We will discuss SAT, the ultimate problem in NP.

Polynomial time reduction from X to Y

→ a polynomial time algorithm
→ transform given input for X into an input for Y ($i \to i'$)
→ Solving Y on i' yields same answer as solving X on i.

This implies
→ Y is at least as hard as X (X is at most as hard as Y)
→ if Y can be solved in polynomial time, then so can X

The ULTIMATE problem in NP:

Imagine a problem in NP, such that any other problem in NP can be reduced to it.

Such problem is called NP complete problem.

We can have many such problems.

If we find a polynomial time algorithm for a np-complete problem, then it means there's a poly. time algo for all np problems.

INTERESTING: If V.C. is np-complete, then clique is also np complete because
→ clique is in np.
→ V.C. can be reduced to clique

How do we prove NP completeness?

2 ways
→ Take another np complete problem & reduce it to the said problem
→ Prove the said problem is in NP and also that any other np problem can be reduced to it.

this means we need a np-complete problem in the first place...

The first NP complete problem has already been found.

→ SAT (cook levin theorem)
↳ boolean satisfiability)

BOOLEAN SATISFIABILITY (SAT) Problem

i/p: a boolean formula with n variables

eg: $\cancel{x_1 \bar{x}_2 x_3 + x_3 x_2 + \bar{x}_1 x_4 + \bar{x}_2 \bar{x}_4}$

$(x_1 + x_2) \wedge \cdot (\bar{x}_2 + x_3) \cdot (x_3 + \bar{x}_1) \cdot (\bar{x}_1 + x_2 + x_4)$

o/p: Yes, if a ~~combination~~ soln exists such that the formula is equal to true, No, otherwise.

eg: $(x_1 \lor x_3) \wedge (x_1 \wedge (x_2 \lor \bar{x}_3)) \wedge (\bar{x}_2 \lor \bar{x}_3) \wedge (\bar{x}_1 \lor \bar{x}_2)$

↳ Yes (x_1 = true, x_2 = false, x_3 = false)

We will assume the length of the formula is polynomial in n.

Cook - Levin Theorem

- Any problem in NP can be solved in polynomial time by using ND-RAM
- There must be a polynomial time algorithm (using if better)
- IDEA! Show that any such algorithm can be encoded as a boolean formula.

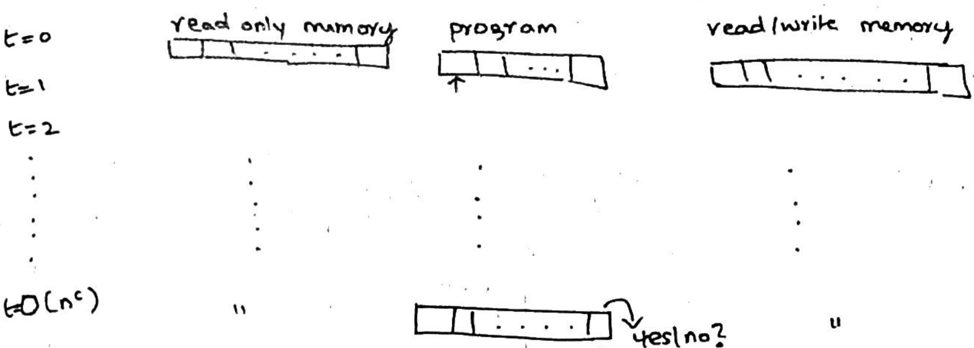
Detailed: Given any decision problem in NP, construct a ND. polynomial time algorithm. ~~that runs on ND ram~~

Then for each input to that ~~machine~~ algorithm, build a boolean expression which computes ~~that~~ whether ~~that~~ the specific algorithm runs and outputs Yes.

The satisfiability of the boolean expression is equivalent to asking whether or not the algorithm will answer Yes.

How do we encode an algorithm as a boolean formula?

Snapshots of execution!



At each time step we have a snapshot.

→ Each snapshot has size polynomial in n.

→ All snapshots together have polynomial size.

rom → polynomial

program → constant

r/w mem → polynomial (c modifications in each time step, polynomial time steps)

Consider boolean formulas of type/form

$$(x_1 \vee x_2 \vee x_3) \wedge$$
$$(\bar{x}_1 \vee \overline{(x_2 \vee x_3)}) \wedge$$
$$(\bar{x}_2 \vee \overline{(x_1 \vee x_3)}) \wedge$$
$$(\bar{x}_3 \vee \overline{(x_1 \vee x_2)})$$

→ Lets call this F_3

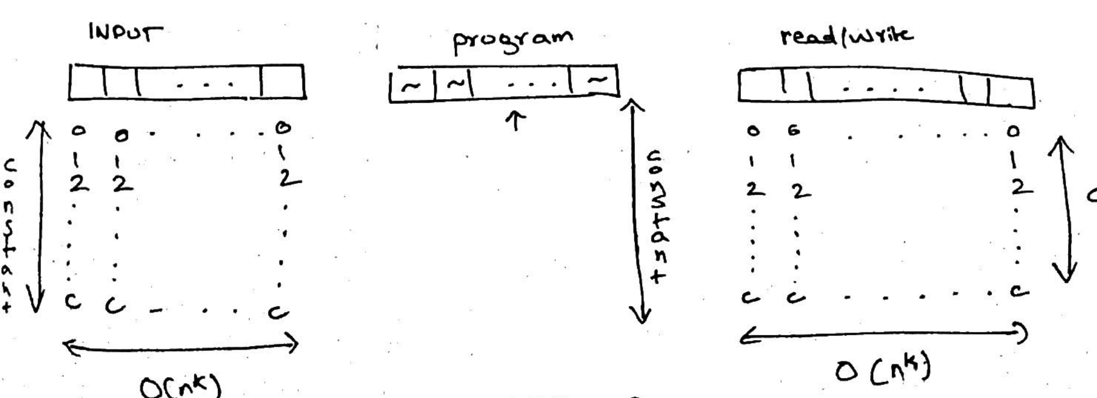
→ has several satisfying assignments

→ can be satisfied if ~~only~~ exactly one variable is true

→ size $O(n^2)$ for n variables.

