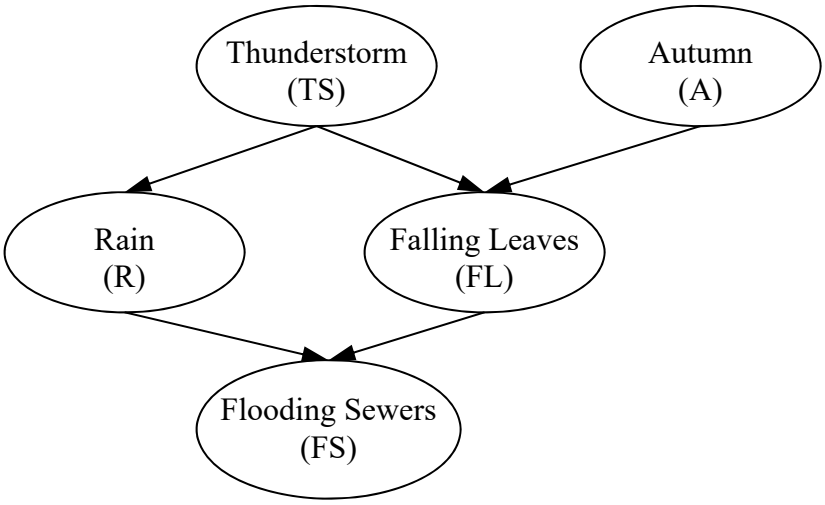
Assignment #4
Artificial Intelligence - CSCE 523
Due: 8:00 AM, Wednesday March 11, 2020
Uncertainty and Machine Learning

1. (10 points) Bayes Rule: Seventy percent of the aircraft that disappear in flight through the Bermuda Triangle are recovered (P(r) = 0.7). Sixty percent of the recovered aircraft have an emergency locator (P(e|r) = 0.60). Unfortunately, 90% of the aircraft not recovered do not have such a locator. Suppose that an aircraft with a locator has disappeared. What is the probability that it will be recovered (P(r|e))?

2. (20 points) Shown below is a Bayes network representing the risk of flooding sewers (**FS**) in a city as dependent on rainfall (**R**), falling leaves (**FL**), thunderstorm (**TS**), and autumn (**A**). Use the conditional probabilities below to determine the conditional probabilities of a thunderstorm for the observable scenarios **FS** $\wedge$ **A**, **FS** $\wedge$ $\neg$**A**, $\neg$**FS** $\wedge$ **A**, and $\neg$**FS** $\wedge$ $\neg$**A**.



FL: **P**(**FL** | **TS** $\wedge$ **A**) = 0.8
P(**FL** | $\neg$**TS** $\wedge$ **A**) = 0.2
P(**FL** | **TS** $\wedge$ $\neg$**A**) = 0.05
P(**FL** | $\neg$**TS** $\wedge$ $\neg$**A**) = 0.01

