EDIT Distance using DP.

Edit distance has overlapping subproblems, ∴ ~~[illegible]~~ use Memoization. → i j

$\therefore$ DP $[\overset{i}{N}] \times [\overset{j}{M}] \Rightarrow$ $\boxed{TC = (N \times M) \text{ no of states}}$

Space = O(N×M) ← DP array
+ O(N+M) ← auxilliary stack space

s1 [0,... i]
s2 [0 ... j]

```
int f(int i, int j, string s1, s2){
    if (i<0) return j+1;
    if (j<0) return i+1;      => //If either string gets exhausted,
                                 return other strings length.

    if (s1[i] = s2[j]) return 0 + f(i-1, j-1, s1, s2);
                              ↑ no insertn, deln or update if character matches
    else
    return 1 + min(f(i-1, j, s1, s2), min(f(i, j-1, s1, s2),
                       f(i-1, j-1, s1, s2)))
```

⇓ using DP

Memoization

```
f(i, j, &s1, &s2, &dp)
{
    if (i<0) return j+1;
    if (j<0) return i+1;
        if(dp[i][j] != -1) return dp[i][j];
    if (s1(i) == s2(j)) return 0 + f(i-1, j-1, s1, s2, dp);
    else
    return dp[i][j] = 1 + min{ f(i-1, j, s1, s2, dp)  //delete
                               f(i, j-1, s1, s2 dp)   //insert
                               f(i-1, j-1, s1, s2, dp) //update
}
```

$\boxed{TC = N \times M}$

Tabulation

```
n = str1.size()
m = str2.size()
```

Base cases. dp[n+1][m+1]

```
① ~~for~~ ~~dp[i]~~ for (i=0 → N)   dp[i][0] = i;
                 for (j=0 → M)   dp[0][j] = j;

② for (i=1 → n) {
      for (j=1 → m) {
        if s1[i-1] = s2[j-1]   dp[i][j] = dp[i-1][j-1]
  else dp[i][j] = min { dp[i-1][j] + 1
                        dp[i][j-1] + 1
                        dp[i-1][j-1] + 1
```

→ m

↓ n

	0	1	2	3
0	0	1	2	3
1	1			
2	2			
3	3			
4	4			

Subset sum equals to target.

arr {$\overset{0}{1}$, $\overset{1}{2}$, $\overset{2}{3}$, $\overset{3}{4}$}

target = 4

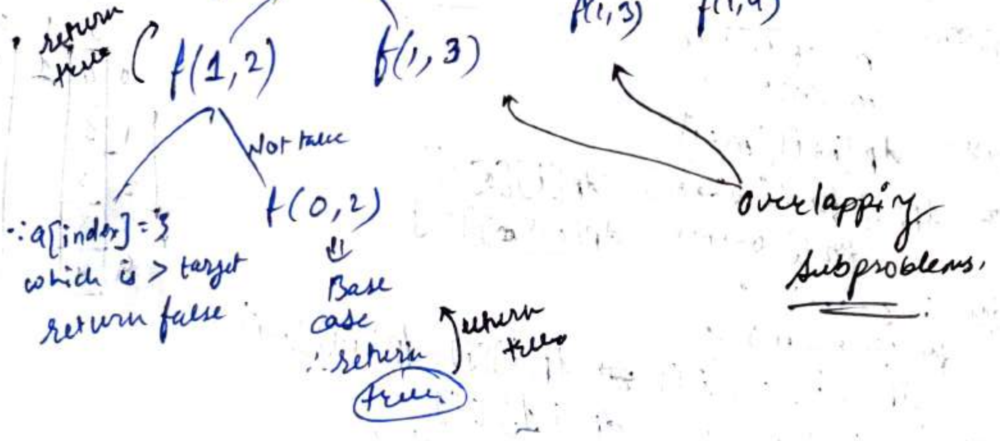
} f(3,4) ⇒ means on the entire array, index 0 to 3 does there exist someone with a target = 4.

```
f(index, target)
{
    Base cases. [ if (target == 0) return true
                [ if (index == 0) return true
                [      if a[0] == target;

    Now explore all possibilities of the index. ⇒
        bool not take = f(index -1 , target)
        bool take = false
            if(a[index] ≤ target)
                take = f(index-1 , target - a[index])

    If any of the possibilities gives true for the target to exist, return it.
        return (take) or (not take);
}
        main() { return f(n-1, target); }
```

