Design and Analysis of Algorithms (24CS2203)

Dynamic Programming For Text Justification

Anahita Bhalme : 2420030708

Atharva Jain : 2420030092

Course Instructor

Dr. J Sirisha Devi

Professor

Department of Computer Science and Engineering

Case study - statement

Text justification is a classic optimization problem in document formatting, where poor line breaks can make paragraphs look jagged, affecting readability. Traditional greedy approaches pack words until the line fills, but this often leads to suboptimal raggedness. Dynamic Programming (DP) provides the globally optimal solution by considering all possible line divisions.

Real-World Example: Newspaper Column Layout Imagine formatting a news article with words of varying lengths into a 80-character column.

Algorithm

•Precompute Sums and Costs: For all $i \le j$, compute total characters in words i to j: $sum\_L = \Sigma L[k]$ for $k=i$ to j. Extra spaces needed: $(j-i)$ for gaps. If $sum\_L + (j-i) \le M$, cost $= (M - sum\_L - (j-i))^2$; else ∞.

•DP Table: dp[0] = 0 (no words, 0 penalty). For j=1 to n: dp[j] = min over i=1 to j of (dp[i-1] + cost[i][j]) if feasible. Track parent[j] = best i.

•Reconstruction: Start from j=n, follow parents to find line starts: lines from parent[j] to j, then recurse to parent[j]-1.

•Output: Print lines with words separated by spaces, padded with extras distributed evenly (but for penalty, just compute).

Pseudo code

Example: Words: ["This", "is", "an", "example", "of", "text", "justification"] Lengths: [4,2,2,7,2,4,13] M: 16
DP Computation (n=7):

- **prefix: [0,4,6,8,15,17,21,34]**
- **For j=1: i=1, line=4+0=4 ≤16, extra=12, cost=144, dp[1]=144? Wait, but typically last line 0, but here all but last. Standard: penalty only for non-last lines.**

Correction in algo: For lines except the very last, add cost; last line cost=0 always. Adjust: When computing for j=n, the last line i to n has cost=0 if fits.

TIME COMPLEXITY

Time Complexity

- Precomputing Prefix Sums: $O(n)$ – single pass.

- DP Fill: For each $j=1$ to n, loop $i=1$ to j: $O(1)$ per (compute sum $O(1)$ via prefix, check, update). Total pairs (i,j): $\sum_{j=1}^{n} j = O(n^2)$.

- Reconstruction: $O(n)$ – follow parents.

- Overall: $O(n^2)$ – Quadratic, efficient for n up to 10^4 (0.1s on modern machines).

Space Complexity

Space Complexity Analysis:

•Prefix Array: O(n).

•DP Array: O(n) – 1D, as we only need previous values.

•Parent Array: O(n) for reconstruction.

•No 2D Cost Table Needed: Compute on-the-fly, saving O(n²).

•Overall: O(n) – Linear, scalable.