1-3: Easy

4-6:Medium

7-9:Hard

1.(https://www.geeksforgeeks.org/implementation-of-bit-stuffing-and-bit-destuffing/)

// Java program for the above approach

class GFG {

// Function for bit stuffing

static void bitStuffing(int N, int arr[])

{

// Stores the stuffed array

int[] brr = new int[30];

// Variables to traverse arrays

int i, j, k;

i = 0;

j = 0;

// Loop to traverse in the range [0, N)

while (i < N) {

// If the current bit is a set bit

if (arr[i] == 1) {

// Stores the count of consecutive ones

int count = 1;

// Insert into array brr[]

brr[j] = arr[i];

// Loop to check for

// next 5 bits

for (k = i + 1; k < N && arr[k] == 1

&& count < 5;

k++) {

j++;

brr[j] = arr[k];

count++;

// If 5 consecutive set bits

// are found insert a 0 bit

if (count == 5) {

j++;

brr[j] = 0;

}

i = k;

}

}

// Otherwise insert arr[i] into

// the array brr[]

else {

brr[j] = arr[i];

}

i++;

j++;

}

// Print Answer

for (i = 0; i < j; i++)

System.out.printf("%d", brr[i]);

}

// Driver Code

public static void main(String[] args)

{

int N = 6;

int arr[] = { 1, 1, 1, 1, 1, 1 };

bitStuffing(N, arr);

}

}

// This code is contributed by shikhasingrajput

Please generate comments after each line of code

class GFG {

// Defines a class named GFG.

static void bitStuffing(int N, int arr[]) {

// Declares a static method named bitStuffing that takes an integer N (size of the input array) and an array of integers arr[] (input array for bit stuffing).

int[] brr = new int[30];

// Initializes a new integer array brr[] with a fixed size of 30 to store the result of bit stuffing.

int i, j, k;

// Declares three integers i, j, and k to be used as counters in the loops.

i = 0;

j = 0;

// Initializes i and j to 0. i is used to track the current position in the input array, and j is used to track the current position in the result array.

while (i < N) {

// Starts a loop that iterates over each element of the input array until it reaches the end.

if (arr[i] == 1) {

// Checks if the current element in the input array is 1.

int count = 1;

// Initializes a counter named count to 1, to track the number of consecutive 1s.

brr[j] = arr[i];

// Copies the current element to the result array.

for (k = i + 1; k < N && arr[k] == 1 && count < 5; k++) {

// Starts a nested loop to check for up to 4 consecutive 1s following the current 1.

j++;

brr[j] = arr[k];

// Copies each consecutive 1 to the result array and increments j.

count++;

// Increments the count of consecutive 1s.

if (count == 5) {

// Checks if there are 5 consecutive 1s.

j++;

brr[j] = 0;

// Inserts a 0 after the 5th consecutive 1 in the result array to prevent the bit pattern from being mistaken as a control sequence.

}

i = k;

// Updates i to the current position in the input array.

}

}

else {

brr[j] = arr[i];

// If the current element is not 1, simply copies it to the result array.

}

i++;

j++;

// Increments both i and j to move to the next positions in the input and result arrays, respectively.

}

for (i = 0; i < j; i++)

System.out.printf("%d", brr[i]);

// Prints each element of the result array to the console.

}

public static void main(String[] args)

{

// The main method - entry point of the program.

int N = 6;

// Defines N, the number of elements in the input array.

int arr[] = { 1, 1, 1, 1, 1, 1 };

// Initializes the input array arr[] with 6 elements, all set to 1.

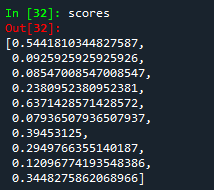
bitStuffing(N, arr);

// Calls the bitStuffing method with N and arr[] as arguments to perform bit stuffing.

}

}

Measure of METEOR Score:





1. (https://www.geeksforgeeks.org/reverse-given-range-of-string-for-m-queries/)

// Java program to find the string after M reversals on

// query ranges

import java.io.\*;

class GFG {

// Function for reversing string according

// to each query

public static String reverseForAll(String S, int N,

int A[], int M)

{

String tmp = "";

for (int j = 0; j < M; j++) {

int start = A[j] - 1;

int end = N - A[j] + 1;

tmp = "";

// reverse(S.begin() + start, S.begin() + end);

// string tmp="";

for (int i = 0; i < start; i++)

tmp += S.charAt(i);

for (int i = end - 1; i >= start; i--)

tmp += S.charAt(i);

for (int i = end; i < S.length(); i++)

tmp += S.charAt(i);

S = tmp;

}

return tmp;

}

// Driver Code

public static void main(String[] args)

{

int N = 6;

String S = "abcdef";

int M = 3;

int A[] = { 1, 2, 3 };

// Function call

String ans = reverseForAll(S, N, A, M);

System.out.println(ans);

}

}

// This code is contributed by Akshay

// Tripathi(akshaytripathi19410)

import java.io.\*;

// Import statement for Java input and output libraries, though not explicitly used in this code.

class GFG {

// Declaration of a class named GFG.

public static String reverseForAll(String S, int N, int A[], int M) {

// Definition of a static method named reverseForAll that takes a String S, an integer N representing the length of S,

// an array of integers A representing positions for substring reversals, and an integer M representing the length of array A.

String tmp = "";

// Initializes an empty String named tmp to temporarily store the modified string in each iteration.

for (int j = 0; j < M; j++) {

// A for loop that iterates M times, each iteration applies a specific substring reversal based on the elements in array A.

int start = A[j] - 1;

// Calculates the starting index of the substring to reverse in the current iteration.

int end = N - A[j] + 1;

// Calculates the ending index of the substring to reverse in the current iteration.

tmp = "";

// Resets tmp to an empty string for the current iteration.

for (int i = 0; i < start; i++)

tmp += S.charAt(i);

// Concatenates characters from the beginning of S up to the start index (exclusive) to tmp.

for (int i = end - 1; i >= start; i--)

tmp += S.charAt(i);

// Concatenates characters from the end index (inclusive) to the start index (inclusive) in reverse order to tmp.

for (int i = end; i < S.length(); i++)

tmp += S.charAt(i);

// Concatenates the remaining characters from the end index (exclusive) to the end of S to tmp.

S = tmp;

// Assigns the value of tmp to S, preparing S with the modified string for the next iteration or final result.

}

return tmp;

// Returns the final modified string after all specified reversals have been applied.

}

public static void main(String[] args) {

// The main method - the entry point of the program.

int N = 6;

// Initializes N to 6, representing the length of the string S.

String S = "abcdef";

// Initializes S with the string "abcdef".

int M = 3;

// Initializes M to 3, representing the number of elements in the array A.

int A[] = { 1, 2, 3 };

// Initializes array A with values {1, 2, 3}, representing positions for substring reversals.

String ans = reverseForAll(S, N, A, M);

// Calls the method reverseForAll with S, N, A, and M as arguments and stores the result in ans.

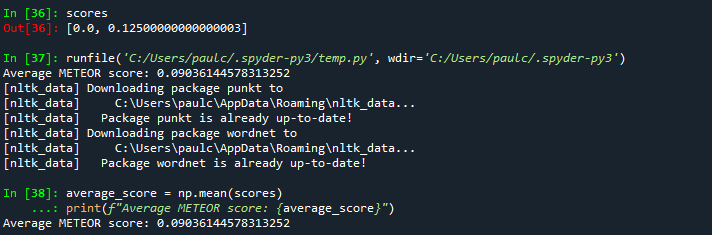
System.out.println(ans);

// Prints the final modified string stored in ans to the console.

}

}

Evaluation:



3. (https://www.geeksforgeeks.org/capitalize-1st-character-of-all-words-having-at-least-k-characters/)

// Java program of the above approach

import java.util.\*;

class GFG

{

// Function to capitalize the 1st

// character of all words having

// at least K characters

public static String capitalizeStr(String str, int K)

{

// Stores location of

// the 1st character of

// the current word

int ptr = 0;

char[] ch = str.toCharArray();

// Loop to traverse string

for (int i = 0; i < ch.length; i++) {

// If the current word

// ends at index i

if (ch[i] == ' ') {

// Update ptr

ptr = i + 1;

}

// Count of characters

// is at least K

else if (i - ptr + 1 >= K) {

ch[ptr] = Character.toUpperCase(ch[ptr]);

}

}

String s = new String(ch);

// Return answer

return s;

}

// Driver Code

public static void main(String args[])

{

String str = "geeksforgeeks is the best";

int K = 0;

System.out.println(capitalizeStr(str, K));

}

}

// This code is contributed by Taranpreet

Please generate comments after each line of code

// Importing necessary classes from the Java utility package

import java.util.\*;

// Declaration of the class named GFG

class GFG

{

// Method to capitalize the first character of each word in the string that is at least K characters long

public static String capitalizeStr(String str, int K)

{

// Variable to keep track of the start index of each word

int ptr = 0;

// Convert the input string into a character array for manipulation

char[] ch = str.toCharArray();

// Loop through each character in the array

for (int i = 0; i < ch.length; i++) {

// If the current character is a space, update ptr to point to the start of the next word

if (ch[i] == ' ') {

ptr = i

// If the current word length is at least K, capitalize the first character of this word

else if (i - ptr + 1 >= K) {

ch[ptr] = Character.toUpperCase(ch[ptr]);

}

}

// Convert the character array back to a String

String s = new String(ch);

// Return the modified string

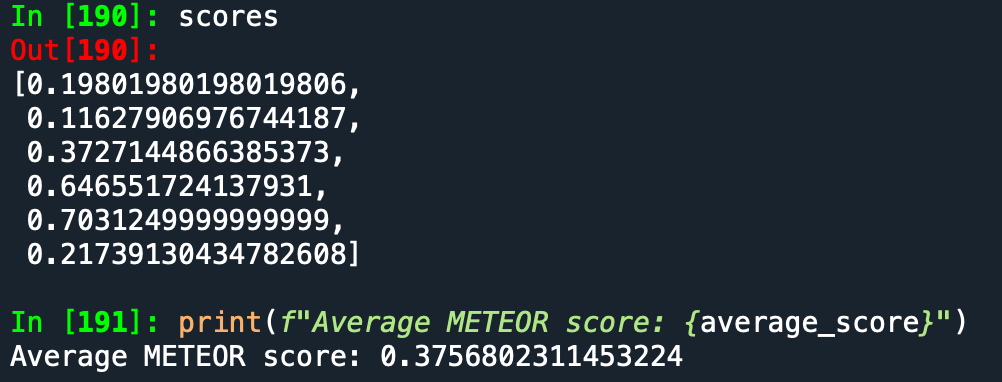
return s;

}

}

Meteor Evaluation:

**The Comment Comparison for each line of code**



**The Comment Comparison for function description**

Screen Shot 2024-03-20 at 9.29.31 AM

4.(https://www.geeksforgeeks.org/game-of-matchsticks/)

// Java program for the above approach

public class GFG {

// Function to calculate the number of matchsticks A

// should pick initially

static int matchGame(long N)

{

// res variable stores the number of matchsticks

// initially picked by A

int res = (int)(N % 5);

// If res or N%5 is 0 then there is no chance of A

// to win the game

if (res == 0) {

return -1;

}

// else return the remainder value after dividing N

// by 5

return res;

}

// Driver code

public static void main(String[] args)

{

// Total number of matchsticks is N

long N = 48;

// Function call

System.out.println(matchGame(N));

}

}

// This code is contributed by Susobhan Akhuli

public class GFG {

// Declares a public class named GFG.

static int matchGame(long N) {

// Defines a static method named matchGame that takes a long N and returns an integer. This method calculates the remainder of N divided by 5.

int res = (int)(N % 5);

// Calculates the remainder of N divided by 5, casts the result to int, and stores it in the variable res.

if (res == 0) {

// Checks if the remainder (res) is 0. This condition means N is perfectly divisible by 5.

return -1;

// Returns -1 if N is perfectly divisible by 5, indicating a specific condition or outcome.

}

return res;

// Returns the remainder (res) if it's not 0, which is the result of N % 5.

}

public static void main(String[] args) {

// The main method, which is the entry point of the program.

long N = 48;

// Declares and initializes a long variable N with the value 48, representing the input to the matchGame method.

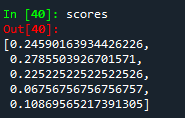
System.out.println(matchGame(N));

// Calls the matchGame method with N as the argument and prints the result. This demonstrates how the method behaves with the given input N = 48.

}

}

Evaluation:





5. (https://www.geeksforgeeks.org/bingo-sort-algorithm/)

// Java Code for the above approach

import java.io.\*;

class GFG {

static int bingo;

static int nextBingo;

// Function for finding the maximum and minimum element

// of

// the Array

static void maxMin(int[] vec, int n)

{

for (int i = 1; i < n; i++) {

bingo = Math.min(bingo, vec[i]);

nextBingo = Math.max(nextBingo, vec[i]);

}

}

// Function to sort the array

static int[] bingoSort(int[] vec, int n)

{

bingo = vec[0];

nextBingo = vec[0];

maxMin(vec, n);

int largestEle = nextBingo;

int nextElePos = 0;

while (bingo < nextBingo) {

// Will keep the track of the element position

// to

// shifted to their correct position

int startPos = nextElePos;

for (int i = startPos; i < n; i++) {

if (vec[i] == bingo) {

int temp = vec[i];

vec[i] = vec[nextElePos];

vec[nextElePos] = temp;

nextElePos = nextElePos + 1;

}

// Here we are finding the next Bingo

// Element for the next pass

else if (vec[i] < nextBingo)

nextBingo = vec[i];

}

bingo = nextBingo;

nextBingo = largestEle;

}

return vec;

}

// Function to print the array

static void printArray(int[] arr, int n)

{

System.out.print("Sorted Array: ");

for (int i = 0; i < n; i++) {

System.out.print(arr[i] + " ");

}

System.out.println();

}

public static void main(String[] args)

{

int[] arr = { 5, 4, 8, 5, 4, 8, 5, 4, 4, 4 };

arr = bingoSort(arr, arr.length);

printArray(arr, arr.length);

int[] arr2 = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };

arr2 = bingoSort(arr2, arr2.length);

printArray(arr2, arr2.length);

int[] arr3 = { 0, 1, 0, 1, 0, 1 };

arr3 = bingoSort(arr3, arr3.length);

printArray(arr3, arr3.length);

}

}

// This code is contributed by lokeshmvs21.

Please generate comments after each line of code

import java.io.\*;

// Imports the Java IO package for input and output operations, although it's not used in this specific program.

class GFG {

// Declaration of a class named GFG.

static int bingo;

// Declaration of a static integer variable 'bingo' to hold the minimum value found in the array during sorting.

static int nextBingo;

// Declaration of a static integer variable 'nextBingo' to hold the next minimum value or the maximum value during the sorting process.

static void maxMin(int[] vec, int n) {

// A static method named maxMin that takes an integer array 'vec' and its size 'n' as parameters. It updates the 'bingo' and 'nextBingo' variables to the minimum and maximum values of the array, respectively.

for (int i = 1; i < n; i++) {

// Iterates over the array starting from the second element.

bingo = Math.min(bingo, vec[i]);

// Updates 'bingo' to the minimum value between 'bingo' and the current element of the array.

nextBingo = Math.max(nextBingo, vec[i]);

// Updates 'nextBingo' to the maximum value between 'nextBingo' and the current element of the array.

}

}

static int[] bingoSort(int[] vec, int n) {

// A static method named bingoSort that takes an integer array 'vec' and its size 'n' as parameters. It sorts the array in ascending order and returns the sorted array.

bingo = vec[0];

nextBingo = vec[0];

// Initializes 'bingo' and 'nextBingo' with the value of the first element of the array.

maxMin(vec, n);

// Calls the maxMin method to find the minimum and maximum values in the array.

int largestEle = nextBingo;

// Initializes 'largestEle' with the value of 'nextBingo', which is the maximum value found in the array.

int nextElePos = 0;

// Initializes 'nextElePos', which will be used to track the position for swapping elements in the sorting process.

while (bingo < nextBingo) {

// Continues the sorting process as long as 'bingo' is less than 'nextBingo'.

int startPos = nextElePos;

// Sets 'startPos' to the current value of 'nextElePos'.

for (int i = startPos; i < n; i++) {

// Iterates over the array starting from 'startPos'.

if (vec[i] == bingo) {

// Checks if the current element is equal to 'bingo' (the minimum value to be placed next in order).

int temp = vec[i];

// Temporarily stores the current element.

vec[i] = vec[nextElePos];

// Swaps the current element with the element at 'nextElePos'.

vec[nextElePos] = temp;

// Completes the swap by placing the temporary value into the position of 'nextElePos'.

nextElePos = nextElePos + 1;

// Increments 'nextElePos' to prepare for the next swap position.

}

else if (vec[i] < nextBingo)

// Checks if the current element is less than 'nextBingo' and updates 'nextBingo' if true.

nextBingo = vec[i];

}

bingo = nextBingo;

// Sets 'bingo' to the value of 'nextBingo' for the next iteration.

nextBingo = largestEle;

// Resets 'nextBingo' to 'largestEle' for finding the next smallest element.

}

return vec;

// Returns the sorted array.

}

static void printArray(int[] arr, int n) {

// A static method to print the elements of an array 'arr' of size 'n'.

System.out.print("Sorted Array: ");

// Prints a prefix message.

for (int i = 0; i < n; i++) {

// Iterates over the array.

System.out.print(arr[i] + " ");

// Prints each element followed by a space.

}

System.out.println();

// Prints a newline character after printing all array elements.

}

public static void main(String[] args) {

// The main method - the entry point of the program.

int[] arr = { 5, 4, 8, 5, 4, 8, 5, 4, 4, 4 };

// Initializes the first test array 'arr'.

arr = bingoSort(arr, arr.length);

// Calls bingoSort on 'arr' and updates

Evaluation:

**The comment comparison for each line of code**

Screen Shot 2024-03-20 at 9.53.31 AM

**The comment comparison for the function description.**

Screen Shot 2024-03-20 at 9.57.54 AM

6. (https://www.geeksforgeeks.org/find-the-longest-common-substring-using-binary-search-and-rolling-hash/)

import java.util.HashSet;

import java.util.Set;

class ComputeHash {

private long[] hash;

private long[] invMod;

private long mod;

private long p;

// Generates hash in O(n(log(n)))

public ComputeHash(String s, long p, long mod) {

int n = s.length();

this.hash = new long[n];

this.invMod = new long[n];

this.mod = mod;

this.p = p;

long pPow = 1;

long hashValue = 0;

for (int i = 0; i < n; i++) {

char c = s.charAt(i);

c = (char) (c - 'A' + 1);

hashValue = (hashValue + c \* pPow) % this.mod;

this.hash[i] = hashValue;

this.invMod[i] = (long)(Math.pow(pPow, this.mod - 2) % this.mod);

pPow = (pPow \* this.p) % this.mod;

}

}

// Return hash of a window in O(1)

public long getHash(int l, int r) {

if (l == 0) {

return this.hash[r];

}

long window = (this.hash[r] - this.hash[l - 1]) % this.mod;

return (window \* this.invMod[l]) % this.mod;

}

}

public class Main {

// Function to get the longest common substring

public static int longestCommonSubstr(String X, String Y) {

int n = X.length();

int m = Y.length();

long p1 = 31;

long p2 = 37;

long m1 = (long) (Math.pow(10, 9) + 9);

long m2 = (long) (Math.pow(10, 9) + 7);

// Initialize two hash objects

// with different p1, p2, m1, m2

// to reduce collision

ComputeHash hashX1 = new ComputeHash(X, p1, m1);

ComputeHash hashX2 = new ComputeHash(X, p2, m2);

ComputeHash hashY1 = new ComputeHash(Y, p1, m1);

ComputeHash hashY2 = new ComputeHash(Y, p2, m2);

// Function that returns the existence

// of a common substring of length k

int low = 0, high = Math.min(n, m);

int answer = 0;

while (low <= high) {

int mid = (low + high) / 2;

if (exists(mid, X, Y)) {

answer = mid;

low = mid + 1;

} else {

high = mid - 1;

}

}

return answer;

}

private static boolean exists(int k, String X, String Y) {

for (int i = 0; i <= X.length() - k; i++) {

for (int j = 0; j <= Y.length() - k; j++) {

if (X.substring(i, i + k).equals(Y.substring(j, j + k))) {

return true;

}

}

}

return false;

}

public static void main(String[] args) {

String X = "GeeksforGeeks";

String Y = "GeeksQuiz";

System.out.println(longestCommonSubstr(X, Y));

}

}

Please generate comments after each line of code

import java.util.HashSet;

import java.util.Set;

// These import statements are unused in this code snippet but are typically used for working with collections in Java.

class ComputeHash {

// Class designed to compute hash values for substrings of a given string using polynomial rolling hash function.

private long[] hash;

// Array to store hash values of all prefixes of the string.

private long[] invMod;

// Array to store modular inverses for hash calculations.

private long mod;

// The modulus value used in the hash function to avoid overflow.

private long p;

// The base value used in the polynomial rolling hash function.

public ComputeHash(String s, long p, long mod) {

// Constructor for ComputeHash class. Initializes and computes the hash and invMod arrays for a string s.

int n = s.length();

// Store the length of the input string.

this.hash = new long[n];

// Initialize the hash array with the length of the string.

this.invMod = new long[n];

// Initialize the invMod array with the length of the string.

this.mod = mod;

// Set the mod field to the passed modulus value.

this.p = p;

// Set the p field to the passed base value.

long pPow = 1;

// Initialize the current power of p to 1.

long hashValue = 0;

// Initialize the cumulative hash value to 0.

for (int i = 0; i < n; i++) {

// Iterate over the string to compute hash values.

char c = s.charAt(i);

// Get the i-th character of the string.

c = (char) (c - 'A' + 1);

// Convert the character to a numeric value assuming 'A' is 1.

hashValue = (hashValue + c \* pPow) % this.mod;

// Update the hash value by adding the current character's hash and taking modulo mod.

this.hash[i] = hashValue;

// Store the cumulative hash value in the hash array.

this.invMod[i] = (long)(Math.pow(pPow, this.mod - 2) % this.mod);

// Compute and store the modular inverse of the current power of p.

pPow = (pPow \* this.p) % this.mod;

// Update pPow for the next iteration by multiplying with the base and taking modulo mod.

}

}

public long getHash(int l, int r) {

// Method to get the hash value of a substring from index l to r.

if (l == 0) {

// If the substring starts from the beginning, simply return the hash of r.

return this.hash[r];

}

long window = (this.hash[r] - this.hash[l - 1] + this.mod) % this.mod;

// Compute the hash value of the substring by taking the difference of hashes and ensuring it's positive by adding mod and then taking modulo mod.

return (window \* this.invMod[l]) % this.mod;

// Adjust the hash value by multiplying with the modular inverse of the power at index l, then take modulo mod.

}

}

public class Main {

// Main class to find the longest common substring between two strings.

public static int longestCommonSubstr(String X, String Y) {

// Method to find the length of the longest common substring.

int n = X.length();

// Length of the first string.

int m = Y.length();

// Length of the second string.

long p1 = 31;

// A prime number used as the base for the first hash function.

long p2 = 37;

// Another prime number used as the base for the second hash function.

long m1 = (long) (Math.pow(10, 9) + 9);

// A large prime number used as the modulus for the first hash function.

long m2 = (long) (Math.pow(10, 9) + 7);

// Another large prime number used as the modulus for the second hash function.

ComputeHash hashX1 = new ComputeHash(X, p1, m1);

// Creates a ComputeHash object for string X using p1 and m1.

ComputeHash hashX2 = new ComputeHash(X, p2, m2);

// Creates another ComputeHash object for string X using p2 and m2.

ComputeHash hashY1 = new ComputeHash(Y, p1, m1);

// Creates a ComputeHash object for string Y using p1 and m1.

ComputeHash hashY2 = new ComputeHash(Y, p2, m2);

// Creates another ComputeHash

int low = 0, high = Math.min(n, m);

// Initialize binary search boundaries to find the length of the longest common substring.

int answer = 0;

// Variable to store the length of the longest common substring found so far.

while (low <= high) {

// Binary search loop to find the maximum length of the common substring.

int mid = (low + high) / 2;

// Calculate the middle point to check if a common substring of this length exists.

if (exists(mid, X, Y)) {

// Call the exists method to check if there's a common substring of length 'mid' in both X and Y.

answer = mid;

// Update answer to mid since we found a longer common substring.

low = mid + 1;

// Move the lower boundary up to search for a longer common substring.

} else {

// If there's no common substring of length 'mid',

high = mid - 1;

// Move the upper boundary down to search for a shorter common substring.

}

}

return answer;

// Return the length of the longest common substring found.

}

private static boolean exists(int k, String X, String Y) {

// Method to check if a common substring of length 'k' exists in both X and Y.

for (int i = 0; i <= X.length() - k; i++) {

// Iterate through all possible substrings of length 'k' in X.

for (int j = 0; j <= Y.length() - k; j++) {

// For each substring in X, iterate through all possible substrings of length 'k' in Y.

if (X.substring(i, i + k).equals(Y.substring(j, j + k))) {

// Check if the substring in X is equal to the substring in Y.

return true;

// If a common substring is found, return true immediately.

}

}

}

return false;

// After checking all possibilities, if no common substring is found, return false.

}

public static void main(String[] args) {

// Main method to run the program.

String X = "GeeksforGeeks";

// Input string X.

String Y = "GeeksQuiz";

// Input string Y.

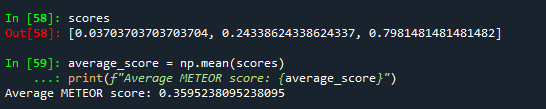
System.out.println(longestCommonSubstr(X, Y));

// Call the longestCommonSubstr method with X and Y as inputs and print the result.

// It prints the length of the longest common substring between X and Y.

}

Evaluation:



7. (https://www.geeksforgeeks.org/find-minimum-area-of-rectangle-formed-from-given-shuffled-coordinates/)

// Java code to count the number of possible triangles using

// brute force approach

import java.io.\*;

import java.util.\*;

class GFG {

// Function to count all possible triangles with arr[]

// elements

static int findNumberOfTriangles(int arr[], int n)

{

// Sort the array

Arrays.sort(arr);

// Count of triangles

int count = 0;

// The three loops select three different values

// from array

for (int i = 0; i < n; i++)

for (int j = i + 1; j < n; j++)

for (int k = j + 1; k < n; k++)

if (arr[i] + arr[j] > arr[k])

count++;

return count;

}

// Driver code

public static void main(String[] args)

{

int arr[] = { 10, 21, 22, 100, 101, 200, 300 };

int size = arr.length;

// Function call

System.out.println(

"Total number of triangles possible is "

+ findNumberOfTriangles(arr, size));

}

}

// This code is contributed by Sania Kumari Gupta

Please generate comments after each line of code

import java.io.\*;

import java.util.\*;

// Imports the IO and utility classes from the Java library. Though IO is not used in this snippet, utility classes like Arrays are used for sorting.

class GFG {

// Defines a class named GFG.

static int findNumberOfTriangles(int arr[], int n) {

// Declares a static method to find the total number of triangles that can be formed with the array elements as sides.

Arrays.sort(arr);

// Sorts the array in ascending order. This is crucial for the logic that checks for the triangle inequality theorem efficiently.

int count = 0;

// Initializes a counter to keep track of the number of triangles that can be formed.

for (int i = 0; i < n; i++)

// Outer loop iterates through each element in the array.

for (int j = i + 1; j < n; j++)

// Middle loop selects the second side of the triangle, ensuring it's different from the first side.

for (int k = j + 1; k < n; k++)

// Inner loop selects the third side of the triangle, ensuring it's different from the first two sides.

if (arr[i] + arr[j] > arr[k])

// Checks if the sum of any two sides is greater than the third side, which is a requirement for a valid triangle.

count++;

// Increments the count if the condition is met, indicating a valid triangle formation.

return count;

// Returns the total count of triangles that can be formed.

}

public static void main(String[] args) {

// Main method to run the program.

int arr[] = { 10, 21, 22, 100, 101, 200, 300 };

// Initializes an array with elements representing possible sides of triangles.

int size = arr.length;

// Calculates the size (length) of the array.

System.out.println(

"Total number of triangles possible is "

+ findNumberOfTriangles(arr, size));

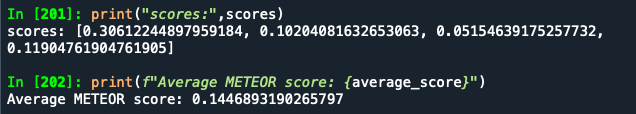
// Prints out the total number of triangles that can be formed by calling the findNumberOfTriangles method and passing the array and its size.

}

}

Evaluation:

**The comment comparison for each line of code**



The comment comparison for function description

Screen Shot 2024-03-20 at 10.01.47 AM

8. (https://www.geeksforgeeks.org/inverse-ackermann-function/)

// Java function for the above algorithm:

import java.io.\*;

public class InverseAckermann {

//The inverse Ackermann function is the function that returns the pair of integers (m, n) that satisfies the equation A(m, n) = x, where x is a given value.

public static int inverseAckermann(int n)

{

// Check if the input is small enough

// to solve directly

if (n <= 4) {

return n;

}

// Divide the problem into

// two smaller problems

int a = inverseAckermann(n - 1);

int b = inverseAckermann(n - 2);

// Combine the solutions of the

// two smaller problems

return a + b;

}

// Driver Code

public static void main(String[] args)

{

// Define the input

int n = 10;

// Solve the problem using the

// inverse Ackermann algorithm

int result = inverseAckermann(n);

// Print the result

System.out.println("Result: " + result);

}

}

Please generate comments after each line of code

import java.io.\*;

// Imports the IO package, although it's not used in this snippet.

public class InverseAckermann {

// Declares a public class named InverseAckermann.

public static int inverseAckermann(int n) {

// Declares a public static method named inverseAckermann that takes an integer n and returns an integer. The method name suggests it's related to the inverse of the Ackermann function, but the implemented logic does not accurately represent any known inverse of the Ackermann function.

if (n <= 4) {

// Base case: if n is less than or equal to 4, the method simply returns n.

return n;

}

int a = inverseAckermann(n - 1);

// Recursively calls itself with n - 1 and stores the result in variable a.

int b = inverseAckermann(n - 2);

// Recursively calls itself with n - 2 and stores the result in variable b.

return a + b;

// Returns the sum of a and b. This logic resembles that of the Fibonacci sequence rather than any direct relation to the Ackermann function or its inverse.

}

public static void main(String[] args) {

// The main method, which is the entry point of the program.

int n = 10;

// Initializes an integer n with a value of 10.

int result = inverseAckermann(n);

// Calls the inverseAckermann method with n as an argument and stores the returned value in the result variable.

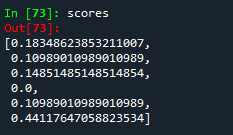
System.out.println("Result: " + result);

// Prints the result to the console.

}

}

Evaluation:



A black background with white text and numbers

Description automatically generated

9. (https://www.geeksforgeeks.org/maximum-length-of-sequence-formed-from-cost-n/)

// Java program for the above approach

import java.io.\*;

class GFG {

// Function to count total number of digits from numbers

// 1 to N

static int totalDigits(int N)

{

int cnt = 0;

for (int i = 1; i <= N; i \*= 10) {

cnt += (N - i + 1);

}

return cnt;

}

// Function to find Maximum length of Sequence that can

// be formed from cost N

static void findMaximumLength(int N)

{

int low = 1, high = 1e9;

while (high - low > 1) {

int mid = low + (high - low) / 2;

// Check if cost for number of digits from 1 to

// N is less than equal to N

if (totalDigits(mid) <= N) {

// atleast mid will be the answer

low = mid;

}

else {

// ignore right search space

high = mid - 1;

}

}

// check if high can be the answer

if (totalDigits(high) <= N) {

System.out.println(high);

}

// else low can be the answer

else if (totalDigits(low) <= N) {

System.out.println(low);

}

// else answer will be zero.

else {

System.out.println(0);

}

}

public static void main(String[] args)

{

int N = 11;

// Function call

findMaximumLength(N);

int N1 = 189;

// Function call

findMaximumLength(N1);

}

}

// This code is contributed by lokeshmvs21.

Please generate comments after each line of code

import java.io.\*;

// Imports the IO package, not used in this snippet.

class GFG {

// Defines a class named GFG.

static int totalDigits(int N) {

// A static method that calculates the total number of digits up to N.

int cnt = 0;

// Initialize a counter to store the total number of digits.

for (int i = 1; i <= N; i \*= 10) {

// Loop through powers of 10 (1, 10, 100, ...) up to N.

cnt += (N - i + 1);

// Add to the counter the range of numbers that have the same number of digits.

}

return cnt;

// Return the total count of digits.

}

static void findMaximumLength(int N) {

// A static method to find the maximum number of digits in a range that adds up to a total of N digits.

int low = 1, high = (int) 1e9;

// Initialize binary search bounds, assuming the upper limit to be 1 billion.

while (high - low > 1) {

// Continue narrowing down the search space while the difference between high and low is greater than 1.

int mid = low + (high - low) / 2;

// Calculate the midpoint of the current search space to use in binary search.

if (totalDigits(mid) <= N) {

// If the total digits up to mid are less than or equal to N, narrow the search towards the higher half.

low = mid;

}

else {

// Otherwise, adjust the search towards the lower half.

high = mid - 1;

}

}

// After exiting the loop, check which of the bounds closely matches the requirement.

if (totalDigits(high) <= N) {

// If high is a valid answer, print it.

System.out.println(high);

}

else if (totalDigits(low) <= N) {

// If not high but low is a valid answer, print it.

System.out.println(low);

}

else {

// If neither is valid, return 0 indicating no number in the range meets the criteria.

System.out.println(0);

}

}

public static void main(String[] args) {

// The main method to test the functionality of the findMaximumLength method.

int N = 11;

// Example 1: Find the maximum number length for a total of 11 digits.

findMaximumLength(N);

// Calls the method with the total digits N.

int N1 = 189;

// Example 2: Find the maximum number length for a total of 189 digits.

findMaximumLength(N1);

// Calls the method with the total digits N1.

}

}

Evaluation:

