SUYASH PRATAP SINGH(181B226)

TASKS:-

1. Write a program to implement the Inverted index information retrieval model.
2. Take the example given in lecture slides for document input and query.
3. Output should be the document number and the resultant document.
4. Make use of the merge algorithm given in the slides

In [1]:
```
class InvertedIndex(dict):
    def __init__(self, docs):
        self.docs = docs

        for doc_index,doc in enumerate(docs):
            for term in doc.split(" "):
                self[term].append(doc_index)

    def __missing__(self, term):
        # operate like defaultdict(list)
        self[term] = []
        return self[term]

    def search(self, term):
        return self.get(term) or 'No results'


docs=["new home sales top forecasts june ",
      "home sales rise in july june",
      "increase in home sales in july",
      "july new home sales rise",
      'beer',
      ]

ix = InvertedIndex(docs)
print(ix.__dict__)
print('sales:',ix.search("sales"))
print('june:', ix.search('june'))
print('beer:', ix.search('beer'))
print('july:', ix.search('july'))
```

```
{'docs': ['new home sales top forecasts june ', 'home sales rise in july jun
e', 'increase in home sales in july', 'july new home sales rise', 'beer']}
sales: [0, 1, 2, 3]
june: [0, 1]
beer: [4]
july: [1, 2, 3]
```

Inverted Index

Associate a collection of terms (lexicon) with the documents that contain those terms.

The data structure is much more dense than a Document Term Matrix.

In [14]:
```
review_1 = "I did enact Julius Caesar I was killed i' the Capitol; Brutus kill
ed me."
review_2 = "So let it be with Caesar. The noble Brutus hath told you Caesar wa
s ambitious"
```

In [15]:
```
docs = [review_1, review_2]
docs
```

Out[15]:
```
["I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.",
 'So let it be with Caesar. The noble Brutus hath told you Caesar was ambitio
us']
```

In [16]:
```
# Gather the set of all unique terms

unique_terms = {term for doc in docs for term in doc.split()}
unique_terms
```

Out[16]:
```
{'Brutus',
 'Caesar',
 'Caesar.',
 'Capitol;',
 'I',
 'Julius',
 'So',
 'The',
 'ambitious',
 'be',
 'did',
 'enact',
 'hath',
 "i'",
 'it',
 'killed',
 'let',
 'me.',
 'noble',
 'the',
 'told',
 'was',
 'with',
 'you'}
```

In [17]:
```
inverted_index = {}

for i, doc in enumerate(docs):
    for term in doc.split():
        if term in inverted_index:
            inverted_index[term].add(i)
        else: inverted_index[term] = {i}

inverted_index
```

Out[17]:
```
{'I': {0},
 'did': {0},
 'enact': {0},
 'Julius': {0},
 'Caesar': {0, 1},
 'was': {0, 1},
 'killed': {0},
 "i'": {0},
 'the': {0},
 'Capitol;': {0},
 'Brutus': {0, 1},
 'me.': {0},
 'So': {1},
 'let': {1},
 'it': {1},
 'be': {1},
 'with': {1},
 'Caesar.': {1},
 'The': {1},
 'noble': {1},
 'hath': {1},
 'told': {1},
 'you': {1},
 'ambitious': {1}}
```

In [18]:
```
# Now we can get posting lists for any term
posting_list = inverted_index['Brutus']
posting_list
```

Out[18]:
```
{0, 1}
```

In [19]:
```
# now we can perform boolean operations on postings lists for Boolean search o
perations

def and_postings(posting1, posting2):
    p1 = 0
    p2 = 0
    result = list()
    while p1 < len(posting1) and p2 < len(posting2):
        if posting1[p1] == posting2[p2]:
            result.append(posting1[p1])
            p1 += 1
            p2 += 1
        elif posting1[p1] > posting2[p2]:
            p2 += 1
        else:
            p1 += 1
    return result
```

In [20]:
```
pl_1 = list(inverted_index['Brutus'])
pl_2 = list(inverted_index['Caesar'])
or_postings(pl_1, pl_2)
```

Out[20]:
```
[0, 1]
```

In [2]:
```
review_1 = "new home sales top forecasts"
review_2 = "home sales rise in july"
review_3 = "increase in home sales in july"
review_4 = "july new home sales rise"
```

In [3]:
```
docs = [review_1, review_2, review_3, review_4]
docs
```

Out[3]:
```
['new home sales top forecasts',
 'home sales rise in july',
 'increase in home sales in july',
 'july new home sales rise']
```

In [4]:
```
# Gather the set of all unique terms

unique_terms = {term for doc in docs for term in doc.split()}
unique_terms
```

Out[4]:
```
{'forecasts', 'home', 'in', 'increase', 'july', 'new', 'rise', 'sales', 'to
p'}
```

In [5]:
```
# Construct an inverted index
# here as a Python dictionary for ease of interpretability

inverted_index = {}

for i, doc in enumerate(docs):
    for term in doc.split():
        if term in inverted_index:
            inverted_index[term].add(i)
        else: inverted_index[term] = {i}

inverted_index
```

Out[5]:
```
{'new': {0, 3},
 'home': {0, 1, 2, 3},
 'sales': {0, 1, 2, 3},
 'top': {0},
 'forecasts': {0},
 'rise': {1, 3},
 'in': {1, 2},
 'july': {1, 2, 3},
 'increase': {2}}
```

In [6]:
```
# Now we can get posting lists for any term
posting_list = inverted_index['sales']
posting_list
```

Out[6]:
```
{0, 1, 2, 3}
```

In [7]:
```
# now we can perform boolean operations on postings lists for Boolean search o
perations
def or_postings(posting1, posting2):
    p1 = 0
    p2 = 0
    result = list()
    while p1 < len(posting1) and p2 < len(posting2):
        if posting1[p1] == posting2[p2]:
            result.append(posting1[p1])
            p1 += 1
            p2 += 1
        elif posting1[p1] > posting2[p2]:
            result.append(posting2[p2])
            p2 += 1
        else:
            result.append(posting1[p1])
            p1 += 1
    while p1 < len(posting1):
        result.append(posting1[p1])
        p1 += 1
    while p2 < len(posting2):
        result.append(posting2[p2])
        p2 += 1
    return result
```

```
In [11]: pl_1 = list(inverted_index['in'])
         pl_2 = list(inverted_index['july'])
         or_postings(pl_1, pl_2)

Out[11]: [1, 2, 3]
```

```
In [12]: def and_postings(posting1, posting2):
             p1 = 0
             p2 = 0
             result = list()
             while p1 < len(posting1) and p2 < len(posting2):
                 if posting1[p1] == posting2[p2]:
                     result.append(posting1[p1])
                     p1 += 1
                     p2 += 1
                 elif posting1[p1] > posting2[p2]:
                     p2 += 1
                 else:
                     p1 += 1
             return result
```

```
In [13]: pl_1 = list(inverted_index['in'])
         pl_2 = list(inverted_index['july'])
         and_postings(pl_1, pl_2)

Out[13]: [1, 2]
```

THANK YOU