STRINGS - SET 1 SOLUTIONS

Note: All solutions are written in C++.

Question 1:

- **Problem link**: https://leetcode.com/problems/reverse-string/
- **Difficulty level**: Easy

Explanation:

Traverse the array from the beginning to n/2th index and for a particular index i , swap ith element from the beginning and the ith element from the end i.e (n-i-1) th index from the beginning to obtain the reverse of the string.

Solution:

```
class Solution {
public:
      void reverseString(vector<char>& s) {
      int n=s.size();
      for(int i=0;i<n/2;i++)
      {
            swap(s[i],s[n-i-1]);
      }


      }
};
    }
};
```

Complexity:

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

Question 2:

- **Problem link**: https://practice.geeksforgeeks.org/problems/palindrome-string0817/1
- **Difficulty level**: Easy

Explanation:

Start iterating through the string from both forward and backward directions i.e declare 0 as low index and (length of string)-1 as high index and compare each character with those indexes. Continue iterating till a mismatch is found or the low index becomes greater than the high index. If there's no mismatch, then it's a palindrome else not.

Solution:

```
class Solution {
Public:
    int isPalindrome(char str[]) {
        int l = 0;
        int h = strlen(str) - 1;

        while (h > l) {
            if(str[l++] != str[h--])
                return 0;
            return 1;
        }
    }
};
```

Complexity:

- Time: O(n)
- Extra space: O(1)

Question 3:

- **Problem link**: https://practice.geeksforgeeks.org/problems/min-number-of-flips3210/1
- **Difficulty level**: Easy

Explanation:

We use brute force and try out all possible results. An alternate string has only 2 possibilities, alternate string starting with 0 and alternate string starting with 1. The string which takes minimum number of flips will be the answer.

Solution:

```
char flip(char ch)
{
    return (ch == '0') ? '1' : '0';
}
 int start(string str, char expected)
{
    int Count = 0;
    for (int i = 0; i < str.length(); i++)
    {
            if (str[i] != expected)
            Count++;
        expected = flip(expected);
    }
    return Count;
}

int minimumFlips(string str)
```

```
{
    return min(start(str, '0'),start(str, '1'));
}
```

Complexity:

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

Question 4:

- **Problem link**: https://practice.geeksforgeeks.org/problems/consecutive-elements2306/1
- **Difficulty level**: Easy

Explanation:

The idea is to keep track of two indexes, index of current character in str and index of next distinct character in str. Whenever we see the same character, we only increment the current character index. We see different characters, we increment the index of distinct character.

Solution:

```
class Solution{
    public:
    string removeConsecutiveCharacter(string S)
    {
        int n = S.length();
        int j = 0;

        for (int i=1; i<n+1; i++) {
            if (S[j] != S[i]) {
                j++;
                S[j] = S[i];
            }
        }
        j++;
        S[j] = '\0';
        return S;
    }
};
```

Complexity:

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

Question 5:

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

Explanation:

This Problem is based on Dynamic Programming. If the solution is not understood , it will be understood by you better once the Dynamic Programming resources and questions are out.

If the sum of lengths of A and B don't match with the length of C it is never possible to obtain C by interleaving A and B . If the sum of length of A and B match the length of C, create a MATRIX of dimensions L1+1 x L2+1. Initialise all values to false. The matrix has a value true if C[0..i+j-1] is an interleaving of A[0..i-1] and B[0..j-1].

If both are empty string, they interleave to form an empty string and thus is true(i==0&&j==0).If A is empty and B[j-1] matches C[j-1], assign MATRIX[i][j] as MATRIX[i][j-1] ans similarly when B is empty and A[i-1]==C[i-1] assign MATRIX[i][j] as MATRIX[i-1][j].

Take three characters x, y, z as (i-1)th character of A and (j-1)th character of B and (i + j – 1)th character of C.If x matches with z and y does not match with z then assign MATRIX[i][j] as MATRIX[i-1][j] similarly if x is not equal to z and y is equal to z then assign MATRIX[i][j] as MATRIX[i][j-1].If x is equal to y and y is equal to z then assign MATRIX[i][j] as bitwise OR of MATRIX[i][j-1] and MATRIX[i-1][j].

Return value of MATRIX[L1][L2].

Note: At early stages of DP, it might be tough to understand how the algorithm might work. So try the algorithm with an example on paper which will enhance your understanding on how the algorithm works .

Solution:

```
class Solution{
  public:
     bool isInterleave(string A, string B, string C)
     {

      int L1 = A.size(), L2 = B.size();


      bool MATRIX[L1+1][L2+1];
```

```
	memset(MATRIX, 0, sizeof(MATRIX));
	if ((L1+L2) != C.size())
	return false;

	for (int i=0; i<=L1; ++i)
	{
		for (int j=0; j<=L2; ++j)
		{

			if (i==0 && j==0)
				MATRIX[i][j] = true;

			else if (i==0 && B[j-1]==C[j-1])
				MATRIX[i][j] = MATRIX[i][j-1];


			else if (j==0 && A[i-1]==C[i-1])
				MATRIX[i][j] = MATRIX[i-1][j];


			else if(A[i-1]==C[i+j-1] && B[j-1]!=C[i+j-1])
				MATRIX[i][j] = MATRIX[i-1][j];


			else if (A[i-1]!=C[i+j-1] && B[j-1]==C[i+j-1])
				MATRIX[i][j] = MATRIX[i][j-1];


			else if (A[i-1]==C[i+j-1] && B[j-1]==C[i+j-1])
				MATRIX[i][j]=(MATRIX[i-1][j] ||
MATRIX[i][j-1]) ;
		}
	}

	return MATRIX[L1][L2];
```

```
        }

};
```

Complexity:

- Time: O(mn)
- Space: O(mn)

Question 6:

- **Problem link**: https://practice.geeksforgeeks.org/problems/print-anagrams-together/1
- **Difficulty level**: Medium

Explanation:

Calculate the hash value of each word in such a way that all anagrams have the same hash value. Populate the Hash Table with these hash values. Finally, print those words together with the same hash values. A simple hashing mechanism can be a modulo sum of all characters. With modulo sum, two non-anagram words may have the same hash value. This can be handled by matching individual characters.

We will take HashMap<HashMap, ArrayList>, the inner hashmap will count the frequency of the characters of each string and the outer HashMap will check whether that hashmap is present or not if present then it will add that string to the corresponding list.

Solution:

```
vector<vector<string> > Anagrams(vector<string>& my_list) {
    map<map<char, int>, vector<string>> my_map;

    for(string str : my_list)
    {
        map<char, int> temp_map;
        vector<string> temp_my_list;
        for(int i = 0; i < str.length(); ++i)
        {
            ++temp_map[str[i]];
        }

        auto it = my_map.find(temp_map);
        if (it != my_map.end())
        {
            it->second.push_back(str);
        }
        else
        {
            temp_my_list.push_back(str);
            my_map.insert({ temp_map, temp_my_list });
        }
    }

    vector<vector<string>> result;

    for(auto it = my_map.begin(); it != my_map.end(); ++it) {
        result.push_back(it->second);
    }
    return result;
}
```

Complexity:

- Time: O(N*|S|*log|S|)
- Space: O(N*|S|*log|S|)

Question 7:

- **Problem link**: https://leetcode.com/problems/rotate-string/
- **Difficulty level**: Medium

Explanation: We first check if the goal string and string s are of same size. If not then we straight away return false because even infinite rotations can never add or delete characters to make the string length same. Then we rotate the string using a for loop that at worst case runs the size of the goal string. We stop rotating the string when the string currently is the same as the goal string.

Solution:

```
class Solution {
public:
    bool rotateString(string A, string B) {
        if(A.size()!=B.size()){
            return false;
        }
        if(A==B){
            return true;
        }
        for(int idx=0;idx<A.size();idx++){
            A=A.substr(1)+A[0];
            if(A==B)
                return true; }
        return false; }
};
```

Complexity:

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

Question 8:

- **Problem link**: https://leetcode.com/problems/parsing-a-boolean-expression/
- **Difficulty level**: Hard

Explanation:

Maintain two stacks one for operators**(op)** and one for operands(**st)**.Push the operators to op stack and operands to st stack. When a closing bracket ')' is encountered, calculate the final result using the top most operator in the op stack and all operands in the st stack until a **)** is encountered. Push this result back into the op stack.

Once we have finished traversing the entire expression, return the top of the st stack i.e the answer of the expression.

Solution:

```
class Solution {
public:
      bool parseBoolExpr(string exp) {
      stack<char> op;
      stack<char>st;
      for(int i=0;i<exp.length();i++)
            {
                  if(exp[i]=='(' || exp[i]=='f' || exp[i]=='t')
                  st.push(exp[i]);
                  else if(exp[i]=='&' || exp[i]=='|' || exp[i]=='!')
                  op.push(exp[i]);
                  else if(exp[i]==')')
                  {
                  char oper;
```

```
                bool result;
                if(!op.empty())
                {
                      oper=op.top();
                      op.pop();
                }
                if(oper=='&')
                      result=true;
                else if(oper=='|')
                      result=false;
                while(!st.empty() && st.top()!='(')
                {
                      char top=st.top();
                      st.pop();
                      if(oper=='&')
                            result&=(top=='t');
                      else if(oper=='|')
                            result|=(top=='t');
                      else if(oper=='!')
                            result=!(top=='t');
                }
                if(!st.empty() && st.top()=='(')
                      st.pop();
                st.push(result?'t':'f');
                }
          }

          if(!st.empty())
                return st.top()=='t'?true:false;
          return false;
          }

};
```

Complexity:

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

Question 9:

- **Problem link**: https://practice.geeksforgeeks.org/problems/numbers-with-one-absolute-difference2416/1/?category[]=Strings&category[]=Strings&difficulty[]=2&page=1&query=category[]Stringsdifficulty[]2page1category[]Strings
- **Difficulty level**: Hard

Explanation: We iterate through the number using a while loop and extract each digit by taking its modulus of 10. Then we maintain two integer variables previous and current which hold the previous number and current number used respectively and it checks if absolute value is nit 1. If it is not 1 'false' is returned. The stepnumbers function takes n=0 and N which is the upper limit given as input to program and the stepnumbers function calls Step function N times and if Step function returns false it does not print that number else it prints that number i. Numbers between 0 and 9 are not printed as they do not satisfy absolute difference of 1 condition.

Solution:

```
#include<bits/stdc++.h>
using namespace std;
bool Step(int n)
{
    int previous = -1;
    while (n) //until its not 0
    {
```

```
        int current = n % 10;
        if (previous == -1)
            previous = current;
        else
        {

            if (abs(previous - current) != 1)
                 return false;
        }
        previous = current;
        n /= 10;
    }

    return true;
}
void stepNumbers(int n, int N)
{
    n=0;
    for (int i=n; i<=N; i++)
        if (Step(i))
        {
            if(i>=0 && i<=9 )
              ;
              else
            cout << i << " ";}

}
```

Complexity:

- Time: O(2^number of digits in N)
- Space: O(2^number of digits in N)