Bansilal Ramnath Agarwal Charitable Trust's
Vishwakarma Institute of Information Technology

Department of
Artificial Intelligence and Data Science

Student Name: **Sahil Dilip Sonawane**		
Class: **TY**	Division: **C**	Roll No.: **373057**
Semester: **5**	Academic Year: **2023-24**	
Subject Name & Code: **Design and Analysis of Algorithm: ADUA31202**		
Title of Assignment: **To find a subset of a given set S = {s1 ,s2 ,.....,s n } of n positive integers whose sum is equal to a given positive integer d.**		
Date of Performance:	Date of Submission:	

Assignment No. 7

Aim:

To find a subset of a given set S = {s1 ,s2 ,.....,s n } of n positive integers whose sum is equal to a given positive integer d.

Problem Statement:

Find a subset of a given set S = {s1 ,s2 ,.....,s n } of n positive integers whose sum is equal to a given positive integer d. For example, if S= {1, 2, 5, 6, 8} and d = 9 there are two solutions {1,2,6} and {1,8}. A suitable message is to be displayed if the given Problem instance doesn't have a solution.

Theory:

It is one of the most important problems in complexity theory. The problem is given an A set of integers a1, a2,…., an upto n integers. The question arises that is there a non-empty subset such that the sum of the subset is given as M integer? For example, the set is given as [5, 2, 1, 3, 9], and the sum of the subset is 9; the answer is YES as the sum of the subset [5, 3, 1] is equal to 9. This is an NP-complete problem again. It is the special case of knapsack.

This problem is mainly an extension of Subset Sum Problem. Here we not only need to find if there is a subset with the given sum but also need to print all subsets with a given sum.

We build a 2D array dp[][] such that dp[i][j] stores true if sum j is possible with array elements from 0 to i.

After filling dp[][], we recursively traverse it from dp[n-1][sum]. For the cell being traversed, we store the path before reaching it and consider two possibilities for the element.

1. Element is included in the current path.

2. Element is not included in the current path.

Whenever the sum becomes 0, we stop the recursive calls and print the current path.

Algorithm:

subsetSum(set, subset, n, subSize, total, node, sum)

Input − The given set and subset, size of set and subset, a total of the subset, number of elements in the subset and the given sum.

Output − All possible subsets whose sum is the same as the given sum.

```
Begin
if total = sum, then
display the subset
//go for finding next subset
subsetSum(set, subset, , subSize-1, total-set[node], node+1, sum)
return
else
for all element i in the set, do
subset[subSize] := set[i]
subSetSum(set, subset, n, subSize+1, total+set[i], i+1, sum)
done
End
```

Pseudo Code:

```
def subset_sum(arr, res, sum)

if sum ==0

return true

if sum < 0

return false

if len(arr) == 0 and sum!= 0

return false

arr.pop(0);

if len(arr) > 0

res.append(arr[0])

select = subset_sum(arr, sum-arr[0], res)

reject = subset_sum(arr, res, sum)

return reject or sum
```

Comparative Study of Complexity Analysis:

Here are the time complexities for all ways to solve the Subset Sum Problem:

Sr.No.	Method Name	Time Complexity	Space Complexity
1.	Recursion	exponential	depends upon size of array
2.	Dynamic Programming	O(sum*size)	O(sum*size)
3.	Memoization Technique	O(sum*size)	O(sum*size) + O(size)

Software Requirements:

Text Editor: VSCode, Online GDB Compiler
Environment: Python

Program Code:

```python
def display(v):
    for i in range(len(v)):
        print(v[i], end=" ")
    print()

def print_subsets_rec(arr, i, target_sum, p):
    if i == 0 and target_sum != 0 and dp[0][target_sum]:
        p.append(arr[i])
        if arr[i] == target_sum:
            display(p)
        return

    if i == 0 and target_sum == 0:
        display(p)
        return

    if dp[i - 1][target_sum]:
        b = p.copy()
        print_subsets_rec(arr, i - 1, target_sum, b)

    if target_sum >= arr[i] and dp[i - 1][target_sum - arr[i]]:
        p.append(arr[i])
        print_subsets_rec(arr, i - 1, target_sum - arr[i], p)

def print_all_subsets(arr, n, target_sum):
    if n == 0 or target_sum < 0:
        return

    global dp
    dp = [[False] * (target_sum + 1) for _ in range(n)]
    dp[0][0] = True

    if arr[0] <= target_sum:
        dp[0][arr[0]] = True

    for i in range(1, n):
        for j in range(target_sum + 1):
            dp[i][j] = dp[i - 1][j] or (j >= arr[i] and dp[i - 1][j - arr[i]])

    if not dp[n - 1][target_sum]:
        print(f"There are no subsets with sum {target_sum}")
        return

    p = []
    print_subsets_rec(arr, n - 1, target_sum, p)

if __name__ == "__main__":
    arr = [1, 2, 5, 6, 8]
    n = len(arr)
    target_sum = 9
    print_all_subsets(arr, n, target_sum)
```

Output:

```
6 2 1
8 1
```

Conclusion:

In this assignment we found a subset of a given set S = {s1 ,s2 ,.....,s n } of n positive integers whose sum is equal to a given positive integer d. We implemented the algorithm for the example,
S= {1, 2, 5, 6, 8} and d = 9. We found two solutions for it, {1,2,6} and {1,8}