Eg:- arr = {$\overset{0}{2}$, $\overset{1}{3}$, $\overset{2}{1}$, $\overset{3}{1}$}   target = 4



Recussion ⇒ • Time comp ⇒ ~~2F~~ $T(n) = 2T(n-1) + c$

$= O(2^n)$

∴ There exist subproblems ↓

• Space comp ⇒ $O(N)$

∴ memorization can be applied

① Figure out change in states
index and target changes here

② So, DP array = [index] x [target]

To reduce auxiliary space

• Time comp = $O(N \times target)$

• Space comp = $O(N \times target) + O(N)$ ← Auxiliary space

↓ Tabulation

① Figure out Base cases -

for (i=0 → n-1) dp[i][0] = true;

dp [0] [a[0]] = true;

② Form the nested loops.

index → (1 to n-1)

target → (1 to target)

③ Copy the recursion code.

— — -

— —

target →

index ↓	0	1	2	3
0	T			
1	T			
2	T			
~~items~~ index 3	T			

• T.C = $O(N \times target)$

• S.C = $O(N \times target)$

Memoiz^n code

```
bool f(index, target, &arr, &dp) {
Base cases { if (target == 0) return true;
             if (index == 0) return (arr[0] = target);
Check dp table ← if (dp[ind][target] != -1) return dp[ind][target];

    bool not take = f (index - 1, target, arr, dp);
    boot take = false ;
    if (arr(index) <= target)
        take = f (index-1,, target - arr(index), arr, dp)
    return dp (index][target] = take || Not take.
}
```

Funct^n call ⇒ f (n-1, target, arr, dp);

Tabulation :

~~bool f (index, target)~~
~~{~~
~~same code as memoization~~
~~}~~

~~bool subset sum to k (n, target, arr)~~

```
bool subset sum to k (int n, target k, vector<int> arr)
{ vector<vector<bool>> dp (n, vector<bool> (k+1, 0));
① Base { for (int i=0 -> n-1) dp[i][0] = true;
  case { dp [0] [arr[0]] = true ;

② for (int index = 1  to n-1) {
      for (int target = 1  to -> k) {
③         bool not take = dp [index-1][target]
          bool take = ~~dp~~ false;
          if (arr(index) <= target)
                take = dp [index-1] [target - arr[index]]
                dp [index][target] = take || Not take;
      }
   Return dp[n-1][k];
}
```

Reduction

- P: { Problems solvable in polynomial time }
- NP:- { Decision problems solvable in non deterministic polynomial time }

$\swarrow$ Answer is Yes (or) NO

$\downarrow$ can guess out of polynomially many options, if any guess leads to YES answer then we get such a guess in ~~poly time~~ O(1) time.

---

* 3SAT : Given Boolean formula of the form

$$\underbrace{(x_1 \vee x_3 \vee \bar{x}_6)}_{\text{clause}} \wedge (\bar{x}_2 \vee x_3 \vee x_7) \wedge --$$

Decision Problem :- Can you set the variables x_1, x_2 — such that formula come out to be True.

↳ This is a hard problem, we dont know polytime algo to solve this.

But we do have a polytime non-deterministic algo. So, this problem is in NP.

3SAT ∈ NP :- guess x_1 = True or False
guess x_2 = True or False } All takes poly time.
⋮
Check formula < If True : set YES
If False : return No.

This a polytime verificatⁿ algo that check if solⁿ is valid. You can check it in polynomial time.

- NP Complete :- A problem x is NP-complete if
  1. $x \in NP$.
  2. $A \le_p x$ for some known NP-complete Prob. A

$\forall A \in NP \leftarrow$ 2.

