Assignment Day-2

Kunal Sudhir Mishra

```
Qn-1
Given an input string and a dictionary of words, find out if the input
string can be segmented into a space-separated sequence of dictionary
words. See following examples for more details.
This is a famous Google interview question, also being asked by many other
companies now a days.
Consider the following dictionary
{ i, like, sam, sung, samsung, mobile, ice,
  cream, icecream, man, go, mango}
Input:  ilike
Output: Yes
The string can be segmented as "i like".
Input:  ilikesamsung
Output: Yes
The string can be segmented as "i like samsung"
or "i like sam sung".
```

In [5]:

```python
def word_break(s, word_dict):
    n = len(s)
    # Initialize a table to store the segmentation information
    dp = [False] * (n + 1)
    dp[0] = True  # Empty string can always be segmented

    for i in range(1, n + 1):
        for j in range(i):
            # Check if the substring s[j:i] can be segmented and if s[j:i] is
            if dp[j] and s[j:i] in word_dict:
                dp[i] = True
                break

    return dp[n]

input_str1 = "ilike"
output1 = word_break(input_str1, word_dict)
print(f"Output for '{input_str1}': {'Yes' if output1 else 'No'}")

input_str2 = "ilikesamsung"
output2 = word_break(input_str2, word_dict)
print(f"Output for '{input_str2}': {'Yes' if output2 else 'No'}")
```

```
Output for 'ilike': Yes
Output for 'ilikesamsung': Yes
```

```
Qn-2
```

A number can always be represented as a sum of squares of other numbers. Note that 1 is a square and we can always break a number as (1*1 + 1*1 + 1*1 + …). *Given a number n, find the minimum number of squares that sum to X.*
Examples :
Input: n = 100
Output: 1
Explanation:
100 can be written as 10^2. Note that 100 can also be written as 5^2 + 5^2 + 5^2 + 5^2, but this representation requires 4 squares.
Input: n = 6
Output: 3

In [6]:
```
def min_squares(n):

    dp = [float('inf')] * (n + 1)
    dp[0] = 0

    for i in range(1, n + 1):
        j = 1
        while j * j <= i:
            dp[i] = min(dp[i], dp[i - j*j] + 1)
            j += 1

    return dp[n]


n1 = 100
output1 = min_squares(n1)
print(f"The minimum number of squares for {n1} is {output1}.")

n2 = 6
output2 = min_squares(n2)
print(f"The minimum number of squares for {n2} is {output2}.")
```

```
The minimum number of squares for 100 is 1.
The minimum number of squares for 6 is 3.
```

Qn-3
Given a number N, the task is to check if it is divisible by 7 or not.
Note: You are not allowed to use the modulo operator, floating point arithmetic is also not allowed.
Naive approach: A simple method is repeated subtraction. Following is another interesting method.
Divisibility by 7 can be checked by a recursive method. A number of the form 10a + b is divisible by 7 if and only if a – 2b is divisible by 7. In other words, subtract twice the last digit from the number formed by the remaining digits. Continue to do this until a small number.
Example: the number 371: 37 – (2×1) = 37 – 2 = 35; 3 – (2 × 5) = 3 – 10 = -7; thus, since -7 is divisible by 7, 371 is divisible by 7.

In [11]:
```
def min_squares(n):

    dp = [float('inf')] * (n + 1)
    dp[0] = 0


    for i in range(1, n + 1):
        j = 1
        while j * j <= i:

            square = j * j
            if square <= i:
                dp[i] = min(dp[i], dp[i - square] + 1)
            j += 1

    return dp[n]

n1 = 100
output1 = min_squares(n1)
print(f"The minimum number of squares for {n1} is {output1}.")

n2 = 6
output2 = min_squares(n2)
print(f"The minimum number of squares for {n2} is {output2}.")
```

```
The minimum number of squares for 100 is 1.
The minimum number of squares for 6 is 3.
```

Qn-4
Find the n’th term in Look-and-say (Or Count and Say) Sequence. The look-and-say sequence is the sequence of the below integers:
1, 11, 21, 1211, 111221, 312211, 13112221, 1113213211, …
How is the above sequence generated?
n’th term is generated by reading (n-1)’th term.
The first term is "1"
Second term is "11", generated by reading first term as "One 1"
(There is one 1 in previous term)
Third term is "21", generated by reading second term as "Two 1"
Fourth term is "1211", generated by reading third term as "One 2 One 1"
and so on

In [12]:
```
def look_and_say(n):
    if n == 1:
        return "1"

    prev_term = "1"
    for _ in range(n - 1):
        new_term = ""
        count = 1

        for i in range(1, len(prev_term)):
            if prev_term[i] == prev_term[i - 1]:
                count += 1
            else:
                new_term += str(count) + prev_term[i - 1]
                count = 1

        new_term += str(count) + prev_term[-1]
        prev_term = new_term

    return prev_term


n = 6
result = look_and_say(n)
print(f"The {n}th term in the Look-and-Say sequence is: {result}")
```

```
The 6th term in the Look-and-Say sequence is: 312211
```

In [ ]: