Importing Dependencies

Loading Data

In [1]:

```
from sklearn.datasets import fetch_20newsgroups
newsgroup = fetch_20newsgroups(subset='all')
```

Importing Libraries

In [2]:

```
import re
import nltk
import time
import string
import numpy as np
import pandas as pd
from nltk import ngrams
from tqdm.auto import tqdm
from nltk.tokenize import word_tokenize
```

Preprocessing Dependencies

In [3]:

```
stopwords = nltk.corpus.stopwords.words('english')
stemmer = nltk.stem.SnowballStemmer('english')
lemmatizer = nltk.stem.WordNetLemmatizer()
```

In [4]:

```
def preprocess(article):
    article = word_tokenize(article.lower().strip())
    article = [
        lemmatizer.lemmatize(w.translate(str.maketrans('', '', string.punctuation)))
        for w in article
            if w not in stopwords and w not in string.punctuation
    ]
    return article
```

In [5]:

```
def get_vocabulary(articles):
    vocabulary = set()
    for article in articles:
        for word in article:
            vocabulary.add(word)
    vocabulary = list(vocabulary)
    vocabulary_size = len(vocabulary)
    return (vocabulary, vocabulary_size)
```

Permuterm Indices

In [6]:

```
preprocessed_articles = list(map(preprocess, newsgroup['data']))
document_ids = list(newsgroup['target'])
```

Obtaining Permuterm Indices

In [7]:

```
def get_word_permutations(word):
    word = f'{word.strip()}$'
    permutations = list()
    for i in range(len(word)):
        permute = word[i:] + word[:i]
        permutations.append(permute)
    return permutations
```

In [8]:

```
permutation_index = dict()
for article in preprocessed_articles:
    for token in article:
        if token not in permutation_index:
            permutation_index[token] = get_word_permutations(token)
```

In [9]:

```
permutation_index['hello']
```

Out[9]:

```
['hello$', 'ello$h', 'llo$he', 'lo$hel', 'o$hell', '$hello']
```

BST

In [10]:

```
class Node:
    def __init__(self, word, document_id):
        self.word = word
        self.postings = [document_id]
        self.document_frequency = 1

        self.left = None
        self.right = None

    def add_doc_to_word_posting(self, document_id):
        if document_id not in self.postings:
            self.postings.append(document_id)
            self.document_frequency += 1
```

In [11]:

```
class BST:
    def __init__(self):
        self.root = None

    def insert(self, word, document_id):
        if self.root is None:
            self.root = Node(word, document_id)
            return
        else:
            self.insert_word(self.root, word, document_id)

    def insert_word(self, node, word, document_id):
        if word < node.word:
            if node.left is None:
                node.left = Node(word, document_id)
            else:
                self.insert_word(node.left, word, document_id)
        elif word > node.word:
            if node.right is None:
                node.right = Node(word, document_id)
            else:
                self.insert_word(node.right, word, document_id)
        else:
            node.add_doc_to_word_posting(document_id)

    def search(self, word):
        if self.root:
            return self.search_word(self.root, word)
        else:
            return None

    def search_word(self, node, word):
        if word < node.word:
            if node.left is None:
                return None
            else:
                return self.search_word(node.left, word)
        elif word > node.word:
            if node.right is None:
                return None
            else:
                return self.search_word(node.right, word)
        else:
            return (node.postings, node.document_frequency)
```

In [12]:

```
def create_search_tree(articles, document_ids):
    search_tree = BST()
    for i in tqdm(range(len(articles))):
        article = articles[i]
        document_id = document_ids[i]
        for word in article:
            search_tree.insert(word, document_id)
    return search_tree
```

In [13]:

```
search_tree = create_search_tree(preprocessed_articles, document_ids)
```

```
100%                                     18846/18846 [00:38<00:00, 492.84it/s]
```

Searching Query with Permuterm Indices

In [14]:

```
def search_permuterm(query):
    if '*' in query:
        query = f'{query.strip()}$'
        for i in range(len(query)):
            permute = query[i:] + query[:i]
            if permute[-1] == '*':
                break
        query = permute

        query = list(filter(bool, query.split('*')))
        search_word = list(filter(lambda x: '$' in x, query))[0]
        filter_words = list(filter(lambda x: not('$' in x), query))

        search_words = set()
        for token in permutation_index:
            if token == query:
                filter_flag = [w in token for w in filter_words]
                if all(filter_flag):
                    search_words.add(token)
            else:
                for permute in permutation_index[token]:
                    if search_word in permute:
                        filter_flag = [w in permute for w in filter_words]
                        if all(filter_flag):
                            search_words.add(token)
        search_words = list(search_words)
    else:
        search_words = [query]

    print(search_words)
    posting_lists = [set(search_tree.search(w)[0]) for w in search_words]
    posting_list = set.intersection(*posting_lists)
    print(posting_list)
```

In [15]:

```
search_permuterm("ind*ia")
```

```
['indonesia', 'india']
{0, 18, 13, 14}
```

K-Gram Indices

In [16]:

```
K = 2
```

Obtaining K-Gram Indices

In [17]:

```
def get_kgrams(word, k):
    word = f'${word.strip()}$'
    kgrams = ngrams(word, k)
    kgrams = list(map(lambda x: ''.join(x), kgrams))
    return kgrams


def create_kgram_index(articles, k=3):
    kgram_index = dict()
    for article in articles:
        for token in article:
            kgrams = get_kgrams(token, k)
            for kgram in kgrams:
                if kgram not in kgram_index:
                    kgram_index[kgram] = set()
                kgram_index[kgram].add(token)

    for kgram in kgram_index:
        kgram_index[kgram] = sorted(list(kgram_index[kgram]))

    return kgram_index
```

In [18]:

```
kgram_index = create_kgram_index(preprocessed_articles, K)
```

Searching with K-Gram Indices

In [19]:

```
def search_kgram(query):
    if '*' in query:
        query_regex = query.replace('*', '.*')
        query_kgrams = get_kgrams(query, K)
        query_kgrams = list(filter(lambda x: not('*' in x), query_kgrams))
        search_words = list()
        for query_kgram in query_kgrams:
            search_words.append(set(kgram_index[query_kgram]))

        search_words = list(set.intersection(*search_words))
        search_words = [w for w in search_words if re.match(query_regex, w).span()[1] == le

    else:
        search_words = [query]

    print(search_words)
    posting_lists = [set(search_tree.search(w)[0]) for w in search_words]
    posting_list = set.intersection(*posting_lists)
    print(posting_list)
```

In [20]:

```
search_kgram('ind*ia')
```

```
['indonesia', 'india']
{0, 18, 13, 14}
```

Word Count

Map Reduce

In [21]:

```
def mapper(articles):
    for article in articles:
        for token in article:
            yield token

def reducer(articles):
    mapper_function = mapper(articles)
    collection_frequency = dict()
    while True:
        try:
            token = mapper_function.__next__()
            if token not in collection_frequency:
                collection_frequency[token] = 0
            collection_frequency[token] += 1
        except StopIteration:
            break
    return collection_frequency
```

In [22]:

```
collection_frequency = reducer(preprocessed_articles)
```

In [23]:

```
for token, frequency in list(collection_frequency.items())[:5]:
    print(f'Word: {token}\nFrequency: {frequency}\n')
```

```
Word: mamatha
Frequency: 12

Word: devineni
Frequency: 12

Word: ratnam
Frequency: 16

Word: mr47
Frequency: 12

Word: andrewcmuedu
Frequency: 495
```

Collection Frequency

In [24]:

```
def get_collection_frequency(articles):
    collection_frequency = dict()
    for article in articles:
        for token in article:
            if token not in collection_frequency:
                collection_frequency[token] = 0
            collection_frequency[token] += 1

    # collection_frequency = list(collection_frequency.items())
    # collection_frequency.sort(key=lambda x: x[0])
    # collection_frequency = dict(collection_frequency)
    return collection_frequency
```

In [25]:

```
collection_frequency = get_collection_frequency(preprocessed_articles)
```

In [26]:

```
for token, frequency in list(collection_frequency.items())[:5]:
    print(f'Word: {token}\nFrequency: {frequency}\n')
```

```
Word: mamatha
Frequency: 12

Word: devineni
Frequency: 12

Word: ratnam
Frequency: 16

Word: mr47
Frequency: 12

Word: andrewcmuedu
Frequency: 495
```