Week 4 Programming Assignment

Due on 2022-08-25, 23:59 IST

Write two Python functions as specified below. Paste the text for both functions together into the submission window. Your function will be called automatically with various inputs and should return values as specified. Do not write commands to read any input or print any output.

- You may define additional auxiliary functions as needed.
- In all cases you may assume that the value passed to the function is of the expected type, so your function does not have to check for malformed inputs.
- For each function, there are normally some public test cases and some (hidden) private test cases.
- "Compile and run" will evaluate your submission against the public test cases.
- "Submit" will evaluate your submission against the hidden private test cases. There are 11 private test cases, with equal weightage. You will get feedback about which private test cases pass or fail, though you cannot see the actual test cases.
- Ignore warnings about "Presentation errors".

1. Write a Python function `frequency(l)` that takes as input a list of integers and returns a pair of the form `(minfreqlist,maxfreqlist)` where
   - `minfreqlist` is a list of numbers with minimum frequency in `l`, sorted in ascending order
   - `maxfreqlist` is a list of numbers with maximum frequency in `l`, sorted in ascending order

   Here are some examples of how your function should work.

```
>>> frequency([13,12,11,13,14,13,7,11,13,14,12])
([7], [13])

>>> frequency([13,12,11,13,14,13,7,11,13,14,12,14,14])
([7], [13, 14])

>>> frequency([13,12,11,13,14,13,7,11,13,14,12,14,14,7])
([7, 11, 12], [13, 14])
```

2. An airline has assigned each city that it serves a unique numeric code. It has collected information about all the direct flights it operates, represented as a list of pairs of the form `(i,j)`, where `i` is the code of the starting city and `j` is the code of the destination.

   It now wants to compute all pairs of cities connected by one intermediate hope — city `i` is connected to city `j` by one intermediate hop if there are direct flights of the form `(i,k)` and `(k,j)` for some other city `k`. The airline is only interested in one hop flights between different cities — pairs of the form `(i,i)` are not useful.

   Write a Python function `onehop(l)` that takes as input a list of pairs representing direct flights, as described above, and returns a list of all pairs `(i,j)`, where `i != j`, such that `i` and `j` are connected by one hop. Note that it may already be the case that there is a direct flight from `i` to `j`. So long as there is an intermediate `k` with a flight from `i` to `k` and from `k` to `j`, the list returned by the function should include `(i,j)`. The input list may be in any order. The pairs in the output list should be in lexicographic (dictionary) order. Each pair should be listed exactly once.

   Here are some examples of how your function should work.

```
>>> onehop([(2,3),(1,2)])
[(1, 3)]

>>> onehop([(2,3),(1,2),(3,1),(1,3),(3,2),(2,4),(4,1)])
[(1, 2), (1, 3), (1, 4), (2, 1), (3, 2), (3, 4), (4, 2), (4, 3)]

>>> onehop([(1,2),(3,4),(5,6)])
[]
```

Private Test cases used for evaluation	Input
Test Case 1	`frequency([17322,271898,374,374,374,423432423,423432423,423432423,423432423,5325,5325,5325,5325,5325])`
Test Case 2	`frequency([17322,374,17322,374,17322,374])`
Test Case 3	`frequency([9842])`
Test Case 4	`frequency([-17322,-271898,-374,-374,-374,-423432423,-423432423,-423432423,-423432423,-5325,-5325,-5325,-5325,-53`
Test Case 5	`frequency([-17322,-374,-17322,-374,-17322,-374])`
Test Case 6	`onehop([(1,3),(1,2),(2,3),(2,1),(3,2),(3,1)])`

Test Case 7

```
onehop([(1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (1, 9), (2, 1), (2, 3), (2, 4), (2, 5), (2, 6),
7), (2, 8), (2, 9), (3, 1), (3, 2), (3, 4), (3, 5), (3, 6), (3, 7), (3, 8), (3, 9), (4, 1), (4, 2), (4, 3), (4,
(4, 6), (4, 7), (4, 8), (4, 9), (5, 1), (5, 2), (5, 3), (5, 4), (5, 6), (5, 7), (5, 8), (5, 9), (6, 1), (6, 2),
3), (6, 4), (6, 5), (6, 7), (6, 8), (6, 9), (7, 1), (7, 2), (7, 3), (7, 4), (7, 5), (7, 6), (7, 8), (7, 9), (8,
(8, 2), (8, 3), (8, 4), (8, 5), (8, 6), (8, 7), (8, 9), (9, 1), (9, 2), (9, 3), (9, 4), (9, 5), (9, 6), (9, 7),
8)])
```

Test Case 8

```
onehop([(1,2),(2,3),(3,4),(4,5),(5,1)])
```

Test Case 9

```
onehop([(1,2),(2,3),(3,4),(4,5),(5,1),(5,6),(6,7),(7,8),(8,9),(9,5)])
```

Test Case 10

```
onehop([(1,2),(2,1),(3,4),(4,3)])
```

Test Case 11

```
onehop([(1,2),(3,4)])
```

The due date for submitting this assignment has passed.
11 out of 11 tests passed.
You scored 100.0/100.

Assignment submitted on 2022-08-25, 12:19 IST

Your last recorded submission was :

```python
def frequency(l):
    SET=set(l)
    LIST=list(SET)
    newl=list()
    for a in LIST:
        newl.append(l.count(a))
    mi=min(newl)
    ma=max(newl)
    mil=[]
    mal=[]
    for b in range(len(newl)):
        if newl[b]==mi:

             mil.append(LIST[b])
        if newl[b]==ma:

             mal.append(LIST[b])
    mil.sort()
    mal.sort()
    return(mil,mal)


def onehop(l):
    new=[]
    l.sort()
    for i in range(len(l)):
        for j in range(len(l)):
                if l[i]!=l[j]:
                    if l[i][1]==l[j][0]:
                        q=l[i][0]
                        w=l[j][1]
                        if q!=w:
                            t=[q,w]
                            t=tuple(t)
                            if t not in new:
                                new.append(tuple(t))
    new.sort()
    return (new)
import ast

def parse(inp):
  inp = ast.literal_eval(inp)
  return (inp)

fncall = input()
lparen = fncall.find("(")
rparen = fncall.rfind(")")
fname = fncall[:lparen]
farg = fncall[lparen+1:rparen]

if fname == "frequency":
  arg = parse(farg)
  print(frequency(arg))

if fname == "onehop":
  arg = parse(farg)
  print(onehop(arg))


```

Sample solutions (Provided by instructor)

```python
def frequency(l):
    count = {}
    for n in l:
        if n in count.keys():
            count[n] = count[n]+1
        else:
            count[n] = 1
    minlist = findmin(count)
    maxlist = findmax(count)
    return((minlist,maxlist))

def findmin(d):
    upperbound = 0
    for n in d.keys():
        if d[n] > upperbound:
            upperbound = d[n]

    minlist = []
    mincount = upperbound

    for n in d.keys():
        if d[n] < mincount:
            minlist = [n]
            mincount = d[n]
        elif d[n] == mincount:
            minlist.append(n)
    return(sorted(minlist))


def findmax(d):
    maxlist = []
    maxcount = 0

    for n in d.keys():
        if d[n] > maxcount:
            maxlist = [n]
            maxcount = d[n]
        elif d[n] == maxcount:
            maxlist.append(n)
    return(sorted(maxlist))

###################

def onehop(l):
    direct = {}
    for (i,j) in l:
        if i in direct.keys():
            direct[i].append(j)
        else:
            direct[i] = [j]

    hopping = []

    for src in direct.keys():
        for dest in direct[src]:
            if dest in direct.keys():
                for remote in direct[dest]:
                    if src != remote:
                        hopping.append((src,remote))

    return(remdup(sorted(hopping)))

def remdup(l):
    if len(l) < 2:
        return(l)

    if l[0] != l[1]:
        return(l[0:1]+remdup(l[1:]))
    else:
        return(remdup(l[1:]))

###################

import ast

def parse(inp):
  inp = ast.literal_eval(inp)
  return (inp)

fncall = input()
lparen = fncall.find("(")
rparen = fncall.rfind(")")
fname = fncall[:lparen]
farg = fncall[lparen+1:rparen]

if fname == "frequency":
  arg = parse(farg)
  print(frequency(arg))

if fname == "onehop":
  arg = parse(farg)
  print(onehop(arg))

```