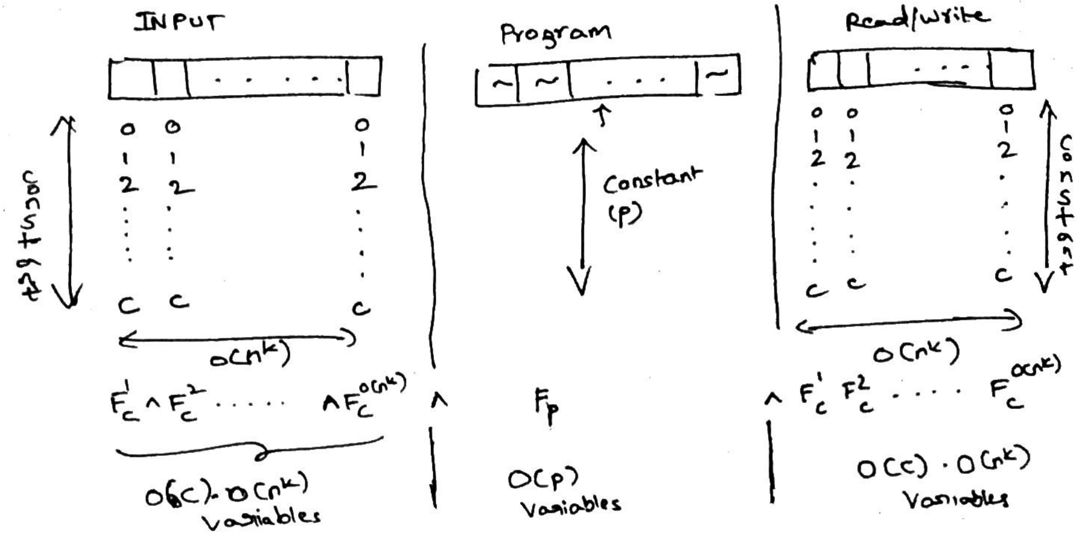
We can write a boolean formula if and only if one of the variables is true.

~~snapshots~~



all possibilities for a snapshot...

For each memory cell, we can use the previously discussed boolean formula.



each memory cell can' take of $\{0, 1, \ldots c\}$ values
→ can be described with F_c

not the same!

$$\underbrace{F_s = \left(F_c^1 \wedge F_c^2 \wedge \ldots \wedge F_c^{O(n^k)}\right) \wedge (F_p) \wedge \left(F_c^1 \wedge F_c^2 \wedge \ldots \wedge F_c^{O(n^k)}\right)}_{\text{size in polynomial of } n}$$

$F_c \Rightarrow O(c^k) \rightarrow O(c)$

↳ $O(n^k)$ such formuals

$\Rightarrow O(n^k)$ size

Every satisfying assignment of F_s presents a potential snapshot.

$t = 0$ → ~~~ (boolean formula)

$t = 1$

⋮

$t = O(n^k)$

① Each timestep is a snapshot

② First snapshot represents memory of RAM at $t=0$

③ Ensure snapshots fit together

④ Ensure boolean formula be satisfied only if algorithm returns <u>Yes</u>

① ✓ DONE

② ✓ DONE, trivial

④ make sure the line that returns Yes is executed at some point

③ ???

$t=0$

$t=1$

$\vdots$

$t = O(n^k)$

$\{ (\ \ ) \wedge (\ \ ) \wedge (\ \ ) \wedge \ldots (\ \ )$

$\vdots$

Solve SAT for this.

→ In RAM, knowing snapshot at t, we know the snapshot at $t+1$

→ In NDRAM, this is not true when if-better is encountered.

<u>RULES</u>

$$\text{time } t \xrightarrow{\text{given line of code and variables in line}} \text{time } t+1$$

k variables

c values for each variable.

$O(k \cdot c) \Leftrightarrow O(1)$

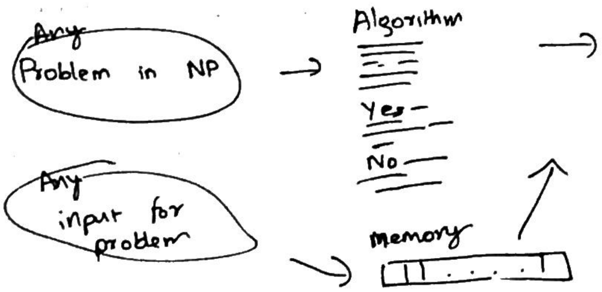
eg: if ... then ... as a boolean formula

$\bar{x} \vee x_2 \iff$ if x_1 then x_2

what about if-better?

t t+1

go to
either one line
of code or the
other
(or of two boolean formulae)



Boolean formula

- encode snapshots
- ensure snapshot 1 is correct
- ensure s.s at t and t+1 are connected.
- only if algorithm returns yes

SAT returns yes ⇒ algorithm returned yes
SAT returns no ⇒ algorithm returned no

~~If we can~~

We proved any problem in NP can be reduced to SAT (in polynomial size & time)

Fun fact : it's 6:09 in the morning