Assignment 8 Solution - String | DSA

Question 1

Given two strings s1 and s2, return the lowest ASCII sum of deleted characters to make two strings equal.

Example 1:
Input: s1 = "sea", s2 = "eat"
Output: 231
Explanation: Deleting "s" from "sea" adds the ASCII value of "s" (115) to the sum.
Deleting "t" from "eat" adds 116 to the sum.
At the end, both strings are equal, and 115 + 116 = 231 is the minimum sum possible to achieve this.

Solution Code:

```
package in.ineuron.pptAssignment08;

public class LowestASCIIDeletedSum {

	public static void main(String[] args) {
		String s1 = "sea", s2 = "eat";
		System.out.println(minimumDeleteSum(s1, s2));
	}

	public static int minimumDeleteSum(String s1, String s2) {
		int m = s1.length();
		int n = s2.length();

		// Create a 2D array to store the minimum ASCII sum of deleted characters
		int[][] dp = new int[m + 1][n + 1];

		// Initialize the first row and column
		for (int i = 1; i <= m; i++) {
			dp[i][0] = dp[i - 1][0] + s1.charAt(i - 1);
		}

		for (int j = 1; j <= n; j++) {
			dp[0][j] = dp[0][j - 1] + s2.charAt(j - 1);
		}

		// Fill the remaining cells of the dp array
		for (int i = 1; i <= m; i++) {
			for (int j = 1; j <= n; j++) {
				if (s1.charAt(i - 1) == s2.charAt(j - 1)) {
					dp[i][j] = dp[i - 1][j - 1];
				} else {
					dp[i][j] = Math.min(dp[i - 1][j] + s1.charAt(i - 1), dp[i][j - 1] +
					s2.charAt(j - 1));
```

```
                        }
                    }
                }

                // The lowest ASCII sum of deleted characters is stored in the bottom-right cell
                // of the dp array
                return dp[m][n];
            }
        }
```

Question 2

Given a string s containing only three types of characters: '(', ')' and '', return true if s is valid.
The following rules define a valid string:
- Any left parenthesis '(' must have a corresponding right parenthesis ')'.
- Any right parenthesis ')' must have a corresponding left parenthesis '('.
- Left parenthesis '(' must go before the corresponding right parenthesis ')'.
- '' could be treated as a single right parenthesis ')' or a single left parenthesis '(' or an empty string "".

Example 1:
Input: s = "()"
Output: True

Solution Code:

```
package in.ineuron.pptAssignment08;
import java.util.Stack;
public class ValidParentheses {

    public static void main(String[] args) {
        String s = "()";
        System.out.println(isValid(s));
    }

    public static boolean isValid(String s) {
        Stack<Character> stack = new Stack<>();

        for (char c : s.toCharArray()) {
            if (c == '(' || c == '[' || c == '{') {
                stack.push(c);
            } else if (c == ')' && !stack.isEmpty() && stack.peek() == '(') {
                stack.pop();
            } else if (c == ']' && !stack.isEmpty() && stack.peek() == '[') {
                stack.pop();
```

```
                } else if (c == '}' && !stack.isEmpty() && stack.peek() == '{') {
                        stack.pop();
                } else {
                        return false;
                }
            }
            return stack.isEmpty();
        }
    }
```

Question 3

Given two strings word1 and word2, return the minimum number of steps required to make word1 and word2 the same.
In one step, you can delete exactly one character in either string.

Example 1:
Input: word1 = "sea", word2 = "eat"
Output: 2

Explanation: You need one step to make "sea" to "ea" and another step to make "eat" to "ea".

Solution Code:

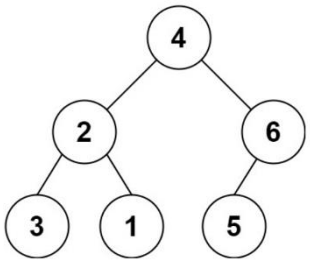
```
package in.ineuron.pptAssignment08;

public class DeleteOperation {

    public static void main(String[] args) {
        String word1 = "sea", word2 = "eat";
        System.out.println(minDistance(word1, word2));

    }

    public static int minDistance(String word1, String word2) {
        int m = word1.length();
        int n = word2.length();

        // Create a 2D array to store the minimum number of steps
        int[][] dp = new int[m + 1][n + 1];

        // Fill the first row and column
        for (int i = 0; i <= m; i++) {
            dp[i][0] = i;
```

```
                }

                for (int j = 0; j <= n; j++) {
                        dp[0][j] = j;
                }

                // Fill the remaining cells of the dp array
                for (int i = 1; i <= m; i++) {
                        for (int j = 1; j <= n; j++) {
                                if (word1.charAt(i - 1) == word2.charAt(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;
                                }
                        }
                }
// The minimum number of steps is stored in the bottom-right cell of the dp array
                return dp[m][n];
        }
}
```

Question 4

You need to construct a binary tree from a string consisting of parenthesis and integers.
The whole input represents a binary tree. It contains an integer followed by zero, one or two pairs of parentheses. The integer represents the root's value and a pair of parenthesis contains a child binary tree with the same structure.
You always start to construct the left child node of the parent first if it exists.

Example 1:



Input: s = "4(2(3)(1))(6(5))"
Output: [4,2,6,3,1,5]

Solution Code:

```
package in.ineuron.pptAssignment08;

class TreeNode {
	int val;
	TreeNode left;
	TreeNode right;

	TreeNode(int val) {
		this.val = val;
	}
}

public class ConstructBinaryTreeFromString {

	public static void main(String[] args) {
		String s = "4(2(3)(1))(6(5))";
		TreeNode node = new TreeNode(0);
	}

	public static TreeNode str2tree(String s) {
		if (s.isEmpty()) {
			return null;
		}

		int firstParen = s.indexOf("(");

		// Create the root node
		int val = (firstParen == -1) ? Integer.parseInt(s) : Integer.parseInt(s.substring(0,
		firstParen));
```

```
                TreeNode root = new TreeNode(val);

                if (firstParen == -1) {
                        return root;
                }

                int start = firstParen;
                int openParenCount = 0;

                // Find the substring for the left child
                for (int i = start; i < s.length(); i++) {
                        if (s.charAt(i) == '(') {
                                openParenCount++;
                        } else if (s.charAt(i) == ')') {
                                openParenCount--;
                        }

                        if (openParenCount == 0 && start == firstParen) {
                                root.left = str2tree(s.substring(start + 1, i));
                                start = i + 1;
                        } else if (openParenCount == 0) {
                                root.right = str2tree(s.substring(start + 1, i));
                        }
                }
                return root;
        }
    }
```

Question 5

Given an array of characters chars, compress it using the following algorithm:
Begin with an empty string s. For each group of consecutive repeating characters in chars:
- If the group's length is 1, append the character to s.
- Otherwise, append the character followed by the group's length.

The compressed string s should not be returned separately, but instead, be stored in the input character array chars. Note that group lengths that are 10 or longer will be split into multiple characters in chars.
After you are done modifying the input array, return the new length of the array.
You must write an algorithm that uses only constant extra space.

Example 1:
Input: chars = ["a","a","b","b","c","c","c"]
Output: Return 6, and the first 6 characters of the input array should be:
["a","2","b","2","c","3"]
Explanation:The groups are "aa", "bb", and "ccc". This compresses to "a2b2c3".

Solution Code :

```
package in.ineuron.pptAssignment08;

public class CompressCharacters {

	public static int compress(char[] chars) {
		if (chars.length == 0) {
			return 0;
		}

		int index = 0;
		int count = 1;

		for (int i = 1; i <= chars.length; i++) {
			if (i < chars.length && chars[i] == chars[i - 1]) {
				count++;
			} else {
				chars[index++] = chars[i - 1];
				if (count > 1) {
					String countStr = String.valueOf(count);
					for (char c : countStr.toCharArray()) {
						chars[index++] = c;
					}
				}
				count = 1;
			}
		}

		return index;
	}
```

```
    public static void main(String[] args) {
        char[] chars = { 'a', 'a', 'b', 'b', 'c', 'c', 'c' };
        int newLength = compress(chars);

        System.out.println("New Length: " + newLength);
        System.out.print("Compressed Array: ");
        for (int i = 0; i < newLength; i++) {
            System.out.print(chars[i] + " ");
        }
    }
}
```

Question 6

Given two strings s and p, return an array of all the start indices of p's anagrams in s. You may return the answer in any order.

An Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1:
Input: s = "cbaebabacd", p = "abc"
Output: [0,6]

Explanation:
The substring with start index = 0 is "cba", which is an anagram of "abc".
The substring with start index = 6 is "bac", which is an anagram of "abc".

Solution Code:

```
package in.ineuron.pptAssignment08;

import java.util.ArrayList;
import java.util.List;
public class AnagramIndices {
    public static List<Integer> findAnagrams(String s, String p) {
        List<Integer> result = new ArrayList<>();

        int[] charCountP = new int[26];
        for (char c : p.toCharArray()) {
            charCountP[c - 'a']++;
        }
        int[] charCountWindow = new int[26];
        int windowSize = p.length();

        for (int i = 0; i < s.length(); i++) {
            charCountWindow[s.charAt(i) - 'a']++;
```

```
                if (i >= windowSize) {
                    charCountWindow[s.charAt(i - windowSize) - 'a']--;
                }

                if (matches(charCountP, charCountWindow)) {
                    result.add(i - windowSize + 1);
                }
            }

            return result;
        }

        private static boolean matches(int[] charCountP, int[] charCountWindow) {
            for (int i = 0; i < 26; i++) {
                if (charCountP[i] != charCountWindow[i]) {
                    return false;
                }
            }
            return true;
        }

        public static void main(String[] args) {
            String s = "cbaebabacd";
            String p = "abc";
            List<Integer> indices = findAnagrams(s, p);

            System.out.println("Start Indices of Anagrams:");
            System.out.println(indices);
        }
    }
```

Question 7

Given an encoded string, return its decoded string.
The encoding rule is: k[encoded_string], where the encoded_string inside the square brackets is being repeated exactly k times. Note that k is guaranteed to be a positive integer.
You may assume that the input string is always valid; there are no extra white spaces, square brackets are well-formed, etc. Furthermore, you may assume that the original data does not contain any digits and that digits are only for those repeat numbers, k. For example, there will not be input like 3a or 2[4].
The test cases are generated so that the length of the output will never exceed 105.

Example 1:
Input: s = "3[a]2[bc]"
Output: "aaabcbc"

Solution Code:

```
package in.ineuron.pptAssignment08;
import java.util.Stack;
public class DecodeString {

	public static String decodeString(String s) {
		Stack<Integer> countStack = new Stack<>();
		Stack<String> stringStack = new Stack<>();

		String currentString = "";
		int currentCount = 0;

		for (char c : s.toCharArray()) {
			if (Character.isDigit(c)) {
				currentCount = currentCount * 10 + (c - '0');
			} else if (c == '[') {
				countStack.push(currentCount);
				stringStack.push(currentString);
				currentCount = 0;
				currentString = "";
			} else if (c == ']') {
				StringBuilder sb = new StringBuilder(stringStack.pop());
				int count = countStack.pop();
				for (int i = 0; i < count; i++) {
					sb.append(currentString);
				}
				currentString = sb.toString();
			} else {
				currentString += c;
			}
		}
		return currentString;
	}
```

```
        public static void main(String[] args) {
                String s = "3[a]2[bc]";
                String decodedString = decodeString(s);

                System.out.println("Decoded String: " + decodedString);
        }
}
```

Question 8

Given two strings s and goal, return true if you can swap two letters in s so the result is equal to goal, otherwise, return false.
Swapping letters is defined as taking two indices i and j (0-indexed) such that i != j and swapping the characters at s[i] and s[j].
- For example, swapping at indices 0 and 2 in "abcd" results in "cbad".

Example 1:
Input: s = "ab", goal = "ba"
Output: true

Explanation: You can swap s[0] = 'a' and s[1] = 'b' to get "ba", which is equal to goal.

Solution Code:

```
package in.ineuron.pptAssignment08;

public class SwapLetters {

        public static boolean canBeEqual(String s, String goal) {
                if (s.length() != goal.length()) {
                        return false;
                }

                int diffCount = 0;
                int[] charCount = new int[26];

                for (int i = 0; i < s.length(); i++) {
                        charCount[s.charAt(i) - 'a']++;
                        charCount[goal.charAt(i) - 'a']--;
                        if (s.charAt(i) != goal.charAt(i)) {
                                diffCount++;
                        }
                }

                if (diffCount == 0 || diffCount == 2) {
                        for (int count : charCount) {
                                if (count != 0) {
```

```
                                    return false;
                                }
                            }
                            return true;
                        }

                        return false;
                }

                public static void main(String[] args) {
                        String s = "ab";
                        String goal = "ba";
                        boolean result = canBeEqual(s, goal);

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