CSE 435/535 PROJECT TWO

Rajkushal Ananthkumar

University at Buffalo The State University of New York

Introduction

- What is project two all about ?
  - Creating your own inverted index, postings list and implementing strategies to answering Boolean queries. Once answered, the result needs to be ranked.

- What are the fundamental IR concepts I need to know to get started ?
  - Dictionary
  - Postings
  - Inverted index
  - Boolean operations (AND/OR)
  - TF-IDF

Introduction

- How much programming background is needed to complete this project ?
  - You are expected to know how to work with functions and be familiar with basic data structures such as Queue, Heap, HashMaps, Lists and so on.
  - You are expected to know how to work with file IO operations.
- Which programming language will we be using for this project ?
  - Python 3
- Can we use other programming language ?
  - No

Introduction

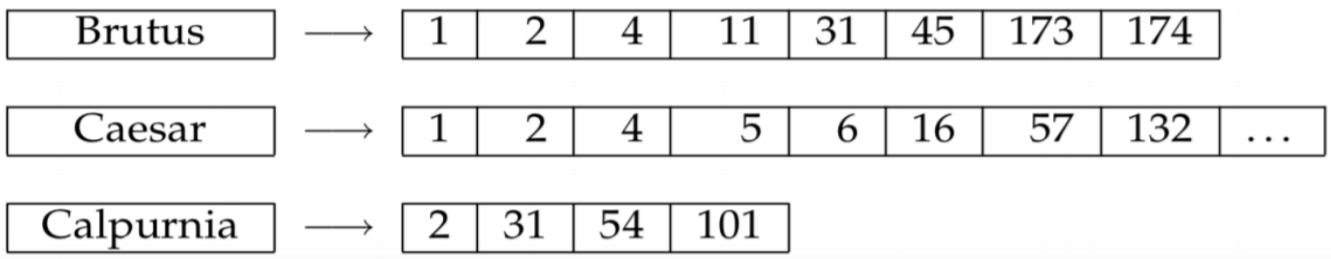
- What dataset are we using for this project ?
  - input_corpus.txt is a tab-delimited file where each line is a document; the first field is the document ID, and the second is a sentence.

1839	Feeling inspired? Make a meal for your family or roommates
1875	Make an effort to get to know someone you don't usually talk to.
2014	Help someone struggling with heavy bags
2095	Recycle 3 things today

- Do I need to perform any normalization on the input corpus (Lowercase, Stopwords, Stemming..) ?
  - No
- Can I expect the same dataset to be used for final grading?
  - No

How does an inverted index look like ?

- Doc 1: " I did enact Julius Caesar: I was killed i' the Capitol; Brutus killed me"
- Doc 2: "So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious."



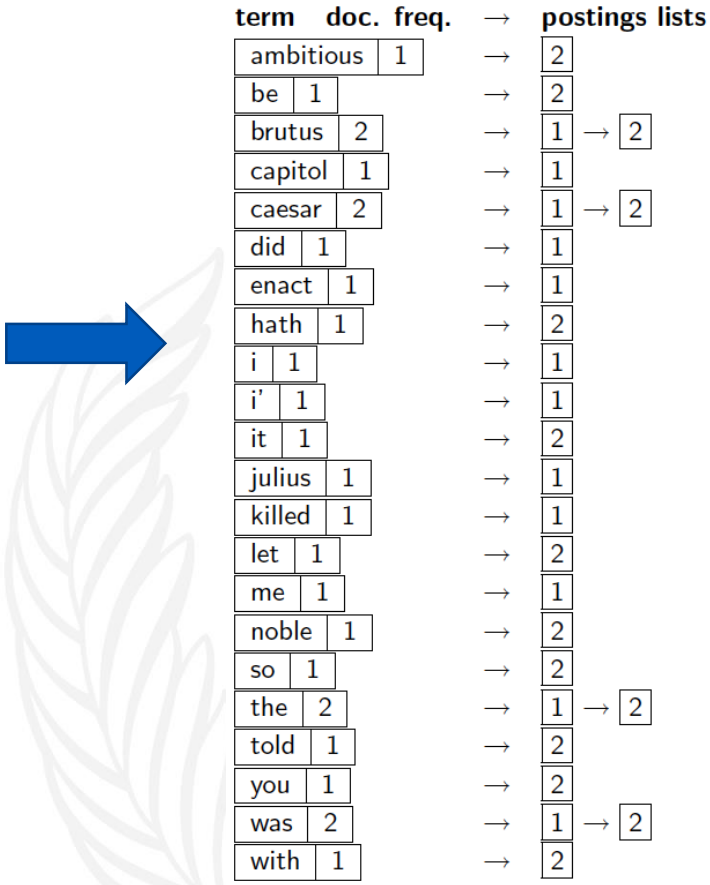
How to create Inverted Index for this project ?

1. Collect the documents to be indexed.
2. Tokenize the text, turning each document into a list of tokens.
3. Index the documents that each term occurs in by creating an inverted index, consisting of a dictionary and postings.

Indexer steps: Dictionary & Postings

- We need variable-size postings lists
- Multiple term entries in a single document are merged.
- Split into Dictionary and Postings
- Doc Frequency and Term frequency information is recorded.

