Information Retrieval Spring 2019
Exercise Session Week 6
Exercise 3: Tolerant Retrieval

BONUS TIME
Exercise 3: Tolerant Retrieval
Moodle questions: 1.1

Edit (Levenshtein) Distance
Complete the following table on a notepad to compute the edit distance between the words and and can (refer to the algorithm in Figure 3.5, page 54 of the text):

<table>
<thead>
<tr>
<th></th>
<th>and</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

1. What are the three missing number in the last row?
- 1, 2, 3
- 1, 1, 2
- 2, 1, 2
Exercise 3: Tolerant Retrieval
Moodle questions: 1.2

2. What does each of these terms represent, in terms of character edits (add, remove, replace)?
   a) \( m[i, i+1] + 1 \) is the cost of removing \( ___[i+1] \) from \( ___[0 \ldots i+1] \), then editing \( ___[0 \ldots i] \) into \( ___[0 \ldots j+1] \).
      - \( s_1, s_2, s_1, s_2 \)
      - \( s_1, s_1, s_1, s_2 \)
      - \( s_2, s_1, s_2, s_2 \)

   b) \( m[i+1, j] + 1 \) is the cost of editing \( ___[0 \ldots i+1] \) into \( ___[0 \ldots j] \), then appending \( ___[j+1] \).
      - \( s_1, s_2, s_1 \)
      - \( s_1, s_2, s_2 \)
      - \( s_1, s_1, s_2 \)

   c) \( m[i, j] \) is the cost of editing \( ___[0 \ldots i] \) into \( ___[0 \ldots j] \). The +1 is the cost of replacing \( ___[i+1] \) with \( ___[j+1] \), if they are not already the same.
      - \( s_1, s_2, s_1, s_2 \)
      - \( s_2, s_2, s_1, s_1 \)
      - \( s_2, s_1, s_1, s_2 \)
Exercise 3: Tolerant Retrieval
Moodle questions: 1.3

3. What is the complexity of this algorithm, as a function of the lengths of the input strings (|s_1| and |s_2|)?

- $O(|s_1| \times |s_2|)$
- $O(|s_1| + |s_2|)$
- $O(|s_1|)$
Exercise 3: Tolerant Retrieval
Moodle questions: 2.1

Jaccard Coefficient (Part 1)
Calculating the edit distance between strings is expensive. A useful heuristic to estimate which strings are likely to have a small edit distance is the number of bigrams they share.
In this example, consider the symbol $ at the start and end of every word when computing bigrams.
1. a) How many bigrams does the misspelt word bord share with possible corrections board?
   - ○3
   - ○2
   - ○4

1. b) How many bigrams does the misspelt word bord share with possible corrections boarded?
   - ○5
   - ○4
   - ○3
Exercise 3: Tolerant Retrieval
Moodle questions: 2.2, 2.3, 2.4

2. Which is the more likely correction for the misspelt word bord, in the sense that it is more likely to be what the user intended to type?
- bord
- boarded

Counting the number of shared elements in two sets will be biased towards larger sets, as the larger the set, the more likely it is to contain a given element, simply by chance. The Jaccard coefficient corrects for this by normalising with respect to the size of the sets: \( J(A,B) = \frac{|A \cap B|}{|A \cup B|} \)

Calculate the Jaccard coefficients for the word pairs in the above example.

3. The Jaccard coefficient for J(bord, board) is ...
- \( \frac{4}{7} \)
- \( \frac{2}{7} \)
- \( \frac{3}{5} \)

4. The Jaccard coefficient for J(bord, boarded) is ...
- \( \frac{2}{5} \)
- \( \frac{4}{9} \)
- \( \frac{3}{7} \)
Exercise 3: Tolerant Retrieval  
Moodle questions: 3

Jaccard Coefficient (Part 2)

Consider $A$ to be the set of bigrams in $s_1$ and $B$ the set of bigrams in $s_2$. When comparing a misspelt word $s_1$ to some candidate replacement $s_2$, found using an index, we may find that although we have all bigrams of $s_1$ (as we have the original string), we don't have those of $s_2$, but rather some encoded (e.g. stemmed) form. It is nevertheless possible to calculate the Jaccard coefficient, as long as we know the number of bigrams in $s_2$.

We know how many bigrams of $s_1$ matched in $s_2$, as we used a bigram index. Thus we know $|A| \leq |A \cap B|$. We also have the original word $s_1$, from which we can calculate

$|A| \leq |A \cup B|$. From elementary set theory, we know that $|A \cup B| = |A| + |B| - |A \cap B|$, and thus if we know $|A|$, $|A \cup B|$, and $|A \cap B|$, i.e. the number of bigrams in $s_2$, we can still calculate $J(A,B)$. 

bigrams, $|A|$, $|A \cup B|$, $|A \cap B|$, $|B|$
Exercise 3: Tolerant Retrieval
Moodle questions: 4

Coding Exercise

In the part about wildcard queries in the coding exercise you implemented the function kgram_wildcard_query.
If you run kgram_wildcard_query('rend**') how many matches are returned?

- 8
- 6
- 12

In the spell-checking part of the coding exercise, if you increase the Jaccard threshold to 0.4, which of the seven tokens in the first assertion test still passes the test?

- lifetime
- betime
- sometime
Exercise 3: Tolerant Retrieval Programming

- Creating k-gram index:

```
for word in the_index.keys():
    ### <assignment>
    for gram in window("$" + word + "$", K):
        if gram != "$$":
            kgram_index.setdefault(gram, set()).add(word)
    ### </assignment>
```

Add $ to show start and ending of word
Use window function to iterate over all k-grams

This time we are using sets.
Thus all 3 cases for indexing can be covered by this
Exercise 3: Tolerant Retrieval Programming

- Parse wildcard query and return list of k-grams

```python
def wildcard_parse(q):
    if q[0] != '*':
        q = '$' + q
    if q[-1] != '*':
        q = q + '$'
    grams = [t for t in window(q, K) if '*' not in t]
    return grams
```

If query does not start with *, add $

Use window function to get all k-grams.
Ignore the ones containing $
Add the rest to grams
Exercise 3: Tolerant Retrieval Programming

- Query k-grams for matching words

```python
def kgram_wildcard_query(q):
    grams = kgram_parse(q)
    if all(gram not in kgram_index for gram in grams):
        print("NOTE: no wildcard matches for " + q) 
        return set()
    words = [kgram_index[gram] for gram in grams]
    freqs = sorted([ (p, len(p)) for p in words ], key=lambda x: x[1])
    result = freqs[0][0]
    del freqs[0]
    while freqs != []:
        result &= freqs[0][0]
        del freqs[0]
    post_filter = re.compile("^" + q.replace("*", r"\w*") + "$")
    res = {r for r in result if post_filter.match(r) is not None}
    print("NOTE: wildcard matches for " + q + " % (q, res))
    return res
```

- Get all k-grams using before implemented `kgram_parse`
- Check for all grams, if they are in `kgram_index`. If not, return empty set
- Build a list consisting of a set of all the words for each gram
- Sort the list
- Intersect the sets of words of all grams
- Post_filter to eliminate false positives
Exercise 3: Tolerant Retrieval Programming

- Spelling correction

```python
def spellcorrect(word):
    word_grams = set(window(word, 3))
    matches = [kgram_index[x] for x in word_grams]
    word_match_count = len(matches)
    all_candidates = set().union(*matches)
    outs = []
    for candidate in all_candidates:
        candidate_grams = set(window(candidate, 3))
        candidate_match_count = len(candidate_grams)
        intersection_count = len(candidate_grams & word_grams)
        union_count = word_match_count + candidate_match_count - intersection_count
        if intersection_count / union_count > Jaccard_threshold:
            outs.append((candidate, intersection_count / union_count))
    outs = sorted(outs, key=lambda x: x[1], reverse=True)
    return [x[0] for x in outs]
```

Get all k-grams of word
Build a list consisting of a set of all the words for each gram
Calculate Jaccard coefficient for each candidate word, keep one over threshold

Note: We did NOT include the ‘$’ in this exercise
Lecture second last week: Tolerant Retrieval
B+-trees: Visualization

- Visualization of Data Structure by Prof. D. Galles:
- www.cs.usfca.edu/~galles/visualization/BPlusTree.html
Lecture last week: Index construction

