Name: Hrishikesh Kumbhar
Class: D20A
Roll no: 32

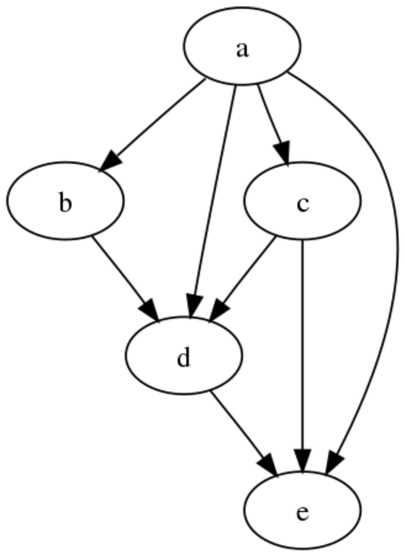
Data Science Lab
Experiment 01

Aim: To implement inferencing with Bayesian Network in Python

Theory:

A Bayesian network (also spelt Bayes network, Bayes net, belief network, or judgment network) is a probabilistic graphical model that depicts a set of variables and their conditional dependencies using a directed acyclic graph (DAG).

Bayesian networks are perfect for taking an observed event and forecasting the likelihood that any of numerous known causes played a role. A Bayesian network, for example, could reflect the probability correlations between diseases and symptoms. Given a set of symptoms, the network may be used to calculate the likelihood of the presence of certain diseases. In graph theory and computer science, a directed acyclic graph (DAG) is a directed graph with no directed cycles. In other words, it's made up of vertices and edges (also called arcs), with each edge pointing from one vertex to the next in such a way that following those directions would never lead to a closed-loop.



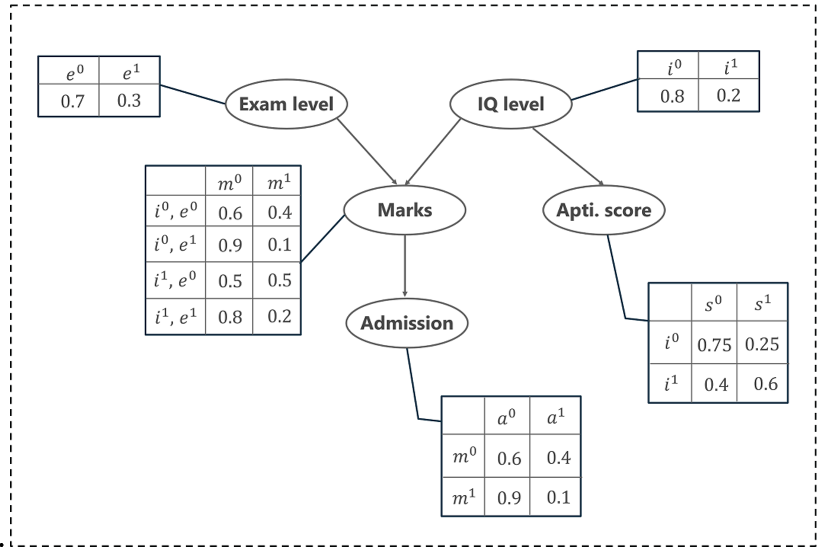
Name: Hrishikesh Kumbhar
Class: D20A
Roll no: 32

A directed graph is one in which all edge directions are consistent and the vertices can be topologically arranged in a linear order. DAGs have various scientific and computing applications, including biology evolution, family trees, epidemiology, and sociology.
Example:
Let’s assume that we’re creating a Bayesian Network that will model the marks (m) of a student on his examination. The marks will depend on:

1. Exam level (e): This is a discrete variable that can take two values, (difficult, easy)
2. IQ of the student (i): A discrete variable that can take two values (high, low)

The marks will in turn predict whether or not he/she will get admitted (a) to a university.

The IQ will also predict the aptitude score (s) of the student.With this information, we can build a Bayesian Network that will model the performance of a student on an exam. The Bayesian Network can be represented as a DAG where each node denotes a variable that predicts the performance of the student.



e^0	e^1
0.7	0.3

i^0	i^1
0.8	0.2

	m^0	m^1
i^0, e^0	0.6	0.4
i^0, e^1	0.9	0.1
i^1, e^0	0.5	0.5
i^1, e^1	0.8	0.2

	s^0	s^1
i^0	0.75	0.25
i^1	0.4	0.6

	a^0	a^1
m^0	0.6	0.4
m^1	0.9	0.1

Name: Hrishikesh Kumbhar
Class: D20A
Roll no: 32

We can now calculate the Joint Probability Distribution of these 5 variables, i.e. the product of conditional probabilities:

$$p(a, m, i, e, s) = p(a \mid m)\, p(m \mid i, e)\, p(i)\, p(e)\, p(s \mid i)$$

Here,

- p(a | m) represents the conditional probability of a student getting an admission based on his marks.
- p(m | I, e) represents the conditional probability of the student's marks, given his IQ level and exam level.
- p(i) denotes the probability of his IQ level (high or low)
- p(e) denotes the probability of the exam level (difficult or easy)
- p(s | i) denotes the conditional probability of his aptitude scores, given his IQ level

The DAG clearly shows how each variable (node) depends on its parent node, i.e., the marks of the student depends on the exam level (parent node) and IQ level (parent node). Similarly, the aptitude score depends on the IQ level (parent node) and finally, his admission into a university depends on his marks (parent node). This relationship is represented by the edges of the DAG.

Name: Hrishikesh Kumbhar
Class: D20A
Roll no: 32

The probability of a random variable depends on his parents. Therefore, we can formulate Bayesian Networks as:

$$P(X_1, \dots, X_n) = \Pi_{i=1}^{n} p(X_i \mid Parents(X_i))$$

Where, X_i denotes a random variable, whose probability depends on the probability of the parent nodes, *Parents*(*X_i*).

Code and Output:

In this demonstration, we'll use Bayesian Networks to solve the well-known Monty Hall Problem. Let me explain the Monty Hall problem to those of you who are unfamiliar with it. This problem entails a competition in which a contestant must choose one of three doors, one of which conceals a price. The show's host (Monty) unlocks an empty door and asks the contestant if he wants to swap to the other door after the contestant has chosen one. The decision is whether to keep the current door or replace it with a new one. It is preferable to enter by the other door because the price is more likely to be higher. To come out from this ambiguity let's model this with a Bayesian network.

Name: Hrishikesh Kumbhar
Class: D20A
Roll no: 32

For this demonstration, we are using a python-based package **pgmpy** is a Bayesian Networks implementation written entirely in Python with a focus on modularity and flexibility. Structure Learning, Parameter Estimation, Approximate (Sampling-Based) and Exact inference, and Causal Inference are all available as implementations.

```
!pip install pgmpy
```

Step 1: To initialize the BayesianModel by passing a list of edges in the model structure.

```
from pgmpy.models import BayesianNetwork

from pgmpy.factors.discrete import TabularCPD

import networkx as nx

import pylab as plt
```

Step 2: Define the CPDs(Conditional Probability Distribution).

```
model = BayesianNetwork([('Guest', 'Host'), ('Price', 'Host')])
# Defining the CPDs:
cpd_guest = TabularCPD('Guest', 3, [[0.33], [0.33], [0.33]])
cpd_price = TabularCPD('Price', 3, [[0.33], [0.33], [0.33]])
cpd_host = TabularCPD('Host', 3, [[0, 0, 0, 0, 0.5, 1, 0, 1, 0.5], [0.5,
0, 1, 0, 0, 0, 1, 0, 0.5], [0.5, 1, 0, 1, 0.5, 0, 0, 0, 0]],
evidence=['Guest', 'Price'], evidence_card=[3, 3])
```

Name: Hrishikesh Kumbhar
Class: D20A
Roll no: 32

Step 3: Add the CPDs to the model.

```
# Associating the CPDs with the network structure.

model.add_cpds(cpd_guest, cpd_price, cpd_host)

model.check_model()
```

```
True
```

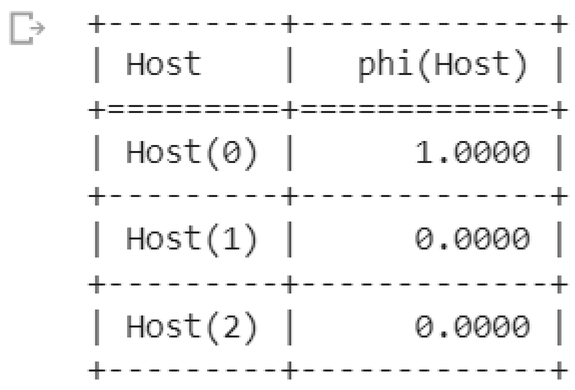
Now let's infer the network, if we want to check at the next step which door will the host open now. For that, we need access to the posterior probability from the network and while accessing we need to pass the evidence to the function. Evidence is needed to be given when we are evaluating posterior probability, here in our task evidence is nothing but which door is Guest selected and where is the Price.

```
from pgmpy.inference import VariableElimination
infer = VariableElimination(model)
posterior_p = infer.query(['Host'], evidence={'Guest': 2, 'Price': 2})
print(posterior_p)
```

```
+---------+-------------+
| Host    |   phi(Host) |
+=========+=============+
| Host(0) |      0.5000 |
+---------+-------------+
| Host(1) |      0.5000 |
+---------+-------------+
| Host(2) |      0.0000 |
+---------+-------------+
```

Name: Hrishikesh Kumbhar
Class: D20A
Roll no: 32

```python
# Inferring the posterior probability
from pgmpy.inference import VariableElimination
infer = VariableElimination(model)
posterior_p = infer.query(['Host'], evidence={'Guest': 1, 'Price': 2})
print(posterior_p)
```



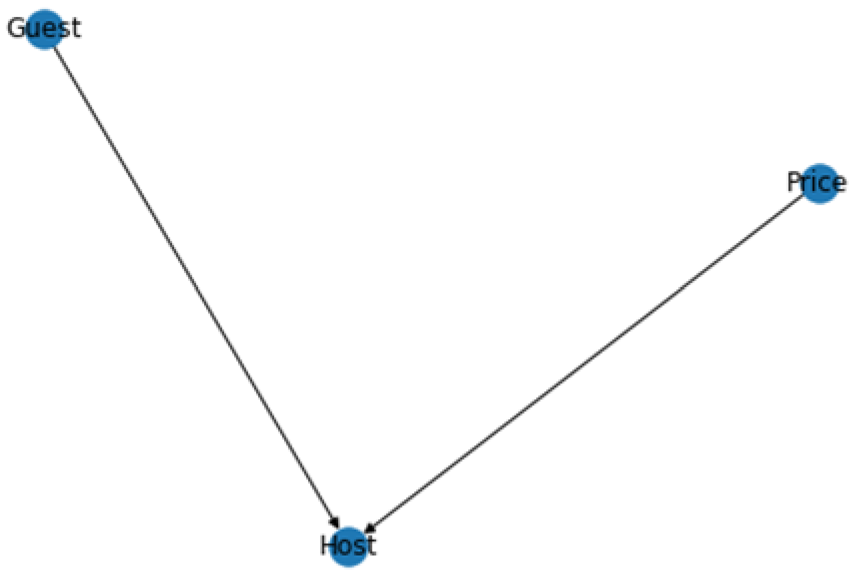
```
+---------+-------------+
| Host    |   phi(Host) |
+=========+=============+
| Host(0) |      1.0000 |
+---------+-------------+
| Host(1) |      0.0000 |
+---------+-------------+
| Host(2) |      0.0000 |
+---------+-------------+
```

The probability distribution of the Host is clearly satisfying the theme of the contest. In reality also, in this situation the host is definitely not going to open the second door; he will open either of the first two and that's what the above simulation tells us.

Now, let's plot our above model. This can be done with the help of Network and Pylab. NetworkX is a Python-based software package for constructing, altering, and researching the structure, dynamics, and function of complex networks. PyLab is a procedural interface to the object-oriented charting toolkit Matplotlib, and it is used to examine large complex networks represented as graphs with nodes and edges.

Name: Hrishikesh Kumbhar
Class: D20A
Roll no: 32

```
nx.draw(model, with_labels=True)
```



Conclusion: Thus we studied an overview of Bayesian networks and implemented Inferencing with Bayesian Network in Python.