CS 429 - Information Retrieval
Homework 1

Shiva Sankar Modala(A20517528)

1.1 Chapter 1
Exercises: 1.1,1.2,1.3,1.7,1.8
Exercise 1.1
Draw the inverted index that would be built for the following document collection. (See Figure 1.3 for an example.)
Doc 1 new home sales top forecasts
Doc 2 home sales rise in july
Doc 3 increase in home sales in july
Doc 4 july new home sales rise
Ans: Inverted index for given document collection.

new	—> 1 —> 4
home	—> 1 —> 2 —> 3 —> 4
sales	—> 1 —> 2 —> 3 —> 4
top	—> 1
forecasts	—> 1
rise	—> 2 —> 4
in	—> 2 —> 3
july	—> 2 —> 3 —> 4
increase	—> 3

Exercise 1.2
Consider these documents:
Doc 1 breakthrough drug for schizophrenia
Doc 2 new schizophrenia drug
Doc 3 new approach for treatment of schizophrenia
Doc 4 new hopes for schizophrenia patients
a. Draw the term-document incidence matrix for this document collection.
Ans: Term-document incidence matrix for the given document collection.

	d1	d2	d3	d4
breakthrough	1	0	0	0
drug	1	1	0	0
for	1	0	1	1
schizophrenia	1	1	1	1
new	0	1	1	1
approach	0	0	1	0
treatment	0	0	1	0
of	0	0	1	0
hopes	0	0	0	1
patients	0	0	0	1

b. Draw the inverted index representation for this collection
Ans: Inverted index representation for the given document collection.

breakthrough	—> 1
drug	—> 1 —> 2
for	—> 1 —> 3 —> 4
schizophrenia	—> 1 —> 2 —> 3 —> 4
new	—> 2 —> 3 —> 4
approach	—> 3
treatment	—> 3
of	—> 3
hopes	—> 4
patients	—> 4

Exercise 1.3
For the document collection shown in Exercise 1.2, what are the returned results for these queries:
Doc 1 breakthrough drug for schizophrenia
Doc 2 new schizophrenia drug
Doc 3 new approach for treatment of schizophrenia
Doc 4 new hopes for schizophrenia patients
a. schizophrenia AND drug
Ans: 1111 AND 1100 => 1100 => Doc 1, Doc 2

b. for AND NOT (drug OR approach)
Ans: 1011 AND NOT (1100 OR 0010) => 1011 AND NOT (1110) => 1011 AND 0001 => 0001 => Doc 4

Exercise 1.7
Recommend a query processing order for
d. (tangerine OR trees) AND (marmalade OR skies) AND (kaleidoscope OR eyes)
given the following postings list sizes:

Term	Postings size
eyes	213312
kaleidoscope	87009
marmalade	107913
skies	271658
tangerine	46653
trees	316812

Ans:
(tangerine OR trees) AND (marmalade OR skies) AND (kaleidoscope OR eyes)
Now we will calculate (tangerine OR trees) then (marmalade OR skies) and then (kaleidoscope OR eyes)
tangerine OR trees = 46653 + 316812 = 363465
marmalade OR skies = 107913 + 271658 = 379571
kaleidoscope OR eyes = 87009 + 213312 = 300321

Start with (kaleidoscope OR eyes) because it has the smallest total, then AND it with (tangerine OR trees), ….

I suggest
(kaleidoscope OR eyes) AND (tangerine OR trees) AND (marmalade OR skies)

Exercise 1.8
If the query is:
e. friends AND romans AND (NOT countrymen)
How could we use the frequency of countrymen in evaluating the best query evaluation order? In particular, propose a way of handling negation in determining the order of query processing.
ANs: We systematically select a term to supplement the final query processing by making use of frequency of the term “countrymen”. Initially, we follow the same steps as used in question 1.7 by sorting the terms in ascending order of the size of the postings. If the term “countrymen” appears more frequently, then it can be used to delete documents when it doesn't exist.

1.2 Chapter 2
Exercises: 2.1,2.6,2.7,2.9,2.10
Exercise 2.1
Are the following statements true or false?
a. In a Boolean retrieval system, stemming never lowers precision.
Ans: False, it does harm precision (less specific words lose precision)
b. In a Boolean retrieval system, stemming never lowers recall.
Ans: True, makes set of retrieved information larger
c. Stemming increases the size of the vocabulary.
Ans: False, it decreases the size of vocabulary.
d. Stemming should be invoked at indexing time but not while processing a query
Ans: False

Exercise 2.6
We have a two-word query. For one term the postings list consists of the following 16 entries:
[4,6,10,12,14,16,18,20,22,32,47,81,120,122,157,180]
and for the other it is the one entry postings list: [47].
Work out how many comparisons would be done to intersect the two postings lists with the following two strategies.
Briefly justify your answers:
a. Using standard postings lists
Ans: Standard postings lists - we have to compare 4 with 47 and so on till it intersects the two postings lists.
4 & 47, 6 & 47, 10 & 47, 12 &47, 14&47, 16&47,18&47, 20&47, 22&47, 32&47, 47 & 47
So, 11 comparisons would be done to intersect the two postings lists.

b. Using postings lists stored with skip pointers, with a skip length of √ P, as suggested in Section 2.3.
Ans: √ P = √16 = 4

comparisons: 4 & 47, 14&47, 22&47, 140&47, 81&47, 47&47
So, 6 comparisons would be done.

Exercise 2.7 [⋆]

Consider a postings intersection between this postings list, with skip pointers:

and the following intermediate result postings list (which hence has no skip pointers):

3 5 89 95 97 99 100 101

Trace through the postings intersection algorithm in Figure 2.10 (page 37).

a. How often is a skip pointer followed (i.e., p1 is advanced to skip(p1))?
Ans: a skip pointer is followed only once.

b. How many posting comparisons will be made by this algorithm while intersecting the two lists?
Ans: comparisons - (3,3), (5,5), (9,89), (15,89), (24,89), (39,89), (60,89), (68,89), (75,89), (81,89), (84,89), (89,89), (92,95), (96,97), (97,97), (100,99), (100,100), (115,101)
Total - 18 posting comparisons will be made by the algorithm

c. How many posting comparisons would be made if the postings lists are intersected without the use of skip pointers?
Ans: Total of 19 comparisons would be made if the postings lists are intersected without the use of skip pointers.

Exercise 2.9 [⋆]

Shown below is a portion of a positional index in the format: term: doc1: ⟨position1, position2, . . . ⟩; doc2: ⟨position1, position2, . . . ⟩; etc.

angels: 2: ⟨36,174,252,651⟩; 4: ⟨12,22,102,432⟩; 7: ⟨17⟩;
fools: 2: ⟨1,17,74,222⟩; 4: ⟨8,78,108,458⟩; 7: ⟨3,13,23,193⟩;
fear: 2: ⟨87,704,722,901⟩; 4: ⟨13,43,113,433⟩; 7: ⟨18,328,528⟩;
in: 2: ⟨3,37,76,444,851⟩; 4: ⟨10,20,110,470,500⟩; 7: ⟨5,15,25,195⟩;
rush: 2: ⟨2,66,194,321,702⟩; 4: ⟨9,69,149,429,569⟩; 7: ⟨4,14,404⟩;
to: 2: ⟨47,86,234,999⟩; 4: ⟨14,24,774,944⟩; 7: ⟨199,319,599,709⟩;
tread: 2: ⟨57,94,333⟩; 4: ⟨15,35,155⟩; 7: ⟨20,320⟩;
where: 2: ⟨67,124,393,1001⟩; 4: ⟨11,41,101,421,431⟩; 7: ⟨16,36,736⟩;

Which document(s) if any match each of the following queries, where each expression within quotes is a phrase query?

a. “fools rush in”

Ans:

Doc 2 (pos 1,2,3)

Doc 4 (pos 8,9,10)

Doc 7 (pos 3,4,5)

So, the answer is Doc 2, 4 and 7 are correct

b. “fools rush in” AND “angels fear to tread”

Ans:

“fools rush in”	“angels fear to tread”
Doc 2 (pos 1,2,3)	Doc 2 (pos 36,87,47,57)
Doc 4 (pos 8,9,10)	Doc 4 (pos 12,13,14,15)
Doc 7 (pos 3,4,5)	Doc 7 (pos 17,18,199,20)

“fools rush in” AND “angels fear to tread” = Doc 4

Exercise 2.10 [⋆]

Consider the following fragment of a positional index with the format:

word: document: ⟨position, position, . . .⟩; document: ⟨position, . . .⟩
. . .

Gates: 1: ⟨3⟩; 2: ⟨6⟩; 3: ⟨2,17⟩; 4: ⟨1⟩;
IBM: 4: ⟨3⟩; 7: ⟨14⟩;
Microsoft: 1: ⟨1⟩; 2: ⟨1,21⟩; 3: ⟨3⟩; 5: ⟨16,22,51⟩;

a. Describe the set of documents that satisfy the query Gates /2 Microsoft.

Ans: Doc 1 and 3

b. Describe each set of values for k for which the query Gates /k Microsoft returns a different set of documents as the answer.
Ans:
If k = 1 then Doc 3
If k = 2 to 4 then Doc 1 and 3
If k >= 5 then Doc 1, 2 and 3

2. Practicum Problems
(I've attached my files in PDF format.)

In [1]:
```python
#2.1 Problem 1
```

In [2]:
```python
#importing the necessary packages to complete the assignment

import pandas as pd
import string
import numpy as np
import nltk
nltk.download('punkt')
from sklearn.datasets import fetch_20newsgroups
from nltk.tokenize import word_tokenize
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\shiva\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [3]:
```python
# Fetching the 20newsgroups.
# I tried to remove headers and footers since they are not related to doc text
newsgroups = fetch_20newsgroups(subset='all', shuffle = False, remove=('headers', 'footers'))
```

In [4]:
```python
# I'm trying to inspect the data that I fetched from 20newsgroups

docsList = newsgroups.data
print(docsList[7])
```

```
In a prior article ajaffe@oddjob.uchicago.edu (Andrew Jaffe) wrote:
  > Hi.
  >
  > I use Emacs and I want to customize my keyboard better.
  > When I set up stuff in my .emacs with a keymap and define-keys,
  > I can only access certain of the keys on my X-Terminal's
  > keyboard. I can't get e.g. F10, Home, End, PgUp, PgDn; they all
  > seem to have either the same or no keycode. I have a feeling
  > this can't be fixed in emacs itself, but that I need to do some
  > xmodmap stuff. Can someone help me?

Unfortunately, the key event handling is pretty much hardwired into
emacs.  I ran into this problem a while back; my solution was to
change the source code so that all of these special keys generated
character sequences which basically encoded the keysym and modifier
state as escape sequences -- for example, the sequence "ESC [ 1 B 7"
would indicate that the "HOME" key was pressed, with the shift key
down.  You could also detect standard keys with odd modifiers, such
as "shift-Return".

If anybody wants these changes, they're welcome to them, but you'll
have to have the source available and be comfortable munging with
it a bit.  Basically you have to replace the keypress handling code
in the source file "x11term.c".  Maybe if someone at OSF is
interested, I can send them the tweaks, but I imagine they've got
bigger fish to fry (hopefully including the much talked about
emacs V19!).  If there's sufficient interest, I'll post the mods
somewhere, although this probably isn't the appropriate group for it.

Notes:

 * This special code will only apply if you let emacs create
   its own (X11) window.  If you run it in plain old tty mode (which
   includes Xterm windows) then it's business as usual.

 * The patches I made were to version 18.58, under Sun OS 4.1.2
   [I also did this a while back under HP-UX].  The patches are
   in a chunk of code between '#if sun ... #endif' but could
   easily be adapted for anything else.
```

In [5]:
```python
#defined tokenizeDocument function to tokenize each document into list of words
#I removed punctuations and special characters

def tokenizeDocument(docString: str):
  finString = (docString.lower()).translate(str.maketrans('', '', string.punctuation))
  finalTokenizeDoc = word_tokenize(finString)
  return finalTokenizeDoc

tokenizedList = []

for item in docsList:
  tokenizedList.append(tokenizeDocument(item))
```

In [6]:
```python
# I'm trying to check if the function works right

len(tokenizedList)
print(tokenizedList[5])
```

```
['help', 'i', 'have', 'an', 'adb', 'graphicsd', 'tablet', 'which', 'i', 'want', 'to', 'connect', 'to', 'my', 'quadra', '950', 'unfortunately', 'the', '950', 'has', 'only', 'one',
'adb', 'port', 'and', 'it', 'seems', 'i', 'would', 'have', 'to', 'give', 'up', 'my', 'mouse', 'please', 'can', 'someone', 'help', 'me', 'i', 'want', 'to', 'use', 'the', 'tablet',
'as', 'well', 'as', 'the', 'mouse', 'and', 'the', 'keyboard', 'of', 'course', 'thanks', 'in', 'advance']
```

In [7]:
```python
#yeah the function works perfectly
```

In [8]:
```python
#2.2 problem 2
```

In [9]:
```python
# defined inverted index function to create inverted index of the tokens

def invertedIndex(tokenizedList: list):
  inverted = {}
  i = -1
  for doc in tokenizedList:
    i += 1
    for word in list(set(doc)):
      if word in inverted:
        newList = inverted[word]
        newList.append(str(i))
        inverted[word] = newList
      else:
        inverted[word] = [str(i)]
  return inverted

newInd = invertedIndex(tokenizedList)

sortedIndex = {key: value for key, value in sorted(newInd.items())}
```

In [10]:
```python
# make sure it works, choose random word make sure its in the docs it claims to be, not in others

print(sortedIndex["friend"][5])
print('friend' in tokenizedList[168])
print('friend' in tokenizedList[169])
```

```
168
True
False
```

In [11]:
```python
#2.3 Problem 3
```

In [12]:
```python
def uniqueIdentifiers(word1: str, word2: str):

  if (word1 or word2) not in newInd:
    print("Entered word is not available in the dictionary")
    return []

  #print(word1)
  #print(word2)

  intersection = []
  list1 = []
  list2 = []

  for posting in newInd[word1]:
    list1.append(posting)
  for posting in newInd[word2]:
    list2.append(posting)

  #print("postings in list1" + str(list1))
  #print("postings in list2" + str(list2))


  while (len(list1) != 0) and (len(list2) != 0):
      #print("non empty list")

      if int(list1[0]) == int(list2[0]):
        #print("adding posting to intersection")
        intersection.append(list1[0])
        list1.pop(0)
        list2.pop(0)

      elif int(list1[0]) < int(list2[0]):
        #print(list1[0] +"<"+ list2[0])
        list1.pop(0)

      else:
        #print(list1[0] +">"+ list2[0])
        list2.pop(0)

  return intersection

print("the intersection is: ")
(uniqueIdentifiers("money", "red"))
```

```
the intersection is:
```

Out[12]:
```
['143',
 '235',
 '333',
 '348',
 '2841',
 '2928',
 '3412',
 '3538',
 '3548',
 '4197',
 '4749',
 '4855',
 '5555',
 '5791',
 '6495',
 '6905',
 '6967',
 '7132',
 '7137',
 '7592',
 '7833',
 '8097',
 '8268',
 '8893',
 '8947',
 '10023',
 '10272',
 '11114',
 '11540',
 '14511',
 '14521',
 '15240',
 '16514',
 '16739',
 '17329',
 '17866',
 '18204',
 '18678']
```

In [13]:
```python
#using the above function, we found out that the common between money and red are 143, 235, 333, ......, 18678.
#lets verify it
common = set(newInd["money"]) - (set(newInd["money"]) - set(newInd["red"]))
common = [int(x) for x in common]
common.sort()
common = [str(x) for x in common]
print(common)

#It matches
```

```
['143', '235', '333', '348', '2841', '2928', '3412', '3538', '3548', '4197', '4749', '4855', '5555', '5791', '6495', '6905', '6967', '7132', '7137', '7592', '7833', '8097', '8268',
'8893', '8947', '10023', '10272', '11114', '11540', '14511', '14521', '15240', '16514', '16739', '17329', '17866', '18204', '18678']
```

In [14]:
```python
 #Lets check for other pair of words

print((uniqueIdentifiers("wine", "blood")))

common = list(set(newInd["wine"]) - (set(newInd["wine"]) - set(newInd["blood"])))
common = [int(x) for x in common]
common.sort()
common = [str(x) for x in common]

print(common)

#its working
```

```
['4463', '4754', '5699', '7623', '10760', '12022']
['4463', '4754', '5699', '7623', '10760', '12022']
```

In [15]:
```python
print(len(newInd["rain"]))
print(len(newInd["coat"]))
print(len(newInd["good"]))
print(len(newInd["blue"]))
print(len(newInd["worry"]))
```

```
57
33
3295
281
203
```

In [16]:
```python
print("we know that the order should be coat, rain, worry, blue, good")
```

```
we know that the order should be coat, rain, worry, blue, good
```

In [17]:
```python
# defined returnSorted function which performs a sort of given list of words and a dict index
# It returns words as a sorted list based on number of postings

def returnSorted(wordList: list):
  lengthList = []
  for word in wordList:
    lengthList.append(len(newInd[word]))
  finalList = [wordList for lengthList, wordList in sorted(zip(lengthList, wordList))]

  return finalList


print("returned list:")
print(returnSorted(['rain', 'coat', 'good', 'blue', 'worry']))
```

```
returned list:
['coat', 'rain', 'worry', 'blue', 'good']
```

In [18]:
```python
#2.4 Problem 4
```

In [19]:
```python
# I'm trying to create a version that actually just takes in the lists themseleves rather than the words

def uniqueIdentifiers2(list1: list, list2: list):

  intersection = []

  while (len(list1)!=0) and (len(list2)!=0):
    if int(list1[0]) == int(list2[0]):
      intersection.append(list1[0])
      list1.pop(0)
      list2.pop(0)
    elif int(list1[0]) < int(list2[0]):
      list1.pop(0)
    else:
      list2.pop(0)
  return intersection

# Trying to check for some values as tests similar to earlier problem

list1 = []
list2 = []

for posting in newInd["money"]:
  list1.append(posting)

for posting in newInd["red"]:
  list2.append(posting)


print("the intersection is: ")
(uniqueIdentifiers2(list1, list2))
```

```
the intersection is:
```

Out[19]:
```
['143',
 '235',
 '333',
 '348',
 '2841',
 '2928',
 '3412',
 '3538',
 '3548',
 '4197',
 '4749',
 '4855',
 '5555',
 '5791',
 '6495',
 '6905',
 '6967',
 '7132',
 '7137',
 '7592',
 '7833',
 '8097',
 '8268',
 '8893',
 '8947',
 '10023',
 '10272',
 '11114',
 '11540',
 '14511',
 '14521',
 '15240',
 '16514',
 '16739',
 '17329',
 '17866',
 '18204',
 '18678']
```

In [20]:
```python
# defined a function that helps to get posting list into a list

def getPostings(word: str):
  list1 = []
  for posting in newInd[word]:
    list1.append(posting)
  return list1

print(getPostings("worm"))
```

```
['591', '1428', '3295', '3350', '4754', '7585', '8384', '9522', '9687', '10046', '11651', '12734', '13206']
```

In [21]:
```python
#defined the queryFunction

def queryFunction(query: str):
  # Replacing '&' with spaces
  query = query.replace("&", " ")

  # tokenize query
  tokenizedQuery = word_tokenize(query)
  print("tokenized query: " + str(tokenizedQuery))

  # if not in dict, []
  for word in tokenizedQuery:
    if word not in newInd:
      print("Entered word is not available in the dictionary")
      return []

  # inverted index to sort
  sorted = returnSorted(tokenizedQuery)
  print("sorted by postings: " + str(sorted) + "\n")

  # interect in order
  i = 0

  list1 = []
  list2 = []
  finalList = []

  currList = getPostings(sorted[0])
  print("current list is from: " +  str(sorted[0]))
  print(str(currList) + '\n')

  for i in range(1, len(sorted)):
    print("merge current list with the word: " + sorted[i])
    currList = (uniqueIdentifiers2(currList, getPostings(sorted[i])))
    print(str(currList) + '\n')


  print("the final intersection is: ")
  #print(currList)
  return currList

# sample query
print(queryFunction("bank&rob&dead"))
```

```
tokenized query: ['bank', 'rob', 'dead']
sorted by postings: ['bank', 'rob', 'dead']

current list is from: bank
['150', '353', '447', '476', '711', '830', '831', '939', '1095', '1098', '1132', '1146', '1153', '1237', '1246', '1567', '1581', '1728', '1919', '1936', '2057', '2088', '2487', '25
90', '2784', '2806', '2892', '3053', '3214', '3420', '3634', '3649', '3677', '3757', '3873', '3928', '4087', '4175', '4241', '4489', '4503', '4545', '4573', '4615', '4682', '4728',
'4731', '4738', '4747', '4877', '4989', '5063', '5109', '5391', '5495', '5527', '5655', '5992', '6175', '6642', '6859', '7524', '7560', '7650', '7876', '7901', '8044', '8079', '842
2', '8466', '8508', '8594', '8605', '8630', '8976', '9042', '9080', '9186', '9561', '9825', '9872', '10047', '10329', '10347', '10549', '10590', '10627', '10630', '10754', '10788',
'10965', '11218', '11368', '11381', '11397', '11467', '11745', '11834', '11996', '12408', '12458', '12486', '12662', '12735', '13002', '13131', '13489', '13546', '13749', '13913',
'13945', '14141', '14387', '14420', '14473', '14489', '14505', '14546', '15231', '15381', '15606', '15687', '15788', '15820', '15861', '15957', '16338', '16450', '16458', '16644',
'16700', '16869', '16971', '17094', '17165', '17334', '17563', '17770', '18204', '18384', '18794']

merge current list with the word: rob
['5655', '8466', '11996', '12735', '14505']

merge current list with the word: dead
['14505']

the final intersection is:
['14505']
```

In [22]:
```python
# more sample queries
print(queryFunction("win"))
```

```
tokenized query: ['win']
sorted by postings: ['win']

current list is from: win
['53', '57', '91', '145', '146', '154', '155', '165', '173', '177', '206', '210', '221', '248', '305', '339', '408', '472', '546', '557', '578', '595', '611', '615', '673', '702',
'731', '764', '803', '904', '967', '970', '1009', '1036', '1075', '1123', '1129', '1167', '1206', '1217', '1226', '1253', '1293', '1397', '1409', '1417', '1469', '1522', '1560', '1
575', '1604', '1613', '1615', '1639', '1676', '1686', '1692', '1701', '1706', '1743', '1785', '1789', '1792', '1814', '1839', '1855', '1975', '2015', '2023', '2026', '2049', '205
2', '2078', '2089', '2138', '2173', '2180', '2249', '2254', '2303', '2369', '2427', '2431', '2485', '2493', '2518', '2525', '2548', '2564', '2603', '2648', '2662', '2709', '2728',
'2738', '2787', '2842', '2852', '2928', '2949', '2972', '2995', '3002', '3014', '3017', '3030', '3083', '3087', '3088', '3090', '3136', '3177', '3254', '3292', '3421', '3423', '343
2', '3434', '3435', '3457', '3554', '3561', '3568', '3665', '3720', '3735', '3744', '3765', '3772', '3774', '3778', '3815', '3823', '3848', '3852', '3883', '3901', '3905', '4026',
'4038', '4054', '4057', '4094', '4106', '4113', '4139', '4197', '4303', '4309', '4357', '4365', '4497', '4514', '4564', '4665', '4692', '4730', '4762', '4822', '4855', '4862', '486
5', '4926', '4940', '4947', '4952', '4983', '5022', '5059', '5151', '5187', '5224', '5249', '5280', '5335', '5344', '5412', '5429', '5439', '5499', '5505', '5508', '5513', '5600',
'5616', '5658', '5674', '5679', '5690', '5717', '5760', '5766', '5809', '5811', '5842', '5857', '5881', '5913', '5946', '5983', '5997', '6070', '6089', '6103', '6215', '6216', '632
1', '6504', '6605', '6610', '6676', '6678', '6684', '6722', '6739', '6741', '6803', '6815', '6895', '6967', '7028', '7059', '7109', '7129', '7132', '7150', '7177', '7220', '7240',
'7260', '7261', '7339', '7346', '7395', '7439', '7452', '7457', '7563', '7598', '7606', '7708', '7714', '7715', '7722', '7731', '7789', '7833', '7864', '7924', '7977', '8001', '804
7', '8050', '8088', '8097', '8153', '8249', '8359', '8364', '8523', '8531', '8555', '8606', '8643', '8656', '8683', '8796', '8826', '8888', '8937', '8949', '8967', '9033', '9037',
'9079', '9093', '9095', '9126', '9243', '9261', '9273', '9274', '9317', '9346', '9421', '9426', '9479', '9514', '9604', '9670', '9679', '9735', '9759', '9774', '9795', '9813', '991
8', '9998', '10008', '10020', '10041', '10046', '10055', '10087', '10105', '10191', '10221', '10315', '10345', '10372', '10390', '10392', '10417', '10447', '10735', '10744', '1084
9', '10850', '10881', '10916', '10940', '10954', '10971', '10980', '10981', '11018', '11022', '11036', '11039', '11059', '11066', '11074', '11114', '11124', '11170', '11177', '1118
3', '11293', '11433', '11442', '11456', '11472', '11537', '11593', '11613', '11625', '11628', '11686', '11734', '11803', '11839', '11841', '11849', '11863', '11864', '11906', '1194
4', '11964', '12042', '12044', '12080', '12088', '12177', '12218', '12309', '12337', '12369', '12405', '12447', '12458', '12460', '12486', '12514', '12574', '12610', '12638', '1269
6', '12716', '12728', '12811', '12826', '12840', '12842', '12899', '12933', '12952', '13053', '13071', '13084', '13099', '13177', '13213', '13232', '13265', '13321', '13336', '1336
3', '13386', '13411', '13450', '13488', '13498', '13526', '13544', '13619', '13692', '13751', '13793', '13806', '13847', '13861', '13863', '13877', '13917', '13958', '13962', '1397
3', '14019', '14167', '14174', '14259', '14278', '14297', '14330', '14372', '14409', '14469', '14577', '14632', '14780', '14795', '14820', '14824', '14835', '15006', '15013', '1514
2', '15212', '15290', '15324', '15340', '15396', '15414', '15426', '15443', '15515', '15645', '15805', '15828', '15871', '15879', '16025', '16026', '16132', '16168', '16171', '1617
3', '16380', '16437', '16507', '16569', '16608', '16641', '16666', '16691', '16765', '16810', '16813', '16875', '16924', '16943', '17108', '17161', '17171', '17204', '17220', '1725
7', '17318', '17350', '17360', '17393', '17433', '17455', '17465', '17683', '17704', '17794', '17829', '17833', '17866', '17942', '18008', '18015', '18016', '18046', '18054', '1812
7', '18133', '18296', '18318', '18329', '18389', '18493', '18517', '18565', '18692', '18733', '18768', '18796']

