Consider the following instance of knapsack prob

$n=3$, $m=20$, $P=(25,24,15)$
$w=(18,15,10)$

$\frac{P_1}{w_1} = \frac{25}{18} = 1.4$ $\frac{P_2}{w_2} = \frac{24}{15} = 1.6$ $\frac{P_3}{w_3} = \frac{15}{10} = 1.5$

Arrange in decreasing order

$\frac{P_2}{w_2} > \frac{P_3}{w_3} > \frac{P_1}{w_1}$

Initially $x[1..3] = [0,0,0]$ solutn vector

Check for 1st obj (i.e obj 2)
$w[i] < U$
$15 < 20$

then $x[2] = 1$
$U = U - w[i]$
$= 20 - 15$
$= 5$

$x[1..3] = [0,1,0]$

Probit $= P_i x_i = 24 \times 1 = 24$

Check for 2nd obj from List (obj 3)

$w_3 > U$
$5 > 10$

then $x[3] =$ ~~to~~ ~~$w[i]$~~ $U/w[i]$
$= 5/10 = 1/2$

$x[1..3] = [0,1,1/2]$

probit $= P_i x_i$
$= P_3 x_3 = 15 \times \frac{1}{2} = 7.5$

weight $= U - w_i x_i$
$5 - 10/2 = 0$

Total probit
$= 24 + 7.5$
$= \underline{\underline{31.5}}$

Ans

$n = 7, \quad m = 15$

$(P_1 \cdots P_7) = (10, 5, 15, 7, 6, 18, 3)$

$(w_1 — w_7) = (2, 3, 5, 7, 1, 4, 1)$

$\frac{P_1}{w_1} = \frac{10}{2} = 5, \quad \frac{P_2}{w_2} = \frac{5}{3} = 1.6 \quad \frac{P_3}{w_3} = \frac{15}{5} = 3, \quad \frac{P_4}{w_4} = \frac{7}{7} = 1$

$\frac{P_5}{w_5} = \frac{6}{1} = 6 \quad \frac{P_6}{w_6} = \frac{18}{4} = 4.5 \quad \frac{P_7}{w_7} = \frac{3}{1} = 3$

So $\frac{P_5}{w_5} > \frac{P_1}{w_1} > \frac{P_6}{w_6} > \frac{P_3}{w_3} \geq \frac{P_7}{w_7} > \frac{P_2}{w_2} > \frac{P_4}{w_4} = 1$

$x[1 \ldots 7] = [0\ 0\ 0\ 0\ 0\ 0\ 0]$

Add obj 5 in knapsack

$w_5 < m$ Solutⁿ vector

$1 < 15 \quad x[5] = 1$ $[0\ 0\ 0\ 0\ 1\ 0\ 0]$

$profit_1 = 6$

$U = U - w_5$

$= 15 - 1 = 14$

Add obj 1 in knap.

$w_1 < U$

$2 < 14 \quad x[1] = 1$ $[1\ 0\ 0\ 0\ 1\ 0\ 0]$

$profit = \Sigma 6 + 10 = 16$

$U = ~~U - m~~ \ U - w[i]$

$= 14 - 2 = 12$

Add obj 6 in knap.

$w_6 < m$ $[1\ 0\ 0\ 0\ 1\ 1\ 0\ 0]$

$4 < 12$

$x[4] = 1$ $\Sigma\, p_i x_i$

$U = U - w[i]$ $Profit = 16 + 18$

$= 12 - 4$ $= 34$

$= 8$

obj 3 in knap.

$w_3 < m$ [1 0 1 0 1 1 0]

$5 < 8$

$x[3] = 1$

$Var\ m = m - w[i]$

$= 8 - 5$

$= 3$

$Profit = \Sigma p_i x_i$

$= 34 + 15$

$= 49$

obj 7 in knap. [1 0 1 0 1 1 1]

$w_7 < m$

$1 < m$

$x[7] = 1$

$m = m - w[i]$

$= 3 - 1$

$= 2$

$Profit = \Sigma p_i x_i$

$= 49 + 3$

$= 52$

soln vector

$[1, 2/3\ 1\ 0\ 1\ 1\ 1]$

obj 2 in knap.

$w_2 > m$

$3 > 2$

$x[i] = U/w[i] = m/w[i]$

$= 2/3$

$Profit = 52 + P_2 x_2$

$52 + 5 \times \frac{2}{3} = 55.33$

$m = m - w[i] \cdot x[i]$

$= 2 - 3 \cdot 2/3$

$= 0$

Soln vector

x_i [1

$1 \le i \le n$

Subject to $\quad \Sigma\, w_i x_i \le m \quad$ ...(2)

$1 \le i \le n$

and $\quad 0 \le x_i \le 1,\ 1 \le i \le n \quad$ ...(3)

The profits and weights are positive.

- A feasible solution is any set $(x_1, x_2, \ldots, x_n)$ satisfying (2) and (3) above.
- Optimal solution is a feasible solution for which (1) is maximized.
- There are following three approaches to generate feasible solutions. We cannot guarantee that any of them will always generate the optimal solution.
  - o <u>Greedy by Profit</u>: At each step select from the remaining items the one with the highest profit (provided the capacity of the knapsack is not exceeded). This approach tries to maximize the profit by choosing the most profitable items first.
  - o <u>Greedy by Weight</u>: At each step select from the remaining items the one with the least weight (provided the capacity of the knapsack is not exceeded). This approach tries to maximize the profit by putting as many items into the knapsack as possible.
  - o <u>Greedy by Profit Density</u>: At each step select from the remaining items the one with the largest profit density, p_i/w_i(provided the capacity of the knapsack is not exceeded). This approach tries to maximize the profit by choosing items with the largest profit per unit of weight.
- Algorithm for greedy strategies for the knapsack problem will be as follows:

```
Algorithm GreedyKnapsack(m, n)
// p[1..n] and w[1..n] contain the profits & weights respectively of
// the n objects
// Ordered such that p[i]/w[i] ≥ p[i+1]/w[i+1].
// m is the knapsack size ad x[1..n] is the solution vector.
{
    for i :=1 to n do x[i] :=0.0;     //Initialize x.
      U:=m;
    for i:=1 to n do
    {
      if (w[i]>U) then break;
      x[i]:=1.0; U:=U-w[i];
    }
    if i≤n then x[i]:=U/w[i];
}
```

decreasing ⇒ [illegible]

increasing ⇒ [illegible]

Greedy by profit density

e.g. Find an optimal solution to the knapsack instance n=3, m=20, (p_1, p_2, p_3)=(25,24,15), and (w_1, w_2, w_3)=(18,15,10).

$\frac{25}{18} \quad \frac{24}{15} \quad \frac{15}{10}$

1.3 1.6 1.5

1.6 1.5 1.3

$\frac{P.\,24}{W.\,15} \quad \frac{15}{10} \quad \frac{25}{18}$

Sol: Arrange p_i's and w_i's in decreasing order of p_i/w_i

(24/15, 15/10, 25/18)

Initial x_i's to 0 and U to 20

i	p_i	w_i	U	x_i	$\Sigma p_i x_i$
1	24	15	5	1	24
2	15	10	5	5/10	24+15*1/2=24+7.5=31.5

Solution Vector = {1,1/2,0}

Maximum profit = 31.5

e.g. Find an optimal solution to the knapsack instance n=7, m=15, $(p_1,p_2,\dots,p_7)$=(10,5,15,7,6,18,3), and $(w_1,w_2,\dots,w_7)$ = (2,3,5,7,1,4,1).

Sol: Arrange p_i's and w_i's in decreasing order of p_i/w_i

(6/1,10/2,18/4,15/5,3/1,5/3,7/7}

Initial x_i's to 0 and U to 15

i	p_i	w_i	U	x_i	Σp_ix_i
1	6	1	14	1	6
2	10	2	12	1	6+10=16
3	18	4	8	1	16+18=24
4	15	5	3	1	24+15=49
5	3	1	2	1	49+3=52
6	5	3	2	2/3	52+5*2/3=55.33

Solution Vector = {1,1,1,1,1,2/3,0}

Maximum profit = 55.33

e.g. Find an feasible solution to the knapsack instance n=7, m=23, $(p_1,p_2,\dots,p_7)$=(15,19,7,5,2,11,7), and $(w_1,w_2,\dots,w_7)$ = (5,6,3,4,1,5,2).

Sol:

<u>Greedy by profit density</u>

Arrange p_i's and w_i's in decreasing order of p_i/w_i

(7/2,19/6,15/5,7/3,11/5,2/1,5/4)

Initial x_i's to 0 and U to 23

$w_i < U$

$\{ U - w_i = U$

$x_i = 1$

i	p_i	w_i	U	x_i	Σp_ix_i
1	7	2	21	1	7
2	19	6	15	1	7+19=26
3	15	5	10	1	26+15=41
4	7	3	7	1	41+7=48
5	11	5	2	1	48+11=59
6	2	1	1	1	59+2=61
7	5	4	1	1/4	61+5*1/4=61+1.25=62.25

Solution Vector = {1,1,1,1,1,1,1/4}

Maximum profit = 62.25

<u>Greedy by profit</u>

Arrange p_i's and w_i's in decreasing order of p_i

(19/6, 15/5, 11/5, 7/3,7/2, 5/4,2/1)

Initial x_i's to 0 and U to 23

i	p_i	w_i	U	x_i	Σp_ix_i
1	19	6	17	1	19
2	15	5	12	1	19+15=34
3	11	5	7	1	34+11=45
4	7	3	4	1	45+7=52

5	7	2	2	1	52+7=59
6	5	4	2	2/4	59+5*2/4=61.5

Solution Vector = {1,1,1,1,1,1/2,0}

Maximum profit = 61.5

Greedy by weight

Arrange p_i's and w_i's in increasing order of w_i

(2/1, 7/2, 7/3, 5/4,15/5, 11/5, 19/6)

Initial x_i's to 0 and U to 23

i	p_i	w_i	U	x_i	$\Sigma p_i x_i$
1	2	1	22	1	2
2	7	2	20	1	2+7=9
3	7	3	17	1	9+7=16
4	5	4	13	1	16+5=21
5	15	5	8	1	21+15=36
6	11	5	3	1	36+11=47
7	19	6	3	3/6	47+19*3/6=56.5

Solution Vector = {1,1,1,1,1,1,1/2}

Maximum profit = 56.5

e.g. Let $a_1, a_2, \ldots, a_n$ be *n* coin types having values such that $a_1 \geq a_2 \geq \ldots \geq a_n$ and $a_n = 1$. Each coin type available in unlimited quantity. Design a greedy algorithm that makes up exact amount 'C' using m number of coins. Solution should give information about the number of coins used for each

3.4 JOB SEQUENCING WITH DEADLINES

We are given a set of *n* jobs. Associated with *i* is an integer deadline $d_i \geq 0$ and a profit $p_i > 0$. For any job *i* the profit p_i is earned iff the job is completed by its deadline. To complete a job, one has to process the job on a machine for one unit of time. Only one machine is available for processing jobs. A feasible solution for this problem is a subset *J* of jobs such that each job in this subset can be completed by its deadline. The value of a feasible solution *J* is the sum of the profits of the jobs in *J*, or $\Sigma_{i \in J} p_i$. An optimal solution is a feasible solution with maximum value.

e.g. Let $n = 4, (p_1, p_2, p_3, p_4) = (100, 10, 15, 27)$ and $(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$

	Feasible \solution	Processing sequence	value
1	(1,2)	2,1	110
2	(1,3)	1,3 or 3, 1	115
3	(1,4)	4, 1	127
4	(2,3)	2, 3	25
5	(3,4)	4,3	42
6	(1)	1	100
7	(2)	2	10
8	(3)	3	15
9	(4)	4	27

Solution 3 is optimal. In this solution only jobs 1 and 4 are processed and the value is 127. These jobs must be processed in the order job 4 followed by job 1. Thus the processing of job 4 begins at time zero and that of job 1 is completed at time 2.

To formulate a greedy algorithm to obtain an optimal solution, we must formulate an optimization measure to determine how the next job is chosen. As a first attempt we can choose the objective function $\Sigma_{i\in J} p_i$ as our optimization measure. Using this measure, the next job to include is the one that increases $\Sigma_{i\in J} p_i$ the most, subject to the constraint that the resulting J is a feasible solution. This requires us to consider jobs in non-increasing order of the p_i's.

To determine whether or not a given J is a feasible solution, one way would be to try out all possible permutations of jobs in J and check if the jobs in J can be processed in any one of these permutations without violating the deadlines.

e.g. Obtain optimal solutions for the following jobs:

$n = 4, (p_1,p_2,p_3,p_4) = (100,10,15,27)$ and $(d_1, d_2, d_3, d_4) = (2,1,2,1)$

Sorted profits $(p_1,p_4,p_3,p_2) = (100,27,15,10)$ and $(d_1, d_4, d_3, d_2) = (2,1,2,1)$

Jobs sequence with (4,1) with profit 127.

e.g. $n=7, (p_1,p_2,p_3,p_4,p_5,p_6,p_7) = (30,20,18,6,5,3,1)$ and $(d_1,d_2,d_3,d_4,d_5,d_6,d_7) = (2,4,3,1,3,1,2)$

Job sequence (4,1,3,2) with profit 74.

e.g. $n=5, (p_1,p_2,p_3,p_4,p_5) = (20,15,10,5,1)$ and $(d_1, d_2, d_3, d_4, d_5) = (2,2,1,3,3)$

Job sequence (1,2,4) with profit 40.

Let us see an algorithm to construct an optimal set of jobs which can be complete by their deadlines.

```
Algorithm GreedyJob(d, J, n)
// J is a set of jobs that can be completed by their deadlines
{
    J := {1};
    for i := 2 to n do
    {
        if all jobs in J∪{i} can be completed by their deadlines then
            J:=J∪{i};
    }
}
```

Let us see greedy algorithm for sequencing unit time jobs with deadlines and profits.

```
Algorithm JS(d, J, n)
// d[i]≥1, 1≤i≤n are the deadlines, n≥1.
// The jobs are ordered such that p[1]≥p[2]≥...≥p[n].
// J[i] is the ith job in the optimal solution, 1≤i≤k.
// Also, at termination d[J[i]]≤d[J[i+1]], 1≤i<k.
{
    d[0] := J[0] := 0;           // Initialize.
    k:=J[1] := 1;                 // Include job 1.
    for i:=2 to n do
    {
        // Consider jobs in non-increasing order of p[i].
        // Find position for i and check feasibility of insertion.
        r = k;
    while ((d[J[r]] > max(d[i],r)) do r:=r-1;
       if ((d[J[r]]≤d[i]) and (d[i]>r))
    {
       // Insert i into J[].
       for q:=k to r+1 step - 1 do J[q+1]:=J[q];
       J[r+1]:=i;
       k:=k+1;
        }
  }
    return k;
}
```

For each job i, schedule i at time t, where t is the largest integer such that $1 \le t \le \min(n, d[i])$ and job to be executed at time t is not yet decided.

In the above algorithm, the jobs are considered in the decreasing order of profits. n is the number of jobs. So $p[1] \ge p[2] \ge \dots p[n]$. Deadlines of the jobs are in the array $d[]$ and $d[i] \ge 1$ for $1 \le i \le n$. At termination of algorithm, the optimal solution will be in the array $J[]$ and k is the number of jobs in optimal set. On the termination of the above algorithm $d[J[i]] \le d[J[i+1]]$ for $1 \le i \le k$.

e.g. Let n=5, $(p_1,\dots p_5)$=(20,15,10,5,1) and $(d_1,\dots, d_5)$=(2,2,1,3,3).
$\max(d_1,\dots, d_5)$=3. Therefore maximum 3 jobs can be completed.

J	1	
d[*i*]	2	
Slot Assigned	[0-1]	

J	1	2	
d[*i*]	2	2	
Slot Assigned	[0-1]	[1-2]	

Reject job 3

J	1	2	4	
d[*i*]	2	2	3	
Slot Assigned	[0-1]	[1-2]	[2-3]	

Reject job 5

The optimal solution is *J*={1,2,4} with a profit of 40.

e.g. Let n=7, $(p_1,\dots p_7)$=(20,15,10,7,5,3,2) and $(d_1,\dots, d_7)$=(3,1,1,3,1,3,4).
$\max(d_1,\dots, d_7)$=4. Therefore maximum 4 jobs can be completed.

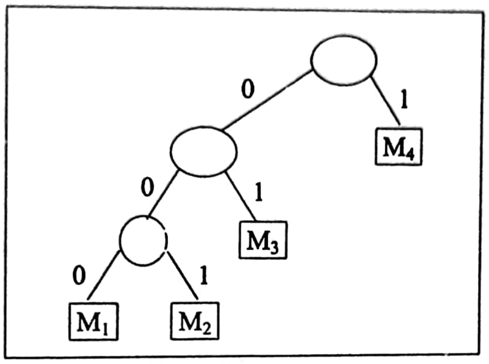
J	1
d[*i*]	3
Slot Assigned	[0-1]

J	2	1
d[*i*]	1	3
Slot Assigned	[0-1]	[1-2]

3.5 HUFFMAN CODES

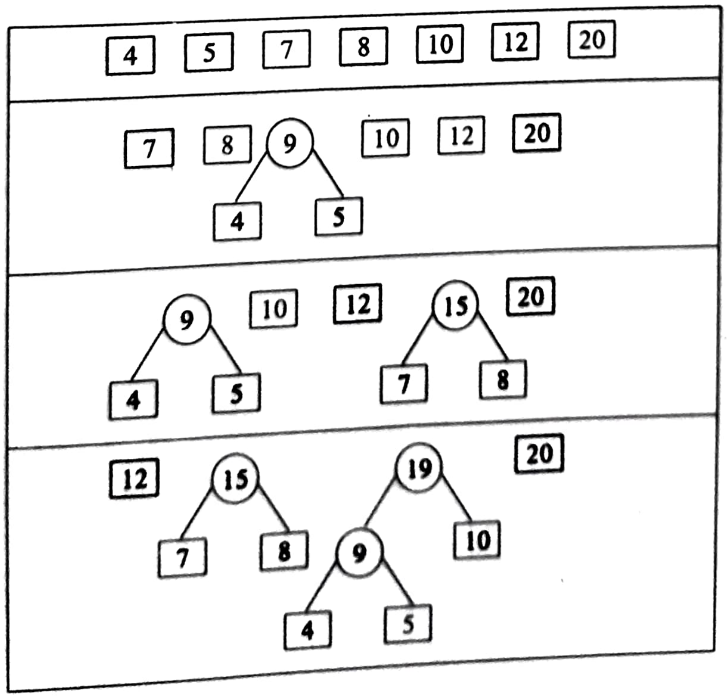
Another application of binary trees with minimal weighted external path length is to obtain an optimal set of codes for messages $M_1, M_2, \ldots M_{n+1}$. Each code is a binary string that is used for transmission of the corresponding message. At the receiving end the code is decoded using a **decode tree**. A decode tree is a binary tree in which external nodes represent messages.

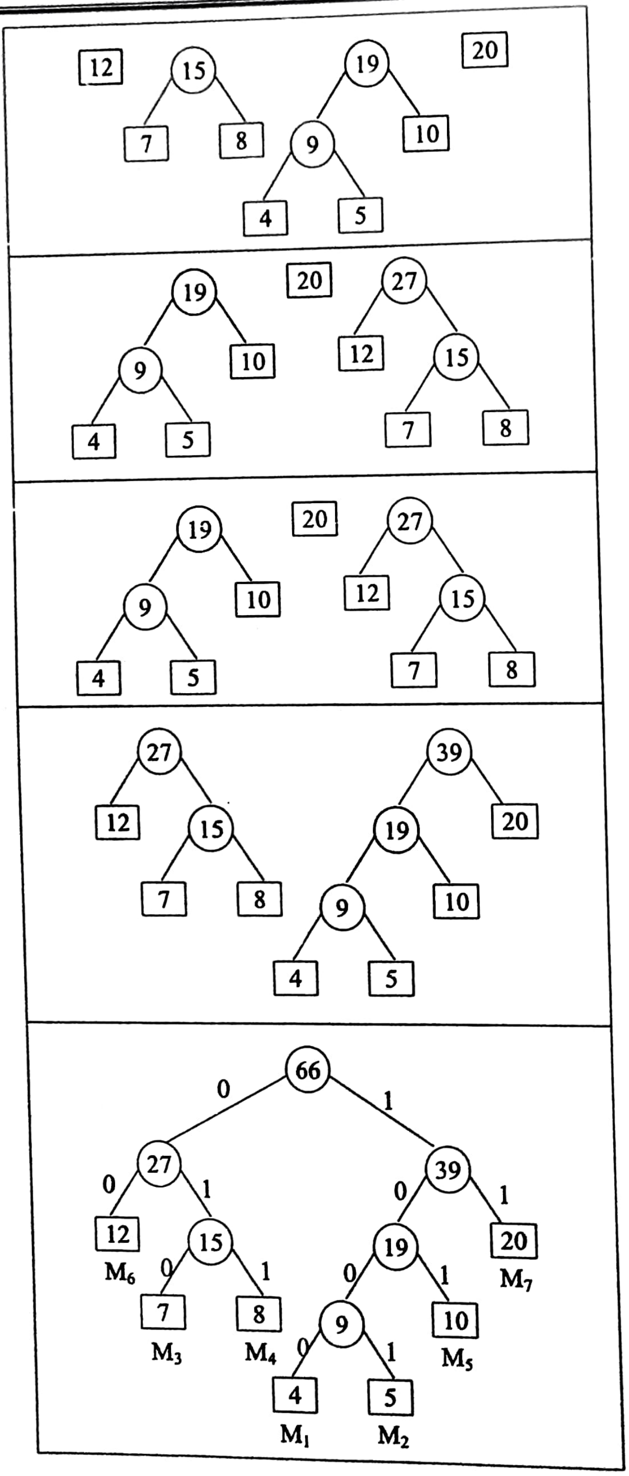
The binary bits in the code word for a message determine the branching needed at each level of the decode tree to reach the correct external node. For example, if we interpret a zero as a left branch and a one as a right branch, then the decode tree of figure below corresponds to codes 000, 001, 01, and 1 for messages M_1, M_2, M_3, M_4 respectively.



These codes are called Huffman codes. The cost of decoding a code word is proportional to the number of bits in the code. This number is equal to the distance of the corresponding external node from the root node. If q_i is the relative frequency with which message M_i will be transmitted, then the expected node decode time is $\sum_{1\le i\le n+1} q_i d_i$, where d_i is the distance of the external node for message M_i from the root node. The expected decode time is minimized by choosing code words resulting in a decode tree with minimal weighted external path length. Note that $\sum_{1\le i\le n+1} q_i d_i$ is also the expected length of a transmitted message. Hence the code that minimizes expected decode time also minimizes the expected length of a message.

e.g. Obtain a set of optimal Huffman codes for the messages $(M_1, \ldots M_7)$ with relative frequencies $(q_1, \ldots q_7) = (4,5,7,8,10,12,20)$





Message	Code word
M_1	1000
M_2	1001
M_3	010
M_4	011
M_5	101
M_6	00
M_7	11