COSC 1285

Algorithms and Analysis

Assignment 2

Design and Implementation Report for Task C and Task D

Semester 2/2020

Qianrui LONG

S3873326

Contents

[1 Introduction 3](#_Toc53954133)

[2 Task C and D Task Specification and Implementation Objectives 3](#_Toc53954134)

[3 Comparison of Final Approach of Task C and D with Alternative Strategies 3](#_Toc53954135)

[3.1 Theoretical Comparison of Design Ideas of Hangman Strategies 3](#_Toc53954136)

[3.2 Empirical Comparison of Hangman Strategies (Stage 1 Implementations) 4](#_Toc53954137)

[4 Task C and D Final Approach 5](#_Toc53954138)

[4.1 Further Analysis of Performance Metrics 5](#_Toc53954139)

[4.2 Discussion on Findings of Empirical Analysis of Stage 1 Implementations 5](#_Toc53954140)

[4.3 Implications for Algorithm Design Choices for Task C and Task D 6](#_Toc53954141)

[4.4 Task C Requirements (Points of Differentiation from Task B) 6](#_Toc53954142)

[4.5 Task D Requirements (Points of Differentiation from Task C) 6](#_Toc53954143)

[4.6 Task C and D Final Approach and Design Idea Explanation (Step by Step Breakdown) 6](#_Toc53954144)

[Appendix A – Analysis of Provided Sample Dictionary 7](#_Toc53954145)

[A.1 Analysis of Word Length Distribution (Table A.1) 7](#_Toc53954146)

[A.2 Calculation of % Words with Word Length in Range 8 to 12 (inclusive) 7](#_Toc53954147)

[Appendix B – Analysis of Performance of Various Approaches 8](#_Toc53954148)

[B.1 [Random Approach] Trial Output 8](#_Toc53954149)

[B.2 [Naïve Approach] Trial Output 8](#_Toc53954150)

[B.3 [Highest Frequency Approach] Trial Output 8](#_Toc53954151)

[B.4 [2-Partition Approach] Trial Output 8](#_Toc53954152)

[References 9](#_Toc53954153)

# 1 Introduction

This report will focus on explaining the implemented approaches to Task C and Task D of the hangman solver assignment. Specifically, this report sets out with the objective of outlining: (i) rationale/strategy behind final chosen approach, (ii) justification for final approach and (iii) advantages of selected approach over previous/alternative approaches.

# 2 Task C and D Task Specification and Implementation Objectives (Choice Rationale)

|  |  |
| --- | --- |
| **Hangman Specification** | Hangman rules as follows:  [i] Guessing takes place in a character by character manner (i.e. not possible to guess a whole word)  [ii] Correct guess of a character results in all positions of the character in the words to be revealed (in the program this is an arraylist of arraylist lPositions)  [iii] Incorrect guess of a character results in an increase in the tally of incorrect guesses  [iv] If all characters in the word are guessed before the incorrect guesses count reaches the specified “max incorrect guesses” then the game is won, else if the number of incorrect guesses surpasses the “max incorrect guesses” then the game is lost |
| **Performance Metrics** | [i] **Maximise win/loss ratio** over multiple trials  [ii] **Minimise total guesses** until win taken in each trial |
| **Task C | Two Word Hangman** | Hangman variation with **two words**. |
| **Task D | Wheel of Fortune** | Hangman variation with **multiple words** with correct grammatical syntax (i.e. phrases). |
| **Implementation Objective** | [i] **Minimise** number of **incorrect guesses** (as they increase incorrect guess counts to be closer to “max incorrect guesses” which once reached causes a loss, thus incorrect guesses are punished), minimise total number of guesses before win  [ii] **Guess all characters** in the word(s), increase win/loss ratio |
| **Additional Notes** | As different strategy may exhibit different behaviour (advantage/disadvantage over other strategy) based on length of word guessed, it is helpful to understand the distribution of word length of dictionary entries in the sample dictionary ausDict (which the class has been informed to be representative of test dictionaries).    *(Graph 1.1: Distribution of Word Length of Words in ausDict, source table A.1 in Appendix A)*  The following conclusions can be drawn from statistics within Graph 1.1:  [i] Range of word length (for ausDict) spans from a minimum of 3 characters to maximum of 24 characters, thus one should expect words of various lengths to be in dictionary  [ii] Majority of words have length of around 10 characters (in fact ausDict has mode of word length 10), with 60% of words in ausDict having word length between 8 and 12 characters. Thus, strategies that perform well for words of length 10  As a result, the report will analyse the effectiveness of different approaches for word length of 3 (representative of short words), 10 (representative of mid-length words) and 21 (representative of long words). Additionally, selection of strategy considers the performance of strategy with mid-sized word length with greater weighting. |

# 3 Comparison of Final Approach of Task C and D with Alternative Strategies

## 3.1 Theoretical Comparison of Design Ideas of Hangman Strategies

|  |  |
| --- | --- |
| **Strategy** | **Design Idea and Algorithm Characteristics** |
| **Random**  (Stage 1 impl.) | * Randomly generates character to guess in each turn * Does not make use of dictionary (non-dictionary aware) * Does not make use of feedback |
| **Naïve**  (Stage 1 impl.) | * Guesses character by frequency in English language. In order: e, t, a, o, i, n, s, r, h, d, l, u, c, m, f, y, w, g, p, b, v, k, ‘, x, q, j, z (Cornell Department of Mathematics, 2003). Note: frequency of apostrophe ( ‘ character) as per Google N-grams was sourced from Cook, V.J. (2013) in ‘Standard punctuation and the punctuation of the street’. * Does not make use of dictionary (non-dictionary aware) * Does not make use of feedback |
| **Highest Frequency**  **(Final Approach, Stage 1 Impl.)** | * Guess character that appears in the greatest number of words (up to date list of plausible words per turn) in each turn * **This strategy prioritises “success” per turn** * Deterministic approach outlined for Task B, compute statistics per letter of how many words the letter appears in then given feedback (information on positions of character or hit/miss) to eliminate entries that are no longer plausible, then updates statistics * Makes use of known dictionary that words to guess are drawn from * Makes use of feedback, filters according to word lengths and positions of hits / misses |
| **2-Partition**  (Inspired by binary search, Stage 1 impl.) | * Guess character that is in closest to 50% of the words (updated per turn taking feedback into account), this char is closest to splitting list of plausible words in half thereby reduce the number of plausible options by half (on average half) each turn Parallels can be drawn to **decrease and conquer, decrease by a constant factor** binary search algorithms where the initial problem size n is reduced by a factor of half and iterated until solution of sub-problem (and also solution of original problem) is found. However, this 2-partition strategy (partitioned into has char and does not have char) is not strictly decrease by a constant factor as char being guessed may sometimes not be in precisely 50% of remaining plausible words. * **This strategy prioritises “information gain”** per turn and reduces the size of the sub problem to be solved for later turns * Makes use of known dictionary and feedback (filters according to word lengths and positions of hits / misses) |
| **N-Partition**  (Information Entropy with Look Ahead, Stage 2 impl.) | * Guess character that partitions the list of plausible words with proportions that have the highest Shannon’s entropy (without look ahead). Guess character (that combined with next guess character) partitions the list of plausible words with proportions that have the highest information entropy. For instance, in the case where character “e” is the first letter being guessed for a for letter words, and the list of plausible words at this stage contains “feet”, “neat”, “cake” and “bank”. This would generate the following partitions:   \_ e e \_ where feet is a possible word to be guessed  \_ e \_ \_ where nest is a possible word to be guessed  \_ \_ \_ e where cake is a possible word to be guessed  \_ \_ \_ \_ where bank is a possible word to be guessed  Where information entropy is    = sum(i) – Pi\*log2(1/Pi), where Pi is the proportion of the partition I to the size of the whole unpartitioned list (Conley, 2014)  Whilst an N-partition strategy without look ahead is a **greedy algorithm**, picking a character that maximises information gain at each iteration it could get stuck in a local maxima. For instance consider four guesses: (i) Guess A splits the list of plausible words into equal thirds and entropy(A) = 3\* -(1/3)log3(1/3) = 1.6, (ii) Guess B splits the words into equal thirds (as per A), (iii) Guess C splits the words into equal halves entropy(C) = 2 \* -(1/2)log2(1/2) = 1, (iv) Guess D also splits the words into equal thirds, entropy(D) = entropy(D). In this case if there was no look ahead implemented, the optimal solution would be suggested to be A or B. However, with 1 look ahead it would find the following probabilities, Guess(A then B) = 1.6 and Guess(C and D) = 2. Thus, guessing either C or D would allow the most information gain over the two guesses.   * **This strategy prioritises information gain per turn**, makes use of known dictionary and feedback (filters according to word lengths and positions of hits / misses). |
| **NLP Approach** \*\*(Language Model + Hidden Markov Model Stage 2 impl.) | * Trained hidden Markov model tag partially revealed words with word sequence that maximise language model probability and word frequencies. Guess letter that appears in most words. * Steps for implementation (Chang, 2012) : (1) train language model on proportion of large text (such as Australian Body Corpus), to define the probability of a sequence of words using probability distribution. (2)Implement hidden Markov model where the hidden states are the English words in the phrases to be guessed e.g., she had a red dog, and the observations are the half revealed sequences of words, e.g \_he ha\_ a \_e\_ d \_ \_. The trained language model is used to calculated the transition state probability (such as P(dog|red) in an order-1 language model. Defining normalised probability of all English words that match the partially-revealed pattern as emission probability and zero otherwise. Viterbi dynamic programming algorithm can then be used to calculate the optimal state sequence. * This strategy is heavily hindered by the inability to guess whole words. **It prioritises success**, makes use of known dictionary and feedback |

