Day 88/180

Recursion: Perfect Sum Problem 

---

1. Subset with sum divisible by m

```
class Solution {
   public:
    bool print(vector<int>& arr, int index, int n, int sum, int m) {
        // Base condition: if we have traversed all elements,
        // check if current sum is divisible by m
        if (index == n) {
            return (sum % m == 0) && (sum > 0);
        }
        // Recurse without including current element
        if (print(arr, index + 1, n, sum, m)) {
            return true;
        }
        // Recurse including current element
        if (print(arr, index + 1, n, sum + arr[index], m)) {
            return true;
        }
        // If none of the above returned true, return false
        return false;
    }

    int DivisibleByM(vector<int>& nums, int m) {
        int n = nums.size();
        if (print(nums, 0, n, 0, m)) {
            return 1;
        } else {
            return 0;
        }
    }
};
```

Code Explanation and Complexity

Recursive print Function:

- Base Condition: When the index reaches n, which means we have considered all elements, the function checks if the current sum is divisible by m. It returns true only if the sum is greater than zero and divisible by m (to rule out the empty set).

- Check without current element: The function recursively calls itself, excluding the current element (arr[index]) from the sum.

- Check with current element: The function then calls itself again, including the current element in the sum. This effectively creates all subsets by choosing whether to include each element or not.

Early Termination: If any recursion path returns true, further processing is stopped, as we only need to determine if at least one valid subset exists.

Conclusion in DivisibleByM: When the print function returns, DivisibleByM interprets the boolean result. If print returned true, the function returns 1; otherwise, it returns 0.

Time Complexity:
The time complexity of this solution is O(2^n), where n is the size of the input array. This is because for each element in the array, we have two choices - either include it in the sum or exclude it. Since there are 2^n possible combinations of including/excluding elements, the time complexity is exponential.

Space Complexity:
The space complexity of the code is O(n), where n is the depth of the recursion stack. Since we only use the stack for recursive calls and do not store any intermediate subsets, the space complexity is linear with respect to the number of elements in the input array.

2. Number of subsequences in a string divisible by n

Recursive Code, it **will give the correct output but will give TLE or Abort signal from abort(3) (SIGABRT)** on GFG at the last because of the time complexity, but don't worry we will optimize this code when we study Dynamic Programming.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
```

```
class Solution {
   public:
    void solve(string str, string output, int index, vector<string>& ans)
{
        // base case
        if (index >= str.length()) {
            if (output.length() > 0) {
                ans.push_back(output);
            }
            return;
        }
        // exclude
        solve(str, output, index + 1, ans);
        // include
        char element = str[index];
        output.push_back(element);
        solve(str, output, index + 1, ans);
    }

    int countDivisibleSubseq(string s, int N) {
        vector<string> ans;
        string output = "";
        int index = 0,c=0;
        solve(s, output, index, ans);

        vector<int> ints;
        transform(ans.begin(), ans.end(), std::back_inserter(ints),
        [&](std::string s) { return stoi(s); });

        for (auto& i : ints) {
            if(i%N==0)
            c++;
        }
        return c;
    }
};
```

Code Explanation and Complexity

1. solve function:
    - This is a recursive helper function to generate all possible subsequences of the input string str.

- It takes the input string str, the current output string output, the current index index, and a vector ans to store the generated subsequences.
- The base case checks if we have traversed all elements of the string. If so, and if the current output is not empty, it adds the output to the ans vector.
- The function then recursively calls itself twice: once without including the current element and once including the current element.

2. countDivisibleSubseq function:
    - This is the main function that initializes the process by calling the solve method with the initial parameters.
    - It converts the generated subsequences from strings to integers using stoi and stores them in the ints vector.
    - It then iterates through the ints vector and counts the number of integers that are divisible by N.
    - The final count is returned.

Time Complexity:
The time complexity is exponential and depends on the number of recursive calls made. In the worst case, it can be O(2^|s|*N), where |s| is the length of the input string.

Space Complexity:
The space complexity is also exponential due to the recursion depth. The auxiliary space is used for the recursive call stack. The space complexity is O(|s|*N).

_____ END _____