TCS NQT 2025 -18th April coding questions with solution.(1st shift)

Learning With Ram

For Python and java code see comment box

Question1: Find Common Height Difference or Detect Invalid Input

Find Common Height Difference or Detect Invalid Input

Given the height of a tree for 4 consecutive weeks, calculate the difference between each week.

If any height is negative, return "Not valid inputs".

If any two weekly differences are the same, return that difference.

If all differences are different, return "None".

APPROACH

Step 1: Check if any height is negative. If so, return "Not valid inputs".
Step 2: Calculate the differences:

diff1 = height2 - height1
diff2 = height3 - height2
diff3 = height4 - height3
Step 3: If any two of the above differences are the same, return the common difference.
Step 4: If all differences are different, return "None".

Input:
Four integers representing the height of the tree in 4 consecutive weeks.

Example 1:

Input: 2, 4, 6, 7
Output: 2

4-2 =2

6-4=2

7-6=1

Explanation: Differences are 2, 2, and 1 → 2 appears twice.

Example 2:

Input: 5, 10, 11, 13
Output: None
Explanation: Differences are 5, 1, and 2 → all different.

Example 3:

Input: -1, 3, 4, 5
Output: Not valid inputs
Explanation: Negative height is invalid.

Code:-

```
#include <iostream>

#include <set>

using namespace std;


string findCommonDifferenceOrInvalid(int h1, int h2, int h3, int h4) {

  // Check for negative inputs

  if (h1 < 0 || h2 < 0 || h3 < 0 || h4 < 0) {

    return "Not valid inputs";

  }


  int diff1 = h2 - h1;

  int diff2 = h3 - h2;

  int diff3 = h4 - h3;


  // Use set to check for uniqueness

  if (diff1 == diff2 || diff1 == diff3) {

    return to_string(diff1);

  } else if (diff2 == diff3) {

    return to_string(diff2);

  } else {

    return "None";

  }

}
```

```
int main() {
    int h1, h2, h3, h4;

    // Example usage:
    // Input format: space-separated integers
    cout << "Enter heights for 4 weeks: ";
    cin >> h1 >> h2 >> h3 >> h4;

    cout << findCommonDifferenceOrInvalid(h1, h2, h3, h4) << endl;

    return 0;
}
```

Q2)Count and Generate All Distinct Permutations of a String

◇ **Problem Statement:**

You are given a string s consisting of lowercase or uppercase letters (the letters may be repeated). Your task is to generate all distinct permutations of the string using all its characters exactly once in each permutation.

If the string contains duplicate characters, identical permutations must be counted only once and not repeated in the output.

Return or print:

All distinct permutations of the string.
The total number of such unique permutations.

◇ **Input: A single string s (1 ≤ s.length ≤ 10), consisting of only English letters.**

◇ **Output:**

A list of all unique permutations (in any order or lexicographical order if specified).
The count of these permutations.

◇ **Example 1: Input:**

```
ABC
```

Output:

```
ABC
ACB
BAC
BCA
CAB
```

```
CBA
Total unique permutations: 6
```

◇ **Example 2: Input:**

```
AAB
```

Output:

```
AAB
ABA
BAA
Total unique permutations: 3
```

Code(approach -1)

```
#include <iostream>

#include <vector>

#include <algorithm>

#include <set>

using namespace std;

void generatePermutations(string& str, int index, set<string>& result) {

   if (index == str.length()) {

      result.insert(str); // Insert permutation into set to avoid duplicates

      return;

   }

   for (int i = index; i < str.length(); i++) {

      swap(str[i], str[index]);

      generatePermutations(str, index + 1, result);

      swap(str[i], str[index]); // Backtrack to previous state

   }

}

int main() {

   int n1, n2, n3;

   cout << "Enter the number of times 'a' appears: ";

   cin >> n1;

   cout << "Enter the number of times 'b' appears: ";
```

```
    cin >> n2;
    cout << "Enter the number of times 'c' appears: ";
    cin >> n3;
    // Construct the string
    string str = string(n1, 'a') + string(n2, 'b') + string(n3, 'c');
    // To store unique permutations
    set<string> result;
    // Generate permutations
    generatePermutations(str, 0, result);
    // Print all unique permutations
    cout << "Unique permutations are: \n";
    for (const auto& perm : result) {
        cout << perm << endl;
    }
    return 0;
}
```

CODE(Approach-2)

```
#include <iostream>
#include <string>
#include <set>
#include <algorithm>

using namespace std;

void printAllDistinctPermutations(string s) {
    set<string> uniquePermutations;

    // Sort the string to ensure next_permutation works correctly
    sort(s.begin(), s.end());
    // Generate all permutations
    do {
        uniquePermutations.insert(s);
    } while (next_permutation(s.begin(), s.end()));

    // Print all unique permutations
    for (const string& perm : uniquePermutations) {
        cout << perm << endl;
    }

    // Print total count
    cout << "Total unique permutations: " << uniquePermutations.size() << endl;
}

int main() {
    string input;
    cout << "Enter the string: ";
    cin >> input;

    printAllDistinctPermutations(input);

    return 0;
}
```