## 3.2 Empirical Comparison of Hangman Strategies (Stage 1 Implementations)

Table 3.2.1 Empirical analysis of strategies using ausDict.txt (supplied) as test input

|  |  |  |
| --- | --- | --- |
| **Strategy** | **Avg. Incorrect Guesses (until solved)**  **[to 2 decimal places]** | **Avg. Total Guesses**  **[to 2 decimal places]** |
| Random | 16.46 | 25.87 |
| Naïve | 9.58 | 18.99 |
| Highest Frequency | 0.45 | 9.86 |
| 2-Partition | 0.45 | 9.86 |

Table 3.2.2 Empirical analysis of strategies’ effectiveness for different word lengths, using ausDict.txt as test input

*Note: Output data tables for these trials can be found in Appendix B.1 to B.4*

|  |  |
| --- | --- |
| *Graph 3.2.1: Distribution of number of incorrect guesses, taken by implementations with various strategies/approaches, until solved for small words (word length 3)* | *Graph 3.2.2: Distribution of number of incorrect guesses, taken by implementations with various strategies/approaches, until solved for mid-length words (word length 10)* |
| *Graph 3.2.3: Distribution of number of incorrect guesses, taken by implementations with various strategies/approaches, until solved for long words (word length 21)* | **Observations and Inferences [Stage 1 Empirical Trials]**  > Highest freq. and 2-partition outperformed random and naïve strategies for small, mid-length and long words, having a higher percentage of the words being solved with number of incorrect guesses lower than random/naïve  > **Further empirical analysis will focus on highest frequency and 2-partition**  > Highest freq. and 2-partition performance (minimal incorrect guesses before solved) improved with longer words: (1) Approx. 15% of words were solved with 0 incorrect guesses for words of length 3. (2) Approx. 55% of words were solved with 0 incorrect guesses for words of length 10 and (3) 100% of words were solved with 0 incorrect for words of length 21  > For majority of words (recalling that most words were of mid length), highest freq. and 2-partition can solve with less than 2 incorrect guesses (from Graph 5.2.2 approx. 90% of the words of length 10 were solved with less than 2 incorrect guesses  > Highest freq. had similar (slightly better) performance than 2-partition |

Table 3.2.3 Further Empirical Analysis with Larger Test Input (300,000+ entries), Single Word

*Note: As a result of inferences made from data in table 5.2.2, further empirical analysis was only conducted for* ***highest frequency*** *and* ***2-parition*** *approaches.*

|  |  |  |
| --- | --- | --- |
| **Strategy** | **Avg. Incorrect Guesses (until solved)**  **[to 2 decimal places]** | **Avg. Total Guesses**  **[to 2 decimal places]** |
| Highest Frequency | 2.29 | 10.38 |
| 2-Partition | 2.32 | 10.42 |

Table 3.2.4 Further Empirical Analysis with Larger Test Input (300,000+ entries), Multiple Words

*Note: Data is drawn from empirical experiments where 15,000+ trials with randomly generated “words to guess” (set of 2)*

|  |  |  |  |
| --- | --- | --- | --- |
| **Strategy** | **Avg. Incorrect Guesses (until solved)**  **[to 2 decimal places]** | **Avg. Total Guesses**  **[to 2 decimal places]** | **Avg. Sum of Word Lengths (of the words to be guessed)** |
| Highest Frequency | 1.53 | 13.88 | 17.37 |
| 2-Partition | 1.54 | 13.89 | 17.35 |

# **[Summary of Section 3 Empirical Data] Highest frequency strategy-based implementation outperforms all other Stage 1 implementations.**

# 4 Task C and D Final Approach

## 4.1 Further Analysis of Performance Metrics

There are a few key factors to consider in regards to the performance metrics:

(i) Both performance metrics (Sect. 2) are to be optimised by **minimising incorrect guesses** before completely guessed (in this report “total incorrect guesses” refers to the total number of incorrect guesses taken, where the solver is allowed as many guesses as the number of guessable characters). Win/loss is essentially a comparison between the integer maxIncorrectGuesses and a word’s “total incorrect guesses” (if the former is larger it’s a win, otherwise a loss), whilst total guesses taken is essentially:

(ii) This is why empirical analysis provided the seemingly surprising result that guessing highest frequency is more effective than 2-partition (a decrease and conquer strategy that at each stage moves to a much smaller sub-problem). It seemed natural to presume that a strategy which would rapidly narrow down possible solutions would be the one that: (a) took fewer total guesses, (b) optimise win/loss. However, this is largely **not** the case in the context of this task due to the inability to guess whole words (if they were allowed, quickly narrowing down possible word(s) can greatly reduce total incorrect guesses and total guesses taken, as once the sole plausible solution is clear the whole word/phrase can be guessed and no further counts of incorrect guesses / general guesses accumulated) .

Out of the factors that influence the key performance metrics, **minimising incorrect guesses** is the **only** one that strategies can optimise for. Thus, strategies that **maximise success in a turn** (guessed character is in word(s)) are ideal.

## 4.2 Discussion on Findings of Empirical Analysis of Stage 1 Implementations

The aforementioned hypothesis (in 4.1) is validated in empirical testing of stage 1 implementations: (i) Highest performing strategy was “highest frequency” strategy (the only one that optimised for success per round), (ii) Strategy that optimised for information gain, 2-partition, did well but not as well as “highest frequency”.

## 4.3 Implications for Algorithm Design Choices for Task C and Task D

Due to 4.2 (ii), N-partition was no longer relevant to finding the optimal strategy as it also prioritised information gain. Similarly, due to the inability to guess whole words (if they were allowed, quickly narrowing down possible word(s) can greatly reduce total incorrect guesses and total guesses taken, as once the sole plausible solution is clear the whole word/phrase can be guessed and no further counts of incorrect guesses / general guesses accumulated) it was decided to not consider the NLP strategy (for Task D in particular) as it would provide marginal gains, i.e. even when the NLP model predicts the sequence of words (the phrase) it still has to guess each character one by one (increasing total guess count nonetheless), additionally the HMM (hidden Markov model) will not be effective unless if a hybrid approach of first guessing characters with highest frequency was adopted and more information was thus made available in partially revealed word sequence.

## 4.4 Task C Requirements (Points of Differentiation from Task B)

**Possibility that character is present in one word but not the other:**

i. Although bGuess will be true for this case, one of the words should actually be handled as guessed char is not in word

ii. [IMPLEMENTATED SOLUTION] Utilise a combination of bGuess and lPositions to determined if guessed char are in each word respectively and pass Boolean isInWord to helper method for further handling (filter by position of guessed char etc.)

**Separate sets of remaining plausible words for word 1 and word 2:**

i. As word 1 and word 2 can be distinctly different words, separate sets of plausible words for word 1 and word 2 should be initialised and maintained (updated each round of guessing), implemented as a freqMap for each word to be guessed, word i.

ii. [IMPLEMENTED SOLUTION] freqArrays is an array of arrays where freqArrays[i] is array of frequencies of each character in plausible words for word i is initialised and maintained. Additionally, freqArraySum, an array of character frequency across all words is updated each round of guessing (this is used to determined character with highest frequency each round)

## 4.5 Task D Requirements (Points of Differentiation from Task C)

**Phrases (sequence of words) that make grammatical sense are used:**

In general, this means that there are more unique characters in the words to be guessed, thus a greater chance of success in a round (i.e. guessing highest frequency letter will be more likely to succeed) and better performance.

**Only alphabetical characters are guessable characters for this task:**

Smaller set of guessable characters thus increases the chance of success in a round and thus better performance.

