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

Prgm:
```
class Delete_sum{
    public static int minimumDeleteSum(String s1, String s2) {
        int m = s1.length();
        int n = s2.length();
        int[][] dp = new int[m+1][n+1];

        for(int i=1; i<=m; i++){
            dp[i][0] = dp[i-1][0] + s1.charAt(i-1);
        }

        for(int i=1; i<=n; i++){
            dp[0][i] = dp[0][i-1] + s2.charAt(i-1);
        }
        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)
                    );
                }
            }
        }
        return dp[m][n];
    }
    public static void main(String[] args) {
        String s1 = "sea";
        String s2 = "eat";
        System.out.println(minimumDeleteSum(s1,s2));
    }
}
```

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

Prgm:
```
class Valid_String{
    public static boolean checkValidString(String s) {
        int bal = 0;
        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) == '(' || s.charAt(i) == '*') bal++;
            else if (bal-- == 0)
              return false;
        }
        if (bal == 0)
          return true;
        bal = 0;
        for (int i = s.length()-1; i >= 0; i--) {
            if (s.charAt(i) == ')' || s.charAt(i) == '*') bal++;
            else if (bal-- == 0)
              return false;
        }
        return true;
    }
    public static void main(String[] args) {
        String s = "()";
       System.out.println(checkValidString(s));
    }
}
```

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

Prgm:
```
class Minimum_Num{
    public static int minNumber(String word1, String word2) {
        char[] arr1 = word1.toCharArray(), arr2 = word2.toCharArray();
        int len1 = arr1.length, len2 = arr2.length;
        int[][] dp = new int[len1 + 1][len2 + 1];
        dp[0][0] = 0;
        for(int i = 1; i <= len1; i++)
            dp[i][0] = i;
        for(int i = 1; i <= len2; i++)
            dp[0][i] = i;
        for(int i = 1; i <= len1; i++){
            for(int j = 1; j <= len2; j++){
                if(arr1[i - 1] == arr2[j - 1])
                    dp[i][j] = dp[i - 1][j - 1];
                else{
                    dp[i][j] = Math.min(dp[i - 1][j - 1] + 2,
                                        Math.min(dp[i - 1][j] + 1, dp[i][j - 1] + 1));
                }
            }
        }
        return dp[len1][len2];
    }
    public static void main(String[] args) {
         String word1 = "sea";
         String word2 = "eat";
         System.out.println(minNumber(word1,word2));
    }
}
```

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]

Prgm:
```
class Anagrams{
    public static List<Integer> findAnagrams(String s, String p) {
    List<Integer> ans = new ArrayList<>();
    int[] count = new int[128];
    int required = p.length();

    for (final char c : p.toCharArray())
```

```
      ++count[c];

    for (int l = 0, r = 0; r < s.length(); ++r) {
      if (--count[s.charAt(r)] >= 0)
        --required;
      while (required == 0) {
        if (r - l + 1 == p.length())
          ans.add(l);
        if (++count[s.charAt(l++)] > 0)
          ++required;
      }
    }
    return ans;
  }
  public static void main(String[] args) {
    String s = "cbaebabacd";
    String p = "abc";
    System.out.println(findAnagrams(s,p));
  }
}
```

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"

Prgm:

```
class Decode{
    public static String decodeString(String s) {
        String res = "";
        Stack<Integer> countStack = new Stack<>();
        Stack<String> resStack = new Stack<>();
        int idx = 0;
        while (idx < s.length()) {
            if (Character.isDigit(s.charAt(idx))) {
                int count = 0;
                while (Character.isDigit(s.charAt(idx))) {
```

```
                count = 10 * count + (s.charAt(idx) - '0');
                idx++;
            }
            countStack.push(count);
        }
        else if (s.charAt(idx) == '[') {
            resStack.push(res);
            res = "";
            idx++;
        }
        else if (s.charAt(idx) == ']') {
            StringBuilder temp = new StringBuilder (resStack.pop());
            int repeatTimes = countStack.pop();
            for (int i = 0; i < repeatTimes; i++) {
                temp.append(res);
            }
            res = temp.toString();
            idx++;
        }
        else {
            res += s.charAt(idx++);
        }
    }
    return res;
  }
  public static void main(String[] args) {
    String s = "3[a]2[bc]";
    System.out.println(decodeString(s));
  }
}
```

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

Prgm:
```
class solution{
    public static boolean buddyStrings(String s, String goal) {
      int n = s.length();
      if (s.equals(goal)) {
          Set<Character> temp = new HashSet<>();
          for (char c : s.toCharArray()) {
              temp.add(c);
          }
          return temp.size() < goal.length(); // Swapping same characters
      }

      int i = 0;
      int j = n - 1;

      while (i < j && s.charAt(i) == goal.charAt(i)) {
          i++;
      }

      while (j >= 0 && s.charAt(j) == goal.charAt(j)) {
          j--;
      }

      if (i < j) {
          char[] sArr = s.toCharArray();
          char temp = sArr[i];
          sArr[i] = sArr[j];
          sArr[j] = temp;
          s = new String(sArr);
      }

      return s.equals(goal);
    }
   public static void main(String[] args) {
      String s = "ab";
      String goal = "ba";
      System.out.println(buddyStrings(s,goal));
   }
}
```