# Dynamic Programming | Set 19 (Word Wrap Problem)

Given a sequence of words, and a limit on the number of characters that can be put in one line (line width). Put line breaks in the given sequence such that the lines are printed neatly. Assume that the length of each word is smaller than the line width.

The word processors like MS Word do task of placing line breaks. The idea is to have balanced lines. In other words, not have few lines with lots of extra spaces and some lines with small amount of extra spaces.

```
The extra spaces includes spaces put at the end of every line except the last one.
The problem is to minimize the following total cost.
 Cost of a line = (Number of extra spaces in the line)^3
 Total Cost = Sum of costs for all lines
```

```
For example, consider the following string and line width M = 15
 "Geeks for Geeks presents word wrap problem"

Following is the optimized arrangement of words in 3 lines
Geeks for Geeks
presents word
wrap problem

The total extra spaces in line 1, line 2 and line 3 are 0, 2 and 3 respectively.
So optimal value of total cost is 0 + 2*2 + 3*3 = 13
```

Please note that the total cost function is not sum of extra spaces, but sum of cubes (or square is also used) of extra spaces. The idea behind this cost function is to balance the spaces among lines. For example, consider the following two arrangement of same set of words:

**1)** There are 3 lines. One line has 3 extra spaces and all other lines have 0 extra spaces. Total extra spaces = 3 + 0 + 0 = 3. Total cost = 3*3*3 + 0*0*0 + 0*0*0 = 27.

**2)** There are 3 lines. Each of the 3 lines has one extra space. Total extra spaces = 1 + 1 + 1 = 3. Total cost = 1*1*1 + 1*1*1 + 1*1*1 = 3.

Total extra spaces are 3 in both scenarios, but second arrangement should be preferred because extra spaces are balanced in all three lines. The cost function with cubic sum serves the purpose because the value of total cost in second scenario is less.

**Method 1 (Greedy Solution)**
The greedy solution is to place as many words as possible in the first line. Then do the same thing for the second line and so on until all words are placed. This solution gives optimal solution for many cases, but doesn't give optimal solution in all cases. For example, consider the following string "aaa bb cc ddddd" and line width as 6. Greedy method will produce following output.

```
aaa bb
cc
ddddd
```

Extra spaces in the above 3 lines are 0, 4 and 1 respectively. So total cost is 0 + 64 + 1 = 65.

But the above solution is not the best solution. Following arrangement has more balanced spaces. Therefore less value of total cost function.

```
aaa
bb cc
ddddd
```

Extra spaces in the above 3 lines are 3, 1 and 1 respectively. So total cost is 27 + 1 + 1 = 29.

Despite being sub-optimal in some cases, the greedy approach is used by many word processors like MS Word and OpenOffice.org Writer.

**Method 2 (Dynamic Programming)**
The following Dynamic approach strictly follows the algorithm given in solution of Cormen book. First we compute costs of all possible lines in a 2D table lc[][]. The value lc[i][j] indicates the cost to put words from i to j in a single line where i and j are indexes of words in the input sequences. If a sequence of words from i to j cannot fit in a single line, then lc[i][j] is considered infinite (to avoid it from being a

part of the solution). Once we have the lc[][] table constructed, we can calculate total cost using following recursive formula. In the following formula, C[j] is the optimized total cost for arranging words from 1 to j.

$$c[j] = \begin{cases} 0 & \text{if } j = 0\,, \\ \min\limits_{1 \le i \le j} (c[i-1] + lc[i,j]) & \text{if } j > 0\,. \end{cases}$$

The above recursion has overlapping subproblem property. For example, the solution of subproblem c(2) is used by c(3), C(4) and so on. So Dynamic Programming is used to store the results of subproblems. The array c[] can be computed from left to right, since each value depends only on earlier values.
To print the output, we keep track of what words go on what lines, we can keep a parallel p array that points to where each c value came from. The last line starts at word p[n] and goes through word n. The previous line starts at word p[p[n]] and goes through word p[n] – 1, etc. The function printSolution() uses p[] to print the solution.
In the below program, input is an array l[] that represents lengths of words in a sequence. The value l[i] indicates length of the ith word (i starts from 1) in theinput sequence.

```
// A Dynamic programming solution for Word Wrap Problem
#include <limits.h>
#include <stdio.h>
#define INF INT_MAX

// A utility function to print the solution
int printSolution (int p[], int n);

// l[] represents lengths of different words in input sequence. For example,
// l[] = {3, 2, 2, 5} is for a sentence like "aaa bb cc ddddd".  n is size of
// l[] and M is line width (maximum no. of characters that can fit in a line)
void solveWordWrap (int l[], int n, int M)
{
    // For simplicity, 1 extra space is used in all below arrays

    // extras[i][j] will have number of extra spaces if words from i
    // to j are put in a single line
    int extras[n+1][n+1];

    // lc[i][j] will have cost of a line which has words from
    // i to j
    int lc[n+1][n+1];

    // c[i] will have total cost of optimal arrangement of words
    // from 1 to i
    int c[n+1];

    // p[] is used to print the solution.
    int p[n+1];

    int i, j;

    // calculate extra spaces in a single line.  The value extra[i][j]
    // indicates extra spaces if words from word number i to j are
    // placed in a single line
    for (i = 1; i <= n; i++)
    {
        extras[i][i] = M - l[i-1];
        for (j = i+1; j <= n; j++)
            extras[i][j] = extras[i][j-1] - l[j-1] - 1;
    }

    // Calculate line cost corresponding to the above calculated extra
    // spaces. The value lc[i][j] indicates cost of putting words from
    // word number i to j in a single line
    for (i = 1; i <= n; i++)
    {
        for (j = i; j <= n; j++)
        {
            if (extras[i][j] < 0)
                lc[i][j] = INF;
            else if (j == n && extras[i][j] >= 0)
                lc[i][j] = 0;
            else
                lc[i][j] = extras[i][j]*extras[i][j];
        }
    }

    // Calculate minimum cost and find minimum cost arrangement.
    //  The value c[j] indicates optimized cost to arrange words
    // from word number 1 to j.
    c[0] = 0;
    for (j = 1; j <= n; j++)
    {
        c[j] = INF;
        for (i = 1; i <= j; i++)
        {
            if (c[i-1] != INF && lc[i][j] != INF && (c[i-1] + lc[i][j] < c[j]
            {
                c[j] = c[i-1] + lc[i][j];
                p[j] = i;
            }
        }
    }

    printSolution(p, n);
}

int printSolution (int p[], int n)
{
    int k;
    if (p[n] == 1)
        k = 1;
    else
        k = printSolution (p, p[n]-1) + 1;
    printf ("Line number %d: From word no. %d to %d \n", k, p[n], n);
    return k;
}

// Driver program to test above functions
int main()
{
    int l[] = {3, 2, 2, 5};
    int n = sizeof(l)/sizeof(l[0]);
    int M = 6;
    solveWordWrap (l, n, M);
    return 0;
}
```

Output:

```
Line number 1: From word no. 1 to 1
Line number 2: From word no. 2 to 3
Line number 3: From word no. 4 to 4
```

Time Complexity: O(n^2)
Auxiliary Space: O(n^2) The auxiliary space used in the above program cane be optimized to O(n) (See the reference 2 for details)

**References:**
http://en.wikipedia.org/wiki/Word_wrap

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

## Related Topics:

- Linearity of Expectation
- Iterative Tower of Hanoi
- Count possible ways to construct buildings
- Build Lowest Number by Removing n digits from a given number
- Set Cover Problem | Set 1 (Greedy Approximate Algorithm)
- Find number of days between two given dates
- How to print maximum number of A's using given four keys
- Write an iterative O(Log y) function for pow(x, y)

Tags: Dynamic Programming

Tweet 0 g+1 1

**Writing code in comment?** Please use **ideone.com** and share the link here.

39 Comments GeeksforGeeks 1 Login

Recommend Share Sort by Newest

Join the discussion…

**peng li** • 2 months ago

check my try and analysis: http://allenlipeng47.com/Perso...

• Reply • Share ›

**peng li** • 2 months ago

Geeks for Geeks
presents word
wrap problem

The total extra spaces in line 1, line 2 and line 3 are 0, 2 and 3 respectively.
So optimal value of total cost is 0 + 2*2 + 3*3 = 13

Is it wrong? Right cost should be 2^3+3^3 = 35

2 • Reply • Share ›

**Ankit Goyal** • 3 months ago

This is a very complicated approach. We can use memoization and the concept of parent pointers to do it in an easier way.
Here is my code in C#.

http://ideone.com/e.js/0TTMUn

This is straightaway an implementation of the algo taught in the following MIT lecture ---

https://www.youtube.com/watch?...

• Reply • Share ›

**Vis** → Ankit Goyal • a month ago

Next time try out your solution before submitting. NullPointerException (i+1) is not in DP prior accessing it.

• Reply • Share ›

**Ankit Goyal** → Vis • a month ago

And what makes you think I haven't "tried out" my solution before submitting. Did you run it is my question? Coz this code works totally fine on my system as well as on ideone.

• Reply • Share ›

**Aditya Goel** • 3 months ago

#MIT
https://www.youtube.com/watch?...

• Reply • Share ›

**Μερκούρης Παπαμιχαήλ** • 4 months ago

hi! Could please anyone explain the last part of the algorithm, how exactly the nested for-loop works, how it calculates the minimum cost?

1 • Reply • Share ›

**Priyal Rathi** • 8 months ago

Another approach with O(n) space complexity and O(n^2) time complexity

Let end=numwords
cost[i] = cost paid in word wrap from i to end
If path[i] = j => from i to j all words will be in a line in optimal case where line is starting from ith word

```
for i=end to 0

for j=i to end
{
//calulate cost considering words from i to j in line
p= linewidth- len(words from i to j in line)
cst= p cube + cost[j+1];
// get the index j such that we will get min cost if we keep words from i to j in the line
}
get cost[i] and path[i]
}
```

Time complexity: O(n^2)
space complexity: O(n)

Link of code: http://ideone.com/Y8naTf

• Reply • Share ›

**lila** → Priyal Rathi • 6 months ago

can u explain/share a link where this approach is explained?
Thanks

• Reply • Share ›

**Kenneth** • 8 months ago

My solution with space complexity O(n). Basic idea is to consider how many words of the first line can result in optimal/minimum cost.

http://ideone.com/epkSLI

• Reply • Share ›

**rishav** • 9 months ago

Has been asked in interview. :)

• Reply • Share ›

**Guest** • 9 months ago

I apologize if this should be an obvious one, but I'm just seeing no mention of it here. Does the code account for a limited number of lines? Or is it assumed that lines are added as they are filled according to their widths?

• Reply • Share ›

**siddhant06** • 9 months ago

can someone plz explain me printSolution()

2 • Reply • Share ›

**hello** • 10 months ago

This algorithm has a limitation, each and every word length should less than or equal to the maximum words per line M.

• Reply • Share ›

**rihansh** → hello • 5 months ago

This is done to maintain the readability of the solution because if there is a word which cannot fit in a line then there does not exist any solution as you can't break words to find an optimal solution .

• Reply • Share ›

**Guest** • 10 months ago

failed in this input,
int l[] ={1, 7, 11, 9, 4, 7, 3, 8, 4, 2, 5, 3, 7, 3, 4, 4, 3, 4, 9};

• Reply • Share ›

**tushar** → Guest • 9 months ago

it is mentioned tht the word lenth shud be less than given line width

• Reply • Share ›

**SG** • 10 months ago

Can you please send the link for reduced time complexity up to O(n)

• Reply • Share ›

**krishna** • 10 months ago

can any one explain what exactly happing heare...?

```
c[0] = 0;
for (j = 1; j <= n; j++)
{
c[j] = INF;
for (i = 1; i <= j; i++)
{
if (c[i-1] != INF && lc[i][j] != INF && (c[i-1] + lc[i][j] < c[j]))
{
c[j] = c[i-1] + lc[i][j];
p[j] = i;
}
}
}
```

7 ^ | v • Reply • Share ›

**AlienOnEarth** • a year ago

**@GeeksforGeeks** Can you please put some example with values for all the arrays used. This code is quite confusing.

6 ^ | v • Reply • Share ›

**dave** • a year ago

your code doesn't work optimally with this test data {1 ,2 ,3, 1, 1} M=6 the ans is suppose to be 1 to 2 then 3 to 5

^ | v • Reply • Share ›

**garvit** ➦ dave • 10 months ago

How can word 3 to 5 be placed on a single line?
M=6,but for 3 to 5 u would need 3+1+1+2 =7 characters

^ | v • Reply • Share ›

**dave** ➦ dave • a year ago

ive made changes and have fixed it ..... if u want u would let me know if to post the sol

^ | v • Reply • Share ›

**Rachel** • a year ago

what is INF?

^ | v • Reply • Share ›

**rihansh** ➦ Rachel • 5 months ago

INFINITE but we should avoid using INT_MAX as INF this is because many a times we don't think and perform some operation with these infinites which results into overflow ..

^ | v • Reply • Share ›

**Dipankar Jana** ➦ Rachel • 7 months ago

INT_MAX
Look at the define statement

^ | v • Reply • Share ›

**Kartik** Rachel • a year ago
"Infinite" :)
3 • Reply • Share ›

**Anonym** • a year ago
India
2 • Reply • Share ›

**its_dark** • a year ago
It has been written ::
""The last line starts at word p[n] and goes through word n. The previous line starts at word p[p[n]] and goes through word p[n] – 1, etc."

shouldn't the previous line start at p[p[n]-1], instead of p[p[n]]??

In the example given above,
n=4, p[4]=4 ==> p[p[4]]=4 and the previous line doesn't start at 4th word obviously !

Correct me if I am wrong.
• Reply • Share ›

**raghson** • 2 years ago
In the explanation, it is mentioned that the previous line starts from p[p[n]] till p[n]-1. I guess it should be from p[p[n]-1] till p[n]-1.
• Reply • Share ›

**magnet** • 2 years ago
In solveWordwrap function after extras[n+1][n+1] in comment line it should be lc[i][j] instead of extras[i][j]..thnx :)
• Reply • Share ›

**GeeksforGeeks** magnet • 2 years ago
Thanks for pointing this out. We have updated the comment.
• Reply • Share ›

**abhishek08aug** • 2 years ago
Intelligent :D
• Reply • Share ›

**Kumar** • 2 years ago
In the greedy approach,

We can do some thing like that
1> Sort the words based on their length

2> run the above mentioned greedy algorithm

for eg: aaa bb cc ddddd

after step 1
ddddd aaa bb cc
after step 2
ddddd
aaa
bb cc

will it work ? Let me know your opinion or counter example.

```
/* Paste your code here (You may delete these lines if not writing code) */
```

Reply • Share ›

**Guest** Kumar • a year ago

you can't sort the words dude ! it is not allowed..you can't change the sequence of words !

2 • Reply • Share ›

**pavi** Kumar • 2 years ago

This will not work because the words are from a paragraph and sorting words and placing in lines will completely distort the meaning of paragraph.
Please see the question in CLRS for more detail.

Second way of looking at it is: Question says sequence of words. And mathematical definition of sequence means "order matters". Hence you cannot sort, 'coz that will break the sequence.

Hope this clarifies your doubt :)

1 • Reply • Share ›

**Kumar** pavi • 2 years ago

Thanks Pavi... I didn't realize the basic of Text Editor :D

```
/* Paste your code here (You may delete these lines if not writing code) */
```

Reply • Share ›

**aryan** • 3 years ago

How would you make sure that the line is also fully(left and right) justified ?

Reply • Share ›

**kartik** aryan • 3 years ago

Kartik › aryan • 3 years ago

I can think of following two step process to justify left and right.
1) Run the above given DP algo to balance extra spaces among lines.
2) Now for each line equally distribute the extra spaces between words.

• Reply • Share ›

Subscribe Add Disqus to your site Privacy

- 
- 
- 
- 
  - Interview Experiences
  - Advanced Data Structures
  - Dynamic Programming
  - Greedy Algorithms
  - Backtracking
  - Pattern Searching
  - Divide & Conquer
  - Mathematical Algorithms
  - Recursion
  - Geometric Algorithms
- 
- **Popular Posts**
  - All permutations of a given string
  - Memory Layout of C Programs
  - Understanding "extern" keyword in C
  - Median of two sorted arrays
  - Tree traversal without recursion and without stack!
  - Structure Member Alignment, Padding and Data Packing
  - Intersection point of two Linked Lists

  - Lowest Common Ancestor in a BST.
  - Check if a binary tree is BST or not
  - Sorted Linked List to Balanced BST
- Follow @GeeksforGeeks

- ## Recent Comments

  - lt_k

    i need help for coding this function in java...

    Java Programming Language · 1 hour ago
  - Piyush

    What is the purpose of else if (recStack[*i])...

    Detect Cycle in a Directed Graph · 1 hour ago
  - Andy Toh

    My compile-time solution, which agrees with the...

    Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 1 hour ago
  - lucy

    because we first fill zero in first col and...

    Dynamic Programming | Set 29 (Longest Common Substring) · 2 hours ago
  - lucy

    @GeeksforGeeks i don't n know what is this long...

    Dynamic Programming | Set 28 (Minimum insertions to form a palindrome) · 3 hours ago
  - manish

    Because TAN is not a subsequence of RANT. ANT...

    Given two strings, find if first string is a subsequence of second · 3 hours ago
-