the final intersection is:
['53', '57', '91', '145', '146', '154', '155', '165', '173', '177', '206', '210', '221', '248', '305', '339', '408', '472', '546', '557', '578', '595', '611', '615', '673', '702',
'731', '764', '803', '904', '967', '970', '1009', '1036', '1075', '1123', '1129', '1167', '1206', '1217', '1226', '1253', '1293', '1397', '1409', '1417', '1469', '1522', '1560', '1
575', '1604', '1613', '1615', '1639', '1676', '1686', '1692', '1701', '1706', '1743', '1785', '1789', '1792', '1814', '1839', '1855', '1975', '2015', '2023', '2026', '2049', '205
2', '2078', '2089', '2138', '2173', '2180', '2249', '2254', '2303', '2369', '2427', '2431', '2485', '2493', '2518', '2525', '2548', '2564', '2603', '2648', '2662', '2709', '2728',
'2738', '2787', '2842', '2852', '2928', '2949', '2972', '2995', '3002', '3014', '3017', '3030', '3083', '3087', '3088', '3090', '3136', '3177', '3254', '3292', '3421', '3423', '343
2', '3434', '3435', '3457', '3554', '3561', '3568', '3665', '3720', '3735', '3744', '3765', '3772', '3774', '3778', '3815', '3823', '3848', '3852', '3883', '3901', '3905', '4026',
'4038', '4054', '4057', '4094', '4106', '4113', '4139', '4197', '4303', '4309', '4357', '4365', '4497', '4514', '4564', '4665', '4692', '4730', '4762', '4822', '4855', '4862', '486
5', '4926', '4940', '4947', '4952', '4983', '5022', '5059', '5151', '5187', '5224', '5249', '5280', '5335', '5344', '5412', '5429', '5439', '5499', '5505', '5508', '5513', '5600',
'5616', '5658', '5674', '5679', '5690', '5717', '5760', '5766', '5809', '5811', '5842', '5857', '5881', '5913', '5946', '5983', '5997', '6070', '6089', '6103', '6215', '6216', '632
1', '6504', '6605', '6610', '6676', '6678', '6684', '6722', '6739', '6741', '6803', '6815', '6895', '6967', '7028', '7059', '7109', '7129', '7132', '7150', '7177', '7220', '7240',
'7260', '7261', '7339', '7346', '7395', '7439', '7452', '7457', '7563', '7598', '7606', '7708', '7714', '7715', '7722', '7731', '7789', '7833', '7864', '7924', '7977', '8001', '804
7', '8050', '8088', '8097', '8153', '8249', '8359', '8364', '8523', '8531', '8555', '8606', '8643', '8656', '8683', '8796', '8826', '8888', '8937', '8949', '8967', '9033', '9037',
'9079', '9093', '9095', '9126', '9243', '9261', '9273', '9274', '9317', '9346', '9421', '9426', '9479', '9514', '9604', '9670', '9679', '9735', '9759', '9774', '9795', '9813', '991
8', '9998', '10008', '10020', '10041', '10046', '10055', '10087', '10105', '10191', '10221', '10315', '10345', '10372', '10390', '10392', '10417', '10447', '10735', '10744', '1084
9', '10850', '10881', '10916', '10940', '10954', '10971', '10980', '10981', '11018', '11022', '11036', '11039', '11059', '11066', '11074', '11114', '11124', '11170', '11177', '1118
3', '11293', '11433', '11442', '11456', '11472', '11537', '11593', '11613', '11625', '11628', '11686', '11734', '11803', '11839', '11841', '11849', '11863', '11864', '11906', '1194
4', '11964', '12042', '12044', '12080', '12088', '12177', '12218', '12309', '12337', '12369', '12405', '12447', '12458', '12460', '12486', '12514', '12574', '12610', '12638', '1269
6', '12716', '12728', '12811', '12826', '12840', '12842', '12899', '12933', '12952', '13053', '13071', '13084', '13099', '13177', '13213', '13232', '13265', '13321', '13336', '1336
3', '13386', '13411', '13450', '13488', '13498', '13526', '13544', '13619', '13692', '13751', '13793', '13806', '13847', '13861', '13863', '13877', '13917', '13958', '13962', '1397
3', '14019', '14167', '14174', '14259', '14278', '14297', '14330', '14372', '14409', '14469', '14577', '14632', '14780', '14795', '14820', '14824', '14835', '15006', '15013', '1514
2', '15212', '15290', '15324', '15340', '15396', '15414', '15426', '15443', '15515', '15645', '15805', '15828', '15871', '15879', '16025', '16026', '16132', '16168', '16171', '1617
3', '16380', '16437', '16507', '16569', '16608', '16641', '16666', '16691', '16765', '16810', '16813', '16875', '16924', '16943', '17108', '17161', '17171', '17204', '17220', '1725
7', '17318', '17350', '17360', '17393', '17433', '17455', '17465', '17683', '17704', '17794', '17829', '17833', '17866', '17942', '18008', '18015', '18016', '18046', '18054', '1812
7', '18133', '18296', '18318', '18329', '18389', '18493', '18517', '18565', '18692', '18733', '18768', '18796']
```

In [23]:
```python
# more sample queries
print(queryFunction("god&birth"))
```

```
tokenized query: ['god', 'birth']
sorted by postings: ['birth', 'god']

current list is from: birth
['268', '424', '720', '875', '1049', '1059', '1261', '1732', '1899', '2051', '2190', '2240', '2270', '2506', '2616', '2813', '2821', '2955', '2988', '3071', '3133', '3418', '3696',
'4130', '4186', '4396', '4676', '4754', '4990', '5507', '5553', '5648', '5704', '5710', '5954', '6253', '6924', '7002', '7138', '7442', '7787', '8085', '8245', '9125', '9259', '940
4', '9459', '9601', '9687', '9707', '9969', '11067', '11085', '11246', '11508', '11630', '11654', '11867', '11967', '12025', '12109', '12271', '12272', '12738', '12997', '13055',
'13101', '13134', '13157', '13223', '13838', '13880', '13923', '14087', '14218', '14279', '14646', '14927', '15042', '15176', '15229', '15725', '15882', '15892', '16506', '16737',
'17033', '17156', '17177', '17224', '17293', '17499', '17516', '17522', '17666', '17815', '18355', '18770']

merge current list with the word: god
['424', '720', '875', '1049', '1261', '2051', '2190', '2616', '4754', '4990', '5648', '7138', '9969', '11067', '11508', '11630', '12025', '12997', '13055', '13101', '13880', '1392
3', '14087', '14218', '14279', '15725', '15882', '16506', '17156', '17499', '17522', '17666']

the final intersection is:
['424', '720', '875', '1049', '1261', '2051', '2190', '2616', '4754', '4990', '5648', '7138', '9969', '11067', '11508', '11630', '12025', '12997', '13055', '13101', '13880', '1392
3', '14087', '14218', '14279', '15725', '15882', '16506', '17156', '17499', '17522', '17666']
```

In [24]:
```python
# more sample queries
print(queryFunction("I woke up in the middle of the night"))
```

```
tokenized query: ['I', 'woke', 'up', 'in', 'the', 'middle', 'of', 'the', 'night']
Entered word is not available in the dictionary
[]
```

In [25]:
```python
# sample queries
print(queryFunction("final&sample&to end my homework"))
```

```
tokenized query: ['final', 'sample', 'to', 'end', 'my', 'homework']
sorted by postings: ['homework', 'sample', 'final', 'end', 'my', 'to']

current list is from: homework
['1732', '3281', '4874', '5608', '6554', '6637', '6718', '9891', '10186', '10716', '10749', '11575', '11807', '13152', '14114', '15816', '17014']

merge current list with the word: sample
['11807']

merge current list with the word: final
[]

merge current list with the word: end
[]

merge current list with the word: my
[]

merge current list with the word: to
[]

the final intersection is:
[]
```

In [ ]: