1 Money Change Again

As we already know, a natural greedy strategy for the change problem does not work correctly for any set of denominations. For example, if the available denominations are $1, 3$, and 4, the greedy algorithm will change 6 cents using three coins $(4 + 1 + 1)$ while it can be changed using just two coins $(3 + 3)$. Your goal now is to apply dynamic programming for solving the Money Change Problem for denominations $1, 3$, and 4.

Problem Description

Input Format. Integer $money$.

Output Format. The minimum number of coins with denominations $1, 3$, and 4 that changes $money$.

Constraints. $1 \leq money \leq 10^3$.

Sample 1.

Input:

```
2
```

Output:

```
2
```

$2 = 1 + 1$

Sample 2.

Input:

```
34
```

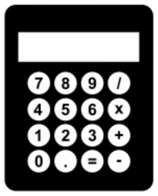
Output:

```
9
```

$34 = 3 + 3 + 4 + 4 + 4 + 4 + 4 + 4 + 4.$

2 Primitive Calculator

Problem Introduction



You are given a primitive calculator that can perform the following three operations with the current number x: multiply x by 2, multiply x by 3, or add 1 to x. Your goal is given a positive integer n, find the minimum number of operations needed to obtain the number n starting from the number 1.

Problem Description

Task. Given an integer n, compute the minimum number of operations needed to obtain the number n starting from the number 1.

Input Format. The input consists of a single integer $1 \leq n \leq 10^6$.

Output Format. In the first line, output the minimum number k of operations needed to get n from 1. In the second line output a sequence of intermediate numbers. That is, the second line should contain positive integers $a_0, a_1, \dots, a_k$ such that $a_0 = 1, a_k = n$ and for all $0 \leq i < k$, a_{i+1} is equal to either $a_i + 1, 2a_i$, or $3a_i$. If there are many such sequences, output any one of them.

Sample 1.

Input:

```
1
```

Output:

```
0
1
```

Sample 2.

Input:

```
5
```

Output:

```
3
1 2 4 5
```

Here, we first multiply 1 by 2 two times and then add 1. Another possibility is to first multiply by 3 and then add 1 two times. Hence “1 3 4 5” is also a valid output in this case.

Sample 3.

Input:

```
96234
```

Output:

```
14
1 3 9 10 11 22 66 198 594 1782 5346 16038 16039 32078 96234
```

Again, another valid output in this case is “1 3 9 10 11 33 99 297 891 2673 8019 16038 16039 48117 96234”.

Starter Files

Going from 1 to n is the same as going from n to 1, each time either dividing the current number by 2 or 3 or subtracting 1 from it. Since we would like to go from n to 1 as fast as possible it is natural to repeatedly reduce n as much as possible. That is, at each step we replace n by $\min\{n/3, n/2, n-1\}$ (the terms $n/3$ and $n/2$ are used only when n is divisible by 3 and 2, respectively). We do this until we reach 1. This gives rise to the following algorithm and it is implemented in the starter files:

```
GreedyCalculator($n$):
$numOperations \leftarrow 0$
while $n > 1$:
  $numOperations \leftarrow numOperations + 1$
  if $n$ mod $3 = 0$:
    $n \leftarrow n/3$
  else if $n$ mod $2 = 0$:
    $n \leftarrow n/2$
  else:
    $n \leftarrow n - 1$
return $numOperations$
```

This seemingly correct algorithm is in fact incorrect. You may want to submit one of the starter files to ensure this. Hence in this case moving from n to $\min\{n/3, n/2, n-1\}$ is not *safe*.

3 Edit Distance

Problem Introduction

The edit distance between two strings is the minimum number of operations (insertions, deletions, and substitutions of symbols) to transform one string into another. It is a measure of similarity of two strings. Edit distance has applications, for example, in computational biology, natural language processing, and spell checking. Your goal in this problem is to compute the edit distance between two strings.

Problem Description

Task. The goal of this problem is to implement the algorithm for computing the edit distance between two strings.

Input Format. Each of the two lines of the input contains a single string consisting of lower case latin letters.

Constraints. The length of both strings is at least 1 and at most 100.

Output Format. Output the edit distance between the given two strings.

Sample 1.

Input:

```
ab
ab
```

Output:

```
0
```

Sample 2.

Input:

```
short
ports
```

Output:

```
3
```

An alignment of total cost 3:

s	h	o	r	t	–
–	p	o	r	t	s

Sample 3.

Input:

```
editing
distance
```

Output:

```
5
```

An alignment of total cost 5:

e	d	i	–	t	i	n	g	–
–	d	i	s	t	a	n	c	e

4 Longest Common Subsequence of Two Sequences

Problem Introduction

Compute the length of a longest common subsequence of two sequences.

Problem Description

Task. Given two sequences $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_m)$, find the length of their longest common subsequence, i.e., the largest non-negative integer p such that there exist indices $1 \le i_1 < i_2 < \cdots < i_p \le n$ and $1 \le j_1 < j_2 < \cdots < j_p \le m$, such that $a_{i_1} = b_{j_1}, \ldots, a_{i_p} = b_{j_p}$.

Input Format. First line: n. Second line: $a_1, a_2, \ldots, a_n$. Third line: m. Fourth line: $b_1, b_2, \ldots, b_m$.

Constraints. $1 \le n, m \le 100$; $-10^9 < a_i, b_i < 10^9$.

Output Format. Output p.

Sample 1.

Input:

```
3
2 7 5
2
2 5
```

Output:

```
2
```

A common subsequence of length 2 is (2,5).

Sample 2.

Input:

```
1
7
4
1 2 3 4
```

Output:

```
0
```

The two sequences do not share elements.

Sample 3.

Input:

```
4
2 7 8 3
4
5 2 8 7
```

Output:

```
2
```

One common subsequence is (2,7). Another one is (2,8).

5 Maximum Amount of Gold

Problem Introduction

You are given a set of bars of gold and your goal is to take as much gold as possible into your bag. There is just one copy of each bar and for each bar you can either take it or not (hence you cannot take a fraction of a bar).

Problem Description

Task. Given n gold bars, find the maximum weight of gold that fits into a bag of capacity W.

Input Format. The first line of the input contains the capacity W of a knapsack and the number n of bars of gold. The next line contains n integers $w_0, w_1, \ldots, w_{n-1}$ defining the weights of the bars of gold.

Constraints. $1 \leq W \leq 10^4$; $1 \leq n \leq 300$; $0 \leq w_0, w_1, \ldots, w_{n-1} \leq 10^5$.

Output Format. Output the maximum weight of gold that fits into a knapsack of capacity W.

Sample 1.

Input:

```
10 3
1 4 8
```

Output:

```
9
```

Here, the sum of the weight of the first and the last bar is equal to 9.

Starter Files

Starter files contain an implementation of the following greedy strategy: scan the list of given bars of gold and add the current bar if it fits into the current capacity (note that, in this problem, all items have the same value per unit of weight, for a simple reason: they are all made of gold). As you already know from the lectures, such a greedy move is not *safe*.

6 Partitioning Souvenirs

You and two of your friends have just returned back home after visiting various countries. Now you would like to evenly split all the souvenirs that all three of you bought.

Problem Description

Input Format. The first line contains an integer n. The second line contains integers $v_1, v_2, \dots, v_n$ separated by spaces.

Constraints. $1 \leq n \leq 20$, $1 \leq v_i \leq 30$ for all i.

Output Format. Output 1, if it is possible to partition $v_1, v_2, \dots, v_n$ into three subsets with equal sums, and 0 otherwise.

Sample 1.

Input:

```
4
3 3 3 3
```

Output:

```
0
```

Sample 2.

Input:

```
1
40
```

Output:

```
0
```

Sample 3.

Input:

```
11
17 59 34 57 17 23 67 1 18 2 59
```

Output:

```
1
```

$34 + 67 + 17 = 23 + 59 + 1 + 17 + 18 = 59 + 2 + 57.$

Sample 4.

Input:

```
13
1 2 3 4 5 5 7 7 8 10 12 19 25
```

Output:

```
1
```

$1 + 3 + 7 + 25 = 2 + 4 + 5 + 7 + 8 + 10 = 5 + 12 + 19.$

7 Maximum Value of an Arithmetic Expression

Problem Introduction

In this problem, your goal is to add parentheses to a given arithmetic expression to maximize its value.

$$\max(5 - 8 + 7 \times 4 - 8 + 9) = ?$$

Problem Description

Task. Find the maximum value of an arithmetic expression by specifying the order of applying its arithmetic operations using additional parentheses.

Input Format. The only line of the input contains a string s of length $2n + 1$ for some n, with symbols $s_0, s_1, \dots, s_{2n}$. Each symbol at an even position of s is a digit (that is, an integer from 0 to 9) while each symbol at an odd position is one of the three operations from `{+,-,*}`.

Constraints. $0 \leq n \leq 14$ (hence the string contains at most 29 symbols).

Output Format. Output the maximum possible value of the given arithmetic expression among different orders of applying arithmetic operations.

Sample 1.

Input:

```
1+5
```

Output:

```
6
```

Sample 2.

Input:

```
5-8+7*4-8+9
```

Output:

```
200
```

$200 = (5 - ((8 + 7) \times (4 - (8 + 9))))$