# MODUL 3: BOOLEAN RETRIEVAL DAN INVERTED INDEX

## 3.1 Deskripsi Singkat

Pada *information retrieval*, indeks dibuat terlebih dahulu untuk menghindari pencarian secara linier dari teks pada setiap *query*. *Inverted index* adalah suatu indeks dimana *term* dihubungkan dengan lokasi dokumen dimana *term* tersebut berada (*posting lists*).

*Boolean Retrieval* merupakan proses pencarian informasi dari *query* yang menggunakan ekspresi *Boolean*, yaitu menggunakan operator logika AND, OR dan NOT. Hasil *boolean retrieval* yaitu dokumen relevan (nilai biner: 1) atau dokumen tidak relevan (nilai biner: 0). Dalam pengerjaan operator *boolean* (AND, NOT, OR) ada urutan pengerjaannya (*operator precedence*), yaitu memprioritaskan yang berada dalam kurung (), baru selanjutnya NOT, AND, dan OR.

## 3.2 Tujuan Praktikum

Setelah praktikum pada modul 3 ini diharapkan mahasiswa mempunyai kompetensi sebagai berikut.

1) Dapat membuat inverted index
2) Dapat melakukan boolean retrieval dengan memproses boolean query

## 3.3 Material Praktikum

Tidak ada

## 3.4 Kegiatan Praktikum

### A. Inverted Index

Diketahui terdapat 3 dokumen dengan *term* masing-masing berdasarkan hasil tokenisasi pada Modul 2C. Kemudian didapatkan *term* pada korpus (keseluruhan koleksi dokumen) yang disimpan di suatu list 2D bernama `corpus_term`.

```
doc1_term = ["pengembangan", "sistem", "informasi",
"penjadwalan"]
doc2_term = ["pengembangan", "model", "analisis", "sentimen",
"berita"]
doc3_term = ["analisis", "sistem", "input", "output"]

corpus_term = [doc1_term, doc2_term, doc3_term]
```

Berikut adalah kode untuk mendapatkan inverted index dengan term pada korpus tersebut. Tambahkan fungsi stemming sehingga term yang tersimpan di inverted index adalah term yang berupa kata dasar dengan memanggil fungsi `stemming` (sudah dipelajari di Modul 2).

```
inverted_index = {}

for i in range(len(corpus_term)):
      for item in corpus_term[i]:
            item = stemming(item)
            if item not in inverted_index:
                  inverted_index[item] = []
            if (item in inverted_index) and ((i+1) not in
inverted_index[item]):
                  inverted_index[item].append(i+1)
print(inverted_index)
```

Perhatikan isi dari variabel `inverted index`.

**B. Boolean Retrieval**

Simpan terlebih dahulu kode `class BooleanModel` berikut dengan nama file `boolean.py`.

```
import math
class BooleanModel():

    @staticmethod
    def and_operation(left_operand, right_operand):
        # perform 'merge'
        result = [] # results list to be returned
        l_index = 0 # current index in left_operand
        r_index = 0 # current index in right_operand
        l_skip = int(math.sqrt(len(left_operand)))
        # skip pointer distance for l_index
        r_skip = int(math.sqrt(len(right_operand)))
        # skip pointer distance for r_index

        while (l_index < len(left_operand) and r_index <
len(right_operand)):
            l_item = left_operand[l_index]
            r_item = right_operand[r_index]

            # case 1: if match
            if (l_item == r_item):
                result.append(l_item)    # add to results
                l_index += 1             # advance left index
                r_index += 1             # advance right index

            # case 2: if left item is more than right item
            elif (l_item > r_item):
                # if r_index can be skipped (if new r_index is
still within range and resulting item is <= left item)
                if (r_index + r_skip < len(right_operand)) and
right_operand[r_index + r_skip] <= l_item:
                    r_index += r_skip
                # else advance r_index by 1
```

```
            # case 3: if left item is less than right item
            else:
                # if l_index can be skipped (if new l_index is still
within range and resulting item is <= right item)
                if (l_index + l_skip < len(left_operand)) and
left_operand[l_index + l_skip] <= r_item:
                    l_index += l_skip
                # else advance l_index by 1
                else:
                    l_index += 1

        return result

    @staticmethod
    def or_operation(left_operand, right_operand):
        result = []     # union of left and right operand
        l_index = 0     # current index in left_operand
        r_index = 0     # current index in right_operand

        # while lists have not yet been covered
        while (l_index < len(left_operand) or r_index <
len(right_operand)):
            # if both list are not yet exhausted
            if (l_index < len(left_operand) and r_index <
len(right_operand)):
                l_item = left_operand[l_index]  # current item in
left_operand
                r_item = right_operand[r_index] # current item in
right_operand

                # case 1: if items are equal, add either one to result
and advance both pointers
                if (l_item == r_item):
                    result.append(l_item)
                    l_index += 1
                    r_index += 1

                # case 2: l_item greater than r_item, add r_item and
advance r_index
                elif (l_item > r_item):
                    result.append(r_item)
                    r_index += 1

                # case 3: l_item lower than r_item, add l_item and
advance l_index
                else:
                    result.append(l_item)
                    l_index += 1

            # if left_operand list is exhausted, append r_item and
advance r_index
            elif (l_index >= len(left_operand)):
                r_item = right_operand[r_index]
                result.append(r_item)
                r_index += 1
```

```
            # else if right_operand list is exhausted, append
l_item and advance l_index
            else:
                l_item = left_operand[l_index]
                result.append(l_item)
                l_index += 1

        return result

    @staticmethod
    def not_operation(right_operand, indexed_docIDs):
        # complement of an empty list is list of all indexed docIDs
        if (not right_operand):
            return indexed_docIDs

        result = []
        r_index = 0 # index for right operand
        for item in indexed_docIDs:
            # if item do not match that in right_operand, it
belongs to compliment
            if (item != right_operand[r_index]):
                result.append(item)
            # else if item matches and r_index still can progress,
advance it by 1
            elif (r_index + 1 < len(right_operand)):
                r_index += 1
```

