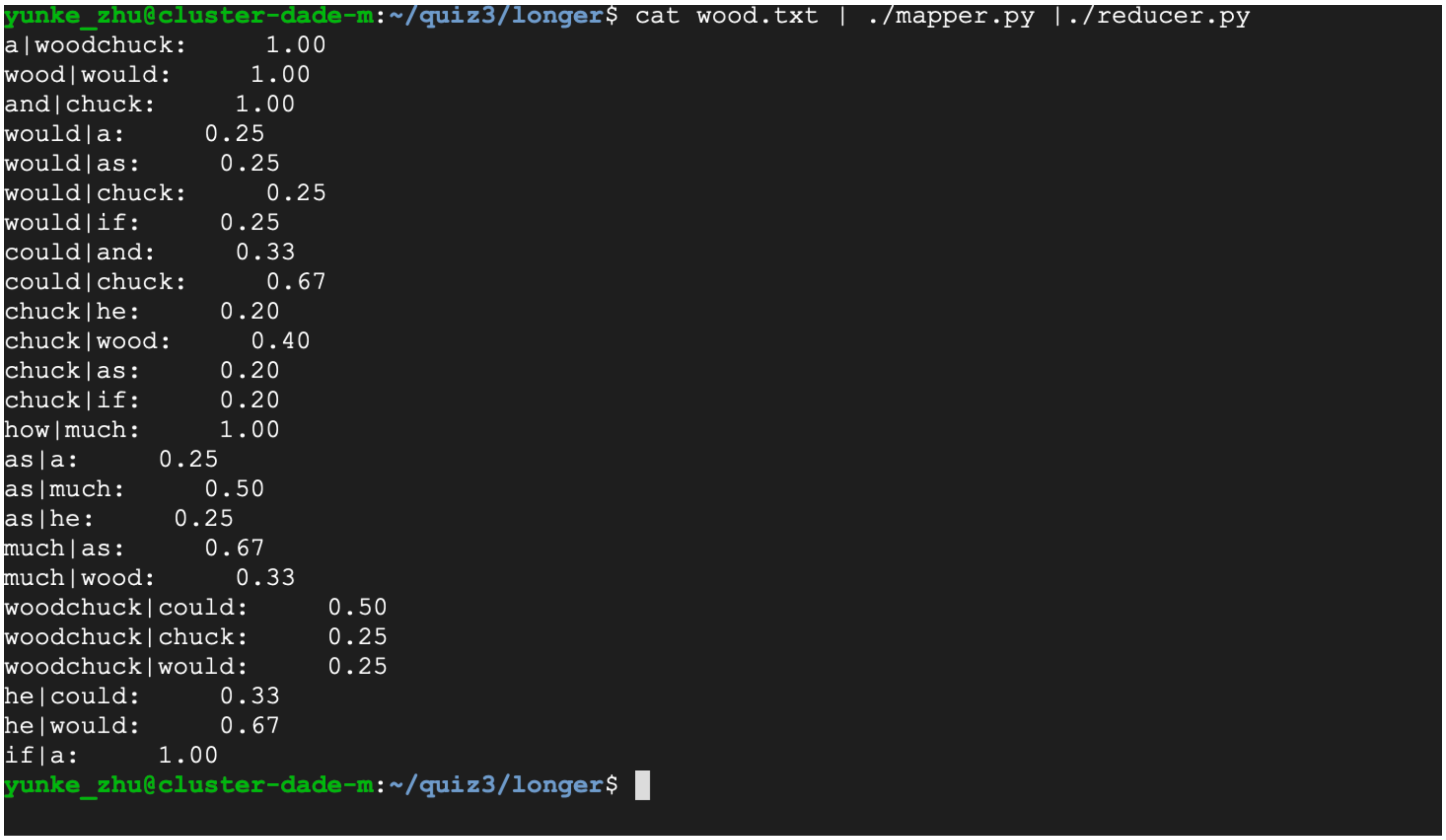
1. Result

Example with two sentences:

1. How much wood would a woodchuck chuck if a woodchuck could chuck wood?
2. He would chuck, he would, as much as he could, and chuck as much as a woodchuck would If a woodchuck could chuck wood.

The float numbers are the relative frequency



```
yunke_zhu@cluster-dade-m:~/quiz3/longer$ cat wood.txt | ./mapper.py |./reducer.py
a|woodchuck:     1.00
wood|would:     1.00
and|chuck:     1.00
would|a:     0.25
would|as:     0.25
would|chuck:     0.25
would|if:     0.25
could|and:     0.33
could|chuck:     0.67
chuck|he:     0.20
chuck|wood:     0.40
chuck|as:     0.20
chuck|if:     0.20
how|much:     1.00
as|a:     0.25
as|much:     0.50
as|he:     0.25
much|as:     0.67
much|wood:     0.33
woodchuck|could:     0.50
woodchuck|chuck:     0.25
woodchuck|would:     0.25
he|could:     0.33
he|would:     0.67
if|a:     1.00
yunke_zhu@cluster-dade-m:~/quiz3/longer$
```

2. Code

Mapper.py

```
#!/usr/bin/env python
import sys
import os
import json
import re
import subprocess
class Mapper:


    def MAP(self):

        for line in sys.stdin:
               #--- remove leading and trailing whitespace---
            line = line.strip()
                               #filepath = "123"
            words = line.split()

            for i in range(len(words)-1):
                word1 = re.sub(r'[^\w]','', words[i]).lower()
                word2 = re.sub(r'[^\w]','', words[i+1]).lower()
                print '%s' % (word1+"\t"+word2+"|" + "1")




exp = Mapper()
exp.MAP()
```

Reducer.py

```
#!/usr/bin/env python
import sys

D = dict()
Counter = dict()
for line in sys.stdin:
    line = line.strip()
    words = line.strip().split('\t')

    key = words[0]
    value = words[1].split('|')[0]
    count = words[1].split('|')[1]
    count = int(count)
    #print "key: %s\tvalue: %s\tcount: %s" % (key,value,count)

    if key in Counter:
        Counter[key] += 1
    else:
        Counter[key] = 1



    if key in D:
        #print key
        if value in D[key]:
            (D[key])[value] += 1
        else:

            D[key][value] = count
    else :
        d = {value:count}
        D[key] = d

        #print d

for x in D:
    for y in D[x]:
        t = float(Counter[x])
        c = float(D[x][y])
        print "%s|%s: %8.2f" %(x,y,c/t)
```

3.Conclusion

The basic idea of this case is to calculate the relative frequency instead of total frequency. Suppose the article is very long, then the pair neighbor won't be accurate to find what we need since there would appear more useless word account for more frequency such as:'to be', 'you are', be a'... So instead, we need to calculate the relative frequency. More detail are shown on the Chapter 3 of Lin and Dyer's book.

1. This case needs to be modified a lot in reducer. In mapper, we just use the single word as the key, and put its neightbor and count as the value to the reducer.
2. In reducer, we make a dictionary to store all the data. The key of the dictionary is each single word while the value of the dictionary is still a dictionary. The value dictionary is used to store every word's relative frequency, so the key of value-dictionary is each neighbor word and the value of value-dictionary is the count number.
3. After we stored all the data into our data structure, we are able to calculate the frequency of each word instead of the total frequency.

4. Command

```
cat filename.txt | ./mapper.py | ./reducer.py
```