{CHAPTER 7, [7.1—7.4]}

"RUNNING TIME"

TIME COMPLEXITY

CONSIDER ONLY COMPUTABLE FUNCTIONS.

→ ALWAYS HALTS / DECIDABLE.

CONSIDER ONLY DETERMINISTIC MACHINES.

That "GUESSING THE RIGHT THING"
or "TRY <u>ALL</u> POSSIBILITIES"
is suspect!

CONSIDER SOME INPUT, W.

Just count the transitions.

CONSIDER ALL INPUTS of SIZE N.

What is the MAXIMUM time that
a Turing Machine might take.

Our goal: Find a function of N
to describe running time.

$f(N) = \ldots$

Often, the function can be ugly!

$$f(N) = 17N^3 + 5N^2 + 3\,\mathrm{Log}\,N + 29$$

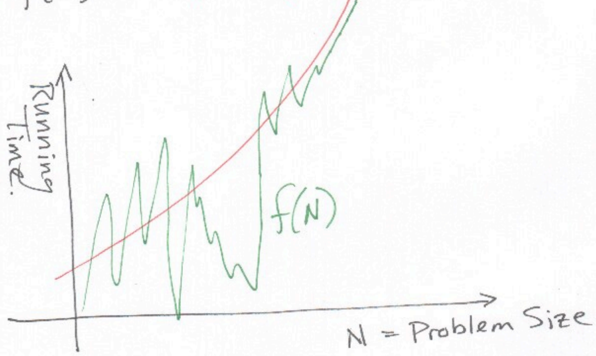
For large values of N, we only care about N^3

We want the

ASYMPTOTIC UPPER BOUND.

The "ORDER" (or "BIG-O") Notation:

$$f(N) = O(N^3)$$



Also: Ignore constant factors.

$\Rightarrow$ Ignore $17N^3$

LET M BE A DETERMINISTIC TURING MACHINE THAT ALWAYS HALTS.

LET n BE THE SIZE OF AN INPUT.

DEFN

THE "TIME COMPLEXITY" (i.e., the RUNNING TIME) OF M IS A FUNCTION f.

$f(n)$ = THE MAXIMUM NUMBER OF STEPS THAT M TAKES ON ANY INPUT OF SIZE n.

NOTE

"SIZE OF INPUT" usually means the LENGTH OF THE INPUT.

... But may sometimes mean something else, such as

- Number of nodes in a graph.
- Number of rules in a CFG.

etc.

BIG-O NOTATION

$17N^3 + 5N^2 + 3N + 29$

$O(N^3)$

FOR POLYNOMIAL FUNCTIONS.

* TAKE THE HIGHEST ORDER TERM
* IGNORE THE COEFFICIENT.

$f(n) = 17n^3 + 5n^2 + 3n + 29$

We say: $f(n) = O(n^3)$

Also:

$$f(n) = O(n^4)$$
$$= O(n^5)$$
$$= O(2^n)$$

DEFN.

Let $f(n)$ be some running time function of interest.

We say $f(n) = O(n^3)$

if, for all $n \geq$ some value (n_0)

(i.e., for all n large enough)

f (looks / behaves) like n^3, ignoring constant factors.

MORE PRECISELY:

$$f(n) = O(g(n))$$

IF $\exists\, c$ and $\exists\, n_0$ such that

$$f(n) \leq c \cdot g(n) \quad \text{for all } n \geq n_0.$$

TYPICAL COMPLEXITY CLASSES

$$\left.\begin{array}{l} N = \text{linear} \\ \log N \\ N \log N \\ N^2 \\ N^k \end{array}\right\} \text{POLYNOMIAL TIME}$$

