Lab3

Data Structures and Algorithms

Maira usman ¶

21B-011-SE

In [19]:
```
class Set:
    def __init__(self,*initElements):
        self._elements= list(initElements)
    def __len__(self):
        return len(self._elements)
    def __contains__(self,item):
        return item in self._elements
    def add(self,item):
        if item not in self:
            self._elements.append(item)
    def remove(self,item):
        assert item in self,"element must be in the set"
        self._elements.remove(item)
    def __eq__(self,setB):
        if len(self)!= len(setB):
            return False
        else:
            return self.isSubsetOf(setB)
    def isSubsetOf(self,setB):
        for item in self:
            if item not in setB:
                return False
        return True
    def union(self,setB):
        newSet=Set()
        newSet._elements.extend(self._elements)
        for item in setB:
            if item not in self:
                newSet._elements.append(item)
        return newSet
    def intersect(self,setB):
        newSet=Set()
        for item in self:
            if item in setB:
                newSet._elements.append(item)
        return newSet
    def difference(self,setB):
        newSet=Set()
        for item in self:
            if item not in setB:
                newSet._elements.append(item)
        return newSet
    def ProperSet(self,setB):
        if len(self._elements) !=len (setB):
            newSet=self.intersect(setB)
            if len (newSet ) == len(self):
                return True
        else:
            return False


    def __iter__(self):
        return _SetIterator(self._elements)
class _SetIterator:
    def __init__(self,elements):
        self._elements=elements
        self._curNdx=0
    def __iter__(self):
        return self
    def __next__(self):
        if self._curNdx <len(self._elements):
            entry = self._elements[self._curNdx]
            self._curNdx+=1
            return entry
        else:
            raise StopIteration



A=Set(150,75,23,86,49)
B=Set(150,75,23,86,49,20,25)

if A == B:
    print("A and B are equal")
else:
    sameelements=A.intersect(B)
if len(sameelements)==0:
    print("A and B does not have same elements")
else:
    print("A and B have some elements common  ")
for element  in sameelements:
    print(element)
uniqueelements=B.difference(A)
print("letters that are not in B ")
for elements in uniqueelements:
    print(elements)
print("elementsletters that are in A & B ")
allelements=A.union(B)
for elements in allelements:
    print(elements)
print(A.ProperSet(B))
```

```
A and B have some elements common
150
75
23
86
49
letters that are not in B
20
25
elementsletters that are in A & B
150
75
23
86
49
20
25
True
```

In [22]:
```
class Map:
    def __init__(self):
        self._list=list()
    def __len__(self):
        return len(self._list)
    def __contains__(self,key):
        ndx=self._findposition(key)
        return ndx is not None
    def add(self,key,value):
        ndx=self._findposition(key)
        if ndx is not None:
            self._list[ndx].value=value
            return False
        else:
            entry=_PairEntry(key,value)
            self._list.append(entry)
            return True
    def valueOf(self,key):
        ndx=self._findposition(key)
        assert ndx is not None,"Invalid map key"
        return self._list.pop(ndx).value
    def remove(self,item):
        ndx=self._findposition(key)
        assert ndx is not None,"Invalid map key"
        return self._list.pop(ndx)
    def __iter__(self):
        return _MapIterator(self._list)
    def _findposition(self,key):
        for i in range(len(self)):
            if self._list[i].key==key:
                return i
        return None
    def keyArray(self):
        lst=[]
        for i in range(len(self)):
            lst.append(self._list[i].key)
        return lst


class _PairEntry:
    def __init__(self,key,value):
        self.key=key
        self.value=value

class _MapIterator:
    def __init__(self,elements):
        self._elements=elements
        self._curNdx=0
    def __iter__(self):
        return self
    def __next__(self):
        if self._curNdx <len(self._elements):
            entry = self._elements[self._curNdx]
            self._curNdx+=1
            return entry
        else:
            raise StopIteration
a=Map()
a.add("maira",19)
a.add("musfira",20)
a.add("hadiqa",19)
print(a.keyArray())
```

```
['maira', 'musfira', 'hadiqa']
```

