ALGORITHMS AND ANALYSIS ASSIGNEMENT 2 REPORT

Solving Hangman

Sriram Senthilnathan(S3812810) & Yogeshwar Girish Chaudhari(S3828116) – Group 20

OCTOBER,2020

Task C: Dictionary-Aware Guessing strategy for Two Word Hangman

Solving Approach and Rationale:

We have used the strategy that is implemented for Task-B – Dictionary Aware and extended it with the following optimization strategy (described below).

Basic Strategy: The basic strategy works as follows,

1. Find the length of the word to be guessed and create a sample set by filtering out all the words of matching length from the provided dictionary.
2. Compute the frequency count of each individual unique characters from all the words in the sample set.
   For example, consider the dictionary have the words "colonel, cook, zack". The frequency counts will be calculated as follows

Character	Frequency (out of 3 words)
c	3
o	2
l	1
n	2
e	1
k	2
z	1
a	1

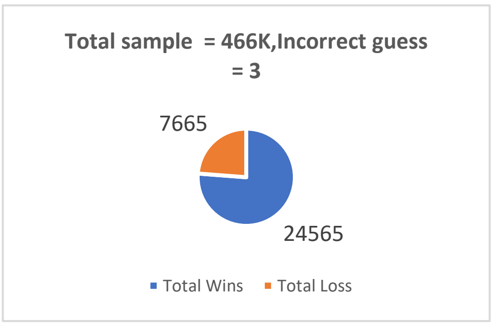
   From the table , it can be seen that count of 'o' is only '2' even though it occurs for a total of 4 times (in colonel & cook).That is because we compute the frequency of a letter 'o' in word "colonel" as 1 though it appears twice in that word. It is because we want to find only number of words a character appears in than the number of times it occurs. So, the frequency count of a single char in single word possibilities would be either '0' (not present) or '1' (present). Likewise we find the count of that single character in all the words in sample set and add them together to find the total frequency count of that character and same is followed for all possible individual characters in the sample set(including special characters). In this case
3. Once the frequency counts are computed, we pick the letter with the highest frequency value and make a guess with that letter. After this guessing, there are two possible cases,
   Case 1: If it is a right guess(Hit), our sample set is reduced by filtering only the words that contains guessed character in the identified position. So, when the guessed character appears more than once in the word, then there is chance of getting greatly reduced sample set.
   Case 2: If it is a wrong guess(Miss), we reduce our sample set to have only the words that does not contain the guessed character.
4. The frequency count of all the unique characters in the reduced sample set is recomputed(excluding already guessed letter) and then make a guess with the character that has highest frequency count. If two characters has same frequency value, then we pick the latter because we identified that this increasing winning chances (from our experimental analysis).
5. The same process repeats until the word is correctly guessed or the maximum number of incorrect guess limit is reached.

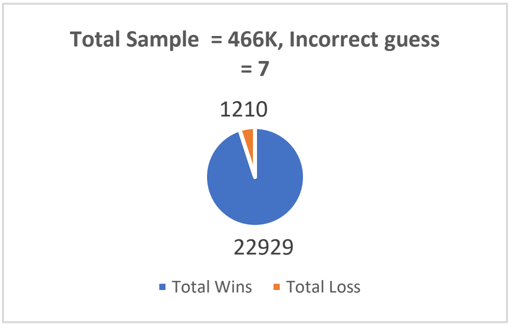
Extended Optimization strategy to implement Two – Word Hangman: In general, when a word has more characters i.e. lengthy word, then the probability of guessing it correctly is higher than guessing a small word.

- In this strategy , we maintain a sample set for each word separately, because the frequency count of the smaller word should not influence the frequency count of bigger word and vice versa. For words of different length, the probability of correctly guessing the longer word is more than guessing smaller words. So, we start with guessing longer word.
- Then we follow the same strategy as descried above in the basic concept, but the difference is the sample set is reduced for both the words whenever a guess is made, irrespective of the word being guessed.
- Even though we are making guesses for longer word, we are also checking the size of the sample set (i.e. total number of words remains) of the smaller word and if that size is reduced to '1' (contains only one word) then it means that our algorithm found the smaller word completely. So, in this case we switch from guessing longer word to smaller word, so we get "Hit" for the smaller word.
- At the same time, as we are reducing both sample sets simultaneously whenever a guess is made, the sample set of longer word is reduced when we make guesses to smaller word.
- Now we just need to find the remaining unfound characters in the longer word using the greatly reduced sample set.

Experimentation Results:

Total number of samples	Max. No of Incorrect guesses	Total Runs	Total Wins	Total Loss
466,000	3	32230	24565	7665
466,000	7	24139	22929	1210
12,539	3	46092	45845	247
12,539	7	46	12539	0





Task D: Wheel of fortune variant

Solving Approach and Rationale:

Wheel of fortune is extended version of Two-Word hangman, but it differs in following aspects,

- Instead of guessing a word, we need to guess a phrase or sentence, so the number of words could be greater than 2.
- In two-word hangman, each word can be independent of the other i.e. there can be no lexical meaning, but in wheel of fortune the words follows the semantics to achieve the lexical meaning, hence they are dependent on each other.

Implemented Approach/Strategy:

1. Create sample sets for each word in the sentence or phrase by filtering out the words of length matching the length of word to be guessed.
2. Identify and select the longest word from the given sentence or phrase and start making guesses to that word.
3. To make guesses, we follow the same method as in Dictionary-aware strategy and Two-word hangman variant, i.e. by computing the characters frequency count and making guess with the character having highest frequency value.
4. Then based on either "Miss" or "Hit" sample set of the word being guessed and all the other words as well are reduced accordingly.
5. Whenever a guess is made , the sample set of each unsolved word in the sentence is validated to check if it has reached the size of 1 (which means that word is completely found as there will be only one word left in sample set, so every guess will be a hit).
6. If the sample set size is reduced to 1 for any word in the sentence , then our algorithm switches to that word and start making guesses(the unguessed characters from the word left in its sample set) to "Hit" that word completely.
7. Simultaneously, the sample sets of other words are also reduced whenever a guess is made(based on if that contains the guessed character or not).
8. So, in making every guess, our strategy greatly reduces the sample sets very efficiently, so the probability of finding the word to be guessed with considerably minimum number of guess is increased.
9. Once a word is completely guessed, we switch back to next longest word from the unsolved words and follow the same strategy for the other words in the sentence.
10. This process happen until the sentence is completely found or the maximum number of incorrect guess limit is reached.

Alternative Approach:

We have also come up with another approach to implement Wheel of fortune variant using the n-gram model, which is a type of probabilistic language model. It helps in predicting the next word in the sentence.

n-gram is the possible set of sequence of words in the given sentence. ''n" represents the number of words. When n = 1, it is called unigram, n = 2 is called bigram, n=3 is trigram.

Consider the sentence "*The book is under the table*".

The possible bigram in the above sentence are,

The book
Book is
is under
under the
The table

We can also consider "blank spaces" as one gram like "The " can also be a bigram, that depends on our requirement.

We can use the n-gram libraries that are already available to use for looking up/searching.

- We can implement a hybrid model, by combining the strategies from previous approach with the use of n-gram libraries.
- First, we will remove all the anagrams that contains words which are not in our dictionary.
- So, if we want to find 3 words that occurs in a sequence, then we will look up on n-gram library and filter out all the 3-grams with length matching with length of the words that we need to guess and take that as our sample set.

  For example, in "The book is under the table", "The Book is" is one 3-gram. Based on this ,all possible 3-grams (of length 3,4,2) will be filtered out.
- If any of the word is completely found, we can apply one more filter condition to get only the 3-grams that contains the completely guessed word(also considering its position) and positions of neighbours.
- Now two words are remaining to be guessed. So, choose the one with highest length and compute the frequencies of letters in words in the reduced 3-gram sample set, that are in the same position as the guessed word to be.

- For example, the 3 words are "under the table" and you have already found "the" then we will switch to "under" which is in the first position. Se will lookup and compute frequencies of unique characters in all the words that are in the first position of the 3-gram library. So, this makes the search more efficient.

- Every time a guess is made, there is a considerably great reduction in sample set(n-gram library).

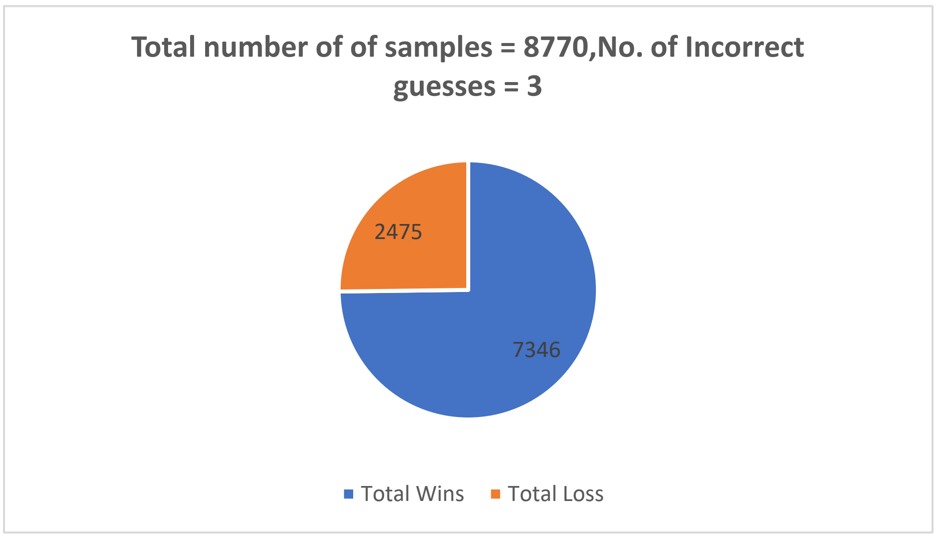
- This can continued be further until all 3 words are found or we reached the number of incorrect guesses limit.

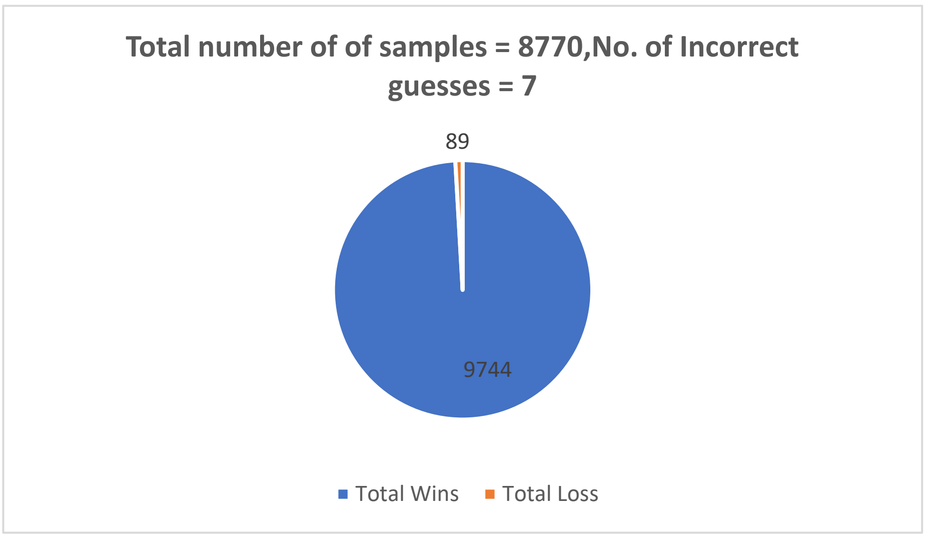
Justification on not implementing the Alternative approach:

- The operation to eliminate the n-grams which contains the words that are not in our dictionary is a very costly step, because we need to compare each word from our dictionary with every n-gram in the library, which is massive in terms of time complexity.
- We cannot be sure that every possible combination of words from our dictionary, to form a phrase or sentence is included in the n-gram library (while constructing it).

Experimental Results:

Total number of samples	Max. No of Incorrect guesses	Total Runs	Total Wins	Total Loss
8,770	**3**	**9821**	**7346**	**2475**
8,770	**7**	**9833**	**9744**	**89**





Conclusion:

The approach we have used to implement the Wheel of fortune has performed not so bad when the number of allowed guesses is '3'. Even it has achieved a very good win/loss ratio when the number of incorrect guesses is increased to '7'. So, we conclude that this implementation is still considerably efficient.

References:

[1]. Anon (2020) Gutenberg.org. Available at: https://www.gutenberg.org/files/18362/18362.txt
(Accessed: 18 October 2020).

[2]. dwyl/english-words. Available at: https://github.com/dwyl/english-words/blob/master/words.txt
(Accessed: 18 October 2020).

[3]. n-gram probabilistic model. Available at: https://en.wikipedia.org/wiki/N-gram
(Accessed: 18 October 2020)