SUM OF SUBSETS

⇒ Variant of knapsack problem.
⇒ Extension of " "
⇒ This problem extends knapsack problem to that of checking whether it is possible to find subsets of items whose sum of weights equals w.
⇒ Given n positive items with weights wi { w1, w2, ... wn } and a positive integer w.

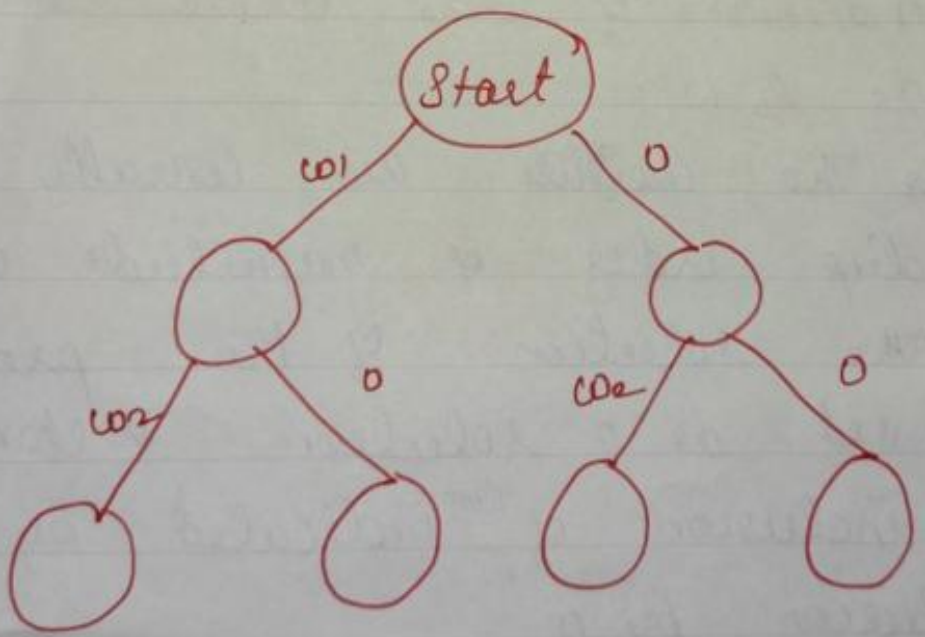
↳ The problem is about finding all the combinations of items whose sum of weights amounts to w.

⇒ The weights are usually in ascending order of magnitude and unique.
⇒ The solution of the problem is expressed as a solution vector X, where the inclusion of rth item indicated by 1 and exclusion by 0.

⇒ State space tree is a binary tree such that every node at level i has 2 branches and at level i+1, where 1 branch ('0') indicates that the item is not included and another branch ('1') indicates the inclusion of the item.

⇒ Cumulative weight of all the items taken is indicated by the variable weight.

∴ Weight indicates the sum of all the items upto the level i.



STATE SPACE TREE

Bounding function

The subsets may include the item x or exclude the item x.

Let the weights be in sorted order.

* At level i, let the smallest remaining weight be w_{i+1}
* and sum of all the weights that include level i be weight, then the non-promising condition is weight + $w_{i+1} > W$.

↪ Implies that addition of an item at level $i+1$ adds to the weight, which crosses W.

⇒ The 2nd node is non-promising if weight + total $< W$.

⇒ Total = sum of all remaining weights

Algorithm

1. Let the weight be the sum of accumulated weights of all items
2. If the weight of all items accumulated equals W, then print the solution. Otherwise go to step 3.

) end

3. Try the following steps for every Branch of the state-space tree
   i) Add the item to the next level and update the weight
   ii) Exclude the item and update the weight
   iii) Goto step2.
4. END

Algorithm sum-of subsets (i, weight, total, W)
// I/p : Item i, weight of items
// Total is available, W - integer weight for which one wishes to find the subsets of
// Items must be equal to W.
// X — Vector whose values are 0 or 1 which
// indicates the inclusion and exclusion of items
// O/p : Items whose sum equals W.
Begin
    if promising_sum_of_subsets(i)
        if (weight == W) then
            print Items x[1...i]
    endif

```
else
    x[i+1] = 1      //Include the item
    Sum-of-subsets (i+1, weight + w_{i+1}, total-
                                    w_{i+1}, w)
    x[i+1] = 0      //Exclude the item.
    Sum-of-subsets (i+1, weight, total - w_{i+1},
                                    w)
  endif
end.
```

⇒ Bounding functions can be used to write promising() function

```
Algorithm promising_sum-of-subsets (i)
// I/p : Item i
// O/p : Status about the feasibility of including
        item as part of the subset
Begin
    flag = true
    if ((weight + total ≥ w) and (weight == w)
                or (weight + w_{i+1} ≤ w) then
    flag = false
    endif
    return flag.
end
```

COMPLEXITY ANALYSIS

⇒ Generated state space tree is binary tree and hence it generates 2 children nodes at every stage.

∴ Number of nodes generated $= \frac{1 + 2 + 2^2 + \cdots + 2^n}{2 - 1}$

$= 2^{n+1} - 1$

∴ $T(n) = O(2^n)$

Eg.

Assume $w_i = [5, 10, 15, 20, 25]$ and $W = 30$. Solve the sum of subset problem using backtracking approach.

To obtain $W = 30$, the different combinations are $[5, 10, 15]$ $[10, 20]$ $[25, 5]$

Solution vector x for $[5, 10, 15]$ is given as $x = [1, 1, 1, 0, 0]$

x for $[10, 20]$ $x = [0, 1, 0, 1, 0]$

x for [25, 5]

$x = [1, 0, 0, 0, 1]$

⇒ Inclusion of item is 1

⇒ Exclusion of " " 0.

State space tree.