STRINGS - SET 2 SOLUTIONS

Note: All solutions are written in C++.

Question 1:

- **Problem link**: https://practice.geeksforgeeks.org/problems/parenthesis-checker/0
- **Difficulty level**: Easy

Explanation:

- Declare a character stack S.
- Now traverse the expression string exp.
    1. If the current character is a starting bracket (**'(' or '{' or '['**) then push it to stack.
    2. If the current character is a closing bracket (**')' or '}' or ']'**) then pop from stack and if the popped character is the matching starting bracket then fine else brackets are not balanced.
- After complete traversal, if there is some starting bracket left in stack then "not balanced"

Solution:

```
class Solution {
public:
    bool ispar(string expr) {
        stack<char> s;
        char x;

        for (int i = 0; i < expr.length(); i++) {
            if (expr[i] == '(' || expr[i] == '['|| expr[i] == '{') {
                s.push(expr[i]);
                continue;
            }
```

```
            if (s.empty())
                return false;

            switch (expr[i]) {
                case ')':
                    x = s.top();
                    s.pop();
                    if (x == '{' || x == '[')
                        return false;
                    break;

                case '}':
                    x = s.top();
                    s.pop();
                    if (x == '(' || x == '[')
                        return false;
                    break;

                case ']':
                    x = s.top();
                    s.pop();
                    if (x == '(' || x == '{')
                        return false;
                    break;
            }
        }
        return (s.empty());
    }

};
```

Complexity:

- Time: O(|S|)
- Space: O(|S|)

Question 2:

- **Problem link**: https://practice.geeksforgeeks.org/problems/longest-repeating-subsequence/0
- **Difficulty level**: Easy

Explanation:

The idea is to find the **LCS(str, str) where str is the input string with the restriction that when both the characters are the same, they shouldn't be on the same index in the two strings.**

Solution:

```
class Solution {
Public:
    int LongestRepeatingSubsequence(string str) {
        int n = str.length();
        int dp[n+1][n+1];

        for(int i=0;i<=n;i++) {
            dp[i][0] =0;
            dp[0][i] =0;
        }
        for (int i=1; i<=n; i++) {
            for (int j=1; j<=n; j++) {
                if (str[i-1] == str[j-1] && i != j)
                    dp[i][j] =  1 + dp[i-1][j-1];
                else
                    dp[i][j] = max(dp[i][j-1], dp[i-1][j]);
            }
        }
        return dp[n][n];
    }
};
```

Complexity:

- Time: $O(n^2)$
- Extra space: $O(n^2)$

Question 3:

- **Problem link**: https://practice.geeksforgeeks.org/problems/isomorphic-strings/0
- **Difficulty level**: Easy

Explanation:

Count the number of occurrences of a particular character in both the string using two arrays, while we will compare the two arrays if at any moment with the loop the count of the current character in both strings becomes different we return false, else after the loop ends we return true.

Solution:

```
class solution {
    public:
        bool areIsomorphic(string str1, string str2) {
            int n = str1.length(), m = str2.length();
            if (n != m)
                return false;

            int count[MAX_CHARS] = { 0 };
            int dcount[MAX_CHARS] = { 0 };

            for (int i = 0; i < n; i++) {
                count[str1[i]]++;
                dcount[str2[i]]++;

                if (count[str1[i]] != dcount[str2[i]])
                    return false;
```

```
            }
            return true;

        }
}
```

Complexity:

- Time: O(|str1| + |str2|)
- Space: O(Number of different characters)

Question 4:

- **Problem link**: https://practice.geeksforgeeks.org/problems/next-permutation/0
- **Difficulty level**: Medium

Explanation: start traversing the number from the rightmost side. Keep traversing left until a digit is found that is smaller than the previously traversed number. For eg in 2371 we start traversing from right and stop at 1 because 1 is lesser than the previously traversed number 7. If no such digit exists, then the output is “not possible”. Then of the right side of the digit which is previously obtained find the smallest number that is just greater than that digit. Example in 2371 as above is 7 as it is just greater than 3. Then swap the two digits, in example case swap 3 and 7. Now sort all digits from position next to the digit found first till the end of number. The number obtained then is the final number. Therefore the next lexicographic number is 2713.

