Implementation of Backtracking Algorithm and Regex on Word Shuffle Game

Kevin Sendjaja / 13517023
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517023@std.stei.itb.ac.id
kevin.sendjaja@gmail.com

***Abstract*—Word Shuffle is a game where several letters are given and the player must take some or all the letters and place them in a specific order in order to create a word with correct spelling. The main goal of this game is to create as many correctly spelled words as possible. This paper will cover about the concept of backtracking algorithm as well as regular expressions and its implementation in solving the Word Shuffle game.**

Keywords*— *Word Shuffle, Backtracking, Regular Expression

I. Introduction

Games are a form of play usually done for the purpose of enjoyment and relaxation. Apart from those, people also play games to obtain a sense of achievement or reward, whether from playing alone, with friends, or against other people. The three major components that compose a game are rules, challenge, and interaction. Games provide either physical or mental stimulation, or both at the same time, which in turn helps in developing the players' practical skill and perform an educational, psychological, or simulational role.

One example of such games is Word Shuffle. Word Shuffle is a game where the player must arrange letters into words with a specific length and correct spelling, in order to finish the game. The higher the difficulty, more letters will be given to the player, resulting in more time needed to check whether each word is correctly spelled or not.

Through this paper, the author would like to explain the concept of Backtracking algorithm, as well as Regular Expressions, and their implementation in solving the Word Shuffle game.

II. BACKTRACKING ALGORITHM

A. Definition

Backtracking algorithm is an algorithm commonly used to find some or all possible solutions for a computational problem. Backtracking algorithm involves building candidates for solutions incrementally, saving each solution that satisfies the problem, while abandoning each partial candidates as soon as it determines that the candidate cannot possibly lead to a satisfying solution, hence called backtracking. The idea of backtracking is building the solution step by step, using recursive method, and as soon as the candidate is found to be unable to procure a solution, we stop processing that solution and return to the step before and compute the next possible candidate. As such, backtracking is considered as an improvement over the exhaustive search, which method involves building every potential solution possible, without pruning candidates that would not lead to a possible solution.

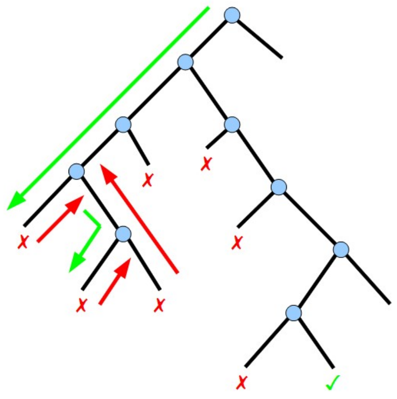


Figure 1: A representation of Backtracking algorithm

(Source: https://www.w3.org/2011/Talks/01-14-steven-phenotype/backtracking.png)

Backtracking algorithm concept was first introduced in 1950, by D. H. Lehmer, an American mathematician. Backtracking can be seen as a phase of the graph tracing algorithm Depth First Search (DFS). Three major components which compose the Backtracking algorithm are solution space, expand function, and bounding function.

1. Solution Space

   Solution space contains the possible solution for the problem. It may contain one or all possible solutions. Solution space is represented as a vector containing n-tuple: $X = (x_1, x_2, ..., x_n)$, $x_i \in S_i$.

2. Expand Function

   Expand function serves to expand the candidate solutions that becomes potential candidates for the solution space. Expand function is represented as the predicate *T(k)*.

3. Bounding Function

Bounding function is used to prune the candidates that would not lead to a possible solution. Bounding function is represented as the predicate $B(x_1, x_2, ..., x_k)$. B would give the value true should $(x_1, x_2, ..., x_k)$ leads toward a possible solution, which will cause the expand function to expand a value for x_{k+1}. Otherwise, $(x_1, x_2, ..., x_k)$ would be discarded, as it would not lead to a potential solution.

Solution space can also be represented in a tree structure, where each node represent a state of the solution and the branch represent a value for x_i. The path taken from the root node to a leaf node is a possible solution. The solution space consists of all the paths from the root to the leaves, and the tree representing the solution space can be referred as a state space tree.

