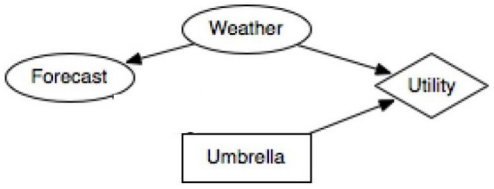
CSCI 561: Foundations of Artificial Intelligence
Homework #3: Decision Network
Due at 11:59pm on 4/17/2017

Introduction

A decision network (Chapter 16, AIMA) uses a Directed Acyclic Graph (DAG) to represent a set of random variables and their conditional dependencies within a probabilistic model, while a decision network extends the Bayesian network to include decision nodes and utility nodes. As in the example given in Figure 1, there are three types of nodes: Rectangles represent ***decision nodes***, Ovals represent ***chance nodes***, and Diamonds represent ***utility nodes***. In this assignment, you will write code to perform inference in Decision Networks of discrete variables.



Figur

Assignment

You will be given a decision network, which may have several decision nodes, several chance nodes, and at most one utility node. You will be asked to answer queries using the given network:

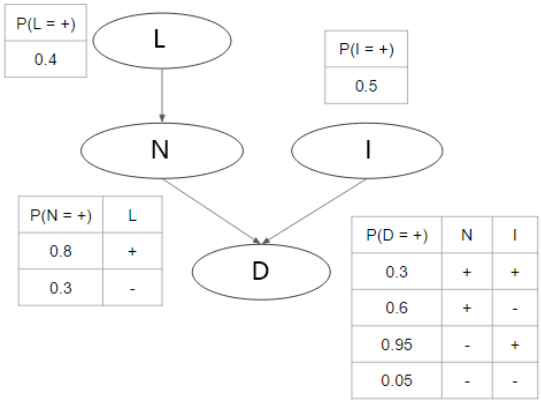
1) Calculate a specific joint, marginal, or conditional probability. (~50%)
2) Calculate the expected utility of a particular decision, or determine the decision with the maximum expected utility. (~50%)

Pseudocode

AIMA Figure 14.9, Section 16.5.2

Input

You will be given a text file ending with a **.txt** extension. For example, the decision network in Figure 2 would be represented by the following file, sample01.txt:



(sample01.txt)

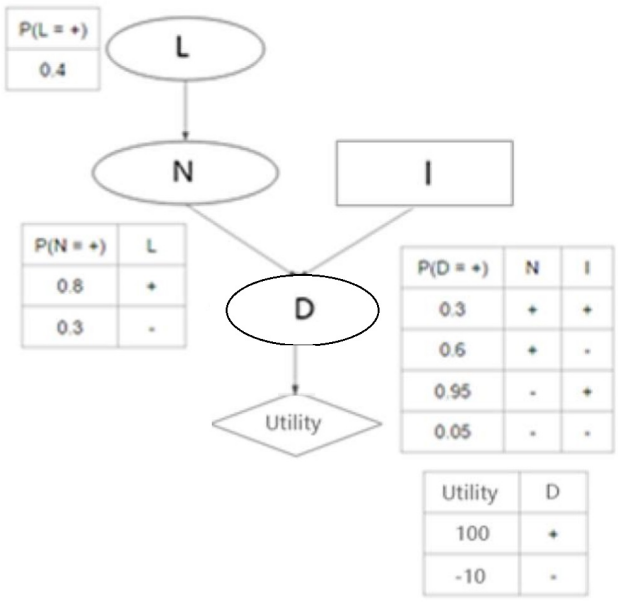
```
P(N = +, I = -)
P(D = + | L = +, I = +)
******
L
```

```
0.4
***
N | L
0.8 +
0.3 -
***
I
0.5
***
D | N I
0.3 + +
0.6 + -
0.95 - +
0.05 - -
```

An example of a decision network having decision nodes and one utility node is given in Figure 3, and it can be represented by sample 02.txt:

(sample02.txt)

```
P(D = + | L = -, I = +)
EU(I = +)
EU(I = + | L = +)
MEU(I)
MEU(I | L = +)
******
L
0.4
***
N | L
0.8 +
0.3 -
***
I
decision
***
D | N I
0.3 + +
0.6 + -
0.95 - +
0.05 - -
******
utility | D
100 +
-10 -
```



Figur

Input

Each line before the "******" will contain a single query. The query can have three forms, signaled by the function name beginning the query:

1) Queries beginning with "P" are asking for a specific joint, marginal, or conditional probability. Any decision node in the network will appear as evidence in the query:
   e.g. `P(N = +, I = -)`
   `P(D = + | L = +, I = +)`
2) Queries beginning with "EU" are asking for expected utility over one or more decision nodes, conditioned on the given observed evidence:
   e.g. `EU(I = +)`
   `EU(I = + | L = +)`
   `EU(L = -, I = +)`
3) Queries beginning with "MEU" are asking for maximum expected utility over one or more decision nodes, conditioned on the given observed evidence:
   e.g. `MEU(I)`
   `MEU(I | L = +)`
   `MEU(L, I)`

The line after all queries will have six "*" as the separator.
The lines following the separator represent the Bayesian or decision network by showing the tables of probabilities / conditional probabilities for each node. Each tables is separated by a line with three asterisks ("***"), and will have the following format:

e.g.
```
D | N I
0.3 + +
0.6 + -
0.9 - +
0.05 - -
```

- The first line contains the node's name. If the node has any parents, its name will be followed by a "|" and then the names of its parents (all separated by spaces).
- All node names begin with an uppercase letter and contain only letters. You can assume that each name has at most 20 letters.
- The remaining lines specify the probabilities that the child node is True, over all combinations of values of its parent nodes. The probability that the child node is False is simply 1−the given probability that it is True.
- **All nodes can have only two values, "+" (True) or "−" (False)**.
- The probability will range from 0 to 1.
- The parent node values follow the order in which they appear in the first line.
- However, with the exception of the first line, there is no specific order between lines of the table (e.g. "`+ -`" may appear after "`- +`", or the other way around).
- When a node has no parent, then there is only a single number (probability of True):

e.g.
```
L
0.4
```

- Every node has a corresponding table in the input file, so you can know the network structure from the first line of each table. For example, your program could figure out the network structure in *sample01.txt* based on solely the following information:

```
L
N | L
I
D | N I
```

- There will not be any directed cycles in the given networks.
- Parent nodes always have their tables appearing before the child node.
- A decision node will not have a parent node or a probability table; there will be only the word “decision” (non-capitalized) on the second line to mark it as a decision node.
- It is possible for there to be multiple decision nodes. For example, “L” might be made a decision node as well in sample01.txt. You may assume there are at most 3 decision nodes.
- When asking for probability in a query, the values of all decision nodes will be given as conditions.
- With at least one decision node in the network, there will always be a utility node as well, and its utility table will be given at the end of the input file, separated from others by six “*” (see sample02.txt for an example).
- The format of a utility table is similar to that of a normal node with parent. The first line begins with the word “utility” (non-capitalized), followed by a “|” sign, then followed by its parent nodes.
- The remaining lines show the utility value over all combinations of parent node values.
- Every utility value is an integer, possibly negative.
- There will be at most one utility node. If there is a utility node, it will have no more than 3 parent nodes.

Output

The result should be printed to a file called **output.txt**. Given the sample input above, the output content should be as follows:

(output.txt for input sample01.txt)

```
0.25
0.43
```

(output.txt for input sample02.txt)

```
0.76
59
37
+ 59
- 44
```

For each query in the input file, your program should generate a corresponding result (one result per line) as the output. The result may have three forms:

- "P" query: A decimal value between 0 and 1, **rounded to two decimals** (for example, we want 0.395 to be 0.40):
  e.g. `0.40`
  1) "EU" query: An integer value:
  e.g. `50`
  2) "MEU" query: One sign("+" or "-") for each decision node, followed by an integer value representing the maximum expected utility, all separated with **a single whitespace**:
  e.g. `+ 50`

- When there are multiple decisions, the order of decisions should be the same as in the query.
  e.g. Input: `MEU(I, L)`
- Output: `+ - 50`
- The test cases will be designed so that there will always be one unique solution with MEU.
- **For EU and MEU queries, all calculations should be done in decimal number accuracy, but the output expected utility value should be rounded to the nearest integer** (for example, we want 3.5 to be rounded to 4).
- **Don't** print additional whitespace after the value, or extra line break in the end.

Grading Notice

- Please follow the instructions carefully. Any deviations from the instructions may lead your grade to be zero for the assignment. If you have any questions, please use the discussion board. Do not assume anything that is not explicitly stated.

- You must use Python 2.7 to implement your code. You are allowed to use standard libraries only. You have to implement any other functions or methods by yourself.

- You need to create a file named "**hw3cs561s2017.py**". The command to run your program would be as follows: (When you submit the homework on labs.vocareum.com, the following command will be executed for grading your homework)
  python hw3cs561s2017.py
- You need to read **input.txt as infile = open("input.txt", "r"),** in your code**.**
- **Your code should generate the output exactly named "output.txt"**
- **The grading script will paste an input.txt file in your work folder, run your code, and compare the generated output.txt from your code with the correct answer (This will be done 50 times for 50 test cases).**

- You will use labs.vocareum.com to submit your code. Please refer to http://help.vocareum.com/article/30-getting-started-students to get started with the system.

Please only upload your code to the “/work” directory. Don’t create any subfolders or upload any other files.

- If we are unable to execute your code successfully, you will not receive any credit.

- For your final submission, please do not print any logs on the console other than the required output.

- When you press “Submit” on Vocareum, your code will be run against the submission script and the sample input/output files. You will receive feedback on where your code is making errors, if any. You can click “Submit” to test new versions of your code as many times as you like up until the deadline. Only your last submission will be graded. The sample input/output files are designed to cover many basic situations of the problem and rules of the output log, so please utilize them as much as you can.

- Your program should handle all test cases within a reasonable time (not more than a few seconds for each sample test case). The complexity of the grading test cases will be similar to or less than the five example test cases.

- You are strongly recommended to submit at least **two hours ahead of the deadline**, as the submission system around the deadline might be slow and possibly cause late submission, and late submissions will **NOT** be graded.