PRACTICAL – 6

AIM: Write a python program to implement Bayesian Network to model probabilistic reasoning for a burglary alarm system.

Code:

```
class BayesNode:
   def __init__(self, name, dependentList, dictValues):
      self.name = name
      self.dependentList = dependentList
      self.dictValues = dictValues

   def get_prob_val(self, boolValue, dictValues):
      listTuple = []
      for eval in self.dependentList:
         listTuple.append(dictValues[eval.name])

      listTuple = tuple(listTuple)
      tupleKey = (boolValue, listTuple)
      return self.dictValues.get(tupleKey)

class BurglaryAlarmProb:
   def __init__(self):
      self.B = BayesNode("B", [],{(True, ()): 0.001, (False, ()): 0.999})
      self.E = BayesNode("E", [], {(True, ()): 0.002, (False, ()): 0.998})
      self.A = BayesNode("A", [self.B, self.E], {(True, (True, True)): 0.95, (True, (True, False)): 0.94, (True, (False, True)): 0.29, (True, (False, False)): 0.001, (False, (True, True)): 0.05, (False, (True, False)): 0.06, (False, (False, True)): 0.71, (False, (False, False)): 0.999})
      self.J = BayesNode("J", [self.A], {(True, (True,)): 0.90, (True, (False,)): 0.05, (False, (True,)): 0.1, (False, (False,)): 0.95})
      self.M = BayesNode("M", [self.A], {(True, (True,)): 0.70, (True, (False,)): 0.01, (False, (True,)): 0.3, (False, (False,)): 0.99})

   def and_prob(self, JM_value, B_value, consider_J):
      prob=0
      JM = self.J if consider_J else self.M
      for A_value in [True, False]:
         prob2=0
         prob1 = JM.get_prob_val(JM_value, {self.A.name: A_value})
         for E_value in [True, False]:
            prob2 += self.A.get_prob_val(A_value, {self.B.name: B_value, self.E.name: E_value}) * self.B.get_prob_val(B_value, {}) * self.E.get_prob_val(E_value, {})
         prob+= prob1 * prob2
```

```
        return prob

    def JM_prob(self, JM_value, consider_J):
        prob=0
        JM = self.J if consider_J else self.M
        for A_value in [True, False]:
            prob2=0
            prob1 = JM.get_prob_val(JM_value, {self.A.name: A_value})
            prob2 = self.A_prob(A_value)
            prob+= prob1 * prob2
        return prob

    def A_prob(self, A_value):
        prob=0
        for B_value in [True, False]:
            for E_value in [True, False]:
                prob += self.A.get_prob_val(A_value, {self.B.name: B_value, self.E.name: E_value})
* self.B.get_prob_val(B_value, {}) * self.E.get_prob_val(E_value, {})
        return prob

    def condition_prob(self, B_value, consider_J):
        return self.and_prob(True, B_value, consider_J) / self.JM_prob(True, consider_J)



obj = BurglaryAlarmProb()

print("P(J,B) = "+ str(obj.and_prob(True, True, True)))
print("P(J',B) = "+ str(obj.and_prob(False, True, True)))
print("P(J',B') = "+ str(obj.and_prob(False, False, True)))
print("P(J,B') = "+ str(obj.and_prob(True, False, True)))
print("P(M,B) = "+ str(obj.and_prob(True, True, False)))
print("P(M',B) = "+ str(obj.and_prob(False, True, False)))
print("P(M',B') = "+ str(obj.and_prob(False, False, False)))
print("P(M,B') = "+ str(obj.and_prob(True, False, False)))
print("P(J) = " + str(obj.JM_prob(True, True)))
print("P(M) = " + str(obj.JM_prob(True, False)))
print("P(J') = " + str(obj.JM_prob(False, True)))
print("P(M') = " + str(obj.JM_prob(False, False)))
print("P(A) = " + str(obj.A_prob(True)))
print("P(A') = " + str(obj.A_prob(False)))
print("P(B|M) = " + str(obj.condition_prob(True, False)))
print("P(B'|M) = " + str(obj.condition_prob(False, False)))
```

NAME: DHRUV SHERE
ENROLLMENT NO.: 23012022021
BATCH: C-2

```
print("P(B|J) = " + str(obj.condition_prob(True, True)))
print("P(B'|J) = " + str(obj.condition_prob(False, True)))
```

Output:

```
P(J,B) = 0.000849017
P(J',B) = 0.000150983
P(J',B') = 0.9477100412999999
P(J,B') = 0.0512899587
P(M,B) = 0.0006586137999999999
P(M',B) = 0.00034138619999999996
P(M',B') = 0.9879222688199999
P(M,B') = 0.01107773118
P(J) = 0.052138975700000006
P(M) = 0.01173634498
P(J') = 0.9478610243
P(M') = 0.98826365502
P(A) = 0.002516442
P(A') = 0.997483558
P(B|M) = 0.056117454038914924
P(B'|M) = 0.943882545961085
P(B|J) = 0.016283729946769934
P(B'|J) = 0.9837162700532299
```