Tutoria English Tutorials ~ Interviews ~

Word Wrap Problem **Word Wrap Problem** 

Home » Technical Interview Questions » Dynamic Programming Interview Questions »

### Difficulty Level Hard

```
Microsoft
Frequently asked in Arcesium
                                       GreyOrange
                              Factset
                                                               Myntra
Ola Cabs
          PayU
      Dynamic Programming
Table of Contents
                             =÷
Problem Statement
Example
Approach for word wrap problem
Code
   C++ Code for word wrap problem
   Java Code for word wrap problem
Complexity Analysis
   Time Complexity: O(n^2)
   Space Complexity: O(n^2)
Problem Statement
```

#### breaks to make the document look nice. Here, by nice we mean we place spaces in an evenly spread manner. There should not be lines with many extra spaces and some

with small amounts.

Here, we also assume that our word length does not exceed the line size. Just to make things a bit more generic we are considering a const function in the question as (Number of extra space)<sup>^3</sup>, else it would have been too easy. This cost function may vary as per the question asked.

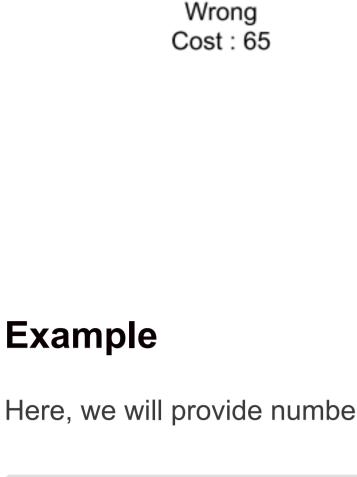
The word wrap problem states that given a sequence of words as input, we need to

find the number of words that can be fitted in a single line at a time. So, for doing this

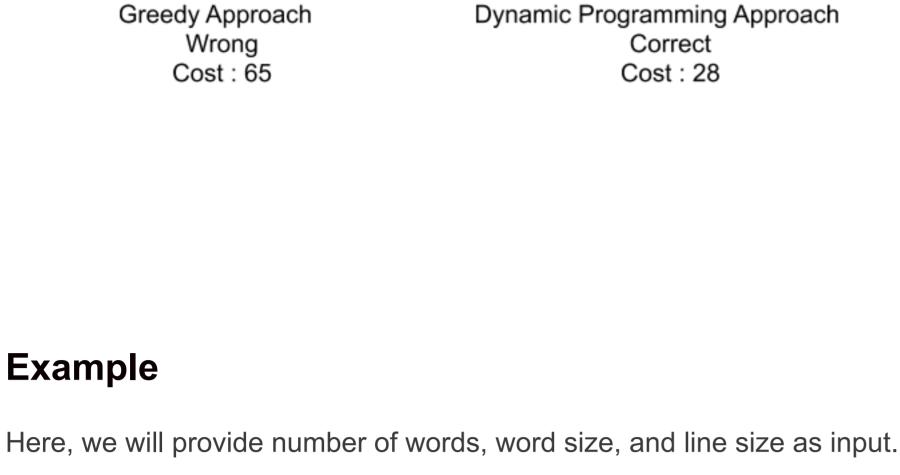
we put breaks in the given sequence such that the printed document looks nice. The

word editors such as Microsoft Office, Libre Office, and other document tools use line

Cat is is an an animal animal



Greedy Approach



#### on 2nd line with 1 extra space, and the last word on 3rd line with no extra space (we won't consider the spaces after the last word as extra spaces). Thus, using our cost

Input

number of words = 3

lineSize = 10

Output

00

Input

65

animal

Code

wordSize =  $\{ 1, 1, 1 \}$ 

Input

number of words = 4

lineSize = 6

Output

28

wordSize =  $\{3, 2, 2, 5\}$ 

function we find cost as 28.

Explanation: We can place 1st word on 1st line with 3 extra space, 2nd and 3rd word

```
0.
Approach for word wrap problem
The first approach which comes to mind is a greedy solution where we simply keep on
placing the words in a single line. When we cannot place a word on the same line, we
move to the second line. This seems to work just fine, but there's a catch. This
algorithm will not produce the optimal result because there may be cases such that if
```

we change the extra spaces we may end up with a better global solution.

Explanation: Here we can place all words on the same line i.e. 1st line and thus cost =

Output

Just for simple understanding, we have shown the input as words instead of wordSize.

Here, there are 4 extra spaces on the second line and 1 on the third line.

# Explanation: cat is\_

an\_\_\_\_

animal

Calculation:  $4^3+1^3 = 65$ 

Output using Greedy Approach

"cat is an animal", line size = 6

```
There exists a better solution,
cat___
is an_
```

Now, we will solve the word wrap problem using Dynamic Programming. We know that

our problem needs a global optimum solution and our previous algorithm was trying to

give us local optimum as a result. Here we will find the extra space taken in each line,

extraSpace which will tell the extraSpace left in a line, of words from i to j are laced on

a single line. Then further we will use this extraSpace matrix to find the minimum cost.

and will thus find the cost for each row respectively. We will maintain a matrix

Code using namespace std;

C++ Code for word wrap problem

cout<<ans<<endl;</pre>

Input

3

int wordWrap (int wordSize[], int n, int lineSize)

Giving us a total cost of  $28 (3^3 + 1^3 + 0^3)$ , which is better than 65.

```
int extraSpace[n+1][n+1];
 int minCost[n+1];
 for (int i=1;i<=n;i++)</pre>
   extraSpace[i][i] = lineSize - wordSize[i-1];
   for (int j=i+1;j<=n;j++)</pre>
        extraSpace[i][j] = extraSpace[i][j-1] - wordSize[j-1] - 1;
  minCost[0] = 0;
 for (int i = 1; i <= n; i++)</pre>
   minCost[i] = INT MAX;
   for (int j = 1; j <= i; j++)
        int cost; // stores cost of storing words[i,j] in single line
        if(extraSpace[j][i]<0)cost = INT MAX;</pre>
        else if(i==n && extraSpace[j][i]>=0)cost = 0;
        else cost = extraSpace[j][i]*extraSpace[j][i]*extraSpace[j][i];
      if (minCost[j-1] != INT_MAX && cost != INT_MAX
      && (minCost[j-1] + cost < minCost[i]))
        minCost[i] = minCost[j-1] + cost;
 return minCost[n];
int main()
    int t;cin>>t;
    while(t--) {
       int n;cin>>n;
       int wordSize[n];
       for(int i=0;i<n;i++)</pre>
            cin>>wordSize[i];
       int lineSize; cin>>lineSize;
       int ans = wordWrap(wordSize, n, lineSize);
```

```
3 2 2 6
 3
 1 1 1
 10
 1 1 1 1
 5
 Output
 28
 0
 0
Java Code for word wrap problem
 Code
  import java.util.*;
  import java.io.*;
 public class Main
   static int wordWrap (int wordSize[], int n, int lineSize)
      int extraSpace[][] = new int[n+1][n+1];
     int minCost[] = new int[n+1];
      for (int i=1;i<=n;i++)</pre>
       extraSpace[i][i] = lineSize - wordSize[i-1];
       for (int j=i+1;j<=n;j++)</pre>
            extraSpace[i][j] = extraSpace[i][j-1] - wordSize[j-1] - 1;
      }
        minCost[0] = 0;
        for (int i = 1; i <= n; i++)</pre>
          minCost[i] = Integer.MAX VALUE;
         for (int j = 1; j <= i; j++)
              int cost; // stores cost of storing words[i,j] in single line
              if(extraSpace[j][i]<0)cost = Integer.MAX VALUE;</pre>
              else if(i==n && extraSpace[j][i]>=0)cost = 0;
```

else cost = extraSpace[j][i]\*extraSpace[j][i]\*extraSpace[j][i];

if (minCost[j-1] != Integer.MAX VALUE && cost != Integer.MAX\_VALUE

&& (minCost[j-1] + cost < minCost[i]))

minCost[i] = minCost[j-1] + cost;

return minCost[n];

public static void main(String args[])

int t = sc.nextInt();

int n = sc.nextInt();

for(int i=0;i<n;i++)</pre>

int wordSize[] = new int[n];

int lineSize = sc.nextInt();

System.out.println(ans);

wordSize[i] = sc.nextInt();

int ans = wordWrap(wordSize, n, lineSize);

while (t-- > 0) {

```
Input
 3 2 2 6
 6
 3
 1 1 1
 10
 1 1 1 1
 5
 Output
 28
 0
Complexity Analysis
Time Complexity: O(n^2)
Here, since our outer loop runs from 1 to n and our inner loop runs from 1 to i, we have
```

## **Space Complexity: O(n^2)** Here, our extraSpace array is a 2D array which is of size N\*N, which gives us

polynomial space complexity of O(N^2).

> Maximum weight transformation of a given string

a polynomial time complexity of O(n^2).

```
Dynamic Programming Interview Questions
Arcesium, Dynamic Programming, Factset, GreyOrange, Hard, Microsoft, Myntra, Ola Cabs, PayU
< The Painter's Partition Problem</p>
```

```
© TutorialCup 2021 | Feeds | Privacy Policy | Terms | Contact Us | Linkedin | About Us
```

Search

Scanner sc = new Scanner(System.in);