Amrita School of Engineering, Bengaluru

Amrita Vishwa Vidyapeetham

Course: 19AIE212 – Design and Analysis of Algorithms

Course Instructor: Dr. Supriya M.

Course Project: Line Breakup for Paragraph Formation

Team – Dynamic Dudes:

1.) BL.EN.U4AIE190**13** - **CHARAN TEJ K.**

2.) BL.EN.U4AIE190**14** - **CHAVALI SURYA TEJA**

3.) BL.EN.U4AIE190**62** - **SUGASH T.M.**

INTRODUCTION

One of the most important operations necessary when text materials are prepared for printing or display is the task of dividing long paragraphs into individual lines. When this job has been done well, people will not be aware of the fact that the words they are reading have been arbitrarily broken apart and placed into a somewhat rigid and unnatural rectangular framework; but if the job has been done poorly, readers will be distracted by bad breaks that interrupt their train of thought.

The line-breaking problem is informally called the problem of ‘justification’, since it is the ‘J’ of ‘H & J’ (hyphenation and justification) in today’s commercial composition and word-processing systems. However, this tends to be a misnomer, because printers have traditionally used justification to mean the process of taking an individual line of type and adjusting its spacing to produce a desired length.

According to the mathematical terms, t there are 2” ways to break a paragraph into lines, if the paragraph has n legal breakpoints that aren’t forced. The general ideas underlying the optimum-fit algorithm for line breaking can probably be understood best by considering an example.

A feasible breakpoint is a place where the text of the paragraph from the beginning to this point can be broken into lines whose adjustment ratio does not exceed a given tolerance. We proceed by locating all of the feasible breakpoints and remembering the best way to get to each one, in the sense of fewest total demerits. This is done by keeping a list of ‘active’ breakpoints, representing all of the feasible breakpoints that might be a candidate for future breaks.

DYNAMIC PROGRAMMING

Dynamic Programming is mainly an optimization over plain recursion. Wherever we see a recursive solution that has repeated calls for same inputs, we can optimize it using Dynamic Programming. The idea is to simply store the results of sub-problems, so that we do not have to re-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial. For example, if we write simple recursive solution for Fibonacci Numbers, we get exponential time complexity and if we optimize it by storing solutions of sub-problems, time complexity reduces to linear.

In our program we applied dynamic programming by breaking the sub problems as suffixes (remaining words in the list) and in the list of words we guessed where to start the consecutive lines. Accordingly, we applied recursive function which includes the penalty function for breaking the line. And finally, we mapped our paragraph.

Advantages of Dynamic Programming over recursion:

- As it is a recursive programming technique, it reduces the line code.
- One of the major advantages of using dynamic programming is it speeds up the processing as we use previously calculated references.
- It is suitable for linear or non-linear problems, discrete or continuous variables, and deterministic problems.

GREEDY ALGORITHM

Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. So, the problems where choosing locally optimal also leads to global solution are best fit for Greedy.

If a greedy algorithm can be proven to yield the global optimum for a given problem class, it typically becomes the method of choice because it is faster than other optimization methods like dynamic programming. Examples of such greedy algorithms are Kruskal's algorithm and Prim's algorithm for finding minimum spanning trees, and the algorithm for finding optimum Huffman trees. Greedy algorithms appear in network routing as well. Using greedy routing, a message is forwarded to the neighbouring node which is "closest" to the destination.

In our program we applied greedy algorithm by blindly confining as much as possible words in the lines. However, it is not the optimal solution.

Advantages of Greedy Approach:

The greedy approach has a few trade-offs, which may make it suitable for optimization:

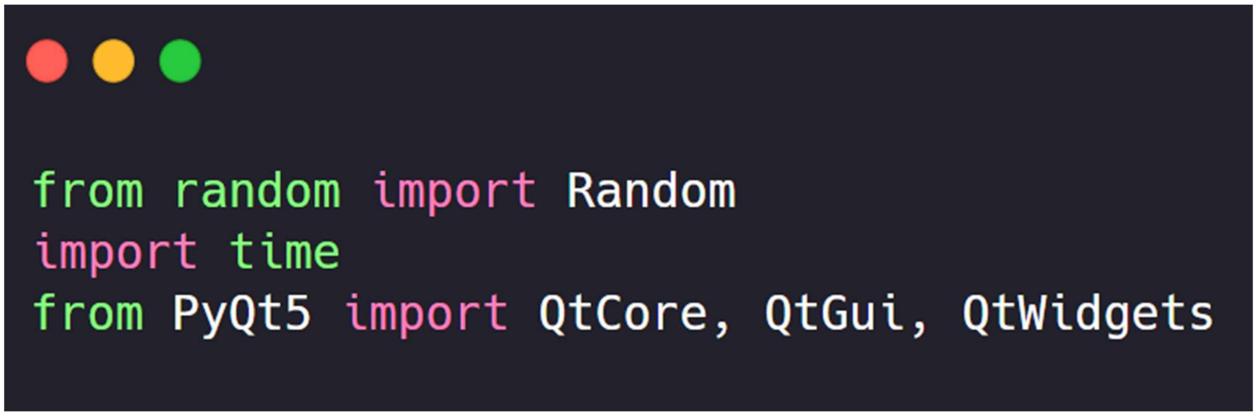
- One prominent reason is to achieve the most feasible solution immediately.
- Greedy choice property: This property says that the globally optimal solution can be obtained by making a locally optimal solution (Greedy). The choice made by a Greedy algorithm may depend on earlier choices but not on the future. It iteratively makes one Greedy choice after another and reduces the given problem to a smaller one.

