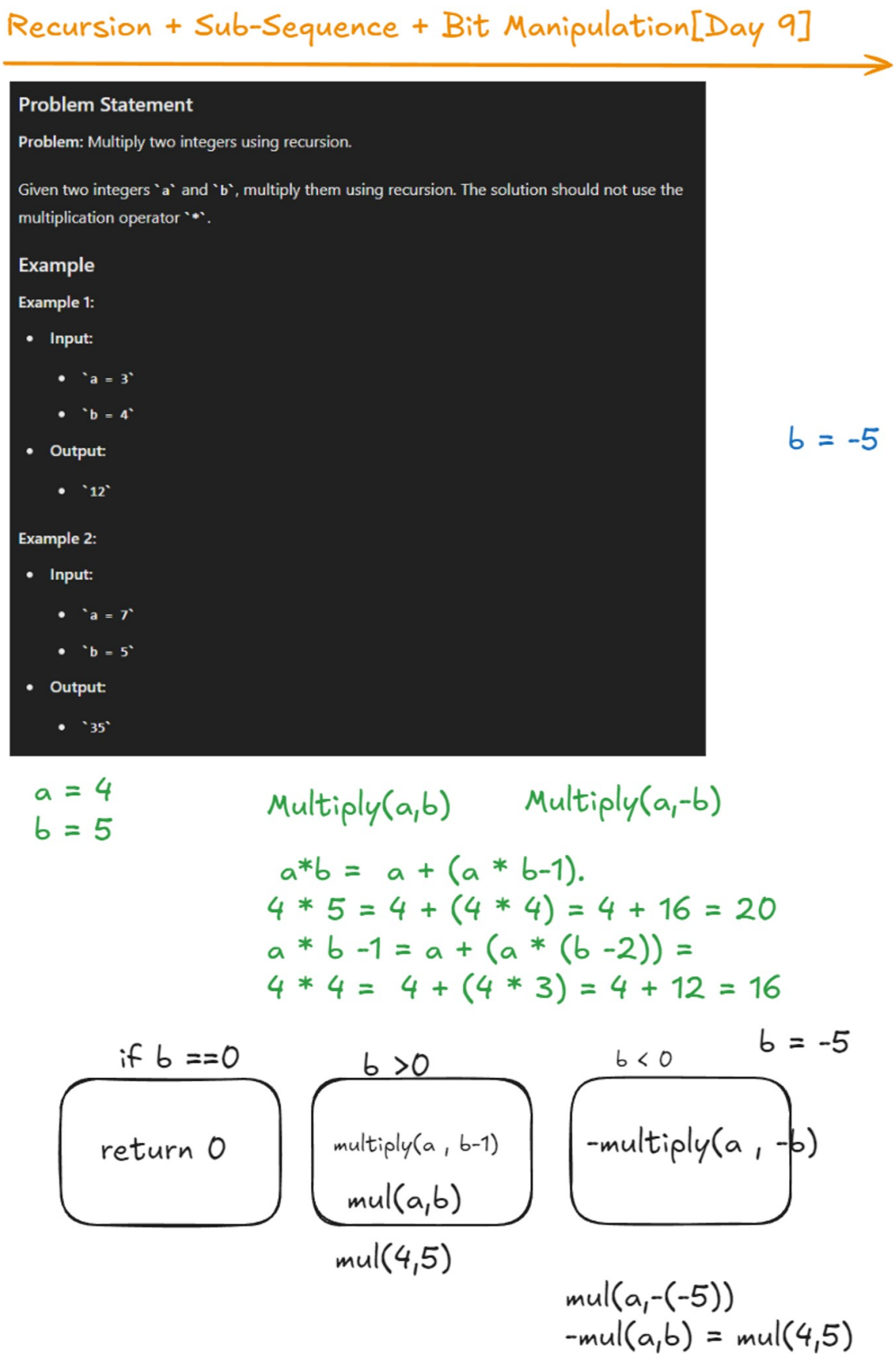
Recursion + Sub Sequence [Day 9]

11 September 2024 19:37



-mul(a,b) = mul(4,5)

```
import java.util.Scanner;
public class RecursionMultiplication {
    public static int multiply(int a , int b){
        if(b == 0){
            return 0;
        }
        if(b > 0){
            return a + multiply(a, b-1);
        }
        return - multiply(a , -b);
    }
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        System.out.println("Enter the first number(a): " );
        int a = scn.nextInt();
        System.out.println("Enter the Second number(b): " );
        int b = scn.nextInt();

        int result = multiply(a,b);
        System.out.println("Result of " + a + " * " + b + " = "
+ result);
    }
}
```

Problem Statement

Problem: The Josephus Problem

Given `n` people standing in a circle, where every `k`-th person is eliminated, find the position of the last person remaining (the safe position).

