Middle East Technical University

Department of Computer Engineering

CENG 111

Fall 2016

Take Home Exam 4

REGULATIONS

Due date: 23:59, 24 January 2017, Tuesday *(Not subject to postpone)*

Submission: Electronically. You should save your program source code as a text file named `the4.py` and submit it to us via the course's COW page.

Team: There is **no** teaming up. This is an EXAM.

Cheating: Source(s) and Receiver(s) will receive zero and be subject to disciplinary action.

INTRODUCTION

A *decision tree* is a binary tree where internal nodes have boolean expressions as datum and each external node has as datum a final decision (outcome) value. Each of the pair of branches emerging from a node are labeled as True and False, correspondingly. The boolean expressions at the internal nodes have variables in them.

When you are asked to *query the decision tree* with a set of variable value settings,

1. you start at the root node.
2. If the node you are at is an internal node then consider the boolean expression of the node. Evaluate it with the given variable settings. if the outcome is True proceed to the True branch of the node, otherwise to the False branch.
3. Else, if the node is an exterior node (a leaf node), you have reached the end. The datum of the node is the outcome, return it.
4. Continue from item (2).

In this Take Home Exam you will be given a decision tree and a set of variable settings where you are expected to query the tree and return the outcome.

PROBLEM & SPECIFICATIONS

You will write a function with the name `query` that has exactly two arguments.

The 1st argument will receive a list that is the representation of the decision tree.

The 2 nd argument will receive a list of two-tuples. Each two-tuple is representing the value setting of one variable and is of the form:
$(variable_i, value_i)$ $variable_i$ is a string. $value_i$ is either a number or a string.

Example tuples:

```
("temperature","high")
("Systolic Blood Pressure",13.5)
("Limit",12.5)
("n",1300)
("m",1200)
```

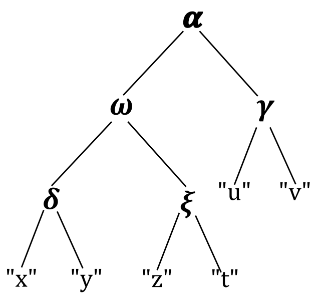
In this THE, the usage of the world *variable* refers to a string that stands for a variable in the boolean expressions and have value associated through the 2nd argument of the query function (as explained above).
Example: `"x"` , `"n"` , `"m"` , `"Limit"`, `"Systolic Blood Pressure"`, `"temperature"` are *variables* according to the example above.

The list representation of the binary tree is trivial and is defined as:

Internal node: The representation of the node datum is the first element of a ternary (a list with three elements) list. For the internal node datum is a boolean expression in prefix form and is explained below). The representations for the left child and right child are located at the second and third element, respectively, of the ternary list.

External node: The representation of the leaf node datum is a string. It is the outcome value if the query ends at that node.

An example decision tree of the structure:



where $\boldsymbol{\alpha}$,$\boldsymbol{\omega}$,γ,$\boldsymbol{\delta}$,$\boldsymbol{\xi}$ are boolean expressions and "x","y","z","t","u","v" are external nodes (outcomes), left branchings stand for the True case and right branchings stand for the False case, will be represented as:

[$\boldsymbol{\alpha}$,[$\boldsymbol{\omega}$,[$\boldsymbol{\delta}$,"x","y"],[$\boldsymbol{\xi}$,"z","t"]],[γ,"u","v"]]

(Note that external nodes are not represented by lists (This representation was explained in the lecture). They are merely strings (each correspond to a specific outcome)).

The prefix form *boolean expression* which is the datum for the internal node is a list. The first element of this list is a string representing an operator. The following possibilities exist.

"==" : The binary operation of testing for equality. It is followed by two list elements one of which is a variable, and the other is either a variable or a numerical or string value.
Example: `["==","x",1.25]` , `["==",10,"n"]` , `["==","m","n"]` , `["==","temperature","high"]`

"!=" : Similar to `"=="` but tests for inequality.

"<" : The binary operation of testing for being lesser. It is followed by two list elements one of which is a variable that has a numeric value, and the other is either a number or variable that has a numeric value. (Comparison is only among numerical values.)
Example: `["<","x",1.25]` , `["<",1.25,"x"]` , `["<","Systolic Blood Pressure",13.5]`, `["<","Systolic Blood Pressure","Limit"]`

`">"` **:** Similar to `"<"` but tests for being greater.

`"in"` **:** The binary operation of testing for an element (which is the variable) to be present in a list. It is followed by two list elements the first of which is a variable the second is list in which the search will be performed.
Example: `["in","x",[2,3,5,7,11]]` , `["in","temperature",["very low","low","normal"]]`

`"and"` **:** Nary logical conjunction operation. The following elements of the list are boolean expressions that are subject to conjunction.
Example: `["and",["in","x",[2,3,5,7,11]], ["==","temperature","high"]]`

`"or"` **:** Nary logical disjunction operation. The following elements of the list are boolean expressions that are subject to disjunction.
Example: `["or",["==",10,"n"],["and",["<","x",1.2],["==","m","n"],["<","SBP",13.5]]]`

`"not"` **:** Unary operation of logical negation. The (single) following element is the boolean expressions that will be negated.

Your implementation of the query function will walk through the decision tree which is given as the first argument to it, by taking the list of variable-value associations given as the second argument into account. This walk will finally arrive at an external node (which is a string). The query function will return this string.