IMPLEMENTATION

Now that we've discussed the pre-requisites for our project, we now move on to the implementation part of both our algorithms and later, we are going to compare the times taken by both the algorithms and decide which one is better for our problem statement accordingly.

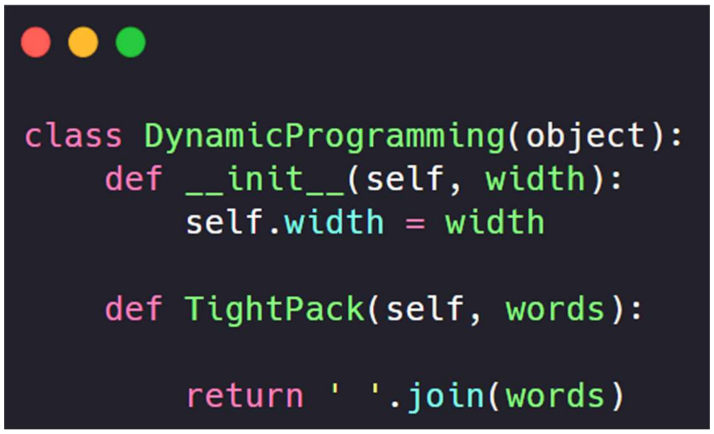
Below, we showcase a sample of our implementation part:

1.) As the first, it is very important for us to import some required dependencies such as:



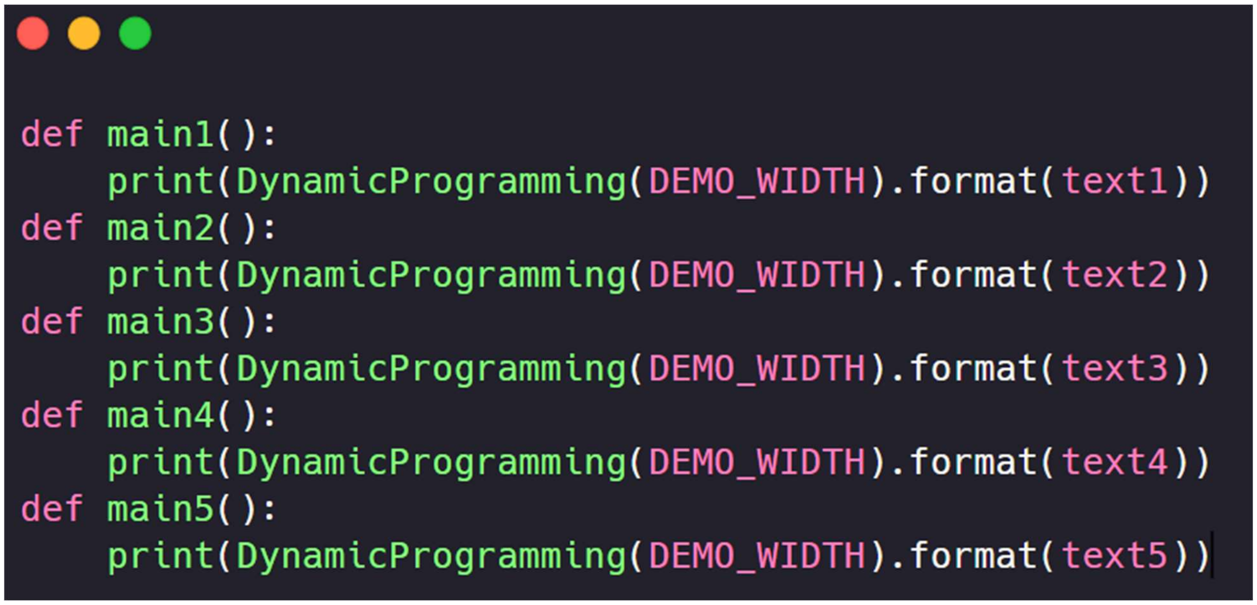
```
from random import Random
import time
from PyQt5 import QtCore, QtGui, QtWidgets
```

2.) We then proceed on to create a dynamic program for our text justification problem:



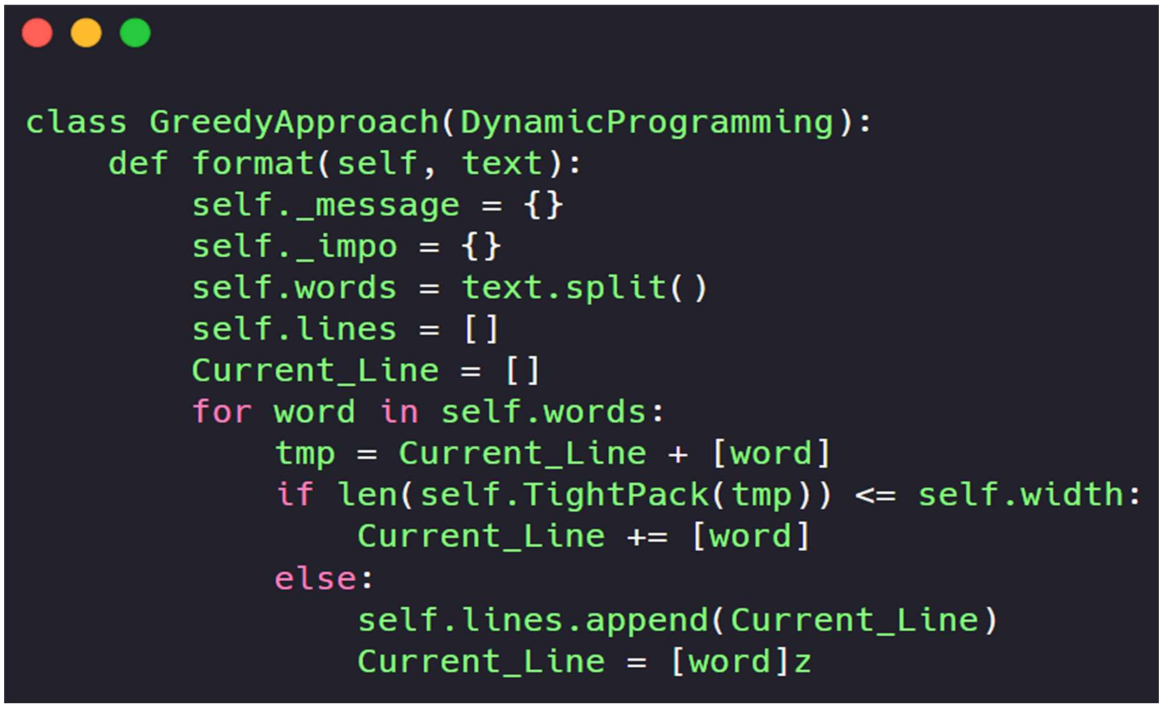
```
class DynamicProgramming(object):
    def __init__(self, width):
        self.width = width

    def TightPack(self, words):

        return ' '.join(words)
```

3.) Later, we create functions for testing all our sample texts and later use the results obtained for calculating the times taken by each:



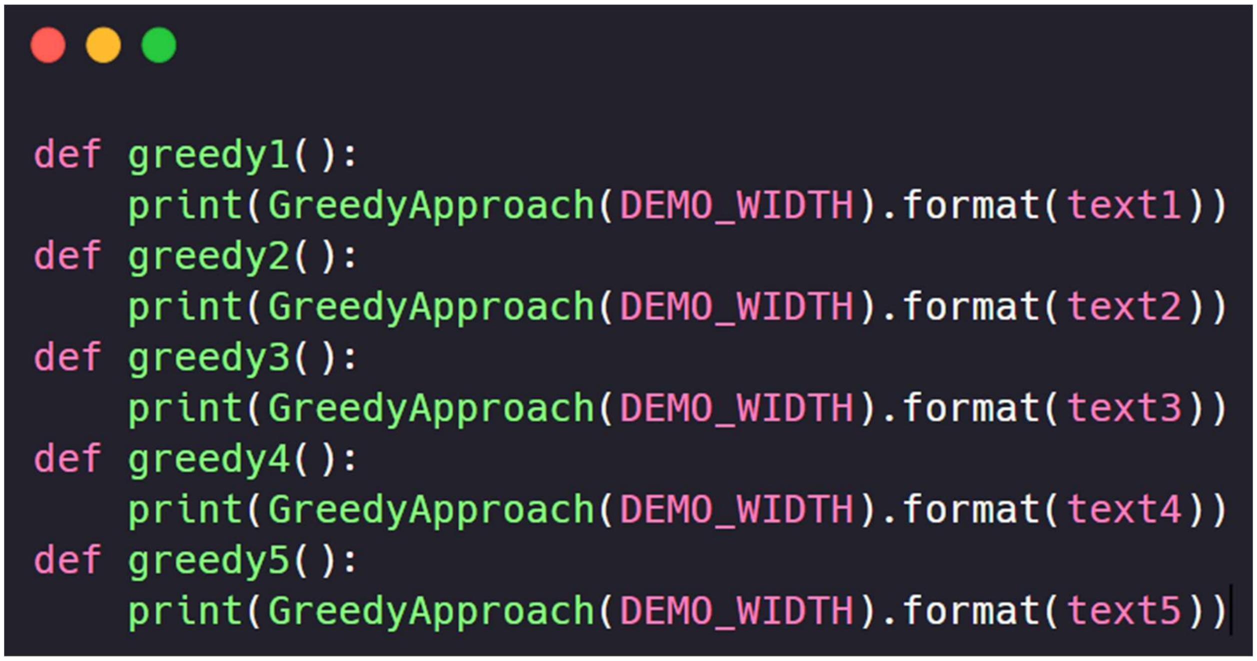
```
def main1():
    print(DynamicProgramming(DEMO_WIDTH).format(text1))
def main2():
    print(DynamicProgramming(DEMO_WIDTH).format(text2))
def main3():
    print(DynamicProgramming(DEMO_WIDTH).format(text3))
def main4():
    print(DynamicProgramming(DEMO_WIDTH).format(text4))
def main5():
    print(DynamicProgramming(DEMO_WIDTH).format(text5))
```

4.) As discussed previously, we now implement Greedy Approach for our problem statement:



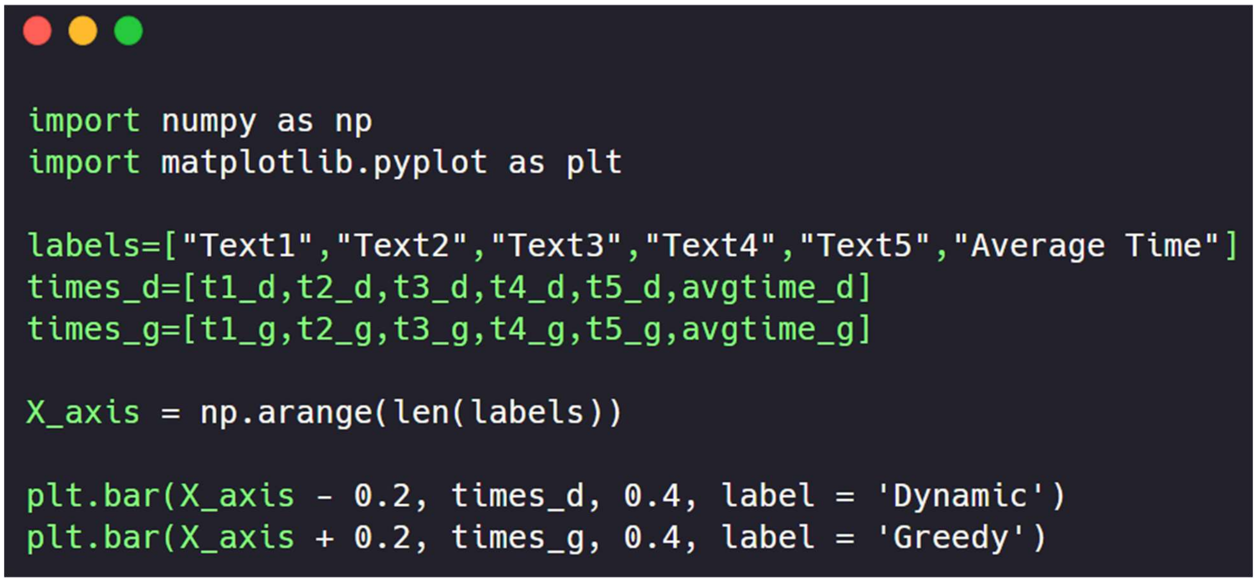
```
class GreedyApproach(DynamicProgramming):
    def format(self, text):
        self._message = {}
        self._impo = {}
        self.words = text.split()
        self.lines = []
        Current_Line = []
        for word in self.words:
            tmp = Current_Line + [word]
            if len(self.TightPack(tmp)) <= self.width:
                Current_Line += [word]
            else:
                self.lines.append(Current_Line)
                Current_Line = [word]z
```

5.) Similarly, we create 5 different functions for executing the sample texts:



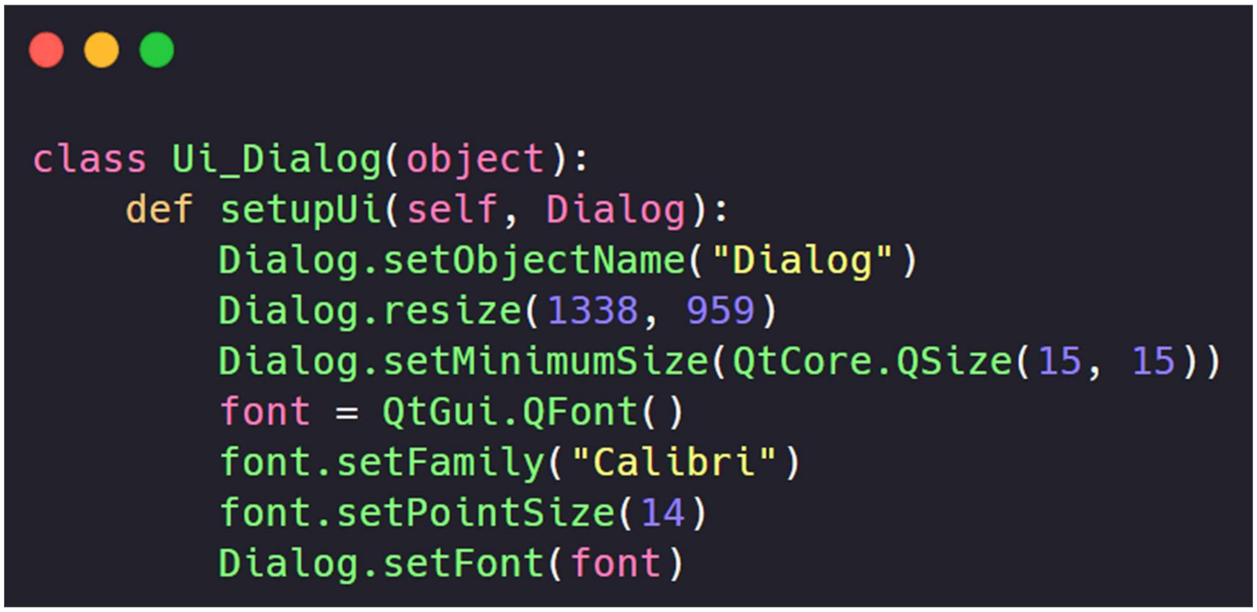
```python
def greedy1():
    print(GreedyApproach(DEMO_WIDTH).format(text1))
def greedy2():
    print(GreedyApproach(DEMO_WIDTH).format(text2))
def greedy3():
    print(GreedyApproach(DEMO_WIDTH).format(text3))
def greedy4():
    print(GreedyApproach(DEMO_WIDTH).format(text4))
def greedy5():
    print(GreedyApproach(DEMO_WIDTH).format(text5))
```

6.) After calculating the respective times for each test case, we move on for the visualization part, by plotting them:



```python
import numpy as np
import matplotlib.pyplot as plt

labels=["Text1","Text2","Text3","Text4","Text5","Average Time"]
times_d=[t1_d,t2_d,t3_d,t4_d,t5_d,avgtime_d]
times_g=[t1_g,t2_g,t3_g,t4_g,t5_g,avgtime_g]

X_axis = np.arange(len(labels))

plt.bar(X_axis - 0.2, times_d, 0.4, label = 'Dynamic')
plt.bar(X_axis + 0.2, times_g, 0.4, label = 'Greedy')
```

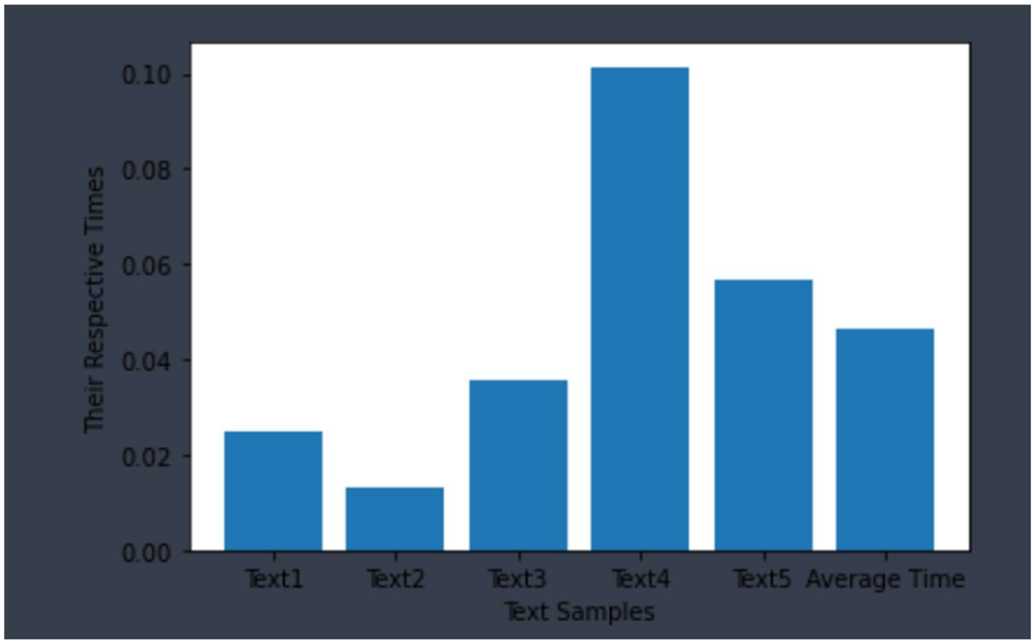
7.) The last part is nothing but, visualizing entire project in a GUI to make much more appealing for the user. Below is the sample code for that:



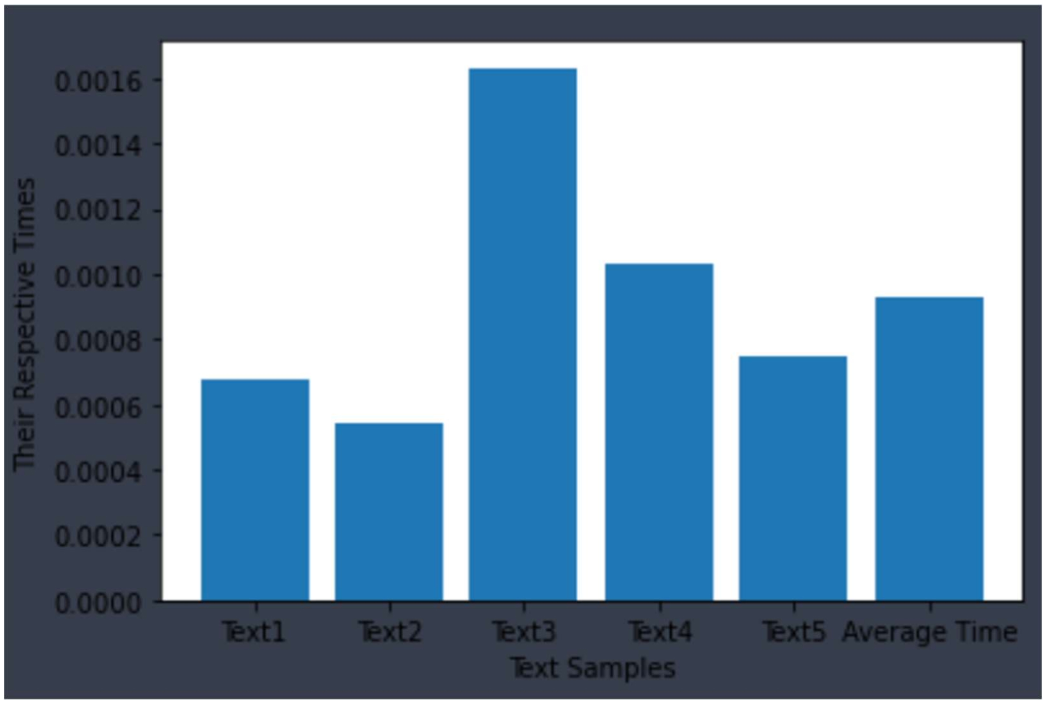
```
class Ui_Dialog(object):
    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
        Dialog.resize(1338, 959)
        Dialog.setMinimumSize(QtCore.QSize(15, 15))
        font = QtGui.QFont()
        font.setFamily("Calibri")
        font.setPointSize(14)
        Dialog.setFont(font)
```

