# Question 6:

# Report

## Introduction

This is the report for analyzing and experiencing the solution for an algorithm is that there is a given dictionary.txt, the algorithm must arrange the word in the order is that the third and second last characters of the preceding word are the second and third characters of the current word, the rule is excluded for the first word. For example, if the first word is dance, then the second word must have 5 characters which are \*nc\*\* (\* can be any character). From this sequence, the algorithm also needs to find where the longest circular sequence is.

## Algorithm Description

There are many ways to deploy this problem to give a solution, however, there is an unknown goal to define how much is the longest sequence when the solution can be diverse.

### Define the longest sequence.

Assuming the number of words in the dictionary.txt is n, the longest sequence (L) must be more than or equal to 1 and less than or equal to n. There are many ways to approach the problem by using BFS, DFS, or Backtracking. All these methods can give some relatively long possible sequences the algorithms can find that eliminated to loop through all the permutations of the dictionary. This report will pick the Deep-First search method to solve the problem.

In the Deep-First search, the algorithm treats every word as a node of a directed graph. Each word will come from the edges that have the second and third substring (\*xx\*\*) and leave to the edges that have the last third and second substring (\*\*xx\*). Alternatively, the algorithm won’t stop when it finds the sequence, but it will stop when it runs out of stack elements. Then, it will look for the node that has the longest generation to generate a vector output.

Because finding the longest sequence is required, there is always a possible chance of a different started word will generate a longer sequence. DFS must loop each word of the dictionary as the started word until it cannot run anymore.

***Analysis of algorithm:***

Node data structure:

Node :

1. Value = string: storing the word of the node
2. Parent = Node\*: storing the previous parent address
3. Size = int: the current size from the starting word.

Dfs function:

1. Making the initial node by the initial word (O(1))
2. Making a stack and storing the first node (O(1))
3. Making a visited map to avoid the repeated node in the same sequence (O(1))
4. Loop to until there is no more stack: (O(N), N is the number of Nodes)
   1. Pop the first stack
   2. Mark it as visited
   3. Loop to all the possible edges it can create (O(E), E is the number word that the current word can create a substring).
      1. Create the new node
      2. Push them to the stack

### Define the longest circular.

After finding the longest sequence, the algorithm will define a list of head substrings and a list of tail substrings. Then, it loops through each substring to find the longest circular sequence.

***Analysis of algorithm:***

Head substring = map<string, int>: storing the first index of the current substring. (O(V))

Tail substring = map<string, int>: storing the last index of the current substring. (O(V))

Run all the pairs of the tail substring and save the substring that has the widest range. O(V)

## Experimental Results and Comparisons

### The possible result can find from 4 to 15-character words.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Length | Dictionary size | Longest sequence | Circular length | CPU times (Time found in the longest sequence) | Output file |
| 4 | 3862 | 92 | 92 | 0.46s | 4\_output\_log.txt  Circular\_4.txt |
| 5 | 8548 | 2971 | 2946 | 101.561s | 5\_output\_log\_fail.txt  (Terminated, bad allocated)  5\_output\_log.txt  Circular\_5.txt |
| 6 | 14383 | 5098 | 5050 | 87.537s | 6\_output\_log\_fail.txt  (Terminated, bad allocated)  6\_output\_log.txt  Circular\_6.txt |
| 7 | 21729 | 7793 | 7790 | 454.337s | 7\_output\_log\_fail.txt  (Terminated, bad allocated)  7\_output\_log.txt  Circular\_7.txt |
| 8 | 26448 | 9494 | 9481 | 1112.17s | 8\_output\_log\_fail.txt  (Terminated, bad allocated)  8\_output\_log.txt  Circular\_8.txt |
| 9 | 18844 | 4784 | 4778 | 1413.52s | 9\_output\_log\_fail.txt  (Terminated, bad allocated)  9\_output\_log.txt  Circular\_9.txt |
| 10 | 12308 | 2215 | 2210 | 443.802s | 10\_output\_log\_fail.txt  (Terminated, bad allocated)  10\_output\_log.txt  Circular\_10.txt |
| 11 | 7850 | 1474 | 1459 | 448.103s | 11\_output\_log\_fail.txt  (Terminated, bad allocated)  11\_output\_log.txt  Circular\_11.txt |
| 12 | 5194 | 868 | 853 | 206.855s | 12\_output\_log\_fail.txt  (Terminated, bad allocated)  12\_output\_log.txt  Circular\_12.txt |
| 13 | 3275 | 472 | 467 | 322.997s | 13\_output\_log\_fail.txt  (Terminated, bad allocated)  13\_output\_log.txt  Circular\_13.txt |
| 14 | 1775 | 219 | 204 | 1.95s | 14\_output\_log.txt  Circular\_14.txt |
| 15 | 954 | 93 | 80 | 0.294s | 15\_output\_log.txt  Circular\_15.txt |

## Conclusion

The problem is detected as a non-polynomial complete (NP) type because there is undefinable whether the result can be the longest. The answer is hard to detect and can be solvable by backtracking, but this method is not friendly for the computer as there is always finite storage of data.

DFS is an algorithm that defines a maximum sequence that is always a subset of the backtracking method. Nevertheless, this method has limited the sequence to optimize the sequence to the maximum length of the sequence since it has a polynomial complete solution.

The graph is visualized how the sequence of this algorithm and the result above generate the sequence. It can search for the sequence in an approximate approach to the longest sequence the DFS can find. However, there are other possible solutions that can have a longer sequence to be formed like the blue area.

The longer sequences

***Cont. of DFS***

The tree graph in the right shows the disadvantage of the DFS. it shows that if the node is visited, the algorithms will skip this node in the future, and it means it skips a sequence that may form a longer sequence for the problem.

Visited

Visited, so it stops here