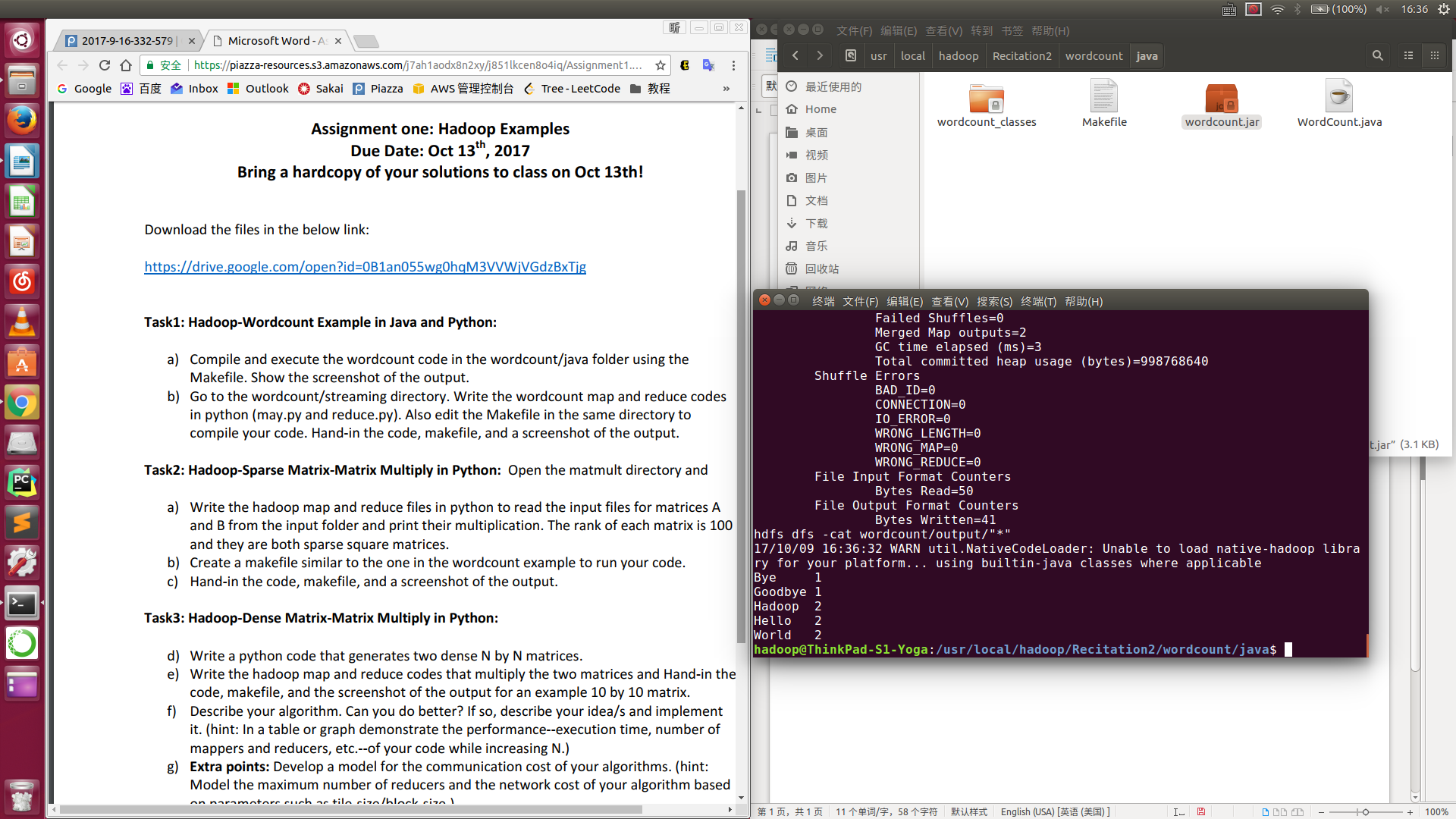
**Homework 1 for Cloud Computing**

Xin Yang

NetID: xy213

**Task1:**

**a) screenshot:**



**b)**

**makefile:**

-hdfs dfs -rm -f -r $(OUTPUT\_DIR)

hadoop jar $(TOOLLIBS\_DIR)/hadoop-streaming-$(HADOOP\_VERSION).jar \

-files ./map.py,./reduce.py \

-mapper ./map.py \

-reducer ./reduce.py \

-input $(INPUT\_DIR) \

-output $(OUTPUT\_DIR)

hdfs dfs -cat $(OUTPUT\_FILE)

run-2reducers: inputs

-hdfs dfs -rm -f -r $(OUTPUT\_DIR)

hadoop jar $(TOOLLIBS\_DIR)/hadoop-streaming-$(HADOOP\_VERSION).jar \

-D mapred.reduce.tasks=2 \

-files ./map.py,./reduce.py \

-mapper ./map.py \

-reducer ./reduce.py \

-input $(INPUT\_DIR) \

-output $(OUTPUT\_DIR)

echo "===$(OUTPUT\_FILE)==="

hdfs dfs -cat $(OUTPUT\_FILE)

echo "===$(OUTPUT\_FILE\_2)==="

hdfs dfs -cat $(OUTPUT\_FILE\_2)

directories:

hdfs dfs -test -e $(EXAMPLE\_DIR) || hdfs dfs -mkdir $(EXAMPLE\_DIR)

hdfs dfs -test -e $(INPUT\_DIR) || hdfs dfs -mkdir $(INPUT\_DIR)

hdfs dfs -test -e $(OUTPUT\_DIR) || hdfs dfs -mkdir $(OUTPUT\_DIR)

inputs: directories

hdfs dfs -test -e $(INPUT\_DIR)/file01 \

|| hdfs dfs -put ../input-small/file01 $(INPUT\_DIR)/file01

hdfs dfs -test -e $(INPUT\_DIR)/file02 \

|| hdfs dfs -put ../input-small/file02 $(INPUT\_DIR)/file02

clean:

-hdfs dfs -rm -f -r $(INPUT\_DIR)

-hdfs dfs -rm -f -r $(OUTPUT\_DIR)

-hdfs dfs -rm -r -f $(EXAMPLE\_DIR)

.PHONY: directories inputs clean run run-2reducers

**map.py:**

#!/usr/bin/env python

import sys

for line in sys.stdin:

line = line.strip( )

words = line.split( )

for word in words:

print('%s\t%s' % (word,1))

**reduce.py:**

#!/usr/bin/env python

import sys

current\_word = None

current\_count = 0

word = None

for line in sys.stdin:

line = line.strip()

word,count = line.split('\t',1)

try:

count = int(count)

except ValueError:

continue

if current\_word == word:

current\_count +=count

else:

if current\_word:

print('%s\t%s' % (current\_word,current\_count))

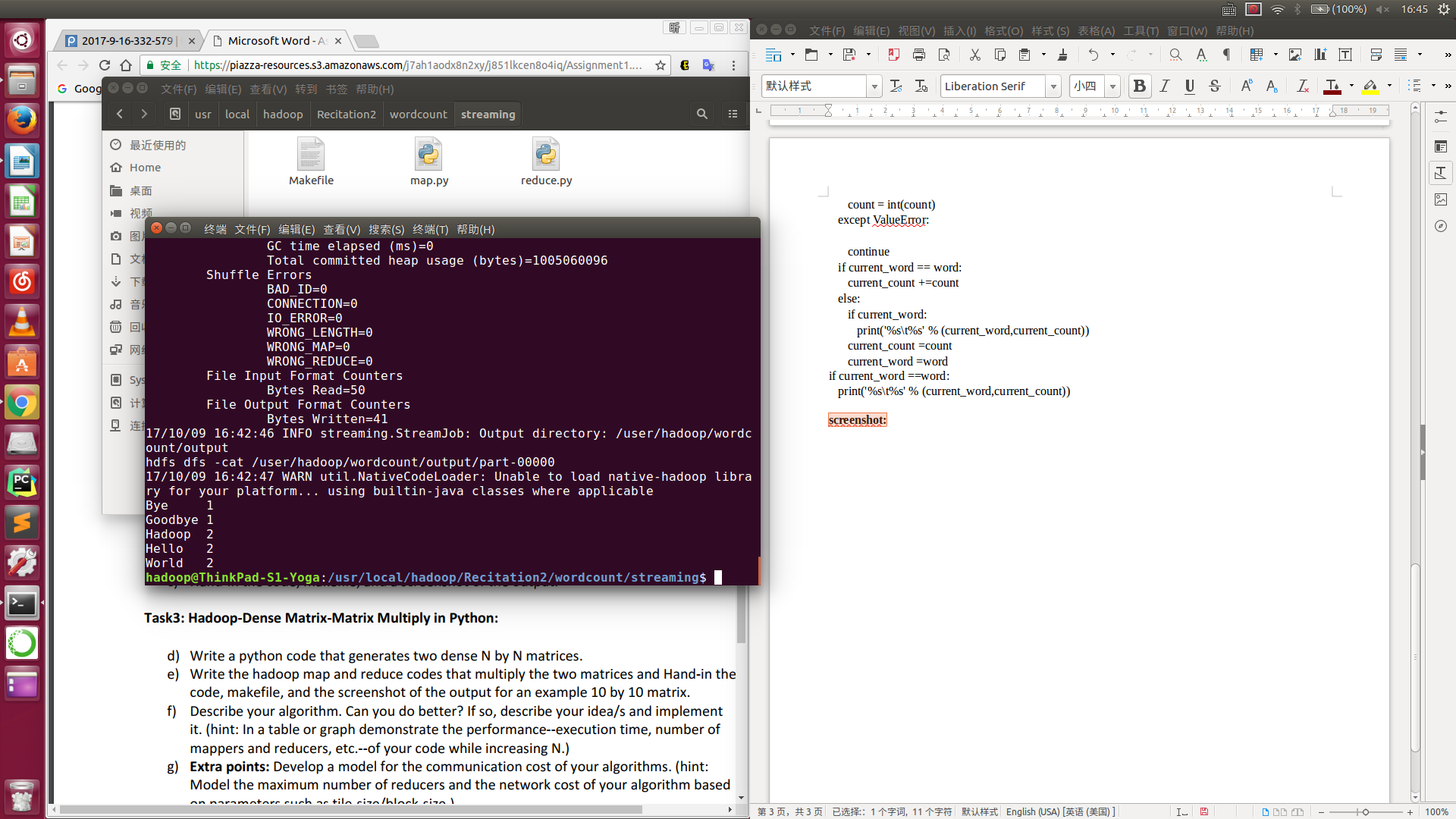
current\_count =count

current\_word =word

if current\_word ==word:

print('%s\t%s' % (current\_word,current\_count))

**screenshot:**



**Task2:**

**a)**

**map.py:**

#!/usr/bin/env python

import sys

for line in sys.stdin:

line = line.strip( )

words = line.split( )

matrix = words[0]

rowindex = words[1]

colindex = words[2]

value = words[3]

if matrix == 'A':

# for i in range(1,101):

for i in range(0,100):

mkey = str(rowindex)+' '+str(i)

mvalue = matrix+' '+str(colindex)+' '+str(value)

print '%s\t%s' % (mkey,mvalue);

elif matrix == 'B':

# for i in range(1,101):

for i in range(0,100):

mkey = str(i)+' '+str(colindex)

mvalue = matrix+' '+str(rowindex)+' '+str(value)

print '%s\t%s' % (mkey,mvalue);

**reduce.py:**

#!/usr/bin/env python

import sys

lastkey = None

result = 0

valA = []

valB = []

for i in range(0,100):

valA.append(0)

valB.append(0)

for line in sys.stdin:

line = line.strip()

key,value = line.split('\t',1)

words = value.split();

if (lastkey != key and lastkey != None):

for i in range(0,100):

result+= valA[i] \* valB[i]

valA[i] = 0

valB[i] = 0

if result != 0:

print '%s\t%d' % (lastkey,result)

lastkey = key

result = 0

if(words[0] == 'A'):

valA[int(words[1])-1] = int(words[2])

if(words[0] == 'B'):

valB[int(words[1])-1] = int(words[2])

lastkey = key

for i in range(0,100):

result+= valA[i] \* valB[i]

valA[i] = 0

valB[i] = 0

if result != 0:

print '%s\t%d' % (lastkey,result)

**b)**

**makefile:**

USER = hadoop

EXAMPLE\_DIR = /user/$(USER)/matmult

INPUT\_DIR = $(EXAMPLE\_DIR)/input

OUTPUT\_DIR = $(EXAMPLE\_DIR)/output

OUTPUT\_FILE = $(OUTPUT\_DIR)/part-00000

HADOOP\_VERSION = 2.7.4

TOOLLIBS\_DIR=$(HADOOP\_HOME)/share/hadoop/tools/lib/

run: inputs

-hdfs dfs -rm -f -r $(OUTPUT\_DIR)

hadoop jar $(TOOLLIBS\_DIR)/hadoop-streaming-$(HADOOP\_VERSION).jar \

-files ./map.py,./reduce.py \

-mapper ./map.py \

-reducer ./reduce.py \

-input $(INPUT\_DIR) \

-output $(OUTPUT\_DIR)

hdfs dfs -cat $(OUTPUT\_FILE)

directories:

hdfs dfs -test -e $(EXAMPLE\_DIR) || hdfs dfs -mkdir $(EXAMPLE\_DIR)

hdfs dfs -test -e $(INPUT\_DIR) || hdfs dfs -mkdir $(INPUT\_DIR)

hdfs dfs -test -e $(OUTPUT\_DIR) || hdfs dfs -mkdir $(OUTPUT\_DIR)

inputs: directories input/part-00000

hdfs dfs -test -e $(INPUT\_DIR)/part-00000 \

|| hdfs dfs -put input/part-00000 $(INPUT\_DIR)/part-00000

clean:

-hdfs dfs -rm -f -r $(INPUT\_DIR)

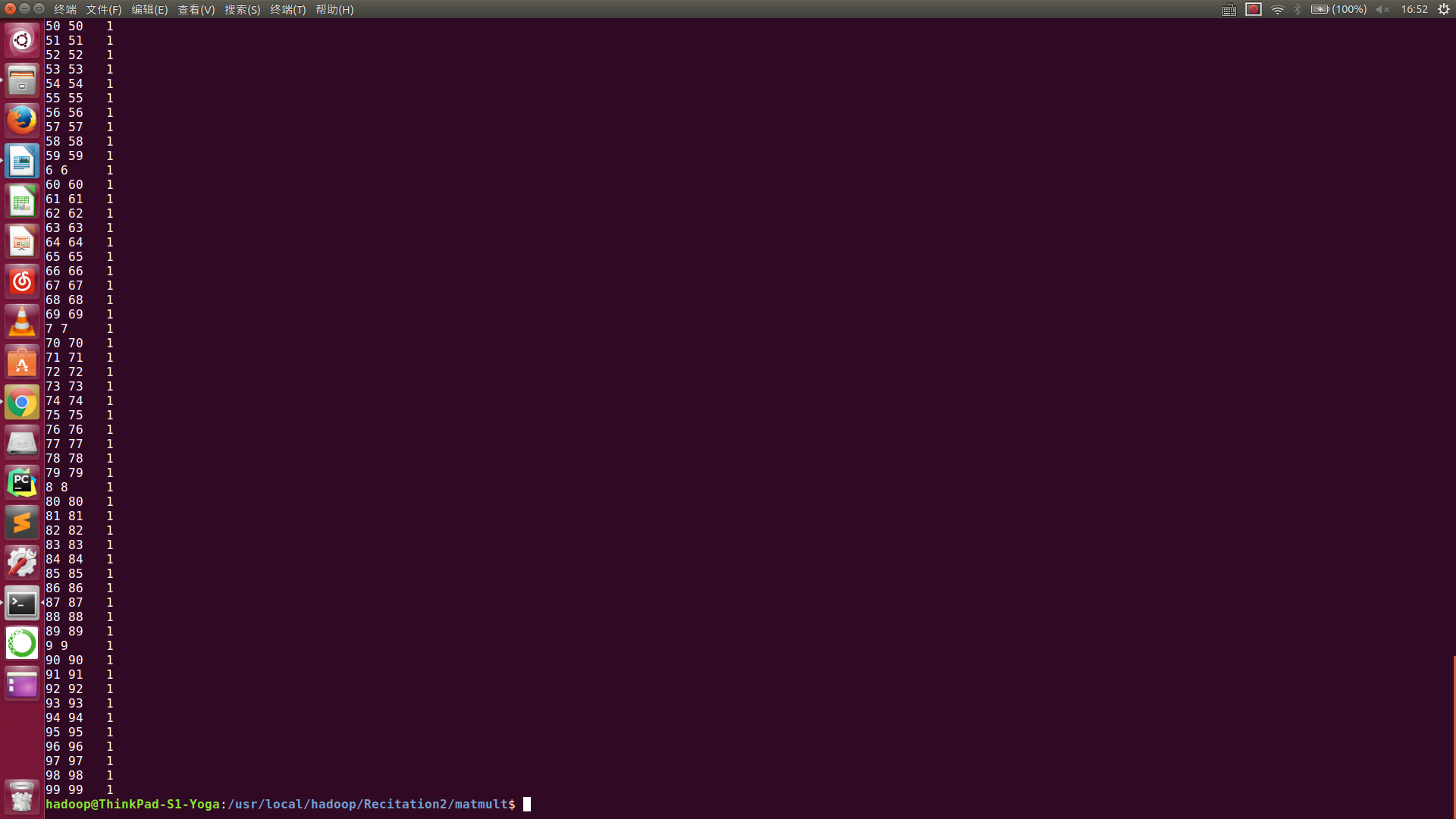
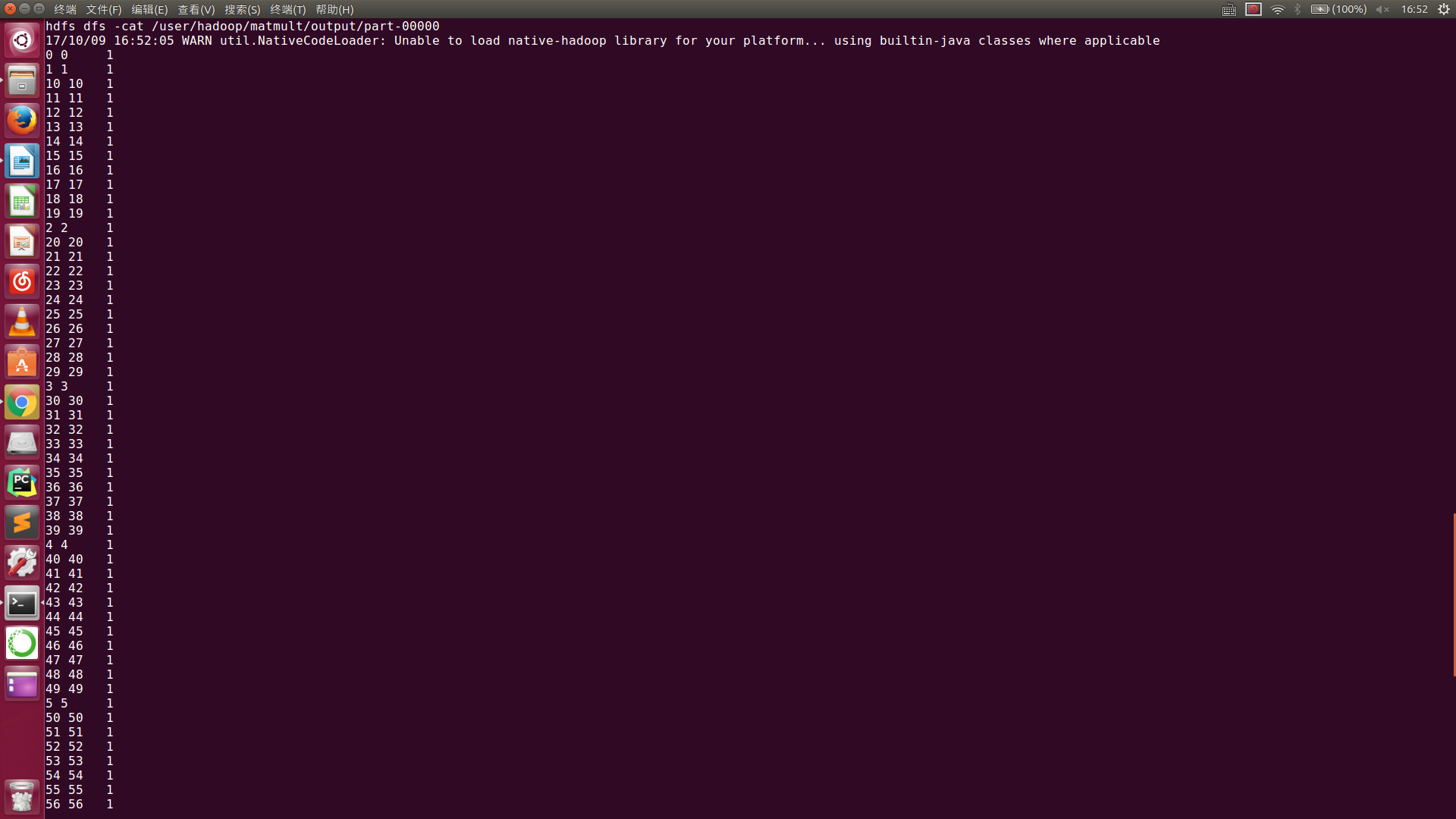
-hdfs dfs -rm -f -r $(OUTPUT\_DIR)

-hdfs dfs -rm -r -f $(EXAMPLE\_DIR)

.PHONY: directories inputs clean run

**c)**

**screenshot:**



**Task3:**

**d)**

**python code:**

import random

import sys

str1 = raw\_input("Please type in the rows/columns of NxN Matrix:")

path = "/usr/local/hadoop/Recitation2/matmult2/input/part-00000"

fo = open(path, "wb")

rc = int(str1)

for i in range(0, rc):

for j in range(0, rc):

stri = str(i)

strj = str(j)

strv = str(int(random.uniform(0, 9)))

if strv != '0':

strres = 'A' + ' ' + stri + ' ' + strj + ' ' + strv + '\n'

fo.write(strres)

print strres

for i in range(0, rc):

for j in range(0, rc):

stri = str(i)

strj = str(j)

strv = str(int(random.uniform(0, 2)))

if strv != '0':

strres = 'B' + ' ' + stri + ' ' + strj + ' ' + strv + '\n'

fo.write(strres)

print strres

fo.close()

**e)**

**makefile:**

USER = hadoop

EXAMPLE\_DIR = /user/$(USER)/matmult2

INPUT\_DIR = $(EXAMPLE\_DIR)/input

OUTPUT\_DIR = $(EXAMPLE\_DIR)/output

OUTPUT\_FILE = $(OUTPUT\_DIR)/part-00000

HADOOP\_VERSION = 2.7.4

TOOLLIBS\_DIR=$(HADOOP\_HOME)/share/hadoop/tools/lib/

run: inputs

-hdfs dfs -rm -f -r $(OUTPUT\_DIR)

hadoop jar $(TOOLLIBS\_DIR)/hadoop-streaming-$(HADOOP\_VERSION).jar \

-files ./map.py,./reduce.py \

-mapper ./map.py \

-reducer ./reduce.py \

-input $(INPUT\_DIR) \

-output $(OUTPUT\_DIR)

hdfs dfs -cat $(OUTPUT\_FILE)

directories:

hdfs dfs -test -e $(EXAMPLE\_DIR) || hdfs dfs -mkdir $(EXAMPLE\_DIR)

hdfs dfs -test -e $(INPUT\_DIR) || hdfs dfs -mkdir $(INPUT\_DIR)

hdfs dfs -test -e $(OUTPUT\_DIR) || hdfs dfs -mkdir $(OUTPUT\_DIR)

inputs: directories input/part-00000

hdfs dfs -test -e $(INPUT\_DIR)/part-00000 \

|| hdfs dfs -put input/part-00000 $(INPUT\_DIR)/part-00000

clean:

-hdfs dfs -rm -f -r $(INPUT\_DIR)

-hdfs dfs -rm -f -r $(OUTPUT\_DIR)

-hdfs dfs -rm -r -f $(EXAMPLE\_DIR)

.PHONY: directories inputs clean run

**map.py:**

#!/usr/bin/env python

import sys

for line in sys.stdin:

line = line.strip( )

words = line.split( )

matrix = words[0]

rowindex = words[1]

colindex = words[2]

value = words[3]

if matrix == 'A':

for i in range(0,10):

mkey = str(rowindex)+' '+str(i)

mvalue = matrix+' '+str(colindex)+' '+str(value)

print '%s\t%s' % (mkey,mvalue);

elif matrix == 'B':

for i in range(0,10):

mkey = str(i)+' '+str(colindex)

mvalue = matrix+' '+str(rowindex)+' '+str(value)

print '%s\t%s' % (mkey,mvalue);

**reduce.py:**

#!/usr/bin/env python

import sys

lastkey = None

result = 0

valA = []

valB = []

for i in range(0,10):

valA.append(0)

valB.append(0)

for line in sys.stdin:

line = line.strip()

key,value = line.split('\t',1)

words = value.split();

if (lastkey != key and lastkey != None):

for i in range(0,10):

result+= valA[i] \* valB[i]

valA[i] = 0

valB[i] = 0

if result != 0:

print '%s\t%d' % (lastkey,result)

lastkey = key

result = 0

if(words[0] == 'A'):

valA[int(words[1])-1] = int(words[2])

if(words[0] == 'B'):

valB[int(words[1])-1] = int(words[2])

lastkey = key

for i in range(0,10):

result+= valA[i] \* valB[i]

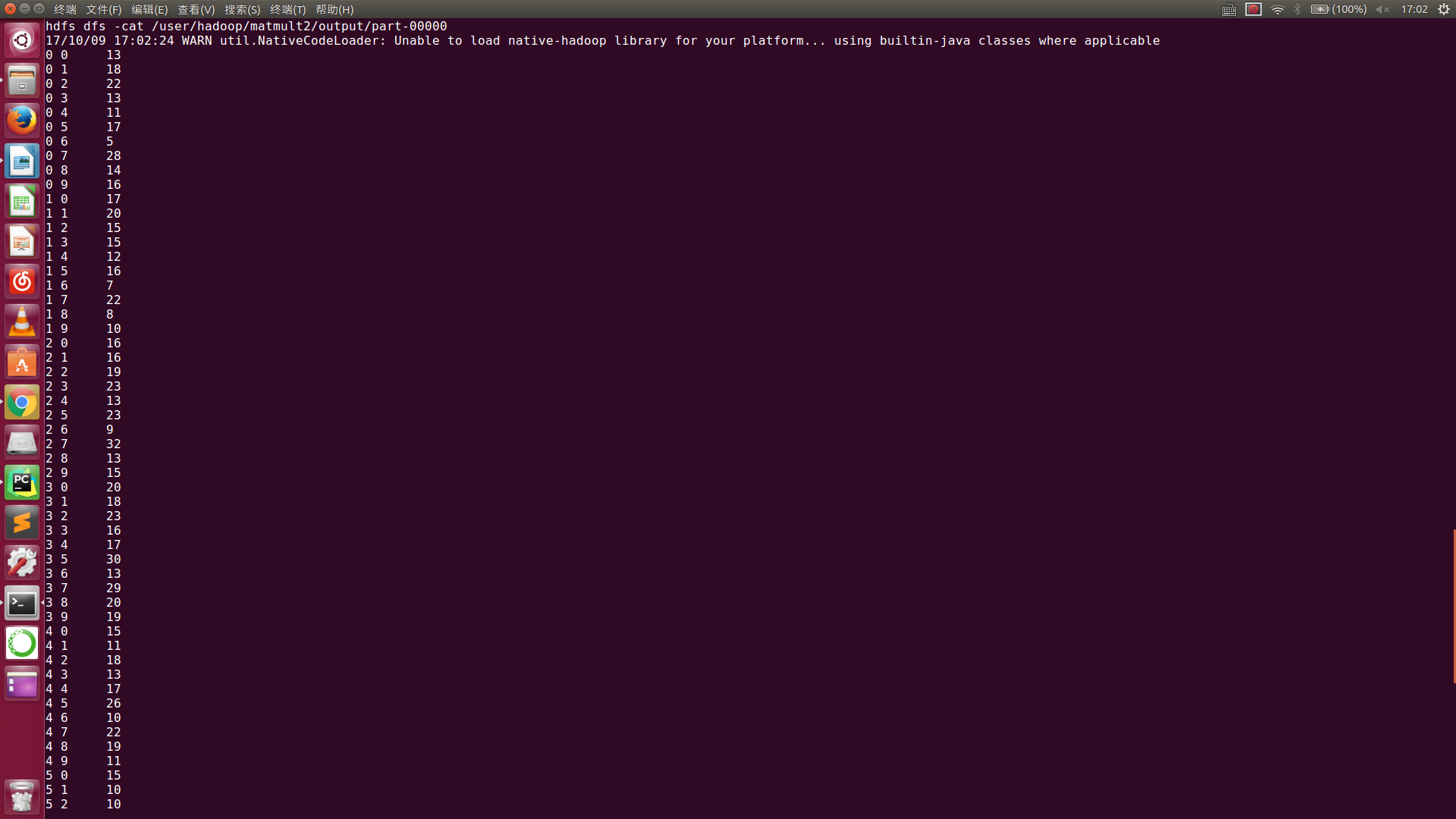
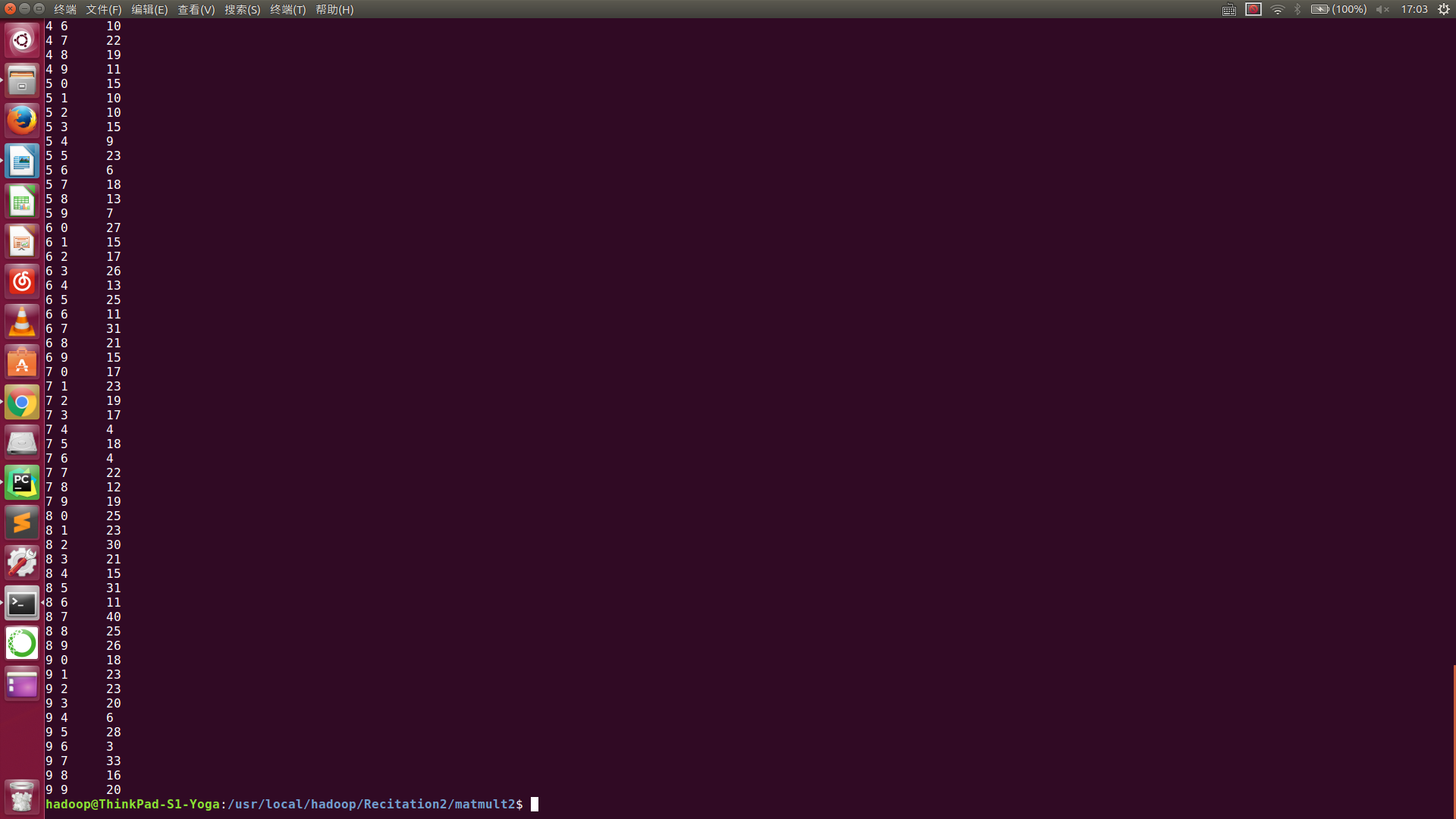
valA[i] = 0

valB[i] = 0

if result != 0:

print '%s\t%d' % (lastkey,result)

**screenshot:**

**f) Algorithm:**

in map.py, the program reads the input file and generates several key- value pairs for each line where the key is the coordinates of the result matrix and value is the coordinates that help calculate as well the value of input matrix, thanks to Hadoop sorting automatically according to keys, calculation required elements for a specific element in result matrix stay together. Then in reduce.py, the program puts the values have the same key into two arrays differ from their source matrix A or B by the order of the coordinates in values, and the product of two arrays with the same key is the result element where the key indicates.

Some key – value pairs are not really used since the other multiplier can be zero, when the matrix is big enough, lots of space resources can be wasted, a detect function can be added into the map process, to determine whether a pair is worth generate or not as they are dense matrices, a single zero in one matrix can cause lots of products of elements in the other matrix becoming zero . However this could add some complexity in algorithm so I’m afraid it only works well when the matrix is really big.