Cloud Computing

CSMCC16

26801076, Jin li
University of Reading
2018-02-05

Abstract

This report shows the details of MapReduce prototype developed to address the task. Regarding to the data error, it also shows how to detect and correct the error. The final prototype will be developed to standalone application used in various platforms independently. All prototype development we use version control.

Contents

1 Introduction

This report will develop a prototype of MapReduce framework and show how this model to solve the problem efficiently. We also explain the logic of error detection and correction. We use GitLab as version control to track the progress of prototype development. In order to apply on different platform independently, we create a standalone application.

2 Task Review and Plan

The aim of this task is to truly understand the model of MapReduce and create mimic MapReduce framework, not implement existing MapReduce to solve the problems. The task includes four main questions. The emphasis of each question is different but associated.

Question 1: Determine the number of flights from each airport; include a list of any airports not used.

We plan to use flight departure airport code as key. The first list will come from AComp_Passenger_data, which is the flights information that happened. The second list will come from Top30_airports_LatLong. We assume this is also flights information with departure airport code but not happened. Finally, Using mimic MapReduce to produce the solution.

Question 2: Create a list of flights with passenger id, IATA/FAA codes, and departure/arrival/flight time.

We plan use each flight id, IATA/FAA codes and departure/arrival/flight time as a tuple to be the unique key, and the passenger id will be the value to be MapReduced. The essential of this question is distinct each unique passenger id.

Question 3: Calculate the number of passengers on each flight

This question seems like question 1, but not actually. It is related to question 2. The difference is it will count all passengers no matter they use the same id or not. If consider the errors in departure code and destination code, it should follow the model of question 2.

Question 4: calculate the total miles travelled by each passenger and output the highest

This question should firstly match each flight with latitude and longitude and use distance function to calculate flight miles of each flight. Then we use passenger id as key and the flight miles as value to mapreduce the highest mile passenger.

3 MapReduce Framework

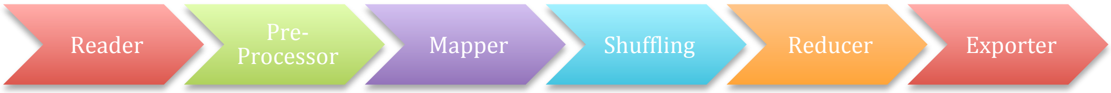
The MapReduce is a distributed and parallel approach processing massive amount of data. It consists of two steps, Map and Reduce. To relieve the data transmission stress and improve the efficiency of computing, it may also include Combiner and Shuffling. In Hadoop framework, Combiner will group by the value of same key of Mapper in same Node working like Reducer. The Shuffling will list all the values of each unique key from many Nodes after Mapper and Combiner and Reduce will group by the value of each key.

In this report, the model of MapReduce we designed consists Reader, Mapper, Shuffling, Reducer and Exporter working on single master node, each part can be copied in multi-mapreduce or multi-threading system.

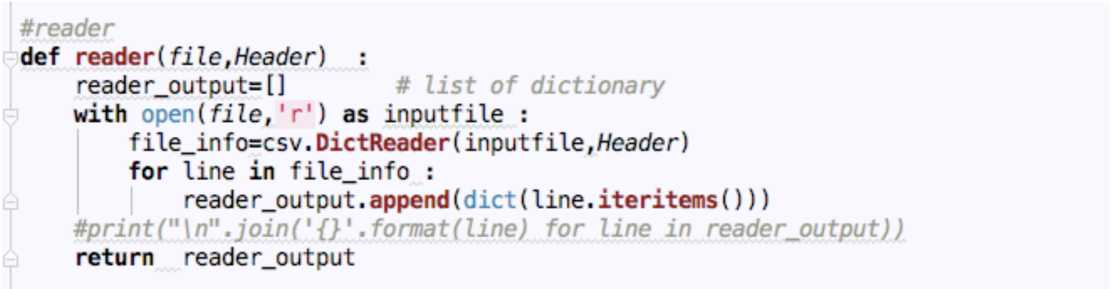
Reader and Shuffling is the same and implemented for all questions. According to different questions, we have Mapper_Q1, Mapper_Q2, Mapper_Q4, Reducer_Q1, Reducer_Q2, Reducer_Q3 and Reducer_Q4.

4 Prototype Details

This section is to fully describe the prototype created to solve the questions and deeply understand the MapReduce framework. The language we use is Python, an object-oriented language similar with java. As we consider the error detection and correction, the structure also includes pre-processor. Each question follows the same steps:

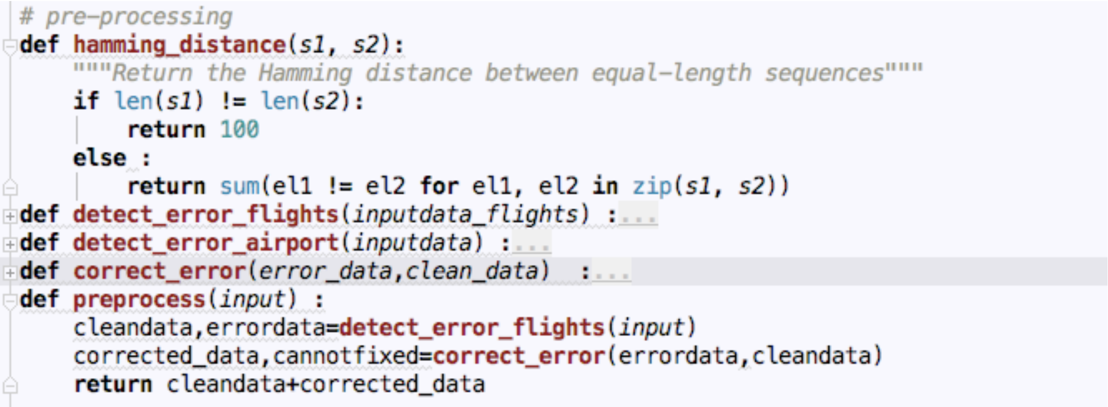


Reader : Import data with header. The data will be list of dictionary with header as dictionary key.



```
#reader
def reader(file,Header)  :
    reader_output=[]        # list of dictionary
    with open(file,'r') as inputfile :
        file_info=csv.DictReader(inputfile,Header)
        for line in file_info :
            reader_output.append(dict(line.iteritems()))
    #print("\n".join('{}'.format(line) for line in reader_output))
    return  reader_output
```

Pre-Processor: includes error detection and error correction. The corrected data will also be processed in following steps.