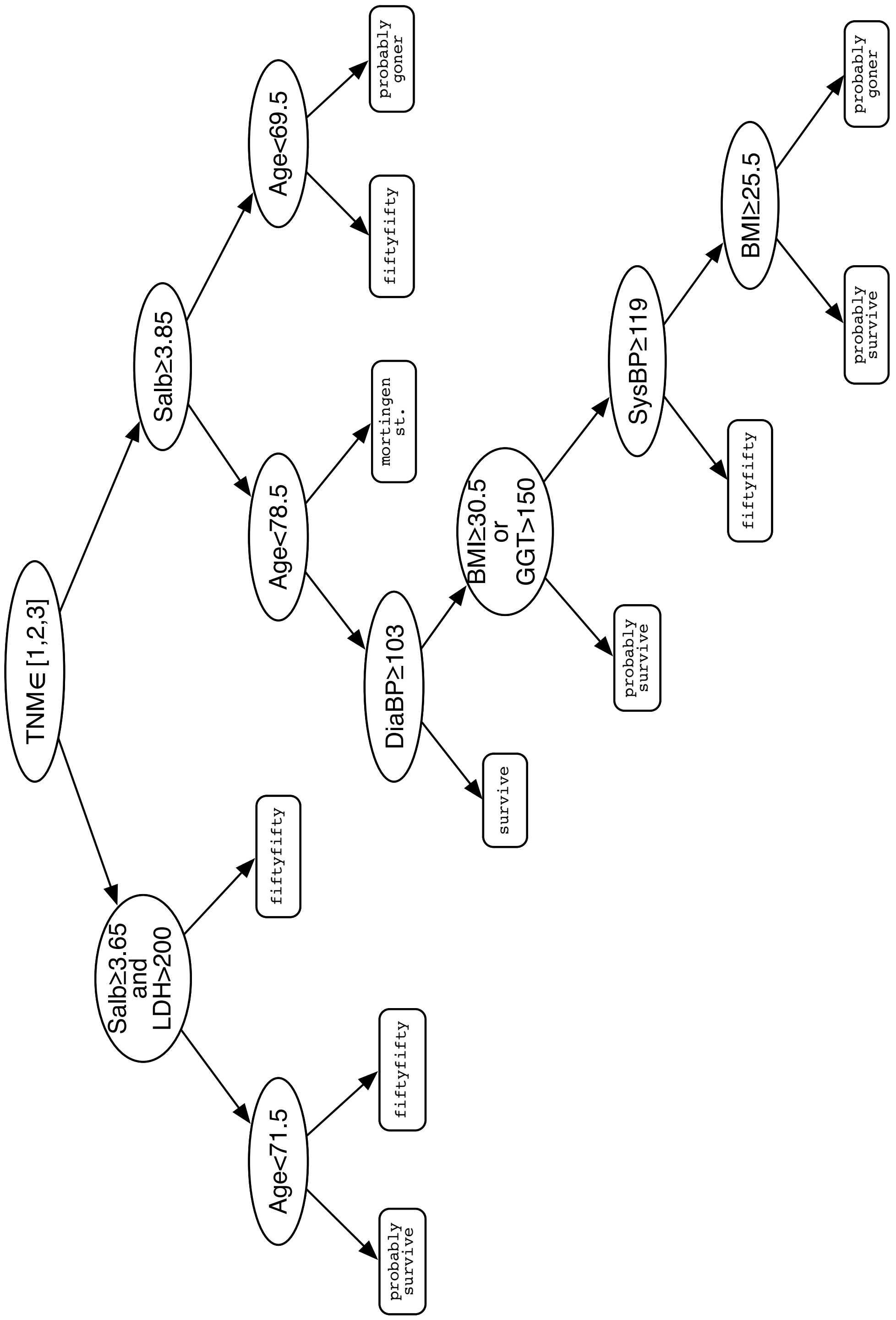
You are allowed to code helper functions as you wish. Also you are allowed to use any standard Python package that comes with it (Actually you don't need any of them).

All string operations involved in this THE are case sensitive.

GRADING

- Comply with the specifications. Your program will implement the function `query`. It will neither read nor print anything.
- During testing your query function will be called by the evaluating script with appropriate arguments and the return value will be expected for correctness.
- Your program will be graded through an automated process.
- Multiple evaluation runs for your query function will be performed.
- Do not attempt to generate random return values. A distinct inspection for this will be performed.
- You do not have to check for erroneous arguments to `query`.
- Any program that performs only 30% and below will enter a glass-box test (eye inspection by the grader TA). The TA will judge an overall THE4 grade in the range of [0,30]. The glass-box test grade is not open to discussion nor explanation.

AN EXAMPLE RUN



```
>>> decision_tree = [
 ['in', 'TNM', [1, 2, 3]],
 [['and', ['not', ['<', 'Salb', 3.65]], ['>', 'LDH', 200]],
  [['<', 'Age', 71.5], 'probably survive', 'fiftyfifty'],
  'fiftyfifty'],
 [['not', ['<', 'Salb', 3.85]],
  [['<', 'Age', 78.5],
   [['not', ['<', 'DiaBP', 103]],
    'survive',
    [['or', ['not', ['<', 'BMI', 30.5]], ['>', 'GGT', 150]],
     'probably survive',
     [['not', ['<', 'SysBP', 119]],
      'fiftyfifty',
      [['not', ['<', 'BMI', 25.5]], 'probably survive', 'probably goner']]]]],
   'mortingen st.'],
  [['<', 'Age', 69.5], 'fiftyfifty', 'probably goner']]
]

>>> variable_value = [
   ('TNM',4),
  ('Salb',3.9),
  ('LDH',210),
  ('Age',66),
  ('DiaBP',96),
  ('BMI',27.8),
  ('GGT',157),
  ('SysBP',105)
 ]

>>> query(decision_tree,variable_value)
'probably survive'
```