Kemudian buat file python, misalnya bernama `latihan_3b.py` .Pada file tersebut, buat terlebih dahulu kode untuk membuat inverted index seperti pada bagian 3A, sehingga didapatkan variabel `inverted_index`. Kemudian buat fungsi untuk melakukan parsing boolean query (dengan Shunting Yard Algorithm), sebagai berikut.

```
def parse_query(infix_tokens):
    """ Parse Query
    Parsing done using Shunting Yard Algorithm
    """
    precedence = {}
    precedence['NOT'] = 3
    precedence['AND'] = 2
    precedence['OR'] = 1
    precedence['('] = 0
    precedence[')'] = 0

    output = []
    operator_stack = []

    for token in infix_tokens:
        if (token == '('):
            operator_stack.append(token)

        # if right bracket, pop all operators from operator stack
onto output until we hit left bracket
        elif (token == ')'):
            operator = operator_stack.pop()
            while operator != '(':
                output.append(operator)
```

```
        # if operator, pop operators from operator stack to queue if
they are of higher precedence
        elif (token in precedence):
            # if operator stack is not empty
            if (operator_stack):
                current_operator = operator_stack[-1]
                while (operator_stack and precedence[current_operator] >
precedence[token]):
                    output.append(operator_stack.pop())
                    if (operator_stack):
                        current_operator = operator_stack[-1]
            operator_stack.append(token) # add token to stack
        else:
            output.append(token.lower())

    # while there are still operators on the stack, pop them into the
queue
    while (operator_stack):
        output.append(operator_stack.pop())

    return output
```

Fungsi di atas dipanggil untuk memproses boolean query untuk suatu inverted index, dengan menggunakan fungsi berikut.

```
import collections
from boolean import BooleanModel

def process_query(query, n_docs, inverted_index):
    # prepare query list
    query = query.replace('(', '( ')
    query = query.replace(')', ' )')
    query = query.split(' ')
    print(query)

    indexed_docIDs = list(range(1, n_docs + 1))
    results_stack = []
    postfix_queue = collections.deque(parse_query(query)) # get query in
postfix notation as a queue

    while postfix_queue:
        token = postfix_queue.popleft()
        result = [] # the evaluated result at each stage
        # if operand, add postings list for term to results stack
        if (token != 'AND' and token != 'OR' and token != 'NOT'):
            token = stemming(token) # stem the token
            # default empty list if not in dictionary
            if (token in inverted_index):
                result = inverted_index[token]

        elif (token == 'AND'):
            right_operand = results_stack.pop()
            left_operand = results_stack.pop()
            result = BooleanModel.and_operation(left_operand,
right_operand)   # evaluate AND
```

```
        elif (token == 'OR'):
            right_operand = results_stack.pop()
            left_operand = results_stack.pop()
            result = BooleanModel.or_operation(left_operand,
right_operand)    # evaluate OR

        elif (token == 'NOT'):
            right_operand = results_stack.pop()
            result = BooleanModel.not_operation(right_operand,
indexed_docIDs) # evaluate NOT
        results_stack.append(result)

    # NOTE: at this point results_stack should only have one item and it
is the final result
    if len(results_stack) != 1:
        print("ERROR: Invalid Query. Please check query syntax.") #
check for errors
        return None

    return results_stack.pop()
```

Gunakan beberapa contoh query berikut untuk mengecek hasil dokumen yang dikembalikan model boolean retrieval dengan memanggil fungsi `process_query`.

1. sistem
2. informasi
3. sentimen
4. sistem AND informasi
5. sistem AND informasi OR sentimen
6. sistem OR informasi OR sentimen
7. (sistem AND informasi) OR sentimen

Pastikan Anda memahami alur pemrosesan boolean query di atas.

## 3.5 Penugasan

1. Menggunakan sekumpulan dokumen pada folder ”berita”, setelah dilakukan preprocessing pada penugasan Modul 2, tambahkan kode untuk menghasilkan inverted index dengan output berupa *term* dan daftar lokasinya (*posting lists*).
2. Kemudian tambahkan kode untuk melakukan boolean retrieval dari inverted index pada Penugasan 1. Perhatikan daftar dokumen yang dikembalikan ketika menuliskan query berikut.
    a. corona
    b. covid
    c. vaksin
    d. corona OR covid
    e. vaksin AND corona
    f. vaksin AND (corona OR covid)