**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Computer Engineering Department**

Program: B.Tech / MBA Tech Sem V

**Course: Design and Analysis of Algorithms**

**Faculty:** Radhika Chapaneri

LAB Manual

PART A

(PART A : TO BE REFFERED BY STUDENTS)

**Experiment No.09**

**A.1 Aim:**  **Implementation of Backtracking Algorithm Design**.

**Task 1:**

**Let w ={5, 7, 10, 12, 15, 18 , 20} and m =35. Find all possible subsets of w that sum to m. Draw the portion of the state space tree that is generated.**

**Task 2:**

**Describe the bounding function for Sum of subset in your own words and explain how backtracking is beneficial as compared to brute force.**

**Task 3:**

**Implement a program to find solution for Sum of Subsets Problem.**

**A.2 Prerequisite:**

1. Concepts of Backtracking Technique of algorithm design.

2. Knowledge of Binary Tree Handling.

**A.3 Outcome:**

**After successful completion of this experiment students will be able to**

1. Design & implement a solution using Backtracking Technique.
2. Identify different problems that can be solved by using Backtracking Technique.
3. Identify applications of Backtracking Technique.

**A.4 Theory:**

**BACKTRACKING**

**GENERAL METHOD**

* Problems searching for a set of solutions or which require an optimal solution can be solved using the backtracking method.
* To apply the backtrack method, the solution must be expressible as an n-tuple(x1,…,xn), where the xi are chosen from some finite set si
* The solution vector must satisfy the criterion function P(x1 , ….. , xn).
* The brute force approach would be to form all of these n-tuples and evaluate each one with P, saving the optimum.
* The backtracking algorithm has the ability to yield the same answer with far fewer than m-trials.
* In backtracking, the solution is built one component at a time.
* Modified criterion functions Pi (x1...xn) called bounding functions are used to test whether the partial vector (x1,x2,......,xi) can lead to an optimal solution.
* If (x1,...xi) is not leading to a solution, mi+1,....,mn possible test vectors may be ignored.

**Sum of Subsets**

Subset-Sum Problem is to find a subset of a given set S= {s1, s2… sn} of n positive integers whose sum is equal to a given positive integer d. It is assumed that the set’s elements are sorted in increasing order. The state-space tree can then be constructed as a binary tree and applying backtracking algorithm, the solutions could be obtained. Some instances of the problem may have no solutions

Problem: Given *n* positive integers *w*1,*... wn* and a positive integer S. Find all subsets of *w*1,*... wn* that sum to S.

Example:   
n=3, S=6, and w1=2, w2=4, w3=6

Solutions:  
 {2,4} and {6}

**A.5 Procedure/Algorithm:**

**Algorithm** SumOfSub(s, k, r)

//Find all subsets of w[1…n] that sum to m. The values of x[j], 1<= j < k, have already

//been determined. s=Σk-1 w[j]\*x[j] and r =Σn w[j]. The w[j]’s are in ascending order.

j=1 j=k

{

x[k] ← 1 //generate left child

if (s+w[k] = m)

write (x[1...n]) //subset found

else if ( s + w[k]+w[k+1] <= m)

SumOfSub( s + w[k], k+1, r-w[k])

//Generate right child

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

{

x[k] ← 0

SumOfSub( s, k+1, r-w[k] )

}

}

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

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical.***

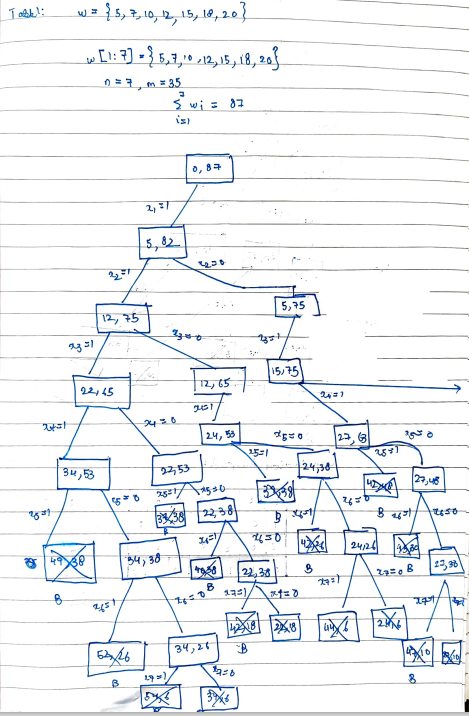
***The soft copy must be uploaded on the Portal.)***

|  |  |
| --- | --- |
| Program: B.Tech CE | Sem: 5 |
| Roll No. 85 | Name: Arya Kapoor |
| Division: B | Batch : B3 |
| Date of Experiment: 11-10-2021 | Date of Submission: 16-10-2021 |
| Grade : |  |

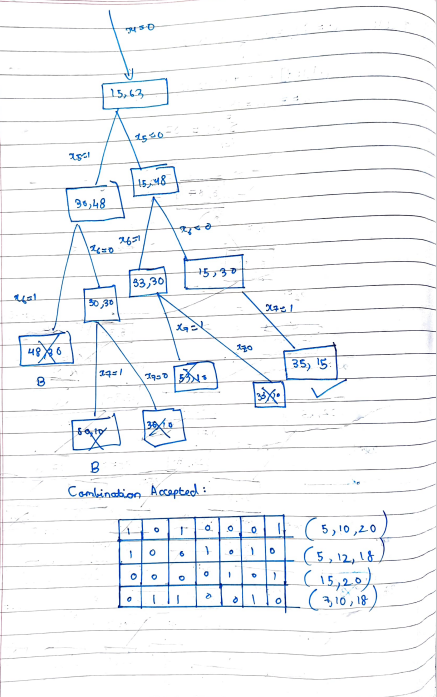
**B.1 Software Code written by student:**