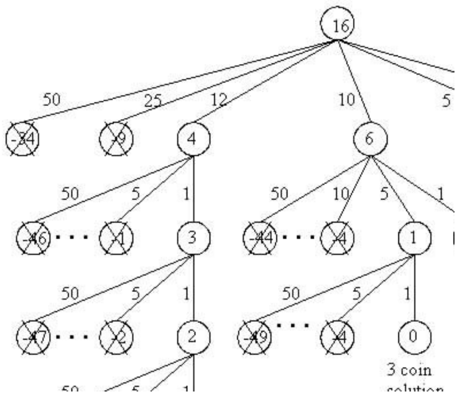


Figure 2: An example of a state space tree

(Source: http://www.cs.uni.edu/~fienup/cs153f03/lectures/lec31_12-9-03.htm)

The process of expanding the nodes follows the DFS rule, which is expanding the leftmost node until it reaches a leaf node. Nodes created from an expansion process are called live nodes, and a live nodes that are in the process of being expanded is called expand nodes. The path becomes longer as more nodes are expanded. Eventually, should the node being expanded has no chance of forming a possible solution, which is determined using the bounding factor, the node will pruned and becomes a dead node, which will never be expanded again.

B. *Advantages*

Backtracking algorithm is effective and efficient for solving problems with constraint. The global optimal solution is guaranteed to be found using the backtracking algorithm, compared to other algorithms such as Greedy algorithm, which may only get the local optimal solution. Backtracking is also relatively easy to implement and its' accuracy is granted.

C. *Disadvantages*

While backtracking is efficient in solving constraint related problems, it is not as effective when handling strategic problems, as the overall runtime of backtracking algorithm is quite slow. To solve a problem with a large amount of data, backtracking would also require a large amount of memory space, while other algorithms such as Branch and Bound might be proven to be more efficient.

III. REGULAR EXPRESSIONS

A. *Definition*

Regular expressions or regex for short, are series of characters that define a search pattern. They are most commonly used in string matching algorithms to find and/or replace a sequence of string within a series of string that satisfies the pattern. The concept of regular expressions was brought by the American mathematician Stephen Cole Kleene, back in the year 1951, and different syntaxes for writing regular expressions have been developed since then. In formal language theory, regular expressions are used to describe regular languages.

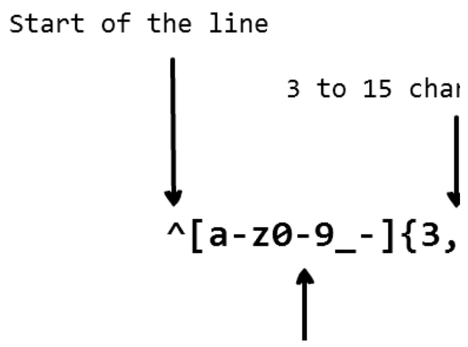
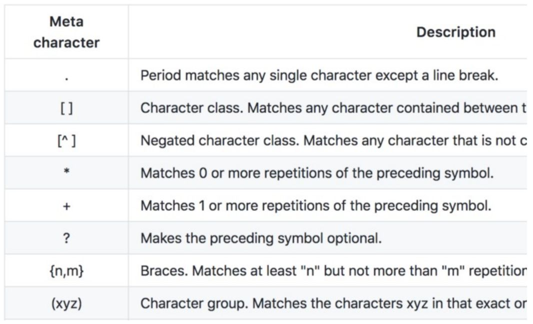


Figure 3: An example of a regular expression pattern

(Source: https://medium.com/tech-tajawal/regular-expressions-the-last-guide-6800283ac034)

The most basic regular expressions are usually comprised of the sequence of letters that is used to search that specific pattern in a text. For example, the regex 'high' simply means the letter 'h', followed by the letter 'i', followed by the letter 'g', followed by the letter 'h'. A key component in forming a regular expressions is a meta character. Unlike regular letters, meta characters have different interpretation and is treated as a replacement for one or a series of letters. Some meta characters can be used by itself, while others need to be placed within square brackets so that it may define something.



Meta character	Description
.	Period matches any single character except a line break.
[ ]	Character class. Matches any character contained between t
[^]	Negated character class. Matches any character that is not c
*	Matches 0 or more repetitions of the preceding symbol.
+	Matches 1 or more repetitions of the preceding symbol.
?	Makes the preceding symbol optional.
{n,m}	Braces. Matches at least "n" but not more than "m" repetitior
(xyz)	Character group. Matches the characters xyz in that exact or

Figure 4: Meta characters and their description

(Source: https://medium.com/tech-tajawal/regular-expressions-the-last-guide-6800283ac034)

Other than meta characters, there are also shorthand character sets, which is represented by the character backslash (\) followed by a specific letter, which is used to determine whether the character satisfies a certain category or not. There are also lookarounds, which are used to get a string which is preceded or followed by a certain pattern.

Regular expressions are usually applied in word processing application, and are used for tasks such as syntax highlighting. Other usage include parsing, data validation, and data scraping.

B. Advantages

Regular expressions are considered to be very flexible. A single line of code using regular expression could process a text without much problem. As such, it is considered to be simpler than calculating indexes and processing substrings during string matching.

C. Disdvantages

Some people who are not used to using regular expressions might have a hard time in understanding the characters' meaning, as some characters might have multiple definitions, depending on where it is used. Not to mention that a block of code using regular expression is quite hard to debug, since it doesn't give any additional info should no match is found within the text. Another disadvantage is that typos are easily made while making a pattern, since the creator must fully understand the purpose of the pattern that he created, as other people might not understand it, so a difference in one character could give a wrong result and other people might not realize which part is wrong. Regular expressions would be useful in search engines, such as Google, however it might consume excessive processing power depending on the pattern design and complexity, as the database is very large. For that reason, most search engines do not offer regular expression support for queries to public.

IV. WORD SHUFFLE

Word Shuffle is a game where the player is given the task of unscrambling given letters and arrange them into words with a specific length with a certain order. While this game are commonly played using application, this game can still be played using traditional methods, such as pen and paper. The player may form any word using the letters, and if the word is correctly spelled, it is counted as a correct answer. Earlier version of the game focused on finding all the possible words, but as time goes, developers also implement point systems in order to make the game more enjoyable.

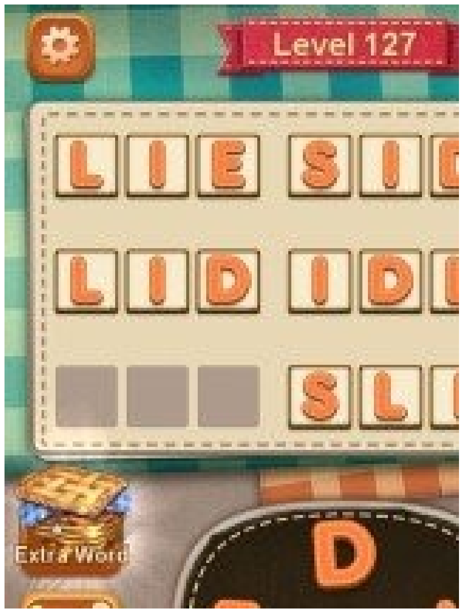


Figure 5: An example of a Word Shuffle game

(Source: https://wordbrainsolver.com/wp-content/uploads/2017/08/Word-Shuffle-Level-127.jpg)

Word Shuffle games usually already had the list of the words that need to be made in order to complete the game. However, they might also compare the created word with the dictionary, should the word created is not in the list but is still spelled correctly. Should that happen, usually the game would offer bonus points or something along that line.

Players would often use the brute force technique, which is done by trying each possible combination of letters possible, in an event where they don't know the words that could be created by the letters given. Word Shuffle games usually display empty spaces that suggests the length of the word that the player need to create, which helps in reducing the time the player need to solve the word. Once the word is found, the word is written over the empty spaces, so that the player can keep track over which words have been found.

Word Shuffle is a type of game that is played for the sake of leisure, instead of competing. It also has educational value, since it helps the player in memorizing or learning new vocabularies. Word Shuffle could also be played by children as an effort to improve their linguistic skills.

V. IMPLEMENTATION OF BACKTRACKING ALGORITHM AND REGULAR EXPRESSIONS FOR WORD SHUFFLE GAME

A. Application

Before implementing both the backtracking algorithm and regular expressions, it is necessary to make a database first that contains the list of letters and list of words that need to be created beforehand. The test will be done twice, using 5 letters and 6 letters each. Both test will be executed using the original brute force strategy, and using backtracking algorithm and regular expression.

Before executing the test, the components of the backtracking algorithm used in this experiment are as follows:

1. Solution Space
   The solution space contains all the correctly spelled words which are recorded in the answer database. The solution space is originally empty and will be filled as each correct answer is found.
2. Expand Function
   The expand function will add the current letter into the string and further expand the potential solution until the length of the string is equal to the number of letters.
3. Bounding Function
   The bounding function will implement regular expression and will be used to ensure that the current string is a substring of a potential answer.

The tracing process are as follows:

```
def backtrack(sentence,step):
    global letters
    global words
    global answersbt

    step += 1
    if((len(sentence)   ==   len(letters))
and (sentence in words)):
      answersbt.append(sentence)
      return step

    elif(sentence in words):
      answersbt.append(sentence)
      found = False
      for word in words:
      if(re.match(sentence+".+",word)):
        found = True
        break

      if found:
        for letter in letters:
         if letter not in sentence:
          nextsentence=sentence+letter
          step=
            backtrack(nextsentence,step)
        return step

    else:
     found = False
     for word in words:
        if(re.match(sentence+".+",word)):
            found = True
            break
        if found:
            for letter in letters:
            if letter not in sentence:
            nextsentence=sentence+letter
                    step               =
backtrack(nextsentence,step)

        return step
```

Figure 6: Pseudocode for the Backtracking Algorithm

(Source: Author's documentation)

The global variable letters is a list containing all the letters, while words contain all the possible words created from the letters. The global variable answersbt represents the solution space generated from the process and is originally empty. Sentence represent the current string in the process, while step represent the total steps taken during the solution generating process. Each expansion is counted as one step. How the algorithm works are as follows:

1. For each recursion, step is increased by one.
2. If the current string is already the same length as the number of letters and it is a word that is recorded in the variable words, it is added into the list answersbt. Otherwise, the answer is discarded.
3. If the length of the current string is less than the number of letters but is also a word that is recorded in the variable words, it is added into the list answersbt.
4. Then, regular expression is used to check whether the current string is a substring of at least one possible answer from the list words.
5. If the result is false, then it means that no matter how much that string is expanded, it won't generate a possible solution, and so, the current candidate is not getting expanded any further.
6. However, if the result is true, that means that there's at least one possible answer that could be created should the current string is expanded even further, and so the expansion process continues.
7. If the current string is not a word that is recorded in the variable words, it will go through the same checking and expansion process as mentioned before, with the exception that the current string is not added into the list answersbt.

The pattern used for the regular expression is "^" followed by the current sentence followed by ".+". The caret (^) represents preceded by, while the dot (.) represent any character, while the plus (+) represent one or more occasion. Simply put, the program searches for a possible answer that starts with the current string and is followed by at least one random character, which suggests that the current string might be expanded to create a possible answer.

The main program consists of calling the backtrack function for each letter in the list letters. The program also executes a different function that process the same letters, using the brute force method, without using the regular expression to prune the candidates. The program will then show the results achieved by each function, as well as the number of steps required by each function to generate the results.

B. *Test Case*

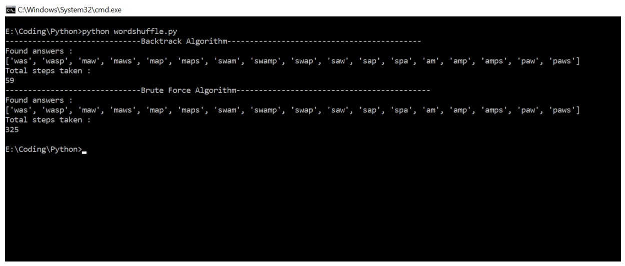
The first test case uses 5 letters and has 16 words that could be formed from those letters. The data are as follows:

Letters	w , m , s , a , p
Words	am , amp , saw , maw , paw , spa , map , was , sap , wasp , swap , maps , paws , amps , maws , swam , swamp

Figure 7: Data for the first test

(Source: Author's documentation)

The result from the test using the data from figure 7 is displayed below:



```
C:\Windows\System32\cmd.exe
E:\Coding\Python>python wordshuffle.py
------------------------------Backtrack Algorithm------------------------------------------
Found answers :
['was', 'wasp', 'maw', 'maws', 'map', 'maps', 'swam', 'swamp', 'swap', 'saw', 'sap', 'spa', 'am', 'amp', 'amps', 'paw', 'paws']
Total steps taken :
59
------------------------------Brute Force Algorithm------------------------------------------
Found answers :
['was', 'wasp', 'maw', 'maws', 'map', 'maps', 'swam', 'swamp', 'swap', 'saw', 'sap', 'spa', 'am', 'amp', 'amps', 'paw', 'paws']
Total steps taken :
325

E:\Coding\Python>
```

