CSCI 3202, Spring 2023

Homework 4

Due: Friday, 4/7, at 9:00 PM

Your name: Julia Troni

Some useful packages and libraries:

In [1]:
```
from scipy import stats
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from typing import Dict, List, Tuple, Union
import unittest
from __future__ import annotations
```

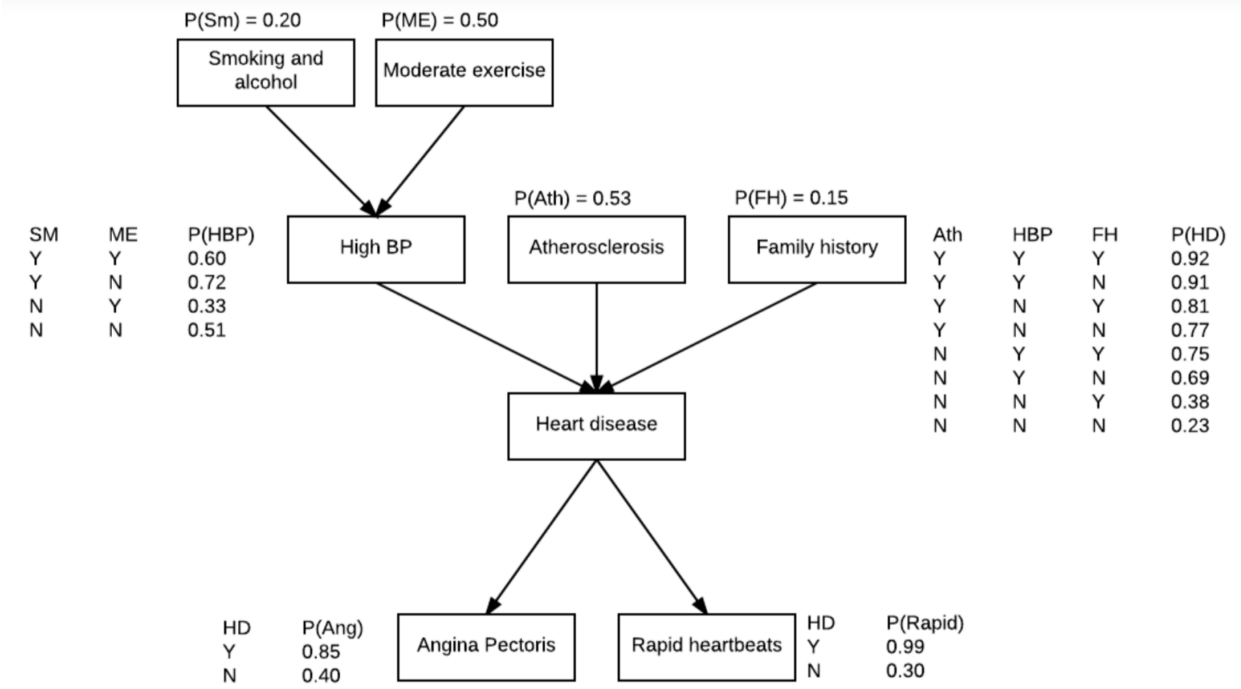
Unit tests

In [2]:
```
# TODO: Reference this to understand how we initialize and use the BayesNode class.
class Tests_Problem1(unittest.TestCase):
    def setUp(self):
        self.p1 = BayesNode('p1', '', [T,F], 0.3)
        self.p2 = BayesNode('p2', '', [T,F], 0.6)
        self.c  = BayesNode('c', ['p1', 'p2'], [T,F], {(T,T):0.1, (T,F):0.2, (F,T):0.3,
    def test_onenode(self):
        self.assertEqual(P(self.p1, T, {}), 0.3)
    def test_twonode(self):
        self.assertEqual(P(self.c, F, {'p1':T, 'p2':F}), 0.8)
```

Problem 1: Bayesian network to model heart disease

The following Bayesian network is based loosely on a study that examined heart disease risk factors in 167 elderly individuals in South Carolina. Note that this figure uses Y and N to represent Yes and

No, which can be represented as True and False Boolean values.