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2



How to get started ?

- Basic data structures/operations -
  - Lists
  - Queue
  - HashMap
  - Sorting
  - Read/Write file

- Creating your own Postings Lists
  - Stored in the form of Linked Lists

Data Structures

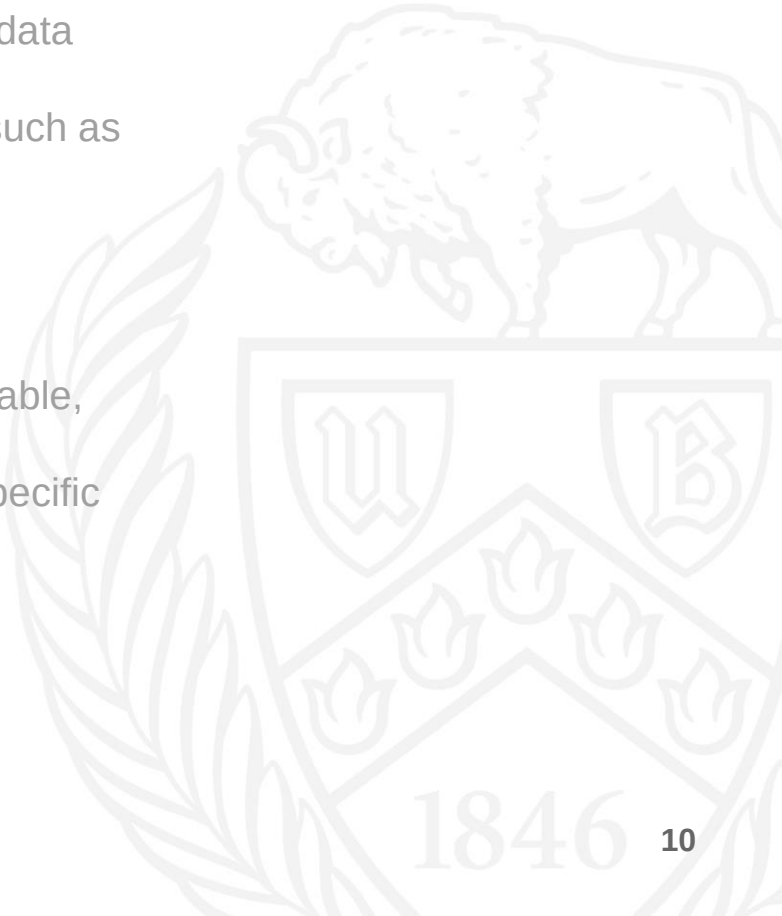
- Lists
  - Ordered collection of objects
  - Duplicate values are allowed
  - Preserves insertion order

- Queue (First-In-First-Out)
  - List like data structure that provides restricted access to its elements.
  - Enqueue an item at rear
  - Dequeue at front

Data Structures

- Map (Dictionary)
  - Unordered collection of data values, used to store data values in a key:value pair
  - Keys must be unique and of *immutable* data type such as Strings, Integers and tuples
  - Values can be repeated and be of any type.

- Set
  - Unordered collection data type that is iterable, mutable, and has no duplicate.
  - Highly optimized method for checking whether a specific element is contained in the set.



