DSA ASSIGNMENT-7(STRING)

Solution 1:

```
class Solution {
    public boolean isIsomorphic(String s, String t) {
         int map1[]=new int[200];
        int map2[]=new int[200];

        if(s.length()!=t.length())
            return false;


        for(int i=0;i<s.length();i++)
        {
            if(map1[s.charAt(i)]!=map2[t.charAt(i)])
                return false;

            map1[s.charAt(i)]=i+1;
            map2[t.charAt(i)]=i+1;
        }
        return true;
    }
}
```

Solution 2:

```
class Solution {
  public boolean isStrobogrammatic(String num) {
    Map<Character, Character> map = new HashMap<Character, Character>();
    map.put('6', '9');
    map.put('9', '6');
    map.put('0', '0');
    map.put('1', '1');
    map.put('8', '8');
    int l = 0, r = num.length() - 1;
    while (l <= r) {
      if (!map.containsKey(num.charAt(l))) return false;
      if (map.get(num.charAt(l)) != num.charAt(r))
        return false;
      l++;
      r--;
```

```
    }
    return true;
  }
}
```

Solution 3:

```
class Solution {
    public String addStrings(String num1, String num2) {
          StringBuilder sb = new StringBuilder();

        int i = num1.length() - 1, j = num2.length() - 1;
        int carry = 0;

        while (i >= 0 || j >= 0) {
            int sum = carry;

            if (i >= 0) sum += (num1.charAt(i--) - '0');
            if (j >= 0) sum += (num2.charAt(j--) - '0');

            sb.append(sum % 10);
            carry = sum / 10;
        }

        if (carry != 0) sb.append(carry);
        return sb.reverse().toString();
    }
}
```

Solution 4:

```
class Solution {
    public String reverseWords(String s) {
         int l=0;
        int r=0;
        char a[]=s.toCharArray();
        while(l<s.length()){
            while(r<s.length() && a[r]!=' '){
                r++;
            }
            //reverse part

            //r-1 because r is pointing current at blank space
            //Let's
            //    |
            //    r
            reverse(a,l,r-1);
            l=r+1;
```

```
            r=l;
        }
        return String.valueOf(a);
    }
     public String reverse(char s[],int l,int r){
         while(l<r){
             char temp=s[l];
             s[l]=s[r];
             s[r]=temp;
             l++;
             r--;
         }
         return String.valueOf(s);
    }
}
```

Solution 5:

```
class Solution {
    char temp ;
    public char[] reverse(char []ar  , int start , int end ){

        while(start<end){
            temp = ar[start];
            ar[start] = ar[end] ;
            ar[end] = temp ;
            start++ ; end-- ;
        }
        return ar ;
    }
    public String reverseStr(String s, int k) {
        char ch[] = s.toCharArray() ;
        int len = ch.length ;
        int i = 0 ;
        int j = k-1 ;
        while(j< len )
        {
            ch = reverse(ch , i , j ) ;
            i = i+ 2*k ;
            j = j+ 2*k ;
        }
        if(j>len-1){
            reverse(ch , i , len-1) ;
        }
        return new String(ch) ;
    }
}
```

Solution 6:

```
class Solution {
    public boolean rotateString(String s, String goal) {
        if(s.length() != goal.length()) return false;
        if(s.equals(goal))
            return true;
        s = s+s;
        return s.contains(goal);
    }
}
```

Solution 7:

```
class Solution {
    public boolean backspaceCompare(String s, String t) {
        String str="";
        String p ="";
        String q="";
        p=isback(s,str);
        q=isback(t,str);
        if(p.equals(q))
        {
            return true;
        }
        else{
            return false;
        }


    }
    String isback(String st,String str)
    {
        Stack<Character> stack=new Stack<>();
        for(int i=0;i<st.length();i++)
        {
            char ch=st.charAt(i);
            if(Character.isLetterOrDigit(ch))
            {
                stack.push(ch);


            }
            else{
                if(!stack.isEmpty())
                {
                    stack.pop();
                }


            }
```

```
        }
        while(!stack.empty())
        {
            str=stack.peek()+str;
            stack.pop();
        }
        return str;

    }
}
```

Solution 8:

```
class Solution {
    public boolean checkStraightLine(int[][] coordinates) {
        /*vector*/
        for(int i = 1; i<coordinates.length-1; i++){

            double coe_1 = (coordinates[i][0] - coordinates[i-1][0])!=0
?(double)(coordinates[i][1] - coordinates[i - 1][1]) / (double)(coordinates[i][0]
- coordinates[i-1][0]):Integer.MAX_VALUE;
            double coe_2 = (coordinates[i+1][0] - coordinates[i][0])!=0
?(double)(coordinates[i+1][1] - coordinates[i][1]) / (double)(coordinates[i+1][0]
- coordinates[i][0]):Integer.MAX_VALUE;

            if(coe_1 != coe_2) return false;
        }

        return true;
    }
}
```
