CSC326: Assignment 1

Deadline: 16 Oct 2016, 23:59

Problem 1. Consider a recursive, pure function (a function without side effect) like *choice* below, which computes the the number of ways for choosing k elements out of n elements:

```
1  def choice(n,k):
2    if k==0 or k==n:
3      return 1
4    else:
5      return choice(n-1,k)+choice(n-1,k-1)
```

A common strategy to speed up the evaluation of such functions is to employ dynamic programming: recognizing that the return values of pure function calls are always the same for the same input argument value, one could remember the computation result for an input value the first time a call for that value is made, and use it later, thus avoiding redundant calculations. For example, when calculating $choice(5,3)$, calls to $choice(4,1)$ will be made multiple times. As explained, dynamic programming can eliminate such redundant computations.

Create a file **q1.py** and using **Python dictionary and dynamic programming**, write a functions $choice(n,k,l)$ where n is the total number of items and k is the number of items to be chosen. The function should return the number of ways of choosing k items from n items. The function should also append the tuple (n,k) to the list l. For example, $l = []; choice(4,2,l)$ should return 6 as the value and the list l would be:

$l = [(4,2),(3,2),(2,2),(2,1),(1,1),(1,0),(3,1),(2,0)]$

Problem 2. Consider a list of tuples of size two, (s,i), where s, is a string and i is the number of rotations to the left: $l = [(s_1,i_1),(s_2,i_2),(s_3,i_3),...]$

Create a file **q2.py** and write a function $rotate(l)$ that accepts such list as input and returns the list of rotated words. For example sending list:

$list = [('Hello',2),('abc',7),('wait',3),('programming',15)]$

should return:

$['lloHe','bca','twai','rammingprog']$

Note: If the number of rotations is larger than the string size the function should continue to rotate to that number.

Hint: You should use **slicing** and **list comprehension**.

Problem 3. A Python function $factorize(n)$ takes an integer as an argument, and returns an ordered list of prime integers whose product equals to n. For example, $factorize(6)$ returns $[1, 2, 3]$.

Create a file **q3.py** containing the implementation of the function $factorize(N)$ and the unit tester for the function.

Note: You should follow **test-driven design methodology** and use the **doctest framework**.

Problem 4. Create a file **q4.py** and without using loops and conditional statements, write a Python function $saturated\_abs\_sum(a, b, max)$, where a and b are 2 dimensional numpy arrays of real numbers with the same shape, and max is a scalar real number. The function should return a numpy array c with the same shape as a and b where:

$$c_{i,j} = \begin{cases} |a_{i,j}| + |b_{i,j}| & |a_{i,j}| + |b_{i,j}| < max \\ max & |a_{i,j}| + |b_{i,j}| >= max \end{cases} \tag{1}$$

Problem 5. *Reduce* is an operator we learned in class. It can be applied on any associate, binary function operating on two scalars resulting another function operating on an array of scalars. For example, applying *reduceadd* on an array of values $array([4, 6, 7])$ leads to the result 17.

1. Consider a $n \times n$ matrix a of real numbers and a vector x of size n. Create a file **q5-1.py** and use numpy to write a function product(a,x) to compute their product $P = ax$ where:

$$P_i = \sum_{j=0}^{n} (a_{i,j} * x_j) \tag{2}$$

   Note: It is required to use the *reduce* operator.

2. Given a weighted graph with n nodes, its adjacency matrix is a matrix a with $a_{i,j}$ equals to the weight of the edge between i and j and 0 if there is no edge. Create a file **q5-2.py** and use numpy to write a function degree(a) to calculate vector x of size n where x_i represents the degree of the node i, *i.e.* the number of node i's neighbours.

   Note: The input matrix is symmetric since the graph is undirected.

3. Bellman-Ford algorithm is for finding the shortest path in a graph. Create a file **q5-3.py** and using numpy and reduce operation, write a function $bellman\_ford(a)$ which a is a $n \times n$ adjacency matrix of a directed weighted graph with n vertices. The function should return a list containing the shortest path from the first node to all nodes.

Submission Instructions Compress all of your files and save them either as <Your UTORID>.tar.gz and submit the files from a computer in one of the following labs: GB243, GB251E, or SF2102, or by SSH by logging on to one of the aformentioned computers (ug132.eecg - ug180.eecg, ug201.eecg - ug249.eecg).

Submission Command Example

```
ug132:~% submitcsc326f 1 <Your UTORID>.zip
```