CSCI 4150 Introduction to Artificial Intelligence, Fall 2001
Assignment 6 (90 points): out Monday November 5, due Monday November 19

Late policy deadlines are as follows: automatic extension until Monday November 19, 5pm for Problem 1 (written) and midnight for the remaining problems; first tier deadline is Tuesday November 20, 5pm for Problem 1 and midnight for the remaining problems; second tier deadline is 10am Monday November 26 for all problems.

Problems

1. (30 points) Bob went snorkeling 12 times and recorded the following data:

Example No.	Tide	Skies	Waves	Time	Good snorkeling?
1	Low	Cloudy	Large	Afternoon	Yes
2	Low	Sunny	Medium	Morning	Yes
3	Low	Sunny	Large	Afternoon	No
4	Low	Cloudy	Small	Afternoon	Yes
5	Low	Sunny	Medium	Morning	No
6	High	Sunny	Large	Afternoon	No
7	High	Cloudy	Large	Afternoon	Yes
8	High	Sunny	Small	Morning	Yes
9	High	Sunny	Medium	Morning	Yes
10	High	Cloudy	Medium	Afternoon	Yes
11	High	Sunny	Small	Afternoon	No
12	High	Sunny	Small	Morning	Yes

   Suppose we start learning a decision tree on the above data using "Good snorkeling?" as the goal predicate. Calculate the information gains for splitting the data for each of the four attributes. Which attribute provides the largest information gain (and would therefore be the top level attribute in a decision tree)? Show and explain your work.

2. (50 points) For this problem, you are to write the procedure:

```
(learn-dtree training-data attribute-names)
```

   which should return a decision tree. The representations I have chosen for decision trees and training data are described in subsequent sections. I am providing support code to do some of the more mundane data manipulation tasks. You may structure the `learn-dtree` procedure however you like, but I will make a few suggestions. I am also providing a few data sets for you to test your code.

⋆3. (10 points) A challenge problem involving discretizing continuous valued attributes. See the assignment web page for details.

Scheme representations & support code

I have chosen representations for training data and decision trees. In addition, I am providing support code for a number of data manipulation tasks. These are described in the following sections.

Training data and examples

Training data consists of a list of training examples, and a *training example* is a list where the first element is the value of the goal predicate (which can be any symbol, not just `yes` or `no`) and the second element is a list of the attribute values. We will also require a list of *attribute names* so we can refer to attributes by name.

Here are the first four training examples from the first problem made into a training data set:

```
(define snorkel-names
  '(tide skies waves time))
(define snorkel-data-small
  '((Yes (Low  Cloudy Large  Afternoon))
    (Yes (Low  Sunny  Medium Morning))
    (No  (Low  Sunny  Large  Afternoon))
    (Yes (Low  Cloudy Small  Afternoon))))
```

Note that the goal predicate is not explicitly named.

Decision trees

A decision tree is either a value for the goal predicate or a list of the following form:

```
(<attribute-name> (<attribute-value-1> <decision-tree-1>)
                  ...
                  (<attribute-value-n> <decision-tree-n>))
```

For the snorkel example, a valid decision tree is:

```
(define snorkel-dtree-example
  '(tide (high yes)
         (low (waves (small  no)
                     (medium yes)
                     (large  yes)))))
```

Support code

You need not use the following procedures, but you will probably find them helpful. They are organized into several different categories in the sections below.

Handling training data

- `(split-tdata training-data attribute-names attribute)`

  This function divides the training data into groups according to the specified attribute. For example, using the training data above:

  ```
  (split-tdata snorkel-data-small snorkel-names 'waves)
  ;Value: ((small  ((yes (low cloudy small  afternoon))))
           (medium ((yes (low sunny  medium morning))))
           (large  ((no  (low sunny  large  afternoon))
                    (yes (low cloudy large  afternoon)))))
  ```

  This procedure returns a list of what I refer to as *splits*. Each split is a list whose first element is a value of the attribute and whose second element is a subset of the training data which all have that value for the given attribute.

- `(tally-tdata training-data)`

  This function tallies up how many instances there are of each value of the goal predicate, returning a list of lists; the first element of each sublist is the value of the goal predicate, and the second element is the number of training examples with that value.

  For example,

  ```
  (tally-tdata snorkel-data-small)
  ;Value: ((no 1) (yes 3))
  ```

  Do not assume that the goal predicate will always have the values “yes” and “no”!

- `(pick-majority tally)`

  Given a tally as returned by `tally-tdata`, returns the majority value. If there is a tie, it returns the first instance it finds. For example:

  ```
  (pick-majority (tally-tdata snorkel-data-small))
  ;Value: yes
  ```

Testing your decision trees

- `(classify example decision-tree attribute-names default-value)`

  This function returns the classification for the example; if it encounters an attribute value that is not in the decision tree, then it will return the default-value. For example:

  ```
  (classify (second (fourth snorkel-data-small))
            snorkel-dtree-example snorkel-names 'attribute-value-not-found)
  ;Value: no

  (classify '(low sunny huge afternoon)
            snorkel-dtree-example snorkel-names 'attribute-value-not-found)
  ;Value: attribute-value-not-found
  ```

  We would generally pick a value of the goal predicate to be the `default-value`.

  Note that `example` is just a list of attribute values (i.e. no goal predicate value).

- `(test-dtree decision-tree training-data attribute-names)`

  This function takes a decision tree and a set of training data. From the training data, it creates a list of examples and a list of correct classifications. It classifies all the examples using the decision tree, compares the results to the correct classifications, and reports the results.

Miscellaneous utility functions

- `(remove-element el Lst)`

  Removes the first occurence of `el` in `Lst`, returning a newly created list.