- x is NP complete if $x \in NP$ and x is NP-hard
- x is NP-hard if every problem $Y \in NP$ reduces to x.

x is NPC if ⇒ $\boxed{\text{Every } Y \in NP \le_p \underset{\uparrow}{A} \le_p x}$ ⇒ So every Y reduces to x.

A is a known NP complete

All NP → SAT → x
∴ All NP → x

Reductions:- From Problem A → Problem B

⇒ Polytime algorithm converting A inputs to equivalent B inputs

* NP complete:-

A problem X is NP-complete if:

① X∈NP

② A ≤p X for some known NP-C Problem A

⇒ To prove B is NP-complete:

✓ • Reduce known NP-C problem to B

✗ • Never reduce B to known problem

→ To Prove B is NP-complete

1. B ∈ NP (∵ a solⁿ can be verified in poly time)
2. Let A be known NP-complete problem
3. We construct a polytime reductⁿ A → B
4. We show A is satisfiable if and only iff B has a s…
5. Hence B is NP-complete

- Identify known NPC problem
- State reductⁿ
- construct mapping
- Proof correctness
- mention poly time

Cook-levin Theorem:-

SAT is NP-complete, Therefore every problem in NP can be polynomially reduced to SAT

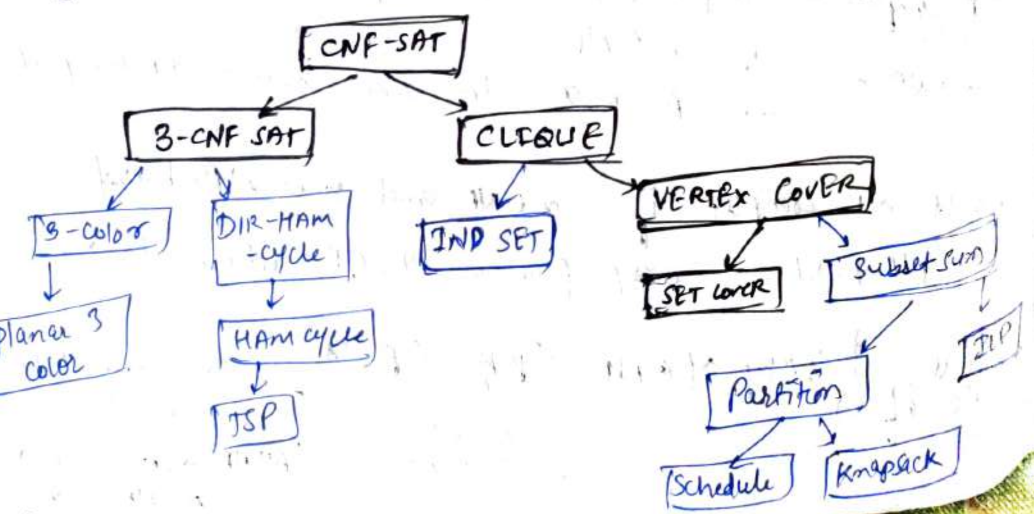
CkT Sat ↓

✓ SAT → $\overset{\text{3 CNF (or)}}{\text{3SAT}}$ → Clique → Indep. set → vertex cover → set cover

✓ Clique ↔ Indep. set (Complement graph)

- Subset Sum → partition → knapsack
- Ham-cycle → TSP

} 4 main reductⁿ chains



*Note :- To prove X is NP-complete, reduce a known NP-complete problem to X and show the reduct^n is polynomial and correctness preserving.

For example :-

To prove clique is NP-complete.

1. Write Clique ∈ NP.
2. Reduce 3SAT → clique (∵ SAT is a known NP-C problem by cooks theo.)
3. Construct a graph.
   Where each literal is a vertex,
   Connect compatible ~~vertex~~ literals, seek k-clique.
4. Satisfiable ⇔ clique exist

- SAT → General satisfiability
- CNF → Format (AND of OR's)
- 3SAT → SAT with 3 literals per clause
- Circuit-SAT → Boolean circuit satisfiability

Q1) Prove that Clique ∈ NPC.

Clique :- Subset of vertices $V' \subseteq V$ such that every two distinct vertex in V' are connected by edge E.

. Decision problem

In a graph $G(V,E)$ and integer k, does G contain a clique of size k?

* **Proof for NP** :- X: Instance of a graph $G(V,E)$ and k

Y: Set $V' \subseteq V(G)$

$V(X,Y)$: Given a subset of vertices, V' then

polytime

```
i) If $|V'| \neq k$ return false    ← no. of vertex not equal to cliquesize
ii) If $|V'| = k$
        for u in V':
            for v in V':
                If $u \neq v$ and $(u,v) \notin E$
                    return false
    else return True
```

- So verifying clique we need $O(|V|^2) = O(k^2)$ [polynomial] time

∴ $\boxed{\text{Clique} \in NP}$

$$\boxed{\text{CKTSAT} \rightarrow \text{SAT} \underset{\equiv_p}{\rightarrow} \text{3CNF SAT} \rightarrow \text{Indep Set} \leftrightarrow \text{Clique} \rightarrow \text{PCS} \rightarrow \text{ILP}}$$

* **Proof of NPC.**

Using cooks Theorem we know circuit sat ∈ NPC, so we show $\text{CKT SAT} \leq_p \text{SAT} \leq_p \text{3CNF} \leq_p \text{Clique}$

Ⓐ . Proof of $\text{CKT SAT} \leq_p \text{SAT}$.

① Assign variables x_i for each input of a circuit.

② Assign variable x_0 for output

③ Set up an if and only if formula for each gate. Let ϕ_k be the formula for kth gate.

④ Let x_0 be final output ∃ $x_{0,f} = \phi_1 \cdot \phi_2 \cdot \phi_3 \ldots$
(CNF formula)

- Reductⁿ for NOT Gate

$\phi = x_1 \longleftrightarrow \overline{x_2}$

$= (\overline{x_1}+\overline{x_2})(x_1+\overline{\overline{x_2}})$

$= (\overline{x_1}+\overline{x_2})(x_1+x_2)$

So NOT gat can be reduced to CNF in poly time.

- Reductⁿ for AND Gate

$\phi = x_3 \Leftrightarrow x_1 \cdot x_2$

$= (x_3 + \overline{x_1 \cdot x_2})(\overline{x_3} + x_1 \cdot x_2)$

$= (x_3 + \overline{x_1} + \overline{x_2})(\overline{x_3} + x_1)(\overline{x_3} + x_2)$

Hence AND Gate can be reduced to CNF in polytime

- Reductⁿ for OR Gate

$\phi = x_3 \Leftrightarrow x_1 + x_2$

$= (x_3 + \overline{(x_1+x_2)})(\overline{x_3} + x_1 + x_2)$

$= (x_3 + \overline{x_1})(x_3 + \overline{x_2})(\overline{x_3} + x_1 + x_2)$

So OR gate can also be reduced to CNF in polytime

Hence each gate can be reduced to CNF formula ϕ in poly time ∴ $\boxed{\text{CKT SAT} \le_p \text{SAT}}$

⑤ Proving 3SAT $\le_p$ Clique

We reduce a known NP-complete problem 3SAT to Clique

- Let the input formula be

$\phi = (x_{11} + x_{12} + x_{13})(x_{21} + x_{22} + x_{23}) \cdots\cdots (x_{n1} + x_{n2} + x_{n3})$

There are n clauses, each clause contain 3 literals.

* Construction of Graph G.

