Name: Atif Ansari **Roll no:** 04 **Class:** D20B

Practical 1: Implementing Inferencing with Bayesian Network in Python

Aim:

To implement and perform probabilistic inferencing using a Bayesian Network in Python, and to query the network to compute marginal and conditional probabilities using different datasets and inputs.

Concept:

A **Bayesian Network (BN)** — also called a *Bayes Net* or *Belief Network* — is a probabilistic graphical model that represents a set of variables and their conditional dependencies via a **Directed Acyclic Graph (DAG)**.

- **Nodes:** Represent random variables (discrete or continuous).
- **Edges:** Represent conditional dependencies between variables.
- **Conditional Probability Tables (CPTs):** Quantify the strength of dependencies for each node given its parents.
- **Inference:** The process of computing marginal or conditional probabilities given evidence.

Bayesian Networks are particularly useful for reasoning under uncertainty, predicting outcomes, and updating beliefs when new information is available.

Example: In medical diagnosis, given a set of symptoms, a BN can compute the probability of various diseases.

Applications of Bayesian Networks:

1. **Medical Diagnosis:** Inferring disease probabilities based on symptoms.
2. **Fault Diagnosis:** Detecting failures in complex systems (e.g., aircraft, nuclear plants).
3. **Decision Support Systems:** Risk assessment in finance and marketing.

4. **Gene Expression Analysis:** Studying dependencies between genes in bioinformatics.

Tools Used:

- **Python** programming language
- **pgmpy** library for probabilistic graphical models
- **matplotlib / networkx** for diagram visualization

Implementation Steps:

1. Install the `pgmpy` library.
2. Define the Bayesian Network structure (nodes & edges).
3. Define the Conditional Probability Distributions (CPDs) for each variable.
4. Add CPDs to the model and validate.
5. Perform inference queries for marginal and conditional probabilities.
6. Test with different datasets and different evidence inputs.

Code:

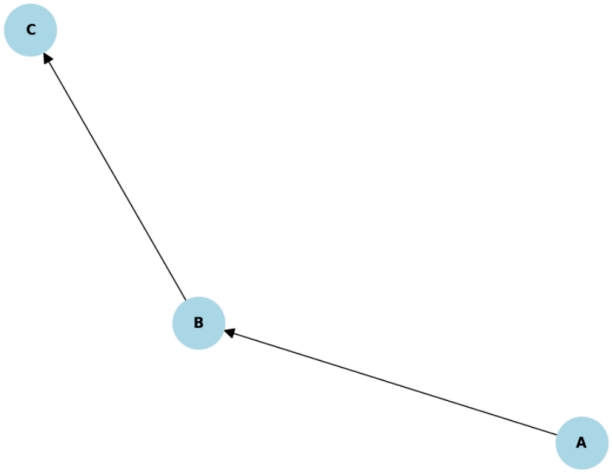
Dataset 1: Simple Chain A → B → C

```
import matplotlib.pyplot as plt

import networkx as nx

from pgmpy.models import DiscreteBayesianNetwork

from pgmpy.factors.discrete import TabularCPD

from pgmpy.inference import VariableElimination

# Define structure

model = DiscreteBayesianNetwork([('A', 'B'), ('B', 'C')])

# CPDs
```

```
cpd_a = TabularCPD('A', 2, [[0.6], [0.4]])

cpd_b = TabularCPD('B', 2, [[0.7, 0.2],

                                                [0.3,  0.8]],  evidence=['A'],
evidence_card=[2])

cpd_c = TabularCPD('C', 2, [[0.9, 0.4],

                                                [0.1,  0.6]],  evidence=['B'],
evidence_card=[2])

# Add and check model

model.add_cpds(cpd_a, cpd_b, cpd_c)

assert model.check_model()

# Create a networkx graph from the Bayesian network structure

graph = nx.DiGraph()

graph.add_edges_from(model.edges())

# Draw network using NetworkX

plt.figure(figsize=(8, 6))

nx.draw(graph,  with_labels=True,  node_size=2000,  node_color="lightblue",
arrowsize=20, font_size=12, font_weight='bold')

plt.show()

# Inference

inference = VariableElimination(model)

print("P(C):")

print(inference.query(variables=['C']))



print("\nP(C | A=1):")
```

```
print(inference.query(variables=['C'], evidence={'A': 1}))

print("\nP(B | C=0):")

print(inference.query(variables=['B'], evidence={'C': 0}))
```

Output:




```
P(C):
+------+----------+
| C    |   phi(C) |
+======+==========+
| C(0) |   0.6500 |
+------+----------+
| C(1) |   0.3500 |
+------+----------+
```

```
P(C | A=1):
+------+----------+
| C    |   phi(C) |
+======+==========+
| C(0) |   0.5000 |
+------+----------+
| C(1) |   0.5000 |
+------+----------+
```

```
P(B | C=0):
+------+----------+
| B    |   phi(B) |
+======+==========+
| B(0) |   0.6923 |
+------+----------+
| B(1) |   0.3077 |
+------+----------+
```

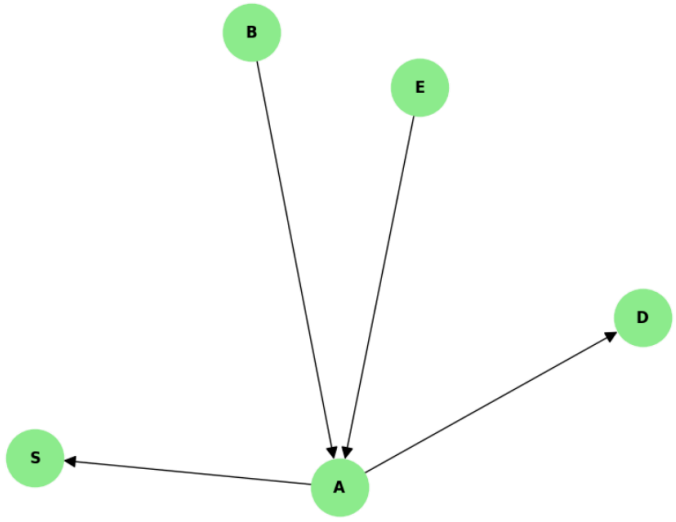
Dataset 2: Burglary–Earthquake–Alarm Example

```
import matplotlib.pyplot as plt

import networkx as nx

from pgmpy.models import DiscreteBayesianNetwork

from pgmpy.factors.discrete import TabularCPD
```

```
from pgmpy.inference import VariableElimination

# Define structure

model2 = DiscreteBayesianNetwork([('B', 'A'), ('E', 'A'), ('A', 'D'),
('A', 'S')])

# CPDs

cpd_b = TabularCPD('B', 2, [[0.002], [0.998]])

cpd_e = TabularCPD('E', 2, [[0.001], [0.999]])

cpd_a = TabularCPD('A', 2, [[0.94, 0.95, 0.29, 0.001],

                            [0.06, 0.05, 0.71, 0.999]],

                   evidence=['B', 'E'], evidence_card=[2, 2])

cpd_d = TabularCPD('D', 2, [[0.91, 0.05],

                            [0.09, 0.95]], evidence=['A'],
evidence_card=[2])

cpd_s = TabularCPD('S', 2, [[0.75, 0.02],

                            [0.25, 0.98]], evidence=['A'],
evidence_card=[2])

# Add and check model

model2.add_cpds(cpd_b, cpd_e, cpd_a, cpd_d, cpd_s)

assert model2.check_model()

# Create a networkx graph from the Bayesian network structure

graph2 = nx.DiGraph()

graph2.add_edges_from(model2.edges())

# Draw network using NetworkX

plt.figure(figsize=(8, 6))
```

```
nx.draw(graph2, with_labels=True, node_size=2000, node_color="lightgreen",
arrowsize=20, font_size=12, font_weight='bold')

plt.show()

# Inference

inference2 = VariableElimination(model2)

print("\nP(A | B=0, E=0):")

print(inference2.query(variables=['A'], evidence={'B': 0, 'E': 0}))

print("\nP(D, S, A | B=0, E=0):")

print(inference2.query(variables=['D', 'S', 'A'], evidence={'B': 0, 'E':
0}, joint=True))
```

Output:



```
P(A | B=0, E=0):
+------+----------+
| A    |   phi(A) |
+======+==========+
| A(0) |   0.9400 |
+------+----------+
| A(1) |   0.0600 |
+------+----------+
```

```
P(D, S, A | B=0, E=0):
+------+------+------+--------------+
| D    | S    | A    |   phi(D,S,A) |
+======+======+======+==============+
| D(0) | S(0) | A(0) |       0.6416 |
+------+------+------+--------------+
| D(0) | S(0) | A(1) |       0.0001 |
+------+------+------+--------------+
| D(0) | S(1) | A(0) |       0.2139 |
+------+------+------+--------------+
| D(0) | S(1) | A(1) |       0.0029 |
+------+------+------+--------------+
| D(1) | S(0) | A(0) |       0.0635 |
+------+------+------+--------------+
| D(1) | S(0) | A(1) |       0.0011 |
+------+------+------+--------------+
| D(1) | S(1) | A(0) |       0.0212 |
+------+------+------+--------------+
| D(1) | S(1) | A(1) |       0.0559 |
+------+------+------+--------------+
```

Results:

- **Dataset 1 Output:** Shows marginal probability of C, conditional probability given A=1, and backward inference B given C=0.
- **Dataset 2 Output:** Gives joint and marginal probabilities for the burglary–earthquake–alarm model.

Conclusion:

In this practical, we learned the fundamentals of **Bayesian Networks** and their real-world uses. Using the **pgmpy** library, we implemented two Bayesian Network models in Python, visualized their structures, and performed probabilistic inference with various inputs.

The results highlighted how Bayesian Networks effectively model uncertainty, updating beliefs as new evidence is introduced. This demonstrates their value in decision-making and predicting outcomes in uncertain environments.