Name : Meet Raut

Batch : S21   Roll No. : 84

Experiment 10 : Sum of Subsets

Aim

To implement the Subset Sum problem.

Theory

Given a set of non-negative integers and a value sum, the task is to check if there is a subset of the given set whose sum is equal to the given sum.

Subset Sum Problem using Recursion:
For the recursive approach, there will be two cases.

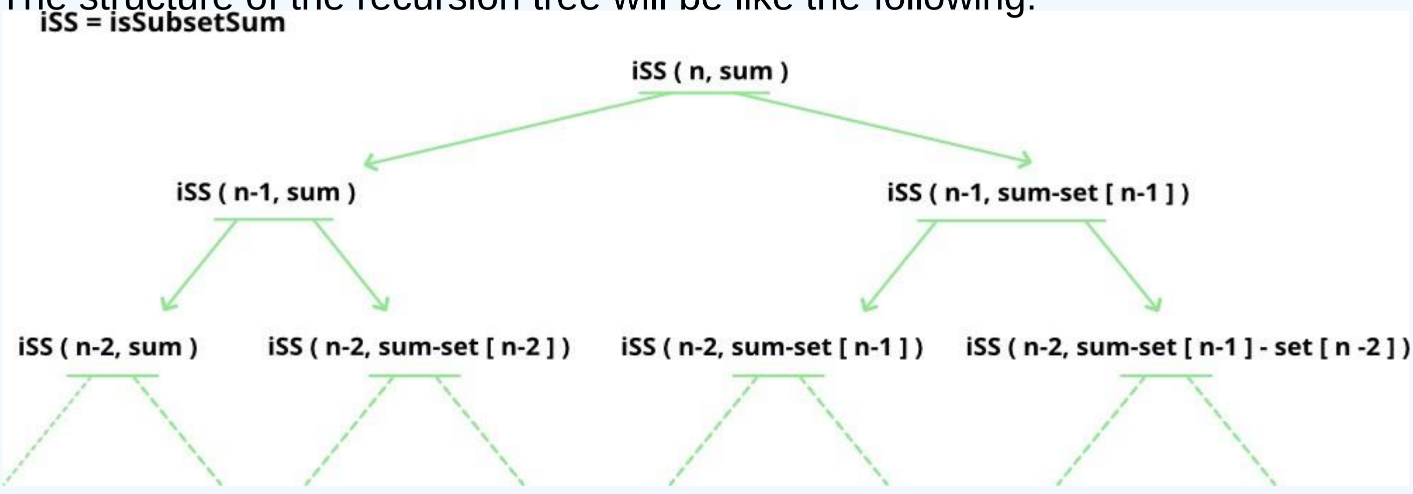
Consider the ‘last’ element to be a part of the subset. Now the new required sum = required sum – value of ‘last’ element.
Don’t include the ‘last’ element in the subset. Then the new required sum = old required sum.
In both cases, the number of available elements decreases by 1.

Mathematically the recurrence relation will look like the following:

isSubsetSum(set, n, sum) = isSubsetSum(set, n-1, sum) | isSubsetSum(set, n-1, sum-set[n-1])

Base Cases:
isSubsetSum(set, n, sum) = false, if sum > 0 and n = 0
isSubsetSum(set, n, sum) = true, if sum = 0

The structure of the recursion tree will be like the following:


iSS = isSubsetSum
iSS (n, sum)
iSS (n-1, sum)
iSS (n-1, sum-set [n-1])
iSS (n-2, sum)
iSS (n-2, sum-set [n-2])
iSS (n-2, sum-set [n-1])
iSS (n-2, sum-set [n-1] - set [n -2])

Follow the below steps to implement the recursion:

- Build a recursive function and pass the index to be considered (here gradually moving from the last end) and the remaining sum amount.
- For each index check the base cases and utilise the above recursive call.
- If the answer is true for any recursion call, then there exists such a subset. Otherwise, no such subset exists.

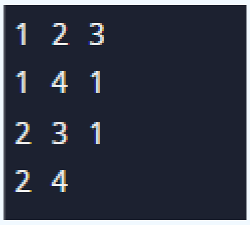
Complexity Analysis:

- Time Complexity: O(2n) The above solution may try all subsets of the given set in the worst case. Therefore the time complexity of the above solution is exponential. The problem is in-fact NP-Complete (There is no known polynomial time solution for this problem).
- Auxiliary Space: O(n) where n is recursion stack space.

Code

```cpp
// Online C++ compiler to run C++ program online
#include <iostream>
#include <vector>
using namespace std;
vector<vector<int>> ans;

void subsetSum(vector<int> &nums, vector<int> &temp, int sum, int i, int k)
{
    if(sum == k)
    {
        ans.push_back(temp);
        return;
    }
    if(sum > k) return;
    if(i >= nums.size()) return;

    temp.push_back(nums[i]);
    subsetSum(nums, temp, sum+nums[i], i+1, k);
    temp.pop_back();
    subsetSum(nums, temp, sum, i+1, k);
}

int main() {
    // Write C++ code here
    vector<int> nums = {1,2,3,4,1};
    vector<int> temp;
     subsetSum(nums, temp, 0, 0, 6);
     for(auto i : ans)
     {
         for(auto j : i)
         {
             cout<<j<<" ";
         }
         cout<<endl;
     }

    return 0;
}
```

Output



```
1 2 3
1 4 1
2 3 1
2 4
```

Conclusion

Hence we have successfully studied and implemented the Subset Sum Problem.