PART B

|  |  |
| --- | --- |
| Roll No. B032 | Name: Naman Garg |
| Class : Btech CS B | Batch : B2 |
| Date of Experiment: 22-09-2020 | Date of Submission |
| Grade : | Time of Submission: |
| Date of Grading: |  |

**B.1 Software Code written by student:**

# Name: Naman Garg

# Roll No: B032

# Aim of Experiment: Implementation of Backtracking Algorithm Design. Write a program to find solution for Sum of subs Problem.

# k: index of the weight being considered

# r: sum of remaining weights

# s: current sum

def SUM\_OF\_SUBS(s, k, r):

    # Generating the left child

    arr[k] = 1

    if s+w[k] == m:

        # Subset is found

        listOfSubsets = []

        for i in range(n):

            if arr[i] == 1:

                listOfSubsets.append(w[i])

        subs.append(listOfSubsets)

    elif s+w[k]+w[k+1] <= m:

        # Bounding function

        SUM\_OF\_SUBS(s+w[k], k+1, r-w[k])

    # Generate right child

    arr[k] = 0

    if (s+r-w[k] >= m) and (s+w[k+1] <= m):

        # Bounding function

        SUM\_OF\_SUBS(s, k+1, r-w[k])

# Driver program to test above function

n = int(input('Enter number of weights (n): '))

w = list(map(int, input('Enter the weights (wi): ').split()))

m = int(input('Enter the required sum (m): '))

w.sort()    # Sorting just in case user did not enter the values in ascending order

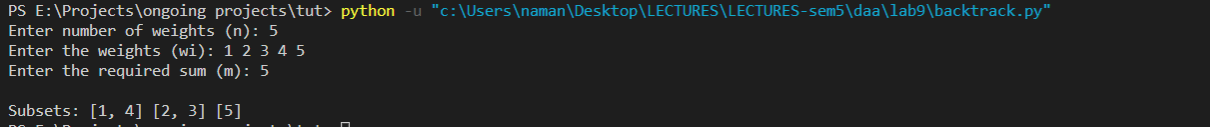
arr = [0]\*n    # List to keep track of the numbers selected from the subset

subs = []

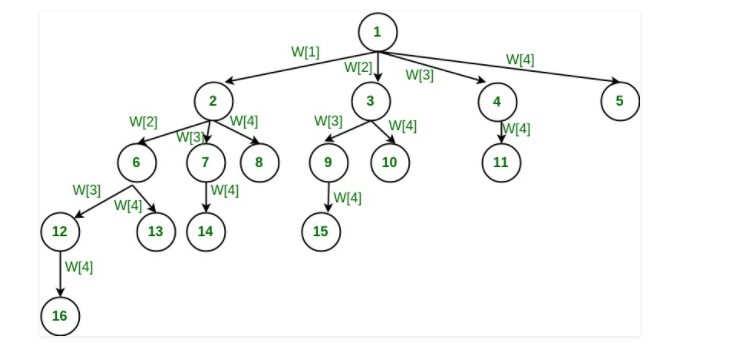
SUM\_OF\_SUBS(0, 0, sum(w))

print('\nsubsets--->>>', \*subs)

**B.2 Input and Output:**

**B.3 Observations and learning:**

Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree).



Since the concept of recursion generates a tree.

Thus in the sum of subsets problem we use the backtracking technique

**B.4 Conclusion:**

Thus we solved the sum of subsets problem using backtracking.

**B.5 Question of Curiosity**

Q.1 Identify & discuss the real life applications Sum of Subsets Problem.

### **Computer passwords**

A computer needs to verify a user's identity before allowing him or her access to an account. The simplest system would have the machine keep a copy of the password in an internal file and compare it with what the user types. A drawback is that anyone who sees the internal file could later impersonate the user.

I believe this alternative is implemented on some systems: the computer generates a large number (say 500) of $a_i$. They are stored in the internal file. A password is a subset of $\{1,\dots,500\}$. (in practice, there is a program to convert a normal sequence-of-symbols password to such a subset.) Instead of having the password for the user, the computer keeps the total associated with the appropriate subset. When the user types in the subset, the computer tests whether the total is correct. It does not keep a record of the subset. Thus, impersonation is possible only if somebody can reconstruct the subset knowing the $a_i$ and the total.

### **Message verification**

A sender (S) wants to send messages to a receiver (R). Keeping the message secret is not important. However, R wants to be sure that the message he is receiving is not from an imposter and has not been tampered with. $S$ and $R$ agree on a set of $a_i$ (say 500) and a set of totals $T_j$ (say 200). These numbers may be publicly known, but only $S$ knows which subsets of the $a_i$ correspond to which $T_j$. The message sent by $S$ is a subset of size 100 of $\{1,\dots,200\}$. He does this by sending 100 subsets of the $a_i$ corresponding to the message he wants to send.

Q.2 Compare different techniques of algorithm design - Divide & Conquer, Greedy, Dynamic Programming and Backtracking.

Backtracking Algorithm: Backtracking Algorithm tries each possibility until they find the right one. It is a depth-first search of the set of possible solution. During the search, if an alternative doesn't work, then backtrack to the choice point, the place which presented different alternatives, and tries the next alternative.

Divide and Conquer Approach: It is a top-down approach. The algorithms which follow the divide & conquer techniques involve three steps:

Divide the original problem into a set of subproblems.

Solve every subproblem individually, recursively.

Combine the solution of the subproblems (top level) into a solution of the whole original problem.

Greedy Technique: Greedy method is used to solve the optimization problem. An optimization problem is one in which we are given a set of input values, which are required either to be maximized or minimized (known as objective), i.e. some constraints or conditions.

Greedy Algorithm always makes the choice (greedy criteria) looks best at the moment, to optimize a given objective.

The greedy algorithm doesn't always guarantee the optimal solution however it generally produces a solution that is very close in value to the optimal.

Q.3 List different problems that can be solved by backtracking technique of algorithm design..

* [The Knight's tour problem](https://www.geeksforgeeks.org/the-knights-tour-problem-backtracking-1/?ref=rp)
* [Sudoku](https://www.geeksforgeeks.org/sudoku-backtracking-7/?ref=rp)
* [m Coloring Problem](https://www.geeksforgeeks.org/m-coloring-problem-backtracking-5/?ref=rp)
* [Hamiltonian Cycle](https://www.geeksforgeeks.org/hamiltonian-cycle-backtracking-6/?ref=rp)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*