(1a) (20 points)

Create a `BayesNet` object to model this. Below are the codes for the (conditional) probability `P` function and `BayesNode` class as well. You can code this however you want, subject to the following constraints:

1. the nodes are represented using the `BayesNode` class and can work with the `P` function for probabilities,
2. your `BayesNet` structure keeps track of which nodes are in the Bayes net, as well as
3. which nodes are the parents/children of which other nodes.

Some *suggested* skeleton codes for a class structure are given. The suggestions for methods to implement are in view of the fact that we will need to calculate some probabilities, which is going to require us to `find_node` s and `find_values` that nodes can take on.

In [ ]:

In [3]:
```
## For the sake of brevity...
T, F = True, False


# Can be raised in BayesNet.
class FindNodeException(Exception):
    pass


#given parents values to determine P(var | parents)
def P(var, value, evidence={}):
```

```
    '''The probability distribution for P(var | evidence),
    when all parent variables are known (in evidence)'''
    if len(var.parents)==1:
        # only one parent
        row = evidence[var.parents[0]]
    else:
        # multiple parents
        row = tuple(evidence[parent] for parent in var.parents)
        #print("evidence", evidence)

       #print("row", row)

    return var.cpt[row] if value else 1-var.cpt[row]


class BayesNode:

    def __init__(self, name, parents, values, cpt):
        if isinstance(parents, str):
            parents = parents.split()

        if len(parents)==0:
            # if no parents, empty dict key for cpt
            cpt = {(): cpt}
        elif isinstance(cpt, dict):
            # if there is only one parent, only one tuple argument
            if cpt and isinstance(list(cpt.keys())[0], bool):
                cpt = {(v): p for v, p in cpt.items()}

        self.variable = name
        self.parents = parents
        self.cpt = cpt
        self.values = values
        self.children = []
        #print(cpt)

    def __repr__(self):
        return repr((self.variable, ' '.join(self.parents)))

class BayesNet:
    '''Bayesian network containing only boolean-variable nodes.'''

    def __init__(self, node_specs=[]):
        '''node_specs is a list of tuples, one per node
        each node has (name, parents, cpt) and nodes must
        be ordered with parents before children.'''
        self.nodes = []
        self.variables = []
        for node_spec in node_specs:
            self.add(*node_spec)

    def add(self, name, parents, values, cpt):
        '''Add a new node to the BayesNet, with the given name, parents and
        conditional probability table (cpt). The parents must already be in
        the net, and the variable itself must not be.'''
        node = BayesNode(name=name, parents=parents, values=values, cpt=cpt)
        assert node.variable not in self.variables
        assert all((parent in self.variables) for parent in node.parents)
        self.nodes.append(node)
        self.variables.append(node.variable)
```

```
        for parent in node.parents:
            self.find_node(parent).children.append(node)

    def find_node(self, var):
        '''Find and return the BayesNode with name `var`'''
        for n in self.nodes:
            if n.variable == var:
                return n
        raise Exception("No such variable: {}".format(var))

    def find_values(self, var):
        '''Return the set of possible values for variable `var`'''
        varnode = self.find_node(var)
        return varnode.values

    def __repr__(self):
        return 'BayesNet({0!r})'.format(self.nodes)
```

In [ ]:

In [ ]:

Unit tests

In [4]:
```
tests_to_run = unittest.TestSuite()
tests_to_run.addTest(Tests_Problem1("test_onenode"))
tests_to_run.addTest(Tests_Problem1("test_twonode"))
unittest.TextTestRunner().run(tests_to_run)
```

