Main

# -\*- coding: utf-8 -\*-

"""

Created on Sat Mar 28 21:31:22 2020

@author: 45242

"""

class TSAM:

def \_\_init\_\_(self, contents = []):

self.Dict = {}

self.BigNameSet = []

self.LitNameSet = []

self.ValSet =[]

self.ENV = [[]]

self.RES = []

self.signs = ['(', ')', '.', '^']

self.arithmetic\_operators = ['+','-','\*','/']

self.comparision\_operators = ['==','<','>','<=','>=','!=']

self.aggregation\_operators = ['COUNT','MAX','SUM','MIN', 'AVERAGE']

self.list\_operators = ['Exists', 'In', 'Contains']

self.Boolean\_operators = ['and','or','not']

for e in contents:

self.Dict[e[0]] = e

self.ENV[0].append(e[0]) #initialize the ENV

if e[1] not in self.BigNameSet:

self.BigNameSet.append(e[1])

if type(e[2]) == list:

for el in e[2]:

self.Dict[el[0]] = el

if el[1] not in self.LitNameSet:

self.LitNameSet.append(el[1])

if el[2] not in self.ValSet:

self.ValSet.append(el[2])

def eval(self, query):

result = None

query\_list = query.split(' ')

i = 0

while i < len(query\_list):

if len(query\_list[i]) > 1:

for j in range(len(query\_list[i])):

if query\_list[i][j] in self.signs:

substr = query\_list[i]

if j == 0:

query\_list[i] = substr[0]

query\_list.insert(i+1, substr[1:])

elif j == len(query\_list[i]) - 1:

query\_list[i] = substr[:-1]

query\_list.insert(i+1, substr[-1])

else:

query\_list[i] = substr[:j]

query\_list.insert(i+1, substr[j])

query\_list.insert(i+2, substr[j+1:])

break

i += 1

result = self.RES.pop()

return result

def \_\_eval(self, query):

command = query[0]

if command == '(':

count = 0

for i in range(len(query)):

if query[i] == '(':

count += 1

elif query[i] == ')':

count -= 1

if count == 0:

break

self.\_\_eval(query[1: i])

if i < len(query) - 1:

self.\_\_eval(query[i+1: ])

elif command == '.':

self.\_\_eval(query[1: ])

self.RES.append(self.proj(self.RES.pop(), self.RES.pop()))

elif command == 'WHERE':

self.\_\_eval(query[1: ])

if self.RES.pop() == False:

self.RES.pop()

self.RES.append(None)

def nested(self, key):

pass

def distinct(self, la):

pass

def deref(self, key):

pass

def cart(self, lb, la):

pass

def equljoin(self, lb, la):

pass

def proj(self, litname, key):

pass

def Lname(self, la, name):

pass

def comparision(self, symbol, b, a):

pass

def arithmetic\_operate(self, symbol, b, a):

pass

def aggregation\_operate(self, symbol, la):

pass

def Boolean\_operate(self, symbol, b, a):

pass

Set = [(1, "Theatre", [(2, "cinema", "Abaton"), (3, "address", "Grindle Alley")]), (4, "Theatre", [(5, "cinema", "Flora"), (6, "address", 'Old Village')]),

(7, 'Theatre', [(8, 'cinema', 'Holi')]), (10, 'Performance', [(11, 'cinema', 'Flora'), (12, 'title', 'The Piano'), (13, 'date', 'May 7')]),

(14, 'Performance', [(15, 'cinema', 'Holi'), (16, 'title', 'Manhattan')]), (20, 'Play', [(21, 'title', 'The Piano'), (22, 'director', 'Campio')]),

(23, 'Play', [(24, 'title', 'Manhattan'), (25, 'director', 'Allen')]), (30, 'Nationalty', [(31, 'director', 'Campio'), (32, 'country', 'USA')]),

(33, 'Nationalty', [(34, 'director', 'Allen'), (35, 'country', 'USA')]) ]

TSAM = TSAM(Set)

"""

print("---------------------\nBigNameSet:")

print(TSAM.BigNameSet)

print("---------------------\nENV[0]:")

print(TSAM.ENV)

print("---------------------\nLitNameSet:")

print(TSAM.LitNameSet)

print("---------------------\nValSet:")

print(TSAM.ValSet)

print("---------------------\nDict:")

print(TSAM.Dict)

"""

# -\*- coding: utf-8 -\*-

"""

Created on Sat Mar 28 21:31:22 2020

@author: 45242

"""

class HashSet:

def \_\_init\_\_(self,contents=[]):

self.items = [None] \* 20

self.numItems = 0

for e in contents:

self.add(e)

def add(self,item):

if HashSet.\_\_add(item,self.items):

self.numItems += 1

load = self.numItems / len(self.items)

if load >= 0.75:

self.items = HashSet.\_\_rehash(self.items, [None]\*2\*len(self.items))

def \_\_contains\_\_(self,item):

index = hash(item[0]) % len(self.items)

while self.items[index] != None:

if self.items[index] == item:

return True

index = (index + 1) % len(self.items)

return False

def delete(self,item):

if HashSet.\_\_remove(item,self.items):

self.numItems -= 1

load = max(self.numItems,20) / len(self.items)

if load <= 0.25:

self.items = HashSet.\_\_rehash(self.items, [None]\*(len(self.items) // 2))

else:

raise KeyError("Item not in HashSet")

# ===== Hidden Class =====

class \_\_Placeholder:

def \_\_init\_\_(self):

pass

def \_\_eq\_\_(self,other):

return False

# ===== Auxiliary Functions =====

# They all have '\_\_' as prefixes to indicate that they are private methods to the class

def \_\_add(item,items):

index = hash(item[0]) % len(items)

location = -1

while items[index] != None:

if items[index] == item:

return False

if location < 0 and type(items[index]) == HashSet.\_\_Placeholder:

location = index

index = (index + 1) % len(items)

if location < 0:

location = index

items[location] = item

return True

def \_\_rehash(olditems,newitems):

for e in olditems:

if e != None and type(e) != HashSet.\_\_Placeholder:

HashSet.\_\_add(e,newitems)

return newitems

def \_\_remove(item,items):

index = hash(item[0]) % len(items)

while items[index] != None:

if items[index] == item:

nextIndex = (index + 1) % len(items)

if items[nextIndex] == None:

items[index] = None

else:

items[index] = HashSet.\_\_Placeholder()

return True

index = (index + 1) % len(items)

return False

def \_\_eq\_\_(self, other):

for e in self.items:

if e != None and type(e) != HashSet.\_\_Placeholder():

if not other.\_\_contains\_\_(e):

return False

def \_\_iter\_\_(self):

for e in self.items:

if e != None and type(e) != HashSet.\_\_Placeholder():

yield e

def Hprint(self):

la = []

for e in self:

la.append(e)

print(la)

class TSAM:

def \_\_init\_\_(self, contents = []):

self.Hset = HashSet()

self.BigNameSet = []

self.LitNameSet = []

self.ValSet =[]

self.ENV = [[]]

self.RES = []

for e in contents:

self.Hset.add(e) #put all objects into Hset with identifier as key

self.ENV[0].append(e[0]) #initialize the ENV

if e[1] not in self.BigNameSet:

self.BigNameSet.append(e[1])

if type(e[2]) == list:

for el in e[2]:

self.Hset.add(el)

if el[1] not in self.LitNameSet:

self.LitNameSet.append(el[1])

if el[2] not in self.ValSet:

self.ValSet.append(el[2])

def eval(self, query):

query\_list = query.split(' ')

i = 0

while i < len(query\_list):

if len(query\_list[i]) > 1:

for j in range(len(query\_list[i])):

if query\_list[i][j] == '(' or query\_list[i][j] == ')':

substr = query\_list[i]

if j == 0:

query\_list[i] = substr[0]

query\_list.insert(i+1, substr[1:])

elif j == len(query\_list[i]) - 1:

query\_list[i] = substr[:-1]

query\_list.insert(i+1, substr[-1])

else:

query\_list[i] = substr[:j]

query\_list.insert(i+1, substr[j])

query\_list.insert(i+2, substr[j+1:])

break

return TSAM.\_\_eval(self, query\_list)

def \_\_eval(self, query):

pass

Set = [(1, "Theatre", [(2, "cinema", "Abaton"), (3, "address", "Grindle Alley")]), (4, "Theatre", [(5, "cinema", "Flora"), (6, "address", 'Old Village')]),

(7, 'Theatre', [(8, 'cinema', 'Holi')]), (10, 'Performance', [(11, 'cinema', 'Flora'), (12, 'title', 'The Piano'), (13, 'date', 'May 7')]),

(14, 'Performance', [(15, 'cinema', 'Holi'), (16, 'title', 'Manhattan')]), (20, 'Play', [(21, 'title', 'The Piano'), (22, 'director', 'Campio')]),

(23, 'Play', [(24, 'title', 'Manhattan'), (25, 'director', 'Allen')]), (30, 'Nationalty', [(31, 'director', 'Campio'), (32, 'country', 'USA')]),

(33, 'Nationalty', [(34, 'director', 'Allen'), (35, 'country', 'USA')]) ]

TSAM = TSAM(Set)

print("---------------------\nBigNameSet:")

print(TSAM.BigNameSet)

print("---------------------\nENV[0]:")

print(TSAM.ENV)

print("---------------------\nLitNameSet:")

print(TSAM.LitNameSet)

print("---------------------\nValSet:")

print(TSAM.ValSet)

print("---------------------\nHashSet:")

TSAM.Hset.Hprint()