**Assignment8**

**1.**

function minimumDeleteSum(s1, s2) {

const m = s1.length;

const n = s2.length;

const dp = Array.from({ length: m + 1 }, () => Array(n + 1).fill(0));

for (let i = 1; i <= m; i++) {

dp[i][0] = dp[i - 1][0] + s1.charCodeAt(i - 1);

}

for (let j = 1; j <= n; j++) {

dp[0][j] = dp[0][j - 1] + s2.charCodeAt(j - 1);

}

for (let i = 1; i <= m; i++) {

for (let j = 1; j <= n; j++) {

if (s1[i - 1] === s2[j - 1]) {

dp[i][j] = dp[i - 1][j - 1];

} else {

dp[i][j] = Math.min(

dp[i - 1][j] + s1.charCodeAt(i - 1),

dp[i][j - 1] + s2.charCodeAt(j - 1)

);

}

}

}

return dp[m][n];

}

const s1 = "sea";

const s2 = "eat";

console.log(minimumDeleteSum(s1, s2));

**Output:** 231

**2.**

function checkValidString(s) {

const stack = [];

const starStack = [];

for (let i = 0; i < s.length; i++) {

const char = s[i];

if (char === '(') {

stack.push(i);

} else if (char === '\*') {

starStack.push(i);

} else if (char === ')') {

if (stack.length > 0) {

stack.pop();

} else if (starStack.length > 0) {

starStack.pop();

} else {

return false;

}

}

}

while (stack.length > 0 && starStack.length > 0) {

if (stack.pop() > starStack.pop()) {

return false;

}

}

return stack.length === 0;

}

const s = "()";

console.log(checkValidString(s));

**Output:** true

**3.**

function minDistance(word1, word2) {

const m = word1.length;

const n = word2.length;

const dp = Array.from({ length: m + 1 }, () => Array(n + 1).fill(0));

for (let i = 1; i <= m; i++) {

dp[i][0] = i;

}

for (let j = 1; j <= n; j++) {

dp[0][j] = j;

}

for (let i = 1; i <= m; i++) {

for (let j = 1; j <= n; j++) {

if (word1[i - 1] === word2[j - 1]) {

dp[i][j] = dp[i - 1][j - 1];

} else {

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

}

}

}

return dp[m][n];

}

const word1 = "sea";

const word2 = "eat";

console.log(minDistance(word1, word2));

**Output:** 2

**4.**

class TreeNode {

constructor(val) {

this.val = val;

this.left = null;

this.right = null;

}

}

function buildTree(s) {

if (!s || s.length === 0) {

return null;

}

const openingParenIndex = s.indexOf('(');

if (openingParenIndex === -1) {

const rootVal = parseInt(s);

return new TreeNode(rootVal);

}

const rootVal = parseInt(s.substring(0, openingParenIndex));

const root = new TreeNode(rootVal);

let leftSubtreeEndIndex = findMatchingClosingParenIndex(s, openingParenIndex);

const leftSubtreeStr = s.substring(openingParenIndex + 1, leftSubtreeEndIndex);

root.left = buildTree(leftSubtreeStr);

if (leftSubtreeEndIndex < s.length - 1) {

const rightSubtreeStr = s.substring(leftSubtreeEndIndex + 2, s.length - 1);

root.right = buildTree(rightSubtreeStr);

}

return root;

}

function findMatchingClosingParenIndex(s, openingParenIndex) {

let count = 0;

for (let i = openingParenIndex; i < s.length; i++) {

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

count++;

} else if (s[i] === ')') {

count--;

}

if (count === 0) {

return i;

}

}

return -1; // No matching closing parenthesis found

}

function inorderTraversal(root) {

const result = [];

function traverse(node) {

if (!node) {

return;

}

traverse(node.left);

result.push(node.val);

traverse(node.right);

}

traverse(root);

return result;

}

Example:

const s = "4(2(3)(1))(6(5))";

const root = buildTree(s);

const result = inorderTraversal(root);

console.log(result);

**Output:** [4, 2, 6, 3, 1, 5]

**5.**

function compress(chars) {

let read = 0;

let write = 0;

let count = 1;

for (let i = 1; i <= chars.length; i++) {

if (i < chars.length && chars[i] === chars[i - 1]) {

count++;

} else {

chars[write] = chars[read];

write++;

if (count > 1) {

const countStr = count.toString();

for (let j = 0; j < countStr.length; j++) {

chars[write] = countStr[j];

write++;

}

}

count = 1;

read = i;

}

}

return write;

}

const chars = ["a", "a", "b", "b", "c", "c", "c"];

const newLength = compress(chars);

const compressedChars = chars.slice(0, newLength);

console.log(newLength);

**Output:** 6

console.log(compressedChars);

**Output:** ["a", "2", "b", "2", "c", "3"]

**6.**

function findAnagrams(s, p) {

const pFreq = Array(26).fill(0);

const sFreq = Array(26).fill(0);

const result = [];

for (let i = 0; i < p.length; i++) {

pFreq[p.charCodeAt(i) - 'a'.charCodeAt(0)]++;

}

let left = 0;

let right = 0;

while (right < s.length) {

sFreq[s.charCodeAt(right) - 'a'.charCodeAt(0)]++;

if (right - left + 1 === p.length) {

if (arraysAreEqual(pFreq, sFreq)) {

result.push(left);

}

sFreq[s.charCodeAt(left) - 'a'.charCodeAt(0)]--;

left++;

}

right++;

}

return result;

}

function arraysAreEqual(arr1, arr2) {

for (let i = 0; i < arr1.length; i++) {

if (arr1[i] !== arr2[i]) {

return false;

}

}

return true;

}

const s = "cbaebabacd";

const p = "abc";

const result = findAnagrams(s, p);

console.log(result);

**Output:** [0, 6]

**7.**

function decodeString(s) {

const stack = [];

let count = 0;

let decodedString = '';

for (let i = 0; i < s.length; i++) {

const c = s.charAt(i);

if (c >= '0' && c <= '9') {

count = count \* 10 + parseInt(c);

} else if (c === '[') {

stack.push(count);

stack.push(decodedString);

count = 0;

decodedString = '';

} else if (c === ']') {

const prevDecodedString = stack.pop();

const prevCount = stack.pop();

decodedString = prevDecodedString + decodedString.repeat(prevCount);

} else {

decodedString += c;

}

}

return decodedString;

}

const s = "3[a]2[bc]";

const decodedString = decodeString(s);

console.log(decodedString);

**Output:** "aaabcbc"

**8.**

function buddyStrings(s, goal) {

if (s.length !== goal.length) {

return false;

}

const diffIndexes = [];

const diffChars = [];

for (let i = 0; i < s.length; i++) {

if (s[i] !== goal[i]) {

diffIndexes.push(i);

diffChars.push(s[i]);

diffChars.push(goal[i]);

}

}

if (diffIndexes.length !== 2) {

return false;

}

return (

diffChars[0] === diffChars[3] && diffChars[1] === diffChars[2]

);

}

const s = "ab";

const goal = "ba";

const result = buddyStrings(s, goal);

console.log(result);

**Output:** true