CS3243: Introduction to Artificial Intelligence **(Due: November 11, 2019; 2359hrs)**

Assignment 3: Bayesian networks

Important: Read all the instructions below carefully before you start working on the assignment, and before you make a submission.

- You must do this assignment in **groups of two only**. Please write the names and matriculation numbers of your team members in the code file you submit.

- Sign up your group on LumiNUS: Class and Groups → Class Groups
  - This time, you can form groups with anyone in the module.
  - i.e., there is no tutorial restriction on group forming.
  - Group size is strictly 2 students (No groups of 3 or individuals submissions are allowed. Reason: It helps us to manage logistics and grading).

- All sources of material must be cited. The University Academic Code of Conduct will be strictly enforced.

- **Submission instructions** (*These instructions are different from Assignment 1 and Assignment 2, please pay attention*)
  - You need to submit only the python code file on LumiNUS.
  - You don't have to write any report this time!
  - Make only one submission (i.e., one python file) per group.
  - File names:
    The code filename should be `b_net_A3_xx.py`
    E.g. `b_net_A3_02.py` (note that it is 02 and not 2).

- Points will be deducted for not following the instructions, including the naming convention. Please follow the file naming convention closely.

- Kindly check that the submitted code will be able to run on SoC's Sunfire account. We will be using Sunfire (our UNIX server) to evaluate all submissions.

1 Submission

Your submissions should contain only ONE python code file (please name it according to the instructions above). **No report for this assignment!**

1.1 Code

Please use Python 2.6.4 (the default Python version on SoC's Sunfire) to do this assignment. The template has been provided to you (`b_net_A3_xx.py`). You may import Python Standard libraries if you need (but no external libraries are allowed). Things to note:

- You may create new classes or functions if you think they are helpful (but all classes/functions should be within the same file and used within the `BayesianNetwork` class). You SHOULD NOT modify the `main` function.
- Your answer should be returned by the `infer()` function. However, do note that you need to `construct()` the network first!
- **Executable**: If your program can not be executed, you will get 0 for your code components.
- **Correctness**: Please make sure your algorithm logic is correct. You will lose mark significantly if you can not pass our test cases.
- **Timing**: We will time your solution and use the timing information as one of the components in grading.

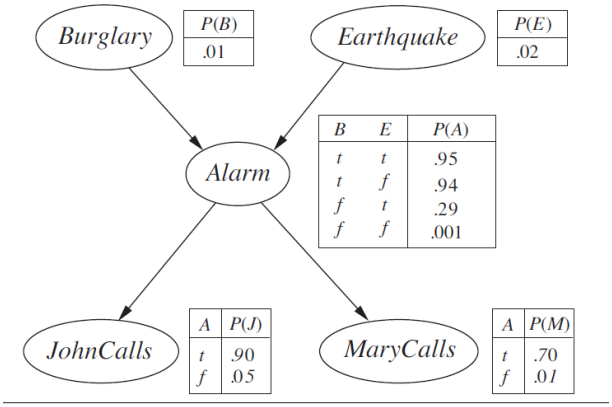

Figure 1: Bayesian network for earthquake problem.

2 Grading

- We will be grading your implementation based on two criteria—correctness and efficiency.
- Each criteria carries the same weightage (i.e., Correctness: 5 points and Efficiency: 5 points).

3 Problem Statement

In this assignment, you are required to implement a Bayesian network, given the structure in the form of `variables` and `dependencies`. Additionally, you need to determine the joint probability given the prior and conditional probabilities. Take care that you don't do unnecessary calculations in your code.

4 Logistics - Input/Output

4.1 Example problem

Consider the Bayesian network given in Figure 1

4.2 Input

Input is provided as a set of JSON[1] files as follows:

- `structure.json`
  - Contains the variables and dependencies.
  - For instance the BN in Figure 1 is encoded as follows.

```
{
    "variables": {
        "Burglary": [ "True", "False" ],
        "Earthquake": [ "True", "False" ],
        "Alarm": [ "True", "False" ],
        "JohnCalls": [ "True", "False" ],
        "MaryCalls": [ "True", "False" ]
    },
    "dependencies": {
        "Alarm": [ "Burglary", "Earthquake" ],
        "JohnCalls": [ "Alarm" ],
        "MaryCalls": [ "Alarm" ]
    }
}
```

[1] JSON can be loaded as Python dictionaries (or `key:value` pairs) in your programs.

  - Each `key` in the `variables` dictionary (e.g., `Burglary`) corresponds to a node in the BN, with the `values` corresponding to the key representing the domain. Hence, in this case, the domain comprises Boolean values.
  - Each `key` in the `dependencies` dictionary (e.g., `Alarm`) corresponds to a node in the BN, with the `values` representing the dependencies. Hence, you read the first entry as `Alarm` depends on `Burglary` and `Earthquake`.
  - The entire structure is specified in the similar manner (i.e., no hidden nodes or traps will be laid!).
  - You are required to construct the BN using this information.
- `values.json`
  - Contains the prior and conditional probabilities associated with the nodes.
  - For instance the probability values associated with the nodes in the BN above are encoded as follows.

```
{
    "prior_probabilities": {
        "Burglary": {
            "True": 0.01,
            "False": 0.99
        },
        "Earthquake": {
            "True": 0.02,
            "False": 0.98
        }
    },
    "conditional_probabilities": {
        "Alarm": [
            {
                "Burglary": "True",
                "Earthquake": "True",
                "own_value": "True",
                "probability": 0.95
            },
            {
                "Burglary": "True",
                "Earthquake": "True",
                "own_value": "False",
                "probability": 0.05
            },
            {
                "Burglary": "False",
                "Earthquake": "True",
                "own_value": "True",
                "probability": 0.29
            },
            {
                "Burglary": "False",
                "Earthquake": "True",
                "own_value": "False",
                "probability": 0.71
            },
            {
                "Burglary": "True",
                "Earthquake": "False",
                "own_value": "True",
                "probability": 0.94
            },
            {
                "Burglary": "True",
                "Earthquake": "False",
                "own_value": "False",
```

```
                    "probability": 0.06
                },
                {
                    "Burglary": "False",
                    "Earthquake": "False",
                    "own_value": "True",
                    "probability": 0.001
                },
                {
                    "Burglary": "False",
                    "Earthquake": "False",
                    "own_value": "False",
                    "probability": 0.999
                }
            ],
            "JohnCalls": [
                {
                    "Alarm": "True",
                    "own_value": "True",
                    "probability": 0.9
                },
                {
                    "Alarm": "True",
                    "own_value": "False",
                    "probability": 0.1
                },
                {
                    "Alarm": "False",
                    "own_value": "True",
                    "probability": 0.05
                },
                {
                    "Alarm": "False",
                    "own_value": "False",
                    "probability": 0.95
                }
            ],
            "MaryCalls": [
                {
                    "Alarm": "True",
                    "own_value": "True",
                    "probability": 0.7
                },
                {
                    "Alarm": "True",
                    "own_value": "False",
                    "probability": 0.3
                },
                {
                    "Alarm": "False",
                    "own_value": "True",
                    "probability": 0.01
                },
                {
                    "Alarm": "False",
                    "own_value": "False",
                    "probability": 0.99
                }
```

```
            ]
        }
    }
```

- The keys `prior_probabilities` and `conditional_probabilities` are self explanatory.
- You read the Prior probabilities as: Probability of `Burglary` happening is 0.01 and not happening is 0.99.
- You read the Conditional probabilities as: $P\left(Alarm = T \mid Burglary = T, Earthquake = T\right) = 0.95$, $P\left(Alarm = T \mid Burglary = F, Earthquake = T\right) = 0.29$ and so on. (`own_value` refers to the value the variable takes. In the above, `own_value` of `Alarm` is `True`).
- Similarly, $P\left(JohnCalls = F \mid Alarm = T\right) = 0.1$.

- `queries.json`
  - Contains a list of queries that we are interested in.
  - For instance, the sample queries could be encoded as follows:

```
[
    {
        "index" : 1,
        "given" : {"Alarm" : "False"},
        "tofind" : {"MaryCalls" : "True"}
    },
    {
        "index" : 2,
        "given" : {"Burglary": "False", "Earthquake": "True"},
        "tofind" : {"Alarm": "False"}
    }
]
```

  - The `index` key refers to the query number. It is important to use the same index number for the answer.
  - You read the first query as "Find $P\left(MaryCalls = T \mid Alarm = F\right)$".
  - Do note that the `given` may be empty in the query. (E.g., In query two above, think how do you evaluate $P(Alarm = F)$.)

4.3 Output

- Your submission should infer the probability of the given query.
- The output should be returned by the `infer()` function in the form of a list of dictionaries.
- E.g., For the queries above, the answer should be returned as follows.

```
[
    { "index" : 1, "answer" : 0.01 },
    { "index" : 2, "answer" : 0.71 }
]
```