# Assignment 1

## Advance Programing

## CS213: Advanced Programming

## Instructor: Fahad Ahmed Satti

## Hamza Mahmood

## BSCS-4B

## 33093

## GitHub repository Link: https://github.com/hamzamahmood/Ap-Assignment1.git

**DESCRIPTION:**

The Problem that we are given to solve is to find chains between two given words by just changing each letter of the word to creating intermediate legal words and in the end, change the first word to the other one. We have to use this solution and provided dictionary to get words to compute the chains in between them and check that the intermediate words are also legal. E.g.

Converting Cold to Warm:

Cold -> Cord -> Card -> Ward -> Warm

Intermediate chains are Cord, Card and Ward which are legal words and it takes 3 hops to convert Cold to Ward.

We have to find the chains between all the words with same length in the dictionary.

**MY SOLUTION:**

**Language used: Python**

**Environment: Program executed on Linux Mint and Analysis done on Windows 10**

In my solution, I have divided the problem into sub problems using the word length.

First I wrote a program called Shortest\_Chain.py this program used Breath-First-Search algorithm to find all the legal conversion of a word in the dictionary to other legal words by just changing ONE character from the word. E.g.

LIE could be converted into these words in just changing one character from it ['DIE', 'FIE', 'GIE', 'HIE', 'KIE', 'PIE', 'RIE', 'TIE', 'VIE', 'LEE', 'LYE', 'LIG'] and these words are in the dictionary so these words are legal.

Shortest\_Chain.py has a for loop that iterates from 1 to 35 and there basically shows the range of word length in the dictionary. It creates Shortest\_Chain1.csv - Shortest\_Chain35.csv. Each file contains the word and its legal conversions by just changing one character. The Number represents that the file contains conversion of words of that length.

Next by just changing a condition in Shortest\_Chain.py, I have written No\_Chains.py that finds the words that do not have any conversions/chains. This python script also creates No\_Chains1.csv – No\_Chains35.csv.

Next to solve the problem of conversion of one word to another using legal intermediate words, I have written ladder.py script. This scripts uses the results from Shortest\_Chain.py which are Shortest\_Chain1.csv - Shortest\_Chain35.csv. What it does is that it loads these files into a Python Dictionary with KEY the word and the VALUE is list of legal conversions of the words in a single hop i.e. by changing one letter. E.g.

For a key LIE the possible conversion by changing one character are:

['DIE', 'FIE', 'GIE', 'HIE', 'KIE', 'PIE', 'RIE', 'TIE', 'VIE', 'LEE', 'LYE', 'LIG']

After loading the dictionary of legal conversion of a word to others in one hop I have taken a word from the dictionary and next word from the dictionary (OF SAME LENGTH) **(NOTE THE SECOND WORD MUST BE IN THE ABOVE DICTIONARY CONTAINING LEGAL CONVERSIONS, IF NOT THAN THAT WORD IS NOT INCLUDED BECAUSE IT DOESN’T HAVE ANY CHAINS**) and applied Depth-First-Search to convert the first word to the next one. The successors of the first words are retrieved from the above dictionary (KEY VALUE) and these successors are pushed along with the chain as a tuple (WORD, PATH\_LIST []) this tuple is then popped and its successors are pushed on the stack until the second word is found or the stack is empty. If the second word appears in the DFS than the Word and the Chain is written into CSV file.

Running the above script ladder.py, it creates Chains2.csv – Chain35.csv each containing the conversion of all the words from the dictionary to the other ones with the same length.

**ANALYSIS:**

Below are the graphs that are extracted from the file that created by the above scripts and some additional scrips.

**Graph 0: Word Length Distribution**

**Graph 1: No Chains**

Data of this graph is collected by the script Frequency\_NoChains.py that collects all the data into a csv file from No\_Chains1.csv – No\_Chains35.csv

This graph shows the word length and the frequency of words that doesn’t have any chains.

**Graph 2: Chain Length**

This graph show the Chain Length mapped against the frequency of words. That is How many words have a given chain length. Created using Chains1.csv – Chains35.csv

Data collected by Frequency\_Chain.py

**Graph 3: Longest Chain**

This graph uses the Chain1.csv – Chains35.csv. The data is collected by the script Longest\_Chain.py

**Graph 4: Word of same length to the Chain Size**

The below graphs represent the Length of chains formed against the word length.

NOTE: The graph of word length 1 is not drawn because ease word could be converted into other one and longest chain would be number of 1 letter words – 1 and shortest chain will be of ONE hop.

The next words do not have any chains.