DESIGN AND ANALYSIS OF ALGORITHMS

Session -28

0/1 Knapsack Problem

- The problem is similar to the zero-one (0/1) knapsack optimization problem is dynamic programming algorithm.

- We are given ‘n’ positive weights Wi and ’n’ positive profits Pi, and a positive number ‘m’ that is the knapsack capacity, the is problem calls for choosing a subset of the weights such that,

$$\max \sum_{1 \le i \le n} p_i x_i \quad \text{subject to} \quad \sum_{1 \le i \le n} w_i x_i \le m$$

$$0 \le x_i \le 1, \qquad 1 \le i \le n$$

- The Solution space is the same as that for the sum of subset’s problem.

- Bounding functions are needed to help kill some live nodes without expanding them. A good bounding function for this problem is obtained by using an upper bound on the value of the best feasible solution obtainable by expanding the given live node.

Solution :

- After assigning the profit and weights ,we have to take the first object weights and check if the first weight is less than or equal to the capacity, if so then we include that object (i.e.) the unit is 1.(i.e.) K□ 1.

- Then We are going to the next object, if the object weight is exceeded that object does not fit. So unit of that object is ‘0’.(i.e.) K=0.

- Then We are going to the bounding function ,this function determines an upper bound on the best solution obtainable at level K+1.

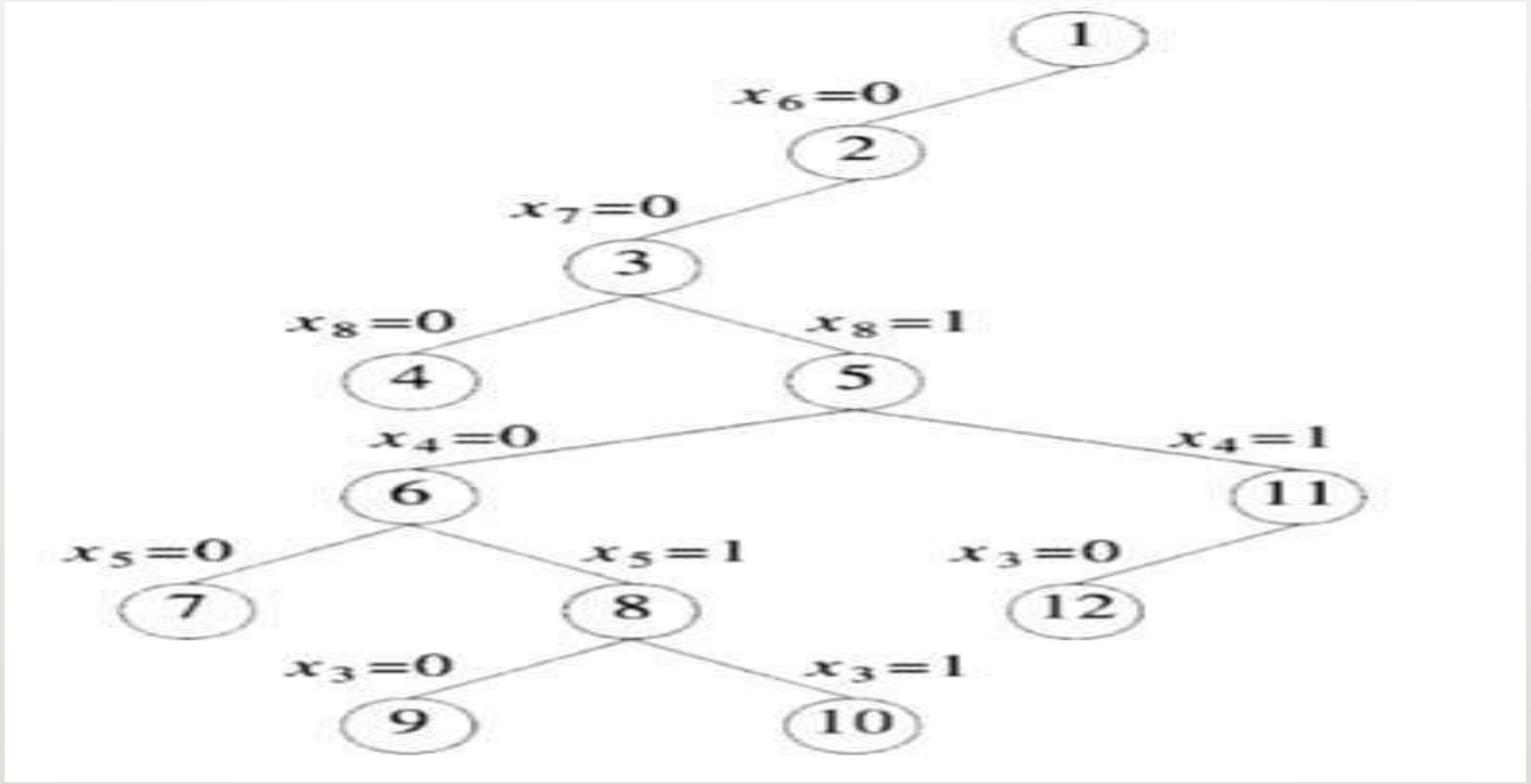
- Repeat the process until we reach the optimal solution.
- Example:

  P={11,21,31,33,43,53,55,65}

  W={1,11,21,23,33,43,45,55}

  m=110 and n=8.
- The greedy solution corresponding to the root node is x={1,1,1,1,1,21/45,0,0} and the value is 164.88.

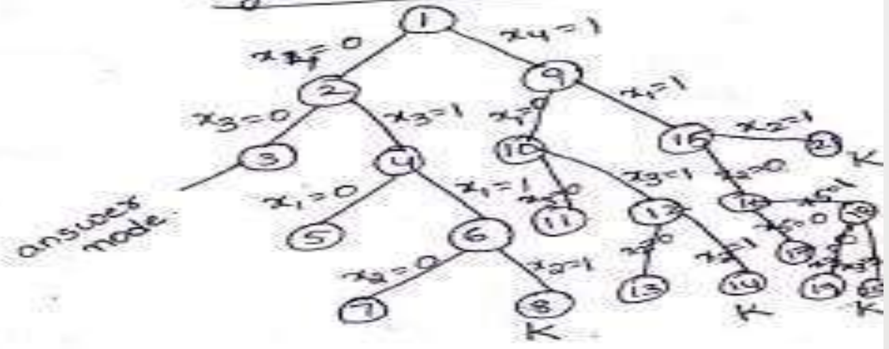
Part of dynamic state space tree generated for above problem is:



Apply Backtracking to solve the Knapsack problem instance when

- $n = 5$, $m = 12$, $P = (10, 15, 6, 8, 4)$
  $W = (4, 6, 3, 4, 2)$
- Arrange objects according to decreasing order of P/w ratio
  ∴ order is 2, 1, 4, 3, 5

dynamic state Space Tree



Node 1
$x = \{1, 1, 2/3, 0, 0\}$
profit = 29

Node 2
$x = \{1, 1, 2/3, 0, 0\}$
profit = 29

Node 3
$x = \{1, 1, 0, 0, 1\}$
profit = 29

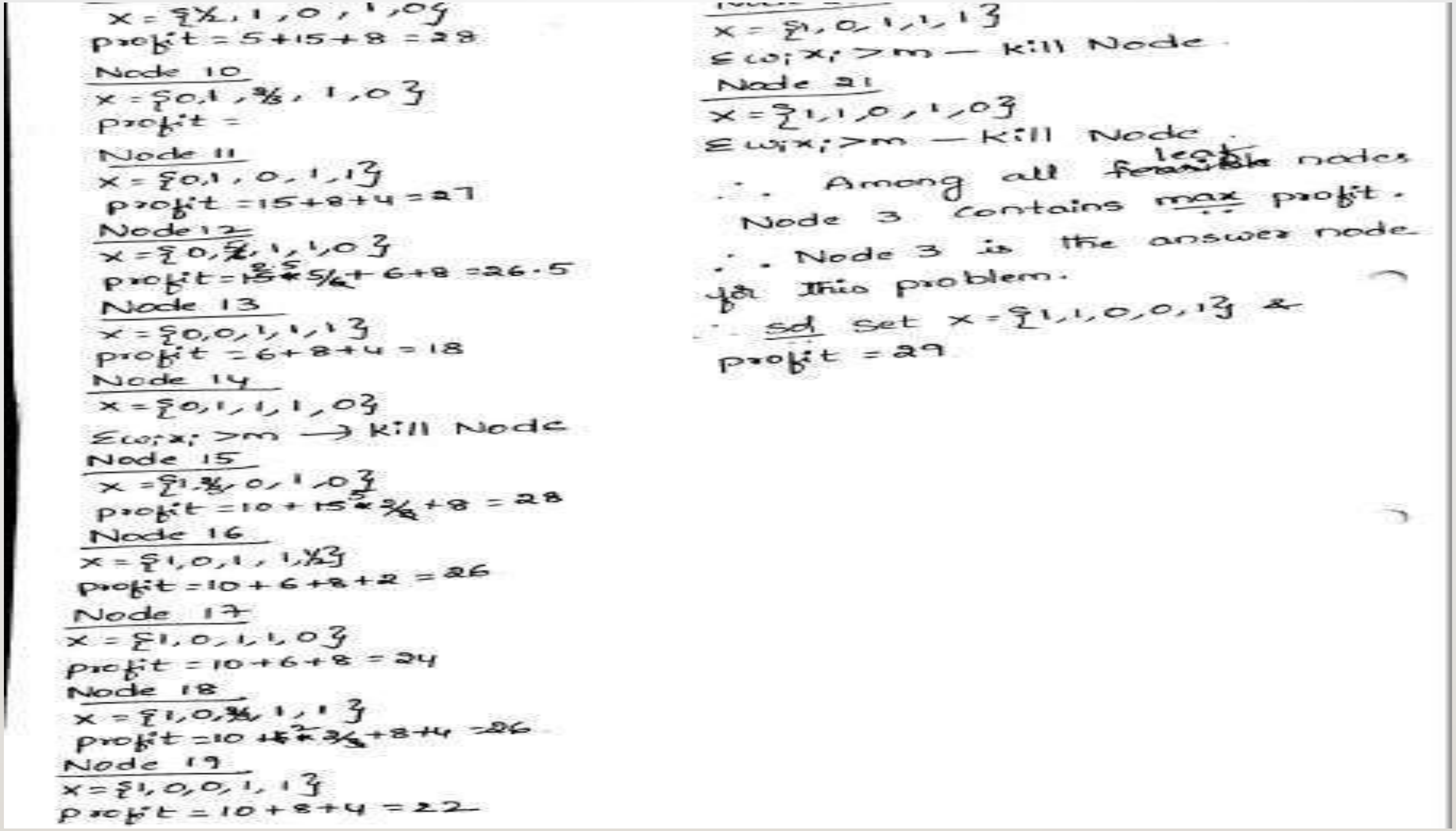
Node 4
$x = \{3/4, 1, 1, 0, 0\}$
profit $= \overset{2.5}{10} * 3/4 + 15 + 6 = 28.5$

Node 5
$x = \{0, 1, 1, 0, 1\}$
profit $= 15 + 6 + 4 = 25$

Node 6
$x = \{1, 5/6, 1, 0, 0\}$
profit $= 10 + \overset{2.5}{15} * 5/6 + 6 = 28.5$

Node 7
$x = \{1, 0, 1, 0, 1\}$
profit $= 10 + 6 + 4 = 20$

Node 8
$x = \{1, 1, 1, 0, 0\}$
$\Sigma w_i x_i > m$ — Kill Node.



$x = \{1/2, 1, 0, 1, 0\}$
profit = 5+15+8 = 28

Node 10
$x = \{0, 1, 2/3, 1, 0\}$
profit =

Node 11
$x = \{0, 1, 0, 1, 1\}$
profit = 15+8+4 = 27

Node 12
$x = \{0, 5/6, 1, 1, 0\}$
profit = $\overset{2.5}{\cancel{15}} * 5/6 + 6 + 8 = 26.5$

Node 13
$x = \{0, 0, 1, 1, 1\}$
profit = 6+8+4 = 18

Node 14
$x = \{0, 1, 1, 1, 0\}$
$\sum w_i x_i > m \longrightarrow$ kill Node

Node 15
$x = \{1, 2/3, 0, 1, 0\}$
profit = $10 + \overset{5}{\cancel{15}} * 2/\cancel{3} + 8 = 28$

Node 16
$x = \{1, 0, 1, 1, 1/2\}$
profit = 10+6+8+2 = 26

Node 17
$x = \{1, 0, 1, 1, 0\}$
profit = 10+6+8 = 24

Node 18
$x = \{1, 0, 2/3, 1, 1\}$
profit = $10 + \overset{2}{\cancel{15}} * 2/\cancel{3} + 8 + 4 = 26$

Node 19
$x = \{1, 0, 0, 1, 1\}$
profit = 10+8+4 = 22

$x = \{1, 0, 1, 1, 1\}$
$\sum w_i x_i > m$ — kill Node.

Node 21
$x = \{1, 1, 0, 1, 0\}$
$\sum w_i x_i > m$ — Kill Node.

$\therefore$ Among all ~~feasible~~ leaf nodes Node 3 contains max profit.

$\therefore$ Node 3 is the answer node for this problem.

$\therefore$ sol set $x = \{1, 1, 0, 0, 1\}$ &
profit = 29

Algorithm: Backtracking solutionto the 0/1knapsackproblem:

```
Algorithm BKnap(k, cp, cw)
// m is the size of the knapsack; n is the number of weights
// and profits. w[ ] and p[ ] are the weights and profits.
// p[i]/w[i] ≥ p[i+1]/w[i+1]. fw is the final weight of
// knapsack; fp is the final maximum profit. x[k] = 0 if w[k]
// is not in the knapsack; else x[k] = 1.
{
    // Generate left child.
    if (cw + w[k] ≤ m) then
    {
        y[k] := 1;
        if (k < n) then BKnap(k + 1, cp + p[k], cw + w[k]);
        if ((cp + p[k] > fp) and (k = n)) then
        {
            fp := cp + p[k]; fw := cw + w[k];
            for j := 1 to k do x[j] := y[j];
        }
    }
```

Algorithm: Backtracking solutionto the 0/1knapsackproblem:

```
// Generate right child.
    if (Bound(cp, cw, k) ≥ fp) then
    {
        y[k] := 0; if (k < n) then BKnap(k + 1, cp, cw);
        if ((cp > fp) and (k = n)) then
        {
            fp := cp; fw := cw;
            for j := 1 to k do x[j] := y[j];
        }
    }
}
```

Algorithm for bounding function:

```
Algorithm Bound(cp, cw, k)
// cp is the current profit total, cw is the current
// weight total; k is the index of the last removed
// item; and m is the knapsack size.
{
    b := cp; c := cw;
    for i := k + 1 to n do
    {
        c := c + w[i];
        if (c < m) then b := b + p[i];
        else return b + (1 - (c - m)/w[i]) * p[i];
    }
    return b;
}
```

SAMPLE QUESTIONS

- Differentiate between fractional knapsack and knapsack
- provide a step-by-step example of solving a specific knapsack problem using dynamic programming
- N = 3, W = 4, profit[] = {1, 2, 3}, weight[] = {4, 5, 1},Construct a state space tree
- Explain in detail about knapsack