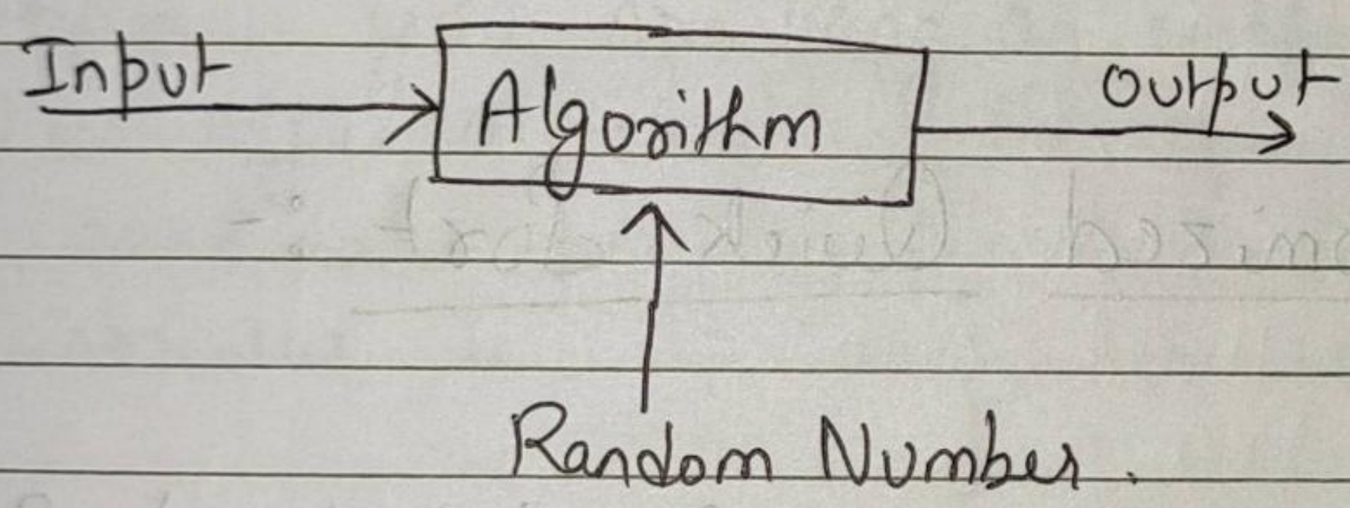
Unit - 5

Randomized Algorithm :-

It is an algorithm that employ a degree of randomness as part of its logic. It depends on random number for its operation.

It use uniform random bit (pseudo random number) as input to guide its output. It is also called as probabilistic algorithm.



Types :-

1) Las Vegas algorithm :- It always generate the same output for the same input. Output is always correct. Ex → Randomized Quick Sort

2) Monte Carlo algorithm :- Output may vary for same input. Output may be incorrect with some probability. Runtime is deterministic
Eg. → Randomized mincut algorithm.

Scanned by TapScanner

Advantage of Randomized Algorithm :-

1) usually simple & easy to implement & more efficient than the deterministic algorithm.

2) Fast with very high probability & produces optimum output with very high probability.

3) Main advantage is that no input can reliably produce worst case results because the algorithm runs differently each time.

Randomized Quick Sort :-

```
Randomized_Quick_Sort (A, p, r) {
    if (p<r) {
        i ← random (p, r);
        Swap (A[i], A[r]);
        q ← partition (A, p, r);
        Randomized_Quick_Sort (A, p, q-1);
        Randomized_Quick_Sort (A, q+1, r);
    }
}
```

5	4	8	9	1	2	11

→ random (pivot)

Swap [A[i], A[r]]

5	4	11	9	1	2	8

→ pivot

Scanned by TapScanner

Benifit :- It decreases worst case time complexity of Quick Sort.

Using Normal → | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

→ pivot

Using Randomized → | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

↓ pivot

Hiring Problem :-

Suppose you are using an employment agency to hire an office assistant. The agency sends you one candidate per day : interview and decide. Cost to interview is C_i and hire is C_h. $C_h > C_i$.

```
Hire-Assistant (n) {
    best = 0
    for i = 1 to n
        interview candidate i.
        if candidate i is better than best
            best = i
            hire candidate
```

Total cost $= O(C_i \cdot n + C_h \cdot m)$.

Best Case $= O(C_i \cdot n)$ → (No hire)

Worst Case $= O(C_i n + C_h \cdot n) = O(C_h n)$

↳ (All hired)

Scanned by TapScanner

Randomise_Hire_Assistant(n)

1. Randomly permute the list of Canditate
2. best = 0.
3. for i = 1 to N
4. if candidate i is better that best
5. best = i
6. Hire Candidat

$T(n) = O(C_h \log n)$

String Matching

Text : a a a a a b
Pattern : a a b.

a a b
$\underbrace{1+1+2}_{h(p) \to \text{hash function}} = \underbrace{4}_{\text{hash code}}$

a → 1	g → 7
b → 2	h → 8
c → 3	i → 9
d → 4	j → 10
e → 5	~~k → 11~~
f → 6	~~l → 12~~

a a a a a b.

1 + 1 + 1 = 3 (not matching)

1 + 1 + 1 = 3 (not matching)

1 + 1 + 1 = 3 (not matching)

1 + 1 + 2 = 4 (matching)

↓

Check the alphabets

Scanned by TapScanner

Problem in this :-

Pattern = dba (4+2+1=7)

c c a c c a a e d b a

3+3+1=7 ← Spurious hit (Character not matching but hash code matching)

3+1+3=7 ← Spurious hit

1+3+3=7 ← Spurious hit

3+3+1=7 ← Spurious hit

3+1+1=5

1+1+5=7 ← Spurious hit

1+5+4=10

5+4+2=11

4+2+1 ← valid hit

character as well as hash code is matching

* Spurious hit is very large. This is problematic

Pattern : dba

m = 3.

$$= 4\times10^{2} + 2\times10^{1} + 1\times10^{0}$$

(m-1, m-2, m-3)

$= 421$

base 10 is taken as we choose a – j (10 character for code).

If we choose all alphabet it will be 26.

c c a c c c a a e d b a

421 → match

331

313

133

333

331

311

115

154

421

now match character.

Scanned by TapScanner

Worst case time Complexity = $O(nm)$

Average case time Complexity = $O(n-m+1)$.

Q Text: c c a c c a a e d b a

Pattern: dba

a → 1 f → 6
b → 2 g → 7
c → 3 h → 8
d → 4 i → 9
e → 5 j → 10

Sol → Pattern : dba

$m = 3$ $\quad 4\times10^2 + 2\times10^1 + 1\times10^0$

$= 421$

cca :- $3\times10^2 + 3\times10^1 + 1\times10^0$

$= 331$ (No matching with hash code)

cac : $3\times10^2 + 1\times10^1 + 3\times10^0$

$= 313$ (No Maching)

acc : $1\times10^2 + 3\times10^1 + 3\times10^0$

$= 133$ (No matching)

cca : $3\times10^2 + 3\times10^1 + 3\times10^0$

$= 331$ (No matching)

caa : $3\times10^2 + 1\times10^1 + 1\times10^0$

311 - (No matching)

aae : $1\times10^2 + 1\times10^1 + 5\times10^0$

$= 115$ (No matching)

aed : $1\times10^2 + 5\times10^1 + 4\times10^0$

$= 154$ (No matching)

edb : $5\times10^2 + 4\times10^1 + 2\times10^0$

542 (No matching)

dba : $4\times10^2 + 2\times10^1 + 1\times10^0$

$= 421$ (Matching)

Matched string pattern found.

Scanned by TapScanner

Pattern Matching :-

Text :- 3 1 2 3 4 8 6 2

Pattern :- 2 3 4

$q = 13$.

$h^p = p \bmod q$

$= 234 \% 13$

$= 0$

Now, we will match digit whenever 0 comes in h^T.

312 → 312 mod 13 = 0 (Spurious hit)

123 → 123 mod 13 = 6

234 → 234 mod 13 = 0 (valid hit)

348 → 348 mod 13 = 10

486 → 486 mod 13 = 5

862 → 862 mod 13 = 4

Example to Practice :-

Q $T = 31415926535897 93$

$p = 826$

$q = 11$

Scanned by TapScanner

Algorithm

```
RabinKarp (T, P)
    n = T.length
    m = P.length
    h^P = Hash (P[- - - ])
    h^T = Hash (T[- - - ])
    For s = 0 to n-m
        if (h^P == h^T)
            if (P[0 - m-1] = T[s+0 + - - s+m-1])
                Print ("Pattern found with shift", s)
        if (s < n-m)
            h^T = Hash (T(s+1 - - s+m))
```

Shift

Approximation Algorithm :-

It is a way of dealing with NP-completeness for optimization problem. The Goal of approximation algorithm is to come as close as possible to an optimal solution in polynomial time.

① Maximization ② Minimization

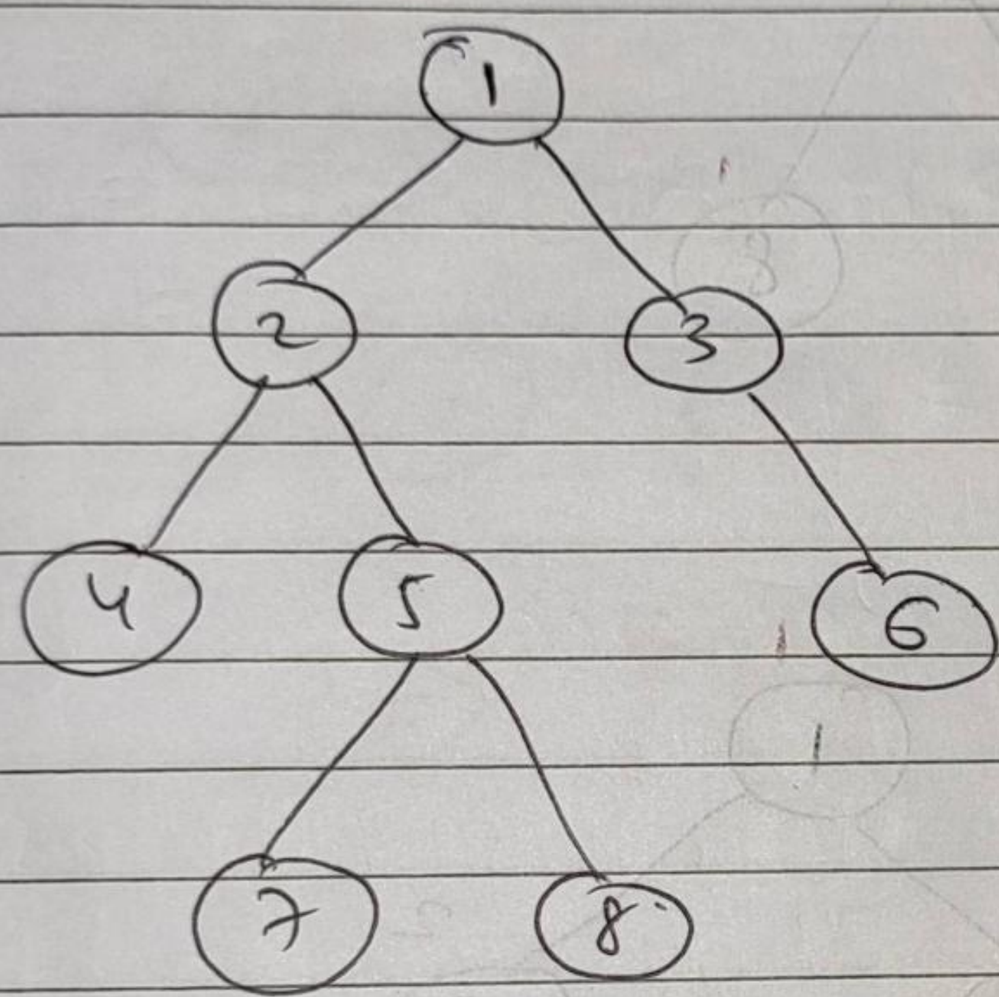
$$\frac{C^*}{C} \leq P(n) \qquad \frac{C}{C^*} \leq P(n)$$

$C \rightarrow$ Cost of solution
$C^* \rightarrow$ Cost of optimal solution
$P(n) \rightarrow$ Approximation ratio
$n \rightarrow$ input size.

Scanned by TapScanner

Vertex Cover Problem :-

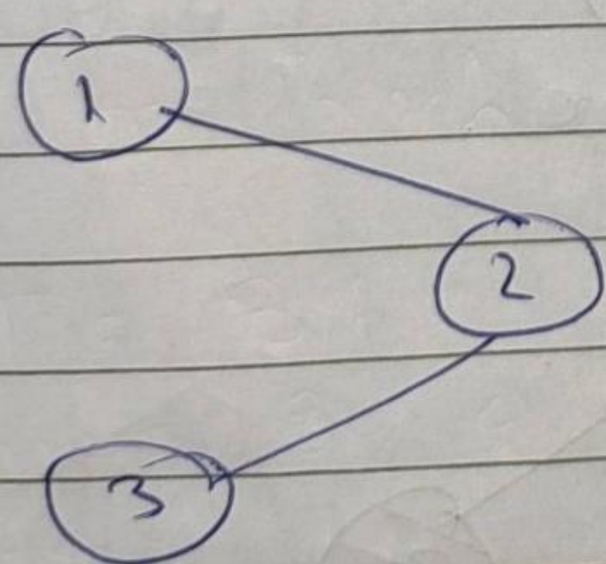
A vertex cover of a graph is a subset of vertex which covers every edge.
Vertex Cover problem is to find minimum size vertex cover.



{1, 3, 5, 2}
{5, 6, 2, 1}

This is vertex cover.

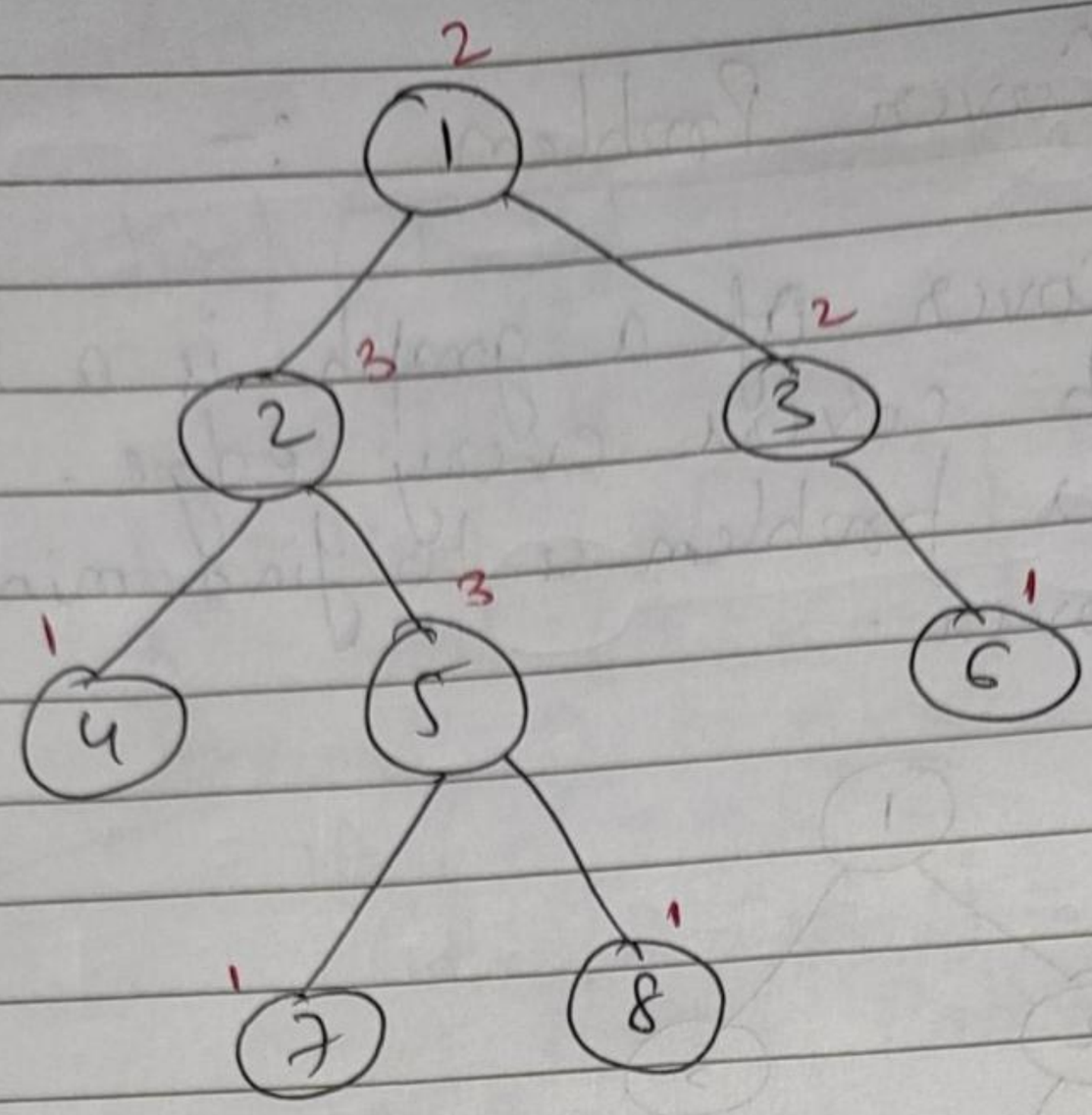
{2, 3, 5} → This is minimum vertex cover.



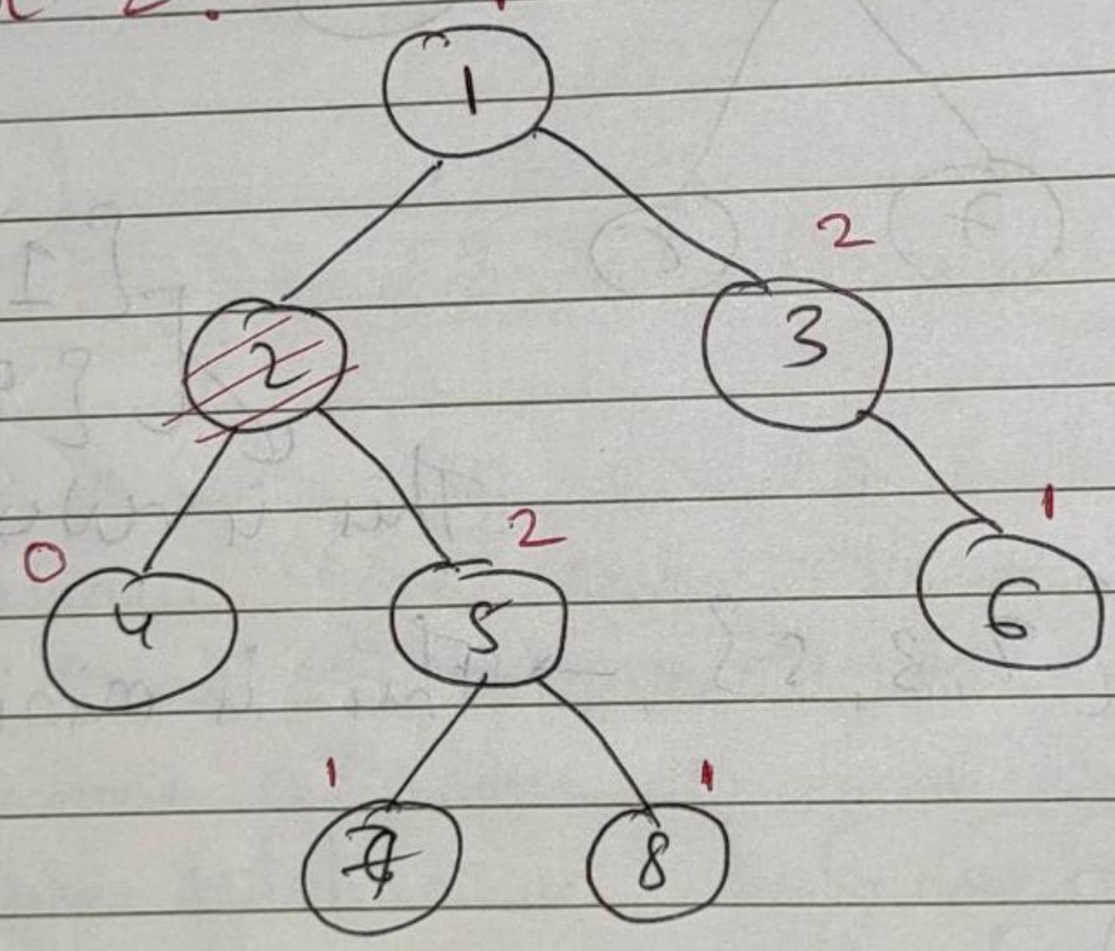
{1, 2}
{3, 2} → vertex cover

{2} → Minimum Vertex Cover

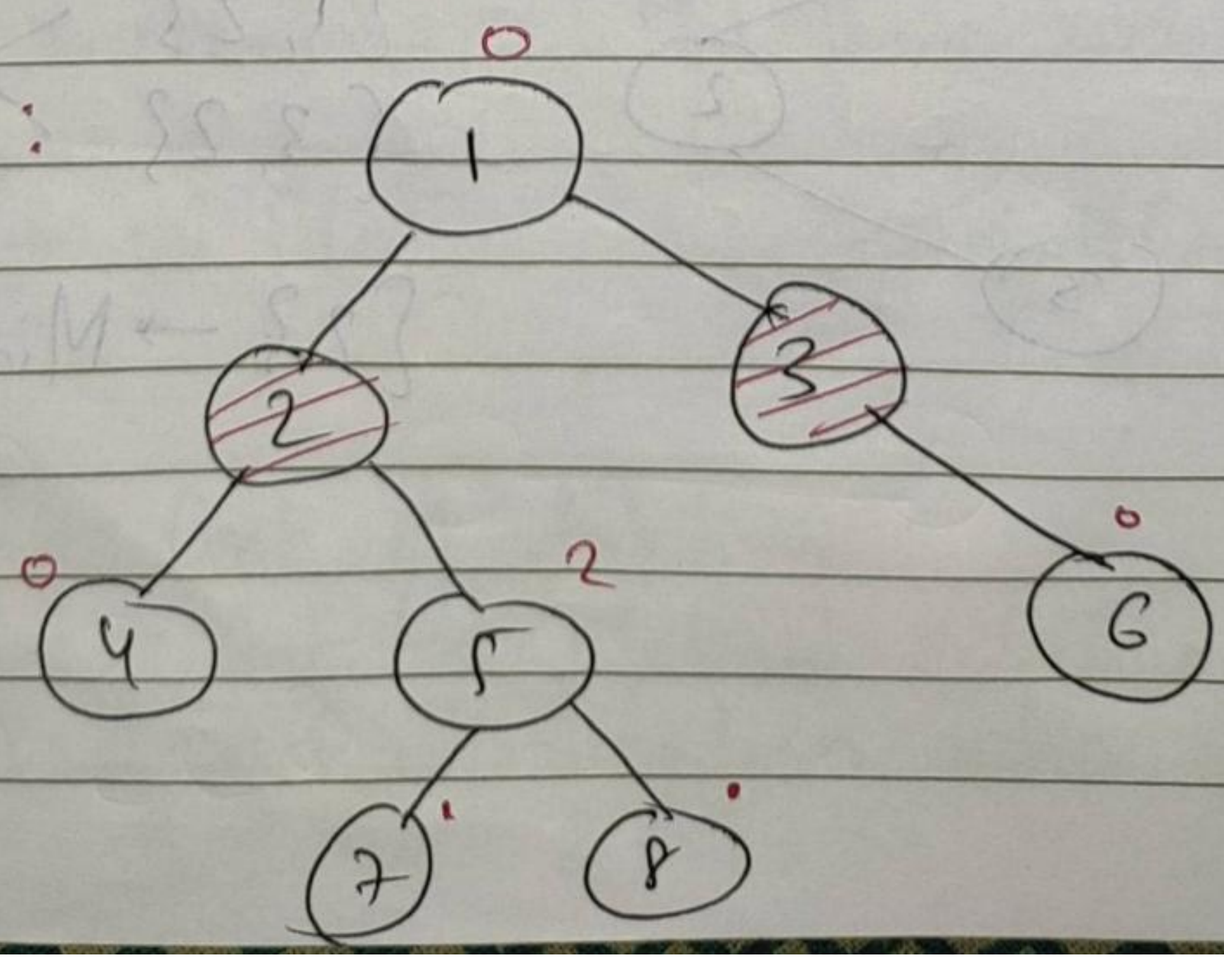
Scanned by TapScanner



After 2:-

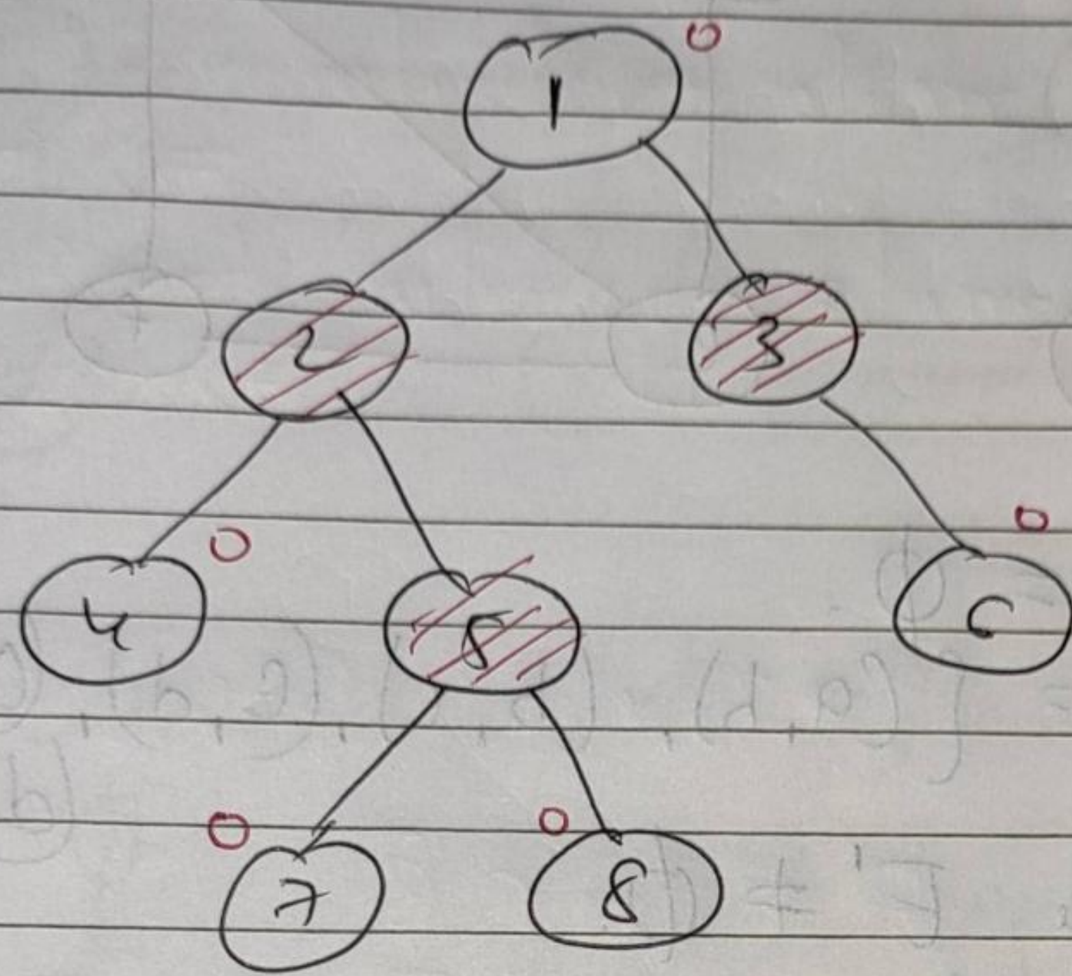


After 3:



Scanned by TapScanner

After 5 :-



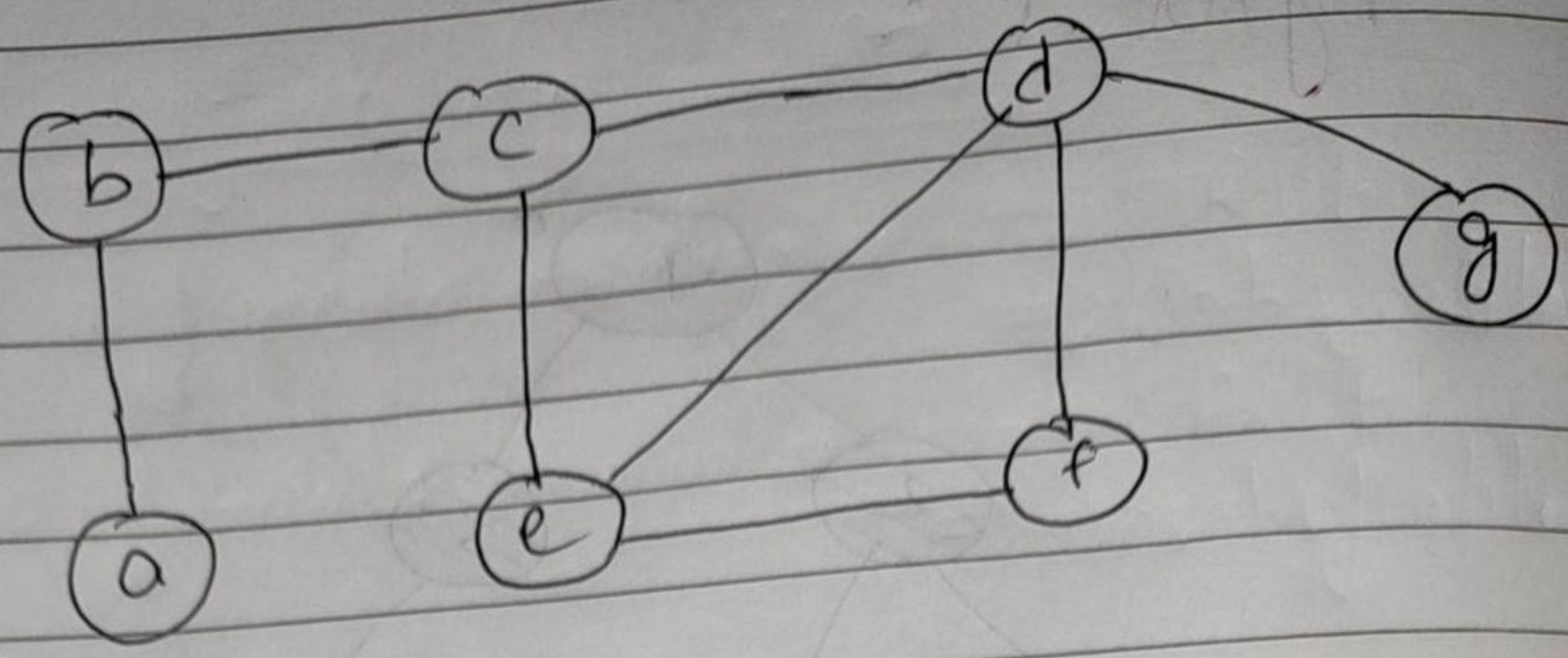
$\Rightarrow \{2, 3, 5\}$

Approximation Algorithm for Vertex Cover :-

Approx_Vertex_Cover_Problem $(G = (V, E))$.

1. $C \leftarrow \phi$
2. $E' \leftarrow E(G)$
3. while $(E' \neq \phi)$
4. do let (u, v) an arbitrary edge in E'
5. $C \leftarrow C \cup \{u, v\}$.
6. Remove from E' every edge incident either u and v.
7. Return C.

Scanned by TapScanner

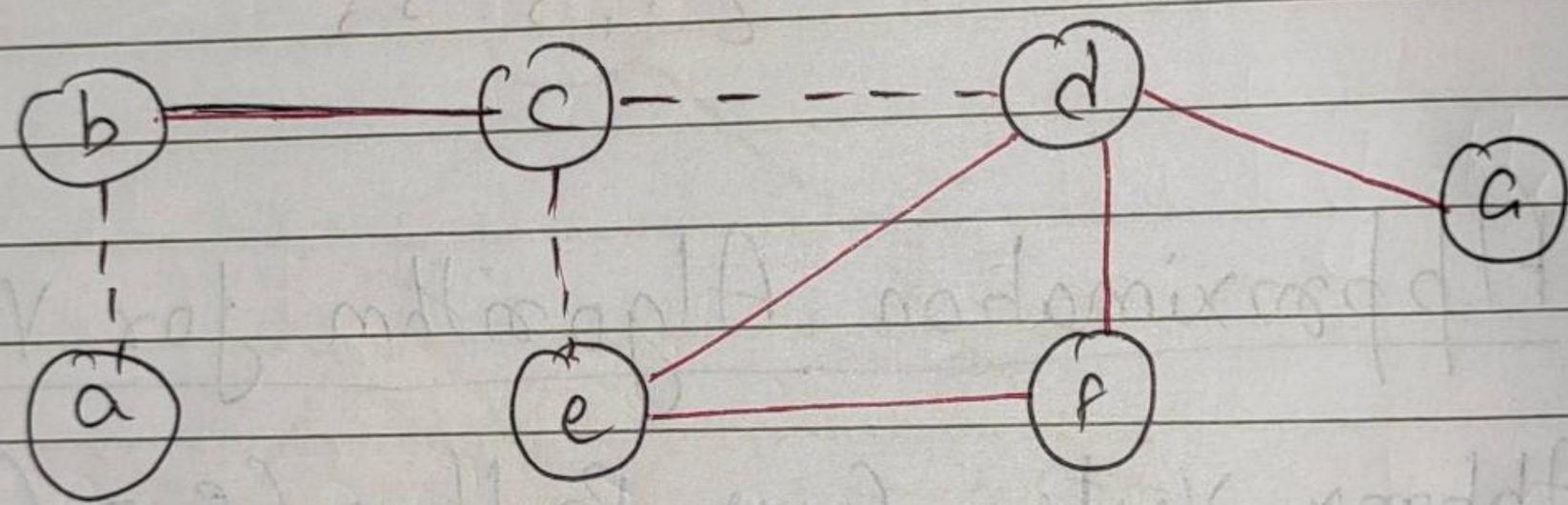


① $C = \phi$.

② $E' = \{(a,b), (b,c), (c,d), (c,e), (d,e), (e,f), (d,f), (d,g)\}$.

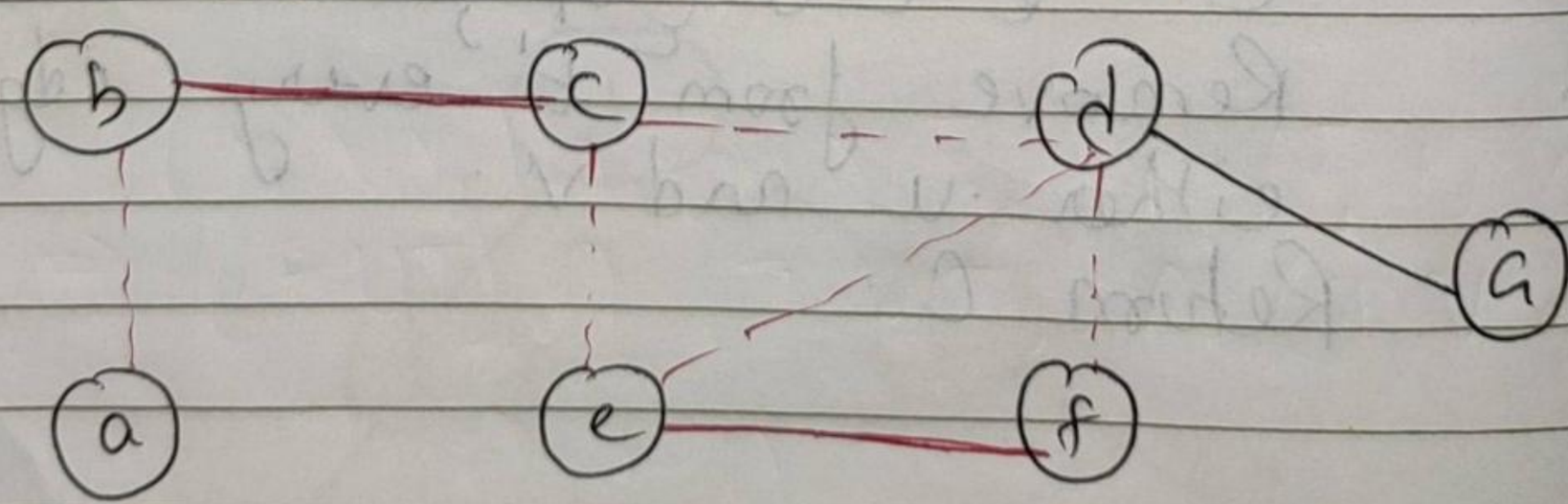
③ while $E' \neq \phi$.

(b,c) is considered $C = \{(b,c)\}$



$E' = \{(d,e), (e,f), (d,f), (d,g)\}$

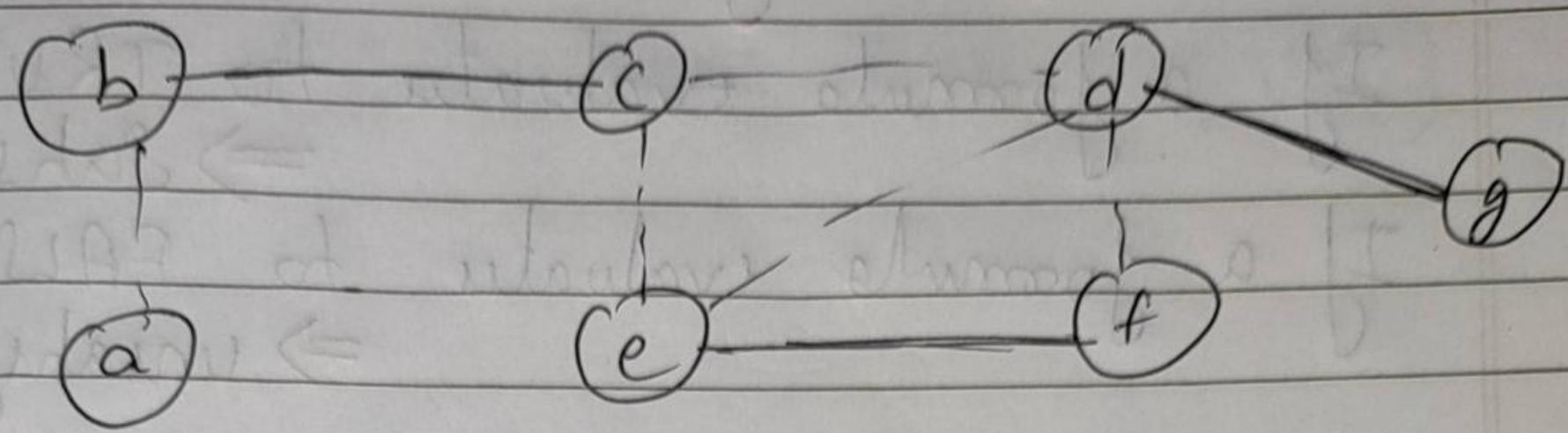
(e,f) is considered $C = \{(b,c), (e,f)\}$



$E' = \{(d,g)\}$

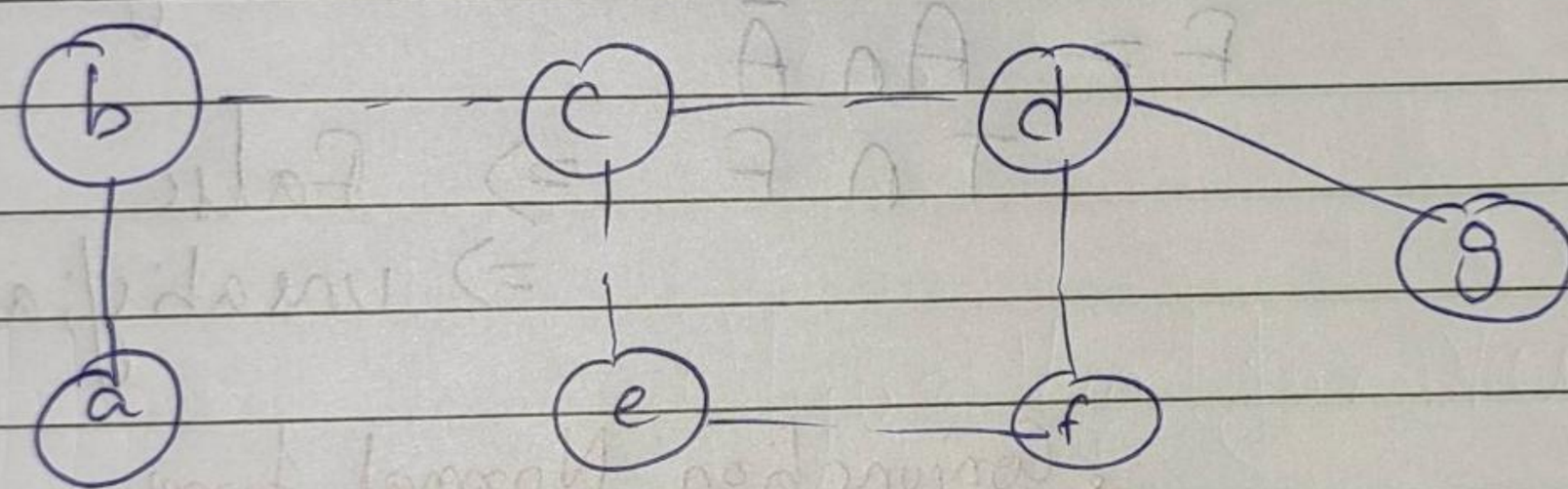
Scanned by TapScanner

(d, g) is considered.



