Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

NAME: ALISTAIR SALDANHA
SAPID: 60009200024
BATCH: K1

Experiment 10

(Backtracking)

Aim: Implementation of Sum of subsets.

Theory:

Subset sum problem is the problem of finding a subset such that the sum of elements equals a given number. The backtracking approach generates all permutations in the worst case but in general, performs better than the recursive approach towards subset sum problem.

A subset A of n positive integers and a value **sum** is given, find whether or not there exists any subset of the given set, the sum of whose elements is equal to the given value of sum.

Example:

Problem statement : We are given 'n' distinct positive integers and a target_sum. We have to find the combinations of these numbers which exactly sum up to the target_sum value.

Let n=5 and given positive integers are my_list={1,2,3,4,5}. Let the given target_sum=6.The subsets that produce the total of 6 are {1,5},{2,4} which is the output of our program. This is because 1+5=2+4=6.

Example 1:

Input: set[] = {4, 16, 5, 23, 12}, sum = 9

Output = true

Subset {4, 5} has the sum equal to 9.

Example 2:

Input: set[] = {2, 3, 5, 6, 8, 10}, sum = 10

Output = true

There are three possible subsets that have the sum equal to 10.

Subset1: {5, 2, 3}

Subset2: {2, 8}

Subset3: {10}

Department of Computer Science and Engineering (Data Science)

Algorithm:

1. Start with an empty set
2. Add the next element from the list to the set
3. If the subset is having sum M, then stop with that subset as solution.
4. If the subset is not feasible or if we have reached the end of the set, then backtrack through the subset until we find the most suitable value.
5. If the subset is feasible (sum of seubset < M) then go to step 2.
6. If we have visited all the elements without finding a suitable subset and if no backtracking is possible then stop without solution.

Pseudocode:

```
Algorithm SUB_SET_PROBLEM(i, sum, W, remSum)
// Description : Solve sub of subset problem using backtracking
// Input :
W: Number for which subset is to be computed
i: Item index
sum : Sum of integers selected so far
remSum : Size of remaining problem i.e. (W – sum)

// Output : Solution tuple X

if FEASIBLE_SUB_SET(i) == 1 then
  if (sum == W) then
    print X[1…i]
  end
else
  X[i + 1] ← 1
  SUB_SET_PROBLEM(i + 1, sum + w[i] + 1, W, remSum – w[i] + 1 )
  X[i + 1] ← 0   // Exclude the ith item
  SUB_SET_PROBLEM(i + 1, sum, W, remSum – w[i] + 1 )
end

function FEASIBLE_SUB_SET(i)
  if (sum + remSum ≥ W) AND (sum == W) or (sum + w[i] + 1 ≤ W) then
    return 0
  end
return 1
```

Time Complexity: O(sum*n), where sum is the ‘target sum’ and ‘n’ is the size of array.

Department of Computer Science and Engineering (Data Science)

Lab Assignment to Complete:

1. Given an array of integers and a sum, the task is to have all subsets of given array, total nodes generated with sum equal to the given sum. Input: set[] = {4, 16, 5, 23, 12}, sum = 9.
2. Given an array of integers and a sum, the task is to have all subsets of given array, total nodes generated with sum equal to the given sum. Input: set[] = { 2, 3, 5, 6, 8, 10}, sum = 10.

Department of Computer Science and Engineering (Data Science)

Code:

```
#include <stdio.h>
#define MAX 10
int reqd_sum = 10;
int set[MAX] = {4, 16, 5, 23, 12};
int len = 6;
// int set[MAX] = { 2, 3, 5, 6, 8, 10},
// int len = 5;
void SOS(int level, int sum, int results[MAX]) {
    int i;
    if (sum==reqd_sum)  {
        for (i=0; i<len; i++) {
            if (results[i]==1) {
                printf("%d ", set[i]);
            }
        }
        printf("\n");
    } else if (sum>reqd_sum || level>len) {
        return;
    } else {
        //select
        results[level] = 1;
        SOS(level+1, sum+set[level], results);

        //dont select
        results[level] = 0;
        SOS(level+1, sum, results);
    }
}
int main()
{
    int resultset[MAX] = {0};
    printf("Numbers forming the required sum are: \n");
    SOS(0, 0, resultset);
    return 0;
}
```

Output:

1.
```
Numbers forming the required sum are:
4 5
```

2.
```
Numbers forming the required sum are:
2 3 5
2 8
10
```