RESULTS

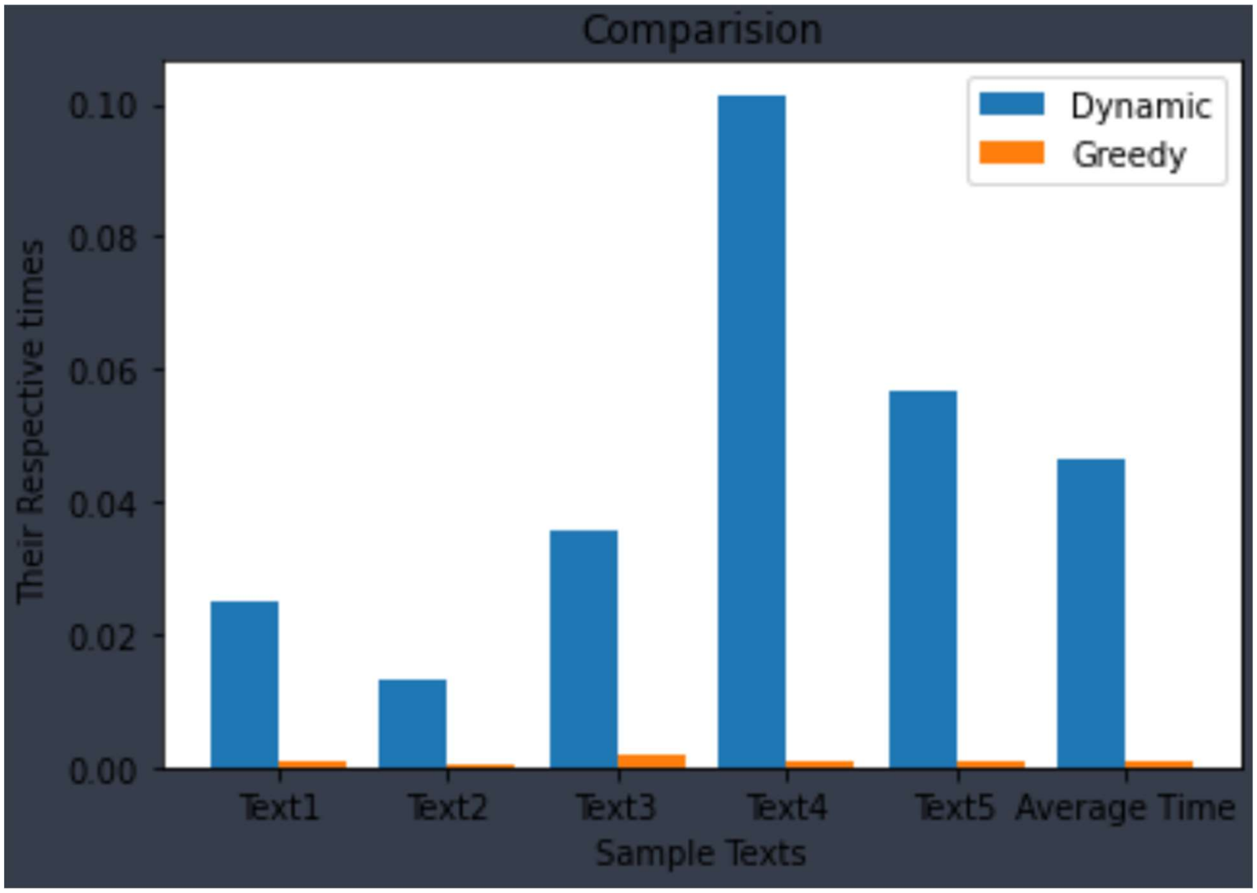
1.) The first is the time comparison of the sample cases, along with the average time too, when tested using Dynamic Programming:



2.) Similarly, we do it for Greedy algorithm also:

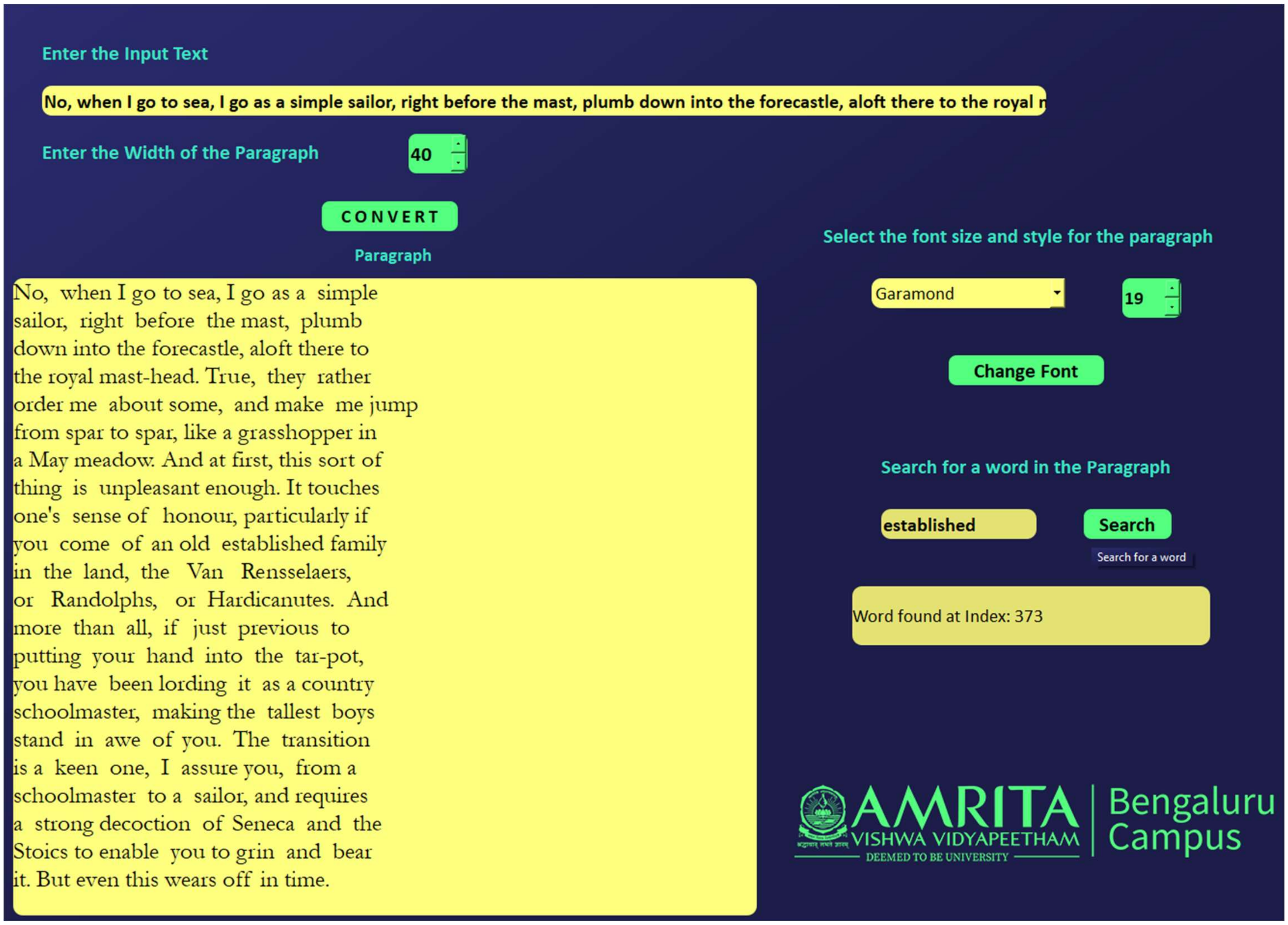


3.) Now comes the most important result that is the time analysis between both dynamic programming approach and greedy algorithm:



We observe that Greedy approach worked excellently well when compared Dynamic Programming approach. Greedy algorithm was around 100 times better than DP, as far as time complexity is concerned.

4.) Below is our application GUI, in which we have added several text beautification techniques such as font style and font size selection along with the facility to find the index of word selected.



Enter the Input Text

No, when I go to sea, I go as a simple sailor, right before the mast, plumb down into the forecastle, aloft there to the royal r

Enter the Width of the Paragraph: 40

Paragraph

No, when I go to sea, I go as a simple
sailor, right before the mast, plumb
down into the forecastle, aloft there to
the royal mast-head. True, they rather
order me about some, and make me jump
from spar to spar, like a grasshopper in
a May meadow. And at first, this sort of
thing is unpleasant enough. It touches
one's sense of honour, particularly if
you come of an old established family
in the land, the Van Rensselaers,
or Randolphs, or Hardicanutes. And
more than all, if just previous to
putting your hand into the tar-pot,
you have been lording it as a country
schoolmaster, making the tallest boys
stand in awe of you. The transition
is a keen one, I assure you, from a
schoolmaster to a sailor, and requires
a strong decoction of Seneca and the
Stoics to enable you to grin and bear
it. But even this wears off in time.

Select the font size and style for the paragraph: Garamond, 19

Search for a word in the Paragraph: established

Word found at Index: 373

CONCLUSION

To the best of our knowledge, we have explained about Dynamic Programming and greedy approach along with working principles, and have successfully implemented and executed them in our project.

In reality, many programmers go for dynamic programming, as it always gives us the optimal solution, whereas greedy algorithm won’t guarantee us the optimal solution every time. Every optimal solution is a feasible solution, but not every feasible solution is an optimal solution, as feasible solution refers to any solution that solves a particular problem, but optimal gives us the maximum result.

Finally, we conclude by saying that we as a team understood the importance algorithmic design techniques and their applications in tackling the real-world problems.

****