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.010s

OK
```

Out[4]:
```
<unittest.runner.TextTestResult run=2 errors=0 failures=0>
```

Create the object representing the Bayesian Network depicted in firgure 1

- Be sure to use the var names in the conditional probability tables in the image
- HINT: Instantiate all of the required nodes, and then initialize the network, making sure you provide it the nodes in the correct order.

In [5]:
```
# TODO: Create your network here.
# IMPORTANT: name the network variable `bnHeart`
# TODO: set this to your BayesianNetwork.
bnHeart = BayesNet([
    ('Sm', '', [T,F], 0.20),
    ('ME', '', [T,F], 0.50),
    ('HBP', ['Sm', 'ME'], [T,F], {(T, T): 0.60, (T, F): 0.72, (F, T): 0.33, (F, F): 0.5

    ('Arth', '', [T,F], 0.53),
```

```
    ('FH', '', [T,F], 0.15),

    ('HD', ['HBP', 'Arth', 'FH'], [T,F], {(T, T, T): 0.92, (T, T, F): 0.91, (T, F, T):

    ('Ang', 'HD', [T,F], {T: 0.85, F: 0.40}),
    ('Rapid', 'HD', [T,F], {T: 0.99, F: 0.30})

])
```

In [6]:
```
bnHeart
```

Out[6]:
```
BayesNet([('Sm', ''), ('ME', ''), ('HBP', 'Sm ME'), ('Arth', ''), ('FH', ''), ('HD', 'HB
P Arth FH'), ('Ang', 'HD'), ('Rapid', 'HD')])
```

In [7]:
```
bnHeart.__dict__
```

Out[7]:
```
{'nodes': [('Sm', ''),
  ('ME', ''),
  ('HBP', 'Sm ME'),
  ('Arth', ''),
  ('FH', ''),
  ('HD', 'HBP Arth FH'),
  ('Ang', 'HD'),
  ('Rapid', 'HD')],
 'variables': ['Sm', 'ME', 'HBP', 'Arth', 'FH', 'HD', 'Ang', 'Rapid']}
```

In [8]:
```
bnHeart.nodes
```

Out[8]:
```
[('Sm', ''),
 ('ME', ''),
 ('HBP', 'Sm ME'),
 ('Arth', ''),
 ('FH', ''),
 ('HD', 'HBP Arth FH'),
 ('Ang', 'HD'),
 ('Rapid', 'HD')]
```

In [9]:
```
bnHeart.variables
```

Out[9]:
```
['Sm', 'ME', 'HBP', 'Arth', 'FH', 'HD', 'Ang', 'Rapid']
```

(1b) (20 points)

Craft a function `get_prob(X, e, bn)` to return the **normalized** probability distribution of variable `X` in Bayes net `bn`, given the evidence `e`. That is, return $P(X \mid e)$. The arguments are:

- `X` is some representation of the variable you are querying the probability distribution of. Either a string (the variable name from the `BayesNode` or a `BayesNode` object itself are good options.
- `e` is some representation of the evidence your probability is conditioned on. When given an empty argument (or `None`) for `e`, `get_prob` should return the marginal distribution $P(X)$.
- `bn` is your `BayesNet` object.

You should do this using the `enumeration` algorithm (pseudocode is in the book, code example is in the class github repo), or by brute force (i.e., use a few `for` loops). Either way, you should be using your `BayesNet` object to keep track of all the nodes and relationships between nodes so your `get_prob` function knows these things.

In [10]:

```
class PDF_discrete:
    """Defines a discrete probability distribution function."""

    def __init__(self, varname: str="?", freqs: Dict[Tuple[bool], int]=None):
        """Initializes the PDF.

        Creates a dictionary of values - frequency pairs, then normalizes the distribut
        Initializes the instance variables `self.prob`, `self.varname`, and `self.value

        Args:
            varname (str, optional): _description_. Defaults to "?".
            freqs (Dict[Tuple[bool], int], optional): _description_. Defaults to None.
        """
        # TODO: Replace the NotImplementedError below with your code.
        self.prob = {}
        self.varname = varname
        self.values = []
        if freqs:
            for (v, p) in freqs.items():
                self[v] = p
        self.normalize()


    def __getitem__(self, value: str) -> float:
        """Given a value, returns P(value).

        Args:
            value (str): The value whose probability we want.

        Returns:
            float: The prob of that value.
        """
        # TODO: Return P(value)
        # HINT: What should you do if the value is not defined?
        # TODO: Replace the NotImplementedError below with your code.
        try:
            return self.prob[value]
        except KeyError:
            return 0

    def __setitem__(self, value: str, p: float):
        """Sets P(value) = p

        Args:
            value (str): The value to add.
            p (float): P(value).
        """
        # TODO: Set P(value) to `p` and make sure `value` is in `self.values``
        # TODO: Replace the NotImplementedError below with your code.
        if value not in self.values:
            self.values.append(value)
        self.prob[value]=p
```

```
    def normalize(self) -> PDF_discrete:
        """Normalizes the probability distribution and returns it.
        If the sum of PDF values is 0, then returns a 0.

        Returns:
            PDF_discrete: The PDF_discrete object with a normalized distribution.
        """
        # TODO: Update the `prob` Dict so it is normalized.
        # TODO: Replace the NotImplementedError below with your code.
        total = sum(self.prob.values())
        if not np.isclose(total, 1.0):
            for value in self.prob:
                self.prob[value] /= total
        return self


# TODO: Can some course staff double check that this s keys are tuples?
def extend(s: Dict[str, bool], var: Tuple[bool], val: float) -> Dict[Tuple[bool]]:
    """Copies the evidence s and extend it by setting var to val; returns the copy.

    Args:
        s (Dict[str, bool]).
        var (Tuple[bool]).
        val (float).

    Returns:
        Dict[str, bool].
    """
    # TODO: Replace the NotImplementedError below with your code.
    s2 = s.copy()
    s2[var] = val
    return s2


def enumerate_all(variables: List[str], e: Dict[str, float], bn: BayesNet) -> float:
    """Returns the sum of the entries in P(variables | e[others])
    consistent with e, where P is the joint distribution represented
    by bn, and e[others] means e restricted to bn's other variables
    (the ones other than those in the `variables` argument). Parents must precede child

    Args:
        variables (List[str]): The List of variables whose probability we want.
        e (Dict[str, float]): THe vidence dictionary.
        bn (BayesNet).

    Returns:
        float: P(variables | e[others])
    """
    # TODO: Compute P(variables | others)
    # HINT: This is a product of probabilities -- you should use the function `P()` her
    # HINT: If a variable is not in e, it can be computed with the Law of Total Prob.
    # TODO: Replace the NotImplementedError below with your code.
    if not variables:
        return 1.0
    Y,rest = variables[0], variables[1:]
    Ynode = bn.find_node(Y)
    if Y in e:
        # Y in evidence, so we know its value and just multiply
```

```
            return P(Ynode, e[Y], e) * enumerate_all(rest, e, bn)
        else:
            # Y not in evidence so we have to sum (Law of Total Prob.)
            return sum(P(Ynode, y, e) * enumerate_all(rest, extend(e, Y, y), bn)
                       for y in bn.find_values(Y))


def get_prob(X: str, e: Dict[str, bool], bn: BayesNet) -> PDF_discrete:
    """Return the conditional probability distribution of variable X
    given evidence e, from BayesNet bn. [Figure 14.9]

    Args:
        X (str): The name of a BayesNode.
        e (Dict[str, float]): The Evidence.
        bn (BayesNet): The BayesNet containing the node X.

    Returns:
        PDF_discrete: The normalized PDF
    """
    # TODO: Initialize the PDF
    # TODO: Find the values of X and add them to the PDF
    # HINT: Here you should use the `enumerate_all` and `extend` functions.
    # TODO: Finalize the PDF
    # TODO: Replace the NotImplementedError below with your code.
    Q = PDF_discrete(X)
    for xi in bn.find_values(X):
        Q[xi] = enumerate_all(bn.variables, extend(e, X, xi), bn)
    return Q.normalize()
```

In [ ]:

In [11]:
```
p1 = get_prob(X='FH', e={}, bn=bnHeart)
p1.prob[True]
```

Out[11]: 0.15

Use your `get_prob` function to calculate the following probabilities. Print them to the screen and compare to the original Bayes net figure given to make sure the output passes these "unit tests".

1. The marginal probability of `Family History` is $P(FH = T) = 0.15$
2. The probability of *not* experiencing `Angina Pectoris`, given `Heart Disease` is observed, is $P(Ang = F \mid HD = T) = 1 - 0.85 = 0.15$
3. The probability of `High Blood Pressure`, given a person does `Smoke and/or use Alcohol` but does not get `Moderate Exercise`, is $P(HBP = T \mid Sm = T, ME = F) = 0.72$

In [12]:
```
# 1.
# TODO: compute the PDF for 1) defined above
p1 = get_prob(X='FH', e={}, bn=bnHeart)
print( "P(FH=T)=", p1.prob[T])

# 2.
# TODO: compute the PDF for 2) defined above
p2 = get_prob(X='Ang', e={'HD':T}, bn=bnHeart)
```

```
print("P(Ang=F|HD=T)= ", p2.prob[F])
# 3.
# TODO: compute the PDF for 3) defined above
p3 = get_prob(X='HBP', e={'Sm':T, 'ME':F}, bn=bnHeart)
print("P(HBP=T|Sm=T,ME=F)= ", p3.prob[T])
```

```
P(FH=T)= 0.15
P(Ang=F|HD=T)=  0.1500000000000001
P(HBP=T|Sm=T,ME=F)=  0.7199999999999999
```

(1c) (25 points)

Calculate the probability of observing someone with `High Blood Pressure`, $P(HBP = T)$, *by hand*. Either use Markdown / LaTeX in the box below or do this on paper and put a photo of your work in the box.

$$P(HBP) = \sum_{SM} \sum_{ME} P(HBP, SM, ME)$$

$$P(HBP) = \sum_{SM} \sum_{ME} P(HBP \mid sm, me) \cdot P(sm \mid me) \cdot P(me)$$

$$P(HBP) = \sum_{SM} \sum_{ME} P(HBP \mid sm, me) \cdot P(sm) \cdot P(me)$$

$$\begin{aligned} P(HBP) = {} & P(HBP \mid sm=T, me=T) \cdot P(sm=T) \cdot P(me=T) \\ & + P(HBP \mid sm=T, me=F) \cdot P(sm=T) \cdot P(me=F) \\ & + P(HBP \mid sm=F, me=T) \cdot P(sm=F) \cdot P(me=T) \\ & + P(HBP \mid sm=F, me=F) \cdot P(sm=F) \cdot P(me=F) \end{aligned}$$

$$P(HBP) = (0.6 \cdot 0.2 \cdot 0.5) + (0.72 \cdot 0.2 \cdot 0.5) + (0.33 \cdot 0.8 \cdot 0.5) + (0.51 \cdot 0.8 \cdot 0.5)$$

$$P(HBP=T) = 0.468$$

In [13]:
```
.6*.2*.5 + .72*.2*.5 + .33*.8*.5 + .51*.8*.5
```

Out[13]:
```
0.468
```

Verify your calculation using your `get_prob` function.

In [14]:
```
# TODO: Replace None with code to compute a PDF as in the end of 1b
pHBP_T = get_prob(X='HBP', e={}, bn=bnHeart)
print("P(HBP=T|Sm=T,ME=F)= ", pHBP_T.prob[T])
```

```
P(HBP=T|Sm=T,ME=F)=  0.4680000000000001
```

(1d) (30 points)

Now calculate the following probabilities using your `get_prob` function.

[i] The probability of an arbitrary individual having `Heart Disease`, $P(HD = T)$

In [15]:
```
# Your code here.
p_HD_T = get_prob(X='HD', e={}, bn=bnHeart)
print("P(HD=T)= ", p_HD_T.prob[T])
```

```
P(HD=T)=  0.65700256
```

[ii] The probability that an individual does *not* have `Heart Disease`, given that `Rapid Heartbeat` was observed, $P(HD = F \mid Rapid = T)$

In [16]:
```
# Your code here.
p_HD_F_RH_T = get_prob(X='HD', e={'Rapid':T}, bn=bnHeart)
print("P(HD=F|Rapid=T)= ", p_HD_F_RH_T.prob[F])
```

```
P(HD=F|Rapid=T)=  0.13659218499670053
```

[iii] The probability of an individual having `High Blood Pressure` if they have `Heart Disease` and a `Family History`, $P(HBP = T \mid HD = T, FH = T)$

In [17]:
```
# Your code here.
p_HBP_T_HD_T_FH_T = get_prob(X='HBP', e={'HD':T, 'FH':T}, bn=bnHeart)
print("P(HBP=T|HD=T,FH=T)= ", p_HBP_T_HD_T_FH_T.prob[T])
```

```
P(HBP=T|HD=T,FH=T)=  0.570056292379206
```

[iv] The probability that an individual is a `Smoker/Alcohol User` if they have `Heart Disease`, $P(Sm = T \mid HD = T)$

In [18]:
```
# Your code here.
p_SM_T_HD_T = get_prob(X='Sm', e={'HD':T}, bn=bnHeart)
print("P(Sm=T|HD=T)= ", p_SM_T_HD_T.prob[T])
```

```
P(Sm=T|HD=T)=  0.22096327904719273
```

[v] How would you expect the probability in [iv] to change if you also know the individual has `High Blood Pressure`? First write your answer, then verify your hypothesis by calculating the relevant probability.

Here were are comparing $P(Sm = T \mid HD = T)$ vs $P(Sm = T \mid HD = T, HBP = T)$

I would expect $P(Sm = T \mid HD = T, HBP = T)$ to be a larger proabilibty than $P(Sm = T \mid HD = T)$. This is because the model shows that High Blood Pressure is an effect

(child) of Smoker/Alcohol User, thus having more effects would be more evidence that the individual does Smoke/Alcohol.

In [19]:
```
# Your code here.
P_verify_v = get_prob(X='Sm', e={'HD':T, 'HBP': T}, bn=bnHeart)
print("P(Sm=T|HD=T,HBP=T)= ", P_verify_v.prob[T])
```

```
P(Sm=T|HD=T,HBP=T)=  0.282051282051282
```

[vi] How would you expect the probability in [v] to change if you also know that the individual does *not* get `Moderate Exercise` (in addition to having `Heart Disease` and `High Blood Pressure`)? Explain your answer using concepts from class. First write your answer, then verify your answer by calculating the relevant probability.

Here were are comparing $P(Sm = T \mid HD = T, HBP = T)$ vs.
$P(Sm = T \mid HD = T, HBP = T, ME = F)$

I would expect $P(Sm = T \mid HD = T, HBP = T, ME = F)$ to be smaller than $P(Sm = T \mid HD = T, HBP = T)$. This is because the model shows that Moderate Exercise is a cause of High Blood Pressure which is an effect of Smoke/Alcohol. Thus, since $ME = F$ we are adding evidence AGAINST another cause for High Blood Pressure, which reduces the probability they are a Smoker/ALcohol.

In [20]:
```
# Your code here.
P_verify_vi = get_prob(X='Sm', e={'HD':T, 'HBP': T, 'ME': F}, bn=bnHeart)
print("P(Sm=T|HD=T,HBP=T,ME=F)= ", P_verify_vi.prob[T])
```

```
P(Sm=T|HD=T,HBP=T,ME=F)=  0.26086956521739124
```