Koneru Lakshmaiah Education Foundation
(Deemed to be University estd. u/s. 3 of the UGC Act, 1956)
Off-Campus: Bachupally-Gandimaisamma Road, Bowrampet, Hyderabad, Telangana - 500 043.
Phone No: 7815926816, www.klh.edu.in

Department of Computer Science and Engineering

2025-2026
Odd Semester

DESIGN AND ANALYSIS ALGORITHMS
(24CS2203)

ALM – PROJECT BASED LEARNING

TEXT JUSTIFICATION (WORD WRAP PROBLEM)

ANAHITA BHALME 2420030708
ATHARVA JAIN 2420030092

COURSE INSTRUCTOR

Dr. J Sirisha Devi
Professor
Department of Computer Science and Engineering

PROBLEM STATEMENT

The **Text Justification (Word Wrap)** problem focuses on arranging a sequence of words neatly across multiple lines within a given maximum width. It is a **classic Dynamic Programming (DP)** problem that demonstrates how to optimize text layout for readability.

In real-world applications like **Microsoft Word, Google Docs, or publishing systems**, text must be aligned properly so that the right margin appears even. The main challenge is to minimize the uneven white spaces (raggedness) at the end of each line.

Problem Definition

Given:

- A list of words with their lengths
- A maximum line width **M**

Goal:
Arrange the words into multiple lines such that the **sum of squares of the extra spaces** at the end of each line (except the last) is minimized.

Example

Words: ["This", "is", "an", "example", "of", "text", "justification"]
Max width (M): 16

Output:

- Line 1: This is an
- Line 2: example of
- Line 3: text justification

Objectives

- Maintain the original word order.
- Avoid splitting words between lines.
- Minimize total raggedness for better readability.

Applications

- Word processors (MS Word, Google Docs)
- E-book and PDF layout engines
- Webpage rendering and publishing tools

ALGORITHM / PSEUDO CODE

Dynamic Programming Approach

Input:
wordLengths[1..n] – array of word lengths
M – maximum line width

Steps:

1. Compute extras[i][j] = M - (sum of wordLengths[i..j]) - (j - i)
   → represents unused spaces if words i to j are on the same line.
2. If extras[i][j] < 0, line i..j does not fit within width M.
3. Compute cost for valid lines:
   cost[i][j] = extras[i][j]^2 if extras[i][j] ≥ 0, else ∞.
4. Initialize dynamic programming array:
   dp[0] = 0
5. Compute minimum total cost:
6. Reconstruct lines using parent[] to determine line breaks

Output:

Lines arranged with **minimum total raggedness**.

Example Execution

Input: ["This", "is", "an", "example", "of", "text", "justification"], M = 16

Output:

- Line 1: This is an
- Line 2: example of
- Line 3: text justification

SPACE COMPLEXITY

The algorithm uses the following data structures:

Structure	Purpose	Space
extras[i][j]	Stores unused spaces for words i..j	$O(n^2)$
cost[i][j]	Stores cost for words i..j	$O(n^2)$
dp[j], parent[j]	Used for DP and reconstruction	O(n)

Total Space Complexity: $O(n^2)$

TIME COMPLEXITY

- Computing extra spaces: O(n^2) (nested loops for i and j).
- Computing cost: O(n^2) (nested loops for i and j).
- Dynamic Programming: O(n^2) (for each j, iterate over i from 1 to j).
- Reconstruction: O(n) (linear traversal of parent array).

Total Time Complexity: O(n^2)

CONCLUSION

- Text Justification is a Dynamic Programming problem that optimally distributes words across lines.
- It balances readability and alignment by minimizing uneven spaces.
- The algorithm runs in $O(n^2)$ time and can be optimized for space.
- This approach is widely used in real-world systems like MS Word, Google Docs, and LaTeX (Knuth's line-breaking algorithm).

==

GitHub repository link-

https://github.com/jain-here/2420030092_DAA

==