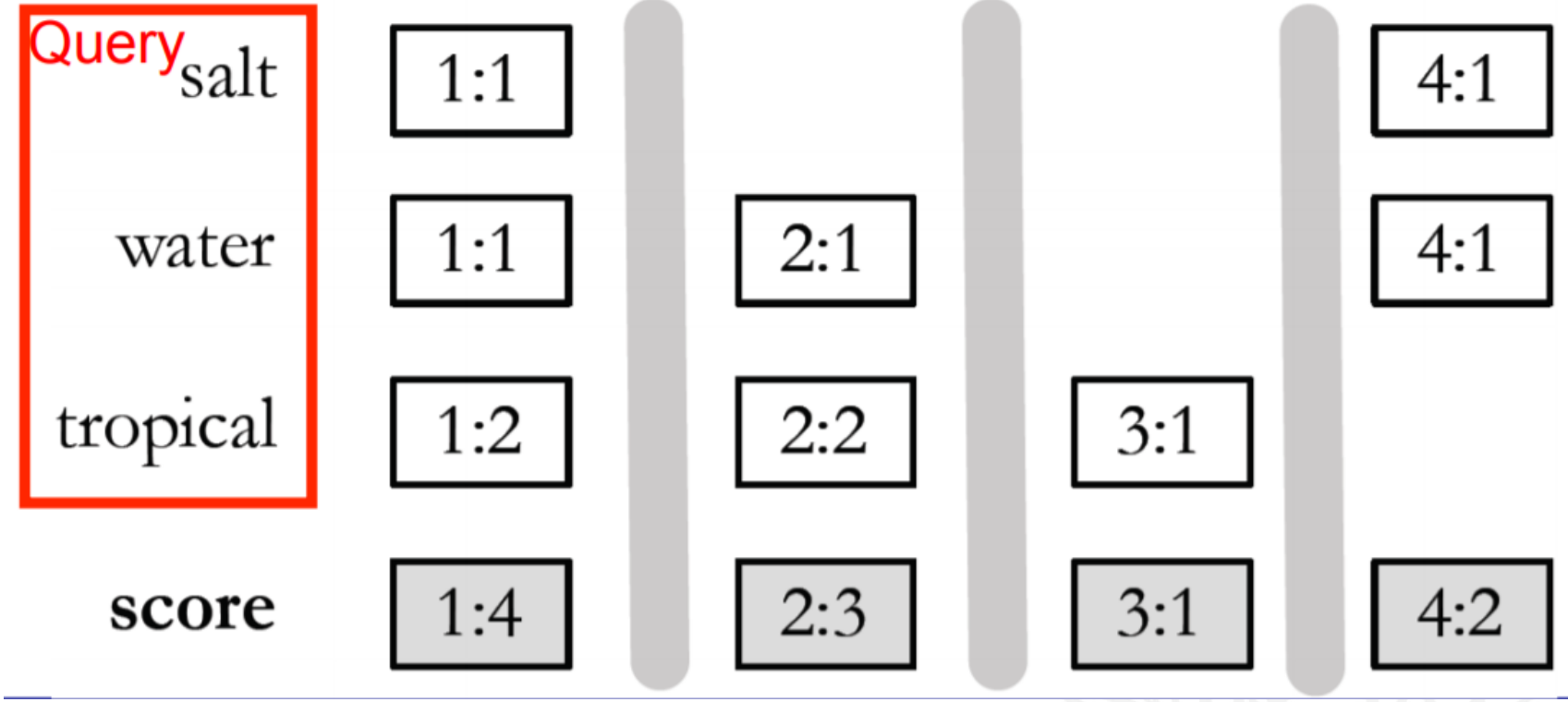
TAAT vs DAAT

- TAAT = “Term at a time”
  - Scan postings list one at a time, maintain a set of potential matching documents along with their partial scores.
  - Reads posting lists for query terms ⟨ t1, …, tn ⟩ successively.
  - Maintains an accumulator for each result document with value.

- DAAT = “Document at a time”
  - Scan postings lists in parallel, identifying at each point the next potential candidate document and scoring it.
  - Reads posting lists for query terms ⟨ t1, …, tn ⟩ concurrently

Document-at-a-time Evaluation

The conceptually simplest query answering method.



Algorithm

```
procedure DocumentAtATimeRetrieval(Q, I, f, g, k)
    L ← Array()
    R ← PriorityQueue(k)
    for all terms w_i in Q do
        l_i ← InvertedList(w_i, I)
        L.add( l_i )
    end for
    for all documents d ∈ I do
        for all inverted lists l_i in L do
            if l_i points to d then
                s_D ← s_D + g_i(Q) f_i(l_i)        ▷ Update the document score
                l_i.movePastDocument( d )
            end if
        end for
        R.add( s_D, D )
    end for
    return the top k results from R
end procedure
```

Find posting lists

Can be implemented efficiently by keeping the top-k list at anytime

Union or Intersection

Term-at-a-time Evaluation

```
procedure TermAtATimeRetrieval(Q, I, f, g k)
    A ← HashTable()
    L ← Array()
    R ← PriorityQueue(k)
    for all terms w_i in Q do
        l_i ← InvertedList(w_i, I)
        L.add( l_i )
    end for
    for all lists l_i ∈ L do
        while l_i is not finished do
            d ← l_i.getCurrentDocument()
            A_d ← A_d + g_i(Q) f(l_i)
            l_i.moveToNextDocument()
        end while
    end for
    for all accumulators A_d in A do
        s_D ← A_d                    ▷ Accumulator contains the document score
        R.add( s_D, D )
    end for
    return the top k results from R
end procedure
```

Compute scores on one term

Can be implemented efficiently by keeping the top-k list at anytime

Adapt to project 2

- Traverse the postings lists in the same way.
- Change the ranking method to binary scoring.
- If the term appears in the doc, score it as 1, otherwise 0.

Comparison

- Memory usage
  - The document-at-a-time only needs to maintain a priority queue R of a limited number of results.
  - The term-at-a-time needs to store the current scores for all documents.
- Disk access
  - The document-at-a-time needs more disk seeking and buffers for seeking since multiple lists are read in a synchronized way.
  - The term-at-a-time reads through each inverted list from start to end – requiring minimal disk seeking and buffer.

DAAT – AND/OR

Consider the below postings lists:

- Allen : 5, 8, 14
- Aldus: 2, 7, 11
- Anthony: 4,10

Results (AND): empty

No of comparison (AND): 12

[{(2,5),(2,4)},{(7,5),(7,4)},{(10,5),(10,7)},{(8,7),(8,10)},{(11,8),(11,10)},{(14,11),(14,10)}]

IMPORTANT – These values are just to provide an illustration, the numbers may vary based on your implementation

Questions ?

- Post on Piazza, participate in discussions. AVOID emails, create a private post if you have to. But, please do your research before posting questions.
- DO NOT post your code on piazza. Be mindful of the academic integrity issues.
- Put comments in your code where necessary.