- Term/TermID mapping

let's take 10 bytes in average  4 bytes

14 bytes x 100,000,000 = 1.4 GB

4 bytes (always)        4 byte

8 bytes x 100,000,000 = 0.8 GB
Lecture last week: Index construction
BSBI: Blocked Sort-Based Indexing

- Shard collection of documents to blocks
- Process each block
  - Parse termID-docID pairs
  - Sort pairs according to termID
  - Merge pairs to postings list
  - Write back intermediate results
- Merge intermediate results into final index
- Complexity: $O(T \log T)$
  - Highest complexity: sorting
Shard collection of documents to blocks
Process each block
  - Parse term-docID pairs
  - Create dictionary (with dynamic postings list size)
  - Sort on terms
  - Write back intermediate results
Merge intermediate results into final index
Complexity: O(T)
Lecture last week: Index construction

MapReduce
Lecture last week: Index construction
Updating index

- Periodic reconstruction
  - Used in practice
- Other solution: Auxiliary index
  - Invalidation bit vector
  - Logarithmic merging
Exercise 4: Index construction
Moodle questions

- Questions about BSBI/SPIMI
- Calculation of running time for MapReduce
- Questions about logarithmic merging
Exercise 4: Index construction
Programming

- Implement BSBI algorithm
- Most of the code already provided
- Your task:
  - bsbi_invert()
  - merge_blocks()
Exercise 4: Index construction Programming

- Implement `bsbi_invert()`
  - block is potentially unordered list of (termid, docid) pairs.
  - Computes inverted index for each block
  - Should return list of `PostingsList()` tuples sorted by termid
  - Similar to previous exercises

```python
def bsbi_invert(block):
    """
    Compute inverted index for block.
    Return inverted index as (sorted by termid) list of PostingsList() tuples.
    """
    postings = []
    # Assignment: implement construction of inverted index for list of tuples
    # given in block
    # 1. Sort block containing term id.
    # Hint python's sort option takes a key function, see e.g.
    # https://developers.google.com/edu/python/sorting#custom-sorting-with-key
    # 2. Construct inverted index from term id/docid pairs
    return postings
```
Some information about `PostingsList`:

- `PostingsList = namedtuple('PostingsList', ['termid', 'postings'])`
- Python’s `namedtuple` function creates simple tuple subclasses
- Creation:
  ```python
  new_elem = PostingsList(termid=your_variable, postings=your_list_of_postings)
  ```
- Can access tuple fields by name:
  ```python
  new_elem.termid
  new_elem.postings
  ```
Exercise 4: Index construction
Programming

- Complete implementation of `merge_blocks()`
  - Merges all the blocks stored in the files listed in `blockfiles`
  - Write a merged index to `outfile`
- Your task:
  - Implement helper function `select_next_termid()`
  - Implement loop going over all `readbufs` and merge postings lists for selected `termid`
Exercise 4: Index construction
Programming

- Implement helper function `select_next_termid()`:  
  - Input `readbufs` is list of input read buffers. 
    - Can possibly be empty 
    - Otherwise contains sorted postings list created earlier 
  - Return the next `termid` for which we compute the complete postings list

```python
def select_next_termid(readbufs):
    
    Select next termid for which to compute complete postings list

    readbufs is the list of input read buffers. This list may contain None for
    input files which we have already processed completely.
    
    # Assignment: implement an algorithm to select the next termid for which
    # we will merge postings lists
    pass
```
Exercise 4: Index construction
Programming

- Implement loop for readbuf in readbufs in function merge_blocks()
- Go over all readbufs and merge postings lists for before selected termid
- Make use of provided helper function merge()

```python
while sum(map(len, filter(None, readbufs))) > 0:
    next_termid = select_next_termid(readbufs)
    print("Merging postings lists for termid=\$d" % next_termid)
    postings = []
    for readbuf in readbufs:
        # Hint: to make the surrounding code work, you should remove any items
        # which you process from the read buffers.
        # You can do this by using the following statement: \`del readbuf[0]\`
        # Make use of already implemented merge()
        pass
    # Write merged postings list to output file
    write_postings_list(outf, PostingsList(termid=next_termid, postings=postings))
```

Here is the other function you implemented called