Algorithms

Complexity & NP-Hardness

Recap:

- HW due
- Next HW - due in one week

~~Next time~~: NP-Hardness!
This time

Fundamental question:

Are there "harder" problems?

How do we rank?

- Polynomial
- ~~Exponential~~
- Unsolvable?

Undecidably:

Some problems are impossible to solve!

The Halting Problem: (Turing)

Given a program P and input I, does P halt or run forever if given I?

Output: True / False

(Utility should be obvious!)

Note: Can't just simulate P on I. Why?

Thm [Turing 1936]:

The halting problem is undecidable.

(That is, no such algorithm can exist.)

Proof: by contradiction - suppose we have such a program h:

$$h(P, I) = \begin{cases} \text{True} & \text{if } P \text{ halts on } I \\ \text{False} & \text{otherwise} \end{cases}$$

Need a contradiction now...

Now define a program g that uses h:

g(X) := if h(X, X) = False
return False
else (X halts on input X)
loop forever

(so X(X) loops)

The contraction: What does g(g) do?

Calls h(g, g): (testing if g(g) halts or loops)

If h(g, g) = true, that means g halts on input g

But then g(g) should run forever!

If h(g, g) = False, then g infinitely loops on input g.

But then g should return false on input g!

So... what next?

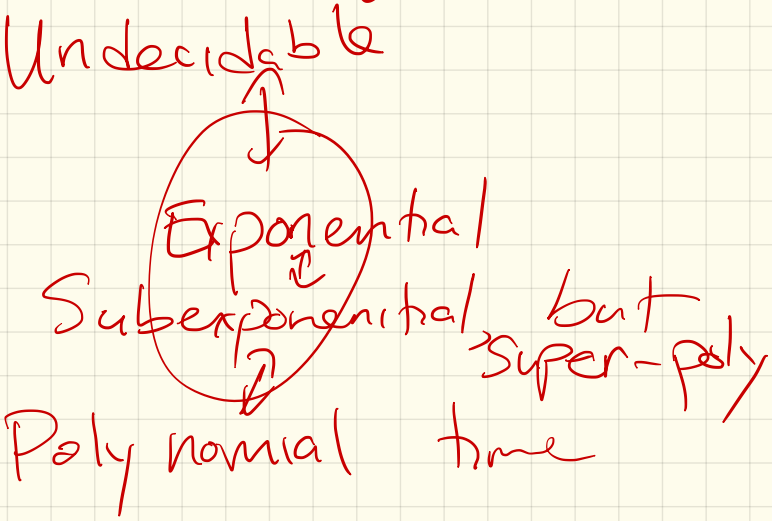
Clearly, many things are solvable in polynomial time.

Some things are impossible.

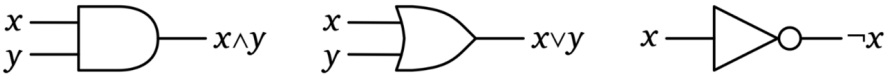
But - what is in between?

Idea:

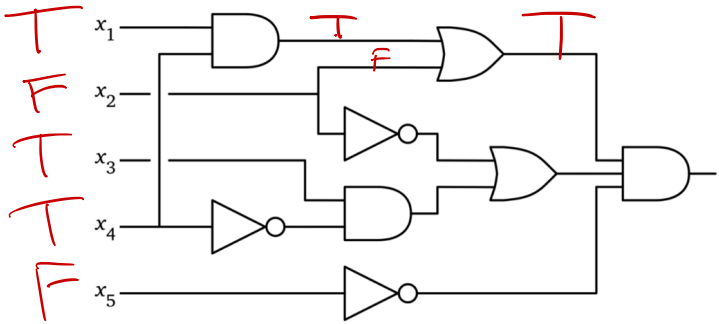
Polynomial Hierarchy:



The first problem found: Boolean circuits



An AND gate, an OR gate, and a NOT gate.

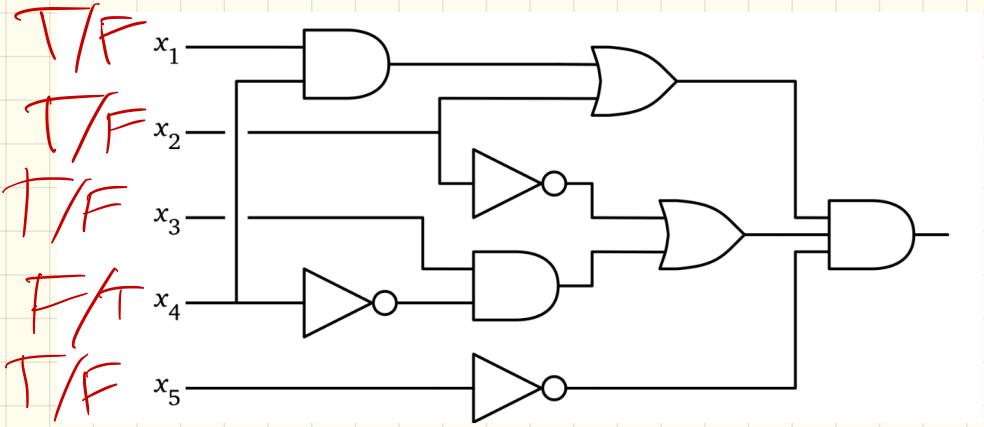


A boolean circuit. inputs enter from the left, and the output leaves to the right.

Given a set of inputs, can clearly calculate output in linear time (in # inputs + # gates).

How? Circuit is a tree → trace T/F values through gates starting from "leaves"

Q: Given such a boolean circuit, is there a set of inputs which result in TRUE output?



Known as CIRCUIT SATISFIABILITY (or CIRCUIT SAT)

Best known algorithm:

Try all 2^n inputs.

Track through gates & check if T/F in output

Running time:

$2^n(n+m)$

Note:

Might be a better way!

P, NP, & co-NP

$P \subseteq NP$

Consider only decision problems:
so Yes/No output

P: Set of decision problems that can be solved in polynomial time.

Ex: - Is x in the list? $O(n)$ or $O(\log n)$
- Is there a cut in G of size 100? ↳ F-F: $O(VE)$

NP: (Non-deterministic poly time) Set of problems such that, if the answer is yes & you hand me proof, I can verify/check in polynomial time.

Ex: Circuit-SAT: hand me inputs, I can check in $O(n+m)$ time

Co-NP: If answer is no, I can check that in poly time.

Dfn: NP-Hard

X is NP-Hard

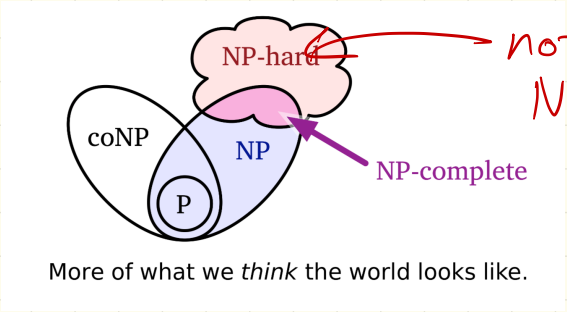
$\Leftrightarrow$

If X could be solved in polynomial time, then P=NP.

So if <u>any</u> NP-Hard problem could be solved in polynomial time, then all of NP could be.

Cook-Levine Thm:

Circuit SAT is NP-Hard.



NP-Complete:

Why?

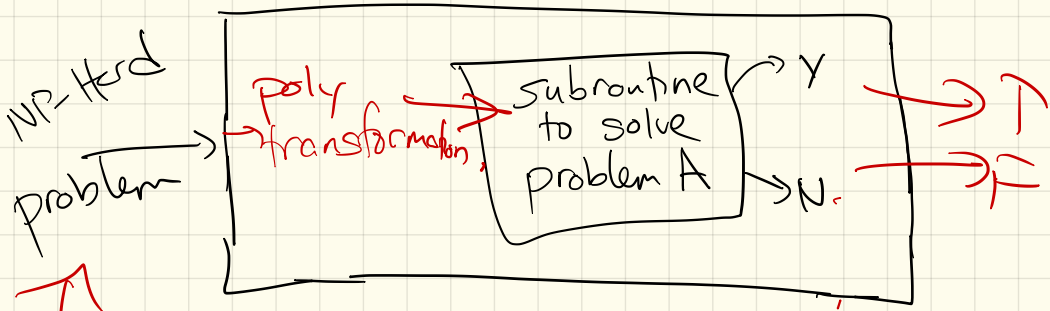
- in NP
- and NP-Hard

They mimic any Turing Machine using a circuit.

Just trust me. :)

To prove NP-Hardness of A:

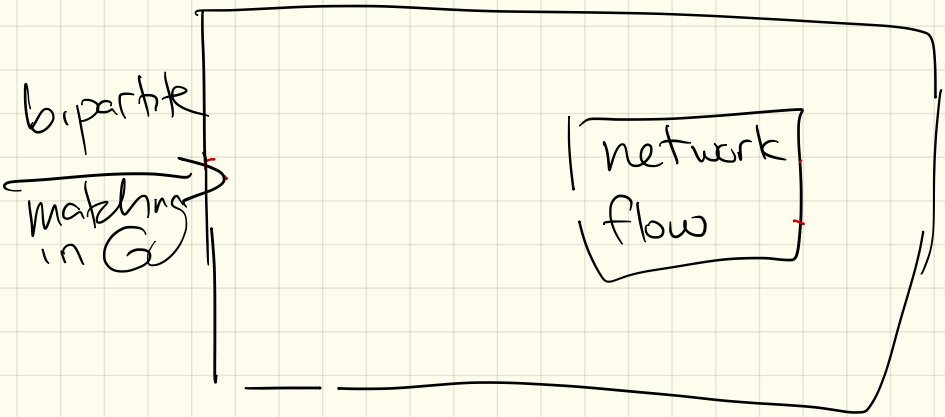
Reduce a known NP-Hard problem to A.



circuitSAT

If tranformation + subroutine for A is poly time, then could solve circuitSAT in that time.

We've seen reductions!



This will feel odd, though:

To prove a new problem is hard, we'll show how we could solve a known hard problem using new problem as a subroutine.

Why?

Well, if a poly time algorithm existed, than you'd also be able to solve the hard problem!

(Therefore, can't be any such solution.)

Other NP-Hard Problems:

SAT: Given a boolean formula, is there a way to assign inputs so result is 1?

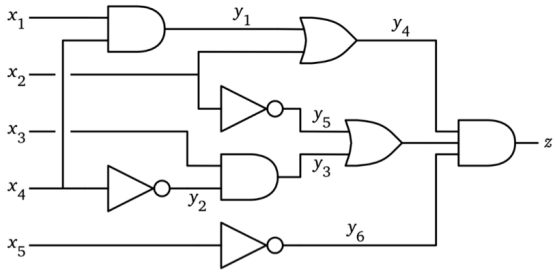
Ex: $(a \vee b \vee c \vee \bar{d}) \Leftrightarrow ((b \wedge \bar{c}) \vee \overline{(\bar{a} \Rightarrow d)} \vee (c \neq a \wedge b)),$

- n variables,
- m clauses

In NP!

Thm: SAT is NP-Hard.

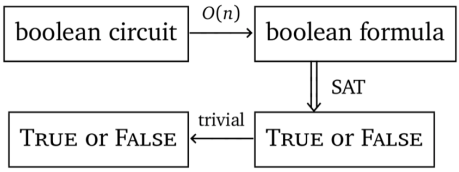
pf: Reduce CIRCUITSAT to SAT:



$$(y_1 = x_1 \wedge x_4) \wedge (y_2 = \overline{x_4}) \wedge (y_3 = x_3 \wedge y_2) \wedge (y_4 = y_1 \vee x_2) \wedge$$
$$(y_5 = \overline{x_2}) \wedge (y_6 = \overline{x_5}) \wedge (y_7 = y_3 \vee y_5) \wedge (z = y_4 \wedge y_7 \wedge y_6) \wedge z$$

A boolean circuit with gate variables added, and an equivalent boolean formula.

So our reduction:



$$T_{\text{CSAT}}(n) \leq O(n) + T_{\text{SAT}}(O(n)) \implies T_{\text{SAT}}(n) \geq T_{\text{CSAT}}(\Omega(n)) - O(n)$$

Tomorrow: More reductions!