

Algorithm and Complexity

Semester Project

Semester: Spring 2025 By

Leo Hysa, Sibora Sinani

**Topic: Entry of Language & Commit Owners**

11 June 2025

**Introduction**

In this assignment we worked on two algorithm problems: measuring language entropy using n-gram models and reconstructing commit history from a welded string of employee IDs.

In Part I, we measure the complexity of Albanian texts using Shannon’s entropy formula for n-grams (n = 0 to 10). Besides that, we also added the bonus language detection system based on bigram probability models of Albanian and English.

In Part II, we solved a substring decomposition problem: given a welded string representing commit author IDs, we determine the sequence of valid employee IDs that created the commits, maximizing the number of commits. This is done by using recursive backtracking.

**Implementation**

The first program reads the text files, changes all letters to lowercase and removes non-letter characters. After this it counts group of letters from size 0 to 10 and calculates their Shannon entropy to show how complex the text is. The program also includes a bonus language identifier, that builds bigram frequency models.

The second program reads employee data from a CSV file and stores it using the employee IDs as keys. After this it searches step by step to find all possible ways to split a joined string of IDs into employee sequences. In the end it picks and shows the split with the most commits and the total number of valid splits found.

**Command Line Interface**

For the first part there is no argument needed. The program reads the .txt files from src/data, calculates and prints n-gram entropies for n = 0 to 10. It also performs language identification, showing whether the text is in English or Albanian.

For the second part, the program can run with two arguments:  
A CSV file of employees and a string made by joining employees’ IDs together. The program shows the best way to split the string into employee IDs and tells you how many different splits are possible.

**Pseudo Code**

First program:

1. main(args):

2. maxSize = 10

3. dataDir = "src/data"

4. if dataDir missing or no .txt files:

5. print error and exit

6. for each file in dataDir:

7. text = readAndNormalize(file)

8. for n from 0 to maxSize:

9. if n == 0:

10. counts = countUniqueLetters(text)

11. else:

12. counts = countNGrams(text, n)

13. entropy = computeEntropy(counts)

14. print entropy, total tokens, top 5 and bottom 5 tokens

15. print "Language Identifier"

16. runLanguageIdentifier()

17. readAndNormalize(file):

18. open file with UTF-8

19. read all lines, normalize to NFC, lowercase

20. replace non-letters with space

21. collapse multiple spaces to one

22. return cleaned text

23. countUniqueLetters(text):

24. return map of each unique letter → 1

25. countNGrams(text, n):

26. lettersOnly = text without spaces

27. slide over lettersOnly to get n-grams and count frequencies

28. return counts map

29. computeEntropy(counts):

30. total = sum of counts

31. entropy = 0

32. for each count c in counts:

33. p = c / total

34. entropy -= p \* log2(p)

35. return entropy

36. runLanguageIdentifier():

37. alb = cleanText(readFile("src/data2/alb2.txt"))

38. eng = cleanText(readFile("src/data2/english.txt"))

39. inpRaw = readFile("src/data2/input.txt")

40. inp = cleanText(inpRaw)

41. albModel = buildBigramModel(alb)

42. engModel = buildBigramModel(eng)

43. albChars = uniqueChars(alb)

44. engChars = uniqueChars(eng)

45. logAlb = computeLogProbability(inp, albModel, albChars)

46. logEng = computeLogProbability(inp, engModel, engChars)

47. print input, logAlb, logEng

48. print "Likely Language: " + (Albanian if logAlb > logEng else English)

49. cleanText(text):

50. lowercase text

51. remove all chars except a-z, ë, ç

52. return cleaned text

53. buildBigramModel(text):

54. count each consecutive 2-char substring frequencies

55. return map

56. uniqueChars(text):

57. return set of unique characters in text

58. computeLogProbability(sent, model, chars):

59. if sent length < 2: return -∞

60. total = sum of model counts

61. V = (size of chars)^2

62. logp = 0

63. for each bigram in sent:

64. count = model.get(bigram) or 0

65. prob = (count + 1) / (total + V) // add-one smoothing

66. logp += log2(prob)

67. return logp

Second program:

1. main(args):

2. if args.length != 2:

3. print usage message and exit

4. fileName = args[0]

5. weld = args[1]

6. loadEmployees(fileName)

7. findCombinations(weld, 0, empty list)

8. if allCombinations is empty:

9. print "No valid decomposition found" and exit

10. best = first combination in allCombinations

11. for combo in allCombinations:

12. if combo.size > best.size:

13. best = combo

14. print "Best decomposition (most commits):"

15. for employee in best:

16. print employee.id + ": " + employee.fullName

17. print "Total valid decompositions: " + size of allCombinations

18. loadEmployees(fileName):

19. open file for reading

20. for each line in file:

21. split line by ','

22. if parts length >= 3:

23. id = parts[0], lastName = parts[1], firstName = parts[2]

24. if id not empty:

25. add Employee(id, lastName, firstName) to employeeMap

26. findCombinations(weld, index, currentList):

27. if index == length of weld:

28. add copy of currentList to allCombinations

29. return

30. for i from index+1 to length of weld:

31. substring = weld[index:i]

32. if substring in employeeMap:

33. add employeeMap[substring] to currentList

34. findCombinations(weld, i, currentList)

35. remove last employee from currentList // backtrack

**Time complexity**

Part I: NGramGameAlbanian.java

1. N-gram Entropy Calculation

For each file and for each n from 0 to 10:

It slides a window of size n across the text to count n-gram frequencies.

Let:

T = total number of characters in the text (after cleaning),

N = max n-gram size (here, 10).

Time Complexity:  
O(N × T) — For each n (up to 10), the code iterates over the entire character sequence to collect n-grams.

2. Sorting n-grams for top/bottom tokens

Sorting the frequency map takes O(K log K) where K is the number of unique n-grams.

Overall Time Complexity (Part I):  
O(N × T + K log K) per file  
(Usually K << T, unless the text is highly diverse.)

Bonus: Language Identifier

Bigram model building:

For each corpus (alb2.txt, english.txt, and input.txt), it reads and slides a window of size 2:  
O(L) per file, where L = length of cleaned text.

Log-probability computation:

For input of length M, it does O(M) work.

Overall Time Complexity (Bonus):  
O(L + M) where L is total training corpus length and M is the input length.

Part II: WeldStringGame.java

Core Recursive Function: findCombinations

This explores every possible way to split the weld string into valid substrings (IDs in the map).

Let:

n = length of the weld string,

k = number of valid employee IDs (map size).

In the worst case, where every substring is valid (e.g., all 1-digit IDs from 0–9), the number of decompositions is exponential — similar to recursive word break:

Time Complexity (Worst Case):  
O(2ⁿ)  
(Exponential — since for every index it may branch multiple ways.)

**Justification of Data Structures**

Part I:

* HashMap<String, Integer> is used to store n-gram counts because it allows fast insertion and lookup of each n-gram and its frequency.
* ArrayList and Set are used to sort the n-grams by frequency and to keep track of unique characters efficiently.
* StringBuilder is used to build and clean the text efficiently, as it is faster than using regular string concatenation.

Part II:

* HashMap<String, Employee> stores employee data with IDs as keys for quick and easy access during the decoding process.
* ArrayList<ArrayList<Employee>> keeps all valid decompositions of the welded string, allowing storage of multiple possible results.
* ArrayList<Employee> is used to track the current sequence of employees during the recursive search and backtracking.

**Proof of Correctness**

In the first part the program reads text files and normalizes them by converting all characters to lower case and getting rid of non-letter characters, ensuring consistent for n-gram analysis. The program goes through the text and collects all parts of length n, making sure it doesn't miss any n-grams. The program calculates entropy by adding probabilities using a standard formula, making sure the values are correct. The bonus part is a language identifier that builds bigram frequency models from known corpora. This process is done to decide whether the language is Albanian or English.

The second program loads employee data into a map keyed by employee ID for fast and reliable lookup. It uses a recursive findCombinations method to explore all possible ways to split the weld string by checking every substring as a potential employee ID. Backtracking makes sure all valid splits are found without repeats by checking every possibility and undoing steps when needed. Finally, the program selects the split with the most employees, which corresponds to the highest number of commits, as required. In both programs, careful handling of edge cases (empty files, no valid decompositions) and use the right data structures to make sure everything runs smoothly and correctly.  
  
**Bonus Features**

In the first part the bonus feature is a language identifier. Beside the main task of computing n-gram entropy, the program has this bonus feature that uses bigram language models to decide whether the text is Albanian or English.

In the second part the program is not only able to find one valid way to decode the welded string of the employee IDs. It tries all possible ways using recursion and backtracking, then picks the best one with the most commits.

**Division of Work**

|  |  |  |
| --- | --- | --- |
| Task | Leo | Sibora |
| Designing algorithm & logic | ✓ | ✓ |
| Java implementation (coding) | ✓ | ✓ |
| Testing and debugging | ✓ | ✓ |
| Writing CLI and file input | ✓ | ✓ |
| Report writing and formatting | ✓ | ✓ |
| Time complexity and pseudocode | ✓ | ✓ |