```
#!/usr/bin/env python3
import sys
# Task 1 count the number of trips and the average distance pre trip for
each taxi
for line in sys.stdin:
    line = line.strip()
    trip = line.split(",")

    try:
        if trip[1].strip() and trip[3].strip(): #trip[1] is taxi no while
trip[3] is distance
            trip[1] = int(trip[1])
            trip[3] = float(trip[3])
            print('%d\t%f' % (trip[1],trip[3]))
    except:
        continue#!/usr/bin/env python3
import sys

# init var
current_taxi = None
current_distance = 0.0
taxi = None
count = 0

# read lines from sorted mapper
for line in sys.stdin:
    line = line.strip()
    taxi,distance = line.split('\t')

    try:
        distance = float(distance)
    except ValueError:
        # trash lines if distance cannot be converted to float quietly
        continue

    if current_taxi == taxi:
        # sum up distance and add counter for same taxi
        current_distance += distance
        count += 1
    else:
        # output taxi, count, average distance in 2 decimal places
        if current_taxi:
            print('%s,%d,%.2f' % (current_taxi, count,
current_distance/count))
        current_distance = distance
        current_taxi = taxi
        count = 1

# print last row
if current_taxi == taxi:
    print('%s,%d,%.2f' % (current_taxi, count,
current_distance/count))#!/bin/bash

hadoop fs -rm -f -r /input
hadoop fs -rm -f -r /output/task1

hadoop fs -mkdir /input
hadoop fs -put ./Taxis.txt /input/Taxis.txt
```

```
hadoop fs -put ./Trips.txt /input/Trips.txt

hadoop jar ./hadoop-streaming-3.1.4.jar \
-D mapred.reduce.tasks=3 \
-file ./task1_mapper.py \
-mapper ./task1_mapper.py \
-file ./task1_reducer.py \
-reducer ./task1_reducer.py \
-input /input/Trips.txt \
-output /output/task1

#hadoop fs -getmerge /output/task1/part* task1_merged_output.txt
#!/bin/bash

hadoop fs -rm -f -r /input
hadoop fs -rm -f -r /output/task1

hadoop fs -mkdir /input
hadoop fs -put .Taxis.txt /input/Taxis.txt
hadoop fs -put .Trips.txt /input/Trips.txt

# specify sort by integer with KeyFieldBasedComparator -n : numeric sort,
-r : reverse sort
# ref: https://stackoverflow.com/questions/13331722/how-to-sort-
numerically-in-hadoops-shuffle-sort-phase
hadoop jar .hadoop-streaming-3.1.4.jar \
-D mapred.reduce.tasks=3 \
-D
mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldB
asedComparator \
-D  mapred.text.key.comparator.options=-n \
-file ./task1-mapper.py \
-mapper ./task1-mapper.py \
-file ./task1-reducer.py \
-reducer ./task1-reducer.py \
-input /input/Trips.txt \
-output /output/task1

hadoop fs -getmerge /output/task1/part* task1_merged_output.txt

#!/usr/bin/env python3
"""task2-mapper.py"""

import sys
from math import sqrt

# get initial medoids from a txt file and add them in an array
def getMedoids(filepath):
    medoids = []

    # read medoids from txt file
    with open(filepath) as fp:
        line = fp.readline()
        while line:
            if line:
                try:
                    line = line.strip()
                    cord = line.split(',')
```

```
                    # cord[0] is x and cord[1] is y point of a medoid
                    medoids.append([float(cord[0].strip()),
float(cord[1].strip())])
                except:
                    break
            else:
                break
            line = fp.readline()
    fp.close()
    return medoids

# create clusters based on initial medoids
def createClusters(medoids):
    #
    for line in sys.stdin:
        line = line.strip()
        cord = line.split(',')[4:6] # only extract pickup_x and pickup_y
of Trips.txt
        min_dist = float('inf') # init minimum distance with infinity
        index = -1

        try:
            cord[0] = float(cord[0])
            cord[1] = float(cord[1])
        except ValueError:
            # float was not a number, so silently
            # ignore/discard this line
            continue

        for medoid in medoids:
            # euclidian distance from every point of dataset
            # to every medoid
            cur_dist = sqrt(pow(cord[0] - medoid[0], 2) + pow(cord[1] -
medoid[1], 2))

            # find the medoid which is closer to the point
            if cur_dist <= min_dist:
                min_dist = cur_dist
                index = medoids.index(medoid)

        var = "%s\t%s\t%s" % (index, cord[0], cord[1])
        print(var)

if __name__ == "__main__":
    medoids = getMedoids('medoids.txt')
    createClusters(medoids)
#!/usr/bin/env python3

from task2_mapper import getMedoids

#check if distance of medoids and medoids_prev have same values
def checkMedoidsDistance(medoids, medoids_prev):
    bool = True
    for idx, medoid in enumerate(medoids):
        if bool:
            bool = ((medoids[idx][0] == medoids_prev[idx][0]) and
(medoids[idx][1] == medoids_prev[idx][1]))
        else:
```

```
            break

    if bool:
        print(1)
    else:
        print(0)

if __name__ == "__main__":
    medoids = getMedoids('medoids.txt')
    medoids_prev = getMedoids('medoids_prev.txt')

    checkMedoidsDistance(medoids, medoids_prev)
#!/usr/bin/env python3
"""task2_reducer.py"""

import sys
from math import sqrt

def convert_line(line):
    medoid_idx, x, y = line.split('\t')

    # convert x and y (currently a string) to float
    try:
        x = float(x)
        y = float(y)
        obj = (x, y)

        return medoid_idx, obj
    except:
        # float was not a number, so silently
        # ignore/discard this line
        return None, None

def calculateMinMedoids(lines, medoid_idx, tmp_medoid_obj, min_dist):
    # looping lines for comparing dist within medoid
    prev_inner_obj = None
    tmp_dist = 0
    for inner_line in lines:
        inner_medoid_idx, inner_obj = convert_line(inner_line)

        # read line without error
        if inner_medoid_idx is not None:

            # handle inner obj associate with same medoid only
            if medoid_idx == inner_medoid_idx:

                # calculate distance for obj diff with prev_obj only
                # as same obj has 0 distance
                if inner_obj != prev_inner_obj or prev_inner_obj is None:
                    # euclidian distance from every point of dataset
                    # to every medoid
                    cur_dist = sqrt(pow(inner_obj[0] - tmp_medoid_obj[0],
2) + pow(inner_obj[1] - tmp_medoid_obj[1], 2))
                    tmp_dist += cur_dist

                    # move to next outer line as this inner line distance
already longer than minimum
                    # for faster processing only
```

```
                    if tmp_dist > min_dist:
                        break
                prev_inner_obj = inner_obj
            else:
                # skip wtih different medoid for inner obj
                continue
    if tmp_dist < min_dist:
        # return shorter distance and new medoid
        return tmp_dist, tmp_medoid_obj
    # return original minimum distance and None for medoid
    return min_dist, None

def calculateNewMedoids():
    current_medoid_idx = None
    # init min medoid for outer loop
    min_dist = float('inf') # init minimum distance with infinity
    min_medoid_obj = None

    # input comes from STDIN to lines array
    lines = sys.stdin.readlines()
    for line in lines:
        # parse the input of mapper.py
        medoid_idx, curr_obj = convert_line(line)

        # read line all good
        if medoid_idx is not None:
            # init min medoid for outer loop
            tmp_medoid_obj = curr_obj

            if current_medoid_idx == medoid_idx or current_medoid_idx is
None:
                # to calculate minimum medoids within the same cluster
                # if None for tmp_min_obj is returned, it means the
medoid does not change
                min_dist, tmp_min_obj = calculateMinMedoids(lines,
medoid_idx, tmp_medoid_obj, min_dist)
                if tmp_min_obj is not None:
                    min_medoid_obj = tmp_min_obj

                # assign medoid_idx to current_medoid_idx
                current_medoid_idx = medoid_idx
            else:
                # handle diff medoid
                # print out new medoid for a cluster
                print(str(min_medoid_obj[0]) + ", " +
str(min_medoid_obj[1]))

                # reset min_dist & min_medoid for next cluster
                min_dist = float('inf')
                min_medoid_obj = None
                current_medoid_idx = medoid_idx
                tmp_medoid_obj = curr_obj

                # to calculate minimum medoids within the same cluster
                min_dist, tmp_min_obj = calculateMinMedoids(lines,
medoid_idx, tmp_medoid_obj, min_dist)
                if tmp_min_obj is not None:
                    min_medoid_obj = tmp_min_obj
```

```
    # print last cluster's medoids
    if current_medoid_idx == medoid_idx and min_dist != float('inf'):
        print(str(min_medoid_obj[0]) + ", " + str(min_medoid_obj[1]))

if __name__ == "__main__":
    calculateNewMedoids()
#!/bin/bash

#############################################################################
#######
# Help
#
#############################################################################
#######
Help()
{
   # Display Help
   echo
   echo "required arguments:"
   echo "k     arg of k-medoid (must be a number) e.g. -k 2"
   echo "v     arg of number of iteration (must be a number)  e.g. -v 10"
   echo
}

num_re='^[0-9]+$'

while getopts k:v: flag
do
    case "${flag}" in
        k)
          k=${OPTARG};;
        v)
          v=${OPTARG};;
        \?) # incorrect option
         echo "Error: Invalid option"
         exit;;
    esac
done

# required arg -k
if  [ -z "$k" ] ; then
  Help
  exit;
fi
# required arg -k with a number
if ! [[ $k =~ $num_re ]] ; then
  Help
  exit;
fi
# required arg -v
if [ -z "$v" ] ; then
  Help
  exit;
fi
# required arg -v with a number
if ! [[ $v =~ $num_re ]] ; then
  Help
```

```
  exit;
fi

rm -f .*.crc # handle bugs of hadoop

# for using first k-lines as medoids
#rm -f medoids.txt
#head -$k Trips2.txt | awk -F "," '{print $5 ", " $6}' > medoids.txt

hadoop fs -rm -f -r /input
hadoop fs -rm -f -r /output/task2

hadoop fs -mkdir /input
hadoop fs -put ./Taxis.txt /input/Taxis.txt
hadoop fs -put ./Trips.txt /input/Trips.txt

i=1
while :
do
      hadoop jar ./hadoop-streaming-3.1.4.jar \
    -D mapred.reduce.tasks=3 \
    -D mapred.text.key.partitioner.options=-k1 \
    -file medoids.txt \
    -file ./task2_mapper.py \
    -mapper ./task2_mapper.py \
    -file ./task2_reducer.py \
    -reducer ./task2_reducer.py \
    -input /input/Trips.txt \
    -output /output/task2/output$i \
    -partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner

    rm -f .*.crc
    # rename medoid as prev medoids and merge new one from hdfs to master
node
    mv medoids.txt medoids_prev.txt
    hadoop fs -getmerge /output/task2/output$i/part-* medoids.txt

    # to check metoid is same as previous metoid
    seeiftrue=`python3 task2_reader.py`

  # if metoid is same as previous metoid or loop number greater than $v
end program
      if [ $seeiftrue == 1 ] || [ $i -ge $v ]
      then
            break
      fi
      i=$((i+1))
done

#!/usr/bin/env python3
import sys
# Task 3 subtask mapper join
# this subtask reads both Taxis.txt and Trips.txt and extract their
meaningful attributes
# i.e. Taxi#, company from Taxis.txt and Taxi#, Trip# from Trips.txt

for line in sys.stdin:
    line = line.strip()
```

```
    splits = line.split(",") # spliting fields with separators ","

    try:
        # Use no. of splits to distinguish the line is from which
documents.
        if len(splits) == 4: # Taxi
            splits[0] = int(splits[0]) # Taxi#
            splits[1] = int(splits[1]) # company
            # stdout all lines with new formats "Taxi#,company,Trip"
            # use "~" to represent empty value
            print('%d,%d,%s' % (splits[0],splits[1],"~"))
        else: # Trip
            splits[1] = int(splits[1]) # Taxi#
            splits[0] = int(splits[0]) # Trip#
            # stdout all lines with new formats "Taxi#,company,Trip"
            # use "~" to represent empty value
            print('%d,%s,%d' % (splits[1],"~",splits[0]))
    except:
        continue#!/usr/bin/env python3
import sys

# Task 3 subtask join reducer
# this subtask reduces the lines from mapper-join by aggregatting the
count by same taxi
# as a taxi is only belongs to a company, the count of each taxi can be
represented with company as key
# output format company,count
# each line represents a single taxi count of trip

current_company = None
current_taxi = None
count = 0
for line in sys.stdin:
    line = line.strip()
    taxi,company,trip = line.split(",")    # Taxi#,company,Trip


    if current_taxi != taxi:
        if current_taxi is not None: # to exclude None company count
(i.e. the first line)
            # output the taxi of previous line and its count as the
previous line is not the same taxi
            print('%s,%d' % (current_company, count))
        # assign this line to current attribute and reset counter
        current_taxi = taxi
        current_company = company
        count = 0

    # "~" is the empty value
    # if company is empty, it means the line is from trip
    # therefore, it should be count as 1 trip
    if company == "~":
        count += 1

# to print the last taxi count
if current_taxi == taxi:
    print('%s,%d' % (current_company, count))#!/usr/bin/env python3
import sys
```

```
# Task 3 part 2 mapper
# format <company>,<count>
# nothing to change for this mapper
# just stdout the same to partioner, sorter and reducer.

for line in sys.stdin:
    line = line.strip()
    company, count = line.split(",") # Line format: <company>,<count>

    try:
        if company.strip() and count.strip():
            company= int(company)
            count = int(count)
            print('%d,%d' % (company, count))
    except:
        continue#!/usr/bin/env python3
import sys

# Task 3 subtask reducer
# this subtask reduces the lines from mapper (actually is from reducer-
join as mapper did nothing)
# by aggregatting the count by same company
# a line represents the count of trips from a taxi with its company
# this reducer to aggregrate the trip count for each company
# output format company,count
# the output of each line represents a trip count of each company

# param initialzation
current_company = None
total_count = 0

for line in sys.stdin:
    line = line.strip()
    company, count = line.split(",") # Line format: <company>,<count>

    if current_company != company:
        if current_company is not None: # to exclude None company count
(i.e. the first line)
            # output the company of previous line and its count as the
previous line is not the same company
            print('%s,%d' % (current_company, total_count))
        # assign this line to current attribute and reset counter
        current_company = company
        total_count = int(count)
    else:
        # if the current line company (i.e. company) is same as previous
line company (i.e. current_company),
        # aggregate the count
        try:
            # convert count to int
            count = int(count)
            total_count += count
        except:
            continue

# to print the last taxi count
if current_company == company:
    print('%s,%d' % (current_company, total_count))#!/bin/bash
```

```
hadoop fs -rm -f -r /input
hadoop fs -rm -f -r /task3
hadoop fs -rm -f -r /output/task3

hadoop fs -mkdir /input
hadoop fs -put ./Taxis.txt /input/Taxis.txt
hadoop fs -put ./Trips.txt /input/Trips.txt

# composite keys are taxi & company
# use -k1 as partitioner key which is taxi
hadoop jar ./hadoop-streaming-3.1.4.jar \
-D stream.num.map.output.key.fields=2 \
-D map.output.key.field.separator=, \
-D mapred.text.key.partitioner.options=-k1,1 \
-D mapred.reduce.tasks=3 \
-file ./task3_1_mapper.py \
-mapper ./task3_1_mapper.py \
-file ./task3_1_reducer.py \
-reducer ./task3_1_reducer.py \
-input /input/Trips.txt \
-input /input/Taxis.txt \
-output /task3 \
-partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner

hadoop fs -getmerge /task3/part* ./task3_join_output.txt
hadoop fs -put ./task3_join_output.txt /input/task3_join_output.txt
hadoop fs -rm -f -r /task3

# only key is company and it is the partition key
# company partition key to make sure same key values are in the same
reducer
hadoop jar ./hadoop-streaming-3.1.4.jar \
-D stream.num.map.output.key.fields=1 \
-D map.output.key.field.separator=, \
-D mapred.text.key.partitioner.options=-k1,1 \
-D mapred.reduce.tasks=3 \
-file ./task3_2_mapper.py \
-mapper ./task3_2_mapper.py \
-file ./task3_2_reducer.py \
-reducer ./task3_2_reducer.py \
-input /input/task3_join_output.txt \
-output /output/task3 \
-partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner

hadoop fs -getmerge /output/task3/part* ./task3_output.txt
hadoop fs -put ./task3_output.txt /output/task3/task3_output.txt
```