In [1]:
```
import ctypes
import sys
sys.path.append('../')
import random
class Array:
    def __init__(self,size):
        assert size>0 ,"Array size mustbe > 0"
        self._size=size
        PyArrayType=ctypes.py_object*size
        self._elements=PyArrayType()
        self.clear(None)
    def __len__(self):
        return self._size
    def __getitem__(self,index):
        assert index>=0 and index<len(self),"Array index out of range"
        return self._elements[index]
    def __setitem__(self,index,value):
        assert index>=0 and index<len(self),"Array index out of range"
        self._elements[index] =value
    def clear(self,value):
        for i in range(len(self)):
            self._elements[i]=value
    def __iter__(self):
        return _ArrayIterator(self._elements)
class _ArrayIterator:
    def __init__(self,Array):
        self._arrayRef=Array
        self.index=0
    def __iter__(self):
        return self
    def __next__(self):
        if self._index < len(self._arrayRef):
            entry=self.arrayRef[self.index]
            self.index+=1
            return entry
        else:
            raise StopIteration
class MultiArray:
    def __init__(self, *dimensions):
        assert len(dimensions) > 1, "Dimensions can not be less than 1"
        self._dims = dimensions
        size = 1
        for d in dimensions:
            assert d > 0, "Dimensions should be greater than 0"
            size *= d
        #create a 1-D array to store all elements
        self._elements = Array(size)
        #create 1-D array to store the equation factors
        self._factors = Array(len(dimensions))
        self._computeFactors()
        #no of dimensions in the array
    def numDims(self):
        return len(self._dims)
    #returns the lenth of the given dimension
    def length(self, dim):
        assert dim >= 1 and dim <= len(self._dims),\
        "Dimension component out of range"
        return self._dims[dim-1]
    #def clears the array by setting all elements to value
    def clear(self, value):
        self._elements.clear(value)
    #returns the cotnest of element(i_1, i_2,..., i_n)
    def __getitem__(self, ndxTuple):
        assert len(ndxTuple) == self.numDims(), "Invalid array subscripts!"
        index = self._computeIndex(ndxTuple)
        assert index is not None, "Array subscript out of range"
        return self._elements[index]
    def __setitem__(self, ndxTuple, value):
        assert len(ndxTuple) == self.numDims(), "Invalid # of array subscripts"
        index = self._computeIndex(ndxTuple)
        assert index is not None, "Array subscript out of range"
        self._elements[index] = value
    #computes a 1-D array offset for element (i_1, i_2, ...i_n)
    #using the equation i_1 * f_1 + i_2 * f_2 +.... + i_n * f_n
    def _computeIndex(self, idx):
        offset = 0
        for j in range(len(idx)):
            #make sure the index componens within the legal range
            if idx[j] < 0 or idx[j] >= self._dims[j]:
                return None
            else:
                offset += idx[j] * self._factors[j]
        return offset
    #computes the factor values used in the index equation
    #done as part of exercise
    def _computeFactors(self):
        self._factors[len(self._factors)-1] = 1
        for j in reversed(range(len(self._factors) - 1)):
            self._factors[j] = self._factors[j+1] * self._dims[j+1]
def generateReport(fileName):
        multiarray = MultiArray(5,20,12)
        inputFO = open(fileName, "r")

        for line in inputFO:

            words = line.split()
            if len(words) == 2:
                item_counter = 0
                f_index = int(words[1][1]) - 1#store index
            if len(words) == 13:
                if words[0] == "Item#":
                    pass
                else:
                    for i in range(12):
#                       print(f_index,item_counter)
                        multiarray[f_index, item_counter, i] = float(words[i+1])
                    item_counter += 1
        inputFO.close()
        inputFO = None
        return multiarray

def totalSalesByStore( SalesData, store ):
# Subtract 1 from the store # since the array indices are 1 less
# than the given store #.
    s = store-1
    # Accumulate the total sales for the given store.
    total = 0.0
    # Iterate over item.
    for i in range( SalesData.length(2) ):
        # Iterate over each month of the i item.
        for m in range( SalesData.length(3) ):
            total += SalesData[s, i, m]
    return total

salesData=generateReport("SalesData.txt")
print ("totalsalesby store 1 : ",totalSalesByStore(salesData,1)) #print stores 1 total sell
print ("totalsalesby store 2 : ",totalSalesByStore(salesData,2))
```

```
print ("totalsalesby store 3 : ",totalSalesByStore(salesData,3))
```

```
totalsalesby store 1 :  534323.5000000001
totalsalesby store 2 :  459560.5
totalsalesby store 3 :  534323.5000000001
```

In [ ]: