DATE 28-04-24

144

RECURSION

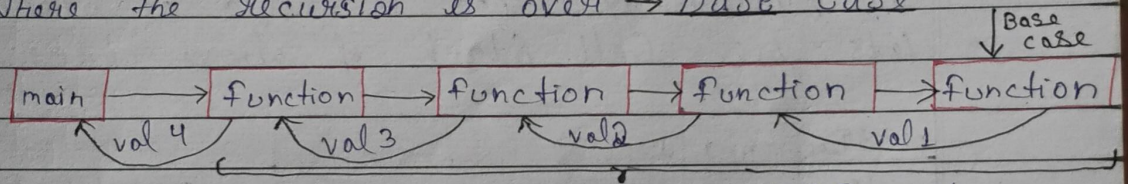
★ WHAT IS RECURSION?

* using Math,

$$f(x) = x^2$$
$$f(f(x)) = (x^2)^2$$
$$\Rightarrow x = 2$$
$$f(x) = 4 = 2^2$$
$$f(f(x)) = f(4) = 4^2 = 16.$$

function calling itself again & again.

* The function which call itself again & again is known as Recursive function.
* Recursion is a method of solving a computational problem where the solution depends on solutions to smaller instances of the same problem.
* Where the recursion is over → Base Case



* parameters are different.

QUES- Print numbers from n to 1 (decreasing Order).

* $f(n) = n + f(n-1)$

$\downarrow$

$10 + f(9)$

$\downarrow$

$9 + f(8)$

$\downarrow$

$8 + f(7)$

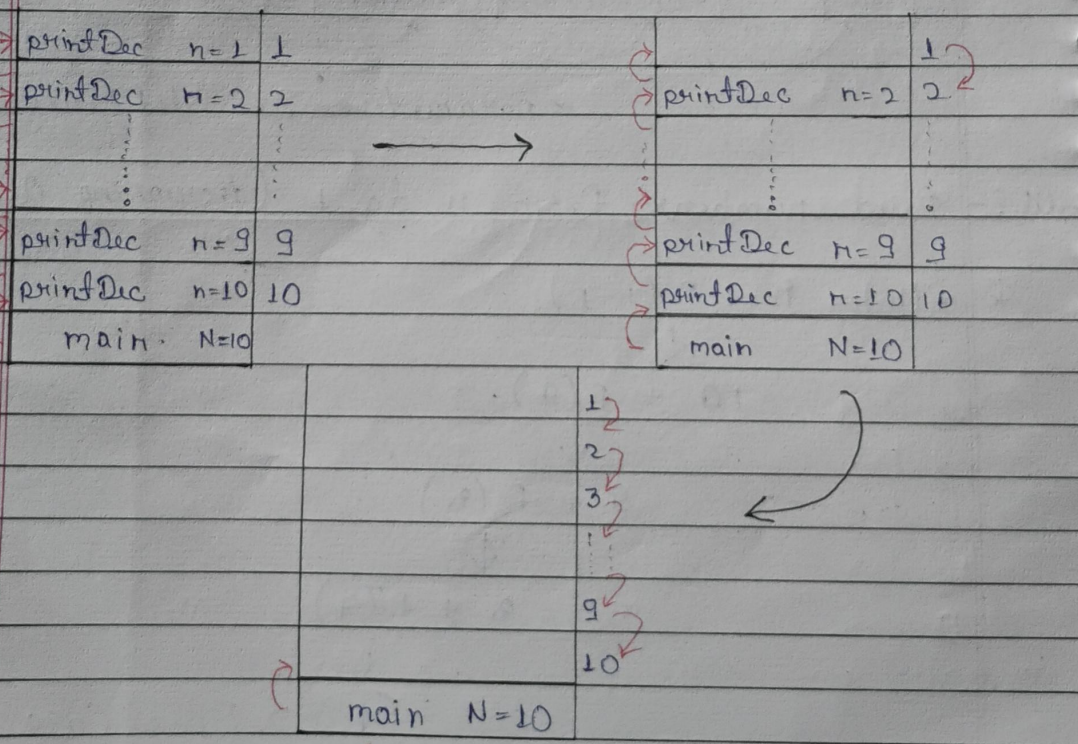
$\vdots$

1 → Base Case

⇒
```
public class Recursion {
    public static void printDec (int n) {
        if (n == 1) {
            System.out.println (n);
            return;
        }
        System.out.print (n + " ");
        printDec (n-1);
    }
    public static void main (String args[]) {
        int n = 10;
        printDec (n);
    }
}
```