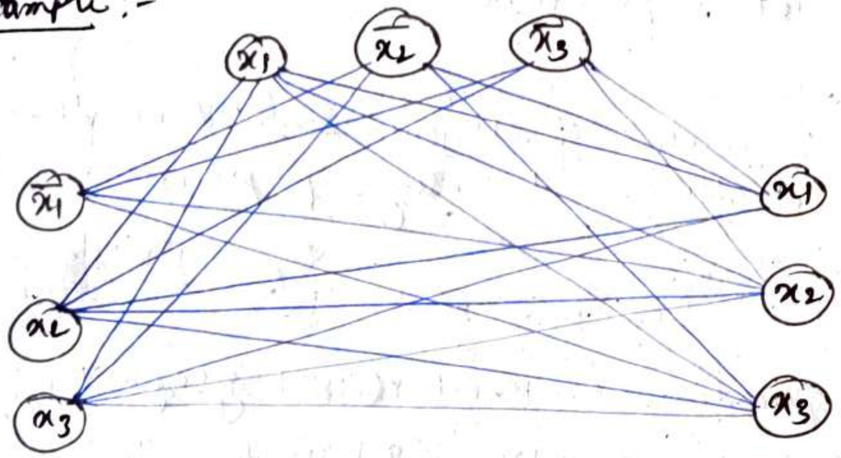
(*)

① Construct a graph G of k clusters with 3 nodes each.
② Each cluster corresponds to a clause (each clause contributes 3 vertices)
③ For each literal in each clause, create a vertex.

④ Add an edge between two vertices if :

1. They come from different clauses , and
2. add edge except for the pairs of the form $(x_1, \bar{x}_1)$

Example :-



Ⓒ Correctness

- If ϕ is satisfiable , then each clause has atleast one true literal . Choosing one true literal from each clause gives K vertices , all pairwise compatible .

  Hence they must form a clique of size $k(n)$

- If G has a clique of size k , it must select one vertex from each clause (∵ no edges exist b/w same clause) . These chose literals cannot contradict, so they give a satisfying assignment for ϕ .

  ∴ ϕ is satisfiable ⇔ G has a clique of size k .

The constructⁿ is clearly polynomial

∴ 3SAT $\le_p$ Clique

So, clique is NP-Hard .

Ⓓ conclusion

since 1. clique ∈NP
2. 3SAT $\le_p$ Clique and 3SAT is NP-complete

$\boxed{\text{clique is NP-complete}}$

*Reductions and Decision Problems:-

(Q1) Prove that PCS is in class NP (10 marks)

~~(PCS = circuit SAT)~~

~~• You need to only show a solⁿ (input assignment) can be verified in poly time.~~

• Decision Problem:-

Is there a schedule of all jobs such that they all finish before deadline d using P processors.

• PCS : $V(X, Y)$

X: set of precedence constraints (Task graph T), containing set of Jobs $J = \{J_1, J_2, J_3 -- J_n\}$, No of processors P and Deadline D.

Y: $t_1 ; t_2 -- t_n$ ← Processing time for Jobs

t_i = time at which Job i is scheduled.

To show a problem is in NP, we must show

Given a proposed solⁿ, it can be verified in polytime.

Proof:- We are given a certificate (proposed solⁿ) consisting of an ordering of all jobs (schedule)

we verify this certificate in polytime by

① Check Precedence constraints

For every constraint $J_i < J_j$:

- verify that Job J_i appears before J_j in schedule
- This can be done by scanning Job order

- takes atmost $O(n^2)$ time

② Check Completion time:

- Simulate the scheduling by adding processing times in order
- Compute total finishing time
- verify that it is $\leq D$ → This takes $O(n)$ time.

③ Time complexity :- Both checks are polynomial
∴ verificatⁿ runs in polytime
∴ $\boxed{PCS \in NP}$

---

*Q) Prove Independent set is in NP:

Decision Problem:- Does G contain Indep. set of size k
(k ~~et~~ vertices with no edges b/w any pair)

Input :-
- A graph $G(V, E)$
- An integer k

Proof:- we show that proposed solⁿ can be verified in polytim
set of vertices $S = \{v_1, v_2 --- v_n\}$ claimed to be indep. set

① Check that $|S| = k$ → $O(1)$
② For every pair (v_i, v_j) with $i \neq j$ check
$(v_i, v_j) \notin E$ → $O(k^2)$

Conclusion :- A certificate can be verified in poly time :
$\boxed{\therefore \text{Indep set} \in NP}$

*Q) Prove that Indep-set is NP-complete.

Step 1:- Indep set ∈ NP ← already proved.

Step 2:- Independent set is NP-Hard.
we reduce a known NP-complete prob **CLIQUE** to Indep-set
Clique $\leq_p$ Indep set.

* Construction

Instance of Clique:
- Graph $G(V, E)$
- Integer k
- Question: Does G have a clique of size k.

Construct a new Graph
Let $\bar{G}$ be the complement of Graph G

- Same vertices : V
- Edge exist in $\bar{G}$ if and only if it does not exist in G

keep the same integer k

- Clique in G : Every pair is connected
- Indep. set in $\bar{G}$: No pair is connected.
- Since edges are flipped, these conditions are equivalent

* correctness of Reductⁿ

$(G, k) \in$ CLIQUE $\Leftrightarrow (\bar{G}, k) \in$ Independent set

So solving Indep. set on $\bar{G}$ solves Clique on G

* Poly time :-

constructing complement graph takes $O(|V|^2)$ time

* conclusion :-

1. Independent set $\in NP$
2. Clique $\leq_p$ Indep. set

} Hence $\boxed{\text{Indep. set is NP-C}}$

---

*Q) Prove that ~~Indep~~ vertex cover is NP-complete.

(Indep set $\leq_p$ vertex cover)

$\downarrow$
known decision problem

Theorem :- Vertex cover decision problem is NP-complete

Step ① :- vertex cover belongs to NP

Decisⁿ prob :- Does G have a vertex cover of size $\leq k$?

(a set of vertices that touches every edge)

Input :-
- Graph $G = (V, E)$
- Integer k

Certificate :- A set $C \subseteq V$ with $|C| \leq k$

verificatⁿ (poly time) :- For every edge $(u, v) \in E$; check $u \in C$ or $v \in C$

This takes O(E) time ← polynomial

Hence $\boxed{\text{vertex cover} \in NP}$

Step ②:- vertex cover is NP-Hard.

We reduce Indep set (known decisⁿ problem) to vertex cover

$\boxed{\text{INDEP set} \leq_p \text{vertex cover}}$

Construction:-

Instance of Indep set

- Graph $G = (V, E)$, • Integer k

Construct instance of vertex cover

- use the same graph G
- set new integer $k' = |V| - k$

Key Lemma:- A set S is an independent set in G if and only iff $V \setminus S$ is a vertex cover in G

Proof:-
- If S is indep set, no two vertices in S share an edge.

  ∴ Every edge must have atleast one endpoint outside S so all edges are covered by $V \setminus S$

- conversly if C is a vertex cover, then no edge has both endpoints outside C. So $V \setminus C$ contains no adjacent vertices → Independent set

Correctness:- $(G, k) \in$ Indep set $\Longleftrightarrow$ $(G, |V| - k) \in$ vertex cover

Thus solving vertex cover solves indep set.

The transformatⁿ is clearly polytime

Conclusion:-
- vertex cover ∈ NP
- Indep set $\leq_p$ vertex cover

∴ $\boxed{\text{Vertex cover is NP-complete}}$

Que 2) Prove that Formula satisfiability $\leq_p$ 3-CNF satisfiability and show SAT is NPC.

PART A :- Formula SAT $\leq_p$ 3-CNF SAT

(i.e SAT $\leq_p$ 3SAT)

Given :-

SAT = Satisfiability of any Boolean.

3SAT = Satisfiability of Boolean Formula in 3CNF form (each clause 3 Literals).

we must show that SAT instance can be converted into 3SAT in poly time

Step①:- convert SAT to CNF

Any Boolean formula can be transferred into Conjuctive Normal form (CNF) using logical equivalence

- De morgans law
- Distributive laws.

This transformation takes poly time.

So:- $\boxed{\text{SAT} \leq_p \text{CNF-SAT}}$

Step②:- Convert CNF to 3-CNF

- If a clause has more than 3 literals

  eg:- $(x_1 + x_2 + x_3 + x_4)$

  replace by:- $(x_1 + x_2 + y_1) \wedge (\neg y_1 + x_3 + x_4)$

- If a clause has less than 3 literals

  eg:- $(x_1 + x_2)$

  replace by:- $(x_1 + x_2 + z) \wedge (x_1 + x_2 + \neg z)$

Hence $\boxed{\text{CNF-SAT} \leq_p \text{3SAT}}$

Part B :- Prove that SAT is NP - complete

Step 1 :- SAT ∈ NP.

Given a truth assignment for variables, we can verify whether the formula is true by evaluating it in polytime

Hence :- $\boxed{\text{SAT} \in \text{NP}}$

Step 2 :- SAT is NP - Hard.

By Cooks Theorem, every problems in NP can be reduced to SAT in polytime. Since SAT and circuit SAT are polynomial - time equivalent, Every NP problem is also reducible to circuit SAT.

Hence :- $\boxed{\text{SAT is NP-Hard}}$

conclusion :-
1. Since SAT ∈ NP
2. SAT is NP - Hard
3. SAT $\le_p$ 3SAT

we conclude $\boxed{\text{SAT is NP-complete and SAT} \le_p \text{3 SAT}}$

Que 4) Prove that Independent set $\le_p$ Circuit Sat and IS is NP complete

Soln
- IS - Decision problem
  - Input : Graph $G = (V, E)$, integer k
  - Questn :- Does G contains indep. set of size atleast k ?
    (no two vertices in the set are adjacent)
- Circuit satisfiability (Circuit - SAT)
  - Input :- Boolean circuit C with some input variables
  - Question :- Is there an assignment to the inputs such that the output of C is 1.

we must build, in polytime, a circuit $C_{G,k}$ that is satisfiable iff G has an indep set of size atleast k.

* Construct^n of cirsat

Claim :- If circuit outputs to 1, then S forms an indep. set

Proof :- ① variable Assignment

For each vertex $v_i \in V$ creates a boolean variable x_i such that $x_i = 1$ if v_i ~~proof~~ ~~is~~ part of indep. set

$x_i = 0$ if v_i is not part of indep set

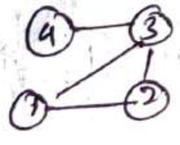
② Non Adjacency Constraint

For an edge $(v_i, v_j) \in E$ we need a constraint that includes only one of v_i, v_j. That is both cant come at a time

$\therefore$ clause is $\neg(x_i \land x_j) \Rightarrow (\overline{x_i} \lor \overline{x_j})$ [Demorgans law]

These are ~~and~~ AND and OR operat^{ns} which can be implemented by logic gates

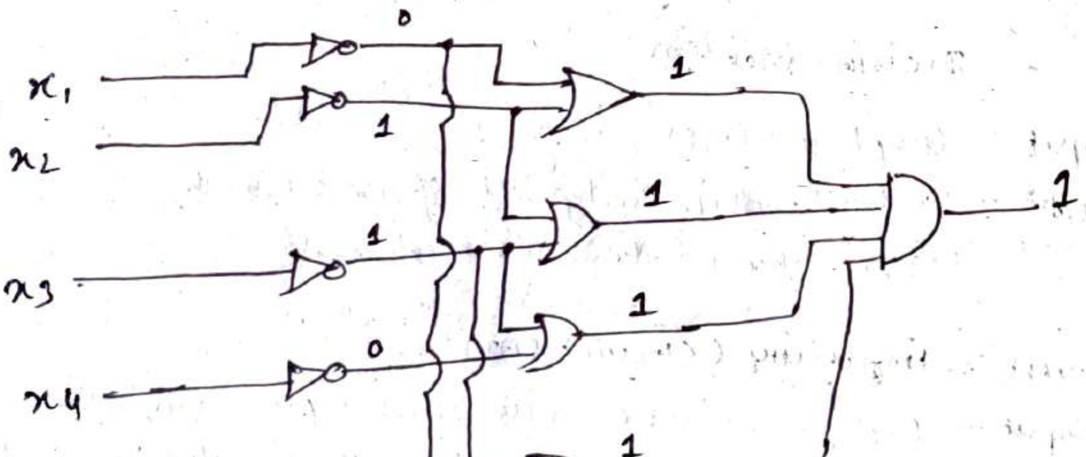
eg:-



$k=2$, $G = \{ v(1,2,3,4)$
$E[(1,2)\ (1,3)\ (1,2)\ (4,3)] \}$

Ind. set → $S = \{1,4\}$

	x_1	x_2	x_3	x_4
Variable assignment ⇒	1	0	0	1



Circuit C_1

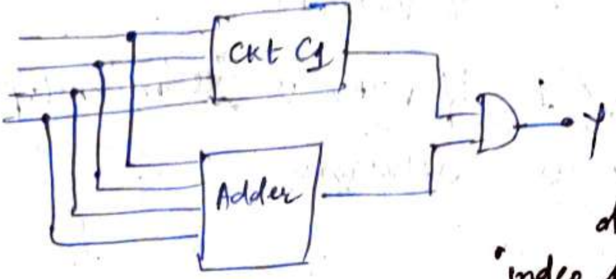
Above is logical circuit handling non adjacency vertices in the given subset of graph

③ Size constraint.

$|S| = k$ needs to be ensured in indep. set using adder circuit.

$$\sum_{v \in V} x_v = k$$

Such circuit can be built in polytime using adders



← The output is single boolean variable that determines whether indep set is valid.

The O/P of gate is satisfied if

(i) Indep set is of size k

(ii) No two adjacent vertices are part of Indep set.

Thus if circuit is satisfiable, this satisfying assignment corresponds to independent set

and mapping $(G, k) \leftrightarrow$ circuit is computable in polytime

$\boxed{\therefore \text{ Independent set } \leq_p \text{ Circuit-SAT}}$

Ques) Show that Max^m Independent set is polytime reducible to ILP.

Proof:- • Max^m Independent set (MIS)

I/P: An undirected graph $G = (V, E)$

O/P: largest subset $V' \subseteq V$ such that no two vertices in V' are adjacent i.e $(u,v) \in V'$ but $(u,v) \notin E$

• Integer linear programming (ILP):

I/P: A set of linear inequalities and an objective f^n with variables i.e A, B and objective f^n

O/P: The values of the variables that maximize or minimize the objective f^n while satisfying constraints i.e x such that $Ax \le B$

*** Reduction from MIS to ILP.**

→ To represent MIS as ILP, we can use binary variables to model the selection of vertices in the independent set.

Step ①→ Let's define binary variable

$$x_i = \begin{cases} 1 & \text{if vertex } v_i \text{ is in indep set} \\ 0 & \text{otherwise} \end{cases}$$

Step ②:- Objective f^n

To maximize the no. of vertices in Indep. set;

$$\text{maximize} \quad Z = \sum_{i=1}^{|V|} x_i$$

Step ③:- Constraints (No two adjacent vertices)

For every edge $(v_i, v_j) \in E$, we cannot select both vertices simultaneously, so add constraint

$$x_i + x_j \le 1 \qquad x_i, x_j \in \{0, 1\} \ \forall_{i,j \in}$$

The above constraint gurantees that if $x_i = 1$ (vertex i is in IS) then $x_j = 0$ and vice versa, thus ensuring no two adjacent vertices are both selected.

*** Polytime reduction**

→ To convert an instance of MIS to ILP, we simply need to

① create a variables x_i for each vertex i ⇒ $O(V)$

② set up objective f^n $\sum_{i \in V} x_i$ ⇒ $O(V)$

③ for each edge $(i,j) \in E$ add constraint $x_i + x_j \le 1$ ⇒ $O(E)$

∴ overall time comp $= O(V + V + E)$
$\simeq O(V + E) \simeq$ polytime

∴ $\boxed{MIS \le_p ILP}$