2. Greedy (Solution) Method

- Greedy method is a straight forward technique if a problem has n inputs and require us to obtain a subset that satisfies some constraints any subset that satisfy these constraints is called a feasible solution.
- We need to find a feasible solution that either maximiz or minimiz a given objective function.
- A feasible solution that does this is called an optimal solution
- The greedy method such as that one can device an algorithm that works in stages considering one input at a time
- At each stage decision is made regarding whether a particular input is in a an optimal solution.

```
Greedy (a,n)
{
 solution = ϕ
 for i = 1 to n do
 {
  x = select (a);
  if feasible (solution, x) then
      solution = Union (solution, x);
 }
 return Solution;
}
```

1-08-24 YOUVA

Method

inputs
bset
any
nts is

ion that
ven

is called

one
ks in
time
regarding
an

In feasible solutions, maximum feasible soln is an optimal solution.

01-08-24 YOUVA

* Knapsack Problem:

Q. 1 mp 5 M = 7 M

n = 7 m = 15

object	1	2	3	4	5	6	7
Profit	10	5	15	7	6	18	3
Weight	2	3	5	7	1	4	1

- P/w: $\frac{5}{2}$ $\frac{1.6}{6}$ $\frac{3}{4}$ $\frac{1}{7}$ $\frac{6}{1}$ $\frac{4.5}{3}$ $\frac{3}{5}$

	2 - 2 = 0	object 2
2/3	3 - 1 = 2	object 7
	8 - 5 = 3	object 4
	12 - 4 = 8	object 6
	14 - 2 = 12	object 1
	15 - 1 = 14	object 5

15

x = { 1 2/3 1 0 1 1 1 }
(1 2 3 4 5 6 7)

- $\Sigma x_i w_i = 1\times2 + \frac{2}{3}\times3 + 1\times5 + 0\times7 + 1\times1 + 1\times4 + 1\times1$

$\Sigma x_i w_i = 2 + 2 + 5 + 1 + 4 + 1$

$\underline{\Sigma x_i w_i = 15}$

- $\Sigma x_i p_i = 1\times10 + \frac{2}{3}\times5 + 1\times15 + 0\times7 + 1\times6 + 1\times18 + 1\times3$

$\Sigma x_i p_i = 10 + 3.3 + 15 + 6 + 18 + 3$

$\therefore \underline{\Sigma x_i p_i = 55.3}$

- Algorithm:

Greedy knapsack (m, n)
P[1:n] and w[1:n] are profit, weight respectively n objects such that

$\frac{P[i]}{w[i]} \geq \frac{P[i+1]}{w[i+1]}$

m is knapsack size
{
for i = 1 to n do x[i] = 0;

```
U = m;
for i = 1 to n do
{
if ( w[i] > U ) then break;
        x[i] = 1;
        U = U - w[i];
}
if ( i ≤ n) then x[i] = U/w[i];
}
```

We are given n objects and a knapsack. object i has weight w_i and knapsack has capacity m. If a fraction x_i of object i is placed into the knapsack then profit of $p_i x_i$ is earned. The objective is to obtain a filling of knapsack that maximiz the total profit earn. Since the knapsack capacity is m we require the total weight of all choosen objects to be atmost m.

$$\text{maximize} \sum_{1 \le i \le n} p_i x_i$$

$$\text{Subject to} \sum w_i x_i \le m$$

and

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

Q. Find an optimal solution to the knapsacks instance n = 7, m = 15 pr

Object	1	2	3	4	5	6	7
profit	15	10	7	18	3	4	2
weight	2	3	5	7	1	4	1

• $\frac{P}{W}$ $\frac{7.5}{1}$ $\frac{3.3}{2}$ $\frac{1.4}{6}$ $\frac{2.5}{4}$ $\frac{3}{3}$ $\frac{1}{7}$ $\frac{2}{5}$

1/5	1 - 1 = 0	object 3
	2 - 1 = 1	object 7
	3 - 1 = 2	object 4
	10 - 1 = 9	object 5
	13 - 3 = 10	object 2
	15 - 2 = 13	object 1

15

$x = \{1, 1, 1/5, 1, 1, 0, 1\}$

• $\Sigma x_i w_i = 1 \times 2 + 1 \times 3 + 1/5 \times 5 + 1 \times 7 + 1 \times 1 + 0 + 1 \times 1$

$= 2 + 3 + 7 + 1 + 1 + 1$

$\therefore \Sigma x_i w_i = 15$

• $\Sigma x_i p_i = 1 \times 15 + 1 \times 10 + 1/5 \times 7 + 1 \times 18 + 1 \times 3 + 0 + 1 \times 2$

$= 15 * 10 + 1.4 + 18 + 3 + 2$

$\therefore \Sigma x_i p_i = 49.4$

* Job sequencing with deadlines.

We are given set of n jobs associated with Job i is an integer deadline, $d_i \geq 0$ and profit $P_i > 0$ for any job i the profit P_i is earn iff the job is completed by its deadline. To complete a job one has to process the job on a machine for 1 unit of time. Only one machine is available for processing job. feasible solution for these problem is a subset j of jobs such that each job in the subset can be completed by its deadline. The value of feasible solution j is the sum of profits of jobs in j.

and optimal solution is feasible solution with maximum value

~~eg~~ ~~$(P_1, P_2, P_3, P_4, P_5)$~~

- Algorithm

```
Greedy Job (d, J, n)
{
 J = {1};
for i = 2 to n do
{
 if (all jobs in J ∪ {i} can be completed
   by then deadlines then J = J ∪ {i};
}
}
```

jobs ← deadline
profit

eg. $(P_1\ P_2\ P_3\ P_4\ P_5) = (20, 15, 10, 5, 1)$
$(d_1\ d_2\ d_3\ d_4\ d_5) = (2, 2, 1, 3, 3)$
$J_1\ J_2\ J_3\ J_4\ J_5$

deadline 0 — $\frac{J_1}{J_2}$ — 1 — $\frac{J_2}{J_1}$ — 2 — $\frac{J_4}{J_4}$ — 3

Job consider	slot assign	solution	Profit
—	—	—	0
J_1	[1,2]	J_1	20
J_2	[0,1] [1,2]	$J_1 J_2$	20+15
J_3	[0,1] [1,2]	$J_1 J_2$	20+15
J_4	[0,1] [1,2] [2,3]	$J_1 J_2 J_4$	20+15+5
J_5	[0,1] [1,2] [2,3]	$J_1 J_2 J_4$	40

Q.