10 9 8 7 6 5 4 3 2 1

* Any function or method is stored as Stack.
* Let's see Call Stack of above code:



There should be always a BASE CASE else STACK OVERFLOW

★ RECURSION VS ITERATION

	RECURSION	ITERATION
1.	It is the process of calling a function itself within its own code.	It is a repeated execution of the set of instructions.
2.	The termination condition is defined within the recursive function.	The termination condition is defined in the definition of the loop.
3.	Smaller code size.	Larger code size.
4.	It is always applied to functions.	It is faster than recursion.
5.	No issue of time complexity.	It is used when we have to balance time complexity.
6.	High time complexity.	Low time complexity.
7.	It has to update and maintain the stack.	No utilization of stack.
8.	It uses more memory as compared to iteration.	It uses less memory as compared to recursion.
9.	If the recursive function does not meet to a termination condition, it leads to an infinite recursion.	Iteration will be infinite, if the control condition of the iteration statement never becomes false.

QUES. Print numbers from n to 1 (increasing order).

⇒
```
public class Recursion {
    public static void printInc (int n) {
        if (n==1) {
            System.out.println (n+" ");
            return;
        }
        printInc (n-1);
        System.out.print (n+" ");
    }
    public static void main (String args[]) {
        int n=10;
        printInc (n);
    }
}
```

1 2 3 4 5 6 7 8 9 10

QUES. Print factorial of a number n.

$n! = n \times (n-1)! \longrightarrow f(n) = n \times f(n-1)$

$\downarrow$

$(n-1) \times f(n-1)$

$\vdots$

```
if (n==0)
    return 1; // 0! = 1.
```

* f_{n-1} = fnm1 //in code.

* We know,

$5! = 5 \times 4 \times 3 \times 2 \times 1 \times (1) \rightarrow 0!$

$= 120.$

⇒
```
public class Recursion {
    public static int fact (int n) {
        if (n==0) {
            return 1;                    // Time Complexity → O(n)
        }                                // Space Complexity → O(n)
        int fnm1 = fact (n-1);
        int fn = n * fact (n-1);
        return fn;
    }
    public static void main (String args[]) {
        int n = 5;
        System.out.println (fact (n));
    }
}
```

120

QUES. Print sum of first n natural numbers.

* $f(n) = n + f(n-1)$ → ∵ $f(5) = 5 + f(4)$ [10] $= 15$.

$4 + f(3)$ [6]

$3 + f(2)$ [3]

$2 + f(1)$ [1]

* Base case, if (n==1)
  return 1;

⇒
```
public class Recursion {
    public static int sum (int n) {
        if (n==1) {
            return 1;
        }
        int Snm1 = sum (n-1);    // Sn-1 = sum(n-1)
        int Sn = n + Snm1;       // Sn = n + Sn-1
        return Sn;               // return Sn
    }
    public static void main (String args[]);
        System.out.print (sum(5));
}
```

Internally जब दो Recursion call होती है तो Tree बनता है.

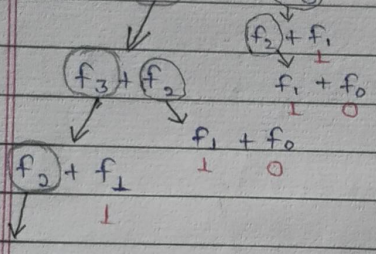
DATE

QUES. Print Nth fibonacci numbers.

0 1 1 2 3 5 8 13 21 .....

0th 1th 2nd 6th

* $f_n = f_{n-1} + f_{n-2}$



* Base case, $f(0) = 0$

* Base case, $f(1) = 1$.

