Assignment 08 Solutions

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.

Ans:

In [1]:
```
def minimumDeleteSum(s1, s2):
    m, n = len(s1), len(s2)
    dp = [[0] * (n+1) for _ in range(m+1)]

    for i in range(1, m+1):
        dp[i][0] = dp[i-1][0] + ord(s1[i-1])
    for j in range(1, n+1):
        dp[0][j] = dp[0][j-1] + ord(s2[j-1])

    for i in range(1, m+1):
        for j in range(1, n+1):
            if s1[i-1] == s2[j-1]:
                dp[i][j] = dp[i-1][j-1]
            else:
                dp[i][j] = min(dp[i-1][j] + ord(s1[i-1]), dp[i][j-1] + ord(s2[j-1]))

    return dp[m][n]
```

In [2]:
```
s1 = "sea"
s2 = "eat"
result = minimumDeleteSum(s1, s2)
print(result)
```

```
231
```

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

Ans:

In [3]:
```
def isValid(s):
    stack = []

    for c in s:
        if c == '(' or c == '*':
            stack.append(c)
        elif c == ')':
            if stack and stack[-1] == '(':
                stack.pop()
```

```
            elif stack and stack[-1] == '*':
                stack.pop()
            else:
                return False

    while stack and stack[-1] == '(':
        stack.pop()

    return len(stack) == 0
```

In [4]:
```
s = "()"
result = isValid(s)
print(result)
```

```
True
```

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".

Ans:

In [5]:
```
def minDistance(word1, word2):
    m, n = len(word1), len(word2)
    dp = [[0] * (n+1) for _ in range(m+1)]

    for i in range(1, m+1):
        dp[i][0] = i
    for j in range(1, n+1):
        dp[0][j] = j

    for i in range(1, m+1):
        for j in range(1, n+1):
            if word1[i-1] == word2[j-1]:
                dp[i][j] = dp[i-1][j-1]
            else:
                dp[i][j] = min(dp[i-1][j] + 1, dp[i][j-1] + 1)

    return dp[m][n]
```

In [6]:
```
word1 = "sea"
word2 = "eat"
result = minDistance(word1, word2)
print(result)
```

```
2
```

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]

Ans:

In [12]:
```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def constructTree(s):
    if not s:
```

```
        return None

    idx = s.find('(')

    if idx == -1:
        return TreeNode(int(s))

    root_val = int(s[:idx])
    root = TreeNode(root_val)

    count = 0
    start = idx

    for i in range(start, len(s)):
        if s[i] == '(':
            count += 1
        elif s[i] == ')':
            count -= 1

        if count == 0 and start == idx:

            root.left = constructTree(s[start + 1:i])
            start = i + 1
        elif count == 0:
            root.right = constructTree(s[start + 1:i])

    return root
```

In [15]:
```
s = "4(2(3)(1))(6(5))"
root = constructTree(s)

def inorderTraversal(root):
    if root is None:
        return []
    return inorderTraversal(root.left) + [root.val] + inorderTraversal(root.right)

result = inorderTraversal(root)
print(result)
```

```
[3, 2, 1, 4, 5, 6]
```

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".

Ans:

In [16]:
```
def compress(chars):
    write = 0
    read = 1
    count = 1

    while read < len(chars):
        if chars[read] == chars[read - 1]:
            count += 1
        else:
            chars[write] = chars[read - 1]
            write += 1
```

```
            if count > 1:
                count_str = str(count)
                for digit in count_str:
                    chars[write] = digit
                    write += 1
            count = 1

        read += 1

    chars[write] = chars[read - 1]
    write += 1
    if count > 1:
        count_str = str(count)
        for digit in count_str:
            chars[write] = digit
            write += 1

    return write
```

In [17]:
```
chars = ["a", "a", "b", "b", "c", "c", "c"]
result = compress(chars)
compressed_chars = chars[:result]
print(result)
print(compressed_chars)
```

```
6
['a', '2', 'b', '2', 'c', '3']
```

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".

Ans:

In [18]:
```
from collections import Counter

def findAnagrams(s, p):
    p_freq = Counter(p)
    window_freq = Counter(s[:len(p)])

    left = 0
    right = len(p) - 1

    result = []

    while right < len(s):
        if window_freq == p_freq:
            result.append(left)

        window_freq[s[left]] -= 1
        if window_freq[s[left]] == 0:
            del window_freq[s[left]]
        left += 1

        right += 1
        if right < len(s):
            window_freq[s[right]] += 1

    return result
```

In [19]:
```
s = "cbaebabacd"
p = "abc"
result = findAnagrams(s, p)
print(result)
```

```
[0, 6]
```

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"

Ans:

In [20]:
```
def decodeString(s):
    stack = []
    current_str = ''
    current_count = 0

    for char in s:
        if char.isdigit():
            current_count = current_count * 10 + int(char)
        elif char == '[':
            stack.append(current_str)
            stack.append(current_count)
            current_str = ''
            current_count = 0
        elif char == ']':
            count = stack.pop()
            prev_str = stack.pop()
            current_str = prev_str + count * current_str
        else:
            current_str += char

    return current_str
```

In [21]:
```
s = "3[a]2[bc]"
result = decodeString(s)
print(result)
```

```
aaabcbc
```

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.

Ans:

In [22]:
```
def buddyStrings(s, goal):
    if len(s) != len(goal):
        return False

    diff_indices = []
    diff_chars = []

    for i in range(len(s)):
        if s[i] != goal[i]:
            diff_indices.append(i)
```

```
            diff_chars.append(s[i])

    if len(diff_indices) != 2:
        return False

    if s[diff_indices[0]] == goal[diff_indices[1]] and s[diff_indices[1]] == goal[diff_indices[0]]:
        return True

    return False
```

In [23]:
```
s = "ab"
goal = "ba"
result = buddyStrings(s, goal)
print(result)
```

```
True
```

In [ ]: