A blue text on a black background

AI-generated content may be incorrect.

Algorithms & Complexity

Assignment 2

Semester: Spring 2025 By

Altonel Alushi Parid Gjerasi

Problem Statement

Algorithms and complexity are important in understanding the structure of natural language and mining data patterns. In our project, we deal with two key tasks that help to examine these ideas in detail. The former will consist in estimating the entropy of a natural language, namely Albanian, based upon n-gram models, which range between simple uniform models (0-gram) and more elaborate sequences of characters (up to 10-gram). By running two large Albanian texts through it and comparing the output to English we can see how entropy acts as the model increases. This assists in comprehending the predictability of language and structure. The second task is dedicated to the decomposition of patterns with the help of a string of numbers composed of unknown employee IDs. The aim is to divide this so-called welded number string into viable segments of IDs according to a list of known employees and come up with the best probable match. A second area that has been worked on is to have both tasks have an automatic language detection of user input depending on bigram frequency and having all valid decompositions of the welded ID string. The project is an algorithmic logic and implementation testing project with emphasis on performance and correctness.

Introduction

In this project we will deal with two algorithmic problems that are aimed to investigate some basic concepts of problem-solving and computational complexity through Java. The former is concerned with entropy analysis of the natural language by building character n-gram models. Entropy values (ranging between 0-gram and 10-gram models) are used to compare two Albanian texts and one English text and how predictability varies with the depth of the model. The system also contains the bigram frequency information which compares the two languages and also has a provision where the user types a sentence and it is automatically recognized as either Albanian or English.

The second task consists of splitting a numeric string into viable employee IDs according to a given list. The task is to compute the longest valid decomposition, emulating a scenario like computing the contributors given a compressed committer string. The program also enumerates all possible valid decompositions, namely it does so by means of recursion and keeping track of sequences.

These assignments have a mix of theoretical ideas (recursion, entropy) and practical application, input/output, and data processing, without losing focus on algorithm complexity and correctness.

Data Structures used in the assignment

The project makes use of a number of standard data structures to achieve the core tasks of entropy analysis and welded ID decomposition. For the entropy and language detection component, the main data structure utilized is the Map<String, Integer> in the case of a Java HashMap which is employed to store the frequency counts of each n-gram (where n ranges from 0 to 10). This allows constant-time lookup and insertion, which is ideal for processing large text files efficiently. In the language detection feature, two such maps are built, one for English and one for Albanian bigrams and are used to score a user-input sentence by summing the corresponding frequencies.

In Part II, which handles the commit decomposition logic, a second set of data structures becomes essential. A Map<String, Employee> is used to associate employee IDs with their metadata (surname and name), enabling quick lookups during decomposition. Additionally, List<String> is used to store the current and best decomposition sequences. The recursive nature of the ID splitting logic means that Java’s call stack implicitly serves as the recursion stack for backtracking. Finally, an int counter tracks the total number of valid decompositions.

Each data structure was chosen for its efficiency in access and mutation, particularly given the need for quick lookups in large text corpora (Part I) and the recursive combinatorial nature of the weld-string analysis (Part II).

Implementation of the Algorithm Pseudo Code:

Entropy Computation Algorithm:

FUNCTION computeNGramFrequencies(text, n):

freqMap ← empty map

FOR i FROM 0 TO length(text) - n:

ngram ← substring(text, i, i + n)

freqMap[ngram] ← freqMap.getOrDefault(ngram, 0) + 1

RETURN freqMap

Entropy Calculation:

CopyEdit

FUNCTION calculateEntropy(freqMap, total):

entropy ← 0.0

FOR each ngram IN freqMap:

p ← freqMap[ngram] / total

entropy ← entropy - (p \* log2(p))

RETURN entropy

**Language Detection:**

scss

CopyEdit

FUNCTION detectLanguage(sentence, freqAlb, freqEng):

cleanSentence ← cleanText(sentence)

sentenceBigrams ← computeNGramFrequencies(cleanSentence, 2)

scoreAlb ← compareFrequencies(sentenceBigrams, freqAlb)

scoreEng ← compareFrequencies(sentenceBigrams, freqEng)

RETURN "Albanian" IF scoreAlb > scoreEng ELSE "English"

Commit Owner Detection (CommitOwners)

Read Employee File:

CopyEdit

FUNCTION readEmployees(filename):

employeeMap ← empty map

FOR each line IN file(filename):

id, surname, name ← split line by comma

employeeMap[id] ← new Employee(id, surname, name)

**Recursive Decomposition:**

scss

CopyEdit

FUNCTION findDecomposition(weld, index, currentPath):

IF index == length(weld):

IF currentPath is longest so far:

bestSequence ← currentPath

totalDecompositions += 1

RETURN

FOR i FROM index + 1 TO length(weld):

prefix ← weld.substring(index, i)

IF prefix in employeeMap:

currentPath.add(prefix)

findDecomposition(weld, i, currentPath)

currentPath.removeLast()

Discussion on Algorithm Correctness

The rigor of the EntropyAnalyzer component is based upon firmly established theory of statistical language modeling, specifically that the frequency distribution of n-grams may be used to estimate character-level entropy. The entropy estimation algorithm starts with preprocessing the text, which involves removing punctuation marks, converting to lower case and removing non alphabetic characters to have a consistent input. Next, every valid n-gram of a given length n is made to be extracted out of the cleaned text and their frequencies are stored in a hash map. The frequency counts are used in the application of the entropy formula H = -sum p \* log 2 (p) to calculate the information entropy. The accuracy of this approach is ensured since the full character n-gram space of the text is taken into consideration and the value of entropy is an effective measure of predictability of the language. More precisely, the language detection extension uses bigram frequency vectors in pre-analyzed corpora as a comparison dataset to the user input, which has also undergoes through the same preprocessing pipeline. This uniformity makes the scoring mechanism valid and gives comparable results across languages.

CommitOwners component uses recursive backtracking algorithm to enumerate all possible valid decompositions of a numeric string representing a "weld" of employees into segments representing employee IDs. Its breakdown is intentionally designed: given any potential prefix of the weld string, it will determine whether the substring is equal to some valid employee ID stored in a preloaded map. In case of a match, it calls itself again to search the rest of the string in a depth-first search manner maintaining the current path. At this point the end of the string is encountered, and a valid decomposition has been found, so the total counter is increased. Of all such decompositions the one which corresponds to the equation x + y = 1 is that in which x = 1/3 and y = 2/3.

At this point the end of the string is encountered, and a valid decomposition has been found, so the total counter is increased. Of all such decompositions the one which corresponds to the equation x + y = 1 is that in which x = 1/3 and y = 2/3.

**Time Complexity Analysis**

**EntropyAnalyzer**

**computeNGramFrequencies(text, n):  
This function operates in linear time with respect to the size of the input text. For a text of length L, there are (L - n + 1) n-grams, and each one is processed and stored in a map. Thus, the complexity is O(L).**

**calculateEntropy(freqMap):  
The entropy calculation requires iterating over all unique n-grams stored in the frequency map. If there are k unique n-grams, then the complexity of this step is O(k).**

**detectLanguage(sentence):  
The sentence is cleaned and converted into a set of bigrams, and then these are compared against precomputed frequency maps. The complexity depends on the number of bigrams in the input sentence, typically O(m), where m is the number of bigrams in the sentence. Since these comparisons are constant-time map lookups, the overall detection runs efficiently in O(m) time.**

**CommitOwners**

**readEmployees(filename)**:  
This function reads the employee records from a file and populates a map for ID lookup. For e employee entries, the function performs one pass over the file and inserts each entry into the map, resulting in a time complexity of **O(e)**.

**findDecomposition(weld, index, path)**:  
The recursive decomposition function explores all valid paths through the weld string using backtracking. In the worst case, where every single-digit prefix could be a valid employee ID, the function explores every possible combination of splits, resulting in **O(2^n)** time complexity for a weld string of length n. However, in practice, the number of valid ID prefixes is limited by the content of the employeeMap. This acts as a pruning mechanism, significantly reducing the actual number of recursive calls. Therefore, while the theoretical complexity is exponential, the effective performance is often acceptable for reasonable input sizes.

​

**Correctness Proof**

The correctness of the **EntropyAnalyzer** relies on applying well-established entropy theory. The input texts are preprocessed to eliminate invalid characters, and only meaningful alphabetic tokens are used. N-gram generation is done in a sliding window manner, ensuring every possible n-gram is accounted for. Entropy is calculated using the formula H(X)=−∑p(x)⋅log⁡2(p(x))H(X) = - \sum p(x) \cdot \log\_2(p(x))H(X)=−∑p(x)⋅log2​(p(x)), ensuring each term contributes correctly to the final result. For language detection, the program compares bigram frequencies of the input sentence with two trained corpora, accurately predicting the most similar language based on frequency overlap.

The **CommitOwners** class ensures correctness by performing exhaustive backtracking. At each recursion level, it explores all valid substrings that match known employee IDs and continues until the string is fully parsed. The function terminates if no more valid matches exist. It guarantees completeness by checking all decomposition paths and optimality by keeping track of the longest valid sequence. Every base case is handled properly, ensuring the final output reflects all valid combinations and highlights the best one.

Complexity Analysis

Training (bigram model building):

Time Complexity: O(n), where n is the length of the input corpus.

Space Complexity: O(k), where k is the number of unique bigrams (typically bounded by 26×26).

Prediction (user input classification):

Time Complexity: O(m), where m is the number of bigrams in the user sentence.

Space Complexity: O(1) since only scalar scores are maintained for each language.

Comparison with Brute Force Alternatives

**CommitOwners with Brute Force**

If implemented naively without pruning or hash maps, decomposition would require checking all possible ways to split the weld string, leading to exponential complexity O(2^n). By using hash maps for ID lookup and backtracking only when the current prefix is valid, the implemented solution reduces the search space dramatically, avoiding exploration of invalid paths.

**Entropy and Language Detection Alternatives**

Brute force character prediction using full probability models with full smoothing or deep learning-based language models would be much more resource-intensive. Our bigram-based frequency model offers a lightweight alternative that performs well enough for two-language classification without requiring large-scale training.

References:

1.Kleinberg, J., & Tardos, É. (2006). Algorithm Design. Pearson.

2. Jurafsky, D., & Martin, J. H. (2009). Speech and Language Processing (2nd ed.). Pearson.

3. Dasgupta, S., Papadimitriou, C. H., & Vazirani, U. V. (2006). Algorithms. McGraw-Hill Education.