R: **P**(**R** | **TS**) = 0.7
P(**R** | $\neg$**TS**) = 0.1

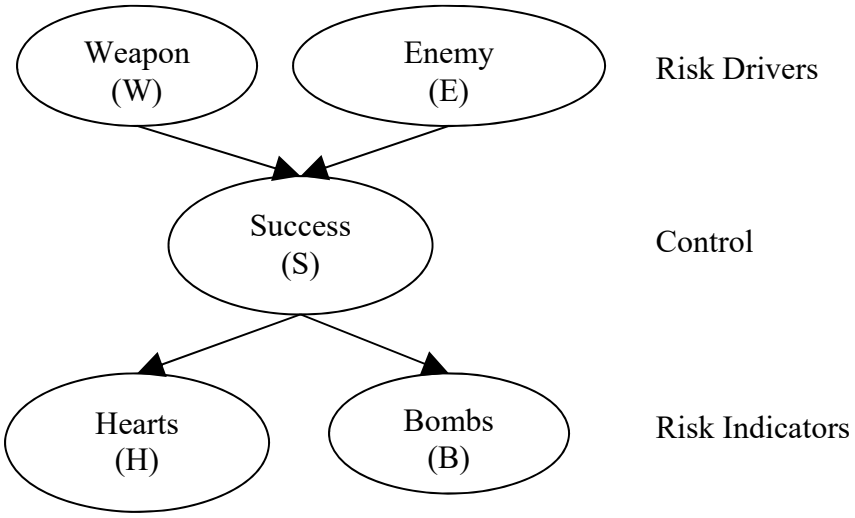
FS: **P**(**FS** | **FL** $\wedge$ **R**) = 0.5
P(**FS** | $\neg$**FL** $\wedge$ **R**) = 0.2
P(**FS** | **FL** $\wedge$ $\neg$**R**) = 0.05
P(**FS** | $\neg$**FL** $\wedge$ $\neg$**R**) = 0.01

TS: **P**(**TS**) **=** 0.5

A: **P**(**A**) = 0.33

3. (20 points) Link sees Sheik on the horizon. Sheik is fighting with magic and has all her hearts. Link wants to determine Sheik's potential for defeating the enemy and whether he should enter the fray.

   Shown below is a risk analysis Bayesian network that Link plans to use. His risk drivers are the type of weapon in use and the enemy faced. Using certain weapons on specific enemies can improve effectiveness in battle. The effectiveness affects the number of hearts the individual has, as well as the number of bombs they may have left. His question is what is the probability of success given magic and hearts: Pr(**S**|**W**=magic,**H**=true)?



W: Pr(**W**={magic,sword}) = <0.5,0.5>
E: Pr(**E**={poe,darknut,lizalfos}) = <0.33,0.33,0.33>
S:

Success	**W=**magic			**W=**sword		
	E=p	**E**=d	**E**=l	**E**=p	**E**=d	**E**=l
true	0.2	0.8	0.3	0.8	0.2	0.2
false	0.8	0.2	0.7	0.2	0.8	0.8

H:

Hearts	**S=**true	**S=**false
true	0.9	0.2
false	0.1	0.8

B:

Bombs	**S=**true	**S=**false
true	0.8	0.2
false	0.2	0.8

4. (20 points) For this problem, you need to build a Bayes network in problem 2 in JavaBayes. Using the Bayes network, double check your solutions for problem 2. And change the table for Rain to:
   R: **P**(**R** | **TS**) = 0.9
   P(**R** | ¬**TS**) = 0.3
   What are the conditional probabilities of thunderstorm (**TS**) given the observable scenarios **FL**, and ¬**FL**. Turn-in your Bayes network file with your assignment.

5. (30 points) Ahhh, life at AFIT- it's not just a job, it's an AI problem! You wake up to the sunrise after studying for a final all night. You find yourself amidst hundreds of cargo containers; your watch reads seven forty-five. Uh oh, you only have fifteen minutes to get to your exam. Unfortunately, you still must negotiate a maze of cargo between here and the classroom. Which brings us to the problem: how many steps does it take to reach your destination?

   There are two challenges for you:

   Part I:
   This challenge requires you to calculate V(s) for a given map, and output the MEU path. The calculation of the MEU should be conducted by performing value or policy iteration (your choice).

   This is a gridworld in which a cell is either occupied or empty, and the agent may move, North, South, East, or West, 1 square at a time. The cost of moving is 1.0. When the agent moves, there is a probability of 0.90 that they will end in the state that they want to be in and a probability of 0.07 that they remain in the current state, and a probability of 0.03 that they go backwards a square (i.e. if they were headed West, they would instead go 1 square East). The world is fully-observable, the agent knows where its location and the locations of the goal and obstacles.

   Part II:

For this challenge, solve the same problem using reinforcement learning. Use TD(0)/SARSA, and TD(λ). Solve the problem for V(s) not Q(s,a). The values should converge to those close to Part I.

Turn-in should include your code (no language stipulation), and a write-up which draws comparisons between the solutions. Include a results graph indicating the path length vs iteration (plot based on the same start location, and curves should go down). Testing should use world sizes of 25x25 and 50x50.

DataFile Format Example:
3 3
O O O
G V V
V V V

World x size, World y size
Each map location either O – obstacle, V – vacant, or G – goal

Have the agent start location be manually inserted or randomly generated at the users request.

Take a look at using the BURLAP (http://burlap.cs.brown.edu/) library as a starting point. You will need to install maven and hook it into your IDE.

Turn-ins: a write-up of your solution, with a graph comparing the convergence between approaches. Be sure to discuss your parameter settings, and how you identified the values you used (a parameter sweep is not unwarranted).

Reinforcement Learning Review
For this problem, you need to learn the best state values for each state in the domain. In order to learn these functions, you will need to make use of TD-λ. In order for the weights to be learned you will need to run your program several times.

To learn the state values using TD-λ you will need to consider the problem as a set of state positions s_0, s_1, …, s_T. The reward for the goal position s_T can be your choice, I suggest 100.

The theoretical/forward view of TD(λ) evaluates the target values for the states s_0, s_1, …, s_{T-1} as

$$V^{target}(s_t) = (1-\lambda)\sum_{k=1}^{T-t-1}\lambda^{k-1}V(s_{t+k}) + \lambda^{T-t-1}V(s_t)$$

Where t is the index of the state, and λ is a value between 0.0 and 1.0.

The issue with applying this representation is that it is a forward view and needs to be converted to an implementable backward view. The way this is handled is by adding the concept of eligibility traces, $e(s)$, (per Sutton) which maintain the reward decay. The resulting algorithm is shown in Figure 2.

Initialize $V(s)$ arbitrarily and $e(s) = 0, \forall s \in S$
Repeat (for each game)
  Initialize s and $t = 0$
  Repeat (for each step of game)
    Take action a_t, observe reward, r_{t+1}, and next state, s_{t+1}
    $\delta = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$
    $e(s_t) = e(s_t) + 1$
    For all s
      $V^{target}(s) = V(s) + \alpha\delta e(s)$
      $e(s) = \gamma\lambda e(s)$
    $t = t + 1$
  until s_t is terminal

Figure 2: Backward view of TD(λ).

Note that in the backward view of TD(λ) that all of the states and all of the eligibility trace state values are updated every single step of the learning iteration. In the interest of ease of implementation, we will assume that the only eligibility trace states we must track are those states that we visit during the course of an iteration (i.e. start to goal) and that all other eligibility trace state values are 0. This means that we must only track and update the states for each game. The reason that this assumption works is that in most cases, the decay values cause $e(s)$ to decrease at a high enough rate that repeat visits to the state within a short time frame are unlikely.

Java Code Stub Reading a maze file:

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

// This section merely parses the input file into an
array.
```

```
      char x[] = new char[1]; //temp variable used for
file parsing

      // open maze file
      // one can change the maze file here (make sure
they are in the same directory)
      File myFile = new File("maze1");
      FileReader fileReader = new FileReader(myFile);
      BufferedReader reader = new
BufferedReader(fileReader);
      String line = null;

      line = reader.readLine(); // read first line

      String[] valueStr = new
String(line).trim().split("\\s+"); // split into two
integers

      int length = Integer.parseInt(valueStr[0]);
      int width = Integer.parseInt(valueStr[1]);

      newMaze = new String[length][width]; // create
maze array

      //System.out.println(length + " " + width);

      //Parse rest of file into an array
      for (int i = 0; i < length; i++) {
       for (int j = 0; j < length; j++) {
            reader.read(x);
            newMaze[i][j] = new String(x);
       }
       reader.read(); // carriage return
      }
      reader.close();
```