***(Paste your code completed during the 2 hours of practical in the lab here)***

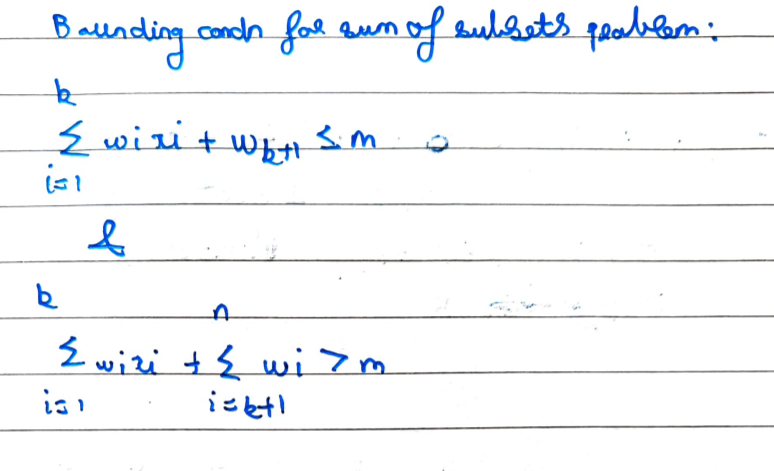
**Task 1**

****

**Task 2**

****

**Task 2**

****

**Task 3**

#Defining recursive function

no\_of\_subsets=0

def subset\_sum(arr,sub\_sum,starting\_index,target\_sum,):

    global no\_of\_subsets

    #When the subset sum is equal to proposed sum

    if target\_sum==sub\_sum:

        no\_of\_subsets+=1 #Incrementing the no of subsets sattisfying condition

        if starting\_index < len(arr):

            subset\_sum(arr,sub\_sum-arr[starting\_index-1],starting\_index,target\_sum)

    else:

        #This portion constructs the subset until target sum is achieved

        for i in range(starting\_index,len(arr)): #Here we ensure that all the elements in the list are checked

            el=arr[i]

            subset\_sum(arr,sub\_sum + el,i+1,target\_sum)

n=int(input("Enter the number of elements"))

arr=[]

for i in range(n):

    print("Enter the element:",(i+1))

    num=int(input())

    arr.append(num)

target\_sum=int(input("Enter the target sum:"))

subset\_sum(arr, 0, 0 , target\_sum)

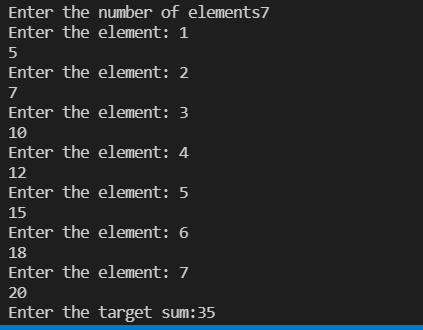
print("No of subsets:",no\_of\_subsets)

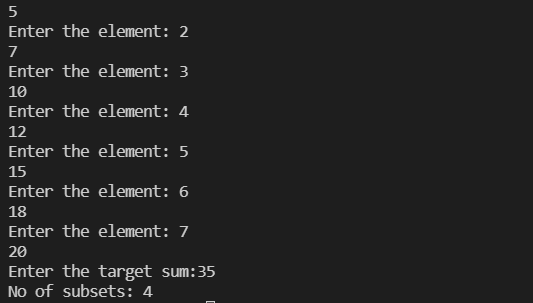
**B.2 Input and Output:**

***(Paste your commented program input and output in following format, If there is error then paste***

***the specific error in the output part. In case of error with due permission of the faculty extension***

***can be given to submit the error free code with output in due course of time.)***

****

****

**B.3 Observations and learning:**

***(Students are expected to comment on the output obtained with clear observations and learning for each task/ sub part assigned)***

In this experiment we learnt how to implement backtracking

**B.4 Conclusion:**

*(****Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)***

After this experiment we will be able to implement this technique in real life projects.

**B.5 Question of Curiosity**

***(To be answered by student based on the practical performed and learning/observations)***

Q.1 Compare different techniques of algorithm design - Divide & Conquer, Greedy, Dynamic

Programming and Backtracking.

f you want the detailed differences and the algorithms that fit into these school of thoughts, please read CLRS.

For a quick conceptual difference read on..

**Divide-and-Conquer:**

**Strategy:**Break a small problem into smaller sub-problems. Smaller sub-problems will most likely be a scaled down version of the original problem. Solve the smaller problems. Combine the smaller solutions to make the complete solution!

**Example:** When you want to eat a big fat chocolate. You break it into smaller pieces. And have it one at a time.

**Greedy:**

**Strategy:**A solution is made up of a number of steps. At each step do what is best for you at that time. (Note: It may or may not be the best choice going forward into the game... Like a local maxima)

**Example:** When you are playing Cricket. At each ball you would want to Hit a six to maximise your score!

**Dynamic Programming :**

**Strategy:**You can't just choose a local maxima always. So you would need an exhaustive search of the solution space. Instead of recomputing each step multiple times. You would calculate smaller solutions, save them in a table and look them up when recall is computationally cheaper than re-computation.

**Example:** Chess!.. You can't just choose the best current move. You need to think of future possibilities and scenarios.

Q.2 List different problems that can be solved by backtracking technique of algorithm design.

* **N-Queens problem**
* **Sum of Subset problem**
* **Seating arrangement problem**
* **Graph coloring problem**

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