Walter Cazzola

Home Page
ADAPT Lab.
Curriculum Vitae
Research Topic

Didactics

Publications

Funded Projects

Research Projects

Related Events

Exam of Advance in Programming

22 July 2010 (Solutions)

Exercise 1: To Speed up Recursion.

```
  def wrapper(*args):
    if not args in wrapper.cache:
      wrapper.cache[args] = f(*args)
    else:
      print("### cached value for {0} --> {1}". \
                    format(args, wrapper.cache[args]))
    return wrapper.cache[args]
  wrapper.cache = dict()
  return wrapper

@memoization
def fact(n):
  return 1 if (n<=1) else n*fact(n-1)

@memoization
def fibo(n):
  return n if n<=1 else fibo(n-1)+fibo(n-2)

@memoization
def sum(n, m):
  return n if m==0 else sum(n+1, m-1)
```

Exercise 2: Pascal's Triangle.

```
class PascalTriangleRowIterator:
  def __init__(self, position, start=0):
    self.k = start
    self.n = position
    self.fact_n = fact(position)
  def __iter__(self):
    return self
  def __next__(self):
    if self.k > self.n:
      raise StopIteration
    tmp = self.fact_n//(fact(self.k)*fact(self.n-self.k))
    self.k = self.k +1
    return tmp
  def prev(self):
    if self.k <= 0:
      raise StopIteration
    self.k = self.k - 1
    return self.fact_n//(fact(self.k)*fact(self.n-self.k))

class PascalTriangleIterator:
  def __init__(self, n, start=0):
    self.dimension = n
    self.position = start
  def __next__(self):
    if self.position >= self.dimension: raise StopIteration
    self.position = self.position+1
    return PascalTriangleRowIterator(self.position-1)
  def prev(self):
    if self.position <= 0: raise StopIteration
    self.position = self.position - 1
    return PascalTriangleRowIterator(self.position,self.position+1)

class PascalTriangle:
  def __init__(self, n, start=0):
    self.rows = n
    self.start=start
  def __iter__(self):
    return PascalTriangleIterator(self.rows, self.start)
```

Exercise 3: Testing Algebra (Again).

```
import unittest

def make_monoid(S, i, op1):
  S.identity=i
  S.__add__ = op1
  return S

def make_tests(monoid, name):
  class check_monoids(unittest.TestCase):
    def test_closure(self):
      print("### Checking Closure on {0}: ".format(name))
      checks = [(x+y).inSet()
         for x in monoid.signature()
            for y in monoid.signature()]
      self.assertTrue(all(checks))
    def test_associativity(self):
      print("### Checking Associativity on {0}: ".format(name))
      checks = [((x+y)+z)==(x+(y+z))
         for x in monoid.signature()
            for y in  monoid.signature()
               for z in monoid.signature()]
      self.assertTrue(all(checks))
    def test_identity(self):
      print("### Checking Identity {1} on {0}: ". \
                        format(name, monoid.identity))
      checks = [monoid.identity in monoid.signature()]
      for x in monoid.signature():
        checks += [(monoid.identity+x) == x]
      self.assertTrue(all(checks))
  return check_monoids

class Z7:
  def __init__(self, val): self.value = val
  def __eq__(self, other): return self.value == other.value
```

```
  def __repr__(self): return str(self.value)
  def signature(): return [Z7(x) for x in range(7)]
  def inSet(self): return (self in Z7.signature())

def firstn(g, n):
   for i in range(n):
      yield next(g)

def gabstar():
  yield ABstar("")
  strings = ["a", "b"]
  while True:
    for elem in strings: yield ABstar(elem)
    tmp = []
    for elem in strings:
      tmp += [elem+"a"]+[elem+"b"]
    strings=tmp

def ngen():
  val = 0
  yield N(val)
  while True:
    val += 1
    yield N(val)

class ABstar:
  def __init__(self, str): self.value = str
  def __eq__(self, other): return self.value == other.value
  def __repr__(self): return self.value
  def signature(n=50): return firstn(gabstar(),n)
  def inSet(self): return (self in gabstar())

class N:
  def __init__(self, val): self.value = val
  def __eq__(self, other): return self.value == other.value
  def __repr__(self): return str(self.value)
  def signature(n=50): return [N(x) for x in range(n+1)]
  def inSet(self): return (self in ngen())

monoid_z7 = make_monoid(Z7, Z7(1),
    lambda x,y: Z7(0 if (y.value == 0) else (x.value//y.value)%7))
test_z7 = make_tests(monoid_z7, "(Z7, *)")

monoid_abstar = make_monoid(ABstar, ABstar(""),
    lambda x,y: ABstar((x.value)+(y.value)))
test_abstar = make_tests(monoid_abstar, "(AB*, *)")

monoid_n = make_monoid(N, N(0), lambda x,y: N(x.value+y.value))
test_n = make_tests(monoid_n, "(N, +)")

all_tests = [test_n, test_abstar, test_z7]
suite = unittest.TestSuite()

if __name__ == "__main__":
  for tc in all_tests:
    suite.addTests(unittest.TestLoader().loadTestsFromTestCase(tc))
```

Last Modified: Fri, 23 Jul 2010 12:11:25 ADAPT Lab.