Documents/leetcode-master/myLeetCode/C++/Backtracking/hw3problem2.cpp

```cpp
#include <iostream>
#include <vector>

using namespace std;

void findArrays(int count, int sum, int start, int n, int target, vector<int>& arr,
int& cnt) {
    if (count == n && sum == target) {
        for (int i = 0; i < arr.size(); i++) {
            cout << arr[i] << " ";
        }
        cout << endl;
        cnt++;
        return;
    }
    if (count >= n || sum > target) {
        return;
    }
    for (int i = start; i <= target - sum + 1; i++) {
        arr[count] = i;
        findArrays(count + 1, sum + i, i, n, target, arr, cnt);
        arr[count] = 0;
    }
}

int main() {
    int n = 3, target = 6;
    int totalCnt = 0;
    for (int i = 1; i <= n; i++) {
        cout << "Arrays with size " << i << ":" << endl;
        vector<int> arr(i);
        int cnt = 0;
        findArrays(0, 0, 1, i, target, arr, cnt);
        cout << "Total count: " << cnt << endl << endl;
        totalCnt += cnt;
    }
    cout << "Total count for all sizes: " << totalCnt << endl;
    return 0;
}
// output
// Arrays with size 1:
// 6
// Total count: 1

// Arrays with size 2:
// 1 5
// 2 4
// 3 3
// Total count: 3

// Arrays with size 3:
// 1 1 4
// 1 2 3
// 2 2 2
// Total count: 3
// Total count for all sizes: 7




// In this code, n represents the size of the array, and target represents the sum of
the array.
// The parameter count in the function findArrays indicates the number of currently
filled numbers,
// sum indicates the sum of the currently filled numbers, start indicates the starting
value of the
// next number to be filled in, arr indicates the currently filled number sequence,
cnt represents
// the number of digit sequences that meet the condition.




// In the findArrays function, we first judge whether the number of numbers and the
sum of numbers
// that have been filled in meet the requirements. If the requirements are met, we
output the
// current number sequence and add one to the number of number sequences that meet the
conditions;
// if the number that has been filled in If the number is greater than or equal to n,
or the sum
// of the current numbers is greater than the target, exit the recursion. Otherwise,
we iterate
// through the numbers from start to target - sum + 1, fill in one number at a time,
then
// recursively call the findArrays function to fill in the next number, and finally
pop the
// filled number. After the function returns, just output the number of number
sequences that
// meet the conditions.


// The time complexity and space complexity of this code are as follows:
// time complexity:
// The code uses the backtracking method. For each array of length n, it needs to
enumerate
// the numbers from 1 to target, so it needs to enumerate all the numbers from 1 to
target
// in total. Therefore, the time complexity is O(target^n).
// Space complexity:
// The code uses a vector as an intermediate variable to store each array of length n,
// so the space complexity is O(n). Also, O(1) extra space is required to store some
variables.
// Therefore, the total space complexity is O(n).
```