QQQ Doorthy

import java.util.\*;

public class Main {

public static int minMovesToCross(int N, int[] bricks) {

if (N == 1) {

return 0; // Already at the last brick

}

// Step 1: Create a map from value to list of indices

Map<Integer, List<Integer>> valueToIndices = new HashMap<>();

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

valueToIndices.computeIfAbsent(bricks[i], k -> new ArrayList<>()).add(i);

}

// Step 2: Initialize BFS

Queue<int[]> queue = new LinkedList<>(); // (current index, moves)

queue.offer(new int[]{0, 0});

boolean[] visited = new boolean[N];

visited[0] = true;

while (!queue.isEmpty()) {

int[] current = queue.poll();

int currentIndex = current[0];

int currentMoves = current[1];

// Step 3: Explore all possible moves from the current brick

// Move to the next brick

if (currentIndex + 1 < N && !visited[currentIndex + 1]) {

if (currentIndex + 1 == N - 1) {

return currentMoves + 1;

}

queue.offer(new int[]{currentIndex + 1, currentMoves + 1});

visited[currentIndex + 1] = true;

}

// Move to the previous brick

if (currentIndex - 1 >= 0 && !visited[currentIndex - 1]) {

queue.offer(new int[]{currentIndex - 1, currentMoves + 1});

visited[currentIndex - 1] = true;

}

// Move to any brick with the same value

List<Integer> sameValueIndices = valueToIndices.get(bricks[currentIndex]);

for (int index : sameValueIndices) {

if (!visited[index]) {

if (index == N - 1) {

return currentMoves + 1;

}

queue.offer(new int[]{index, currentMoves + 1});

visited[index] = true;

}

}

// Clear the list to prevent redundant checks

sameValueIndices.clear();

}

return -1; // If we never reach the last brick (shouldn't happen with valid input)

}

public static void main(String[] args) {

// Example static input:

int N = 5;

int[] bricks = {1, 2,3,4,1};

System.out.println(minMovesToCross(N, bricks));

}

}

2. Xavier’s QuesƟon

#include <iostream>

#include <vector>

#include <unordered\_map>

#include <queue>

#include <stack>

#include <limits>

#include <algorithm>

using namespace std;

int main() {

// Static input values

int N = 12;

int K = 4;

vector<int> powers = {5, 10, 4, 8, 3, 6, 5, 10, 4, 8, 3, 6};

// Validate if N is divisible by K

if (N % K != 0) {

cout << -1 << endl;

return 0;

}

// Count frequencies of each power level

unordered\_map<int, int> counters;

for (int p : powers) {

counters[p]++;

}

// Use a priority queue to process teams

priority\_queue<pair<int, int>> pq;

for (auto &it : counters) {

pq.push({it.second, it.first});

}

stack<pair<int, int>> team;

int ans = 0;

int maxValue = 0;

int minValue = numeric\_limits<int>::max();

// Form teams and calculate the minimum difference in each team

while (pq.size() >= K) {

while (!pq.empty()) {

auto p = pq.top();

pq.pop();

maxValue = max(maxValue, p.second);

minValue = min(minValue, p.second);

--p.first;

team.push({p.first, p.second});

if (team.size() == K) {

ans += maxValue - minValue;

while (!team.empty()) {

if (team.top().first > 0) {

pq.push(team.top());

}

team.pop();

}

maxValue = 0;

minValue = numeric\_limits<int>::max();

break; // break inner while loop to form next team

}

}

}

// If there are remaining elements in pq, it means forming teams is not possible

if (!pq.empty()) {

cout << -1 << endl;

return 0;

}

// Output the answer as per the required format

cout << ans << endl;

return 0;

}

Mincut

// Approach - 0 Recursion

import java.util.\*;

class Main {

public static int minCut(String str) {

int n = str.length();

return f(0, n, str) - 1;

}

public static boolean isPalindrome(int i, int j, String s) {

while (i < j) {

if (s.charAt(i) != s.charAt(j)) return false;

i++;

j--;

}

return true;

}

public static int f(int i, int n, String str) {

// Base case:

if (i == n) return 0;

int minCost = Integer.MAX\_VALUE;

// String[i...j]

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

if (isPalindrome(i, j, str)) {

int cost = 1 + f(j + 1, n, str);

minCost = Math.min(minCost, cost);

}

}

return minCost;

}

public static void main(String[] args){

int ans =minCut("qwdefijewjwq");

System.out.println(ans);

}

}

String s1 and s2 super palindrome

import java.util.\*;

class Main {

public static int superpalindromesInRange(String left, String right) {

int ans = 9 >= Long.parseLong(left) && 9 <= Long.parseLong(right) ? 1 : 0;

for (int dig = 1; dig < 10; dig++) {

boolean isOdd = dig % 2 > 0 && dig != 1;

int innerLen = (dig >> 1) - 1,

innerLim = Math.max(1, (int)Math.pow(2, innerLen)),

midPos = dig >> 1, midLim = isOdd ? 3 : 1;

for (int edge = 1; edge < 3; edge++) {

char[] pal = new char[dig];

Arrays.fill(pal, '0');

pal[0] = (char)(edge + 48);

pal[dig-1] = (char)(edge + 48);

if (edge == 2) {

innerLim = 1;

midLim = Math.min(midLim, 2);

}

for (int inner = 0; inner < innerLim; inner++) {

if (inner > 0) {

String innerStr = Integer.toString(inner, 2);

while (innerStr.length() < innerLen)

innerStr = "0" + innerStr;

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

pal[1+i] = innerStr.charAt(i);

pal[dig-2-i] = innerStr.charAt(i);

}

}

for (int mid = 0; mid < midLim; mid++) {

if (isOdd) pal[midPos] = (char)(mid + 48);

String palin = new String(pal);

long square = Long.parseLong(palin) \* Long.parseLong(palin);

if (square > Long.parseLong(right)) return ans;

if (square >= Long.parseLong(left) && isPal(Long.toString(square))) ans++;

}

}

}

}

return ans;

}

public static boolean isPal(String str) {

for (int i = 0, j = str.length() - 1; i < j; i++, j--)

if (str.charAt(i) != str.charAt(j)) return false;

return true;

}

public static void main(String[] args){

System.out.println(superpalindromesInRange("4","9"));

}

}

Three equal parts  
  
class Solution {

public int[] threeEqualParts(int[] arr) {

int oneSum = 0;

for(int x : arr) oneSum +=x;

if(oneSum % 3 !=0)return new int[]{-1,-1};

if(oneSum == 0)return new int[]{0,2};

int oneCount = oneSum / 3;

int i = 0;

int firstOne = -1;

int n= arr.length;

int lastZeroCount = 0;

i = n-1;

while(arr[i] ==0){

lastZeroCount++;

i--;

}

i=0;

while(oneCount > 0){

if(arr[i] == 1 && firstOne == -1)firstOne = i;

oneCount-=arr[i++];

}

while(lastZeroCount > 0){

if(arr[i++] == 1)return new int[]{-1,-1};

lastZeroCount--;

}

int k = i--;

int t = firstOne;

while(arr[k] == 0)k++;

while(firstOne <=i){

if(arr[firstOne++] != arr[k++])return new int[]{-1,-1};

}

int j = k;

while(arr[k] == 0)k++;

firstOne = t;

while(firstOne <=i){

if(arr[firstOne++] != arr[k++])return new int[]{-1,-1};

}

return new int[]{i,j};

}

}

Arthamatic slice  
  
  
class Solution {

public int numberOfArithmeticSlices(int[] nums) {

int res =0;

int key = Integer.MIN\_VALUE;

int val = Integer.MIN\_VALUE;

for(int i=1; i< nums.length; i++){

int d = nums[i]-nums[i-1];

if(key != d){

key = d;

val = 0;

}

else val++;

if(val >= 1) res+= val;

}

return res;

}

}  
  
  
  
find kth smallest lexicographic number  
  
class Solution {

public int findKthNumber(int n, int k) {

int prefix=1;

for(int count=1;count<k;){

int currCount=getCountWithPrefix(prefix,prefix+1,n);

if(currCount+count<=k){

count+=currCount;

prefix++;

}else{

prefix\*=10;

count++;

}

}

return prefix;

}

private int getCountWithPrefix(long startPrefix,long endPrefix,int max){

int count=0;

while(startPrefix<=max){

count+=Math.min(max+1,endPrefix)-startPrefix;

startPrefix\*=10;

endPrefix\*=10;

}

return count;

}

}

longest continuous subarray  
  
  
class Solution {

public int longestSubarray(int[] nums, int limit) {

LinkedList<Integer> increase = new LinkedList<>();

LinkedList<Integer> decrease = new LinkedList<>();

int max = 0;

int left = 0;

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

int n = nums[i];

while (increase.size() > 0 && n < increase.getLast()) {

increase.removeLast();

}

increase.add(n);

while (decrease.size() > 0 && n > decrease.getLast()) {

decrease.removeLast();

}

decrease.add(n);

while (decrease.getFirst() - increase.getFirst() > limit) {

if (nums[left] == decrease.getFirst()) {

decrease.removeFirst();

}

if (nums[left] == increase.getFirst()) {

increase.removeFirst();

}

left++;

}

int size = i - left + 1;

max = Math.max(max, size);

}

return max;

}

}  
  
  
min distance question

class Solution {

public int minDistance(String word1, String word2) {

final int m = word1.length();//first word length

final int n = word2.length();///second word length

// dp[i][j] := min # of operations to convert word1[0..i) to word2[0..j)

int[][] dp = new int[m + 1][n + 1];

for (int i = 1; i <= m; ++i)

dp[i][0] = i;

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

dp[0][j] = j;

for (int i = 1; i <= m; ++i)

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

if (word1.charAt(i - 1) == word2.charAt(j - 1))//same characters

dp[i][j] = dp[i - 1][j - 1];//no operation

else

dp[i][j] = Math.min(dp[i - 1][j - 1], Math.min(dp[i - 1][j], dp[i][j - 1])) + 1; //replace //delete //insert

return dp[m][n];

}

}

Maximum profit

import java.util.Arrays;

public class Main {

public static void main(String[] args) {

// Sample test case

int[] prices = {2,4,1};

int k = 2;

Solution sol = new Solution();

int result = sol.maxProfit(k, prices);

System.out.println("The maximum profit is: " + result); // Output should be 7

}

}

class Solution {

int[][][] dp;

public int f(int[] v, int i, int k, boolean on) {

if (i >= v.length) return 0;

if (dp[i][k][on ? 1 : 0] != -1) return dp[i][k][on ? 1 : 0];

int profit = 0;

profit = Math.max(profit, f(v, i + 1, k, on));

if (on) {

profit = Math.max(profit, v[i] + f(v, i + 1, k - 1, false));

} else if (k > 0) {

profit = Math.max(profit, f(v, i + 1, k, true) - v[i]);

}

return dp[i][k][on ? 1 : 0] = profit;

}

public int maxProfit(int k, int[] prices) {

dp = new int[prices.length][k + 1][2];

for (int[][] arr2d : dp) {

for (int[] arr1d : arr2d) {

Arrays.fill(arr1d, -1);

}

}

return f(prices, 0, k, false);

}

}

Number to integer

class Solution {

//globally declare arrays

String[] ones = {"", " One", " Two", " Three", " Four", " Five", " Six", " Seven", " Eight", " Nine", " Ten", " Eleven", " Twelve", " Thirteen", " Fourteen", " Fifteen", " Sixteen", " Seventeen", " Eighteen", " Nineteen"};

String[] tens = {"", " Ten", " Twenty", " Thirty", " Forty", " Fifty", " Sixty", " Seventy", " Eighty", " Ninety"};

String[] thousands = {"", " Thousand", " Million", " Billion"};

//helper function

public String helper(int n) {

if (n < 20)

return ones[n];

if (n < 100)

return tens[n / 10] + helper(n % 10);

if (n < 1000)

return helper(n / 100) + " Hundred" + helper(n % 100);

for (int i = 3; i >= 0; i--) {

if (n >= Math.pow(1000, i)) {

return helper((int)(n / Math.pow(1000, i))) + thousands[i] + helper((int)(n % Math.pow(1000, i)));

}

}

return "";

}

public String numberToWords(int num) {

// edge case

if (num == 0)

return "Zero";

return helper(num).substring(1);

}

}

Maximum rectangle

class Solution {

public int largestRectangleArea(int[] heights) {

Stack<Integer> st = new Stack<>();

//PSE

int[] pse = new int[heights.length];

pse[0] = -1;

st.push(0);

for (int i = 1; i < heights.length; i++) {

while(st.size()>0 && heights[st.peek()]>=heights[i]) st.pop();

if(st.size()==0) pse[i] = -1;

else pse[i] = st.peek();

st.push(i);

}

st.clear();

//NSE

int[] nse = new int[heights.length];

nse[nse.length-1] = nse.length;

st.push(nse.length-1);

for (int i = heights.length-2; i >= 0; i--) {

while(st.size()>0 && heights[st.peek()]>=heights[i]) st.pop();

if(st.size()==0) nse[i] = nse.length;

else nse[i] = st.peek();

st.push(i);

}

st.clear();

//Area

int area = Integer.MIN\_VALUE;

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

int temp = (nse[i] - pse[i] - 1)\*heights[i];

if(temp > area) area = temp;

}

return area;

}

}

Pizaa

import java.util.\*;

public class Main {

public static void main(String[] args) {

// Static input values

int n = 6;

int x = 4;

int y = 3;

int[] a = {4,6,3,7,2,4};

int[] b = {5,3,4,2,3,7};

// Priority queue to store pairs of absolute differences and their indices

PriorityQueue<AbstractMap.SimpleEntry<Integer, Integer>> pq = new PriorityQueue<>(

(p1, p2) -> Integer.compare(Math.abs(b[p2.getValue()] - a[p2.getValue()]), Math.abs(b[p1.getValue()] - a[p1.getValue()])));

// Insert all pairs (|a[i] - b[i]|, i) into the priority queue

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

pq.offer(new AbstractMap.SimpleEntry<>(i, Math.abs(a[i] - b[i])));

}

int total = 0;

// Process elements from the priority queue

while (!pq.isEmpty()) {

AbstractMap.SimpleEntry<Integer, Integer> pair = pq.poll();

int idx = pair.getKey();

int valueA = a[idx];

int valueB = b[idx];

if (x > 0 && y > 0) {

if (valueA > valueB) {

total += valueA;

x--;

} else if (valueA < valueB) {

total += valueB;

y--;

} else {

if (x > y) {

total += valueA;

x--;

} else {

total += valueB;

y--;

}

}

} else if (x > 0) {

total += valueA;

x--;

} else if (y > 0) {

total += valueB;

y--;

}

}

// Output the total sum

System.out.println(total);

}

}