* Algo :- fib (int n) {

if (n = 0) → 0

|| (n = 1) → 1

fib_{N-1} = fib (n-1)

fib_{N-2} = fib (n-2)

fib_N = fib_{N-1} + fib_{N-2}

return fib_N.

⇒

```
public class Recursion {
    public static int fib (int n) {
        if (n == 0 || n == 1) {
            return n;                    // Space Complexity, O(n)
        }                                // Time Complexity, O(2^n)
        int fnm1 = fib (n-1);            // Exponential time complexity
        int fnm2 = fib (n-2);
        int fn = fnm1 + fnm2;
        return fn;
    }
    public static void main (String args[]) {
        System.out.println (fib (10));
    }
}
```

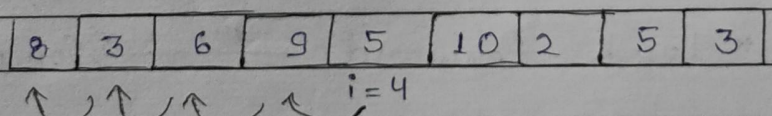
QUES. Check if a given array is sorted or not.

a = 1 2 3 4 5 $\Longrightarrow$ $Sorted_N = a[0] < a[1] + Sorted_{N-1}$

* Base case, i == arr.length - 1.

⇒
```
public class Recursion {
    public static boolean isSorted (int arr[], int i) {
        if (i == arr.length -1) {
            return true;
        }
        if (arr[i] > arr[i+1]) {
            return false;
        }
        return isSorted (arr, i+1);
    }
    public static void main (String args[]) {
        int arr[] = {1, 2, 3, 4, 5};
        System.out.println (isSorted (arr, 0));
    }
}
```

QUES. Write a recursive function to find the first occurence of an element in an array.



* Key = 5 return (4)
* 5 is first occuring at index 4.
* -1 if -1 index is returned so it means NOT FOUND.

⇒
```
Public class Recursion {
    public static int firstOccurence (int arr[], int key,
                                      int i) {
        if (i == arr.length) {
            return -1;
        }
        if (arr[i] == key) {
            return i;
        }
        return firstOccurence (arr, key, i+1);
    }
    public static void main (String args[]) {
        int arr[] = {8, 3, 6, 9, 5, 10, 2, 5, 3};
        System.out.println (firstOccurence (arr, key, i));
    }
}
```
5, 0

4

Ques. Write a function to find the last occurence of an element in an array.

8	3	6	9	5	10	2	5	3
				i=4			i=7	

* Key = 5 returns 7;
  * At i=7, 5 is the last occurence.
  * Look forward —then→ Compare
* Base case → if (i == arr.length)
  return -1;
  NOT FOUND

⇒
```
public class Recursion {
    public static int lastOccurence(int arr[], int key, int i) {
        if (i == arr.length) {
            return -1;
        }
        int isFound = lastOccurence(arr, key, i+1);
        if (isFound == -1 && arr[i] == key) {
            return i;
        }
        return isFound;
    }
    public static void main(String args[]) {
        int arr[] = {8, 3, 6, 9, 5, 10, 2, 5, 3};
        System.out.println(lastOccurence(arr, 5, 0));
    }
}
```

7

Ques. Print x to the power n. (x^n).

* $x^n \rightarrow x \times x^{n-1}$

$x \times x^{n-2}$

$x \times x^{n-3}$

$x \times x^{n-4}$

⋮

∴ ⇒ pow(x, n) = x × pow(x, n-1)

* x = 2, n = 10 → 2^{10} = 2×2×2...×2 (n times)

* Base case, if (n == 0) return 1;

```
public class Recursion {
    public static int power (int x, int n) {
        if (n == 0) {
            return 1;
        }
        int xnm1 = power (x, n-1);
        int xn = x * xnm1;
        return xn;                    // Time Complexity, O(n)
                  or
        return x * power (x, n-1);
    }
    public static void main (String args[]) {
        System.out.println (power (2, 10));
    }
}
```

1024

QUES. Print x^n in $O(\log n)$.