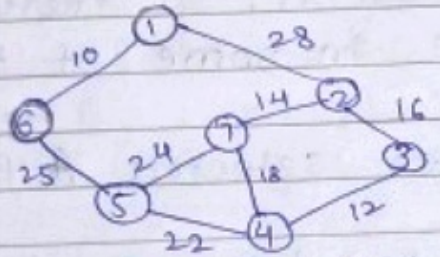
Job	J_1	J_2	J_3	J_4	J_5	J_6	J_7
profit	35	30	25	20	15	12	5
deadline	3	4	4	2	3	1	2

02-08-24

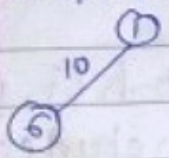
deadline 0 —J4— 1 —J3— 2 —J1— 3 —J2— 4

Job consider	slot assign	solution	Profit
	—	—	0
J_1	[2,3]	J_1	35
J_2	[2,3][3,4]	J_1J_2	35+30
J_3	[2,3][3,4][1,2]	$J_1J_2J_3$	35+30+25
J_4	[0,1][1,2][2,3][3,4]	$J_4J_3J_1J_2$	35+30+25+20
J_5	[0,1][1,2][2,3][3,4]	$J_4J_3J_1J_2$	35+30+25+20
J_6	[0,1][1,2][2,3][3,4]	$J_4J_3J_1J_2$	35+30+25+20
J_7	[0,1][1,2][2,3][3,4]	$J_4J_3J_1J_2$	35+30+25+20
			110

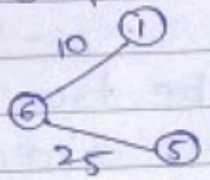
* Minimum Cost Spanning Tree :



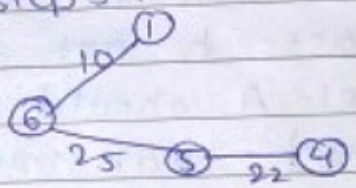
step 1 :



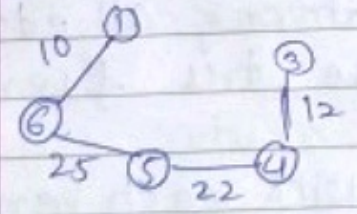
step 2 :



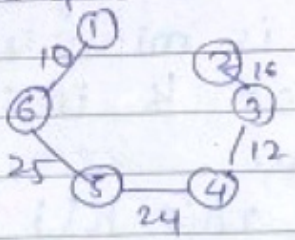
step 3 :



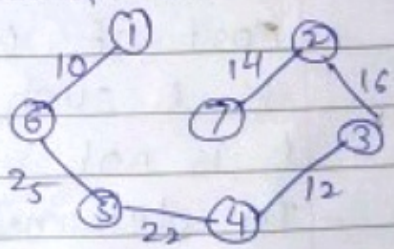
step 4 :



step 5 :

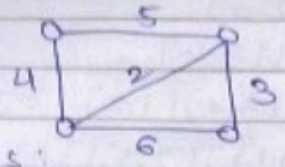


step 6 :

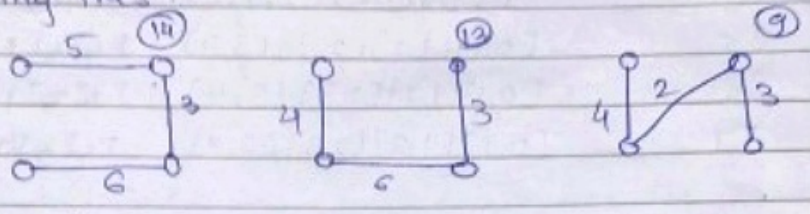


- * Let $G = V, E$ be undirected connected graph. subgraph $t = V, E$ of graph G is a spaning tree of graph G iff t is a tree.

- Its spanning tree number of vertices are set & connected with pure no. of edges. 'n' vertices atleast 'n-1' edges & connected graph is a tree ie. no cycle in graph



Spanning trees:



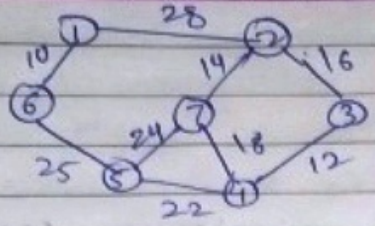
* Prims Algorithm:
- Select minimum cost h, so there is minimum increase in some of the cost of edges.
- 'A' is a set of edges selected A forms a tree
- Next h is u,v included in A is minimum cost h not in A.
- set A should always be tree. select h(i,j) such that vertex i is already included in tree and j is not yet included.
- cost of i,j is minimum among all edges k,l such that k is in the tree & vertex l is not in tree.
- To determine this h(i,j) with each vertex j not get included in the tree. A value near [j] is minimum among all choices.

• Algorithm

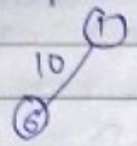
```
Prims(E, cost, n, t)
E is set of Edges
{
 let (k, l) be an edge of mimimum cost in E,
  mincost = cost [k, l]
   t[i,1] = k , t[i,2] = l        value to assign adjcent vertex as k, l
  for i = 1 to n do
  if (cost [i, l] < cost [1, k]) then
      near [i] = l  else
       near [i] = k ;
 for i = 2 to  n-1  do
{
let j be an index such that
        near [j] ≠ 0                    next vertex of
 cost [j, near [j]] is minimum          their edges are
   t [i, 1] = j ; t = [i, 2] = near [j]   j & near(j)

 mincost = mincost + cost [j, near [j]]; add of
  near [j] = 0;                          weights
  for k = 1 to n do
if ((near k ≠ 0) & cost( k, near[k]] > cost[k, j]
then
    near [k] = j;
}
   return mincost;
}
```

The graph should be tree in all stages.

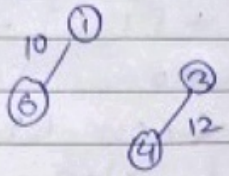
* Kruskal's method:



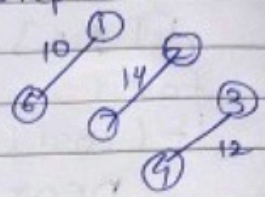
Step 1 :



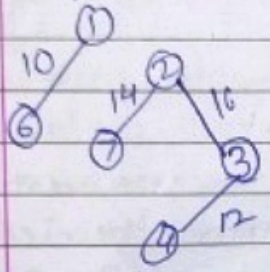
Step 2 :



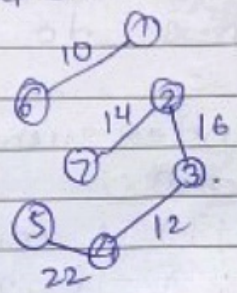
step 3 :



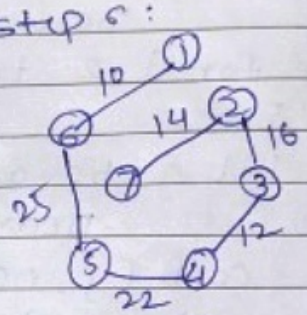
step 4 :



step 5 :



step 6 :

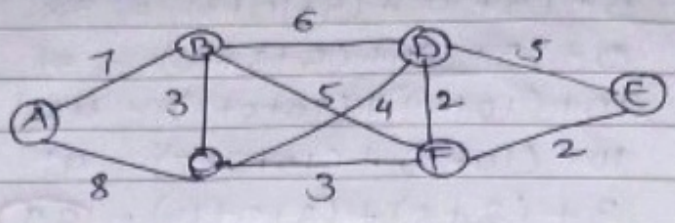


t = ϕ;
while(t has less than n-1 edges) && (E ≠ 0). do
{
choose an edge (v, w) from E of lowest cost;
Delete (v, w) from E;
if (v, w) does not create a cycle in t
then add (v, w) to t,
else discard (v, w);
}

- A set of edges so far selected for the spaning tree we such that it is possible to complete t into a tree. * t may not be tree in all stages in algorithm.

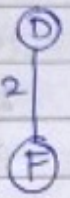
it only being a for set since a set of edges t can be completed in root tree iff there are no cycle in tree.

eg ①

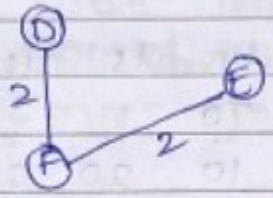


→ • Prim's method:

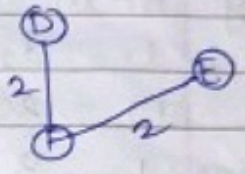
Step 1:



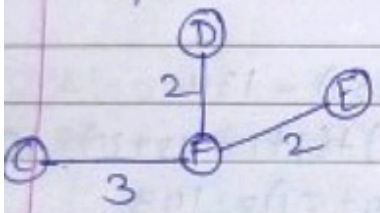
Step 2:



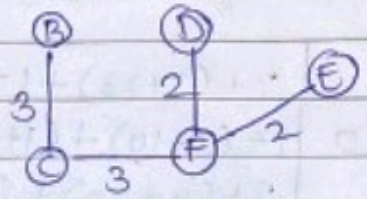
Step 3:



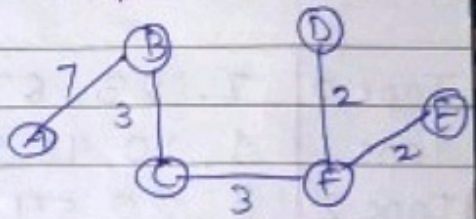
Step 4:



Step 5:

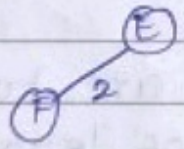


Step 6:

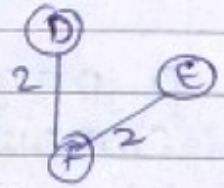


• Kruskat's method:

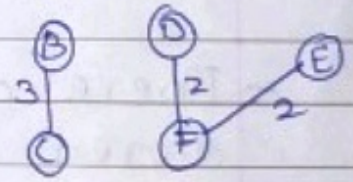
Step 1:



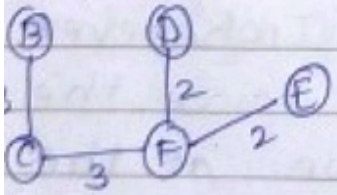
Step 2:



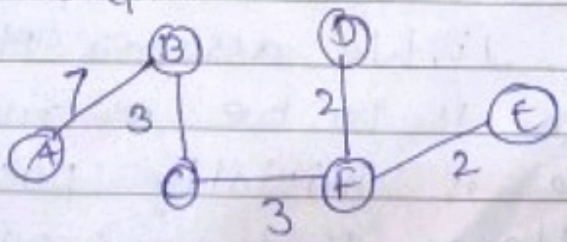
Step 3:



Step 4:



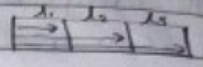
Step 5:



index mod no. of tapes
mod 3

Page No.
Date: 09-08-24

* Optimal storage on tape:



$n = 3$

$(l_1, l_2, l_3) = (5, 10, 3)$

$(1,2,3) = 5 + (5+10) + (5+10+3) = 38$

$(1,3,2) = 5 + (5+3) + (5+3+10) = 31$

$(2,1,3) = 10 + (10+5) + (10+5+3) = 43$

$(2,3,1) = 10 + (10+3) + (10+3+5) = 41$

$(3,1,2) = 3 + (3+5) + (3+5+10) = 29$

$(3,2,1) = 3 + (3+10) + (3+10+5) = 34$

eg. $n = 10$

	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}
	10	20	45	70	1	3	7	54	23	67
Asending	1	3	7	10	20	23	45	54	67	70
mod.	1	2	0	1	2	0	1	2	0	1

Tape 0	7, 23, 67	$7 + (7+23) + (7+23+67) = 134$
Tape 1	1, 10, 45, 70	$1 + (1+10) + (1+10+45) + (1+10+45+10) = 194$
Tape 2	3, 20, 54	$3 + (3+20) + (3+20+54) = 103$

$\therefore$ Mean retrival time $= \frac{134 + 194 + 103}{3} = \frac{431}{3} = \underline{143.6}$

- There are n program that are to be stored on computer tape of length l associated with each program i is a length l_i. We assume that whenever a program is to be retrive from the stake to tape is initially position at the front. Hence, if the programs are stored in order,

$I = i_1, i_2, i_3 \cdots i_n$

The time t_j needed to retrive program l_j is proportional to $\Sigma_{1 \le k \le j} l_{ik}$

- If all programs are retrive equally offened then expented mean retrival time (MRT) is

$$(1/n) \Sigma_{1 \le j \le n} t_j$$

- In a optimal storage on tape problem we are require to find a permutation for n programs so that when they are stored on the tape in this order the mean MRT is minimized.
- Minimizing the MRT is equalient to minimizing

$$d(I) = \Sigma_{1 \le j \le n} \Sigma_{1 \le k \le j} l_{ik}$$

- Algorithm:

```
Store (n, m)
// n is no. of programs
// m is no. of tapes
{
  j = 0 ; // next tape to store on
for i = 1 to n do
{
write ("Append program" i to permutation
 for tape", j);
  j = (j+1) mod m;
}
}
```

Q. Find an optimal placement for 13 programs on 3 tapes $t_0, t_1 \& t_2$ where programs are of lengths

l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}	l_{11}	l_{12}	l_{13}
12	5	8	32	7	5	18	26	4	3	11	10	6

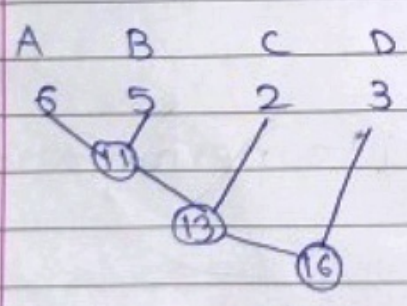
	1	2	3	4	5	6	7	8	9	10	11	12	13
Assending	3	4	5	5	6	7	8	10	11	12	18	26	32
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Mod	1	2	0	1	2	0	1	2	0	1	2	0	1

