TP Session 4

30 September 2024 16:27

Given x number of 1's followed by y number of 0's, your task is to find the decimal representation of the x and y.

```
    int decimal = Math.pow(2, x) - 1
     decimal =decima1 * Math.pow(2, y)
```

Better approach

```
         int res=0
         int decimal = (1 << x) - 1
         decimal <<= y
```

if number is too large use big integer class

```
        import java.math.BigInteger;
         BigInteger x = scanner.nextBigInteger();
         BigInteger y = scanner.nextBigInteger();

          BigInteger res = BigInteger.ZERO;

         BigInteger res =
BigInteger.ONE.shiftLeft(x.intValue()).subtract(BigInteger.ONE).shiftLeft(y.intValue());
```

Given Xth and Yth Bit position. Create a number where X and Yth bit are Set:

```
int result= (1<<x) | (1<<y)
```

In a given integer - N, check whether the ith bit is set or not.

suppose i=2 and N=10, 1010 in binary: 2nd bit is 0(starting from 0,1,2)

```
if(n>>i & 1==1) then
    print true
else
    print false
```

Calculate number of bits required to represent an integer value:

```
                while n>0 do
                 count++
                 n>>=1
                 print count
```

instead: `int bits = (int)(Math.log10(num) / Math.log10(2)) + 1;`

For example Take num=4

---

this is equal to $\log_{10}(4)/\log_{10}(2)$--> $\log_2(4)=2+1=3$

Reverse the Bits of an integer number and print the value in decimal.

```
    while N>0 Do
            res=0
           res = (res << 1) | (n & 1);
            n >>= 1;
        }
       print res
```

N&1 to extract LSB bit of original number.
Res<<1 to shift the result bit to the left side so that we can reverse the original number.
OR operation to combine the shifted res with the extracted bit.
n>>=1 to move to the next bit in the number

Largest Power of 3 less than or equal to given number N:

```
initalize res to 1
initalize power to 1

until power < N Do
   res=power
   power =power * 3

print(res)
```

If the number is too large use BigInteger Class

```
-----------------------------------------------------
   String input = scanner.nextLine();
   BigInteger N = new BigInteger(input);
         // or read directly as: BigInteger x = scanner.nextBigInteger();

       BigInteger res = BigInteger.ONE;
       BigInteger power = BigInteger.ONE;

           while (power.compareTo(N) < 0) {
           res = power;
           power = power.multiply(BigInteger.valueOf(3));
       }

       print(res)
```

Print Number of trailing zeroes in the factorial of a given number N

GENERAL APPROACH

```
     mainfunction()
     {
        read long int value
        long fact=factorial(n)
        long count=counttrailingzeroes(fact)
        print(count)
```

```
}

factorial(long n)
{
      intitalize long res=1
      for i=1 to n   (i=1;i<=n;i++)
      res=res*i
      return res
}

counttrailingzeroes(long n)
  {
     int count = 0;
    while n > 0 Do
      if n mod 10 == 0 then
         count++
      else
         break the loop
      n= n/10;
    END WHILE
     return count
  }
```

BETTER APPROACH

every multiple of 5 contributes to number of trailing zeroes in N!

use formula: n/5+n/25+n/125 and so on!!

initalize count=0

```
while n >0 Do
count = count + (n/5)
n = n/5
END WHILE

print(count)
```

Check if a given number is Prime Number or Not

```
int flag=1
if n<=1
   print not Prime
   return

for i=2 to Math.sqrt(n) DO
   if n mod i==0 then
   set flag to 0
    break the loop

if flag==1 then
   print "prime"
else
   print "not Prime"
```

Instead of using Math.sqrt the better efficient way could be:

```
for i=2 to i*i<=n
   if n mod i==0
      set flag=0
      break the loop
```

Print Prime numbers from 1 to given range N

```
mainfunction()
{
      read n
      for i=1 to n+1 Do
       if(checkprime(i)):
           System.out.print(i+" ")
}

boolean checkprime(int n)
{
    if n is less than or equal to 1
        return False
    for i=2 to Math.sqrt(n) Do
        if n%i==0
            return false
    return true
}
```

Print Prime Numbers upto given Count

```
read N
set count=0
set i=1
while count<n Do:
    res=checkprime(i)
    if res==true
        System.out.print(i+" ")
        increment count++
    increment i++


boolean checkprime(int n)
{
    if n is less than or equal to 1
        return False
    for i=2 to Math.sqrt(n) Do
        if n%i==0 then
            return false
    return true
}
```

SIEVE OF ERATOSTHENES: THE MOST OPTIMIZED PRIME NUMBER LOGIC

```
        boolean[] prime = new boolean[n + 1]
```

```
        for int i=0 to n Do
          set prime values to true


        OuterLoop for int i=2 to i*i<=n Do
           check if prime[i] is true if so then gotoinner loop
                  for int j=p*p to n (Note: increment innerloop by j=j+p)
                     set prime[j] to false


           for int i=2 to n Do
            if prime[i] is true then
              print(prime[i]+" ")
```

Print prime numbers upto given range N for T test cases each on new line

```
Take input number of test cases T
iterate from i=1 to T Do
    read integer number n
   call printupton(n) function


function printupton(n):
    iterate from i to n Do
        if(checkforrime(i)==true):
            print(i+" ")
    println()
End Function


function boolean checkforprime(int n)
    if n<=1 then
        return false
    iterate from i=2 to sqrt(n) D0
        if n%i==0 then
            return false
    return true
End function
```

for example:

```
T=3
10
2 3 5 7
20
2 3 5 7 11 13 17 19
100
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

Sum Between the largest and second largest in the arraylist

01 October 2024 00:39

```
--------------------------------------------------------------------------------
input=2 1 6 7 8 9 5 4 10
quit
output=
Sum between the largest and second-largest numbers: 9


STEPs:
--------
  INITIALIZE ArrayList named as numbers

  WHILE (sc.hasNextInt())
      READ number and ADD to numbers

  IF (numbers.size() < 2)
      PRINT "At least two distinct numbers are required."
      RETURN

  INITIALIZE Bigindex = 0
  INITIALIZE secondBig index= -1

Iterate from i=1 to i less than numbers.size() Do
      IF (numbers[i] > numbers[Bigindex])
          secondBigindex = Bigindex
          Bigindex = i
      ELSE IF (secondBigindex == -1 OR numbers[i] > numbers[secondBigindex])
          secondBigindex = i

  INITIALIZE sum = 0
  k = MIN(Big, secondBig)

  Iterate from j=k+1 TO k < (Big + secondBig - k)
      sum = sum + numbers[j]

  PRINT "Sum between the largest and second-largest numbers: " + sum
```

Harmony in Array

01 October 2024 00:52

```
input=5
3 2 5 3 2
output=
Harmony Index: 2
```

STEPs:

Read the number of elements, n.
Read the elements of the array into a list called arr.

Set totalSum to 0 (to hold the sum of all elements in the array).
Set leftSum to 0 (to hold the sum of elements to the left of the current index).

Iterate through the array from index 0 to n-1:
sum each element and store it in totalSum.

Iterate through the array again from index 0 to n-1:
Subtract the current element arr[i] from totalSum (this gives the sum of elements to the right of the current index).
Check if leftSum is equal to totalSum:
If they are equal, return the current index i (this is the harmony index).
else Add the current element arr[i] to leftSum.

If no harmony index is found after the loop, print no such element.

Kth Smallest ELement in the Array

01 October 2024 00:52

```
input=
2 3  2 rows 3 columns
7 6 8
6 8 10
3
output=
The 3rd smallest element is: 7


STEPs:
--------
Function findKthSmallest(int[][] matrix, int k)

   Input: A 2D array (matrix) and an integer k.

   Get the number of rows rows in the matrix. (matrix.length)
   Get the number of columns cols in the matrix. (matrix[0].length)

   Create a 1D array of size rows * cols to store all elements from the matrix.

   Use nested loops to iterate through each element of the matrix:
      For each element at position matrix[i][j], assign it to the flatArray[index], where
      index is incremented after each assignment.
          i.e flatarray[i++]=matrix[i][j]

   Use a sorting algorithm (e.g., Arrays.sort) to sort 1D array in ascending order.

   Return the element at index k - 1 of the sorted 1D Array
END
```

Unique Pairs and Count in 2D array of Strings

01 October 2024 00:52

```
input=5
virat kohli
rohit sharma
ishan kishan
virat kohli
KL rahul
output=
1
2
3
3
4
```

STEPs:

Read N which indicates number of strings

Initialize a 2D array input of size N x 2 to store the pairs of strings.
  String[][] input= new String[N][2]

Use a loop to read n lines of input. For each line, split the line into two strings and store them in the input array. (u can use input.split(" "))

initialize a HashSet<String> called set to keep track of unique pairs and an integer variable count set to 0 to track the number of unique pairs.

Loop through each pair in the input array. For each pair, merge the two strings into a single string with a space in between.

```
for (String[] i : pairs) {
String merged = i[0] + " " + i[1]
```

Check if the merged string is already present in the set. If it is, print the current count.

else if the merged string is not in the set, add it to the set, increment the count by 1, and print the updated count.

Winning Candidate

01 October 2024 00:52

```
input=5
3 1 3 3 2
output=
Winning Candidate: 3
```

STEPs:

Function WinningCandidate(ArrayList<Integer> List)

intialize count to 0 and candidate to -1.
Iterate through each number in the list.
    first if: If count is 0, set the candidate to the current number.
    second if: If the current number is equal to candidate, increment count.
        else the current number is not equal to candidate, decrement count.

After the first loop, initialize finalCount to 0.
Iterate through the list again to count how many times the candidate appears.
    If a number matches the candidate, increment finalCount.
Check if finalCount is greater than half the size of the list.
    If it is, return the candidate. (original element)
    If it is not, print no such candidate present

Longest Consecutive Subsequence

01 October 2024 00:52

```
input=2 1 0 3 quit
output=
Length of the longest consecutive subsequence: 4

input=44 45 2 6 47 90 48 12 56 49 100 50
quit
output=
Length of the longest consecutive subsequence: 4
```

Explaination: 47,48,49,50 is the longest consecutive sequence hence output is 4.

STEPs:

Function Subsequence(HashSet<Integer> numberSet)

Initialize the variable longestStreak to zero

Iterate over each element in numberSet

For each num in numberSet, check if num - 1 is not in numberSet

    If true, this indicates the start of a new consecutive sequence

        Set currentNum to num and currentStreak to 1

    While numberSet contains currentNum + 1

        Increment currentNum by 1

        Increment currentStreak by 1

    End While

    Update longestStreak to be the maximum of longestStreak and currentStreak

End For Loop

After iterating through all elements, return longestStreak