PAGE No.
DATE / /

(4) Job Sequencing using deadlines
- List of n jobs with each job i has a deadline $d_i \geq 0$ and profit $p_i > 0$ earned only if job completed

- Find a subset J s.t. each job of this subset can be completed within deadline; maximizing profit

① Sort p_i into decreasing order
② Add next job i to the soln set J if i can be completed by deadline.
- Assign i to time slot $[r-1, r]$, where r is $1 \leq r \leq d_i$ and slot $[r-1, r]$ is free.
* Schedule in latest possible free slot for optimal soln
③ Repeat step 2 till slot full or jobs examined.

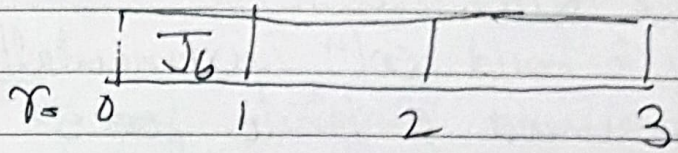
Complexity :- $O(n^2)$

$p = \{p_1, p_2, p_3 \ldots p_7\} = [5, 4, 8, 7, 6, 9, 3]$
$d = \{d_1, d_2, d_3 \ldots, d_7\} = [3, 2, 1, 3, 2, 1, 2]$

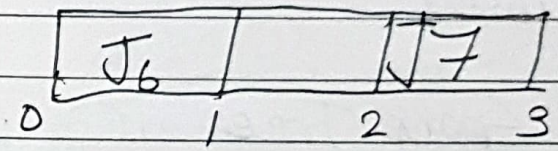
Sort in descending order

i	6	3	4	5	1	2	7
p	9	8	7	6	5	4	3
d	1	1	3	2	3	2	2

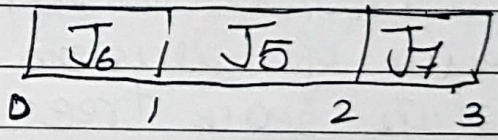
Job scheduling



Early loop:-
J_3 - Rejected
J_4 - Accepted



J_5 - Accept



→ Optional :- ① Optimal Storage on tapes
② Huffmann Coding

PAGE No.
DATE / /

8. BACKTRACKING

Try to build soln incrementally
Refinement of brute force.
Systematically search set of feasible soln "solution space" to solve given problem.
From n solutions $(x_1, x_2 \dots x_n)$, x_i is chosen.

⇒ State Space Tree

State Space :- Set of states a system can exist in
Soln to problem is obtained by sequence of decisions resulting in state space tree.
Root :- State before any decision

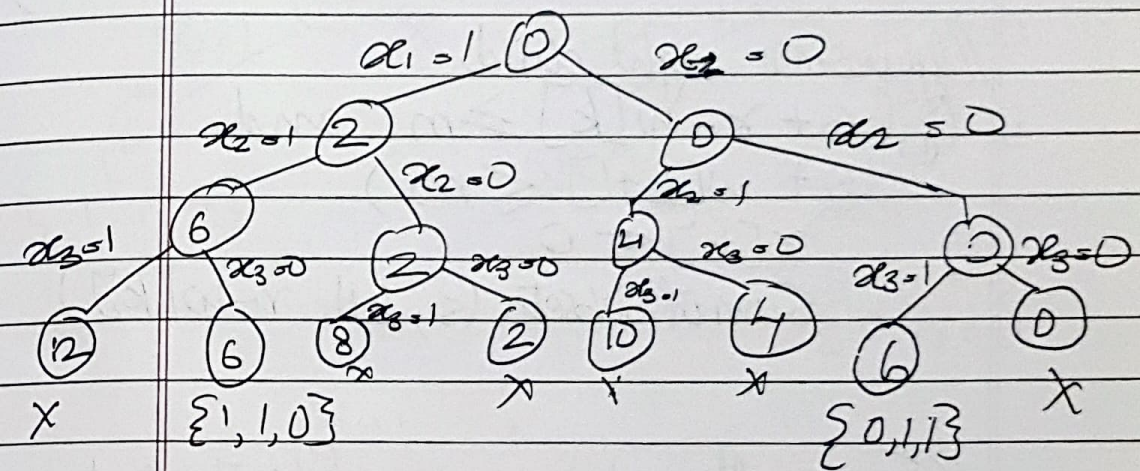
① Sum of Subsets

Given n positive integers find subsets of $w_1 \dots w_n$ that sum to S.
Soln vector $\{x_1, x_2 \dots x_n\}$ x_i is 1 if included, else 0.

Binary State Space Tree :-
Node at depth 1 is for item1 (yes/no) at depth 2 for item2, etc.
left ieft branch include w_i;
right branch excludes w_i.

Nodes contain sums of weights so far
At every stage, we check whether node is valid or not. If invalid, we backtrack to parent.

Ex
$w_1 = 2,\ w_2 = 4,\ w_3 = 6$ & $S = 6$ ← o/p



Initial call :- (0, 1, r)
s :- current sum; initialised to 0
k :- current elt to consider
r :- sum of all elts w[1 : n]

PAGE No.
DATE / /

```
SumofSubsets(s, k, r)
// Generate left child
x[k] ← 1
if (s+w[k] = m)
    write x[1:k] // one subset found
else if (s+w[k]+w[k+1] ≤ m)
    SumofSubsets(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
    SumofSubsets(s, k+1, r-w[k])
```

// Find all subsets of w[1:n] that sum to m

$$s = \sum_{j=1}^{k-1} w[j] * x[j]$$

$$r = \sum_{j=k}^{n} w[j]$$

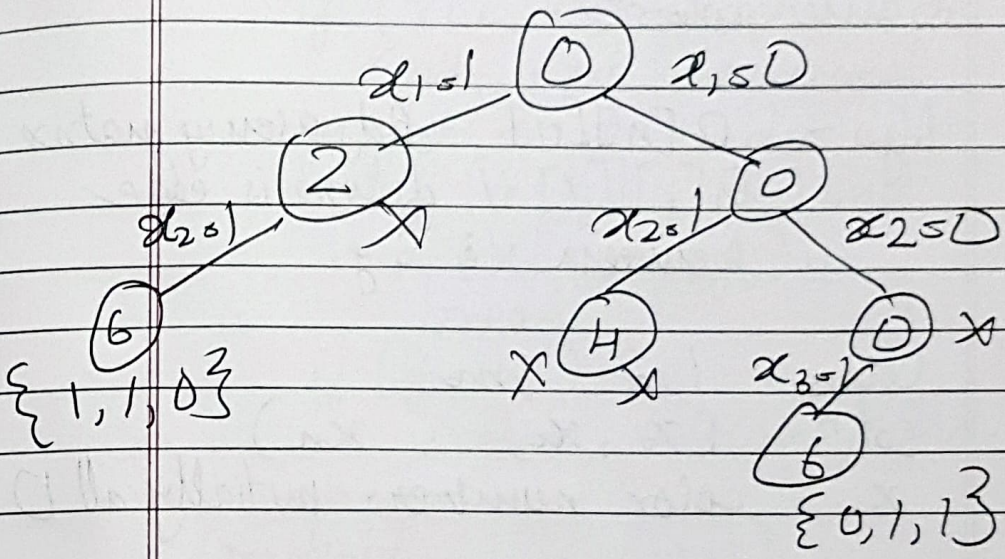
w[j] are in ascending order (sorted array)

∴ it is assumed that $w[i] \leq m$ & $\sum_{i=1}^{n} w[i] \geq m$

left &

Abv generation of right child results in a pruned tree.

$w_1 \leq 2$, $w_2 \leq 4$, $w_3 \leq 6$ $S \leq 6$



Complexity = $O(2^n)$

(2) Graph Coloring

G - Graph, m - No. of colors (positive int)

No two adjacent nodes can have same color.

Algo:- G[n][n] - Adjacency matrix
G[i][j] = 1 if there is edge between i & j.

Colors = 1, 2 ..., m

Soln $(x_1, x_2, \ldots x_n)$

x_i = color number. initially all 0

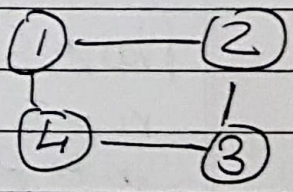
↓ Node (Initial k = 1)

```
Graph Coloring (k)
while (true)
NextValue (k)
if (x[k] = 0)
    return // No new color possible
if (k = n)
    write(x[1..n])
else
    GraphColoring (k+1)
```

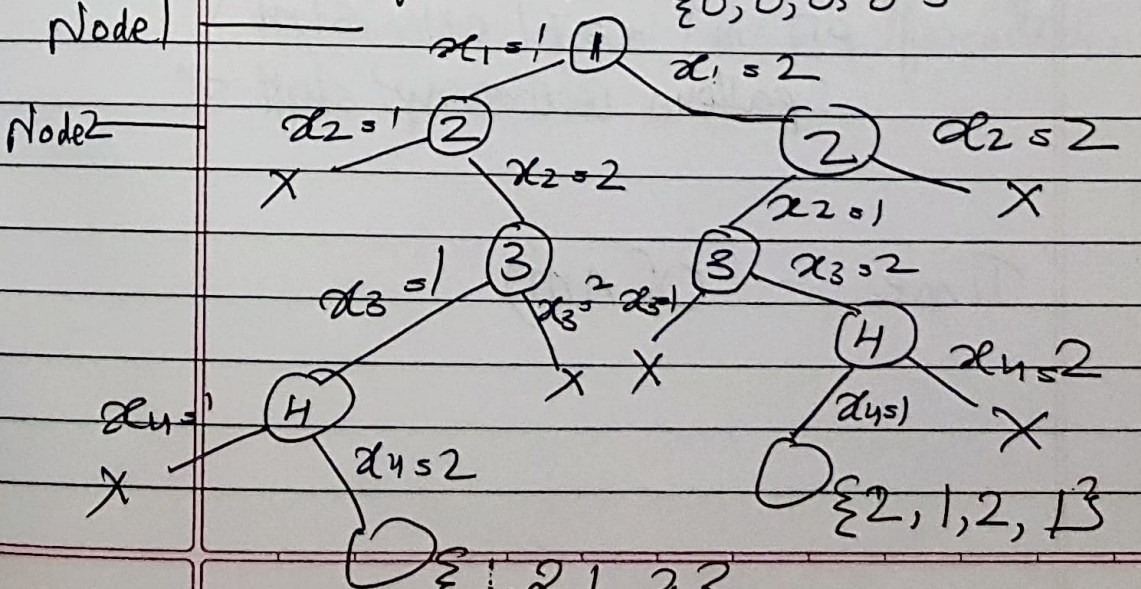
```
NextValue (k)
while true
    x[k] = (x[k]+1) mod(m+1) // Next
                  // Highest color
    if (x[k] = 0) return
      // All colors are used
    for j ← 1 to n do
        if (G[k][j] ≠ 0) and
           (x[k] = x[j])
          break
    if (j = n+1)
          return // Color found.
```

complexity = $O(mn)$

Ex: 4 nodes, 2 colors



State Space tree- {0, 0, 0, 0}



PAGE No.
DATE / /

9. STRING AND PATTERN MATCHING

→ PATTERN MATCHING
Given Text (T) & Pattern (P), pattern matching finds substring of T equal to P
Algorithms:

① Naive String Matching Algo :-
Brute force approach.
Compares first char of P wt T
if matched, increment both ptrs
else, shift pointer of T only & reset P.
Repeat till end of text or pattern found

```
Naive String Match (T, P)
n = T.length
m = P.length
for s=0 to (n-m)
    if P[1..m] == T[s+1..s+m]
        "pattern occurs w/ shift s"
```

Time :- O(nm)

(2) RABIN KARP

Generalizes well to related problems like 2-D pattern matching.

Idea

Compare a string's hash value.

Algo slides pattern one by one matching hash value of P w/ hash of current substr of T.

If value matches, check if P matches substring.

Converting substr & P to hash & comparing nos is easier.

Input :- Text T, Pattern P, Radix d (UNICODE, ASCII(256), etc), prime no q.

Radix $|\Sigma|$ - set of possible characters in a string.

For ascii, 256 characters are possible.

```
RABIN-KARP (T, P, d, q)
n = T.length
m = P.length
h = d^(m-1) mod q
p = 0
t0 = 0
for i = 1 to m          // Preprocess
    p = (dp + P[i]) mod q
    t0 = (d t0 + T[i]) mod q

for s = 0 to n-m        // Match
    if p == ts
        if P[1..m] == T[s+1 ... s+m]
            print "Pattern at shift" s
    if s < n-m
        ts+1 = (d(ts - T[s+1].h)
                + T[s+m+1]) mod q
```

Time = $O((n-m+1)m)$

Optimal Time $O(m+n)$