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

Department of

Artificial Intelligence and Data Science

Name: Siddhesh Dilip Khairnar		
Class: TY	Division: B	Roll No: 372028
Semester: V	Academic Year: 2023-24	
Subject Name & Code: Design and Analysis of Algorithm: ADUA31202		
Title of Assignment: 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.		

ASSIGNMENT NO. 7

Page No.
Date

DAA Assignment no7

Name : Siddhesh Dilip Khairnar

PRN no : 22110398

Roll no : 372028

Aim: find a subset of a given set S = {S_1, S_2, ⋯ S_n} of n positive integer 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 soln {1, 2, 6} and {1, 8}. A suitable message is to be display if the given problem instance doesn't have a solution.

Learning objective :→

1) Understanding the efficiency of the algorithm, including its ~~tem~~ time complexity in term of the size of the input set and the target sum.
2) Learning how to reconstruct the actual soln from the 'dp' array after solving the problem.

Theory :→

1. Initialization : we create a 2D array 'dp' with dimension (n+1)x(d+1) where 'n' is the no. of element is set 's'. Initially, all values in 'dp' are set to 'False'.
2. Base ~~case~~ case : we set the values in the first column of 'dp' to 'True' because ~~#~~ for any subset, if the sum required ('d') is 0, it's always possible to achieve that by selecting an empty subset.
3. Dynamic Programmings : we iterate over each element in the set 's' and for each element 's[i]', we consider two possibilities :→

- Exclude the element 's[i]', which means the soln remain the same as the previous row ('dp[i-1][j]')

- Include the element 's[i]', which means we check if it's possible to achieve the remaining sum 'j-s[i]' using the previous row ('dp[i-1][j-s[i]]').

4. Reconstruction : After filling in the 'dP' array, we can reconstruct the subset by tracing back the 'dP' array. If 'dP[n][d]' is '~~True~~ True', it means there is a subset that sum to 'd'. We can then backtrack to find the element that make up this sum.
5. Output : If there is a valid subset, the function return that subset. If no sol^n exists ('dp[n][d]' is 'False'), it return a message indicating that there is no sol^n.

This algorithm efficiently solves the subset sum problem by using dynamic programming, ensuring that it runs in polynomial time with respect to the size of the input set ~~S~~ 's' and the target sum 'd'.

Pseudo Code: →

```
def subset-sum (arr, res, sum)

if sum == 0
return true
if sum < 0
return False
if len(arr) == 0 & 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.
```

Conclusion: → In this assignment we found a subset of a given set S = {S1, S2 ... Sn} of ~~a~~ n positive integer whose sum is equal to a given positive integer d. We implemented the algorithm for the example; S = {1, 2, 5, 6, 8} & d = 9. We found two sol^n for it {1,2,6} & {1,8}. A suitable message was also displayed if the given problem ~~is~~ instance ~~did~~ didn't have a sol^n.

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.

Background Information:

The problem at hand is a fundamental challenge in complexity theory. Given a set of integers *a1, a2,..., an* up to *n* integers, the question is whether there exists a non-empty subset whose elements sum up to a given integer *M*. For instance, consider the set [5,2,1,3,9] with a desired subset sum of 9; the answer is YES as the subset [5,3,1] sums up to 9. This problem is known to be NP-complete and is a specialized case derived from the knapsack problem. It's an extension of the Subset Sum Problem in which the task isn't solely to ascertain the existence of a subset with a specified sum but also to enumerate and print all such subsets.

To tackle this, a 2D array *dp[i][j]* is constructed, where *dp[i][j]* stores true if the sum *j* is achievable using array elements from 0 to *i*. Once this array is populated, a recursive traversal is conducted starting from *dp[n−1] [sum]*. During traversal, the path leading to the current cell is recorded, considering two possibilities for each element:

1. Including the current element in the ongoing path.
2. Excluding the current element from the ongoing path.

When the sum becomes 0, the recursive calls are halted, and the current path is printed.

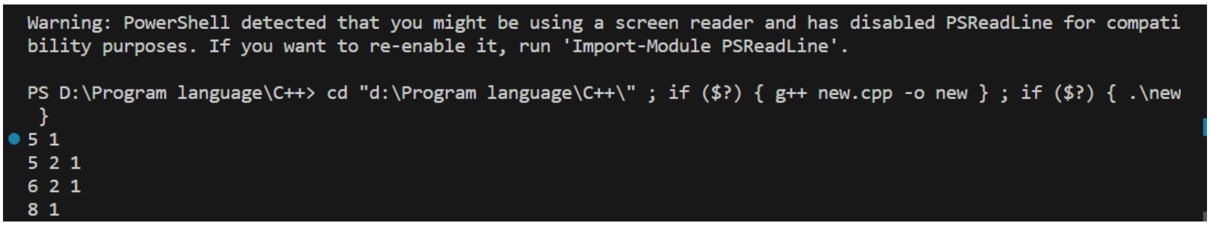
Software Requirements:

Text Editor: VSCode, Online GDB Compiler
Environment: GCC C++

Program Code:

```
#include <iostream>
#include <vector>
using namespace std;
bool **dp;

void display(const vector<int> &v)
{
    for (int i = 0; i < v.size(); ++i)
        cout << v[i] << " ";
    cout << endl;
}

void printSubsetsRec(int arr[], int i, int sum, vector<int> &p)
{
    if (i == 0 && sum != 0 && dp[0][sum])
    {
        p.push_back(arr[i]);
        display(p);
        return;
    }

    if (i == 0 && sum == 0)
    {
        display(p);
        return;
    }

    if (dp[i - 1][sum])
    {
        vector<int> b = p;
        printSubsetsRec(arr, i - 1, sum, b);
    }

    if (sum >= arr[i] && dp[i - 1][sum - arr[i]])
    {
        p.push_back(arr[i]);
        printSubsetsRec(arr, i - 1, sum - arr[i], p);
    }
}

void printAllSubsets(int arr[], int n, int sum)
{
    if (n == 0 || sum < 0)
        return;

    dp = new bool *[n];
    for (int i = 0; i < n; ++i)
    {
        dp[i] = new bool[sum + 1];
        dp[i][0] = true;
    }

    if (arr[0] <= sum)
        dp[0][arr[0]] = true;

    for (int i = 1; i < n; ++i)
        for (int j = 0; j < sum + 1; ++j)
            dp[i][j] = (arr[i] <= j) ? dp[i - 1][j] || dp[i - 1][j - arr[i]] : dp[i - 1][j];

    if (dp[n - 1][sum] == false)
    {
        cout << "There are no subsets with sum " << sum << endl;
        return;
    }

    vector<int> p;
    printSubsetsRec(arr, n - 1, sum, p);
}

int main()
{
    int arr[] = {1, 2, 5, 6, 8};
    int n = sizeof(arr) / sizeof(arr[0]);
    int sum = 9;
    printAllSubsets(arr, n, sum);

    // Free allocated memory
    for (int i = 0; i < n; ++i)
        delete[] dp[i];
    delete[] dp;

    return 0;
}
```

Output:



```
Warning: PowerShell detected that you might be using a screen reader and has disabled PSReadLine for compati
bility purposes. If you want to re-enable it, run 'Import-Module PSReadLine'.

PS D:\Program language\C++> cd "d:\Program language\C++\" ; if ($?) { g++ new.cpp -o new } ; if ($?) { .\new
 }
5 1
5 2 1
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}. A suitable message was also displayed if the given Problem instance didn’t have a solution.