Case 1: $x^n \rightarrow n = even \rightarrow 2^{10} = x^{n/2} \times x^{n/2}$

$= 2^5 \times 2^5 = 2^{5+5} = 2^{10}$

Case 2: $x^n \rightarrow n = odd \rightarrow 2^5 = x \times x^{n/2} \times x^{n/2}$

$= 2 \times 2^2 \times 2^2 = 2^{1+2+2} = 2^5$

Ex → $n = 10$

$\downarrow \frac{n}{2}$ $\downarrow \frac{10}{2}$

$\downarrow \frac{n}{4}$ $\downarrow \frac{10}{4}$

$\vdots$ $\vdots$

1 1

l levels

हर बार n की value आधी हो रही है

$\therefore O(\log n)$

$\because \frac{n}{2^l} = 1 \Rightarrow n = 2^l$

$\Rightarrow \log_2 n = l$

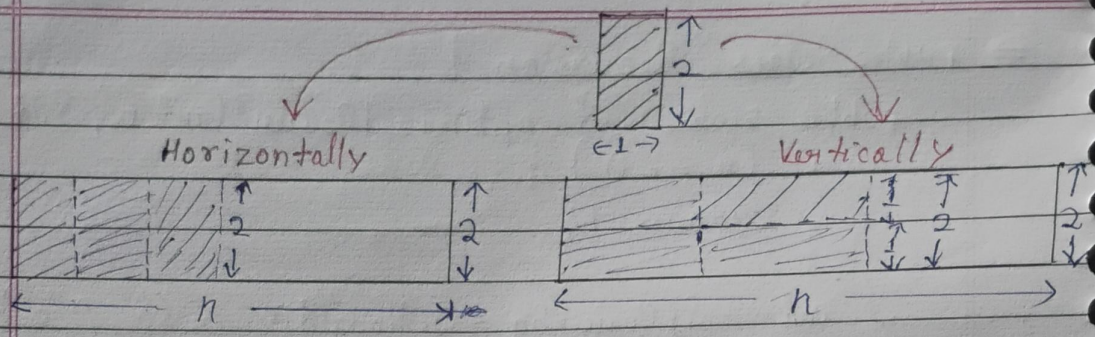
⇒
```
public class Recursion {
    public static int optimizedPower (int a, int n) {
        if (n == 0) {
            return 1;
        }
        int halfPowerSq = optimizedPower (a, n/2) *
                          optimizedPower (a, n/2);
        // n is odd.
        if (n % 2 != 0) {
            halfPowerSq = a * halfPowerSq;
        }
        return halfPowerSq;
    }
    public static void main (String[] args) {
        int a = 2;
        int n = 10;
        System.out.println (optimizedPower (a, n));
    }
}
```
// O(n)

OR

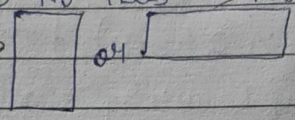
```
// O(log n)
int halfPower = optimized power (a, n/2);
int halfPowerSq = halfPower * halfPower;
```

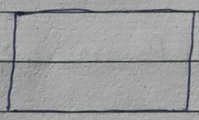
QUES. Given a "2 × n" board & tiles of size "2 × 1", count the no. of ways to tile the given board using the 2 × 1 tiles.
(A tile can either be placed horizontally or vertically.)

The above problem is known as Tiling Problem.

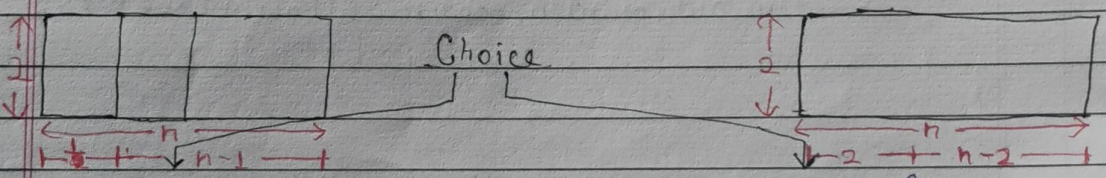


Approach: $n=0 \longrightarrow 2\times0 \longrightarrow$ No tiles $\longrightarrow$ Ways = 1.

$n=1 \longrightarrow 2\times1 \longrightarrow$  $\longrightarrow$ ways = 1.

$n=2 \longrightarrow 2\times2 \longrightarrow$  $\longrightarrow$ ways = 2

$n=3 \longrightarrow 2\times3 \longrightarrow$ ways = 3.



Vertical ⇓ fill $2\times(n-1)$ ⇓ $f(n-1)$ ⇓ vertical + $f(n-1)$

Horizontal ⇓ horizontal + $f(n-2) = 2(n-2)$

* Base Case, $n=0, 1 \longrightarrow$ ways = 1

* Total ways = $f(n-1) + f(n-2)$,
we will return this.

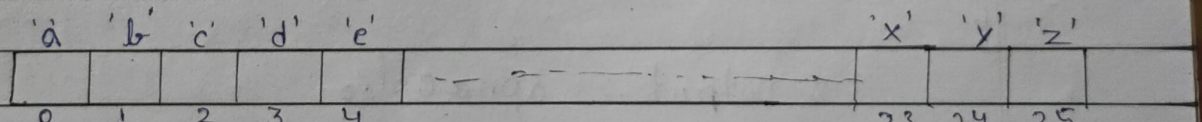
```
⇒ public class Recursion {
    public static int tilingProblem (int n) { //2 X n
        //base case
        if (n == 0 || n == 1) {
            return 1;
        }
                    //KAAM
        //vertical choice
        int fnm1 = tilingProblem (n-1);

        //horizontal choice
        int fnm2 = tilingProblem (n-2);

        int totWays = fnm1 + fnm2;
        return totWays;
    }
    public static void main (String args[]) {
        System.out.println (tilingProblem (3));
    }
}
                    3
```

QUES. Remove Duplicates in a String.

* "appnnacollegx"
  ↑↑↑↑↑↑↑↑↑↑↑↑
* "apncolg" → Unique characters.



* map[] (boolean)

* 'a' - 'a' = 0
* 'b' - 'a' = 1
* 'c' - 'a' = 2

index (map) = currchar - 'a'

// base case, idx = 0 to (n) → String-length
print "new string".

// Kaam, char → present in map ✓ → newStr ✗
char → absent in map ✗ → newStr ✓

⇒
```
public class Recursion {
  public static void removeDuplicates (String str,
        int idx, StringBuilder newStr, boolean map[]);
  if (idx == str.length()) {
    System.out.println (newStr);
    return;
  }
  char currChar = str.charAt (idx);
  if (map [currChar - 'a'] == true) {
    removeDuplicates (str, idx+1, newStr, map);
  } else {
    map [currChar - 'a'] = true;
    removeDuplicates (str, idx+1, newStr.append
          (currChar), map);
  }
}
public static void main (String args []) {
  String str = "appnnacollege";
  removeDuplicates (str, 0, new StringBuilder(""),
        new boolean [26]);
}
}
```

Output :- apnacolg

QUES. Friends Pairing Problem

Given n friends, each one can remain single or can be paired up with some other friend. Each friend can be paired only once. Find out the total number of ways in which friends can remain single or can be paired up.

* n = 1 ⟶ (single) ⟶ ways = 1.
* n = 2 ⟶ a b ⟶ a, b / (a, b) ⟶ ways = 2.
* n = 3 ⟶ a b c ⟶ a, b, c / a (b, c) / b (a, c) / c (a, b) ⟶ ways = 4.

* Base Case, n = 1 || n = 2
  ways = n;

* Choice (n)
  - Single: (n-1) → f(n-1)
  - Pair: (n-2) → (n-1) * f(n-2)
    - (n-1) → pair choices
    - f(n-2) → 1 pair बनने के बाद ways

∴ Total ways = f(n-1) + (n-1) * f(n-2)

⇒
```
public static int friendsPairing (int n) {
    if (n == 1 || n == 2) {
        return n;
    }
    //single.
    int fnm1 = friendsPairing (n-1);
    //pair.
    int fnm2 = friendsPairing (n-2);
    int pairWays = (n-1) * fnm2;
    int totWays = fnm1 + pairWays;
    return totWays;
}
```

⇒
```
return friendsPairing (n-1) + (n-1) * friendsPairing (n-2);
```

QUES. Binary Strings Problem

Print all binary strings of size N without consecutive ones.

without Consecutive ones – मतलब एक साथ दो One नहीं आ सकते

* n=0 → "_"
* n=1 → "1", "0"
* n=2 → "00", "01", "10", "11" consecutive ones.
* n Size Binary String → n=2 → _ _
* Base case, n=0 empty;.

3 →
0 0
0 1
1 0 ↖ same
1 0
1 → consecutive ones

⇒
```
public class Recursion {
    public static void printBinStrings (int n, int
                                lastPlace, String str) {
        if (n==0) {
            System.out.println (str);
            return;
        }
        printBinStrings (n-1, 0, str + "0");
        if (lastPlace == 0) {
            printBinStrings (n-1, 1, str + "1");
        }
    }
    psvm {
        printBinStrings (3, 0, "");
    }
}
```

- **Problem 1 :** For a given integer array of size N. You have to find all the occurrences (indices) of a given element (key) & print them. Use a recursive function to solve this problem.

  Sample input : arr [] = {3, 2, 4, 5, 6, 2, 7, 2, 2}, key = 2

  Sample output : 1 5 7 8

⇒

```
public class Recursion {
    public static void allOccurences (int arr[], int key, int i){
        if (i == arr.length) {
            return;
        }
        if (arr[i] == key) {
            System.out.print(i+" ");
        }
        allOccurences (arr, key, i+1);
    }
    public static void main (String args[]) {
        int arr[] = {3, 2, 4, 5, 6, 2, 7, 2, 2};
        int key = 2;
        allOccurences (arr, key, 0);
        System.out.println();
    }
}
```

- **Problem 2 :** You are given a number (eg - 2019), convert it into String of english like "two zero one nine". Use a recursive function to solve this problem.

NOTE : The digits of the number will only in range 0-9 & the last digit of a number can't be 0.

Sample input : 1947

Sample output : "one nine four seven".

- Problem 2:

⇒
```
public class Recursion {
    static String digits[] = {"zero", "one", "two", "three",
        "four", "five", "six", "seven", "eight", "nine"};
    public static void printDigits (int number) {
        if (number == 0) {
            return;
        }
        int lastDigit = number % 10;
        printDigits (number/10);
        System.out.print (digits[lastDigit] + " ");
    }
    public static void main (String [] args){
        printDigits (1234);
        System.out.println();
    }
}
```

- Problem 3: Write a program to find Length of a String using Recursion.

⇒
```
public class Recursion {
    public static int length (String str) {
        if (str.length() == 0) {
            return 0;
        }
        return length (str.substring (1)) + 1;
    }
    public static void main (String[] args) {
        String str = "abcde";
        System.out.println (length (str));
    }
}
```

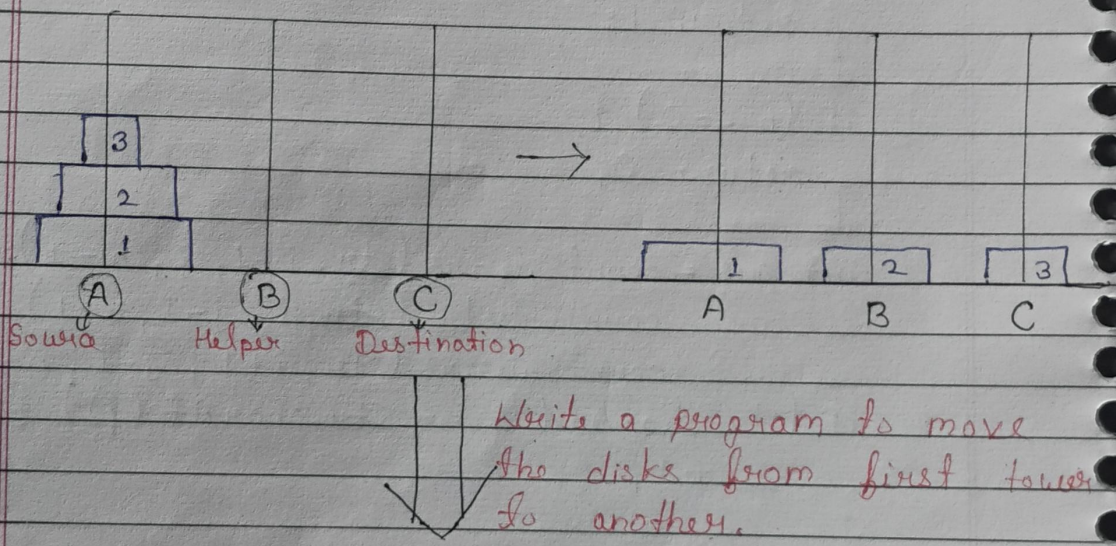
- Problem 4: We are given a string S, we need to find the count of all contiguous substrings starting & ending with the same character.
  Sample input: S = "abcab"
  Sample output: 7

There are 15 substrings of "abcab": ab, a, abc, abca, abcab, b, bc, bca, bcab, c, ca, cab, a, ab, b.
Out of the above substrings, there are 7 substrings: a, abca, b, bcab, c, a & b.

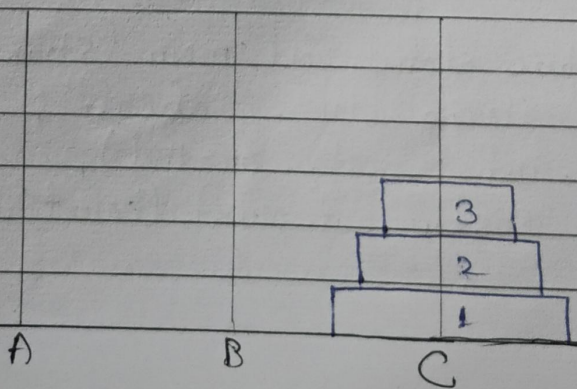
⇒
```
public class Recursion {
    public static int countSubstrs (String str, int i, int j, int n){
        if (n == 1) {
            return 1;
        }
        if (n <= 0) {
            return 0;
        }
        int res = countSubstrs (str, i+1, j, n-1) +
                  countSubstrs (str, i, j-1, n-1) -
                  countSubstrs (str, i+1, j-1, n-2);
        if (str.charAt(i) == str.charAt(j)) {
            res++;
        }
        return res;
    }
    public static void main (String[] args) {
        String str = "abcab";
        int n = str.length();
        System.out.print (countSubstrs (str, 0, n-1, n));
    }
}
```

• Problem 5: Tower of Hanoi Problem

* We have 3 towers & N disks of different sizes which can slide onto any tower. The puzzle starts with disks sorted in ascending order of size from top to bottom (i.e., each disk sits on top of an even larger one).
* We have to shift disks to another tower so we have following constraints :
1) Only one disk can be moved at a time.
2) A disk is slid off the top of one tower onto another tower.
3) A disk cannot be placed on top of a smaller disk.



Write a program to move the disks from first tower to another.



⇒
```
public class Recursion {
    public static void towerOfHanoi (int n, String src, String helper, String dest) {
        if (n == 1) {
            System.out.println ("Transfer disk " + n + " from " + src +
                                " to " + dest);
            return;
        }
    //transfer top n-1 from src to helper using dest as 'helper'.
        towerOfHanoi (n-1, src, dest, helper);
    //transfer nth from src to dest
        System.out.println ("Transfer disk " + n + " from " + src +
                            " to " + dest);
    //transfer n-1 from helper to dest, using src as 'helper'.
        towerOfHanoi (n-1, helper, src, dest);
    }
    public static void main (String args[]) {
        int n = 4;
        towerOfHanoi (n, "A", "B", "C");
    }
}
```