2^N EXPONENTIAL

$O(N)$ = linear time algorithms

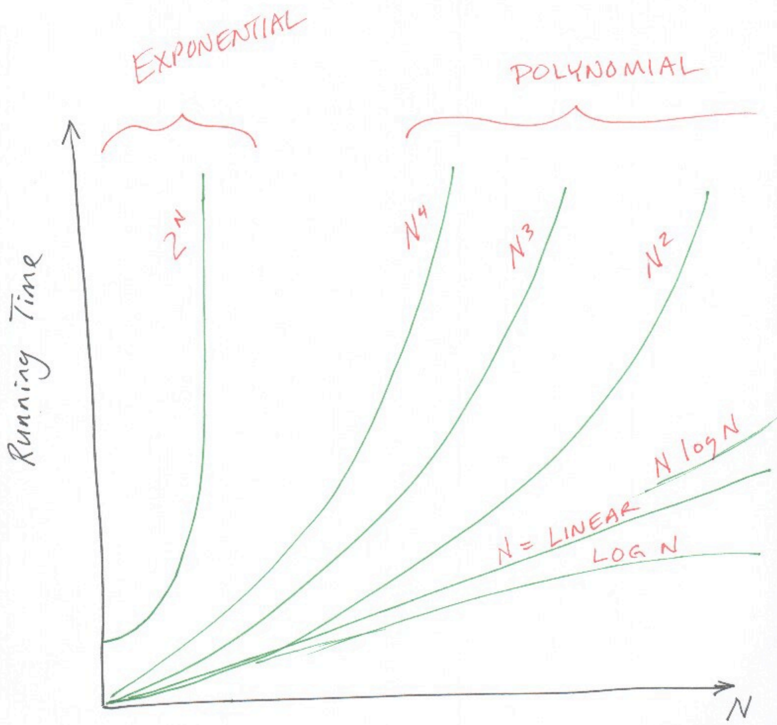
$O(N \log N)$

$O(N^2)$

$O(N^3)$

$O(N^4)$

$O(2^N)$


EXPONENTIAL
POLYNOMIAL
Running Time
2^N
N^4
N^3
N^2
N log N
N = LINEAR
LOG N
N

Q: Why aren't there many $O(\log N)$ algorithms?

A: The input has size n. Just to read all the input requires $O(n)$

TIME COMPLEXITY CLASSES

TIME(n)

The set of all languages/problems that can be DECIDED in $O(n)$ time.

TIME(n^2)

... that can be DECIDED in $O(n^2)$ time.

TIME($n \log n$) ... in $O(n \log n)$

TIME(n^3) ... in $O(n^3)$

TIME(2^N) ... in exponential time.

etc.

NOTE:

$$\text{TIME}(n) \subset \text{TIME}(n \log n) \subset \text{TIME}(n^2) \subset \text{TIME}(n^3) \subset \text{TIME}(n^k) \subset \text{TIME}(2^N)$$

EXAMPLE

WANT AN ALGORITHM TO DECIDE

$$\{ 0^k 1^k \mid k \geq 0 \}$$

ALGORITHM

INPUT: w

- Scan input to make sure it is in the form $0^* 1^*$.

  n steps to scan.
  n steps to reposition to left end.
  2n steps $O(n)$

- Repeat While the tape contains at least one 0 and at least one 1...
  - Scan across tape and change a 0 to X and a 1 to X.

    $O(n)$ steps here.

  END

  $n/2$ repetitions.
  Whole loop takes $\frac{n}{2} \cdot O(n) = O(n^2)$

- If tape contains all X's then ACCEPT Else REJECT.

  $O(n)$ steps.

$$O(n) + O(n^2) + O(n) \implies O(n^2)$$

$\boxed{\text{So: } \{0^k 1^k \mid k \geq 0\} \in TIME(n^2)}$

But there is a better algorithm!

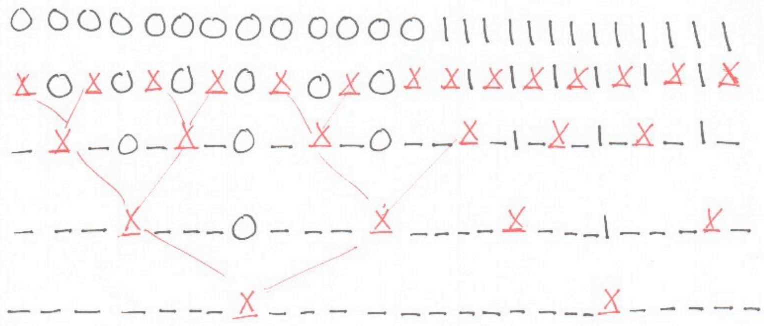
- Scan input to make sure it is in the form $0^* 1^*$ ← $O(n)$

- Repeat while the tape contains at least one 0 and at least one 1...
  - Scan tape to see if number of 0's plus number of 1's is ODD or EVEN
    $O(n)$
  - If ODD then REJECT. ← $O(1)$
  - Scan across the entire tape.
    Cross off every other 0, starting with the first 0.
    Cross off every other 1, starting with the first 1.
    $O(n)$
- END ← Number of reps is $1 + \log_2 n$

$(1 + \log_2 n) \cdot O(n) = O(n \log n)$

If no 0's and no 1's remain then ACCEPT, else REJECT ← $O(n)$

$\boxed{\text{So: } \{0^k 1^k \mid k \geq 0\} \in TIME(n \log n)}$



Cross off every other 0

Cross off every other 1

Repeat until nothing remains.

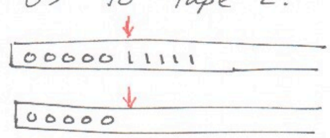
At each stage we should have the same number of 0's as 1's.

What about a different model of computation?

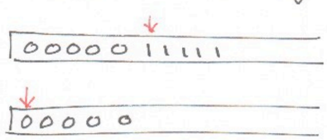
Assume we have multiple tapes.

ALGORITHM USING 2 TAPES.

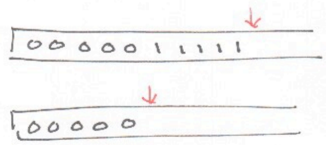
- Copy all 0's to tape 2. O(n)



- Reposition tape 2 to beginning. O(n)



- Scan both tapes simultaneously.
- Make sure both heads hit ␣ at the same time. O(n)



Theorem

For every multitape Turing machine algorithm that takes time $t(n)$, There is an equivalent single tape Turing machine that takes time $O(t^2(n))$.

Proof

In time $t(n)$, the longest the tapes can be is $t(n)$.
You can simulate the multitape algorithm on a machine with one tape.
Each step of the simulation can be done in $O(t(n))$ time.
To simulate the entire algorithm:

$$t(n) \cdot O(t(n)) = O(t^2(n))$$

Bottom Line

The model of computation matters!

However, the differences are "relatively small".

A polynomial-time algorithm will remain polynomial-time, regardless of the details of the model of computation!

AS LONG AS THE MACHINES ARE DETERMINISTIC!

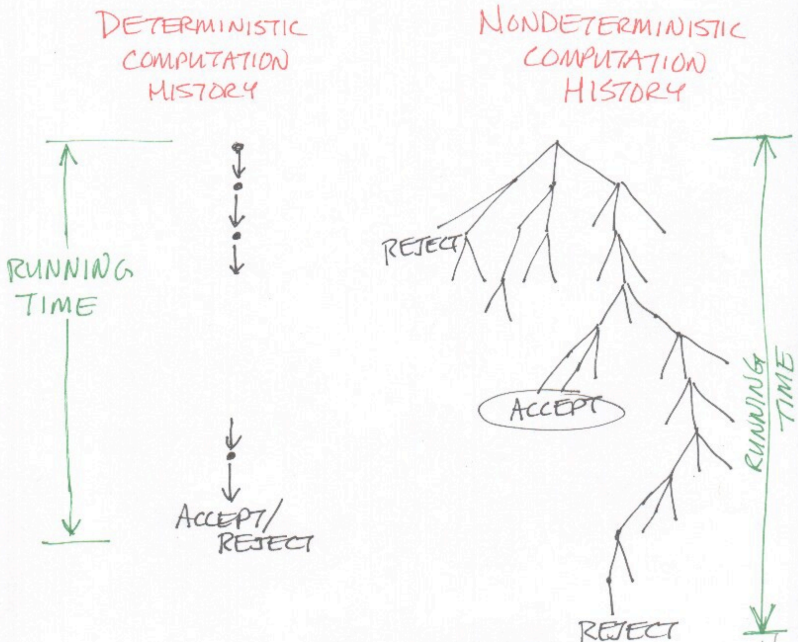
The class of Polynomial-time Problems seems quite ROBUST.

(Details of the computer don't matter.)

NON-DETERMINISTIC T.M.s.

RUNNING TIME:

The number of steps the TM uses on the longest branch of computation.



EVERY NONDETERMINISTIC TM CAN BE SIMULATED ON A DETERMINISTIC TM, USING EXPONENTIALLY MANY MORE STEPS.

NON DET TM

TAKES 419 STEPS ON INPUT w

DET SIMULATION

CAN BE DONE IN 2^{419} STEPS

NON DET TM

TAKES $O(N^2)$ TIME

DET SIMULATION

CAN BE DONE IN 2^{N^2} TIME

THE CLASS P

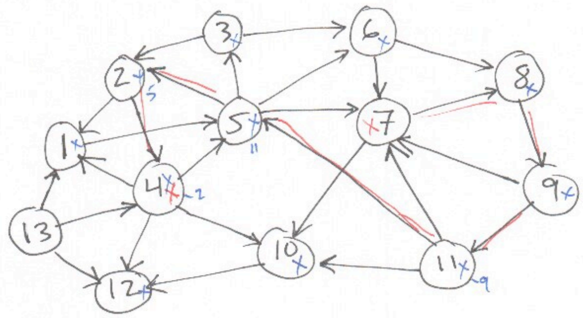
All reasonable deterministic models of computation are POLYNOMIALLY EQUIVALENT.

The class of languages that can be decided...
[i.e., the set of problems that can be solved...]
in POLYNOMIAL TIME on a DETERMINISTIC TURING MACHINE.

$$P = \bigcup_{k} \text{TIME}(n^k)$$

THE "PATH" PROBLEM

Given a directed graph G,
is there a path from one node
(s) to another (t)?



Is there a Path from 7 to 4?

PATH $\in$ P

PROOF

- PROVIDE AN ALGORITHM.
  USE A "MARKING" ALGORITHM
- SHOW IT'S RUNNING TIME.
  $O(m^2)$ where $m = \#$ of nodes

Theorem

Every context-free language is in P.

Proof

Provide an $O(n^3)$ algorithm.

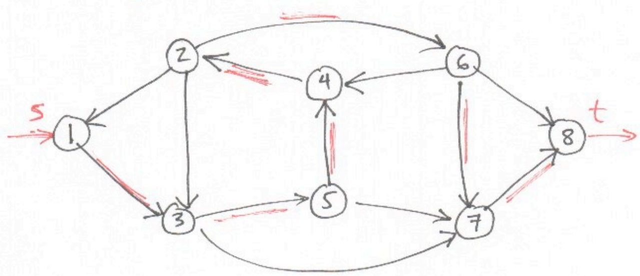
A "dynamic programming" algorithm.

- Use a table to store partial results.
- Avoid having to recompute things over and over.
- Build bigger results out of smaller results.

```
For i = 1 to N.
    Compute all results
        of size i
    Store each result.
    Make use of results
        of size < i.
End
```

THE HAMILTONIAN PATH PROBLEM

Given a directed graph, is there a path that goes through every node exactly once?



We are given the starting and ending nodes.

$$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph and there is a "HAMILTONIAN PATH" from } s \text{ to } t\}$$

13542678

EXPONENTIAL ALGORITHM

GENERATE ALL POSSIBLE PATHS.

1 2 3 4 5 6 7 8
1 4 3 2 8 7 5 6
⋮

TEST EACH PATH TO SEE IF IT IS LEGAL.

NOTE: This "test" can be done quickly! IN POLYNOMIAL TIME

This problem is in class NP.

It seems to require exponential time.

But given the answer, we can VERIFY it in polynomial time.

POLYNOMIAL VERIFIABILITY

Given a language A,
A "VERIFIER" is an algorithm
that is given some extra
information, "c", which it can
use to check (in polynomial time)
to verify that w is in A.

EXAMPLE: HAMPATH

Given a problem, such as
$w = \langle G, s, t \rangle$
is there a Hamiltonian Path?
EXPONENTIALLY HARD [Probably]
But the verifier algorithm is passed
some info: $c =$ "13542678"
and can then CONFIRM that
$w \in HAMPATH$
IN POLYNOMIAL TIME.

DEFINITION

A "VERIFIER" for a language A is an algorithm V where

$$A = \{ w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c \}$$

A "POLYNOMIAL-TIME VERIFIER" runs in polynomial time in the length of w.

A language is "POLYNOMIALLY VERIFIABLE" if it has a polynomial-time verifier.

The string c is called the "CERTIFICATE" (or "PROOF").

We don't care about the length of c; but note that a polynomial-time verifier does not have time to read a certificate that is longer than polynomial in the length of w.

DEFINITION

"NP" is the class of languages that have polynomial-time verifiers.

THEOREM

A language is in NP iff it is decided by some NONDETERMINISTIC POLYNOMIAL-TIME Turing Machine

Sometimes this is given as the definition of "NP".

PROOF

- Convert a Polynomial-time Verifier into an equivalent polynomial-time nondeterministic Turing Machine.