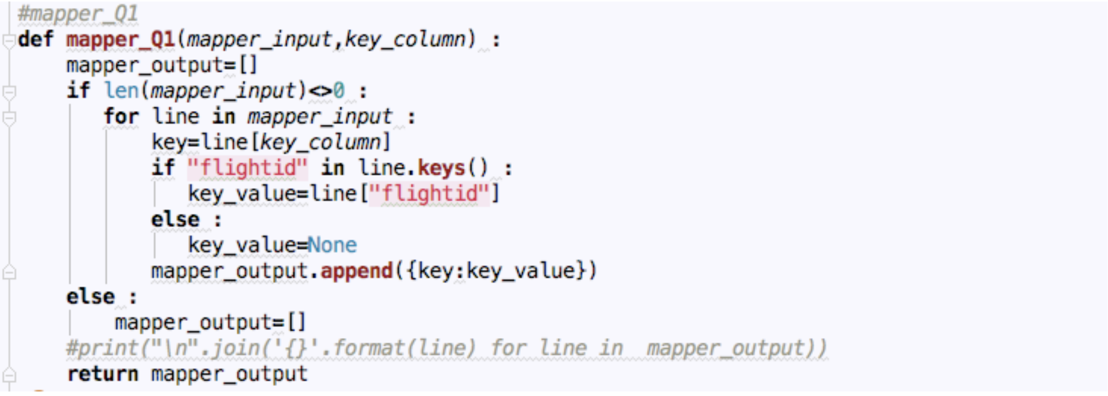


```
# pre-processing
def hamming_distance(s1, s2):
    """Return the Hamming distance between equal-length sequences"""
    if len(s1) != len(s2):
        return 100
    else :
        return sum(el1 != el2 for el1, el2 in zip(s1, s2))
def detect_error_flights(inputdata_flights) :...
def detect_error_airport(inputdata) :...
def correct_error(error_data,clean_data)  :...
def preprocess(input) :
    cleandata,errordata=detect_error_flights(input)
    corrected_data,cannotfixed=correct_error(errordata,cleandata)
    return cleandata+corrected_data
```

The essential of the pre-processor is the distance measurement approach between two strings. It is noticed that the errors in AComp_Passenger_data are mainly typing error but the string length is the same, comparing to Levenshtein distance or Damerau-Levenshtein distance, the Hamming distance between two strings of equal length is measuring the minimum number of substitutions required to change one string into the other[1].

Mapper: After pre-processing, the mapper is mapping the key with value as a pair according to different tasks.

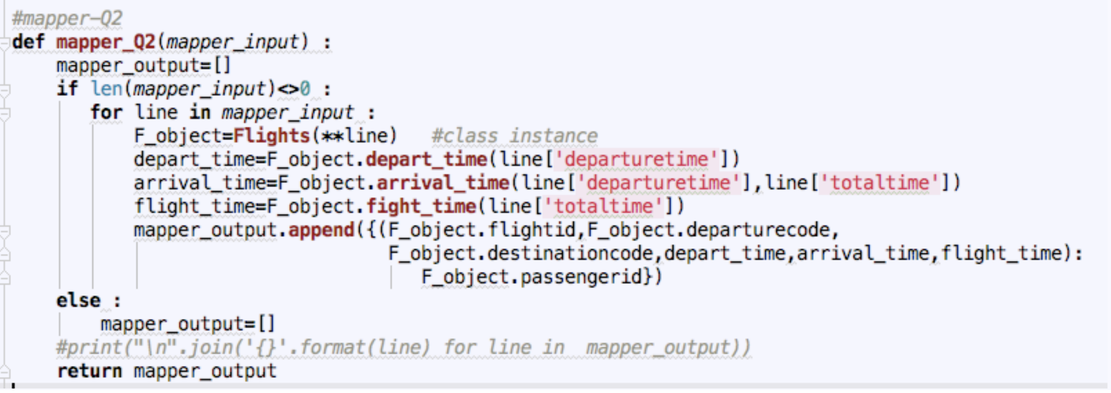
The mapper of question one is to read data line by line and mapping the departure code with flightid, finally the output buffer of mapper will be {key: value}. According to the additional requirement of question one, we use same mapper to mapping airportcode from Top30_airports_LatLong with None in order to include the list of any airports not used.



```
#mapper_Q1
def mapper_Q1(mapper_input,key_column) :
    mapper_output=[]
    if len(mapper_input)<>0 :
        for line in mapper_input :
            key=line[key_column]
            if "flightid" in line.keys() :
                key_value=line["flightid"]
            else :
                key_value=None
            mapper_output.append({key:key_value})
    else :
        mapper_output=[]
    #print("\n".join('{}'.format(line) for line in  mapper_output))
    return mapper_output
```

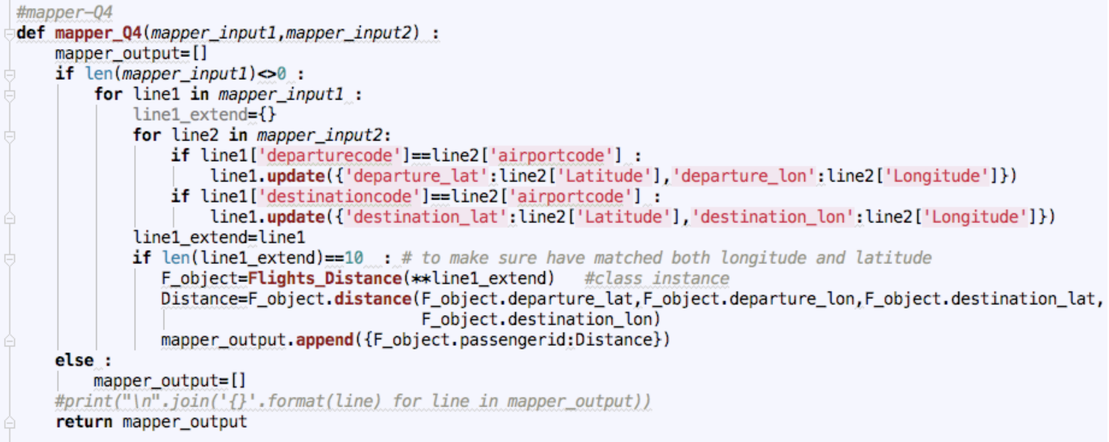
The key of mapper for question two is a tuple like (flightid, departurecode, destinationcode, depart_time, arrival_time, flight_time) and value is passengerid. So the output of mapper_Q2 is {Key: passengerid}. We combine class concept to design the mapper. In Flight class, it has time conversion function. In addition, mapper_Q2 is implemented in question three.

[1] https://en.wikipedia.org/wiki/Hamming_distance



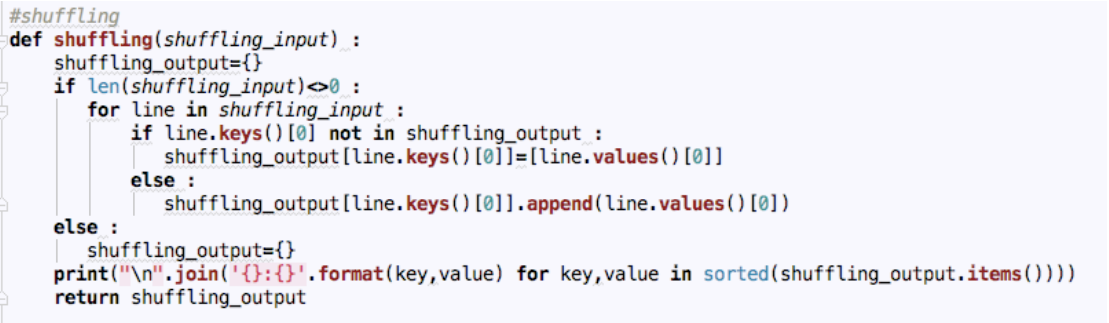
```
#mapper-Q2
def mapper_Q2(mapper_input) :
    mapper_output=[]
    if len(mapper_input)<>0 :
        for line in mapper_input :
            F_object=Flights(**line)    #class instance
            depart_time=F_object.depart_time(line['departuretime'])
            arrival_time=F_object.arrival_time(line['departuretime'],line['totaltime'])
            flight_time=F_object.fight_time(line['totaltime'])
            mapper_output.append({(F_object.flightid,F_object.departurecode,
                                  F_object.destinationcode,depart_time,arrival_time,flight_time):
                                    F_object.passengerid})
    else :
        mapper_output=[]
    #print("\n".join('{}'.format(line) for line in  mapper_output))
    return mapper_output
```

Mapper_Q4 is created for question 4, which includes the latitude and longitude match and mileage calculation before mapping miles to passenger id. The output of mapper is {passengerid: Distance}



```
#mapper-Q4
def mapper_Q4(mapper_input1,mapper_input2) :
    mapper_output=[]
    if len(mapper_input1)<>0 :
        for line1 in mapper_input1 :
            line1_extend={}
            for line2 in mapper_input2:
                if line1['departurecode']==line2['airportcode'] :
                    line1.update({'departure_lat':line2['Latitude'],'departure_lon':line2['Longitude']})
                if line1['destinationcode']==line2['airportcode'] :
                    line1.update({'destination_lat':line2['Latitude'],'destination_lon':line2['Longitude']})
            line1_extend=line1
            if len(line1_extend)==10  : # to make sure have matched both longitude and latitude
               F_object=Flights_Distance(**line1_extend)   #class instance
               Distance=F_object.distance(F_object.departure_lat,F_object.departure_lon,F_object.destination_lat,
                                          F_object.destination_lon)
               mapper_output.append({F_object.passengerid:Distance})
    else :
        mapper_output=[]
    #print("\n".join('{}'.format(line) for line in mapper_output))
    return mapper_output
```

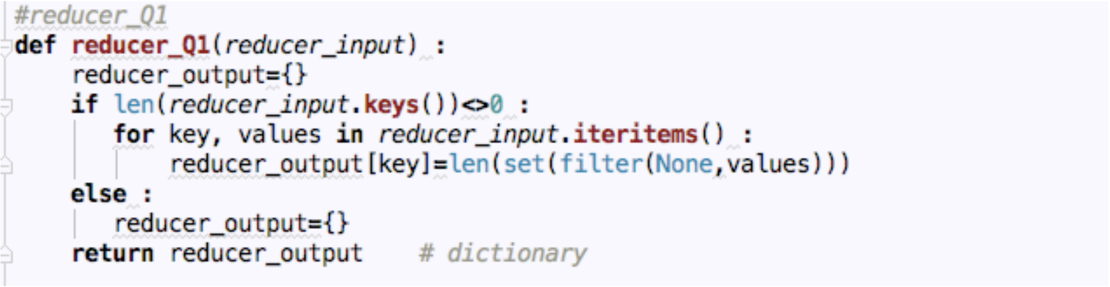
Shuffling: Combine all the value as a list of each key and sort by key, the output will be {key: list of value}, like {PIT2755XC1: [1199.34, 8064.39, 4201.07,...]}



```
#shuffling
def shuffling(shuffling_input) :
    shuffling_output={}
    if len(shuffling_input)<>0 :
        for line in shuffling_input :
            if line.keys()[0] not in shuffling_output :
                shuffling_output[line.keys()[0]]=[line.values()[0]]
            else :
                shuffling_output[line.keys()[0]].append(line.values()[0])
    else :
        shuffling_output={}
    print("\n".join('{}:{}'.format(key,value) for key,value in sorted(shuffling_output.items())))
    return shuffling_output
```

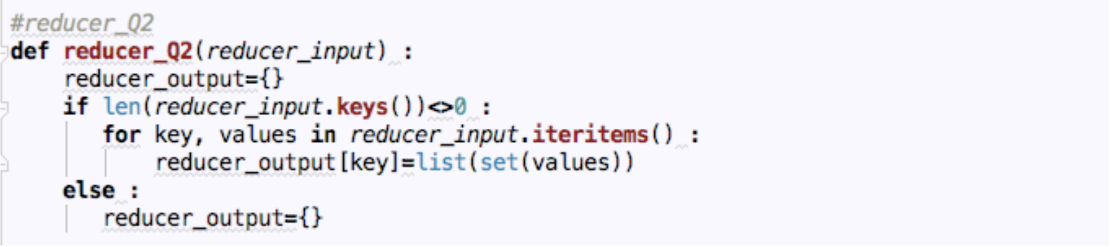
Reducer: Summary the total number or miles or distinct the passenger id

Reduce_Q1 is used in question 1. The output of reducer will be the number of unique flightid, if there is no flightid, the value will be zero, which means this airport is not used.



