Assignment Questions 8

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.

code:-

```
class Solution {
    public int helper(String r,String s,int i,int j,int[][] dp){
        if(i==-1 && j==-1) return 0;
        if(i==-1){
            int ascii=0;
            for(int k=0;k<=j;k++){
                ascii += (int)s.charAt(k);
            }
            return ascii;
        }
        if(j==-1){
            int ascii=0;
            for(int k=0;k<=i;k++){
                ascii += (int)r.charAt(k);
            }
            return ascii;
        }
        if(dp[i][j]!=-1) return dp[i][j];
        if(r.charAt(i)==s.charAt(j)){
            dp[i][j]=helper(r.substring(0,i),s.substring(0,j),i-1,j-1,dp);
        }
        else{
dp[i][j]=Math.min((int)s.charAt(j)+helper(r,s.substring(0,j),i,j-1,dp),
(int)r.charAt(i)+helper(r.substring(0,i),s,i-1,j,dp));
        }
        return dp[i][j];
    }
    public int minimumDeleteSum(String s1, String s2) {
        int m = s1.length();
        int n = s2.length();
        int[][] dp = new int[m][n];
        for(int[] d:dp){
            Arrays.fill(d,-1);
        }
        return helper(s1,s2,m-1,n-1,dp);
    }
}
```

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

code:-

```
class Solution {
    boolean ans = false;

    public boolean checkValidString(String s) {
        solve(new StringBuilder(s), 0);
        return ans;
    }

    public void solve(StringBuilder sb, int i) {
        if (i == sb.length()) {
            ans |= valid(sb);
        } else if (sb.charAt(i) == '*') {
            for (char c: "() ".toCharArray()) {
                sb.setCharAt(i, c);
                solve(sb, i+1);
                if (ans) return;
            }
            sb.setCharAt(i, '*');
        } else
            solve(sb, i + 1);
    }

    public boolean valid(StringBuilder sb) {
        int bal = 0;
        for (int i = 0; i < sb.length(); i++) {
            char c = sb.charAt(i);
            if (c == '(') bal++;
            if (c == ')') bal--;
            if (bal < 0) break;
        }
        return bal == 0;
    }
}
```

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".

code:-

```
class Solution {
```

```
    public int minDistance(String word1, String word2) {
        int l1 = word1.length(), l2 = word2.length();
        int[][] dp = new int[l1+1][l2+1];
        // Initialize Table
        for (int i=1; i<=l1; i++) {
            dp[i][0] = i;
        }
        for (int i=1; i<=l2; i++) {
            dp[0][i] = i;
        }
        // Fill Table
        for (int i=1; i<=l1; i++) {
            for (int j=1; j<=l2; j++) {
                if (word1.charAt(i-1) == word2.charAt(j-1)) {
                    dp[i][j] = dp[i-1][j-1];
                } else {
                    dp[i][j] = Math.min(2 + dp[i-1][j-1], Math.min(1 + dp[i][j-1],
1 + dp[i-1][j]));
                }
            }
        }
        return dp[l1][l2];
    }
}
```

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

code:-

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {

    StringBuilder str=new StringBuilder();

    public void convert(TreeNode root){
```

```
    if(root==null){
        return;
    }
    str.append(root.val);
    if(root.left==null&&root.right!=null){
        str.append("()");
    }
    else if(root.left!=null){
        str.append('(');
        convert(root.left);
        str.append(')');
    }
    if(root.right!=null){
        str.append('(');
        convert(root.right);
        str.append(')');
    }
    }
    public String tree2str(TreeNode root) {

    convert(root);

    return str.toString();

    }
}
```

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".

code:-

```
class Solution {
    public int compress(char[] chars) {
        char prevChar = chars[0];
        int count = 1, k = 0 ;
        for(int i = 1 ; i < chars.length ; i++){
            char c = chars[i];
            if(c == prevChar)
                count++;
            else{
```

```
                chars[k++] = prevChar;
                if(count==1){
                    prevChar = c;
                    continue;
                }
                String cs = "";
                cs+=count;
                char[] countArray = cs.toCharArray();
                for(char countChar :  countArray)
                    chars[k++] = countChar;

                prevChar = c;
                count = 1;
            }
        }
        chars[k++] = prevChar;
        if(count == 1)
            return k;

        String cs = "";
        cs+=count;
        char[] countArray = cs.toCharArray();
        for(char countChar :  countArray)
            chars[k++] = countChar;

        return k;
    }
}
```

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".

code:-

```
class Solution {
    public List<Integer> findAnagrams(String s, String p) {
        List<Integer> res = new ArrayList<>();

        if(s.length() < p.length()){
            return res;
        }
        int[] ct1 = new int[26];
        int[] ct2 = new int[26];

        int k = p.length();

        for(char it : p.toCharArray()){
            ct1[it-'a']++;
        }
```

```
        int i = 0;

        while(i < k){
            ct2[s.charAt(i++)-'a']++;
        }

        k = s.length();
        int j = 0;

        if(Arrays.equals(ct1, ct2)){
            res.add(j);
        }

        while(i < k){

            ct2[s.charAt(j++)-'a']--;
            ct2[s.charAt(i++)-'a']++;

            if(Arrays.equals(ct1, ct2)){
                res.add(j);
            }
        }

        return res;
    }
}
```

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"

code:-

```
int i = 0;
public String decodeString(String s) {
    StringBuilder sb = new StringBuilder();
    int count = 0;
    String tmp_string = "";

    while (i < s.length()) {
        char c = s.charAt(i);
        i++;

        if (c == '[') {
```

```
            tmp_string = decodeString(s); // do subproblem
            for (int j = 0; j < count; j++) {
                sb.append(tmp_string);
            }
            count = 0; // reset counter
        } else if (c == ']') { // subproblem complete
            break;
        } else if (Character.isAlphabetic(c)) {
            sb.append(c);
        } else {
            count = count * 10 + c - '0';
        }
    }

    return sb.toString();
}
```

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.

code:-

```
class Solution {
    public boolean buddyStrings(String s, String goal) {
        if (s.length() != goal.length()) return false;

        // if both are string equal
        // then check does s contain only unique character
        // if yes, then after swapping is not possible to get the goal string
        // so return false;
        // but if s contains any duplicate character then it is possible
        if (s.equals(goal)) {
            int[] count = new int[26];
            for (int i = 0; i < s.length(); i++) {
                count[s.charAt(i) - 'a']++;
            }

            for (int i = 0; i < count.length; i++) {
                if (count[i] > 1) return true;
            }

            return false;
        }


        // if both string are not equal
        // then check at what position they differ
        // the number of differ positions should be 2
        // if it is less than 2 or more than 2 then we will not be able to get the goal after swapping
```

```
        // so return false

        // but if it is 2, then check the characters first string first position
and second string second postion
        // and first string second position and second string first position
        else {
            int first = -1, second = -1;
            for (int i = 0; i < s.length(); i++) {
                if (s.charAt(i) != goal.charAt(i)) {
                    if (first == -1) first = i;
                    else if (second == -1) second = i;
                    else return false;
                }
            }

            return second > -1 && s.charAt(first) == goal.charAt(second) &&
s.charAt(second) == goal.charAt(first);
        }

    }
}
```