The TM:

INPUT: w (of length n.)

ALGORITHM:

- Nondeterministically guess string c (length atmost n^k)
- Run V on $\langle w, c \rangle$
- If V accepts, accept. Else Reject.

- Assume you have a polynomial-time non-deterministic TM. Construct a polynomial-time Verifier.

The Verifier:

INPUT: $\langle w, c \rangle$

ALGORITHM:

Simulate the Non-deterministic TM. Use c as a guide about which choice to make. at each step. If this branch accepts, then ACCEPT else REJECT.

P = The class of languages for which membership can be DECIDED quickly.*

NP = The class of languages for which membership can be VERIFIED quickly.

That is, given some information [the "certificate/proof"], you can quickly confirm that w is in the language.

* "quickly" means "in Polynomial time"

DEFINITION

$$\mathrm{NTIME}(t(n)) = \{ L \mid L \text{ is a language decided by an } O(t(n)) \text{ time nondeterministic T.M.} \}$$

TIME(n^2) = The set of languages that can be decided by a <u>DETERMINISTIC</u> T.M. in $O(n^2)$ time.

NTIME(n^2) = The set of languages that can be decided by a <u>NONDETERMINISTIC</u> T.M. in $O(n^2)$ time.

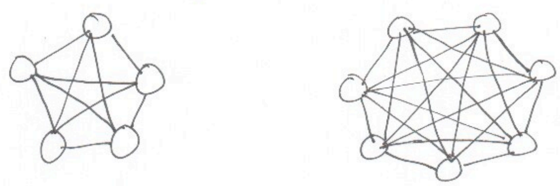
$$\mathrm{NP} = \bigcup_k \mathrm{NTIME}(n^k)$$

The "CLIQUE" Problem

Given an undirected graph...

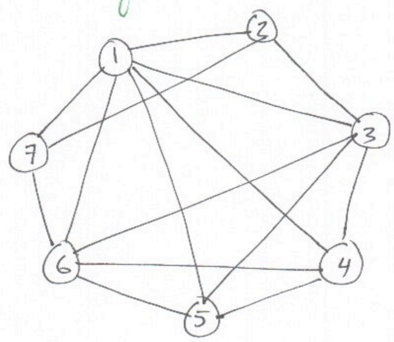
A "clique" is a set of nodes such that every node in the clique is connected to every other node in the clique.

A K-clique is a clique with K members.



A 5-CLIQUE   A 7-CLIQUE.

Does this graph contain a 5-clique?



CLIQUE = $\{ \langle G, k \rangle \mid$ G is an undirected graph with a k-clique $\}$

THEOREM

CLIQUE $\in$ NP

PROOF

- PROVIDE A POLYNOMIAL-TIME VERIFIER

  — OR —

- PROVIDE A POLYNOMIAL-TIME NONDETERMINISTIC TURING MACHINE.

THE CLASS "P"

THE CLASS OF LANGUAGES THAT CAN BE DECIDED...

[THE SET OF PROBLEMS THAT CAN BE SOLVED...]

... IN POLYNOMIAL TIME ON A DETERMINISTIC TURING MACHINE

THE CLASS "NP"

THE CLASS OF LANGUAGES THAT CAN BE DECIDED...

[THE SET OF PROBLEMS THAT CAN BE SOLVED...]

...IN POLYNOMIAL TIME ON A NONDETERMINISTIC TURING MACHINE.

UNSOLVED QUESTION:

$P = NP$
$P \subset NP$ } Which is it?

There are lots of problems known to be in NP.

- NONE of these problems can be solved in poly. time on a deterministic T.M.

These problems seem to require exponential time to solve.

EXPONENTIAL-TIME PROBLEMS

$$\text{EXPTIME} = \bigcup_k \text{TIME}(2^{n^k})$$

RESULTS

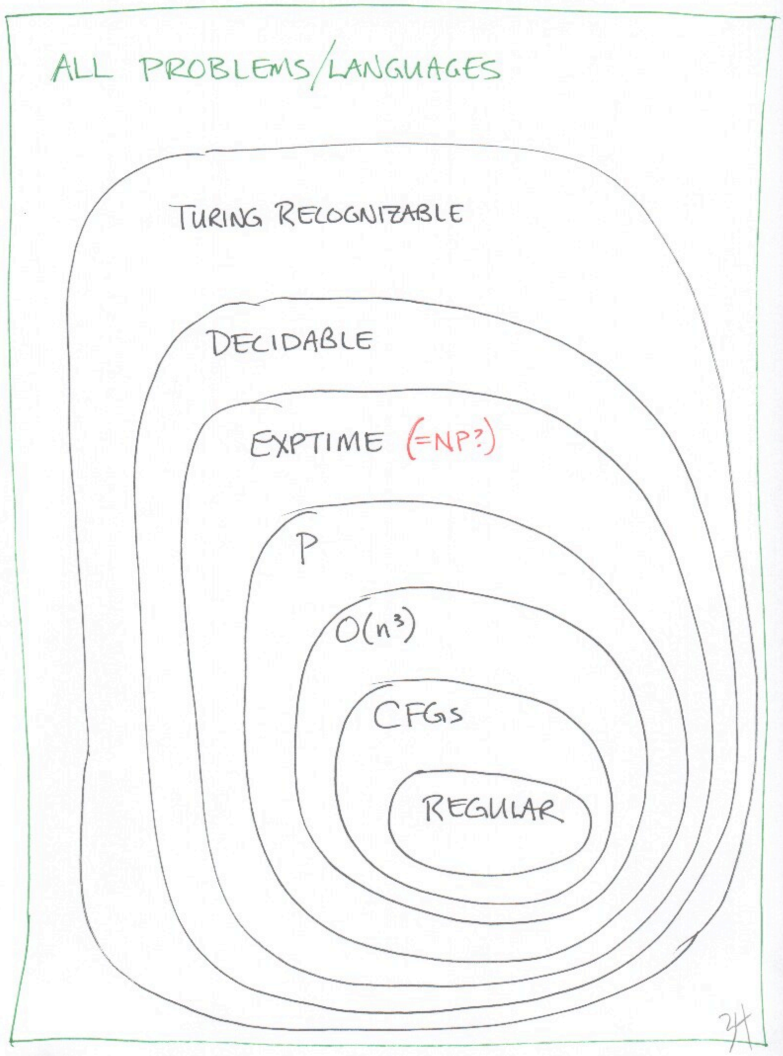
$$P \subseteq NP \subseteq \text{EXPTIME}$$

APPARENTLY:

$$P \subset NP = \text{EXPTIME}$$

BUT THIS IS ALSO POSSIBLE:

$$P = NP \subset \text{EXPTIME}$$


ALL PROBLEMS/LANGUAGES
TURING RECOGNIZABLE
DECIDABLE
EXPTIME (=NP?)
P
O(n³)
CFGs
REGULAR

NP-COMPLETENESS

- An interesting subset of NP problems. THE "NP-COMPLETE PROBLEMS."
- If a polynomial time algorithm is ever found (on a deterministic machine) for any "NP-Complete" problem, then P=NP follows!

$\therefore$ And polynomial time algorithms exist for all problems in NP!

Many interesting problems are NP-Complete.

They seem to require exponential time.

THE SATISIFIABILITY Problem "SAT"

Boolean variables; $x_1, x_2, x_3, \ldots$
TRUE, FALSE

Boolean operations: $\wedge$ $\vee$ $\neg$ — $\neg x = \bar{x}$

Boolean formulas, e.g.,

$$\phi = (\bar{x} \wedge y) \vee (\bar{y} \wedge z)$$

"Satisfiable" = If there is an assignment to the variables to make the formula true.

x = FALSE, y = TRUE, z = FALSE.

$$\text{SAT} = \{ \langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula} \}$$

$\boxed{SAT \in NP}$

Nondeterministically guess the solution (eg. X = FALSE, Y=TRUE, ...)

CHECK that it satisfies the Boolean formula in Polynomial time.

THEOREM

$SAT \in P$ iff $P = NP$.

OR EQUIVALENTLY...

SAT is "NP-COMPLETE".

Finding a polynomial time algorithm to solve a Boolean formula on a deterministic machine, would:

- Prove that <u>ALL</u> problems in NP have polynomial time algorithms.
- Rock the world.

RECALL...

- The Turing Machine Acceptance Problem, $A_{TM} = \{\langle M, w\rangle \mid M \text{ accepts } w\}$
- A_{TM} is undecidable.
- We "REDUCED" A_{TM} to an instance of the POST CORRESPONDENCE PROBLEM.
- This proved that the PCP was undecidable.
- We showed how to simulate the execution of a TM with the tiles of a PCP instance.
- The "computation history" was a sequence of "configurations."
- Finding a solution to the PCP was equivalent to finding an accepting computation history.

