**Assignment 2 Part 1: Entropy**

**1. Data Structures we used and argumentation**

* **String**: For text content and n-grams. This is fundamental for text manipulation.
* **File**: For file reading (book1 and book2 text files)
* **HashMap**: Stores n-gram frequencies.
* **HashSet**: Stores unique characters for 0-gram calculation.
* **PriorityQueue**: Used as fixed-size (5) min/max heaps for finding top/bottom 5 n-grams. As soon as the priority queue exceeds length 5, n-gram with the smallest frequency is polled from the priority queue with maxheap for least frequent n-gram and the opposite. The reason is to maintain sorted elements avoiding full sorting. It
* **ArrayList**: Stores the formatted lists of most/least frequent n-grams. The reason is a more flexible dynamic array for result presentation.

### **2. Pseudo-code for the Algorithm**

#### **main** For each .txt file in data folder: Print file name Text = readAndCleanText(file) For n = 0 to 10: CalculateAndPrintEntropy(text, n)

#### **ReadAndCleanText** Read file content Lowercase all text Remove non-letters (keep spaces) Replace multiple spaces with one Trim leading/trailing spaces Return cleaned text

#### **calculateAndPrintEntropy**

If n == 0:

Add unique non-space chars in a HashSet

entropy = log2(unique\_count)

Print result

Return;

Remove spaces from text

If text length < n:

print warning,

return;

Create HashMap for n-gram counts

For each n-gram in text:

Store the substrings in the HashMap

Count occurrences

If map is empty:

print warning

return;

totalOccurrences = sum of counts

Entropy for all n-grams using probability \* log2(probability)

Use minHeap and maxHeap (size 5 max) to track top/bottom n-grams

Print entropy, total n-grams, most/least frequent (at most 5)

### **3. Correctness of Algorithm**

* **Processing Files:** Iterating through each file in the data folder inside project , ensuring both **readAndCleanText** and **calculateAndPrintEntropy** methods for n-grams up to 10, handling any exception
* **readAndCleanText**: Using UTF-8 encoding we ensure reading the file and turning it into a string with minimal data corruption. Successfully cleans text by fixing it to the needed format. By lowercasing, we ensure same-letter grams are treated as the same. Ensures the Albanian characters are not replaced, while the other ones are. Trims all the multiple whitespaces, since we don’t need them for the entropy.
* **calculateAndPrintEntropy**:
  + **N-gram = 0:** We enter each unique character in a HashSet, which is ideal for unique characters, assuming uniform probability. If it occurs a duplicate character during the iteration, it doesn’t put it in the set, therefore ensuring uniqueness. Then, if there were no characters, we don’t have Albanian language content to calculate entropy, therefore it will be zero, otherwise we use the formula for log2(size of set)
  + **N-gram 1-10:** Based on the value of n in the string, we iterate from start to finish moving by one character and putting the ngram in the HashMap with key-value pair. If the ngram is new in HashMap, we initialize a new ngram with default value 1 and then increase if we occur it again. Then finding the sum of all counts from the HashMap and successfully applying the entropy formula. Then, using minheap priority queue for most frequent, each ngram and its value is enqueued. When queue exceeds 5 elements, we remove the root(ngram with smallest frequency), ensuring the queue with largest occurrences and not taking up much space. The opposite as well for least frequent using maxheap priority queue. Then poll each element of priority queues in respecting lists and successfully print the results for entropy, total occurrences, unique n-grams number, five most frequent and less frequent lists.

### **4. Overall Time Complexity**

**O(N × L × nₘₐₓ)**

Since nₘₐₓ = 10 is constant, the time depends mostly on the total number of characters in all files.

* **L** = length of a single text file (number of characters)
* **N** = number of text files
* **nₘₐₓ** = maximum n-gram size (10)

If we consider all the lengths of text files as a single big conceptual length, then the time complexity is the same, but simplified in terms **O(L\_total x** **nₘₐₓ)**

**5. Build Version - 1.0**