Experiment - 6

Student Name: Ronak Jain **UID:** 23BCS10225
Branch: BE-CSE **Section/Group:** KRG-2B
Semester: 5th **Date of Performance:** 10/10/25
Subject Name: Design and Analysis of Algorithms
Subject Code: 23CSH-301

1. **Aim:** Develop a program and analyze complexity to implement subset-sum problem using Dynamic Programming.
2. **Objective:** Objective is to implement subset-sum problem using Dynamic programming.
3. **Input/Apparatus Used:** Standard sum value is taken in order to find subsets of that sum using dynamic programming.

4. **Procedure/Algorithm:**

Procedure:

So we will create a 2D array of size (arr.size() + 1) * (target + 1) of type boolean. The state DP[i][j] will be true if there exists a subset of elements from A[0….i] with sum value = „j“. The approach for the problem is:

if (A[i-1] > j) DP[i][j]

= DP[i-1][j]

else

DP[i][j] = DP[i-1][j] OR DP[i-1][j-A[i-1]]

1. This means that if current element has value greater than „current sum value“ we will copy the answer for previous cases

2. And if the current sum value is greater than the „ith“ element we will see if any of previous states have already experienced the sum=“j“ OR any previous states experienced a value „j

– A[i]‟ which will solve our purpose.

The below simulation will clarify the above approach: set[]={3, 4, 5, 2}

target=6

0 1 2 3 4 5 6

Dynamic Programming – Subset Sum Problem

Given a set of positive integers, and a value *sum S*, find out if there exist a subset in array whose sum is equal to given *sum S.*

Example:

int[] A = { 3, 2, 7, 1}, S = 6

Output: True, subset is (3, 2, 1}

Recursive Approach:

For every element in the array has two options, either we will include that element in subset or we don't include it.

§ So if we take example as int[] A = { 3, 2, 7, 1}, S = 6

§ If we consider another int array with the same size as A.

§ If we include the element in subset we will put 1 in that particular index else put 0.

§ So we need to make every possible subsets and check if any of the subset makes the sum as S.

Time Complexity: O(2n).

Approach: Dynamic Programming

Base Cases:

§ If no elements in the set then we can't make any subset except for 0.

§ If sum needed is 0 then by returning the empty subset we can make the subset with sum 0.

Given – Set = **arrA[]**, Size = **n**, sum = **S**

§ Now for every element in he set we have 2 options, either we include it or exclude it.

§ for any ith element-

§ If include it => S = S-arrA[i], n=n-1

§ If exclude it => S, n=n-1.

Recursive Equation:

Base Cases:

SubsetSum (arrA, n, S) = false, if sum > 0 and n == 0 SubsetSum (arrA, n, S)= true, if sum == 0 (return empty set)

Rest Cases

SubsetSum (arrA, n, S) = SubsetSum (arrA, n-1, S) || SubsetSum (arrA, n-1, S-arrA[n-1])

How to track the elements.

§ Start from the bottom-right corner and backtrack and check from the True is coming.

§ If value in the cell above if false that means current cell has become true after including the current element. So include the current element and check for the sum = sum – current elements.

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

5. Code and Output:

```
#include <bits/stdc++.h>
using namespace std;

// Recursive approach
bool subsetSumRecursive(vector<int>& arr, int n, int sum) {
    if (sum == 0) return true;
    if (n == 0 && sum != 0) return false;
    if (arr[n - 1] > sum) return subsetSumRecursive(arr, n - 1, sum);
    return subsetSumRecursive(arr, n - 1, sum) || subsetSumRecursive(arr, n - 1, sum - arr[n - 1]);
}

// Dynamic Programming approach (Bottom-Up)
bool subsetSumDP(vector<int>& arr, int sum, vector<vector<bool>>& dp) {
    int n = arr.size();
    for (int i = 0; i <= n; i++)
        dp[i][0] = true;
    for (int j = 1; j <= sum; j++)
        dp[0][j] = false;

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= sum; j++) {
            if (arr[i - 1] > j)
                dp[i][j] = dp[i - 1][j];
            else
```

```
                dp[i][j] = dp[i - 1][j] || dp[i - 1][j - arr[i - 1]];
            }
        }
        return dp[n][sum];
}

// Track the subset elements from DP table
vector<int> getSubsetElements(vector<int>& arr, vector<vector<bool>>& dp, int sum) {
    vector<int> subset;
    int n = arr.size();
    int i = n, j = sum;
    while (i > 0 && j > 0) {
        if (dp[i][j] && !dp[i - 1][j]) {
            subset.push_back(arr[i - 1]);
            j -= arr[i - 1];
        }
        i--;
    }
    reverse(subset.begin(), subset.end());
    return subset;
}

int main() {
    vector<int> arr = {3, 2, 7, 1};
    int sum = 6;
```

```
    int n = arr.size();

    cout << "Subset Sum Problem using Dynamic Programming\n";
    cout << "Set = { ";
    for (int x : arr) cout << x << " ";
    cout << "}, Target Sum = " << sum << "\n\n";

    // Recursive method
    bool recResult = subsetSumRecursive(arr, n, sum);
    cout << "Recursive Approach: " << (recResult ? "Subset Exists" : "Subset Does Not Exist") << "\n";
    cout << "Time Complexity: O(2^n)\n\n";

    // Dynamic Programming method
    vector<vector<bool>> dp(n + 1, vector<bool>(sum + 1, false));
    bool dpResult = subsetSumDP(arr, sum, dp);

    cout << "Dynamic Programming Approach: " << (dpResult ? "Subset Exists" : "Subset Does Not Exist") << "\n";
    cout << "Time Complexity: O(n * sum)\n";
    cout << "Space Complexity: O(n * sum)\n\n";

    if (dpResult) {
        vector<int> subset = getSubsetElements(arr, dp, sum);
        cout << "Subset with sum " << sum << " is: { ";
        for (int x : subset) cout << x << " ";
```

```
        cout << "}\n";
    } else {
        cout << "No subset found with the given sum.\n";
    }

    return 0;
}
```