$C = \{b, c, e, f, d, g\} = 6.$

$E' = \phi$ It is not an optimal solution



Optimal solution $= \{b, d, e\}$
$\Downarrow$
3.

minimization $\frac{C}{C^*} \leq P(n)$

$$\frac{6}{3} \leq P(n)$$

$$2 \leq P(n)$$

or $\boxed{P(n) \geq \cancel{P(n)}\ 2}$

Scanned by TapScanner

Satisfiability (SAT) :-

If a formula evaluates to TRUE
$\Rightarrow$ satisfiable
If a formula evaluates to FALSE
$\Rightarrow$ unsatisfiable.

Ex:-

$A \rightarrow$ True $\quad B \rightarrow$ false

$F = A \cap \bar{B}$
$T \cap T = $ True
$\Rightarrow$ satisfiable

$F = A \cap \bar{A}$
$T \cap F \Rightarrow$ False
$\Rightarrow$ unsatisfiable

Conjunction Normal form

2-SAT $\overbrace{\text{CNF}}$ $\rightarrow$ Clauses of 2 term each is joined by conjunction.

Ex:-

$F = (A_1 \cup B_2) \cap (A_2 \vee B_2)$.

Q $x_1 \rightarrow$ False, $x_2 =$ True.

$F = (x_1 \vee x_2) \cap (x_2 \vee \bar{x}_1) \cap (\bar{x}_1 \vee \bar{x}_2)$.
$\Rightarrow (F \cup T) \cap (T \cup T) \cap (T \vee F)$.
$T \cap T \cap T$
$=$ True
$\Rightarrow$ Satisfiable.

Scanned by TapScanner

3-SAT CNF :- 3 terms in each clause.

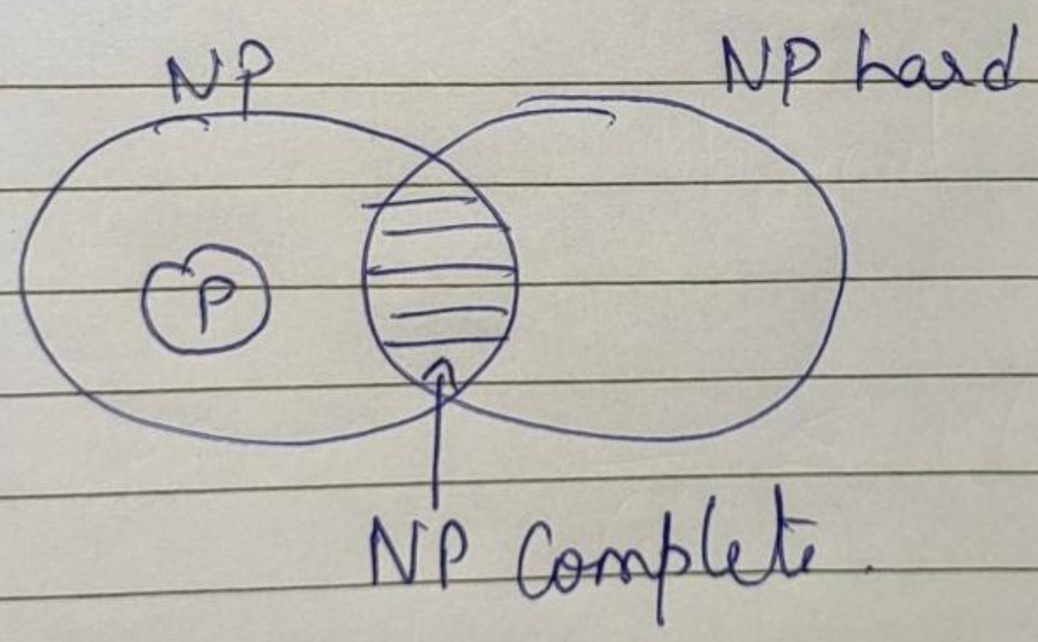
Ex:- $(A_1 \vee B_1 \vee C_1) \wedge (A_2 \vee B_2 \vee C_2)$.

P & NP problem

* We just need to study basic definition and relation for MCQ.

Definition of

* Non-deterministic Algorithm
* Deterministic Algorithm
* P
* NP
* NP-Complete



* In this, we try to relate the problem so that if 1 problem is solved then other can also be solved
* When we unable to write polynomial time deterministic algo. we try to write non-deterministic algo with polynomial time.

Scanned by TapScanner

P and NP

1. Try to relate the problems so that if 1 problem is solved then others also solved
2. When you are unable to write polynomial time Deterministic Algo try to write non-Deterministic polynomial time algorithm.

Non Deterministic Algorithm

```
Algorithm NSearch(A, n, key)
{
    j = choice();          — 1
    if (key = A[j])
    {
        write(j);          — 1
        success();         — 1
    }
    write(0);
    failure();             — 1
}
```

$O(1)$

A | 10 | 8 | 6 | 9 | 4 | 2 |
1 2 3 4 5 6

key = 9

j = 4

How choice is working we don't know

Polynomial time

- Linear Search = n
- Binary Search = log n
- Insertion Sort – n^2
- Merge Sort – n log n
- Matrix Multiplication n^2

Exponential time

Satisfiability → 2^n

- 0/1 Knapsack – 2^n
- Travelling SP – 2^n
- Sum of subsets – 2^n
- Graph Coloring – 2^n
- Hamiltonian Cycle – 2^n

↓

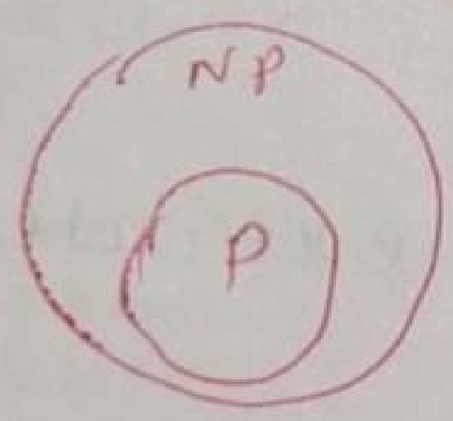
takes much bigger time as compared to polynomial time algo.

NP-Hard and NP-Complete

P → set of those Deterministic Algorithms which takes Polynomial time
Eg:- Linear search, Binary search, Bubble Sort, MST etc.

NP → set of those Non Deterministic Algorithm which takes Polynomial time

P NP



initially or earlier some Algos which are non-deterministic became Deterministic today.

Deterministic Algo → In Deterministic Algorithm each & every statement how it works we know it clearly i.e. the working of the Algorithm is clear.

Non-Deterministic Algo → most of the statements may be deterministic i.e. clear but some of the statements are not-deterministic. → Research → to convert non-deterministic part into deterministic

Scanned by TapScanner

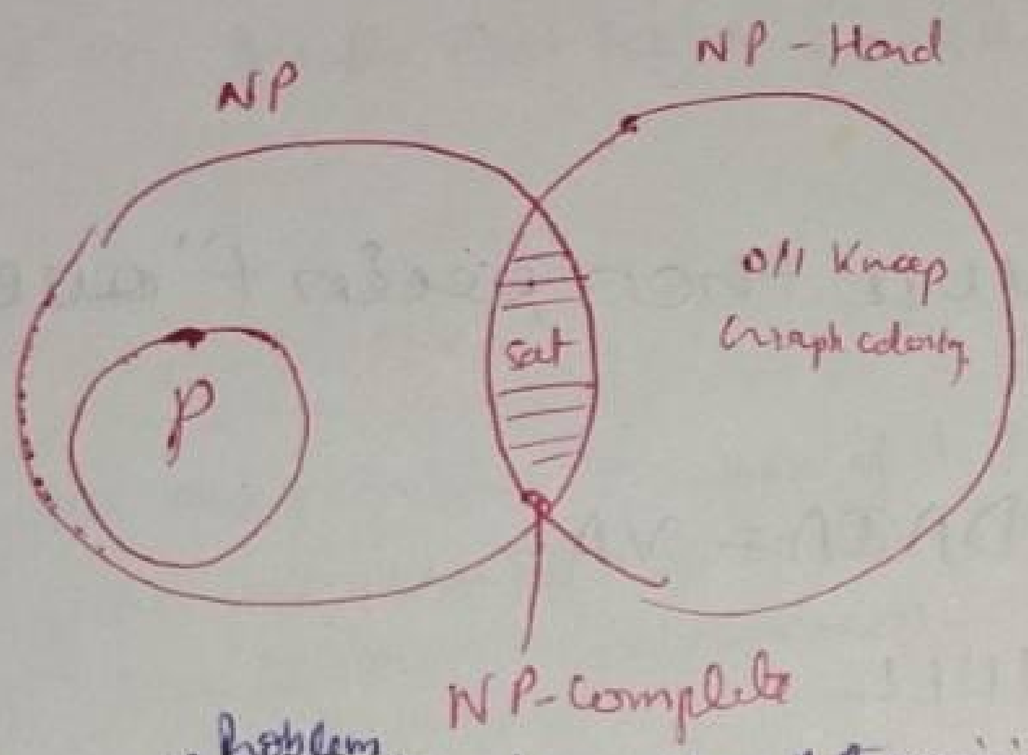
Reduction

NP-Hard — sat I_1 $\xrightarrow[\text{reduces}]{\alpha}$ 0/1 Knapsack I_2 — NP-Hard

Polynomial time

Sat α L

NP-Hard



NP-Complete → If a problem has Non-Deterministic Polyno[illegible] Algorithm then it is NP-Complete.

NP-Hard → Satisfiability - 2^n

NP-complete

If satisfiability is solved in polynomial time then all these problems can be solved in polynomial time.

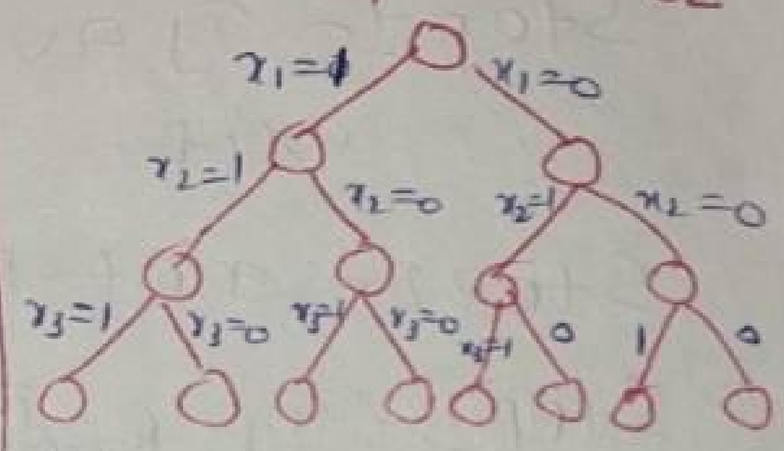
CNF-Satisfiability — 2^n

$x_i = \{x_1, x_2, x_3\}$

CNF = $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee$

$= (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}$

C_1 C_2



Tautology satisfiability on base proble[illegible]

Scanned by TapScanner

Hamiltonian Cycle Problem

CS42005

Definition: A Hamiltonian cycle is a cycle in a graph that visits each vertex exactly once.

To show Hamiltonian Cycle Problem is NP-complete, we first need to show that it actually belongs to the class NP, and then use a known NP-complete problem to Hamiltonian Cycle.

Does Hamiltonian Cycle Problem ∈ NP?

Given: $Graph\ G\ =\ (V, E)$

Certificate: List of vertices on Hamiltonian Cycle

To check if this list is actually a solution to the Hamiltonian cycle problem, one counts the vertices to make sure they are all there, then checks that each is connected to the next by an edge, and that the last is connected to the first.

It takes time proportional to n, because there are n vertices to count and n edges to check. n is a polynomial, so the check runs in polynomial time.

Therefore, Hamiltonian Cycle ∈ NP.

Prove Hamiltonian Cycle Problem ∈ NP-Complete

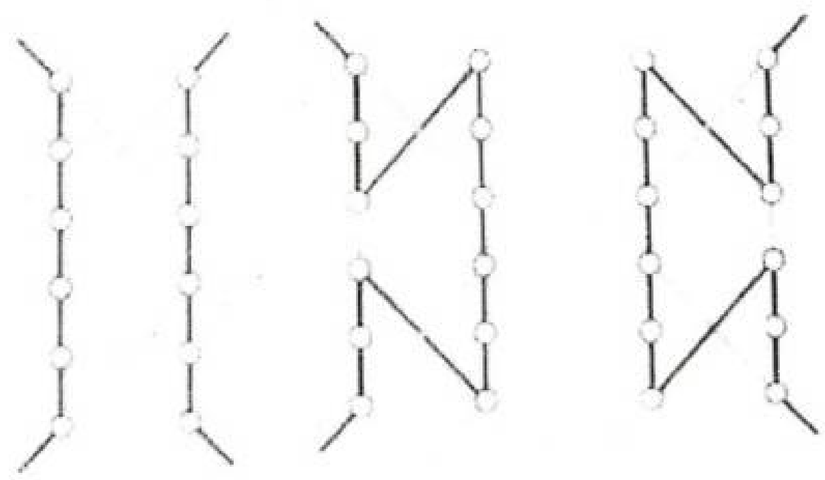
Reduction: Vertex Cover to Hamiltonian Cycle

Definition: Vertex cover is set of vertices that touches all edges in the graph.

Given a $graph\ G$ and integer k, construct a $graph\ G'$ such that G has a vertex cover of size k iff G' has a $Hamiltonian\ cycle.$

Idea: To construct widget for each edge in the graph.

$i.e \qquad \forall\ uv$ in the $Graph\ G,$ create a widget shown below;



REDMI NOTE 9 PRO
AI QUAD CAMERA

1. Enter from u, go somewhere else in the graph, and then come back through the other side i.e v
2. Enter and Exit through u
3. Enter and Exit through v

Construct G' for G (Vertex cover) of size $k\ =\ 2$

With the construction, any graph with a vertex cover, can be used to make a graph with a Hamiltonian Cycle graph. Since creating such a graph can be done under polynomial time, simply replace edges with widgets and make proper connections, we have a reduction from Vertex Cover to Hamiltonian Cycle.

This means that finding whether a graph has a Hamiltonian Cycle or not is NP Hard. As we have seen earlier it's also in NP, therefore, Hamiltonian Cycle is an NP Complete Problem

Scanned by TapScanner