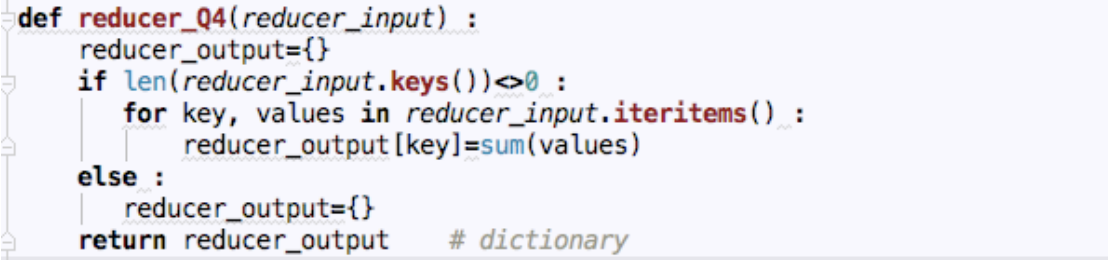
```
#reducer_Q1
def reducer_Q1(reducer_input) :
    reducer_output={}
    if len(reducer_input.keys())<>0 :
        for key, values in reducer_input.iteritems() :
            reducer_output[key]=len(set(filter(None,values)))
    else :
        reducer_output={}
    return reducer_output    # dictionary
```

Reduce_Q2 and Reduce_Q3 are very similar. The only difference is that Reduce_Q2 generates unique passenger id, while Reduce_Q3 counts all passengers without distinction. Reduce_Q3 will be show in appendix.



```
#reducer_Q2
def reducer_Q2(reducer_input) :
    reducer_output={}
    if len(reducer_input.keys())<>0 :
        for key, values in reducer_input.iteritems() :
            reducer_output[key]=list(set(values))
    else :
        reducer_output={}
```

Reduce_Q4 sums up all values, similar with groupby



```
def reducer_Q4(reducer_input) :
    reducer_output={}
    if len(reducer_input.keys())<>0 :
        for key, values in reducer_input.iteritems() :
            reducer_output[key]=sum(values)
    else :
        reducer_output={}
    return reducer_output    # dictionary
```

Exporter: Print and save the result in form of different layout in text document.

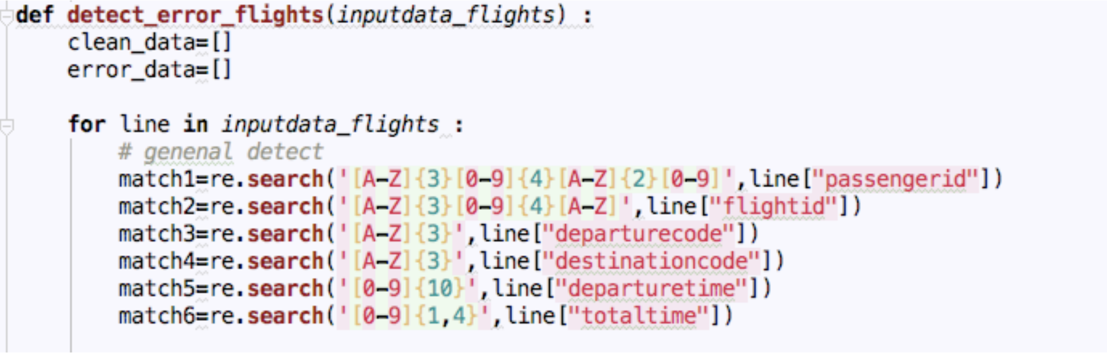
5 Error detection and correction

The preliminary error detection is done by comparing data with error and no error data. There is only one error in each record. Thirty-seven errors can be recognised by regular expression rule, but five errors cannot. It includes two passenger id(UMH6360YP0, VZT2993ME1), one fight id (PNE8178S), one departure code (UGK) and one destination code (FSA). Although these five records are obviously errors in this small amount of data set, logically it could be correct in big data set. The same flight could have different departure airport or destination airport when the flight company share the same flight for long journey. So this kind of errors will not be corrected.

The more strict regular expression check is done in pre-processing by python. Thirty-seven errors were detected, including 10 empty errors.

In pre-processing phase, we simultaneously correct the errors. Twenty-seven errors corrected together with correct data as input to mapper. When the passenger id error is corrected, the first string found by hamming function that has only 1 difference with error passenger id will replace the error. When we correct the error of flight id, departure code or destination code, we combine these three strings together as one string to find the correct one. This method will make sure the replace string is the right one and avoid wrong correction.

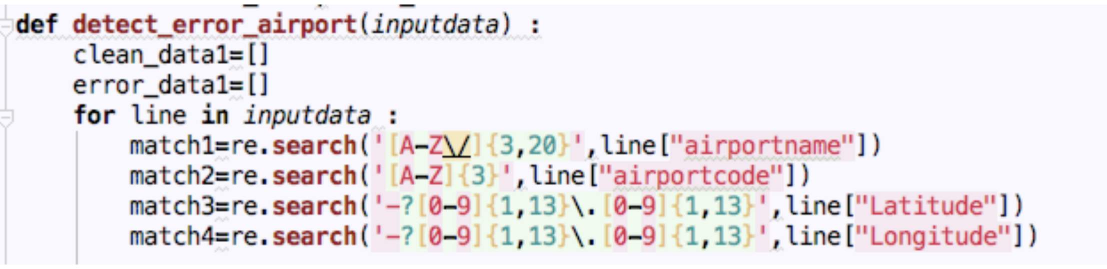
There is no error in airport file, except one empty line.



```
def detect_error_flights(inputdata_flights) :
    clean_data=[]
    error_data=[]

    for line in inputdata_flights :
        # genenal detect
        match1=re.search('[A-Z]{3}[0-9]{4}[A-Z]{2}[0-9]',line["passengerid"])
        match2=re.search('[A-Z]{3}[0-9]{4}[A-Z]',line["flightid"])
        match3=re.search('[A-Z]{3}',line["departurecode"])
        match4=re.search('[A-Z]{3}',line["destinationcode"])
        match5=re.search('[0-9]{10}',line["departuretime"])
        match6=re.search('[0-9]{1,4}',line["totaltime"])
```

…



```
def detect_error_airport(inputdata) :
    clean_data1=[]
    error_data1=[]
    for line in inputdata :
        match1=re.search('[A-Z\/]{3,20}',line["airportname"])
        match2=re.search('[A-Z]{3}',line["airportcode"])
        match3=re.search('-?[0-9]{1,13}\.[0-9]{1,13}',line["Latitude"])
        match4=re.search('-?[0-9]{1,13}\.[0-9]{1,13}',line["Longitude"])
```

…

6 Version Control

In coding project, version control is very important. Using the GitLab manage this project. The commands used are described as below.

- git add .   Add all files that has been changed
- git commit –a –m "message" commit the changes with message
- git push -u origin master   push all commits to central repository
- git pull   Fetch any changes from central repository to local repository
- git log   Print all change log

7 Standalone application

In order to run independently on various systems, it is better to develop standalone application that can be free used without development environment.

User only needs to press the button to choose the input data files and save repository. The answer will display in the text box when user presses the question button.

8 Results

Question 1: The output is a list of number of flights from each airport. The number is zero means it is unused airport. As we assume ‘PNE8178S’ is correct flightid, the number of flights from DEN is 4 not 3.

```
DEN:4
ATL:2
KUL:2
…
It has 8 unused airports
IST SIN HKG FRA SFO PHX LAX DXB
```

Question2: The output includes flight details and unique passenger id list.

```
['ATT7791R', 'AMS', 'DEN', '17:13:14', '09:54:14', '16:41:00']
DAZ3029XA0
PIT2755XC1
MXU9187YC7
EDV2089LK5
YMH6360YP0
…
```

Question3: As we mentioned in error detection section, there are three errors in terms of flightid, departure code and destination code. But we assume they are correct. So we have to use flight id, departure and destination code as key in this question.

```
('ATT7791R', 'AMS', 'DEN'):15
('BER7172M', 'KUL', 'LAS'):17
('DAU2617A', 'CGK', 'SFO'):12
…
('TMV7633W', 'CGK', 'DXB'):14
('TMV7633W', 'UGK', 'DXB'):1
…
```

Question4: As the input includes the corrected error data, so the highest miles of passenger UES9151GS5 changed from 146746.74 to 151772.59

```
UES9151GS5:151772.59
```

9 Conclusion and Further Work

In the project we have done shows the details of MapReduce and understanding of how this model works. We learn how to use version control and develop standalone. The prototype can be implemented in similar task. However, file splitting and multi-threading has to further develop to mimic Hadoop MapReduce framework. If we expanded the corrected model, it would be more flexible to address different errors.

10 Discussion

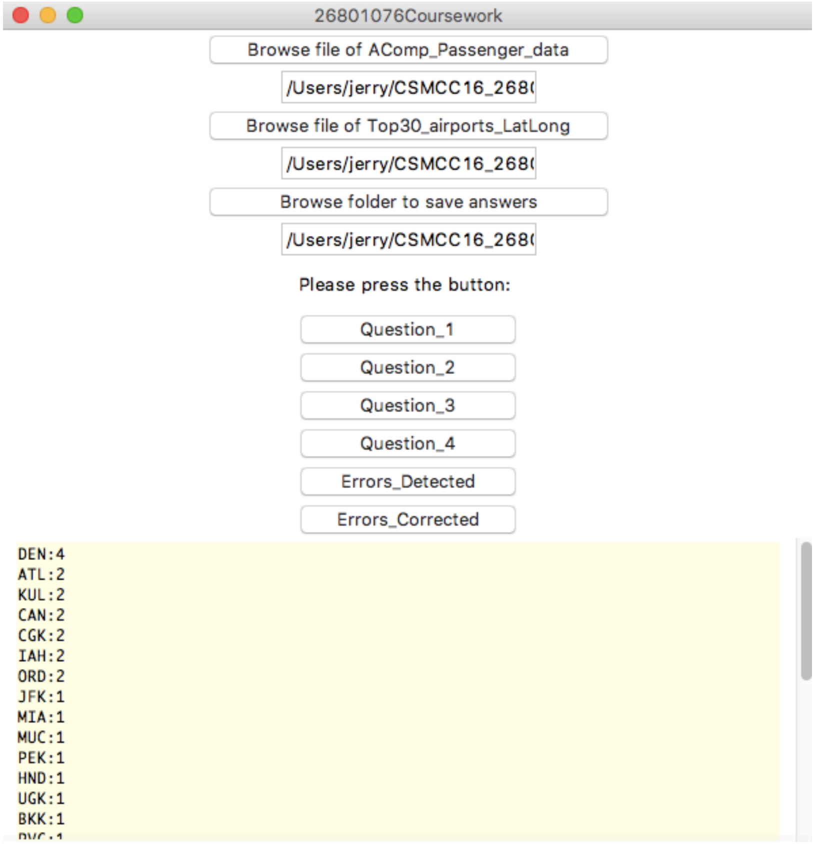
As we noticed, the error in this data set is single character error. There could be more complicated error in big database. It is necessary to develop a classifier to distinguish all kinds of error. It is necessary to consider the risk and set accuracy threshold related to whether some errors should be corrected.

11 Reference

1. Sandeep Karanth.Mastering Hadoop.2014
2. Jeffrey Dean,Sanjay Ghemawat.MapReduce: Simplified Data Processing on Large Clusters.2004
3. https://csgitlab.reading.ac.uk
4. Atta Badii.Lectures.2017

12 Appendix

Standalone application



/Users/jerry/CSMCC16_268(

/Users/jerry/CSMCC16_268(

/Users/jerry/CSMCC16_268(

```
DEN:4
ATL:2
KUL:2
CAN:2
CGK:2
IAH:2
ORD:2
JFK:1
MIA:1
MUC:1
PEK:1
HND:1
UGK:1
BKK:1
PVG:1
```

Full prototype scripts

```
import csv
import time
from math import sin, cos, sqrt, atan2, radians
import re
from Tkinter import *
import Tkinter,tkFileDialog
# Data files
Header_flights=['passengerid', 'flightid' ,'departurecode'
,'destinationcode','departuretime','totaltime']
Header_airport=['airportname', 'airportcode' ,'Latitude','Longitude']

class Flights(object) :

    def __init__(self, passengerid, flightid ,departurecode
,destinationcode,departuretime,totaltime):
        self.passengerid=passengerid
        self.flightid = flightid
        self.departurecode = departurecode
        self.destinationcode = destinationcode
        self.departuretime = departuretime
        self.totaltime = totaltime
    def depart_time(self,epochtime):
        return  time.strftime("%H:%M:%S", time.gmtime(float(epochtime)))
    def arrival_time(self,epochtime,mins):
        return  time.strftime("%H:%M:%S", time.gmtime(float(epochtime)+float(mins)*60))
    def fight_time(self,mins):
        return  time.strftime("%H:%M:%S", time.gmtime(float(mins)*60))


class Flights_Distance(Flights) :
    def __init__(self,passengerid, flightid ,departurecode
,destinationcode,departuretime,totaltime,departure_lat,departure_lon,destination_lat,destination_
lon):
        super(Flights_Distance,self).__init__(passengerid, flightid ,departurecode
,destinationcode,departuretime,totaltime)
        self.departure_lat = departure_lat
        self.departure_lon = departure_lon
        self.destination_lat = destination_lat
        self.destination_lon = destination_lon
    def distance(self,departure_lat,departure_lon,destination_lat,destination_lon):
        #R=6373.0  #km
        R=3959.0 #mile
        lat1 = radians(float(departure_lat))
        lon1 = radians(float(departure_lon))
        lat2 = radians(float(destination_lat))
        lon2 = radians(float(destination_lon))
        dlon = lon2 - lon1
        dlat = lat2 - lat1
        a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
        c = 2 * atan2(sqrt(a), sqrt(1 - a))

        flight_distance =round(R * c,2)
        return flight_distance

#reader
def reader(file,Header)  :
    reader_output=[]        # list of dictionary
    with open(file,'r') as inputfile :
        file_info=csv.DictReader(inputfile,Header)
        for line in file_info :
            reader_output.append(dict(line.iteritems()))
    #print("\n".join('{}'.format(line) for line in reader_output))
    return  reader_output

# pre-processing
def hamming_distance(s1, s2):
    """Return the Hamming distance between equal-length sequences"""
    if len(s1) != len(s2):
        return 100   # using 100 to avoid the running error
    else :
        return sum(el1 != el2 for el1, el2 in zip(s1, s2))
def detect_error_flights(inputdata_flights) :
    clean_data=[]
    error_data=[]

    for line in inputdata_flights :
        # genenal detect
        match1=re.search('[A-Z]{3}[0-9]{4}[A-Z]{2}[0-9]',line["passengerid"])
        match2=re.search('[A-Z]{3}[0-9]{4}[A-Z]',line["flightid"])
        match3=re.search('[A-Z]{3}',line["departurecode"])
        match4=re.search('[A-Z]{3}',line["destinationcode"])
        match5=re.search('[0-9]{10}',line["departuretime"])
        match6=re.search('[0-9]{1,4}',line["totaltime"])

        if match1<>None and match2<>None and match3<>None and match4<>None and
match5<>None and match6<>None :
           clean_data.append(line)
        elif match1==None :
            line['error_type']='passengerid_error'
            error_data.append(line)
        elif match2==None :
            line['error_type']='flightid_error'
            error_data.append(line)
        elif match3==None :
            line['error_type']='departurecode_error'
            error_data.append(line)
        elif match4==None :
            line['error_type']='destinationcode_error'
            error_data.append(line)
        elif match5==None :
            line['error_type']='departuretime_error'
            error_data.append(line)
        elif match6==None :
            line['error_type']='totaltime_error'
            error_data.append(line)

    #print(len(error_data))
    return clean_data,error_data
def detect_error_airport(inputdata) :
    clean_data1=[]
    error_data1=[]
    for line in inputdata :
        match1=re.search('[A-Z\/]{3,20}',line["airportname"])
        match2=re.search('[A-Z]{3}',line["airportcode"])
        match3=re.search('-?[0-9]{1,13}\.[0-9]{1,13}',line["Latitude"])
        match4=re.search('-?[0-9]{1,13}\.[0-9]{1,13}',line["Longitude"])


        if match1<>None and match2<>None and match3<>None and match4<>None :
           clean_data1.append(line)
        else :
           error_data1.append(line)

    return clean_data1
def correct_error(error_data,clean_data)  :
    corrected_output=[]
    cannot_correct=[]
    for line in error_data :
         k=0
         if line["error_type"]=='passengerid_error' :
             lists=[j["passengerid"] for j in clean_data]
             for value in list(lists) :
                 if hamming_distance(line["passengerid"] ,value)==1 :
                     line["passengerid"]=value
                     del line["error_type"]
                     corrected_output.append(line)
                     k=1
                     break
             if k==0 :  cannot_correct.append(line)

         elif  line["error_type"]=='flightid_error' :
             lists=[[i["flightid"],i['departurecode'],i['destinationcode']] for i in clean_data]
             for value in lists :
                 if hamming_distance(line["flightid"]+line['departurecode']+line['destinationcode']
,value[0]+value[1]+value[2])==1 :
                     line["flightid"]=value[0]
                     del line["error_type"]
                     corrected_output.append(line)
                     k=1
                     break
             if k==0 :  cannot_correct.append(line)
         elif  line["error_type"]=='departurecode_error' :
             lists=[[l["flightid"],l['departurecode'],l['destinationcode']] for l in clean_data]
             for value in list(lists) :
                 if hamming_distance(line["flightid"]+line['departurecode']+line['destinationcode']
,value[0]+value[1]+value[2])==1 :
                     line["departurecode"]=value[1]
                     del line["error_type"]
                     corrected_output.append(line)
                     k=1
                     break
             if k==0 :  cannot_correct.append(line)
         elif  line["error_type"]=='destinationcode_error' :
             lists=[[n["flightid"],n['departurecode'],n['destinationcode']] for n in clean_data]
             for value in list(lists) :
                 if hamming_distance(line["flightid"]+line['departurecode']+line['destinationcode']
,value[0]+value[1]+value[2])==1 :
                     line["destinationcode"]=value[2]
                     del line["error_type"]
                     corrected_output.append(line)
                     k=1
```

```
                break
            if k==0 :  cannot_correct.append(line)
        else :
            cannot_correct.append(line)

    return corrected_output,cannot_correct
def preprocess(input) :
    cleandata,errordata=detect_error_flights(input)
    corrected_data,cannotfixed=correct_error(errordata,cleandata)
    return cleandata+corrected_data


#mapper_Q1
def mapper_Q1(mapper_input,key_column) :
    mapper_output=[]
    if len(mapper_input)<>0 :
        for line in mapper_input :
            key=line[key_column]
            if "flightid" in line.keys() :
                key_value=line["flightid"]
            else :
                key_value=None
            mapper_output.append({key:key_value})
    else :
        mapper_output=[]
    return mapper_output

#mapper-Q2
def mapper_Q2(mapper_input) :
    mapper_output=[]
    if len(mapper_input)<>0 :
        for line in mapper_input :
            F_object=Flights(**line)   #class instance
            depart_time=F_object.depart_time(line['departuretime'])
            arrival_time=F_object.arrival_time(line['departuretime'],line['totaltime'])
            flight_time=F_object.fight_time(line['totaltime'])

mapper_output.append({(F_object.flightid,F_object.departurecode,F_object.destinationcode,depart_time,arrival_time,flight_time):F_object.passengerid})
    else :
        mapper_output=[]
    return mapper_output

#mapper-Q4
def mapper_Q4(mapper_input1,mapper_input2) :
    mapper_output=[]
    if len(mapper_input1)<>0 :
        for line1 in mapper_input1 :
            line1_extend={}
            for line2 in mapper_input2:
                if line1['departurecode']==line2['airportcode'] :
                    line1.update({'departure_lat':line2['Latitude'],'departure_lon':line2['Longitude']})
                if line1['destinationcode']==line2['airportcode'] :
                    line1.update({'destination_lat':line2['Latitude'],'destination_lon':line2['Longitude']})
            line1_extend=line1
            if len(line1_extend)==10  : # to make sure have matched both longitude and latitude
                F_object=Flights_Distance(**line1_extend)   #class instance
```

```
Distance=F_object.distance(F_object.departure_lat,F_object.departure_lon,F_object.destination_lat,F_object.destination_lon)
                mapper_output.append({F_object.passengerid:Distance})
    else :
        mapper_output=[]
    return mapper_output


#shuffling
def shuffling(shuffling_input) :
    shuffling_output={}
    if len(shuffling_input)<>0 :
       for line in shuffling_input :
           if line.keys()[0] not in shuffling_output :
              shuffling_output[line.keys()[0]]=[line.values()[0]]
           else :
              shuffling_output[line.keys()[0]].append(line.values()[0])
    else :
       shuffling_output={}
    return shuffling_output


#reducer_Q1
def reducer_Q1(reducer_input) :
    reducer_output={}
    if len(reducer_input.keys())<>0 :
       for key, values in reducer_input.iteritems() :
           reducer_output[key]=len(set(filter(None,values)))
    else :
       reducer_output={}
    return reducer_output    # dictionary


#reducer_Q2
def reducer_Q2(reducer_input) :
    reducer_output={}
    if len(reducer_input.keys())<>0 :
       for key, values in reducer_input.iteritems() :
           reducer_output[key]=list(set(values))
    else :
       reducer_output={}
    return reducer_output    # dictionary

def reducer_Q3(reducer_input) :
    reducer_output={}
    if len(reducer_input.keys())<>0 :
       for key, values in reducer_input.iteritems() :
           key=(key[0],key[1],key[2])
           reducer_output[key]=len(values)
    else :
       reducer_output={}
    return reducer_output

def reducer_Q4(reducer_input) :
    reducer_output={}
    if len(reducer_input.keys())<>0 :
        for key, values in reducer_input.iteritems() :
            reducer_output[key]=sum(values)
    else :
        reducer_output={}
    return reducer_output    # dictionary

#exporter
def exporter(exporter_input,filename) :
    print("\n".join('{}:{}'.format(key,value) for key,value in sorted(exporter_input.items(),key =
lambda (k,v): v, reverse=True)))
    with open(filename,'w') as exporter_file:
        for key,values in sorted(exporter_input.items(),key = lambda (k,v): v, reverse=True) :
            exporter_file.write(key+":"+str(values)+"\n")

def exporter_Q2(exporter_input,filename) :
    with open(filename,'w') as exporter_file:
        for key,values in sorted(exporter_input.items()) :
            print(list(key))
            print("\n".join(values))
            exporter_file.write(' '.join(key))
            exporter_file.write('\n')
            exporter_file.write('\n'.join(values))
            exporter_file.write('\n')

def exporter_Q3(exporter_input,filename) :
    print("\n".join('{}:{}'.format(key,value) for key,value in sorted(exporter_input.items())))
    with open(filename,'w') as exporter_file:
        for key,values in sorted(exporter_input.items()) :
            exporter_file.write(' '.join(key)+":"+str(values)+"\n")

def exporter_errors(exporter_input,filename) :
    with open(filename,'w') as exporter_file:
        for line in sorted(exporter_input) :
            exporter_file.write(" ".join([line['passengerid'],line['flightid'] ,line['departurecode']
,line['destinationcode'],line['departuretime'],line['totaltime']]))
            exporter_file.write("\n")
###########################################################################
# Question 1
def run_Q1() :

    #step0- reading data
    file1=file1box.get()
    file2=file2box.get()
    file_Q1=file3box.get()+r"/Question1.txt"
    reader_output1=reader(file1,Header_flights)
    reader_output2=reader(file2,Header_airport)
    # step1- pre-processing data
    pre_output1=preprocess(reader_output1)
    pre_output2=detect_error_airport(reader_output2)

    # step2-mapper
    mapper_output1=mapper_Q1(pre_output1,'departurecode')
    mapper_output2=mapper_Q1(pre_output2,'airportcode')
    mapper_output=mapper_output1+mapper_output2

    # step3-#shuffling
    shuffling_output=shuffling(mapper_output)
    # step4-#reducer
    reducer_output=reducer_Q1(shuffling_output)
    textbox.insert(END,"\n".join('{}'.format(line) for line in sorted(reducer_output)))
    #step5-#exporter
    exporter(reducer_output,file_Q1)
    textbox.delete(0.0,END)
    textbox.insert(END,"\n".join('{}:{}'.format(key,value) for key,value in
sorted(reducer_output.items(),key = lambda (k,v): v, reverse=True)))
# ############################################################################
# Answer files

#Question 2
def run_Q2() :
    # step-0 reading data
    file1=file1box.get()
    file_Q2=file3box.get()+r"/Question2.txt"
    reader_output1=reader(file1,Header_flights)   # list of dictionary
    # step-1- pre-processing data
    pre_output1=preprocess(reader_output1)
    # step-2 mapping flight information by passengerid instead of 1
    mapper_output=mapper_Q2(pre_output1)
    # step-3 shuffling
    shuffling_output=shuffling(mapper_output)
    # step-4 reduce the same passengerid
    reducer_output=reducer_Q2(shuffling_output)
    #step5-#exporter
    exporter_Q2(reducer_output,file_Q2)
    textbox.delete(0.0,END)
    for key,values in sorted(reducer_output.items()) :
        textbox.insert(END,list(key))
        textbox.insert(END,"\n")
        textbox.insert(END,"\n".join(values))
        textbox.insert(END,"\n")
#########################################################################

#Question 3
def run_Q3() :
    # step-0 reading data
    file1=file1box.get()
    file_Q3=file3box.get()+r"/Question3.txt"
    reader_output1=reader(file1,Header_flights)   # list of dictionary
    # step-1- pre-processing data
    pre_output1=preprocess(reader_output1)
    # step-2 mapping flight information by passengerid instead of 1
    mapper_output=mapper_Q2(pre_output1)
    # step-3 shuffling
    shuffling_output=shuffling(mapper_output)
    # step-4 reduce the same passengerid
    reducer_output=reducer_Q3(shuffling_output)
    #step5-#exporter
    exporter_Q3(reducer_output,file_Q3)
    textbox.delete(0.0,END)
    textbox.insert(END,"\n".join('{}:{}'.format(key,value) for key,value in
sorted(reducer_output.items())))
############################################################################
#Question 4
def run_Q4() :
    # step0- reading data
    file1=file1box.get()
    file2=file2box.get()
    file_Q4=file3box.get()+r"/Question4.txt"
    reader_output1=reader(file1,Header_flights)
    reader_output2=reader(file2,Header_airport)
    # step1- pre-processing data
    pre_output1=preprocess(reader_output1)
    pre_output2=detect_error_airport(reader_output2)
    # step-2 mapping  passengerid by Distance value instead of 1
    mapper_output=mapper_Q4(pre_output1,pre_output2)
    # step-3 shuffling
    shuffling_output=shuffling(mapper_output)
    # step-4 reduce the same passengerid
    reducer_output=reducer_Q4(shuffling_output)
    # #step5-#exporter
    exporter(reducer_output,file_Q4)
    textbox.delete(0.0,END)
    textbox.insert(END,"\n".join('{}:{}'.format(key,value) for key,value in
sorted(reducer_output.items(),key = lambda (k,v): v, reverse=True)))
# ######################################################################
# error/corrected print
def run_errors_detected() :
    file1=file1box.get()
    error_file=file3box.get()+"/errors.txt"
    reader_output=reader(file1,Header_flights)
    cleandata,errordata=detect_error_flights(reader_output)

    exporter_errors(errordata,error_file)
    textbox.delete(0.0,END)
    textbox.insert(END,"The total number errors detected :"+str(len(errordata))+"\n")
    textbox.insert(END,"\n".join('{}'.format([line['passengerid'],line['flightid']]
,line['departurecode'] ,line['destinationcode'],line['departuretime'],line['totaltime']]) for line in
sorted(errordata)))

def run_error_corrected() :
    file1=file1box.get()
    corrected_file=file3box.get()+"/errors_corrected.txt"
    reader_output=reader(file1,Header_flights)
    cleandata,errordata=detect_error_flights(reader_output)
    corrected_data,cannotfixed=correct_error(errordata,cleandata)
    exporter_errors(corrected_data,corrected_file)

    textbox.delete(0.0,END)
    textbox.insert(END,"The total number errors corrected :"+str(len(corrected_data))+"\n")
    textbox.insert(END,"\n".join('{}'.format([line['passengerid'],line['flightid']]
,line['departurecode'] ,line['destinationcode'],line['departuretime'],line['totaltime']]) for line in
sorted(corrected_data)))

def inputfile1() :
    file = tkFileDialog.askopenfile(parent=window,mode='rb',title='Choose the file of
AComp_Passenger_data')
    file1box.delete(0, END)
    file1box.insert(0, file.name)
def inputfile2() :
    file = tkFileDialog.askopenfile(parent=window,mode='rb',title='Choose a file of
Top30_airports_LatLong')
```

```
    file2box.delete(0, END)
    file2box.insert(0, file.name)
def outputfolder() :
    file = tkFileDialog.askdirectory(parent=window,title='Choose a folder')
    file3box.delete(0, END)
    file3box.insert(0, str(file))
#################interface configuration#######################
window=Tk()
# define four labels title author year isbn
window.title("26801076Coursework")
window.geometry("600x600+200+200")


# input files
button0_1=Button(window, text="Browse file of AComp_Passenger_data",width=30,command=inputfile1)
button0_1.pack()
f1=StringVar(None)
file1box=Entry(window,textvariable=f1)
file1box.pack()

button0_2=Button(window, text="Browse file of Top30_airports_LatLong", width=30, command=inputfile2)
button0_2.pack()
f2=StringVar(None)
file2box=Entry(window,textvariable=f2)
file2box.pack()
# outputfile
button0_3=Button(window, text="Browse folder to save answers",width=30,command=outputfolder)
button0_3.pack()
f3=StringVar(None)
file3box=Entry(window,textvariable=f3)
file3box.pack()

# run answer button
labelText=StringVar()
labelText.set("Please press the button: ")
label1=Label(window,textvariable=labelText,height=2)
label1.pack()
button1=Button(window, text="Question_1", width=15,command=run_Q1).pack()
button2=Button(window, text="Question_2", width=15,command=run_Q2).pack()
button3=Button(window, text="Question_3", width=15,command=run_Q3).pack()
button4=Button(window, text="Question_4", width=15,command=run_Q4).pack()
button5=Button(window, text="Errors_Detected", width=15,command=run_errors_detected).pack()
button6=Button(window, text="Errors_Corrected", width=15,command=run_error_corrected).pack()
# text box
scrollbar = Scrollbar(window)
scrollbar.pack(side=RIGHT, fill=Y)
textbox=Text(window,width=80,height=22,wrap=NONE,background="LightYellow",yscrollcommand=scrollbar.set)
textbox.pack()
scrollbar.config(command=textbox.yview)
```

```
window.mainloop()
```

Version Control

```
commit 2c9170ff42c0221d6a1b5ad8045f60bade11e5c0 (HEAD ->
master, origin/master)
Author: Jin Li <fx801076@live.reading.ac.uk>
Date:   Mon Feb 19 23:45:51 2018 +0000

    amend the answer of question1

commit 1f6ecd4c5a57e4a3c8c3ad75e7908c7675546702
Author: Jin Li <fx801076@live.reading.ac.uk>
Date:   Mon Feb 12 08:37:03 2018 +0000

    generate exe

commit 1ac25e372fc46f41e5f629dad95b7b5fa53fa78c
Author: Jin Li <fx801076@live.reading.ac.uk>
Date:   Tue Feb 6 12:48:39 2018 +0000

    change a little of CC_26801076.py

commit 9ca8a2262b56492ddc1f151f005c49cff52cf273
Author: Jin Li <fx801076@live.reading.ac.uk>
Date:   Tue Feb 6 09:29:40 2018 +0000

    change interface
:...skipping...
```