Assignment 3

Posterior Proabilites and Bayesian Networks

Max possible score:

- 4308: 100 Points
- 5360: 100 Points

Task 1

50 points

The task in this part is to implement a system that:

- Can determine the posterior probability of different hypotheses, given priors for these hypotheses, and given a sequence of observations.
- Can determine the probability that the next observation will be of a specific type, priors for different hypotheses, and given a sequence of observations.

As in the slides provided here, there are five types of bags of candies. Each bag has an infinite amount of candies. We have one of those bags, and we are picking candies out of it. We don't know what type of bag we have, so we want to figure out the probability of each type based on the candies that we have picked.

The five possible hypotheses for our bag are:

- h_1 (prior: 10%): This type of bag contains 100% cherry candies.
- h_2 (prior: 20%): This type of bag contains 75% cherry candies and 25% lime candies.
- h_3 (prior: 40%): This type of bag contains 50% cherry candies and 50% lime candies.
- h_4 (prior: 20%): This type of bag contains 25% cherry candies and 75% lime candies.
- h_5 (prior: 10%): This type of bag contains 100% lime candies.

Command Line arguments:

The program takes a single command line argument, which is a string, for example CLLCCCLLL. This string represents a sequence of observations, i.e., a sequence of candies that we have already picked. Each character is C if we picked a cherry candy, and L if we picked a lime candy. Assuming that characters in the string are numbered starting with 1, the i-th character of the string corresponds to the i-th observation. The program should be invoked from the commandline as follows:

```
compute_a_posteriori observations
```

For example:

```
compute_a_posteriori CLLCCLLLCCL
```

We also allow the case of not having a command line argument at all, this represents the case where we have made no observations yet.

Output:

Your program should create a text file called "result.txt", that is formatted exactly as shown below. ??? is used where your program should print values that depend on its command line argument. At least five decimal points should appear for any floating point number.

```
Observation sequence Q: ???
Length of Q: ???

After Observation ??? = ???: (This and all remaining lines are repeated for every observation)

P(h1 | Q) = ???
P(h2 | Q) = ???
P(h3 | Q) = ???
P(h4 | Q) = ???
P(h5 | Q) = ???

Probability that the next candy we pick will be C, given Q: ???
Probability that the next candy we pick will be L, given Q: ???
```

Task 2

50 points

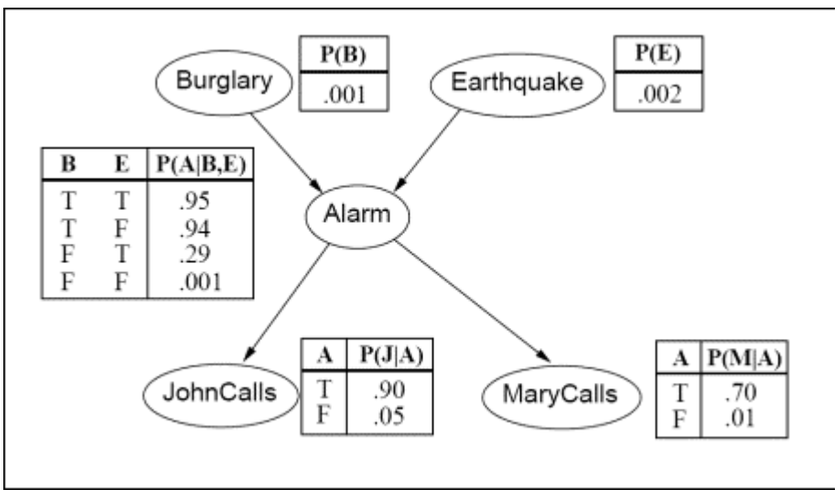

Figure 1: A Bayesian network establishing relations between events on the burglary-earthquake-alarm domain, together with complete specifications of all probability distributions.

For the Bayesian network of Figure 1, implement a program that computes and prints out the probability of any combination of events given any other combination of events. If the executable is called bnet, here are some example invocations of the program:

1. To print out the probability P(Burglary=true and Alarm=false | MaryCalls=false).

   ```
   bnet Bt Af given Mf
   ```

2. To print out the probability P(Alarm=false and Earthquake=true).

   ```
   bnet Af Et
   ```

3. To print out the probability P(JohnCalls=true and Alarm=false | Burglary=true and Earthquake=false).

   ```
   bnet Jt Af given Bt Ef
   ```

4. To print out the probability P(Burglary=true and Alarm=false and MaryCalls=false and JohnCalls=true and Earthquake=true).

   ```
   bnet Bt Af Mf Jt Et
   ```

In general, bnet takes 1 to 6(no more, no fewer) command line arguments, as follows:

- First, there are one to five arguments, each argument specifying a variable among Burglary, Earthquake, Alarm, JohnCalls, and MaryCalls and a value equal to true or false. Each of these arguments is a string with two letters. The first letter is B (for Burglary), E (for Earthquake), A (for Alarm), J (for JohnCalls) or M (for MaryCalls). The second letter is t (for true) or f (for false). These arguments specify a combination C1 of events whose probability we want to compute. For example, in the first example above, C1 = (Burglary=true and Alarm=false), and in the second example above C1 = (Alarm=false and Earthquake=true).
- Then, optionally, the word "given" follows, followed by one to four arguments. Each of these one to four arguments is again a string with two letters, where, as before the first letter is B (for Burglary), E (for Earthquake), A (for Alarm), J (for JohnCalls) or M (for MaryCalls). The second letter is t (for true) or f (for false). These last arguments specify a combination of events C2 such that we need to compute the probability of C1 given C2. For example, in the first example above C2 = (MaryCalls=false), and in the second example there is no C2, so we simply compute the probability of C1, i.e., P(Alarm=false and Earthquake=true).

The implementation should not contain hardcoded values for all combinations of arguments. Instead, your code should use the tables shown on Figure 1 and the appropriate formulas to evaluate the probability of the specified event. It is OK to hardcode values from the tables on Figure 1 in your code, but it is not OK to hard code values for all possible command arguments, or probability values for all possible atomic events. More specifically, for full credit, the code should include and use a Bayesian network class. The class should include a member function called computeProbability(b, e, a, j, m), where each argument is a boolean, specifying if the corresponding event (burglary, earthquake, alarm, john-calls, mary-calls) is true or false. This function should return the joint probability of the five events.

Sample output for both Tasks given here.

How to submit

Implementations in C, C++, Java, and Python will be accepted. Points will be taken off for failure to comply with this requirement unless previously cleared with the Instructor.

Create a ZIPPED directory called <net-id>_assmt3.zip (no other forms of compression accepted, contact the instructor or TA if you do not know how to produce .zip files).
The directory should contain the source code for each task in seperate folders titled task1 and task2 (no need for any compiled binaries). Each folder should also contain a file called readme.txt, which should specify precisely:

- Name and UTA ID of the student.
- What programming language is used for this task. (Make sure to also give version and subversion numbers)
- How the code is structured.
- How to run the code, including very specific compilation instructions, if compilation is needed. Instructions such as "compile using g++" are NOT considered specific if the TA needs to do additional steps to get your code to run.
    - If your code will run on the ACS Omega (not required) make a note of it in the readme file.
- Insufficient or unclear instructions will be penalized.
- Code that the Instructor cannot run gets AT MOST 75% credit (depending on if the instructor is able to get it to run with minor edits).