CptS 415 | Assignment-05

1. MapReduce

a. Common Friends

Facebook updates the "common friends" of you and respond to hundreds of millions of requests every day.

The friendship information is stored as a pair: `(Person, [List of Friends])` for every user in the social network.

Write a MapReduce program *to return a dictionary of common friends* of the form:

```python
user_pair, list_of_common_friends =\
        (user_i, user_j),
        [list(
                #Common Friends of user_i and user_j)
                )]
# for all pairs of i and j who are friends.
```

The order of i and j you returned should be the same as the lexicographical order of their names. You need to give the pseudo-code of a main function, and both Map() and Reduce() function. Specify the key/value pair and their semantics (what are they referring to?).

Solution

```python
class Person:
    def __init__(self, name: str):
        self.name: str = name


class User:
    """
    Each User object contains:
    - k1: a Person, a unique object in database
    - v1: friendship, a dict associated with the primary key, where
        each entry use the address of the person in database as key
        for uniqueness.
    """
    def __init__(self, person: Person, friends: dict[str, Person]):
        self.person: Person                = person
        self.friendship: dict[str, Person] = friends
        self.friends_count: int            = len(self.friendship)
    def __repr__(self):
        return f"({self.person.name}, {[name for name in self.friendship]})"


def Map(i: User, j: User):
    """
    Compare the two User objects and return a new 2-tuple:
    - k2: the shorter friend list as upper bound for potential friends
    - v2: the other person as target for matching
    """
    if (i.friends_count > j.friends_count):
        return (i.friendship, j)
    else:
        return (j.friendship, i)


def Reduce(potential: dict[str, Person], target: User):
    """
    From a list of potential friends and a target, check every
    key in the potential list against the target's frienship.
    Return the list of common keys.
    """
    if len(potential) == 0 or len(target.friendship) == 0:
        return []
    else:
        common: list[Person] = list()
        for person in potential:
            if person in target.friendship:
                common += [potential[person]]
        return common


def MapReduce(i: User, j: User):
    """
    Map and Reduce together.
    - k3: the pair of users
    - v3: a list of Person objects in database
    """
    potential, target = Map(i, j)
    common: list[Person] = Reduce(potential, target)
    user_pair = sorted([i.person.name, j.person.name])
    return {"pair": tuple(user_pair), "common": common}
```

b. Top-10 Keywords

Search engine companies like Google maintains hot webpages in a set $\boldsymbol{R}$ for keyword search. Each record $r \in \boldsymbol{R}$ is an article, stored as a sequence of keywords. Write a MapReduce program to report the top 10 most frequent keywords appeared in the webpages in $\boldsymbol{R}$.

Give the pseudo-code of your MR program.

Solution

Python

```
def Map(R: list[list[str]]):
        """
        Map every word in every article r in R to an entry in a map:
        - k_w: word
        - v_w: frequency of the word, incremented at current iteration
        - intput: list of lists of words
        - output: sorted map of 2-tuple of word and frequency
        """
        bag: dict[str, int] = dict()
        for r in R:
                for word in r:
                        bag[word] = bag.get(word, 0) + 1
        ordered_bag = dict(
                        sorted(bag.keys(), key=lambda item: item[1])
                )
        return orderd_bag

def Reduce(ordered_bag: dict[str, int]):
        """
        From a bag of words sorted by frequencies, return the 10
        most common words.
        input: ordered map
        output: 10 most frequent words
        """
        words = list([x[0] for x in ordered_bag])
        w = len(words)
        return words[w-1-10:-1]

def MapReduce(R: list[list[str]]):
        """
        Return the last 10 entries as most frequent words.
        """
        ordered_bag = Map(R)
        most_frequent = Reduce(ordered_bag)
        return most_frequent
```

2. Graph Parallel Models: MR for Graph Processing

a. 2-hop Common

Consider the common friends problem in Problem 1.a. We study a “2-hop common contact problem”, where a list should be returned for any pair of friends i and j, such that the list contains all the users that can reach both i and j within 2 hops. Write a MR algorithm to solve the problem and give the pseudo code.

Solution

```
Map:
- G, user, target := min(i.friendship, j.friendship)
Dijkstra(G, user, target, w):
- common := list
- For all nodes v in user.friendship:
    - d[v] := infty
- d[user] := 0; Q := user.friendship
- While Q is non-empty, do:
    - u := ExtraMin(Q)
    - For all nodes v in adj(u):
        - if v is target and v not in common:
            - common[v] = 1, then
            - break

Reduce:
- Dijkstra(Map, 2)
```

b. d-bounded reachability

We described how to compute distances with mapReduce. Consider a class of d-bounded reachability queries as follows. Given a graph G, two nodes u and v and an integer d, it returns a Boolean answer `YES`, if the two nodes can be connected by a path of length no greater than d. Otherwise, it returns `NO`. Write an MR program to compute the query $Q(G, u, v, d)$ and give the pseudo code.

Provide necessary correctness and complexity analysis.

Solution

```
Map:
- G, user, target := min(i.friendship, j.friendship)
Dijkstra(G, user, target, w):
- common := list
- For all nodes v in user.friendship:
    - d[v] := infty
- d[user] := 0; Q := user.friendship
- While Q is non-empty, do:
    - u := ExtraMin(Q)
    - For all nodes v in adj(u):
        - Q += v.friendship
        - if v is not target and v not in common:
            - if common[v] > w:
```

```
            • continue
          • common[v] += 1, then
        • else:
          • common[v] = 1

Reduce:
  • Dijkstra(Map, d)
```

3. Hadoop

Hadoop Program:

The attached CSV file contains hourly normal recordings for temperature and dew point temperature at Asheville Regional Airport, NC, USA. *The unit of measurement* is in **tenths of a degree Fahrenheit**. For example, 344 is 34.4 F.

Write a program using Hadoop to compute and output daily average measurements for temperature and dew point temperature.

The daily average measurements *should include measurements for 24-hour period*. For example, from:

```
20100101 00:00 (2010, January 1st, 00:00)
```

to:

```
20100101 23:00 (2010, January 1st, 23:00)
```

Output the result in the format shown below - the columns are date and the combined result (separated by comma) of daily temperature and daily dew point temperature:

```Plain Text
20100101    377.04, 285.58
20100102    378.67, 286.92
....        ....  , ....
```

You may write the application in Java, C/C++ or Python language. Provide both source code and compiled code, if applicable, for your program.

△ **Solution**

```Python
import pandas as pd
from hdfs import InsecureClient
import os

client_hdfs = InsecureClient('http://hdfs_ip:50070')

datafile = 'normal_hly_sample_temperature.csv'

with client_hdfs.read('/user/hdfs/' + datafile, encoding = 'utf-8') as reader:
    i = 0
    for chunk in pd.read_csv(reader, sep=',', chunksize=24):
        df    = chunk
        today = df['DATE'][i].split(' ')[0]
        temp  = round(df['HLY-TEMP-NORMAL'].sum() / df.shape[0], 2)
        dew   = round(df['HLY-DEWP-NORMAL'].sum() / df.shape[0], 2)
        print(f"{today}\t{temp}, {dew}")
        i += 24
```

Ouput:

```Plain Text
20100101        377.04, 285.58
20100102        378.67, 286.92
20100103        379.46, 288.25
```

```
20100104        377.75, 286.42
20100105        375.42, 283.17
20100106        375.08, 281.79
20100107        374.46, 281.58
20100108        371.96, 277.29
20100109        368.75, 272.58
20100110        366.29, 269.58
20100111        364.96, 266.5
20100112        363.04, 263.08
20100113        360.42, 260.21
20100114        357.38, 256.83
20100115        355.12, 254.25
20100116        354.75, 253.58
20100117        355.21, 253.46
20100118        355.29, 251.67
20100119        354.71, 249.88
20100120        353.29, 247.46
20100121        352.5, 244.75
20100122        353.79, 246.5
20100123        356.21, 250.42
20100124        358.54, 251.88
20100125        360.92, 253.12
20100126        363.21, 255.67
20100127        365.71, 258.46
20100128        368.58, 261.21
20100129        369.83, 261.21
20100130        370.67, 260.88
20100131        371.75, 261.04
```