- Rope Cutting problem: Worst case → $O(3^n)$, $O(n)$ space
  ↳ if we traverse till end by considering a, b, c = 1

① I/P → n = 5, a = 2, b = 5, c = 1.

O/P ⇒ 5

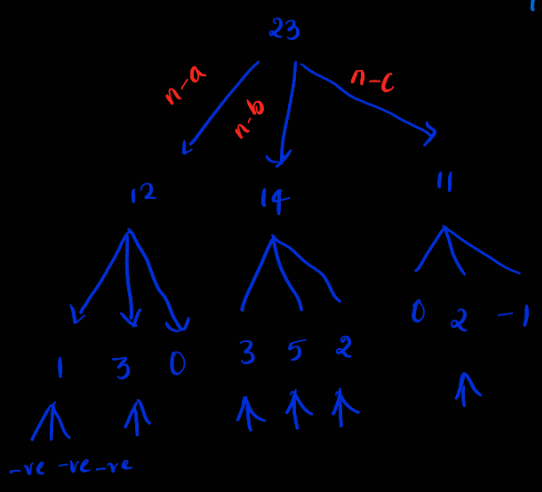
Explanation : we can make max of 5 pieces of 1 length.

② I/P → n = 23, a = 12, b = 9, c = 11

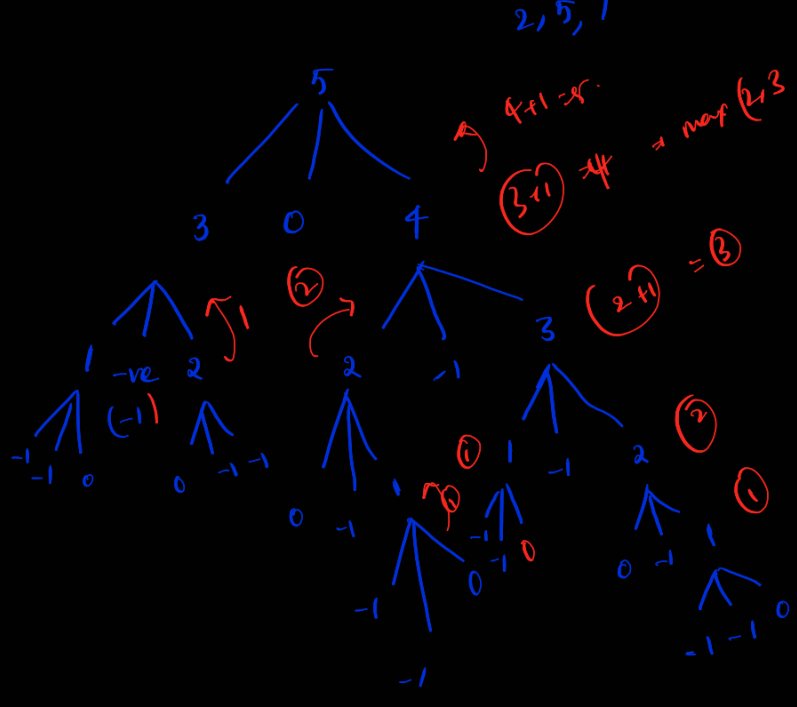
O/P → 2

Explanation : we can make 2 pieces of length 11 & 12

③ I/P → 9, 2, 2, 2
O/P ⇒ -1 } .. Corner Case ..



→ if n == 0:
  return 0
→ if n < 0:
  return -1
→ max (fun (n-a, a, b, c)
  fun (n-b, ,)
  fun (n-c,)



→ Add 1 to result
  → it's for count
  → initially it will be zero
  → as you make a cut we need to add 1 to it for considering that new cut way made

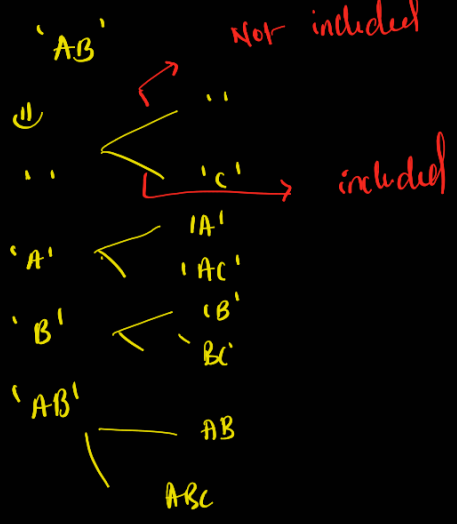
- Generate Subsets / subsequences of a string.

  I/p = 'A,B'

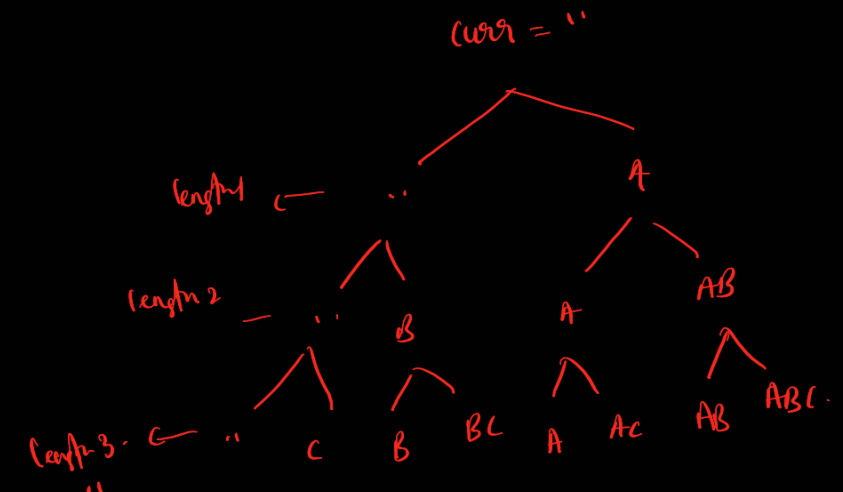
  o/p → '', 'A', 'B', 'AB'

- All char's in input string are distinct.

→ Consider 'AB'



subsets of ABC



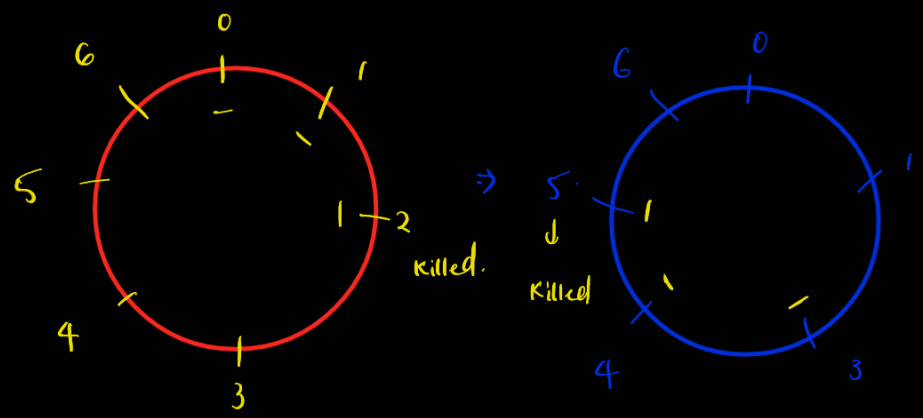
as we reached the str length we can print curr & return.

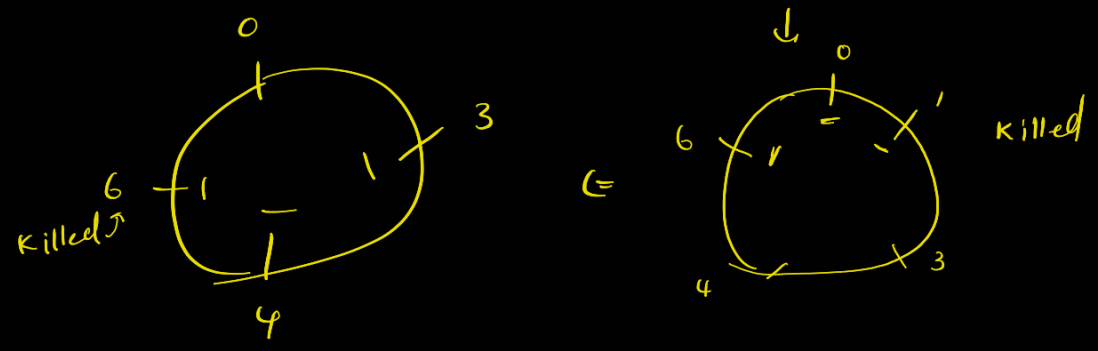
Josephus

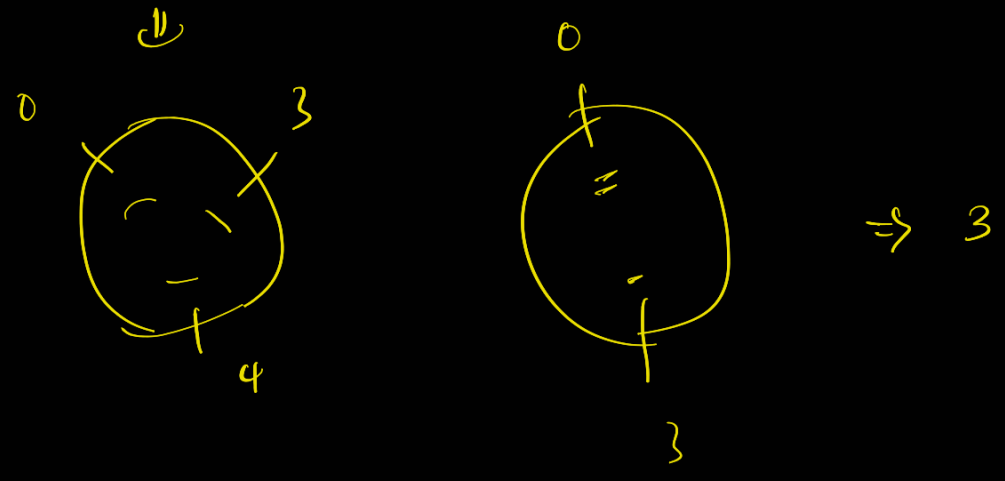
I/p ⇒ n = 7, k = 3 , o/p = 3.

↓ ↓

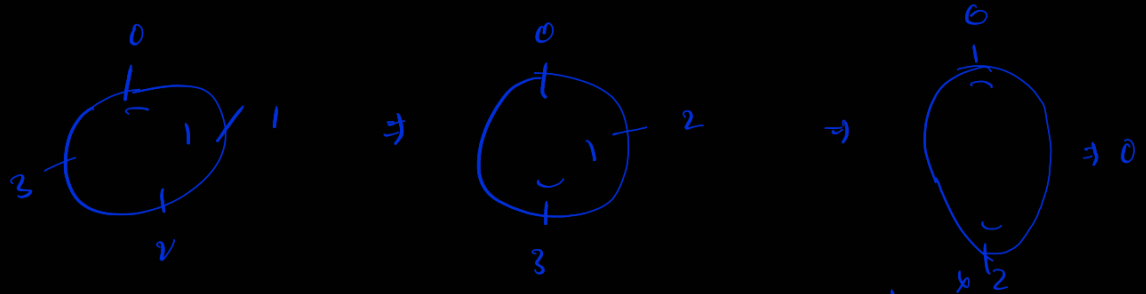
no. of persons in a circle. kill every 3rd person.

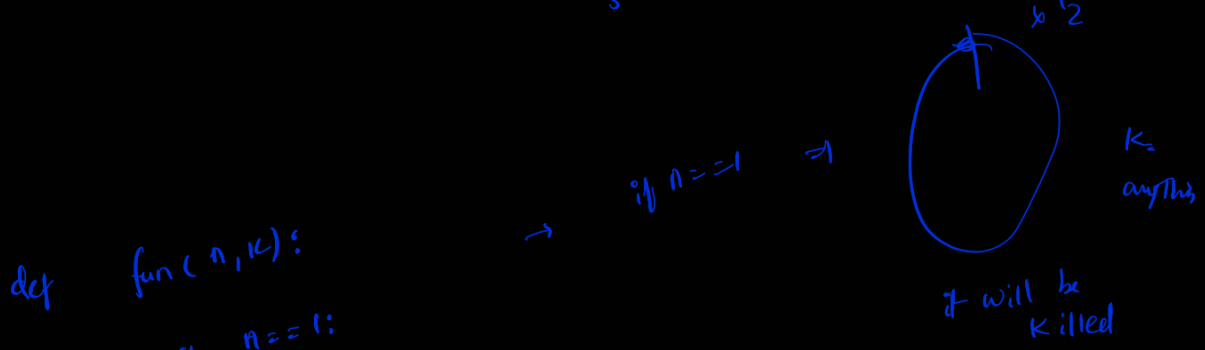






⇒ n = 4, K = 2

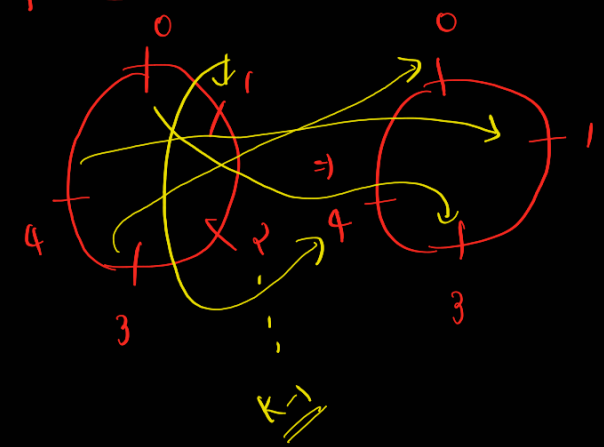





```
def fun(n,K):
    if n==1:
        return 0
    return fun(n-1, K)
```

↳ This will consider a fresh recursive call from 5, 4, 3 -- like this (fun(n-1,K)+K)

fun(5,3)
↓
fun(4,3)



⇒

(5,3)	(4,3)
3	0
4	1
0	2
1	3

n,K	n-1,K
K	0
K+1	1
K+2	2
K+3	3
⋮	⋮

But K+i may become more than n

n, n-1, ---, 1, 0

So we will just need the numbers from 0 to n

So we need to modulo Arithmatic

$$(fun(n-1, K) + K) \% n$$

Digital Root ..

I/P	1
O/P	1

99999

sum digits

45 = 9

sum them till we arrive at single digit

```
def fun(n):
    if n < 10:
        return n

    sum = 0
    while n > 0:
        sum += n%10
        n /= 10

    fun(sum)
```

Lucky numbers are subset of integers. Rather than going into much theory, let us see the process of arriving at lucky numbers,
Take the set of integers
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,.....
First, delete every **second** number, we get following reduced set.
1, 3, 5, 7, 9, 11, 13, 15, 17, 19,.....
Now, delete every **third** number, we get
1, 3, 7, 9, 13, 15, 19,.....
Continue this process indefinitely.....
Any number that does **NOT** get deleted due to above process is called "**lucky**".

You are given a number **N**, you need to tell whether the number is lucky or not. If the number is lucky return 1 otherwise 0.

Example 1:

Input:
N = 5
Output: 0
Explanation: 5 is not a lucky number as it gets deleted in the second iteration.

Example 2:

Input:
N = 19
Output: 1
Explanation: 19 is a lucky number because it does not get deleted throughout the process.

Your Task:
You don't need to read input or print anything. You only need to complete the function **isLucky()** that takes N as parameter and returns either False if the N is not lucky else True.

Expected Time Complexity: O(sqrt(N)).
Expected Auxiliary Space: O(1).

Constraints:
$1 <= N <= 10^5$

(5) → 1 2 3 4 5 (2)
(3) → 1 3 5 (3)

if n%.k == 0:
return False

19 (2)
→ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
12, 13, 14, 15, 16, 17, 18, 19

→ 1, 3, 5, 7, 9, 11, 13, 15, (3)
17, 19

→ 1, 3, 7, 9, 13, 15, 19 (4)

⇒ 1, 3, 7, 13, 15, 19 (5)

→ 1, 3, 7, 13, 19 (6)

BaseCase

if n < k:
return k.

fun(n - n//k , k+1)

we will remove these many nums as we go

if n%.k = 0

??

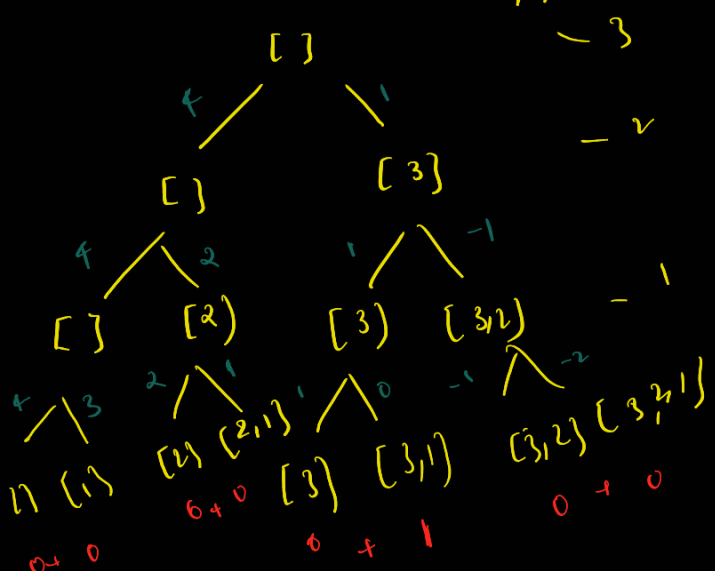
• Count subset sum : O (2^n)

10, 2015 , 25

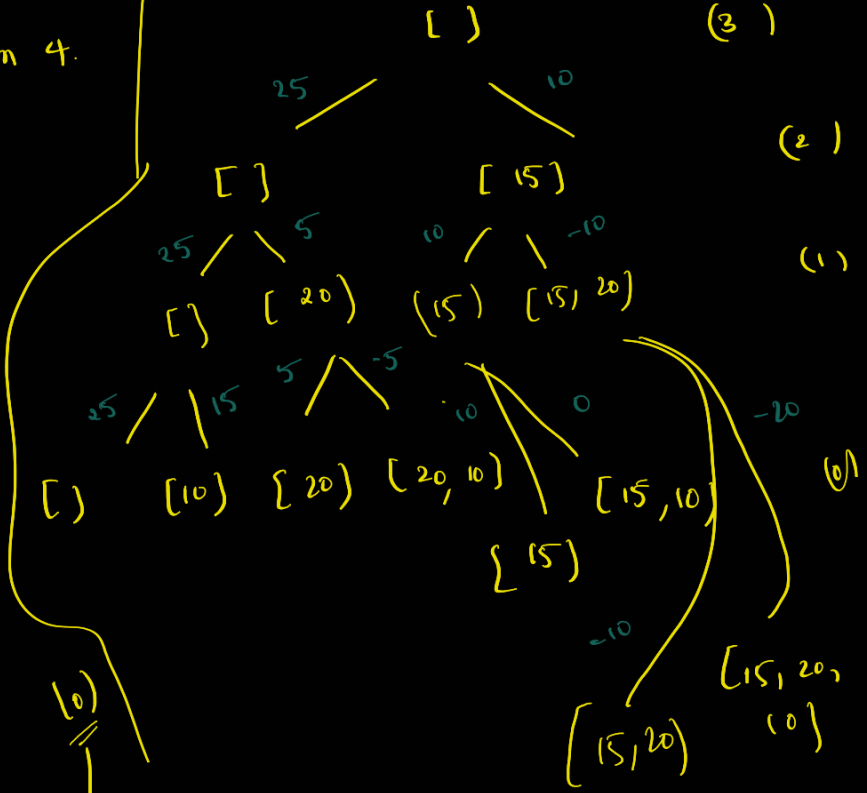
① [10, 5, 2, 3, 6] ⇒ Sum = 8

2 ways.

② [1, 2, 3] ⇒ Sum 4.



→ We can traverse from any direction i'm travelling from Right side using index



return included + excluded.

```
if n == 0:
    if sum == 0:
        return 1
    else:
        return 0
```

Combinational sum:

→ [2,3, 6,7] target = 7 $(2^t \times k)$.

[2,2,3) [7]

sum 7 7