Example

Example 1:

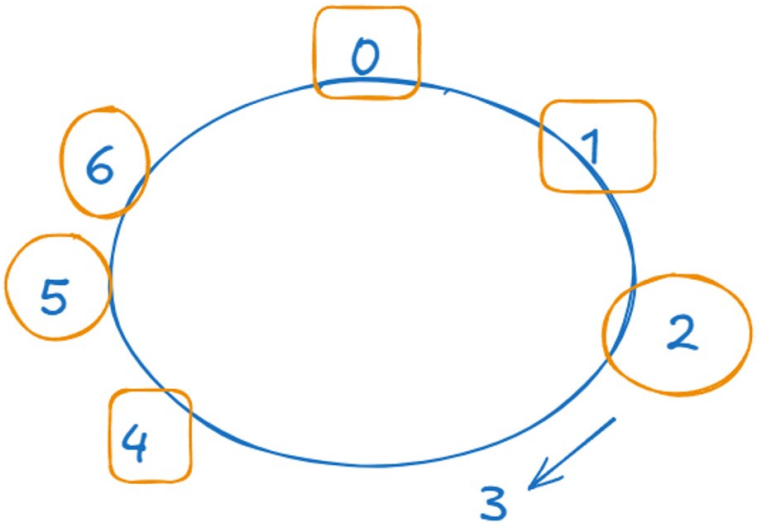
- **Input:**
  - `n = 7` (total people)
  - `k = 3` (every 3rd person is eliminated)
- **Output:**
  - `4` (the safe position)

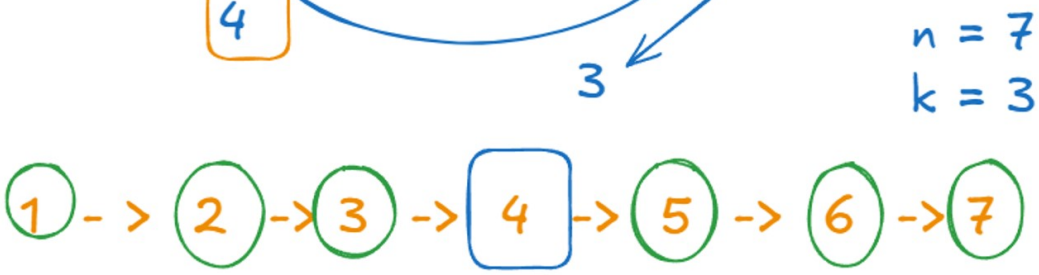
Example 2:

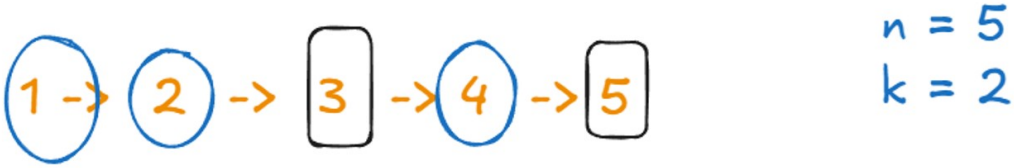
- **Input:**
  - `n = 5`
  - `k = 2`

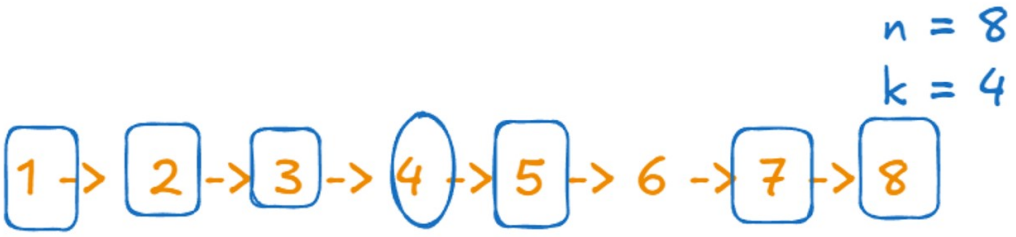
- `n = 5`
- `k = 2`

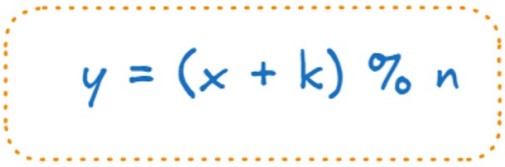
- Output:
  - `3`

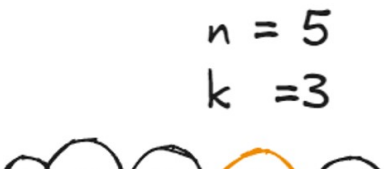


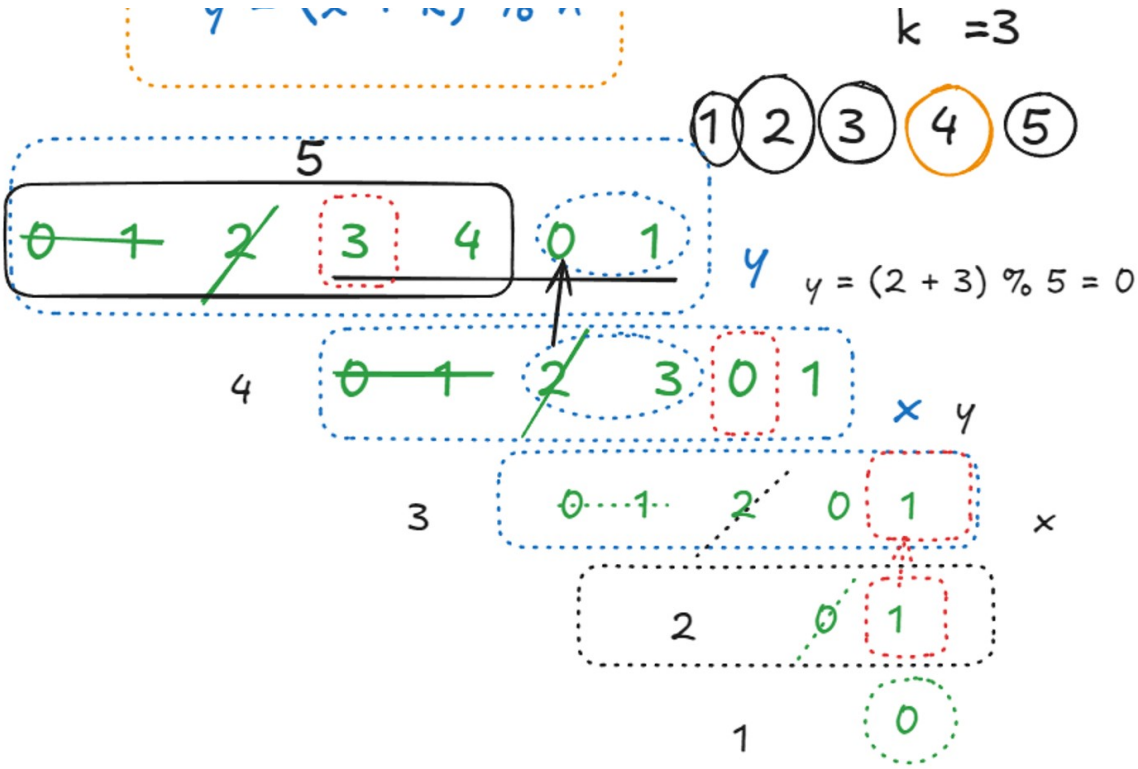












$y = (0 + 3) \% 5 = 3$

0 1 2 3 4 -> 1 2 3 4 5

```
import java.util.Scanner;
public class JosephusProblem {
    public static int josephus(int n , int k){
        if(n ==  1){
            return 0;
        }
        return (josephus(n-1, k) + k ) % n;
    }
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);

        System.out.println("Enter the number of people(n) :");
        int n = scn.nextInt();
        System.out.println("Enter the step count (k) :");
        int k = scn.nextInt();

        int safePosition = josephus(n , k) + 1;
        System.out.println("The safe position is : " + safePosition);

    }
}
```

```
Enter the number of people(n) :
5
```

```
Enter the number of people(n) :
5
Enter the step count (k) :
2
The safe position is : 3
```