Solution:

```
class Solution{
    public:
    vector<int> nextPermutation(int N, vector<int> arr){
        int i,j;
        for(i = N - 2; i >= 0 ; i--)
            if(arr[i] < arr[i+1]) break;

        if (i == -1) {
            reverse(arr.begin(), arr.end());
            return arr;
        }

        for(j = N - 1; j > i; j--)
            if(arr[j] > arr[i]) break;

        swap(arr[i], arr[j]);
        reverse(arr.begin() + i + 1, arr.end());
        return arr;
    }
};
```

Complexity:

- Time: O(N)
- Space: O(1)

Question 5:

- **Problem link**: https://practice.geeksforgeeks.org/problems/count-the-reversals/0
- **Difficulty level**: Medium

Explanation:

Since the expression only contains one type of brackets, the idea is to maintain two variables to keep count of left bracket as well as the right bracket.

If the expression has balanced brackets, then we decrement the left variable else we increment the right variable. Then all we need to return is ceil(left/2) + ceil(right/2).

Solution:

```
class Solution{
  public:
      int countRev (string s) {
          int len = s.length();

          if (len%2)
              return -1;

          stack<char> st;
          for (int i=0; i<len; i++) {
              if (s[i]=='}' && !st.empty()) {
                  if (st.top()=='{')
                      st.pop();
                  else
                      st.push(s[i]);
              }
              else
                  st.push(s[i]);
          }
          int red_len = st.size();

          int n = 0;
          while (!st.empty() && st.top() == '{') {
              st.pop();
              n++;
          }
          return (red_len/2 + n%2);
      }
};
```

Complexity:

- Time: O(|S|)
- Space: O(1)

Question 6:

- **Problem link**: https://leetcode.com/problems/zigzag-conversion/
- **Difficulty level**: Medium

Explanation:

By iterating through the string from left to right, we can easily determine which row in the Zig-Zag pattern that a character belongs to. We can use min(numRows,len(*s*)) lists to represent the non-empty rows of the Zig-Zag pattern. Iterate through *s* from left to right, appending each character to the appropriate row. The appropriate row can be tracked using two variables: the current row and the current direction. The current direction changes only when we moved up to the topmost row or moved down to the bottommost row.

Solution:

```
class solution {
    public:
        string convert(string s, int numRows) {
        if (numRows == 1) return s;

        string ret;
        int n = s.size();
        int cycleLen = 2 * numRows - 2;

        for (int i = 0; i < numRows; i++) {
            for (int j = 0; j + i < n; j += cycleLen) {
```

```
                ret += s[j + i];
                if (i != 0 && i != numRows - 1 && j + cycleLen - i < n)
                    ret += s[j + cycleLen - i];
            }
        }
        return ret;
};
```

Complexity:

- Time: O(n)
- Space: O(n)

Question 7:

- **Problem link**: https://practice.geeksforgeeks.org/problems/excel-sheet5448/1
- **Difficulty level**: Medium

Explanation:

Suppose we have a number n, let's say 28. so corresponding to it we need to print the column name. We need to take the remainder with 26.

If remainder with 26 comes out to be 0 (meaning 26, 52, and so on) then we put 'Z' in the output string and new n becomes n/26 -1 because here we are considering 26 to be 'Z' while in actuality it's 25th with respect to 'A'.

Similarly, if the remainder comes out to be non-zero. (like 1, 2, 3, and so on) then we need to just insert the char accordingly in the string and do n = n/26.

Finally, we reverse the string and print.

Solution:

```
class Solution {
public:
    string ExcelColumn(int N) {
        char str[50];
        int i = 0;

        while (N > 0) {
            int rem = N % 26;

            if (rem == 0) {
                str[i++] = 'Z';
                N = (N / 26) - 1;
            }
            else {
                str[i++] = (rem - 1) + 'A';
                N = N / 26;
            }
        }
        str[i] = '\0';

        reverse(str, str + strlen(str));

        return str;
    }

};
```

Complexity:

- Time: O(log(N))
- Space: O(log(N))

Question 8:

- **Problem link**: https://practice.geeksforgeeks.org/problems/find-all-possible-palindromic-partitions-of-a-string/1
- **Difficulty level**: Hard

Explanation: We start at function allPartitions which uses a 2d vector to store all the partitions of the string. It then calls the allPartUntil function which acts as the recursive function for this problem. The base case is when the start index exceeds the length of the string which makes sense because by then there is nothing else to scan. Then a for loop runs n times which signifies all different possible ending points for the substring. Then we check if that string with the endpoint from the for loop is a palindrome by calling the isPalindrome function. If it returns true then we recursively call allPartUntil function recursively. This continues until all the recursive functions terminate after executing the base case. Thus we have successfully tested all the substrings of a string and checked if they are palindromes.

Solution:

```
void allPartUtil(vector<vector<string> >&allPart, vector<string>
&currPart,
                 int start, int n, string str)
{

    if (start >= n)
    {
        allPart.push_back(currPart);
        return;
    }
```

```
    for (int i=start; i<n; i++)
    {

        if (isPalindrome(str, start, i))
        {
            currPart.push_back(str.substr(start, i-start+1));
            allPartUtil(allPart, currPart, i+1, n, str);
            currPart.pop_back();
        }
    }
}
bool isPalindrome(string str, int low, int high)
{
    while (low < high)
    {
        if (str[low] != str[high])
            return false;
        low++;
        high--;
    }
    return true;
}
void allPartitions(string str)
{
    int n = str.length();
    vector<vector<string> > allPart;
    vector<string> currPart;
    allPartUtil(allPart, currPart, 0, n, str);
    for (int i=0; i< allPart.size(); i++ )
    {
        for (int j=0; j<allPart[i].size(); j++)
            cout << allPart[i][j] << " ";
        cout << "\n";
    }
}
```

Complexity:

- Time: $O(N*2^N)$
- Space: $O(N^2)$

Question 9:

- **Problem link**: https://leetcode.com/problems/tag-validator/
- **Difficulty level**: Hard

Explanation:

If the beginning of the string does not begin with < or doesn't end with >, it can never be a Valid tag.Traverse through the array. If a < is encountered check what the next character is to determine if a TAGNAME is to be expected or ![CDATA[CDATA_CONTENT]] is to be expected. If a TAGNAME is expected , check if the TAGNAME is valid i.e length lies in range [1,9]. Make sure all the rules are being followed while writing the code for this problem.

Solution:

```
bool isValidTAG( string s ){
    if( s.size() < 1 || s.size() > 9 )  return false;
    for( char c : s){ if( !isupper(c) ) return false; }
    return true;
}
```

```
class Solution {
public:
    bool isValid(string code) {
        if( code[0] != '<' || code[code.size()-1] != '>') return
false;
        vector<string> v;
        for( int idx = 0 ; idx < code.size() ; ++idx)
        {
            if( idx > 0 && v.empty() ) return false;
            int close_index = 0;
            if ( code[idx] == '<')
            {
                if ( code[idx+1] == '!')
                {
                    if( v.empty() ){ return false; }
                    if( code.substr( idx, 9 ) != "<![CDATA[" ) return
false;
                    close_index  = code.find( "]]>", idx); if(
close_index == string::npos ) return false;
                    close_index += 2;
                }

                else if ( code[idx+1] == '/')
                {
                    close_index     = code.find( ">", idx); if(
close_index == string::npos ) return false;
                    string TAG_NAME = code.substr( idx+2,
close_index-(idx+2) );
                    if( !isValidTAG(TAG_NAME) ) return false;
                    if(  v.empty() || v.back() != TAG_NAME ) return
false;
                    v.pop_back();
                }

                else
```

```
                {
                    close_index     = code.find( ">", idx); if(
close_index == string::npos ) return false;
                    string TAG_NAME = code.substr( idx+1,
close_index-(idx+1) );
                    if( !isValidTAG(TAG_NAME) ) return false;
                    v.push_back(TAG_NAME);
                }
                idx = close_index;
            }
        }
        return v.empty();
    }
};
```

Complexity:

- Time: O(n)
- Space: O(n)