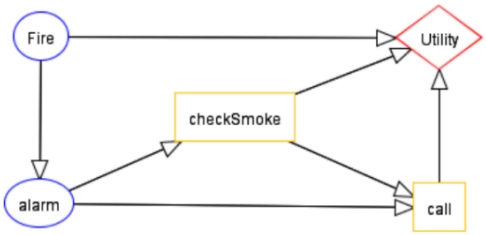
	$P(A\|\ +t, -f, -s, -r)$
True	0.65978
False	0.34022

However, we note that by conditioning on less information we increase our computation for the conditional probability, given that we have added another floating random variable L, which differs from our previous instance where all evidence values were known with full information. In doing so, we must consider both True and False values for the *Learning* node.

Problem 2: Fire Alarm Decision Network (16 points)

The following Bayes net is a simplified version of the "Fire Alarm Decision Problem" from the Sample Problems of the Belief and Decision Networks tool on AIspace. All variables are binary, and the chance node CPTs as well as utility values are shown below.

While you are not required to do so, we recommend that you use `pandas` functionality to carry out the operations in this problem. It is also acceptable if you prefer to perform all calculations manually. Please include your work in either case.



Fire	CheckSmoke	Call	Utility
True	True	True	−220
True	True	False	−5020
True	False	True	−200
True	False	False	−5000
False	True	True	−220
False	True	False	−20
False	False	True	−200
False	False	False	0

Fire	Pr(Fire)
T	0.01
F	0.99

Fire	Alarm	Pr(Alarm | Fire)
True	True	0.95
True	False	0.05
False	True	0.01
False	False	0.99

1. We compute the joint factor required to remove the *Fire* chance nodes, as it doesn't serve as a parent to any of the decision nodes present (i.e., *CheckSmoke*, *Call*) using Python below:

```
# Merge probability values for the Fire and Alarm - if Alarm True then checkSmoke is True
X1 = pd.merge(left=prob_aGf, right=prob_f, on=['Fire'])
X1['proba'] = X1['proba_x'] * X1['proba_y']
X1 = X1[['Fire', 'Alarm', 'proba']]
```

```
# Sum out the Fire chance node, is it isn't a parent of any decision nodes (i.e. checkSmoke, call)
X2 = pd.merge(left=X1, right=utility_matrix, on=['Fire'])
X2['weighted_utility'] = X2['Utility'] * X2['proba']
X2 = X2[['Alarm', 'Fire', 'checkSmoke', 'Call', 'weighted_utility']]
```

```
X2.groupby(['Alarm', 'checkSmoke', 'Call'])[['weighted_utility']].sum().reset_index()
```

	Alarm	checkSmoke	Call	weighted_utility
0	F	F	F	-2.500
1	F	F	T	-196.120
2	F	T	F	-22.112
3	F	T	T	-215.732
4	T	F	F	-47.500
5	T	F	T	-3.880
6	T	T	F	-47.888
7	T	T	T	-4.268

2. We then determine the optimal decision function and expected utilities for *Call* by grouping by our residual *Alarm* and *CheckSmoke* fields and returning the maximum utility.

```
# Retrieve the optimal policy and corresponding expected utilities for Call
joint_factor.loc[joint_factor.groupby(['Alarm', 'checkSmoke'])[['weighted_utility']].idxmax().values.flatten()]
```

	Alarm	checkSmoke	Call	weighted_utility
0	F	F	F	-2.500
2	F	T	F	-22.112
5	T	F	T	-3.880
7	T	T	T	-4.268

3. Since *Call* depends on the combination of the decision outcomes, where no policy has been specified, we do require both of Call's parent nodes to determine our optimal decision function. Having said that, if we were to determine the optimal decision function for CheckSmoke first, followed by Call, we would reduce the number of optimal combinations whilst preserving the most optimal decision values for *Call* i.e. *+Alarm -CheckSmoke* and *+Alarm -CheckSmoke*.
4. Performing a similar exercise for CheckSmoke, we note that there are overlaps with the Call node for optimal decisions, though we screen over only Alarm given its direct influence on our CheckSmoke node. This produces a maximum expected utility (MEU) of -6.38.

```
display(joint_factor.loc[joint_factor.groupby(['Alarm'])[['weighted_utility']].idxmax().values.flatten()])
meu = joint_factor.loc[joint_factor.groupby(['Alarm'])[['weighted_utility']].idxmax().values.flatten()]['weighted_utility'].sum()
print(f'MEU:{meu}')
```

	Alarm	checkSmoke	Call	weighted_utility
0	F	F	F	-2.50
5	T	F	T	-3.88

```
MEU:-6.38
```

We will be working with the following data set for Problems 3 and 4. There are two trinary features x_1 and x_2, and a class variable y with values 0 and 1.

Sample	x_1	x_2	y
1	$+1$	-1	0
2	$+1$	$+1$	0
3	0	-1	0
4	0	$+1$	0
5	$+1$	$+1$	1
6	0	0	1
7	0	$+1$	1
8	-1	$+1$	1

Naive-Bayes

1. In estimating the conditional probability tables (CPTs) given our sample data, we simply perform an approximate sum count based on the prevailing states and adjust our final probabilities by our initially provided maximum likelihood class for $Pr(Y)$. For generalization we will perform one example, i.e. $X_1 = 0$ given $Y = 0$, there are 2 instance where $X_1 = 0$ when $Y = 0$, for which there are 4. As a result, we have that our frequency probability $Pr(X_1 = 0|Y) = 2/4 = 0.5$

   Approximating the CPT for $Pr(X_1|Y)$

	$Pr(X_1 =- 1\|Y)$	$Pr(X_1 = 0\|Y)$	$Pr(X_1 =+ 1\|Y)$
$Y = 0$	0.00%	50.00%	50.00%
$Y = 1$	25.00%	50.00%	25.00%

   Approximating the CPT for $Pr(X_2|Y)$

	$Pr(X_2 =- 1\|Y)$	$Pr(X_2 = 0\|Y)$	$Pr(X_2 =+ 1\|Y)$
$Y = 0$	50.00%	0.00%	50.00%
$Y = 1$	0.00%	25.00%	75.00%

2. When observing our x_1 marginal distribution, it is apparent for combinations with no positive probability e.g., $Pr(X_1 =- 1|Y = 0) = 0$ will always predict 1 regardless of the value of x_2, our other feature. This is the same for cases where x_2 combinations yield a probability of zero, namely $Pr(X_2 = 0|Y = 0)$ and $Pr(X_2 =- 1|Y = 1)$. As a result, given these overlaps, when $x_2 =- 1$ and $x_1 =- 1$ our prediction is not well defined given that our x_1 feature points to a y value of 1, while our x_2 feature points to a y value of 0, which is contradicting.

1. Leveraging the estimated marginal conditional probabilities in Part 1 as a starting point for expectation-maximization we compute the expected counts for an updated $Pr(Y|X_1X_2)$ for each sample combination in the data set. This will follow first through Bayes theorem, which conditions our expression to the following:

   $Pr(Y|X_1X_2) \propto P(Y)P(X_1X_2|Y) \propto P(Y)P(X_1|Y)P(X_2|Y)$

   Where we then scale according to the outstanding conditions for $Y = \{0, 1\}$ according to the selection criteria for (x_1, x_2) combinations:

Sample	X_1	X_2	Y	$Pr(Y\|X_1X_2)$
1	1	-1	0	1
2	1	1	0	0.571429
3	0	-1	0	1
4	0	1	0	0.4
5	1	1	1	0.428571
6	0	0	1	1
7	0	1	1	0.6
8	1	1	1	1

```
prob_x1y = pd.DataFrame({'X1': [-1, 0, 1, -1, 0, 1], 'Y': [0, 0, 0, 1, 1, 1], 'proba': [0, 0.5, 0.5, 0.25, 0.5, 0.25]})
prob_x2y = pd.DataFrame({'X2': [-1, 0, 1, -1, 0, 1], 'Y': [0, 0, 0, 1, 1, 1], 'proba': [0.5, 0, 0.5, 0, 0.25, 0.75]})
```

```
samples = [(1, -1 , 0), (1, 1, 0), (0, -1, 0), (0, 1, 0), (1, 1, 1), (0, 0, 1), (0, 1, 1), (-1, 1, 1),]
```

```
tracker = []

for i in samples:
    x1, x2, x = i

    # compute both the Y=0, Y=1 vector for each condtional probability
    t1 = prob_x1y[(prob_x1y['X1'] == x1) & (prob_x1y['Y'] == 0)]['proba'].values * prob_x2y[(prob_x2y['X2'] == x2) & (prob_x2y['Y'] == 0)]['proba'].values
    t2 = prob_x1y[(prob_x1y['X1'] == x1) & (prob_x1y['Y'] == 1)]['proba'].values * prob_x2y[(prob_x2y['X2'] == x2) & (prob_x2y['Y'] == 1)]['proba'].values

    if x == 0:
        tracker.append((t1/(t2+t1))[0])
    else:
        tracker.append((t2/(t2+t1))[0])
```

```
expected_counts = pd.DataFrame({'Pr(X1X2|Y)': tracker})
expected_counts = expected_counts.join(pd.DataFrame(samples, columns=['X1', 'X2', 'Y']))
```

Perceptron Rule

```
# Drop samples 4 and 5 from the original 8 sample list
new_sample = pd.DataFrame(samples, columns=['X1', 'X2', 'Y'], index=np.arange(1, 9)).drop(index=[4, 5])
```

```
def activation_function1(val:float):
    "Denote the treatment given a f_w value"
    if val <= 0:
        return 0
    else:
        return 1
```

```
weightV = np.array([1, -2, 0])
alpha = 1

for w in new_sample.index:
    _x1, _x2, _y0 = new_sample.loc[w].values
    y_pred = activation_function1(weightV[0] + weightV[1]*_x1 + weightV[2]*_x2)

    if (_y0-y_pred) == 0:
        print(f'Sampling {(_x1, _x2)} No weight updated occured')
    else:
        old_weight = weightV
        weightV += alpha*(_y0-y_pred)*np.array([1, _x1, _x2])
        print(f'Sampling {(_x1, _x2)} Weight update occured')
        print(f'\t{old_weight} -> {weightV}')
```

```
Sampling (1, -1) No weight updated occured
Sampling (1, 1) No weight updated occured
Sampling (0, -1) Weight update occured
        [ 0 -2  1] -> [ 0 -2  1]
Sampling (0, 0) Weight update occured
        [ 1 -2  1] -> [ 1 -2  1]
Sampling (0, 1) No weight updated occured
Sampling (-1, 1) No weight updated occured
```