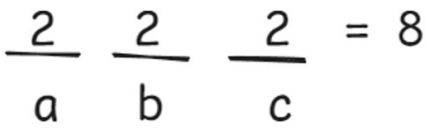
1. Print All Subsequences of a String

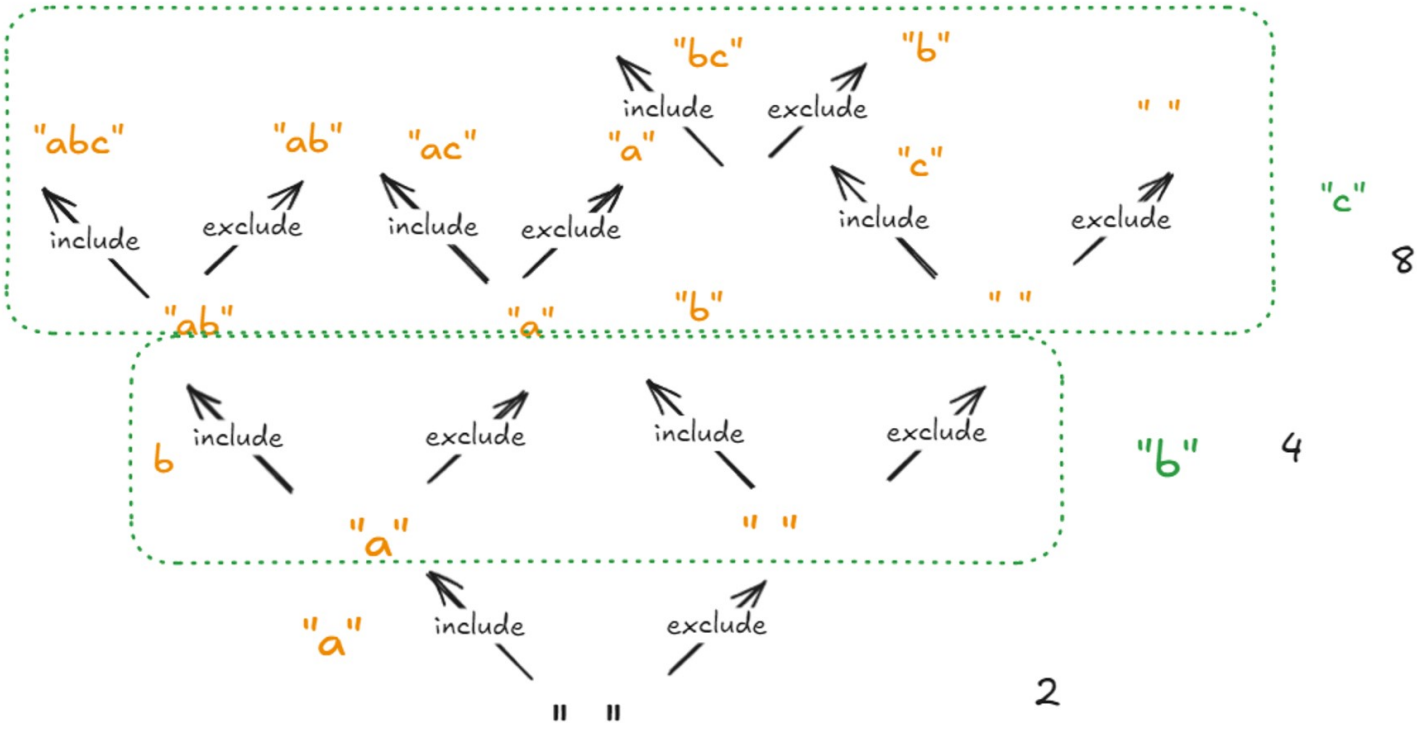
Problem Statement

Given a string `S`, print all possible subsequences of the string. A subsequence is a sequence that can be derived from another sequence by deleting some or no elements without changing the order of the remaining elements.

Examples

- **Input:** `S = "abc"`
- **Output:** `["", "a", "b", "c", "ab", "ac", "bc", "abc"]`





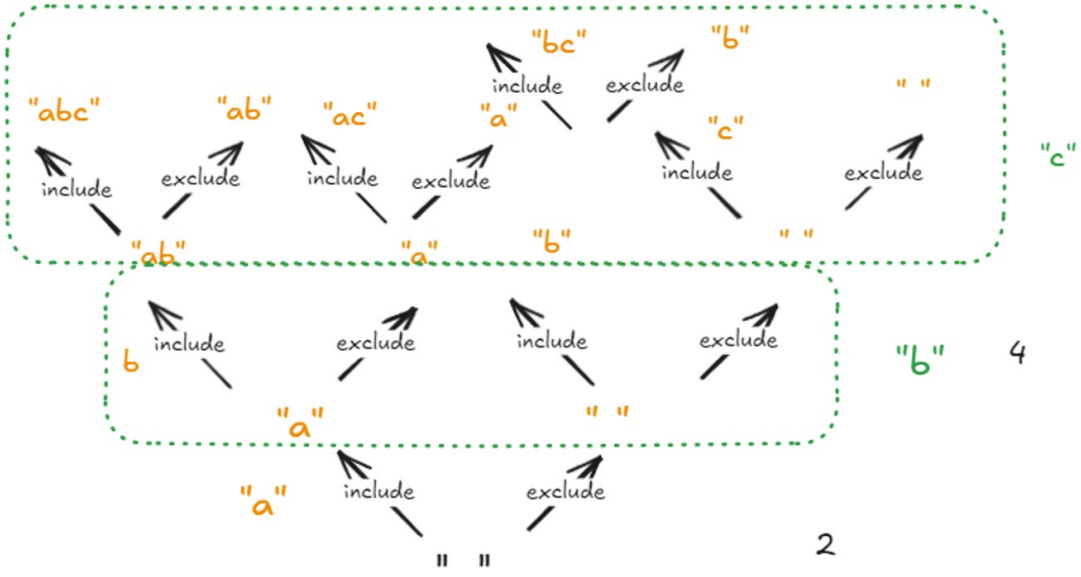

```
public class SubSequence {

    public static void printSubsequence(String str , int index , String current){
        if(index == str.length()){
            System.out.println(current);
            return;
```

```
    if(index == str.length()){
        System.out.println(current);
        return;
    }

    // Include the character
        printSubsequence(str, index + 1 , current + str.charAt(index));
        // Exclude the character
        printSubsequence(str, index + 1 , current);
    }
    public static void main(String[] args) {
        String str = "abc";
        printSubsequence(str , 0 , "");
    }
}
```

```
abc
ab
ac
a
bc
b
c
```




```
if(index == str.length()){
    System.out.println(current);
    return;
}

// Include the character
    printSubsequence(str, index + 1 , current + str.charAt(index));
    // Exclude the character
    printSubsequence(str, index + 1 , current);
```

abc , ab , ac , a , bc , b , c , " "

P.s("abc" , 3 , " ")

P.s("abc" , 2 , " ")

P.s("abc" , 1 , " ")

P.s("abc" , 0 , " ")

Call Stack

Call Stack

3. Longest Continuous Increasing Subsequence

Problem Statement

Given an unsorted array of integers `nums`, find the length of the longest continuous increasing subsequence (LCIS).

Examples

- **Input:** `nums = [1, 3, 5, 4, 7]`
- **Output:** `3`
- **Explanation:** The longest continuous increasing subsequence is `[1, 3, 5]` with length `3`.

max = 3
curr = 2

```
for (1 to <n){
    if(arr[i] > arr[i-1]){
        curr ++;
        max  = Math.max(max , curr);
    else curr = 1;
}
```

```
public class LCIS {
    public static int FindLengthOfLCIS(int[] nums){
        if(nums.length == 0){
            return 0;
        }
        int maxLength = 1;
        int currentLength = 1;
        for(int i = 1; i<nums.length; i++){
            if(nums[i] > nums[i-1]){
                currentLength++;
                maxLength = Math.max(maxLength, currentLength);
            }else{
                currentLength = 1;
            }
```

```
            }else{
                currentLength = 1;
            }
        }
        return maxLength;
    }
    public static void main(String[] args) {
        int[] nums = {1 , 3, 5, 4 , 7};
        System.out.println(FindLengthOfLCIS(nums));
    }
}
```

Completed

Attendance Code: B1C270B8

Problem Statement

Given an array of positive integers `nums` and a positive integer `target`, find the minimal length of a contiguous subarray of which the sum is greater than or equal to `target`. If no such subarray exists, return `0`.

Examples

- **Example 1:**
  - **Input:** `nums = [2, 3, 1, 2, 4, 3]`, `target = 7`
  - **Output:** `2`
  - **Explanation:** The subarray `[4, 3]` has the minimal length with a sum equal to `7`.
- **Example 2:**
  - **Input:** `nums = [1, 4, 4]`, `target = 4`
  - **Output:** `1`
  - **Explanation:** The subarray `[4]` has the minimal length with a sum of `4`.
- **Example 3:**
  - **Input:** `nums = [1, 1, 1, 1, 1, 1, 1, 1]`, `target = 11`
  - **Output:** `0`
  - **Explanation:** No subarray has a sum greater than or equal to `11`.

target = 7

2 3 1 2 4 3

minLength = 3