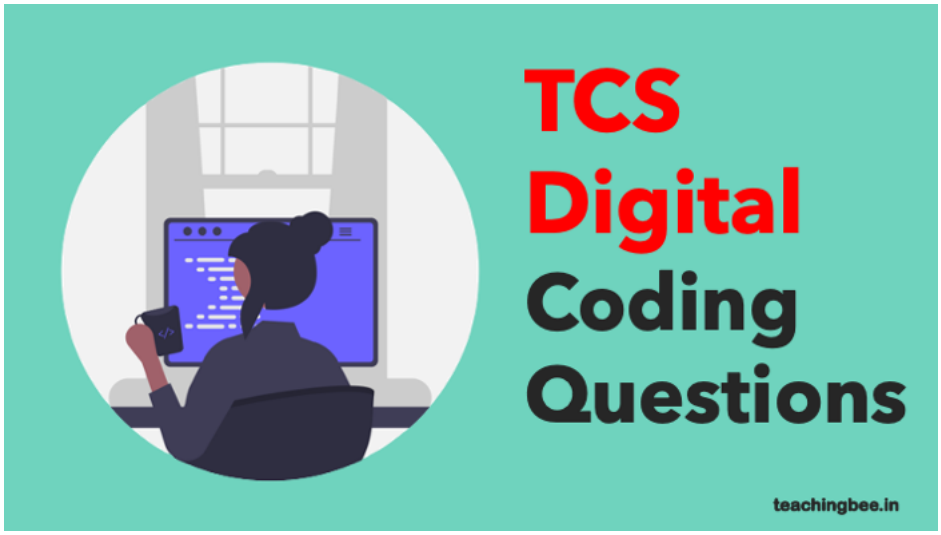

teachingbee(https://teachingbee.in/author/teachingbee/)
February 5, 2022(https://teachingbee.in/2022/02/05/)

TCS Digital Recruitment Drive has an Advance Coding round. TCS Digital Coding Questions round is different from the regular Coding round which is required in TCS. There will be two problems statements, and each will have a the total of two accessible and 3 hidden test cases. The topics which are generally tested in TCS Coding Questions are ***Arrays, Strings, Data Structures, Algorithms Based and Math-Based concepts***.

In this post we will discuss some of the ***TCS Digital Coding Questions*** which are previously asked.

TCS Coding Questions Format

Number of Questions	2
Time Allotted	60 mins
Cut Off	Output is Necessary

TCS Digital Coding Questions Format

TCS Digital Coding Questions

1. Sum of the digits of the given positive integer number N is UNO or not.

Problem Statement

Given a positive integer number N, reduce the number of digits of N by computing the sum of all the digits to get a new number. If this new number exceeds 9, then sum the digits of this new number to get another number and continue this way until a single digit value is obtained as the 'digit sum'. The task here is to find out whether the result of the digit sum done this way is '1' or not. If the result is 1 return UNO else not.

Algorithm

> *Idea is to keep doing sum until sum is not less than or equal to 9*

Implementation

```
// TCS Digital Coding Questions

int sumOfDigits(int n) {
  int sum = 0;

  while (n > 0 || sum > 9) {
    if (n == 0) {
      n = sum;
      sum = 0;
    }
    sum += n % 10;
    n /= 10;
  }
  return sum;
}

int main() {
  int n = 235;
  if (sumOfDigits(n) == 1) {
    cout << "UNO";
  } else {
    cout << "Not UNO";
  }

  return 0;
}
```

Another Approach

1. A number can be represented in either of the form **9n or 9n + k.**
2. If number is of 9n then its sum of digits will also be of the form 9n. Therefore, final single digit sum will always be 9 at last. **For eg 378 ==> 3+7+8 ==>18 ==> 1+8 ==> 9**
3. If number is of 9n + k then final single digit sum will always be k. **For eg 367 (9*40 +7) ==> 16 ==> 7.**
4. So if n%9==0 sum of digits will be 9 else sum of digits will be n%9.

Implementation

```
// TCS Digital Coding Questions

int sumOfDigits(int n) {
  // Base case
  if (n == 0)
    return 0;

  return (n % 9 == 0) ? 9 : (n % 9);
}

int main() {
  int n = 235;
  if (sumOfDigits(n) == 1) {
    cout << "UNO";
  } else {
    cout << "Not UNO";
  }
  return 0;
}
```

2. Find out whether the jet plane returns to the same position from where it took off.

Problem Statement

Most modern aircrafts are equipped with an autopilot system – one of the most useful features in fighter jets. In the beta testing of autopilot mode, one of the inputs is a string of literals containing the directions, which is fed into the flight computer before the take-off. The jet plane is put on an auto-landing mode that enables the plane to land automatically once all the directions in the string are complete. Given the directions in the string “str”, the task here is to find out whether the jet plane returns to the same position from where it took off.

- Each direction in the string changes at an interval of 1 minute(1<=i<= N)

- Each direction in the string changes at an interval of 1 minute(1<= i<= N), where N is the number of directions in the input string.
- The directions are North (N), South(S), West(W) and East(E).

Output:

- “Returned successfully”, if the plane returns to the starting position.
- “Not returned successfully”, if the plane does not return to the starting position.

Example 1: Input: NESW **Output**: Returned successfully

Example 2: Input: NNWESW. **Output**: Not returned successfully

Algorithm:

1. The idea is to keep track of x and y coordinate of jet plane.
2. For **'N'** 'increment y coordinate, for **'E'** increase x coordinate . Similarly for **'S' and 'W'** decrease y and x coordinate respectively.
3. Lastly check if both x and y are 0 if yes jet plane has returned successfully else no.

```
1.  // TCS Digital Coding Questions
2.
3.  public class Main {
4.      static int getJetDirections(String directions) {
5.          int x = 0, y = 0;
6.          for (int i = 0; i < directions.length(); i++) {
7.
8.              if (directions.charAt(i) == 'N') {
9.                  y++;
10.             }
11.             if (directions.charAt(i) == 'E') {
12.                 x++;
13.             }
14.             if (directions.charAt(i) == 'W') {
15.                 x--;
16.             }
17.             if (directions.charAt(i) == 'S') {
18.                 y--;
19.             }
20.         }
21.         if (x == 0 && y == 0)
22.             return 1;
23.         else
24.             return 0;
25.     }
26.     public static void main(String[] args) {
27.         String str = "NESW";
28.         if (getJetDirections(str) == 1)
29.             System.out.println("Returned successfully");
30.         else
```

```
                System.out.println("Not Returned successfully");

        }
    }
```

3. Possible combinations of the coins that can be inserted to get rupees from the kiosk.

A ‘coin vend’ kiosk is installed all the major metro stations The machine allows one to obtain cash of ‘R’ rupees in exchange for coins. The machine operates with the following conditions:

1. Only coins of denomination 1 rupee and 2 rupee can be exchanged.
2. Coins of denomination 2 rupees should not be inserted successively twice.

The task here to find all the possible combinations of the coins that can be inserted to get rupees from the kiosk.

Input: 3 (1 + 1 + 1), (2 + 1), (1 + 2), **Output**: 6

Algorithm:

> The idea is to use recursion. If 1 is inserted then we can insert 1 and 2 both. But if we have inserted 2 then again 2 cannot be inserted so only 1 can be inserted. Thus, are recursion becomes ***coinCombinations(n - 1) + coinCombinations(n - 3);*** *[As 2 comes in combination with 1 so 3 will be subtracted instead of 1*]

Implementation

```
// TCS Digital Coding Questions

class Main {
    static int coinCombinations(int n) {
        if (n == 0) {
            return 1;
        }
        if (n == 1) {
            return 1;
        }
        if (n == 2) {
            return 2;
        }
        return coinCombinations(n - 1) + coinCombinations(n - 3);
    }

```

```
    public static void main(String[] args) {
        int n=5;
        System.out.println(coinCombinations(n));
    }
}
```

4. Find the number of students whose height is less than the height of their adjacent students.

Problem Statement

A physical education teacher asks students to assemble in a straight line for the morning assembly.Given an array of N in which each element represents the height of the student in that position. The task here is to find the number of students whose height is less than the height of their adjacent students.

Input: 35, 15, 45,25,55 **Output**: 2 (35>15<45 and 45>25<55)

Algorithm

> The idea is to traverse array and compare with its left and right adjacent elements.If it is less than both the elements increment the count

Implementation

```
// TCS Digital Coding Questions

public class Main {
    public static void main(String[] args) {
        Integer[] arr = {
            35,
            15,
            45,
            25,
            55
        };
        int count = 0;
        for (int i = 1; i < arr.length - 1; i++) {
            if (arr[i - 1] > arr[i] && arr[i + 1] > arr[i]) {
                count++;
            }
        }
        System.out.println(count);
    }
}
```

5. Find the longest subsequence such that difference between adjacent

5. Find the largest number among them which does not contain 9.

Problem Statement

Rakesh once had trouble finding the numbers in a string. The numbers are distributed across a string.You need to parse only those numbers which do not contain 9. For eg, if the string contains *"hello this is alpha 5051 and 9475 TCS Coding Questions"*. You will extract *5051* and *not 9475*. Print the largest number.

Input: This is alpha 5057 and 97 **Output:** 5057

Algorithm:

1. The idea is to traverse string and check if it is a digit.
2. If digit has been encountered loop until character is found and form the number from these digits.
3. If the number doesn't contains 9 then find the maximum from current max and the number formed.

Implementation

```
1.  // TCS Coding Questions
2.
3.  public class Main {
4.      static long ExtractNumber(String S) {
5.          long ans = -1;
6.          long result = 0;
7.          for (int x = 0; x < S.length(); ++x) {
8.              // check if it is a digit.
9.              if (Character.isDigit(S.charAt(x))) {
10.                 boolean temp = true;
11.                 result = 0;
12.
13.                 // Form number from digits encountered.
14.                 while (x < S.length() && Character.isDigit(S.charAt(x)
15.                     int value = (int) Integer.parseInt(S.valueOf(S.cha
16.
17.                     // check if number contains 9. If yes, we don't ne
18.                     if (value == 9) {
19.                         temp = false;
20.                     }
21.
22.                     result = result * 10 + value;
23.
24.                     ++x;
25.                 }
26.
27.
                    // Consider only non 9 digit numbers
```

```
                    if (temp) {
                        ans = Math.max(result, ans);
                    }
                }
            }

            return ans;
        }

        public static void main(String[] args) {
            String s = "hello this is alpha 5051 and 9475";
            System.out.println(ExtractNumber(s));

        }

    }
```

6. Order Management

Problem Statement:

A store has different categories of products in stock as shown below.
Item Number=[101, 102, 103, 108] **Price**=[42, 50, 500, 40] **Stock** =[10, 20, 15, 16]

User Inputs two values:

- Item number for item which user wish to buy
- Quantity for the item entered above

1. **If quantity is less than stock and item is available display a notification message showing** Output: Total price in float with precision and updated stock for item after after purchase
2. **If the quantity and stocks less than quantity entered by the user while placing order, then** Output: NO STOCK and quantity left
3. **If user enter character as input for item number and quantity or enter item number which is not available** Output: INVALID INPUT

Algorithm

> The idea is to make hashmap with key as item number and value as price and stock. Check if item number is present or not. If present return required result or else return Invalid input.

Implementation

```
// TCS Coding Questions
```

```
                -
2.
3.  import java.util.*;
4.  class Item {
5.      int price;
6.      int stock;
7.      Item(int p, int s) {
8.          this.price = p;
9.          this.stock = s;
10.     }
11. }
12.
13. public class Main {
14.
15.     public static void main(String[] args) {
16.         int[] itemNumber = {
17.             101,
18.             102,
19.             103,
20.             108
21.         };
22.         int[] price = {
23.             42,
24.             50,
25.             500,
26.             40
27.         };
28.         int[] stock = {
29.             10,
30.             20,
31.             15,
32.             16
33.         };
34.         HashMap < Integer, Item > itemStore = new HashMap < > ();
35.
36.         for (int i = 0; i < itemNumber.length; i++) {
37.             Item item = new Item(price[i], stock[i]);
38.             itemStore.put(itemNumber[i], item);
39.         }
40.
41.         int user_input = 101;
42.         int user_quantity = 2;
43.
44.         if (itemStore.containsKey(user_input)) {
45.             Item item = itemStore.get(user_input);
46.             if (item.stock < user_quantity) {
47.                 System.out.println("No Stock");
48.                 System.out.println(item.stock + " Quantity left");
49.             } else {
50.                 System.out.println("Total Price: " + item.price * user
51.                 System.out.println(item.stock - user_quantity + " Quan
52.                 Item itemUpdated = new Item(item.price, item.stock - u
53.                 itemStore.put(user_input, itemUpdated);
54.             }
55.         } else {
56.             System.out.println("Invalid Input");
57.         }
58.
59.     }
60.
61. }
```

7. Split String into the smallest dictionary words

7. Split String into three palindromic substrings

Problem Statement:

Given an input string word, split the string into exactly 3 palindromic substrings. Working from left to right, choose the smallest split for the first substring that still allows the remaining word to be split into 2 palindromes.

Input: *str = "ababbcb"*

Output: *a bab bcb*

Explanation: *Possible splits are {"aba", "b", "bcb"} and {"a", "bab", "bcb"}. Since, {"a", "bab", "bcb"} has splits at earlier indices, it is the required answer.*

Algorithm:

The idea is to run two loops and checking palindrome property for every 3 substrings.

Implementation

```
1.  // TCS Digital Coding Questions
2.
3.  public class Main {
4.
5.      // Check If String is Pallindrome
6.      static boolean checkPallindrome(String str) {
7.
8.          int i = 0, j = str.length() - 1;
9.          while (i < j) {
10.             if (str.charAt(i) != str.charAt(j))
11.                 return false;
12.             i++;
13.             j--;
14.         }
15.
16.         return true;
17.     }
18.
19.     public static void main(String[] args) {
20.         String s1, s2, s3;
21.         String s = "ababbcb";
22.         int n = s.length();
23.         for (int i = 1; i < n - 1; i++) {
24.             // Substring one
25.             s1 = s.substring(0, i);
26.             // If substring is pallindrome
27.             if (checkPallindrome(s1)) {
28.                 for (int j = 1; j < n - i; j++) {
29.                     // Check if Substring two and Substring three is p
30.                     s2 = s.substring(i, i + j);
31.                     s3 = s.substring(i + j, n);
32.                     if (checkPallindrome(s2) && checkPallindrome(s3))
33.                         System.out.println(s1);
```

```
                            System.out.println(s2);
                            System.out.println(s3);
                            return;
                        }
                    }
                }
            }
            System.out.println("Not Possible");
            return;
        }

    }
```

Above solution is not optimal.For Optimal Solution check here (https://leetcode.com/discuss/interview-question/664885/optimum-solution-for-splitting-a-string-into-three-palindromes-with-earliest-cuts).

Conclusion

In this post we discussed tcs coding questions that are previously asked. Hope these tcs digital coding questions will help you to prepare for tcs coding questions rounds.

Got a question or just want to chat? Comment below or drop by our (http://forum.kirupa.com/)forums (https://teachingbee.in/contact/), where a bunch of the friendliest people you'll ever run into will be happy to help you out!

(https://teachingbee.in/how-to-reverse-a-string (https://teachingbee.in/is-python-case-sensitive