University of Augsburg, Institute of Computer Science
Prof. Dr. P. Fischer
J. Kastner, L. Rudenko

SS 2019
13. May 2019
Solution 2

Analyzing Massive Data Sets

Exercise 1: MapReduce - Find and List Duplicate Files (homework)

One typical, important decision are the key attribute(s) of the map phase. In our case, the output of the **map phase** should be **[md5sum, filename]**, thus files with the same content will be collected to the same group. At the end the reducer will eliminate the unique files and store the files with the same content to the result.

```
from mrsim import mr_simulator

def map(key, val):
    # res - List
    res = []
    # (key, val) - Tupel, der der Liste hinzugefuegt wird
    res.append((key, val))
    return res

def reduce (key, val):
    res = []
    if (len(val) > 1):
        # remove duplicates (original order is not preserved)
        val = list(set(val))

        #res.append((key, val))
        res.append((str(key) + ':' + str(val)))
    return res

inputs = [(123, 'Name1'), (123, 'Name2'), (456, 'Name1'),
          (345, 'Name3'), (456, 'Name2'), (123, 'Name1')]

res = mr_simulator(inputs, map, reduce)
print(res)
```

Exercise 2: MapReduce - Vector Length (live)

The solution was discussed in the exercise.

Exercise 3: MapReduce - Find the Common Friends (live)

Two possible cases should be considered:

- The two users for which common friends are searched are also friends.

  The solution was discussed in the exercise.

- The two users are not connected directly, but share common friends.

  If the nature of relation is **bi-directional** (as in the fist case), the solution is quite simple:

  (Note, that the *map*-function **does not necessarily** have a one-to-one mapping between input and output tuple. One input tuple can, for example, produce several output tuples after processing with the map function.)

```
from mrsim import mr_simulator
import itertools

#build all combinations of friends pairs of each person.
#Fetch combination as key and person as value.
def map(person, friends):
    res = []

    if (len(friends) > 1):
        pairs = itertools.combinations(friends,2)
        for p in pairs:
            res.append((p[0]+p[1],person))
    return res

# after combining put it out in res
def reduce(key, val):
    res = []
    res.append((key, val))
    return res

inputs = [['Pa', ['Pb', 'Pc', 'Pe']],
          ['Pb', ['Pa']],
          ['Pc', ['Pa', 'Pd', 'Pe']],
          ['Pd', ['Pc', 'Pe']],
          ['Pe', ['Pa', 'Pc', 'Pd']]]

result = mr_simulator(inputs, map ,reduce)
print(result)
```

If relation is **NOT bidirectional**, the solution is a little bit complicated:

```
#combine each person and friend of a person an choose friend as key and the person as value.
#(in order to group the followers of the persons friends)
def map(person, friends):
    res = []

    for friend in friends:
        res.append((friend, person))

    return res

# combine for each person the followers and choose them as key
def reduce (person, friendsList):

    res = []

    if (len(friendsList) > 1):
        pairs = itertools.combinations(friendsList,2)
        for p in pairs:
            res.append((p[0]+p[1],person))
    return res

def identity_map(key,val):
    res = []
    res.append((key,val))
    return res

def identity_group (key, val):
    res = []
    res.append((key,val))
    return res
```

```
inputs = [['Pa', ['Pb', 'Pc', 'Pd']],
          ['Pb', ['Pd']],
          ['Pc', ['Pb', 'Pd']],
          ['Pd', ['Pa']]]

linker = mr_simulator(inputs, map, reduce)
print (linker)

result= mr_simulator(linker, identity_map, identity_group)
print(result)
```