Figure 8: First test case results
(Source: Author's documentation)

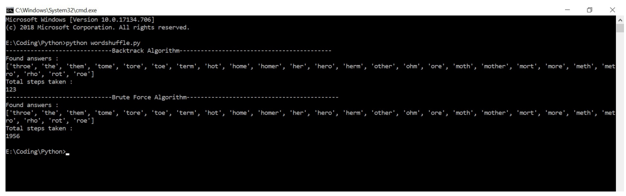
Both the backtracking algorithm and the brute force algorithm give the same results, which is the 16 words that were added into the list words. However, there's a significant difference in the number of steps taken to produce the same result. The backtracking algorithm only requires 59 steps, while the brute force algorithm requires 325 steps, more than five times the previous result.

The second test case use 6 letters and 25 words that could be formed from those letters. The data are as follows:

Letters	t , h , o , m , r , e
Words	mother , other , homer , metro , throe , moth , home , meth , tore , them , hero , herm , term , mort , more , tome , ohm , roe , rot , her , ore , the , toe , rho , hot

Figure 9: Data for the second test case
(Source: Author's documentation)

The result from the test using the data from figure 9 is displayed below:



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.17134.706]
(c) 2018 Microsoft Corporation. All rights reserved.

E:\Coding\Python>python wordshuffle.py
------------------------------Backtrack Algorithm------------------------------------------
Found answers :
['throe', 'the', 'them', 'tome', 'tore', 'toe', 'term', 'hot', 'home', 'homer', 'her', 'hero', 'herm', 'other', 'ohm', 'ore', 'moth', 'mother', 'mort', 'more', 'meth', 'met
ro', 'rho', 'rot', 'roe']
Total steps taken :
123
------------------------------Brute Force Algorithm------------------------------------------
Found answers :
['throe', 'the', 'them', 'tome', 'tore', 'toe', 'term', 'hot', 'home', 'homer', 'her', 'hero', 'herm', 'other', 'ohm', 'ore', 'moth', 'mother', 'mort', 'more', 'meth', 'met
ro', 'rho', 'rot', 'roe']
Total steps taken :
1956

E:\Coding\Python>
```

Figure 10: Second test case results
(Source: Author's documentation)

Like before, both algorithms give exactly the same results, which were 25 words. However, difference in the number of steps taken to produce the results is even higher than before, in which the backtracking algorithm only require 123 steps, while the brute force algorithm requires 1956 steps, more than ten times the previous result.

C. *Analysis*

From both test cases, it can be concluded that both the backtracking and brute force algorithm are effective in procuring all the correct answers. However, the backtracking algorithm with regular expression are much more efficient in producing the result than the brute force algorithm. This is proven by comparing the number of steps taken, where brute force algorithm requires much more steps than the backtracking algorithm. This shows that the Word Shuffle game could be solved much quicker by using the backtracking algorithm and regular expression.

VI. CONCLUSION

Backtracking algorithm is an algorithm used to solve computational problem by pruning the candidates that would not lead to a possible solution using a bounding function. On the other hand, regular expressions are used to find a specific string pattern within a text. Both of them could be used to solve the Word Shuffle game faster than the usual brute force method.

VII. ACKNOWLEDGEMENT

First of all, the author would like to thank God for His blessing, because without it, the author would not be able to finish this paper. The author also would like to thank the authors' parents and friends which have given the author their support, may it be directly or indirectly, which have helped the author to finish this paper. Lastly, the author would like to apologize for any mistakes that may have been made accidentally. May this paper be useful for future references and research purposes.

REFERENCES

[1] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Algoritma-Runut-balik-(2018).pdf accessed on April 25th, 2019

[2] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf accessed on April 25th, 2019

[3] https://medium.com/@andreaiacono/backtracking-explained-7450d6ef9e1a accessed on April 25th, 2019

[4] https://medium.com/tech-tajawal/regular-expressions-the-last-guide-6800283ac034 accessed on April 25th, 2019

[5] https://www.slideshare.net/FahimFerdous6/backtracking-algorithm-technique-and-examples accessed on April 26th, 2019

[6] https://www.slideshare.net/niekschmoller/regex-external accessed on April 26th, 2019

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 26 April 2019

[illegible]

Kevin Sendjaja / 13517023