**Hence,** it was found that the performance metrics of using highest frequency strategy for D were even better than for C. This paired with the reasoning in 4.3 regarding NLP strategies being hindered by the inability to guess words (rather than character by character) was the rationale for still implementing the highest frequency strategy for Task D. **Furthermore,** data generation of a mock NLP strategy (which start with highest frequency to attain partially revealed word sequence and then HMM until sequence of words predicted with significant confidence – emulated to be 2 to 3 guesses, which finally guessed characters in phrase in a character by character fashion) found no significant advantage.

## 4.6 Task C and D Final Approach and Design Idea Explanation (Step by Step Breakdown)

1) Initialise global variables and global constants

2) [newGame] For each word to be guessed, initialise variables

i.[filterByLengthInitialise] For each word to be guessed a separate freqMap (which stores key-value pairs for each plausible word, with the plausible word as a string and with a corresponding integer array of the character presence, i.e. plausible word “a” would have int array [1, 0 ,0…0]) of plausible words for that word to be guessed is maintained. Thus, the implementation first initialises freqMap for each word to be guesses, adding key-value entries for each word in the dictionary that is of the right length (effectively filtering by length) and afterwards [initialiseFreqArraySum] a sum tally of all individual freqArray (which exist for each word to be guessed), representing the frequency that each character appears in a word across all words to be guessed. At this stage our implementation **reduces the size of the problem** by filtering out entries with incompatible word lengths, also ensuring a greater relevancy of calculated frequency statistics.

3) [makeGuess] utilises helper method [findMaxIndexInArray] which traverses freqArray, an array which stores the numbers of words that each character appears in to find the index of the character that appears in the most words (that are still plausible). This is an implementation of the highest frequency strategy that prioritises success at each round of the game (thereby minimising chance of incorrect guess in a round, and minimising total incorrect guesses).

4) [guessFeedback] utilises helper method [updateFreq] which updates frequency of characters in word in freqArray:

i. Words that have character when it’s a miss, or don’t have character when it’s a hit have the frequency of all letters in the word decremented from overall tally (freqArray)

ii. Words that **have the letter that is a correct guess** are to be further filtered by position with aid of [arePlausiblePositions]

This part of our strategy **reduces the size of the problem** and ensures a greater relevancy of statistics (words that are not plausible no longer contribute to frequency statistics, and statistics become a closer representation of frequency of characters in word to be guessed)

5) Steps 3 to 4 are repeated until all characters in words to be guessed are guessed

# Appendix A – Analysis of Provided Sample Dictionary

## A.1 Analysis of Word Length Distribution (Table A.1)

|  |  |
| --- | --- |
| **Word Length** | **Number of Entries** |
| 3 | 18 |
| 4 | 74 |
| 5 | 181 |
| 6 | 319 |
| 7 | 505 |
| 8 | 851 |
| 9 | 1417 |
| 10 | 1874 |
| 11 | 1849 |
| 12 | 1531 |
| 13 | 1232 |
| 14 | 971 |
| 15 | 726 |
| 16 | 436 |
| 17 | 295 |
| 18 | 130 |
| 19 | 88 |
| 20 | 45 |
| 21 | 24 |
| 22 | 9 |
| 23 | 2 |
| 24 | 1 |
| **Total # words** | 12578 |

Note: Conditional formatting applied (darker green indicating greater concentration of words with corresponding word length, closer to white indicating lower concentration words with corresponding word length).

## A.2 Calculation of % Words with Word Length in Range 8 to 12 (inclusive)

# Appendix B – Analysis of Performance of Various Approaches

## B.1 [Random Approach] Trial Output



## B.2 [Naïve Approach] Trial Output



## B.3 [Highest Frequency Approach] Trial Output



## B.4 [2-Partition Approach] Trial Output



# References

Chang, J., 2012. A Hangman Game Solver Based On Language Models. [online] Joseph Chee Chang. Available at: <https://joseph.nlpweb.org/hangman+&cd=1&hl=en&ct=clnk&gl=au> [Accessed 18 October 2020].

Conley, C., 2014. The Game Of Hangman, Decision Theory And Delivering Software - Redsnake Philly 2014. [video] Available at: <https://www.youtube.com/watch?v=dSSNLyaZBUU> [Accessed 18 October 2020].

Cook, V., 2013. Standard Punctuation and the Punctuation of the Street. *Second Language Learning and Teaching*, pp.267-290.

Cornell Department of Mathematics, 2003. *Frequency Table*. [online] Cornell Department of Mathematics. Available at: <http://pi.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.html> [Accessed 15 October 2020].