PROOF That SAT $\in$ P iff P=NP

- A problem is in NP if there is a NONDETERMINISTIC TURING MACHINE that will solve it in Polynomial time.
- Got a Problem? $\langle N, w \rangle$

  A non-det T.M.

  An input.
- Convert it into an instance of the SAT problem.*

  (A huge Boolean formula)
- Do this conversion in Polynomial time.
- If you can solve this SAT problem in Polynomial time, i.e. if SAT $\in$ P,

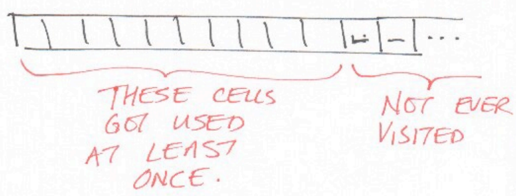
  Then, you can solve any problem in NP in Polynomial time.

* Such that, there is a branch in the Nondeterministic computation that ACCEPTS IFF the Boolean Formula is SATISFIABLE.

SPACE COMPLEXITY

How to measure?

The number of cells on the tape that we visit.



THE CLASS P-SPACE

QUESTION: What is the relationship between P ~~[illegible]~~ AND PSPACE?

- An algorithm that uses 30 tape cells must use at least 30 time steps.
- An algorithm that uses 30 tape cells may use many more steps.

$$\boxed{P \subseteq PSPACE}$$

Most problems are in NP

BUT...

There are problems in PSPACE for which there is no known NP algorithm!

Game

- 2 Players; they alternate.
- Each says the name of a geographic place.
- Kids play this in the car.

player 1: Portland
player 2: Denver
player 1: Rio

Until one player gets stuck.

- There is a list/dictionary of valid words. Each word can only be used once.

The Problem: Given the dictionary, can the 1st player win if he chooses carefully?

AND-OR TREE (MIN-MAX SEARCH)

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 \exists x_5 \cdots$$

Non-determinism doesn't seem to help trim the search time.

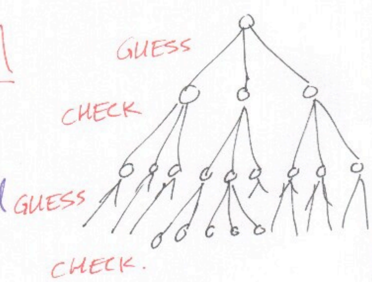
- Guess a good move for me.
- Check all his possible moves.
- Guess another good move.
- Check all his possible moves.

⋮

Nondeterminism helps here

But does not help here.

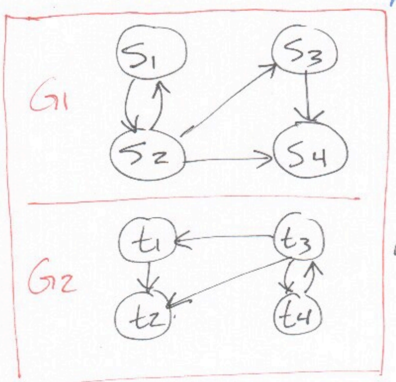
$\boxed{\text{A P-SPACE ALGORITHM}}$

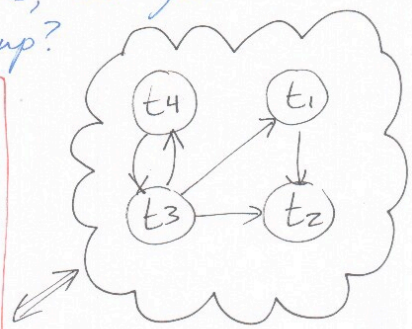


- This is a Search of a tree.
- The tree is exponential in size.
  - ⟹ We cannot store the tree.
- Do a depth first search of this tree.
- Time taken to search the tree: EXPONENTIAL.

GRAPH ISOMORPHISM

Given two graphs, can you match them up?





BTW: This problem is <u>NOT</u> NP-complete

PROBLEM:

Are 2 graphs isomorphic?

This problem is in NP.

Given an answer / correspondence, it can be checked in Polynomial time.

Are 2 graphs <u>NOT</u> isomorphic?

This problem is NOT in NP.

There are N! different possible correspondences.

You have to check each of them.