Tape 0	5, 7, 11, 26	(5)+(5+7)+(5+7+11)+(5+7+11+26) = 89
Tape 1	3, 5, 8, 12, 32	3+(3+5)+(3+5+8)+(3+5+8+12)+(3+5+8+12+32) = 115
Tape 2	4, 6, 10, 18	4+(4+6)+(4+6+10)+(4+6+10+18) = 72

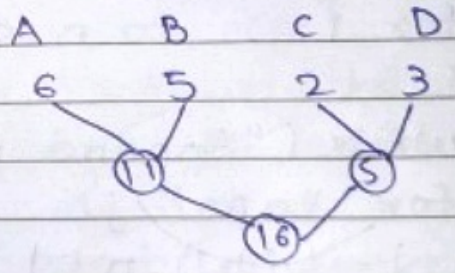
Tape 0 = 89
Tape 1 = 115
Tape 2 = 72

* Optimal merge pattern

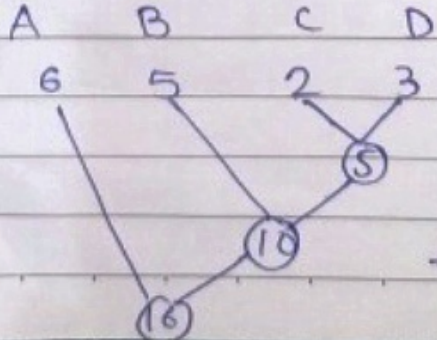
A	B	C	D
6	5	2	3



= 40



= 32



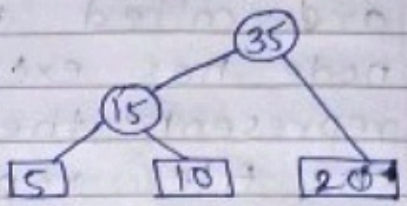
= 81

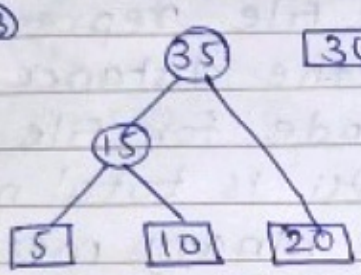
List	x_1	x_2	x_3	x_4	x_5
Sizes	20	30	10	5	30
Asc.	5	10	20	30	30
	x_4	x_3	x_1	x_2	x_5

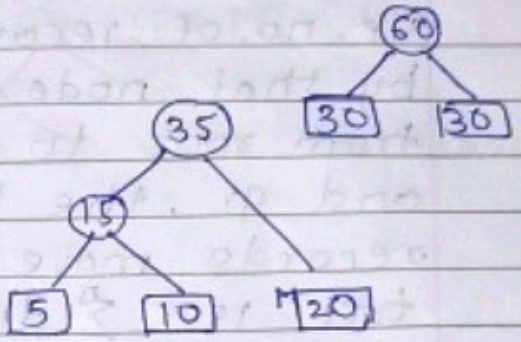
Steps

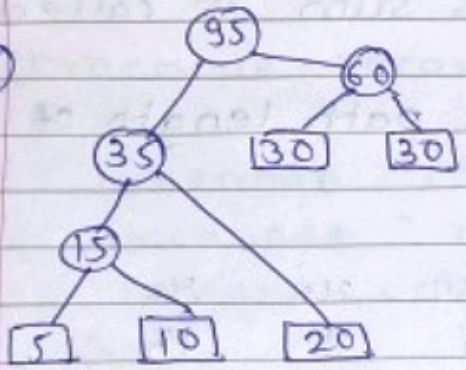
(1) 

20, 30, 30

(2) 

(3) 

(4) 

(5) 

= 205

$5 \times 3 + 10 \times 3 + 20 \times 2 + 30 \times 2 + 30 \times 2 = 205$

- We have to sorted files containing n & m ~~n &~~ records respectively. We have to merge them together. When more than two sorted files ~~are~~ to merge together the merge can be accomplish by repeadly merging sorted file in pairs. Thus if file x_1, x_2, x_3 & x_4 are to be merge we put ① merge x_1 & x_2 to get file by 2. Then we could merge ~~y1~~

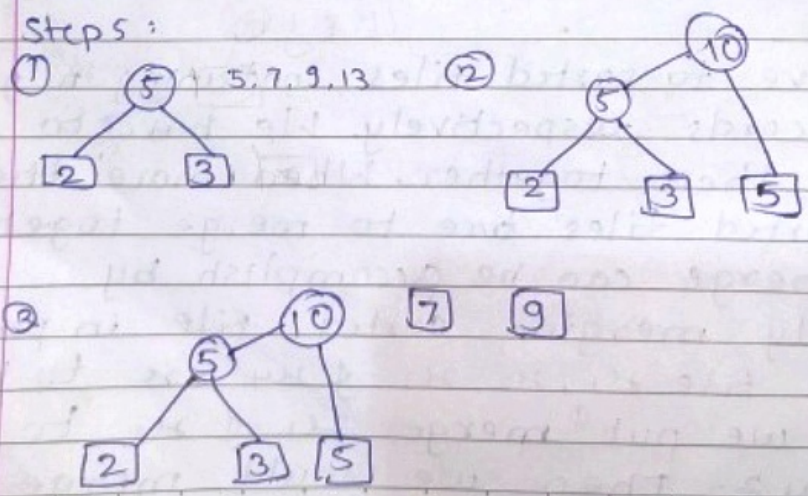
y1 & x3 to get y2 so on the leaf nodes are drawn as squares. and represent the given files. The nodes are called external nodes. The remaining nodes are drawn as. circles are called internal nodes. Each internal node has exctaly two children and it represents the file obtain by merging the files represented by its two children The number in each node is the length i.e. no. of records of the file represented by that node. if di is the distance from root to external node for file xi and qi, the length of xi is total no. of records move from its binary merge tree is $\sum_{i=1}^{n} d_i q_i$ This sum is called the weighted external path length of the tree.

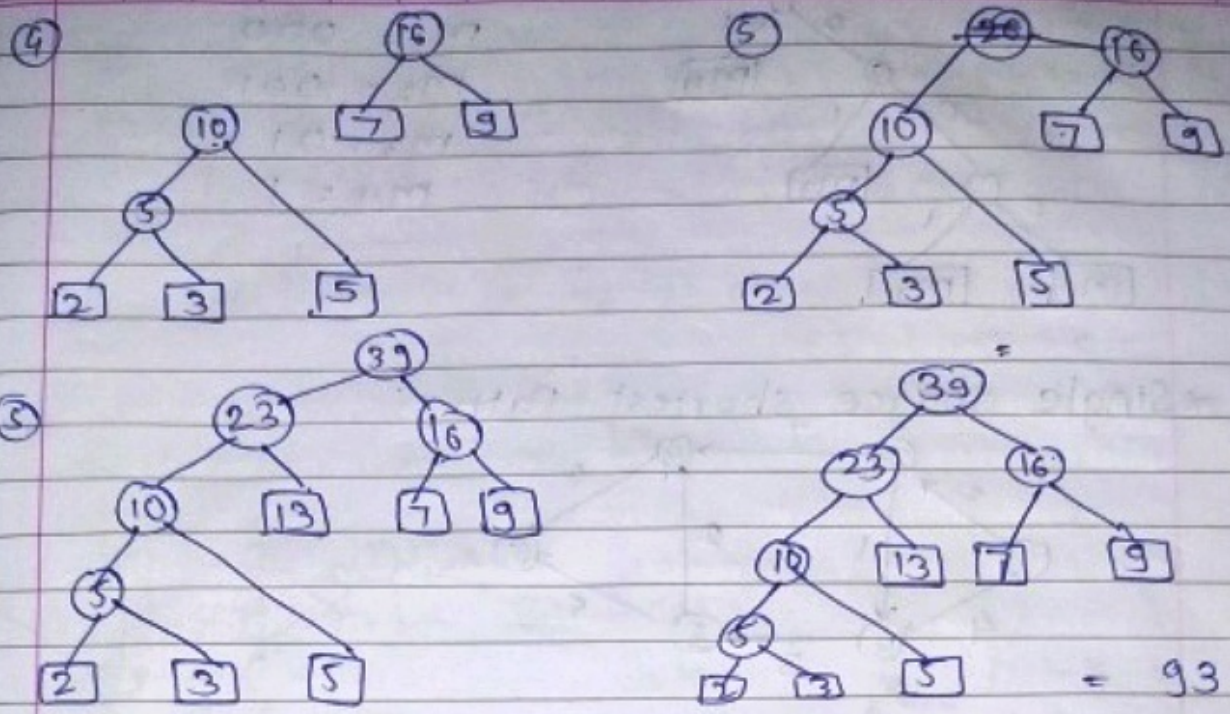
eg. list x_1 x_2 x_3 x_4 x_5 x_6

sizes 2 3 5 7 9 13

Asc

Steps:

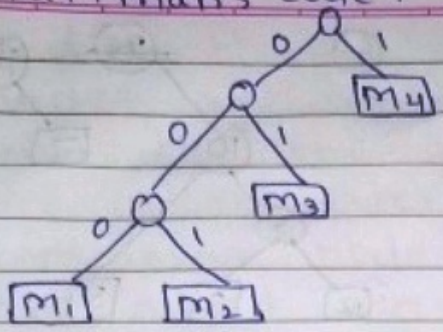




- Algorithm:

```
treenode = record
{
   treenode * lchild;
   treenode * rchild;
   integer weight;
};
Algorithm Tree(n)
{
   for i = 1 to n-1 do
   {
      pt = new treenode;
      (pt→lchild) = least(list)
      (pt→rchild) = least(list)
      (pt→weight) = ((pt→lchild)→weight)
                  + ((pt→rchild)→weight)
      insert(list, pt);
   }
   return least(list);
}
```

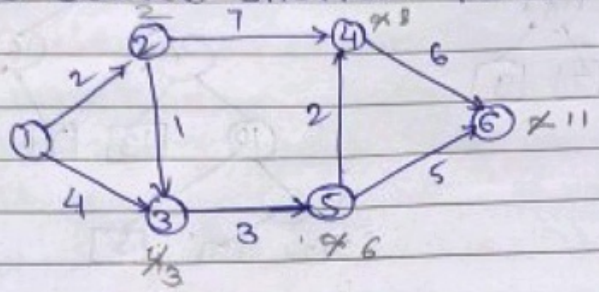
Huffman's code:

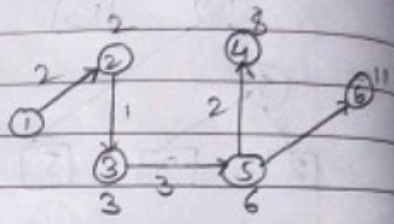


$m_1 = 000$
$m_2 = 001$
$m_3 = 01$
$m_4 = 1$

Q.2

Single source shortest Path:





① $d(2) + c(2,3) < d(3)$
$2 + 1 < 4$
$3 < 4$
$d(v) + c(v,w) < d[w]$
$d[w] = d(v) + c(v,w)$

② $d(3) + c(3,5) < d(5)$
$3 + 3 < \infty$
$6 < \infty$

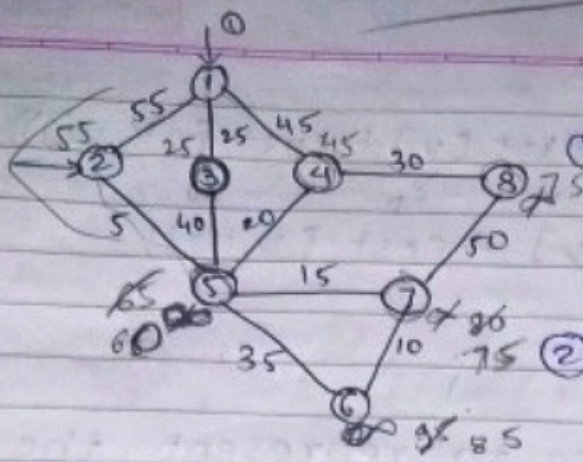
③ $d(5) + c(5,4) < d(4)$
$6 + 2 < d(4)$
$8 < \infty$

④ $d(4) + c(4,6) < d(6)$
$8 + 6 < \infty$
$11 < \infty$

$d(u) + c(u,w) < d(w)$

Page No. | Date: 22-8-24 | YOUVA

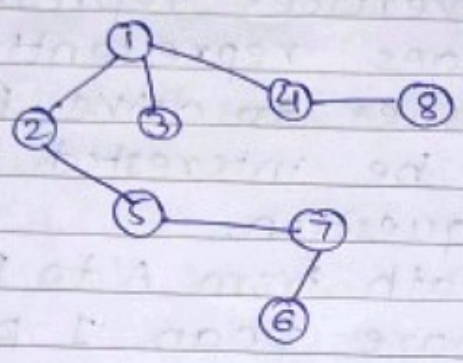
Q.2



① $d(3) + c(3,5) < d(5)$
$25 + 40 < \infty$
$65 < \infty$

② $d(5) + c(5,7) < d(7)$
$65 + 15 < \infty$
$80 < \infty$

③ $d(7) + c(6,7) < d(6)$



- Algorithm:

```
Shortestpaths(v, cost, dist, n)
  dist[j] - set to length of shortest path
        from v to j
  dist[v] = 0
{
  for i = 1 to n do
  {
    s[i] = false, dist[i] = cost[v,j];
  }
  s[v] = true ; dist[v] = 0.0;
  for num = 2 to n do
  {
    s[v] = true
  for( each w adjust to u with-
        s[w] = false)
```

```
do
if (dist[w] > dist[v] + cost[v,w]))
then
d[w] = dist[v] + cost[v,w];
}
}
```

- Graphs can be use to represent the highway structure of a state or country with vertices representing cities and edges representing highway. A voteriest wishes to drive from city A to B would be interested in answers to following question.
  (1) Is there a path from A to B
  (2) Is there more than 1 path from A to B which is the smallest path.
- The problem define is called shortest path problem. The length of path is defined by ~~some~~ sum of the bits of the edges on path
- The starting vertex is refered as source, last vertex is refered as destination.
- Graphs are ~~bigra~~ di graphs in problem we consider we are given directed graph $G = V, E$.
- A waiting function cost for edge of ~~cost~~ graph and ~~co~~ source vertex v_0
- Problem is to determine shortest path from V_0 to all remaining vertices of graph
- It is assume that all the weights are positive.

- Shortest path bet^n Vo & some other is an ordering amount of among su... of edges.

3. Backtracking

Algorithm Backtrack(k)

Find all possible sol^n & use the one you want

- Backtracking represents one of most ... technique many problems which deal with searching set of solution or which ask for optimal solution satisfying some constraint can be solved using backtracking formulation.
- In many application of backtrack method the desire solution is express as n tuple x_i to x_n where x_i are choosen for some finite set S_i. The problem to be solved for finding one vector that maximize or minimize a criterian function $P(x_i, \ldots x_n)$. Sometimes it all vectors that satisfy P.
- Suppose m_i is size of set S_i then their are $m = m_1, m_2, \ldots m_n$. n tuple that are possible candidates for satisfying f^n p. The two-force approach could be form n tuples evaluate each one with P. and save those which yet optimum.
- The backtrack algorithm has ability to answer with far fever with n trials.

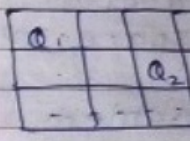
- The basic idea is to build solution vector one component at a time and to use modified criterian function called bounding function to test whether the vector form has any chance of success

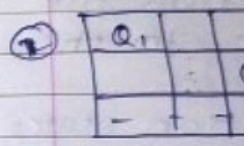
```
Backtrack(k)
//x[1], x[2] .... x[k-1] of solution vector
{
for (each x[k] ∈ T (x[1],... x[k-1]) do
 {
   If (Bk (x[1], x[2]... x[k] ≠ 0) then
   {
    if (x[1], x[2]. ... x[k] is path to
    answer node)
    then write x[1-k];
    if (k<n)
    then
      Backtracking (k+1)
   }
 }
}
```

- let T of x_1, x_1 be set of all possible values for x_{i+1} such that $x_1, x_2, \ldots x_{i+1}$ is also a path to problem state
- If $(x[1] \ldots x[n]) =$ Null, we assume the existance of bounding function $B(i+1)$ such that if B of $f(i+1) = [x_1, x_2 . x_{i+1}]$ is false for a path $(x_1, x_2 \ldots x_{i+1})$ from root node to appr problem state
- Then path cannot be extended to rich an answer node. Thus the cadidates for

position 1
are those
by T and

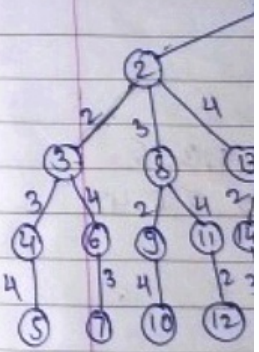
IMP * N queens
① Q_1 Q_2 Q_3



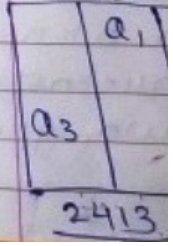


* State sp

Imp Imp



Mirror

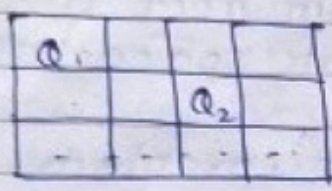


2413

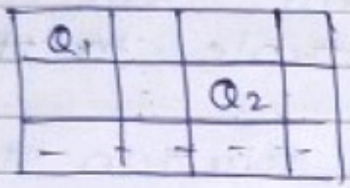
position $1+1$ of the solution vector $x_1 \ldots x_n$ are those values which are generated by T and satisfy $B(1+1)$

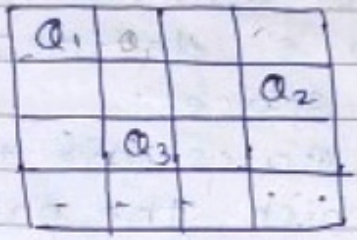
Imp * N queens problem:

① $Q_1\ Q_2\ Q_3\ Q_4$



row
column
diagonal

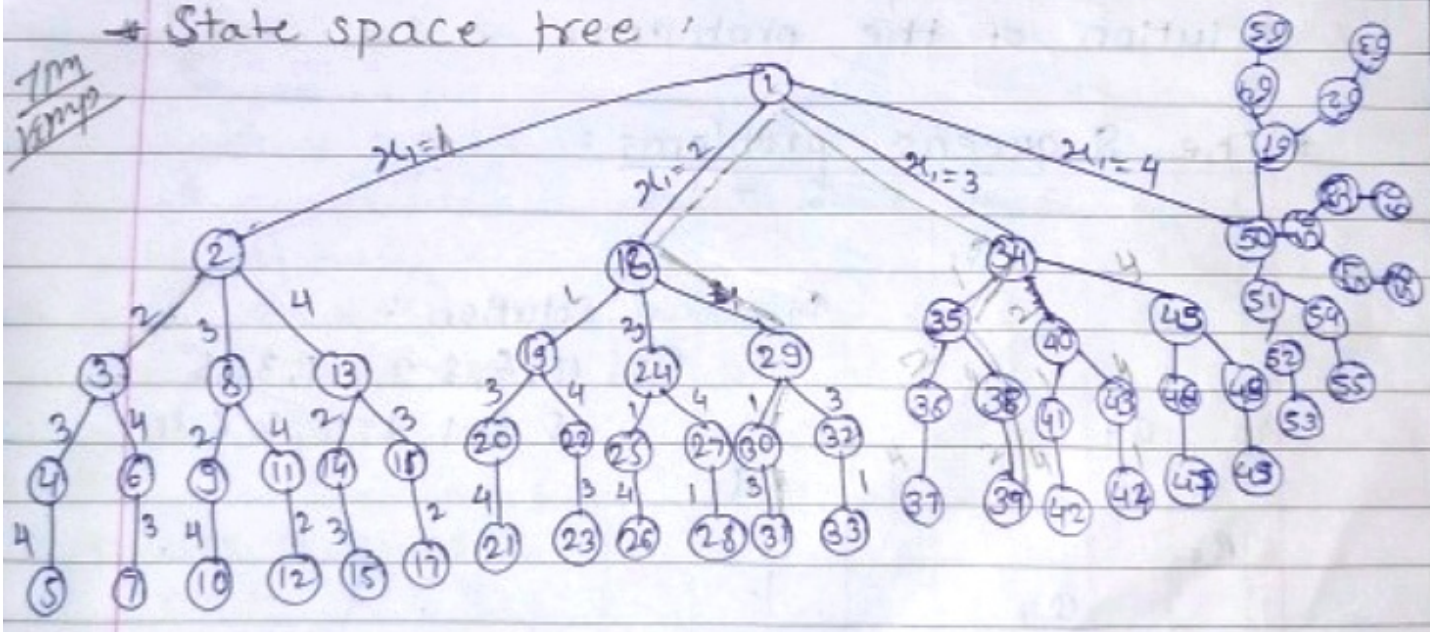
① 

② 

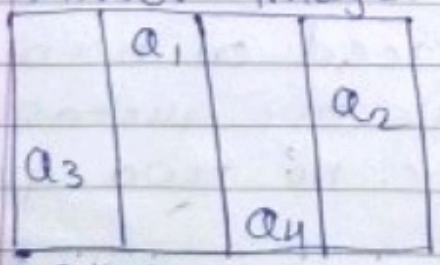
③ 

* State space tree:

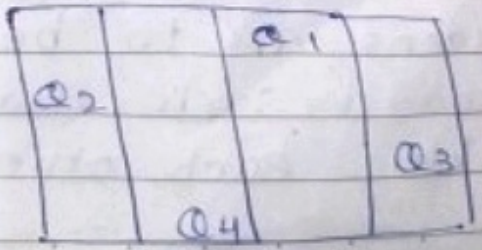
Im Imp



Mirror image:



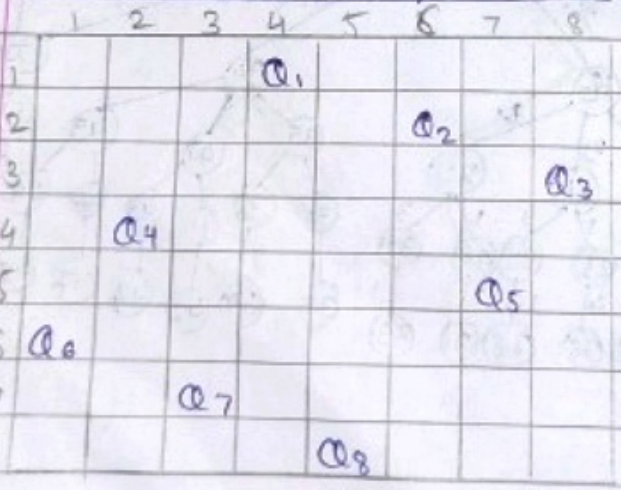
2413



3142

- Backtracking algorithm determine problem solution by systematically searching solution space for the given problem instant these search is facilitated by using tree organization for solution space.
- Each node in these tree determines problem state all paths from root to, othernodes define the state space of the problem.
- Solution states are those problem states 's' for which the path, from the root to 's' defines a tuple in the solution space.
- Answer states are those solution states s for which the path from root to 's' defines a tuple i.e. member of the set of solution of the problem.

* The 8 queens problems:



Solution:
4, 6, 8, 2, 7, 1, 3, 5
5, 3, 1, 7, 2, 8, 6, 4

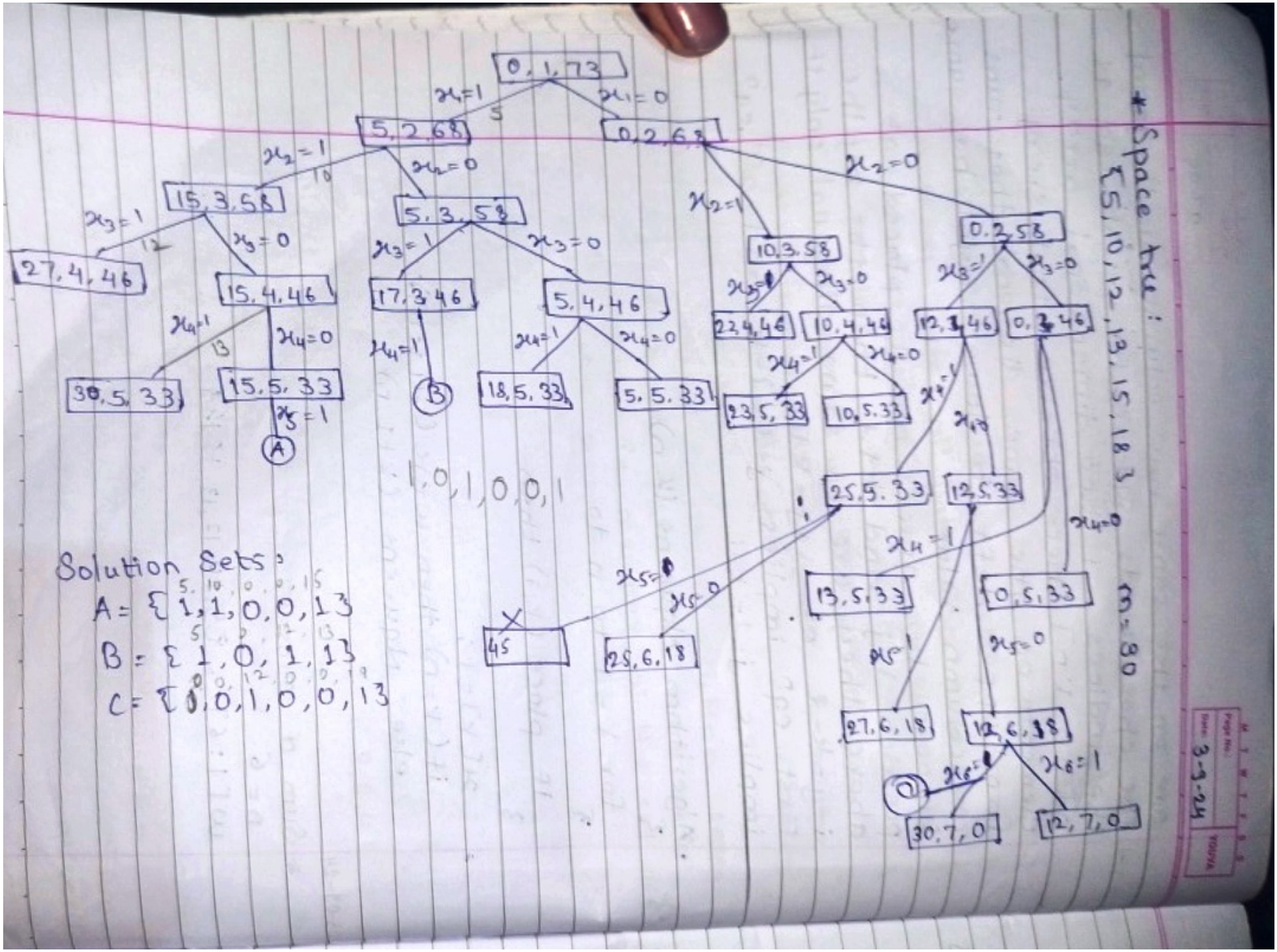
- N queens are to be placed on n × n cheesboard such that no two queens are attacking each other i.e. no two queens

are on the same row, column or diagonal
- The chessboards square be numbered as the indices of two dimensional array. a[1:n, 1:n] then observe that every element on the same diagonal that runs from upper left to lower right has same row, column value.
- Suppose two queens are to placed at position i,j and k,l then by the above they are on same diagonal only if i-j = k-l or i+j = k+l
- First eqn implies j-l = i-k and 2nd eqn implies j-l = k-i

- Algorithm Nqueens (K, n)
{
for i = 1 to n do
{
if place (k, i) then
}
x[k] = i;
if (k=n) then write (x[1:n]);
else Nqueens (k+1, n)

09-24

* Sum of subset:
n = 6
w[1:6] = {5, 10, 12, 13, 15, 18}


0, 1, 73
x1=1
x1=0
5
5, 2, 68
0, 2, 63
x2=1
x2=0
10
15, 3, 58
5, 3, 58
10, 3, 58
0, 2, 58
x3=1
x3=0
12
27, 4, 46
15, 4, 46
17, 3, 46
5, 4, 46
23, 4, 46
10, 4, 46
12, 3, 46
0, 3, 46
x4=1
x4=0
13
30, 5, 33
15, 5, 33
18, 5, 33
5, 5, 33
23, 5, 33
10, 5, 33
25, 5, 33
12, 5, 33
0, 5, 33
x5=1
x5=0
A
B
45
25, 6, 18
13, 6, 18
27, 6, 18
12, 6, 18
x6=1
x6=0
C
30, 7, 0
12, 7, 0
1, 0, 1, 0, 0, 1
Solution Sets:
A = { 1, 1, 0, 0, 1 }
B = { 1, 0, 1, 1 }
C = { 0, 0, 1, 0, 0, 1 }
* Space tree:
{ 5, 10, 12, 13, 15, 18 }
3-3-24

- Suppose, we are given n distinct positive numbers usually called dates. we desire to find all combination of these numbers whose sum are m. This is called sum of subset problem.
- The child of any node are easily generated for node at level i, the left child corresponds to $x_i = 1$ & the right child $x_i = 0$. The simple choice for bounding function is true. If

① Sum of weights + included so far + weight for next object

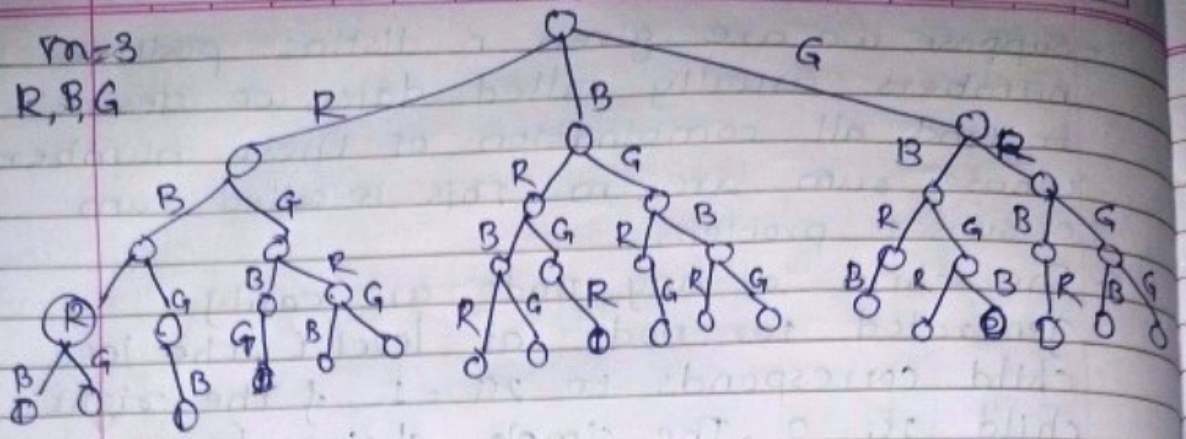
$$\sum_{i=1}^{k} w_i x_i + w_{k+1} \leq m$$

②
$$\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \geq m$$

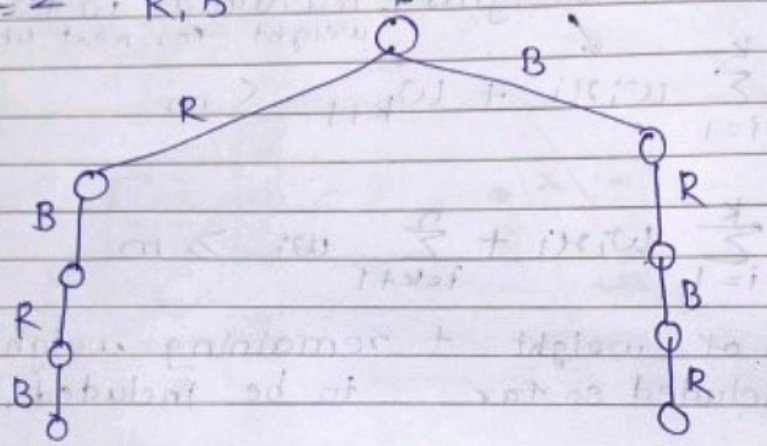
Sum of weight included so far + remaining weight to be included.

```
*SumofSub (s,k,r)
{
Generate left child
s+w[k] ≤m since Bk-1 is true.
   x[k]=1
if(s+w[k]=m) then write (x[1:k]);
else if ( s+w[k]+w[k+1]≤m)
then sumofsub (s+w[k], k+1, r-w[k])
if (s+r-w[k]≥ m) & (s+w[k+1]≤m)
then
{    x[k]=0;
   Sum of subset (s, k+1, r-w[k])
}
}
```

m=3
R, B, G



② m = 2 . R, B




```
Algorithm mcoloring(k)
{
 repeat
{
  NextValue(k);
  if (x[k]=0) then return;
  if (k=n) then
  write (x[1:n]);
  else mcoloring(k+1);
}
   until (false);
}
```