UNIT - V

Branch & Bound, NP-Hard & NP-Complete Problems

⇒ Branch & Bound:-

→ To systematically search a solution space.

→ Avoid sub trees which not contain answer node.

→ a) Branching function:- (growth) DFS/BFS based on bounding function.

Differ from Backtracking in 2 manners

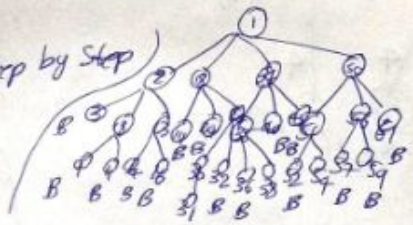
b) Bounding function:- (constraints) goes far beyond the feasibility test as a mean to prune efficiently the search tree.

nodes
- leaf
- E live
- parent

BFS – FIFO (queue)
DFS – LIFO (Stack)

Explored fully the node.

Live → when children's node attached → E-node



Peculiar → step by step

4 queen's problem
↓
no optimality, only pattern finding.

answer node

⇒ Least Cost (LC) Search:-

ranking method

rank(i) :- optimal, least answer giving path

Answer node
↑
x
↑
root

⇒ C(x)

⇒ C(x) = f(h(x)) + g(x)

- C(x) → ranking function
- f(h(x)) → approximation function
- g(x) → cost of reaching x from root
- Additional effort for reaching answer node from the x.

Cost → height/length of tree
→ computational complexity

⇒ P, NP, NP-complete problems :-

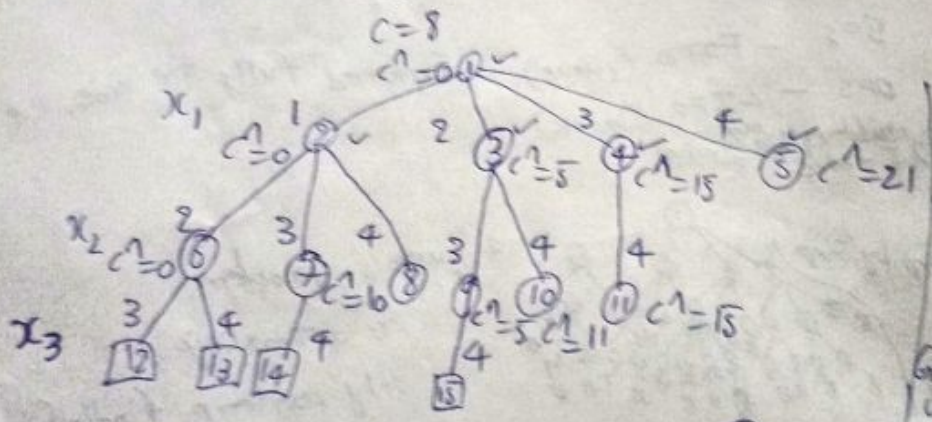
P, NP, NP-complete

Could solve in polynomial time

Can't solve in non-deterministic polynomial time } (Time Expressed in polynomial)

optimization,

⇒ LC Branch & Bound :-



upper = ∞,
upper = 14 ; in FIFO
4, 5 killed
> c
6, 7, 8 generated
upper =
Node 6 (node)
9, generated
upper = 8
10, killed

Node 9 min. cost answer node.

Variable tuple Size formulation

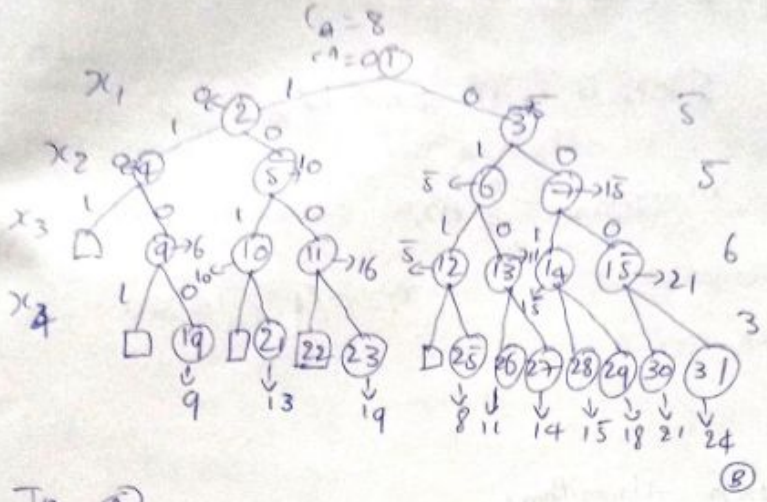
⇒ FIFO Branch & Bound :-

general algorithm (or systematic method) for finding an optimal solution to various optimization problems, especially in discrete & combinatorial optimization.

Scanned By Camera Scanner

the child nodes are explored in first in first out manner.

fixed tuple size formation



In, (A)

$u(2) = 19, u(3) = 14, u(4) = 18, u(\bar{5}) = 21$

$u(2) =$ penalties sum of jobs.

next E-node

upper = 14

4, $\bar{5}$ killed > upper. (Bounded)

6 7 8

$u(6) = 9$

upper = 9

$c\hat{}(7) = 10 >$ upper, killed.

Node 8 – infeasible.

3 → E-node

9 10

$u(9) \Rightarrow$ upper = 8

$c\hat{}(10) = 11 >$ upper, node is killed

minimum cost = 8

minimum cost answer node is 9

⇒ 0/1 Knapsack problem

$\sum P_i x_i$ is $\uparrow$, iff. $-\sum P_i x_i$ is $\downarrow$

minimize $-\sum_{i=1}^{n} P_i x_i$ Subject to $\sum_{i=1}^{n} w_i x_i \leq m$

$x_i = 0/1$, $1 \leq i \leq n$

⇒

LC-Search Algorithm :-

```
listnode = record
{
    listnode *next, *parent;
    float cost;
};
Algorithm LCSearch(t)
{
    if *t is an answer node then output *t & return;
E := t;
Initialize the list of live nodes to be empty.
repeat
{ for each child x of E do
  { if x is an answer node then output
      the path from x to t & return;
```

Add(x);
(x->
}
if the
}
}
⇒ Funct

```
Add(x);
(x→ parent) := E;
}
if there are no more live nodes then
    { write ("No answer node");
      return;
    }
E := Least();
} until (false);
}
```

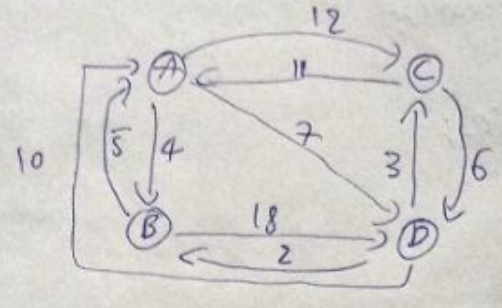
⇒ Function ~~u(.)~~ Algorithm for Knapsack problem:-

```
                    (current profit, current weight, item removed index, size)
Algorithm   UBound (cp, cw, k, m)
{
    b := cp, c := cw;
    for i := k+1 to n do
    {
       if (c+w[i] ≤ m) then
       {
          c := c + w[i];
          b := b - p[i];
       }
    }
    return b;
}
```

⇒ Travelling sales person problem:

Using a least order,

observe the nodes in which order they are generated. Calculate the cost of the node simultaneously. If cost(any node) > upper bound, we will remove that node.



①

	A	B	C	D
A	∞	4	12	7
B	5	∞	∞	18
C	11	∞	∞	6
D	10	2	3	∞

Row Reduction (every Row at least one 0)

A − 4	∞	0	8	3
B − 5	0	∞	∞	13
C − 6	5	∞	∞	0
D − 2	8	0	1	∞

Sum = 18

(cost(1) = 18)

column reduction (every column at least one 0)

∞	0	7	3
0	∞	∞	13
5	∞	∞	0
8	0	0	∞

— (A)

② Set, A, B to ∞

to (A)

∞	∞	∞	∞
∞	∞	∞	13
5	∞	∞	0
8	∞	0	∞

If 0 → ∞

(R.R)

∞	∞	∞	∞
∞	∞	∞	0
5	∞	∞	0
8	∞	0	∞

+13

C.R.

③ Set,

m(tree) = 7

④ Se

m

C.R.

∞	∞	∞	∞
∞	∞	∞	0
0	∞	∞	0
3	∞	0	∞

18

Sum = 18 + Sum(1)

= 36

Cost(2) = 36

A → B

③ Set, A, C to ∞ to ④

m(A,C) = 7

∞	∞	∞	∞
0	∞	∞	13
∞	∞	∞	0
8	0	∞	∞

R.R. Same

C.R. Same

Cost(3) = Cost(1) + 7

= 18 + 7 = 25 ✓

A → C

④ Set, A,D to ∞ to ④

m(A,D) = 3

∞	∞	∞	∞
0	∞	∞	∞
5	∞	∞	∞
∞	0	0	∞

R.R.

C-5

∞	∞	∞	∞
0	∞	∞	∞
0	∞	∞	∞
∞	0	0	∞

C.R.

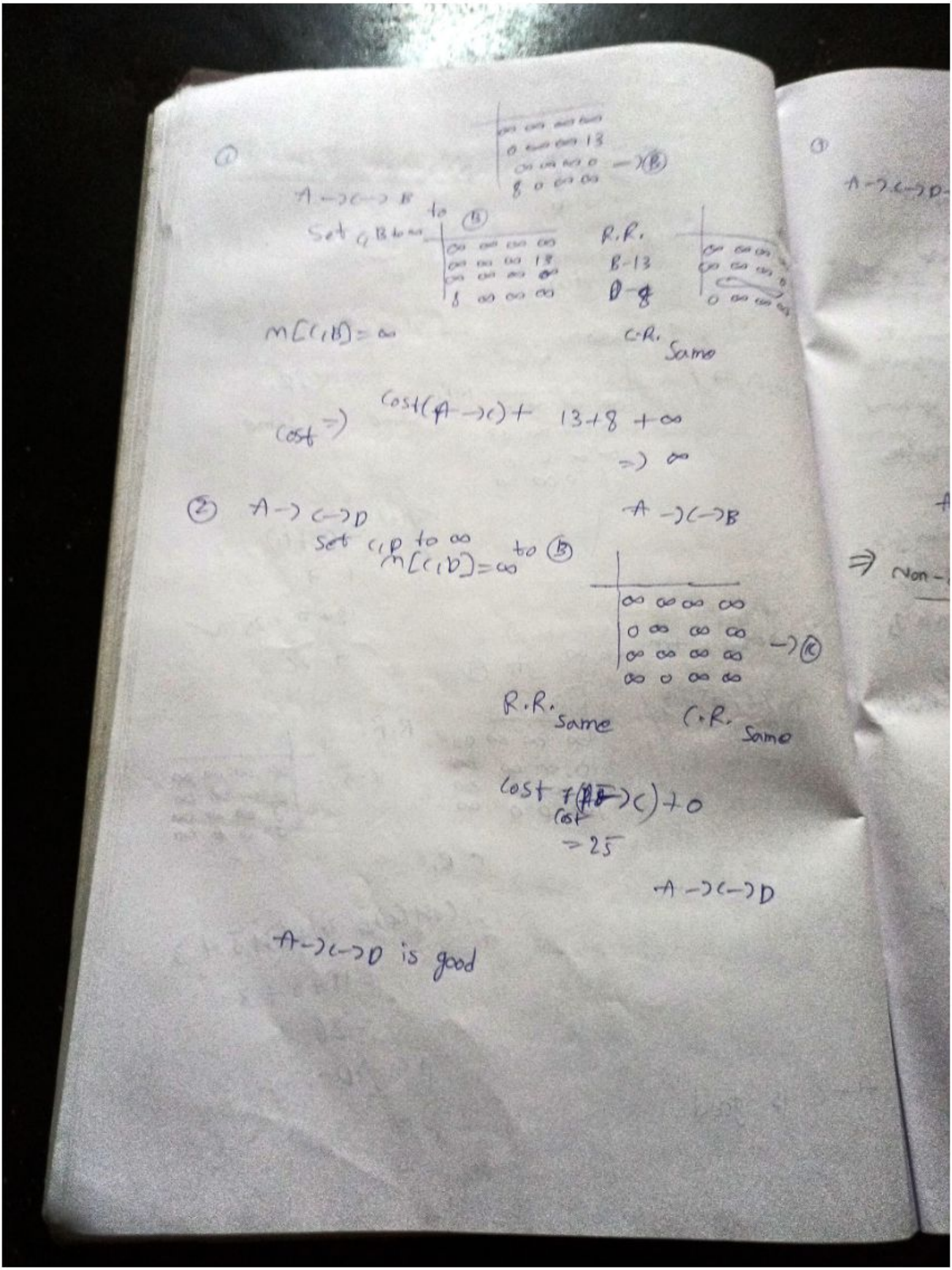
Cost(4) = Cost(1) + 5 + 3

= 18 + 5 + 3

= 26

A → D

A → C is good



Scanned By Camera Scanner

①

A → C → D → B

Set, D|B to ∞

∞	∞	∞	∞
∞	∞	∞	∞
∞	∞	∞	∞
∞	∞	∞	∞

R.R. Same / C.R. Same

Cost = cost(A → C → D) + 0 + 0
= 25

A → C → D → B → A = 25

⇒ Non-deterministic Algorithms:-

It exhibits different behaviours on different runs for same input.

Choice(S) - 1 element of S. - O(1)

Failure() - unsuccessful completion - O(1)

Success() - successful completion - O(1)

It terminates unsuccessfuly iff ~~there~~ there exists no set of choices leading to a success signal.

=> NP-hard & NP-complete:-

NP-hard -> if Algorithm solving to 1 then solving NP problem "at least as hard as any NP problem"

NP-complete -> if problem is in NP & all NP problems polynomial time reducible to it,

NP-hard	NP-complete
By NP-complete problem	By non-deterministic Algorithm
no NP	both NP & NP-hard
no Decision problem	Decision problem.
Ex:- Halting problem, Vertex cover problem etc.	Ex:- Hamiltonian cycle in graph/not, Boolean formula is satisfiable/not, circuit satisfiability problem etc.

=> Cook's theorem:-

It proves that Satisfiability is NP-complete by reducing all non-deterministic Turing machines to SAT.

SAT => If assinged values to equation is true, Satisfiable

if $p \wedge q$ ↗ $p=1, q=1$ else $p \wedge \sim p$

3SAT

=>

{ ϕ | ϕ is satisfiable

3-CNF formula}

$(x \vee y \vee z) \wedge (\bar{x} \vee \bar{y} \vee z)$ ← Conjuctive Normal form

↓ clause

Every clause exactly ↓ OR 3 literals

Theorem:-

SAT is NP- complete

Steps

a) NDTm - Non-deterministic Time machine,

Execution (polynomial time) → well formed formulas

such that formula satisfie if machine accept input.

b) Show Sum of lengths of formulae is polynomial in the size of problem.

c) Steps

→ NP-Hard (L) - can polynomial reduce NP to L

→ NP-complete - $L \in NP$

→ $L \in NP$ → NDTm for L runs in polynomial time.

→ NDTm → only model we have for NP problem.

→ SAT $\in$ NP.

→ ∴ we can polynomially reduce ξ an arbitrary Polynomial NDTM to SAT. it means we have proven SAT is NP complete.

⇒ NP-scheduling problems:- optimal schedule for set of Jobs is NP-complete even in 2 restricted cases:- time.
a) All Jobs require 1 unit. b) All Jobs require 1/2 time units, ξ there are only 2 processors.

resolving,

$P_1, P_2, P_3 \dots P_n$ → m machines

$T_1, T_2, T_3 \dots T_k$ → processing time.

Schedule s/ for each Job J_k.

a) processor which Job J_k is carried out.

b) Time period during which Job J_k carried out processing.

only 1 Job, f_k – finish time for Job J_k.

mean finish time

MFT ⇒ $\frac{1}{n} \sum_{k=1}^{n} f_k$

weight mean finish time

WMFT ⇒ $\frac{1}{n} \sum_{k=1}^{n} w_k \times f_k$

Finish time of Schedule =>

$$FT(S) = \max_{1 \le k \le m} \{T_k\}$$

Schedule S

Non-preemptive
Stoost to finish Job
on Same processon

Preemptive
Stoost to finish Job
on different processon