|  |
| --- |
| 2025 |

Algorithms and Complexity

ASSIGNMENT 2

Aleksandre Avduli

Contents

1. Entropy of Language …………………………………..……………

2. Commit Owners ………………………………………………..

3. Bonus Tasks ……………………………………………………

4.References…………………………………………………………..

Algorithms and Complexity

# Part I - Entropy of Language

Problem Overview:

This part calculates the entropy of texts in the student's native language using n-gram models (n = 0 to 10).

Entropy gives a measure of the information content in the language data, where:

E(X) = -sum p(x) \* log2(p(x))

Two different categories of native texts were used, and for each, entropy and frequency stats were reported.

Data Structures Used:

* Counter from Python's collections: to count n-gram frequencies.
* list: to store and iterate over n-grams.
* str: for preprocessing and handling text.

Pseudocode: Preprocessing:

function preprocess(text): convert text to lowercase remove punctuation and non-letter characters return cleaned text

N-gram Generation:

function generate\_ngrams(text, n):

if n == 0:

return list of '\_' \* len(text)

remove spaces from text

return [text[i:i+n] for i in range(0, len(text) - n + 1)]

Entropy Calculation:

function calculate\_entropy(ngrams): count frequencies using a dictionary total = sum of all frequencies entropy = -sum(p \* log2(p)) for each token return entropy, total

Time Complexity:

* Preprocessing: O(N )
* N-gram generation: O(N )
* Entropy computation: O(K), where K is number of unique n-grams
* Overall per file: O(N x 11)

Correctness Justification:

* The entropy formula is correctly applied using token probabilities.
* The script handles all values of n from 0 to 10.
* Tokens are correctly counted, including support for special characters.

# 2.Commit Owners

Problem Overview:

Given a weld string (a concatenated string of employee IDs), identify which sequence of employee IDs best explains the string using a valid mapping from an employee list. The goal is to find the sequence with the most commits (i.e., the longest decomposition).

Data Structures Used:

* dict: for mapping employee IDs to names.
* memo (dictionary): to store intermediate results for optimization (memoization ).
* list: to store possible ID sequences.

Pseudocode:

Employee File Parser:

function load\_employees(filename):

for each line in file:

split line by ','

add {id: full\_line} to dictionary return dictionary

Weld Resolver:

function resolve\_weld(weld, employee\_ids, memo):

if weld is empty: return empty list if weld in memo: return memo[weld]

max\_sequence = None

for id in employee\_ids: if weld starts with id:

remainder\_sequence = resolve\_weld(weld after removing id)

if valid:

sequence = [id] + remainder if longer than max\_sequence: max\_sequence = sequence

memo[weld] = max\_sequence return max\_sequence

Time Complexity:

* Let W be the length of the weld string.
* Let M be the number of employee IDs.
* With memoization: O(W x M x L), where L is average ID length.

Correctness Justification:

* Each prefix is matched against valid employee IDs.
* Memoization ensures repeated states are not recomputed.
* Among valid sequences, the one with most commits is selected.

# 3. Bonus Tasks

Language Identification Using Bigrams:

* Construct bigram frequency models for both English and native language.
* For a new sentence: compute log-probability for each language model.
* Classify sentence as belonging to the language with higher likelihood.

Count All Valid Decompositions:

* Modify the weld resolver to count all ways the weld can be split into valid IDs using dynamic programming.

# Challenges and Notes

* Unicode and multilingual characters required extra care in preprocessing.
* Entropy values varied across genres - expected due to vocabulary diversity.
* Weld resolution was NP-hard in theory, but manageable due to memoization.

# Files Included

* entropy\_analyzer.py
* commit\_resolver.py
* data/ folder with .txt files
* employees.txt and example weld string
* This project report