TIME COMPLEXITY AND BIG-O NOTATION

"TIME COMPLEXITY"
= RUNNING TIME FOR PROGRAMS

- CONSIDER ONLY COMPUTABLE FUNCTIONS.
  → DECIDABLE (Always Halt)
- CONSIDER ONLY DETERMINISTIC MACHINES

That "guess the right thing" or "try all possibilities" is a questionable operation on a real machine.

- CONSIDER SOME INPUT, w.
  FOR TMs: JUST COUNT THE TRANSITIONS.
- CONSIDER ALL INPUTS OF SIZE N.

WHAT IS THE MAXIMUM TIME THE TURING MACHINE MIGHT TAKE?

<u>OUR GOAL:</u> FIND A FUNCTION OF N TO DESCRIBE THE RUNNING TIME.

$$f(N) = \ldots$$

Often, the function can be ugly!

$$f(N) = 17N^3 + 5N^2 + 3\,Log_2 N + 29$$

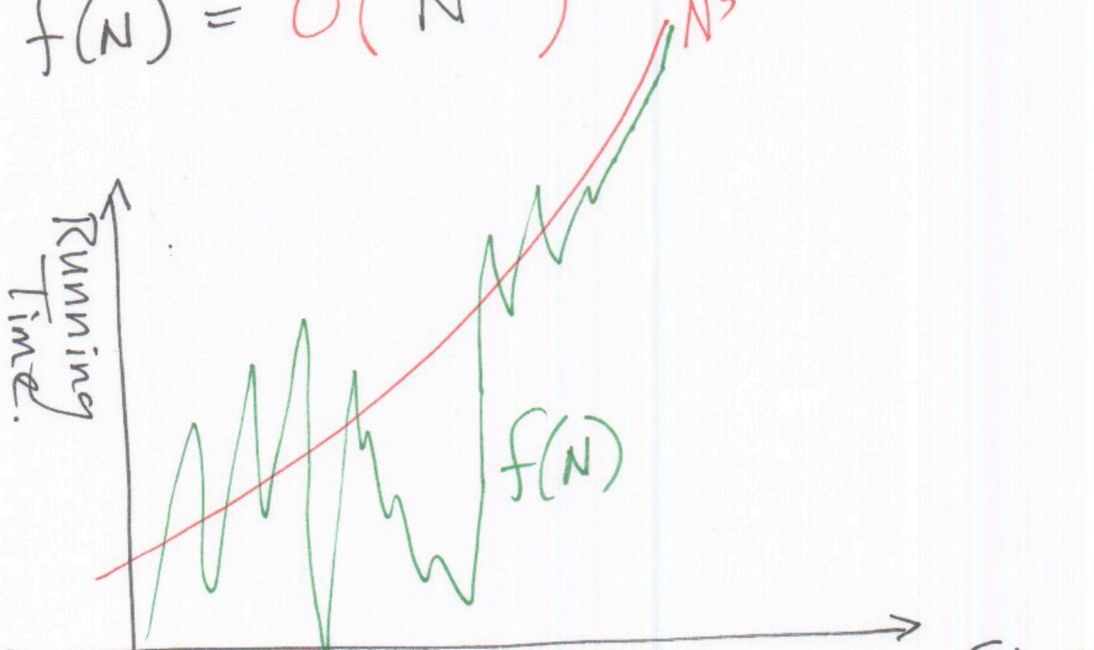
For large values of N, we only care about N^3

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:

$$\begin{aligned} f(n) &= O(n^4) \\ &= O(n^5) \\ &= O(2^n) \end{aligned}$$

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

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

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

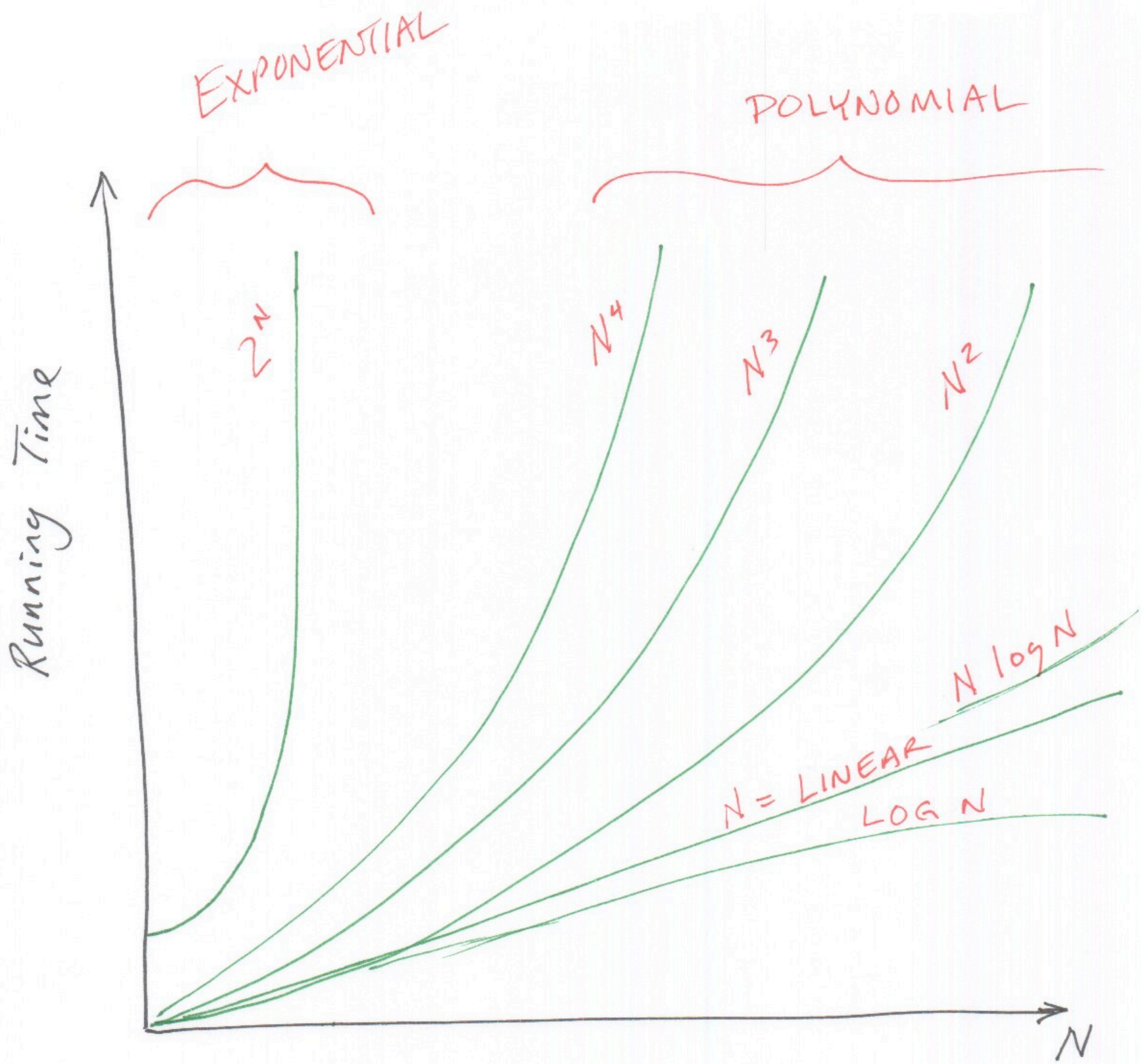
$O(N \log N)$

$O(N^2)$

$O(N^3)$

$O(N^4)$

$O(2^N)$


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

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

THE TIME COMPLEXITY OF AN EXAMPLE ALGORITHM

ALGORITHM TO DECIDE $\{0^k 1^k \mid k \geq 0\}$

- 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 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.

    $2n$ STEPS $O(n)$

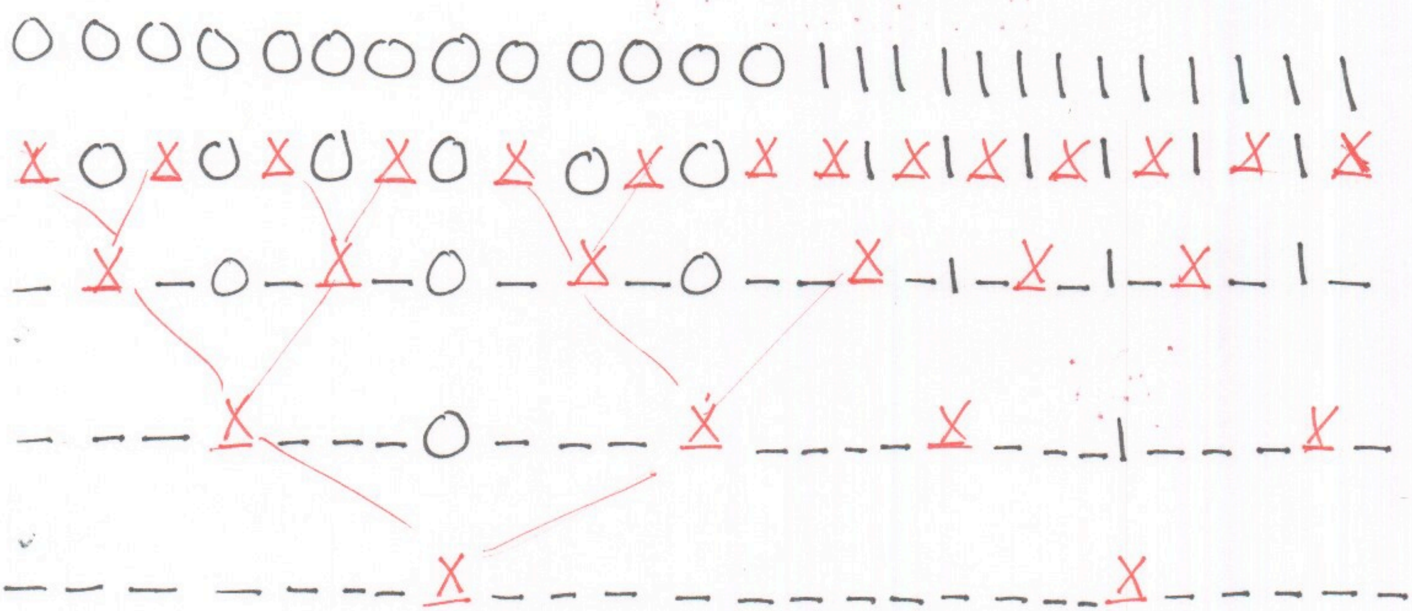
  END LOOP

  $n/2$ REPETITIONS
  WHOLE LOOP TAKE $\frac{N}{2} \cdot O(n)$ $O(n^2)$

- IF TAPE CONTAINS ALL Xs, THEN ACCEPT ELSE REJECT.

  n STEPS $O(n)$

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



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.

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

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. ← $O(n)$
    Cross off every other 0, starting with the first 0.
    Cross off every other 1, starting with the first 1.
- 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 ← $O(n)$

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

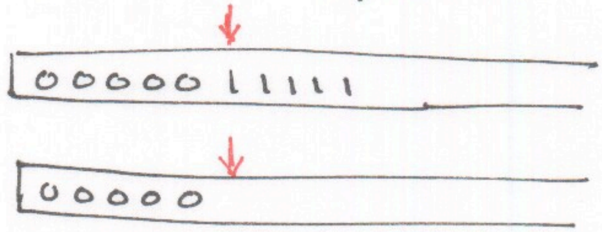
DIFFERENT MODELS OF COMPUTATION

What about a different model of computation?

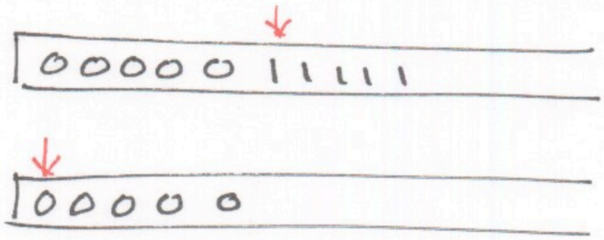
Assume we have multiple tapes.

$\boxed{\text{ALGORITHM USING 2 TAPES.}}$ $\{0^k 1^k \mid k \geq 0\}$

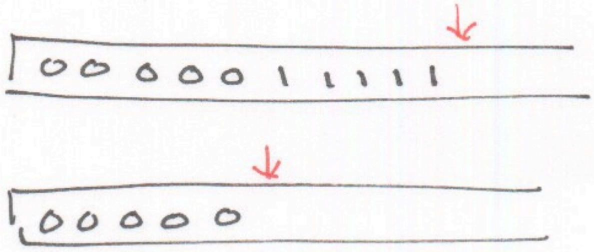
- Copy all 0's to tape 2. $O(n)$



- 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!

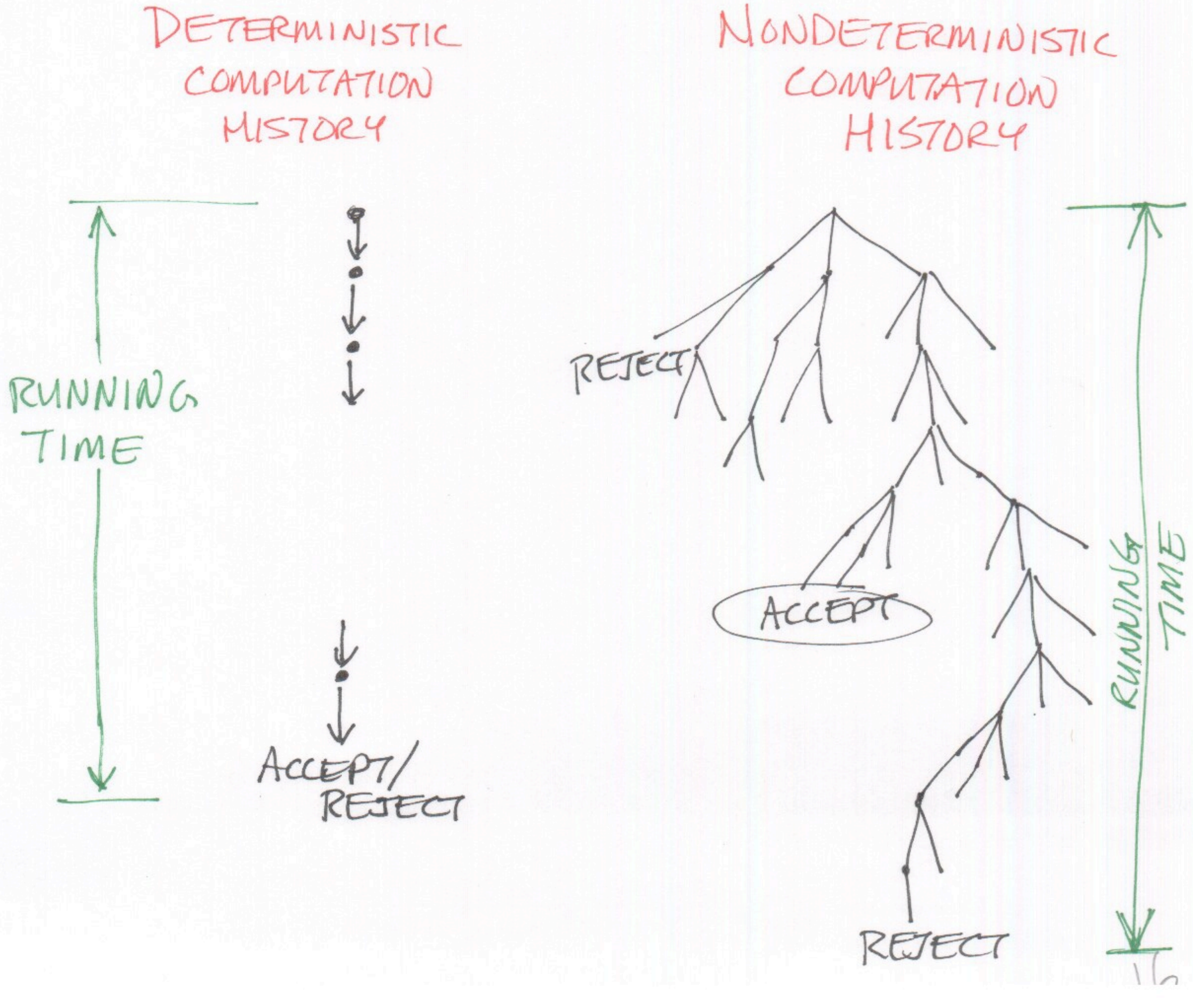
The class of Polynomial-time Problems seems quite ROBUST.

(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.

NONDET TM

TAKES 419 STEPS ON INPUT w

DET SIMULATION

CAN BE DONE IN 2^{419} STEPS

NONDET TM

TAKES $O(N^2)$ TIME

DET SIMULATION

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

THE COMPLEXITY CLASSES

P AND NP

THE CLASS P

All reasonable deterministic models of computation are POLYNOMIALLY EQUIVALENT.

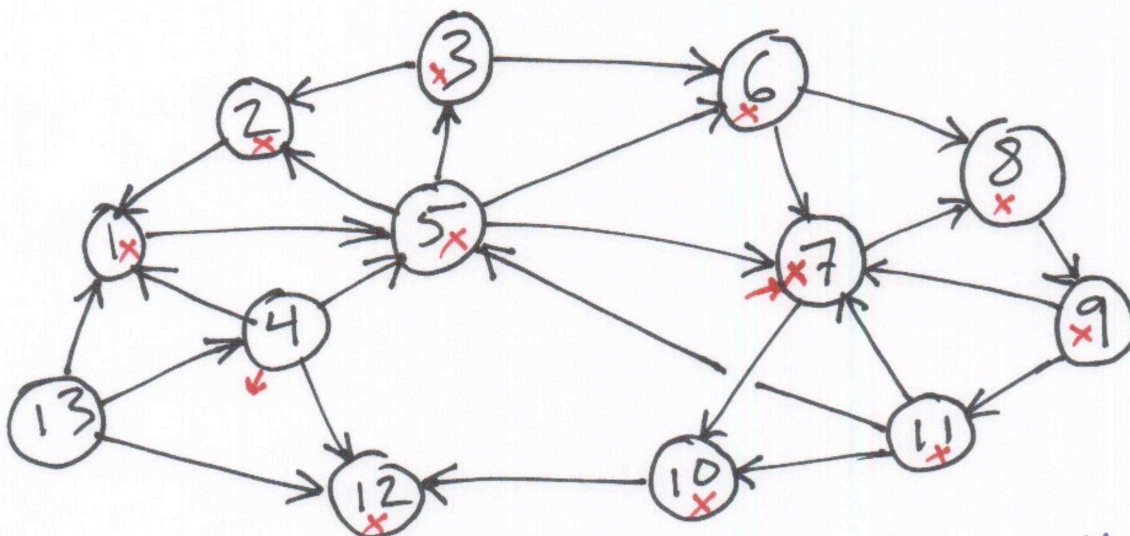
The class of languages that can be decided...
[i.e., the set of problems that can be solved...]
in POLYNOMIAL TIME on a <u>DETERMINISTIC</u> TURING MACHINE.

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

THE "PATH" PROBLEM

INPUT: A DIRECTED GRAPH **G**
AND TWO NODES, s and t…

IS THERE A PATH FROM s TO t?



IS THERE A PATH FROM 7 TO 4?

PATH ∈ P

PROOF

- PROVIDE AN ALGORITHM
- SHOW ITS RUNNING TIME.

USE A MARKING ALGORITHM

$O(m^2)$ where m = number 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.
    ~~[illegible]~~ Compute all results of size i
    Store each result.
    Make use of results of size $< i$.
End

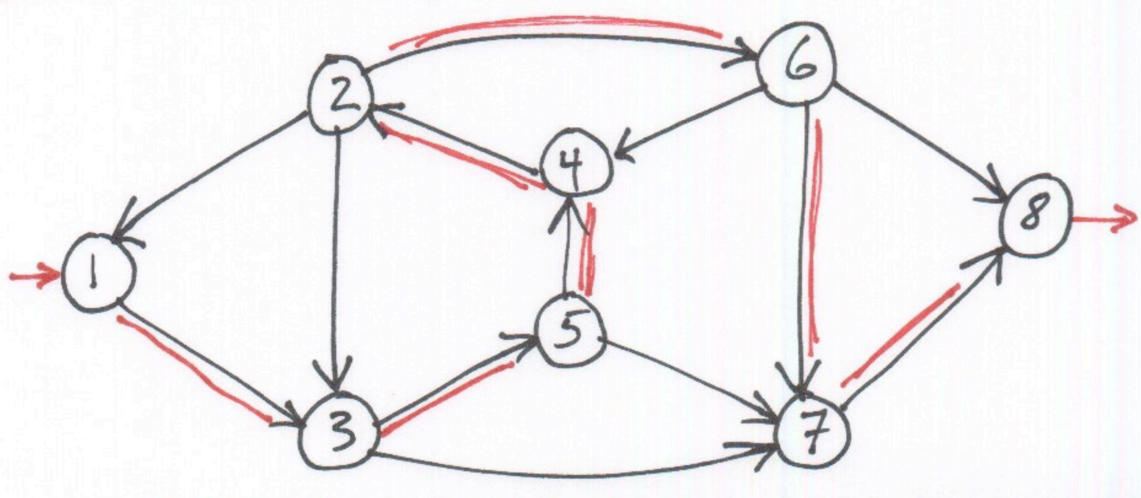
HAMPATH: THE "HAMILTONIAN" PATH PROBLEM

GIVEN A DIRECTED GRAPH, IS THERE A PATH THAT GOES THROUGH EVERY NODE EXACTLY ONCE?

$$HAMPATH = \left\{ \langle G, s, t \rangle \;\middle|\; \begin{array}{l} G \text{ is a directed} \\ \text{graph and there is} \\ \text{a "HAMILTONIAN"} \\ \text{Path from } s \text{ to } t \end{array} \right\}$$

WE ARE GIVEN THE STARTING AND ENDING NODES.

EXAMPLE: G, 1, 8



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.

<u>NOTE:</u> This "test" can be done quickly! IN POLYNOMIAL TIME

This problem is in class NP.

It <u>seems</u> to require exponential time.

But given the answer, we can <u>VERIFY</u> it in polynomial time.

DEFINITION OF NP

POLYNOMIAL VERIFIABILITY

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 dont care about the length of c; but note that a polynomial-time verifier ~~[illegible]~~ 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

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

$\mathrm{TIME}(n^2)$ = The set of languages that can be decided by a DETERMINISTIC T.M. in $O(n^2)$ time.

$\mathrm{NTIME}(n^2)$ = The set of languages that can be decided by a NONDETERMINISTIC T.M. in $O(n^2)$ time.

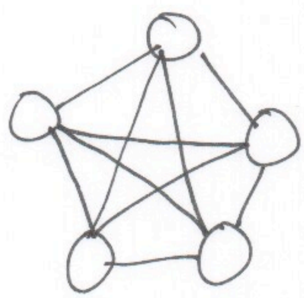
$$\mathrm{NP} = \bigcup_k \mathrm{NTIME}(n^k)$$

The "CLIQUE" Problem

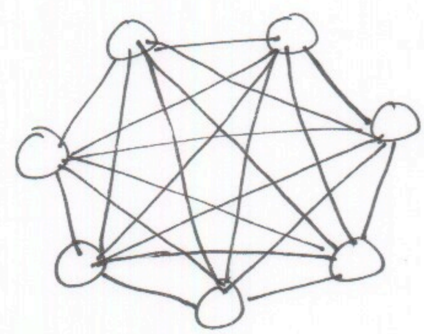
Given an undirected graph...

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.

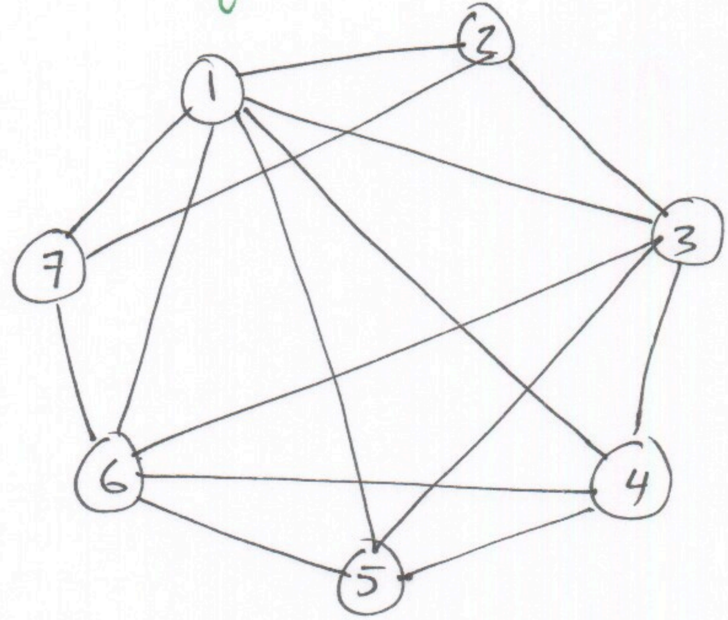


A 5-CLIQUE



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:

$$\left.\begin{array}{l} P = NP \\ P \subset NP \end{array}\right\} \text{Which is it?}$$

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

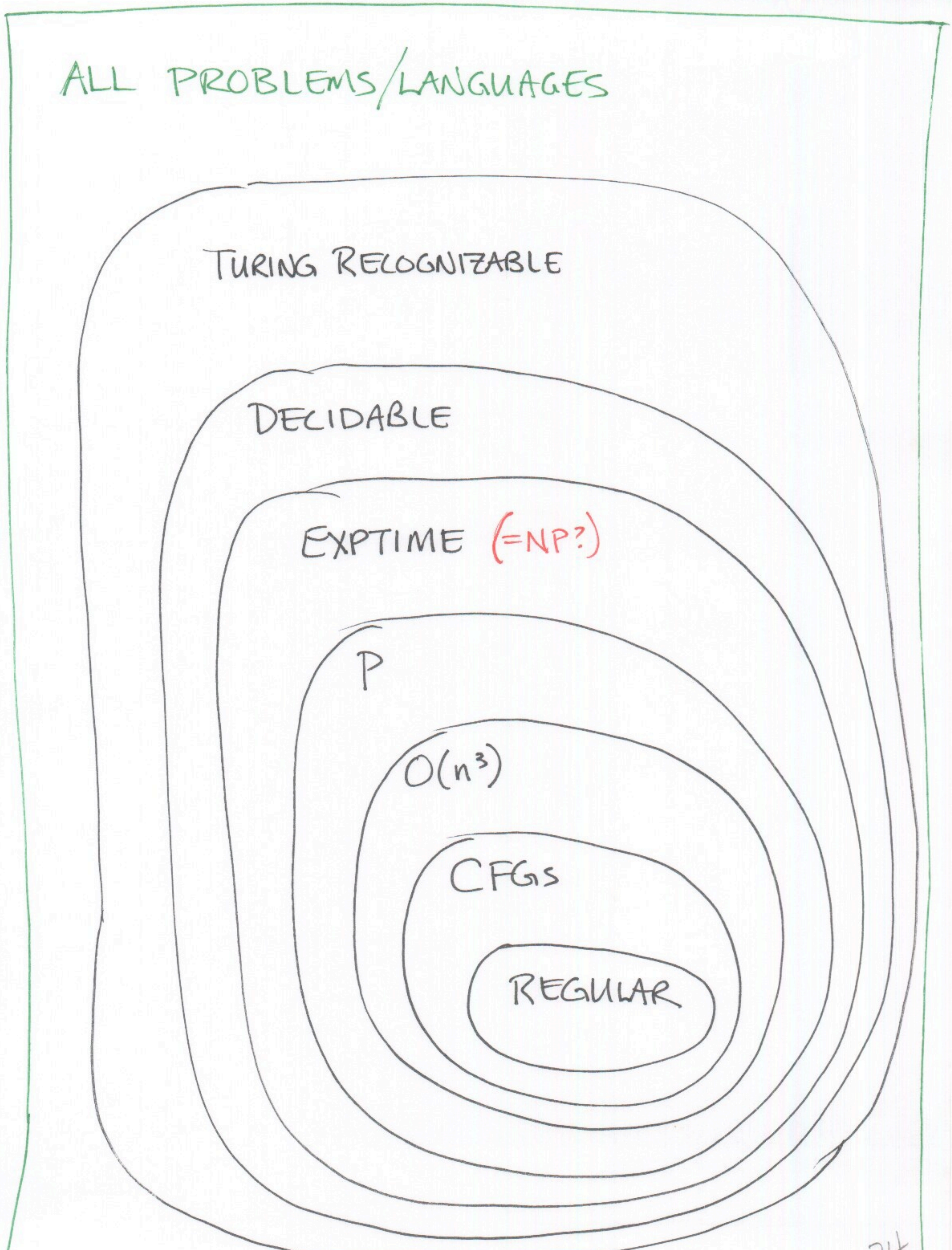
$$P \subseteq NP \subseteq \text{EXPTIME}$$

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-COMPLETE PROBLEMS

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!

∴ 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: $\wedge \quad \vee \quad \neg$

$\neg x = \overline{x}$

Boolean formulas, e.g.,

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

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

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

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

$\boxed{SAT \in NP}$

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.

PROOF THAT SAT IS NP-COMPLETE

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 <u>huge</u> 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.

THEOREM ("COOK-LEVIN")

SAT IS NP-COMPLETE

- ANY NP problem can be reduced into a SAT problem.
- This reduction can be done in poly-time.
- So if you can solve SAT in poly-time on a det. Machine, you can solve any problem in NP on a det. Machine in poly-time; i.e., P=NP.

GIVEN A PROBLEM IN NP...

GIVEN: N = A nondeterministic TM.

w = An input to that TM.

CONVERT IT INTO A BOOLEAN FORMULA, ϕ.

Such that ϕ is satisfiable iff N accepts w.

(iff N has an accepting computation history for w).

The accepting computation history on N can take at most n^k steps, for some k.

Therefore, it can use at most n^k tape cells.

Step 1 in the reduction:

$= n = |w|$

CREATE AN $N^K \times N^K$ "TABLEAU" TO MODEL THE ACCEPTING COMPUTATION HISTORY.

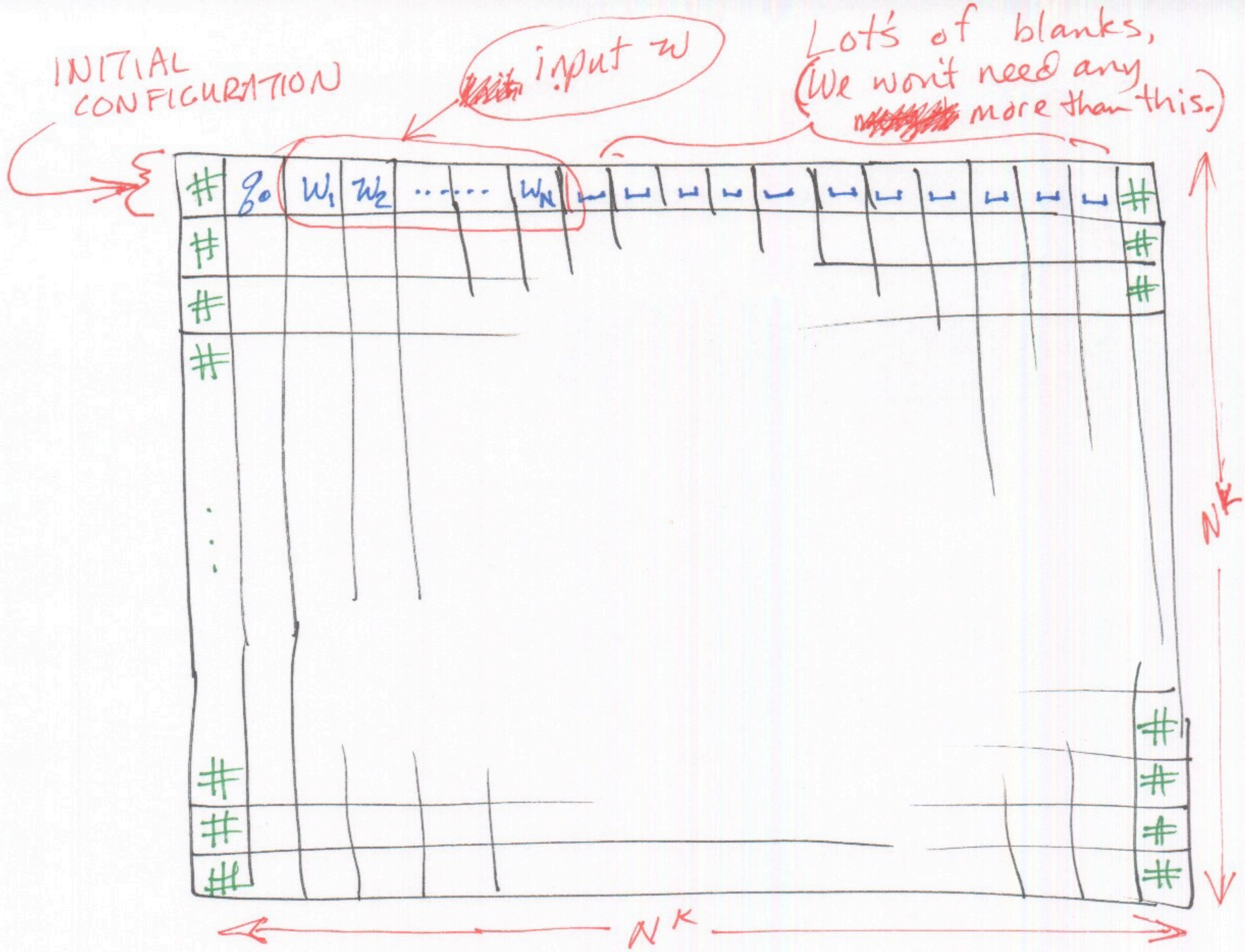
(It's big, but Polynomial in size)

Step 2:

Create lots of boolean variables to model what could be in each cell of the TABLEAU.

Step 3:

Create a formula to express all the constraints on the TABLEAU to guarantee it models a legal, accepting computation history.



Each cell contains a single symbol.

$\#$ ← to mark end-of-tape

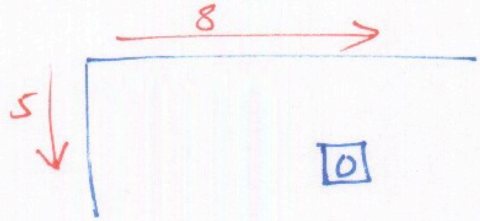
State ← Each row should have a state, q_4

Tape Symbol

$$Q \times \Gamma \times \{\#\}$$

What is in each cell?

What is in cell 5,8?



It could be 0, or 1 or ␣ or # or q_4 ...

Create a boolean variable for each possibility.

$x_{5,8,0}$ = TRUE iff the cell contains "0"

$x_{5,8,1}$ = TRUE iff the cell contains "1"

$x_{5,8,␣}$ = TRUE iff the cell contains "␣"

$x_{5,8,\#}$ = TRUE iff the cell contains "#"

$x_{5,8,q_4}$ = TRUE iff the cell contains "q_4"

$$\boxed{x_{i,j,s} \text{ for all } 1 \le i,j \le n^k \text{ and } s \in Q \cup \Gamma \cup \{\#\}}$$

Now build the formula.

Goal: Add all constraints to assure the TABLEAU is a legal computational history that accepts.

ϕ_{CELL}

CONSTRAINT #1

Every cell contains exactly one symbol.

ϕ_{START}

CONSTRAINT #2

~~Every~~ The First Row is the starting configuration.

ϕ_{ACCEPT}

CONSTRAINT #3

Some cell contains the symbol q_{ACCEPT}

ϕ_{MOVE}

CONSTRAINT #4

Each Row (i.e. each "configuration") can legally follow the previous configuration, according to the transitions in the Nondeterministic TM we are modelling.

Construct the entire Boolean formula:

$$\phi = \phi_{CELL} \wedge \phi_{START} \wedge \phi_{ACCEPT} \wedge \phi_{MOVE}$$

This formula, while really huge, is polynomial in size (of w)

If ϕ has a solution, then there is an accepting computation history.

If there is an accepting computation history, then this formula has a solution.

If you can determine whether this formula ϕ has a solution in poly-time, then you can determine in poly-time whether a Nondeterministic TM N will accept w.

$$SAT \in P \Rightarrow P = NP$$

$$\phi = \phi_{CELL} \wedge \phi_{START} \wedge \phi_{ACCEPT} \wedge \phi_{MOVE}$$

$\boxed{\text{CONSTRAINT \#1}}$

Every cell contains exactly one symbol.

$x_{5,8,\#}$ = TRUE iff cell contains "#"

$x_{5,8,q_4}$ = TRUE iff cell contains "q_4"

$$\bigvee_{s \in (Q \cup \Gamma \cup \{\#\})} x_{i,j,s}$$

At least one of the variables for this cell is TRUE

$$\bigwedge_{s \neq t} \left(\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}} \right)$$

For every pair of variables for this cell, at least one is FALSE

Combining ($\wedge$): Exactly one variable is true.
And this is true of <u>all</u> cells, (i,j):

$$\phi_{CELL} = \bigwedge_{1 \le j,k \le N^k} \left[\left(\bigvee_{s \in \ldots} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s \neq t \\ s \in \ldots \\ t \in \ldots}} \left(\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}} \right) \right) \right]$$

$$\phi = \phi_{CELL} \wedge \phi_{START} \wedge \phi_{ACCEPT} \wedge \phi_{MOVE}$$

CONSTRAINT #2:

The first row describes the initial configuration.

$$\phi_{START} = \ldots \wedge \ldots \wedge \ldots \wedge \ldots \wedge \ldots$$

$$= x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{2^N,2^N,\#} \wedge \cdots$$

all blanks

input string $w = w_1 w_2 w_3 \cdots w_N$

$$\cdots \wedge x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge x_{1,5,w_3} \cdots \wedge x_{1,N+2,w_N}$$

w is in the first N cells of the tape.

$$\cdots \wedge x_{1,N+3,\sqcup} \wedge x_{1,N+4,\sqcup} \wedge \cdots \wedge x_{1,N^k+2,\sqcup}$$

The remaining cells of the tape contain the blank symbol, $\sqcup$.

$$\phi = \phi_{CELL} \wedge \phi_{START} \wedge \phi_{ACCEPT} \wedge \phi_{MOVE}$$

CONSTRAINT #3:

The ACCEPT state is reached. in the computation history.

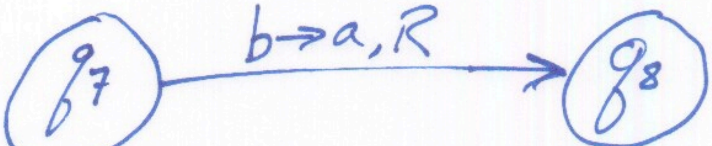
Some cell somewhere ~~represents~~ contains q_{ACCEPT}.

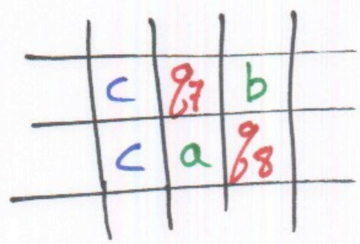
$$\phi_{ACCEPT} = \bigvee_{1 \le i,j \le 2^N} x_{i,j,q_{ACCEPT}}$$

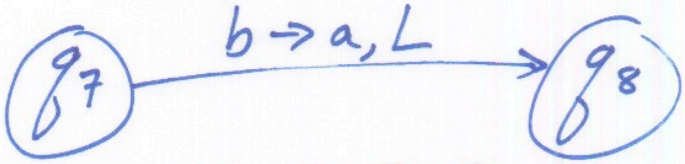
$$\phi = \phi_{CELL} \wedge \phi_{START} \wedge \phi_{ACCEPT} \wedge \phi_{MOVE}$$

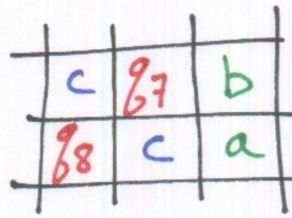
CONSTRAINT #4:

EVERY CONFIGURATION CAN LEGALLY FOLLOW THE PREVIOUS CONFIGURATION, ACCORDING TO THE DETAILS OF THE NONDET. TM'S TRANSITION FUNCTION.



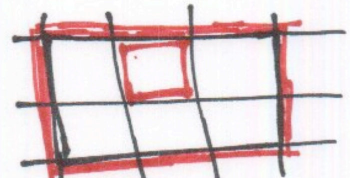




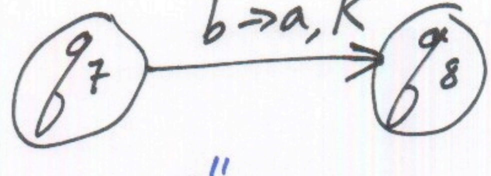


FOR ANY SYMBOL c

THE "WINDOW" CENTERED ON CELL i, j.

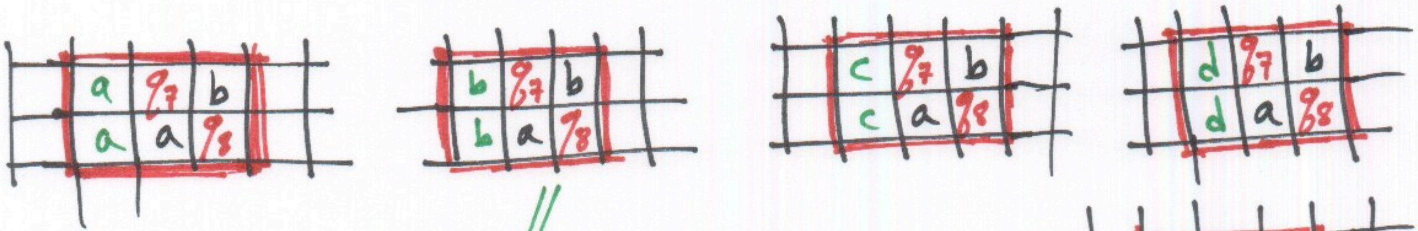


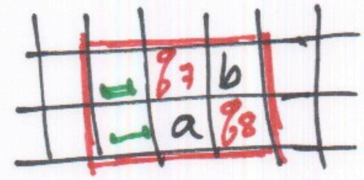
THE TRANSITION FUNCTION TELLS US WHAT THE LEGAL WINDOWS ARE.

THIS IS A TRANSITION: 

SO THESE ARE LEGAL "WINDOWS":

$\Gamma = \{a, b, c, d, \sqcup\}$





FOR EACH WINDOW, MAKE A FORMULA TO DESCRIBE IT.

$$\begin{aligned} &x_{i,j-1,b} \wedge x_{i,j,q_7} \wedge x_{i,j+1,b} \wedge \\ &x_{i+1,j-1,b} \wedge x_{i+1,j,a} \wedge x_{i+1,j+1,q_8} = W_{37} \end{aligned}$$

GIVEN AN i,j POSITION, IT MUST CONTAIN (OR MATCH) ONE OF THE LEGAL WINDOWS.

$$W_1 \vee W_2 \vee W_3 \vee \cdots \vee W_{37} \vee \cdots \vee W_{592}$$

NOW MAKE SURE EVERY WINDOW IN THE TABLE IS LEGAL.

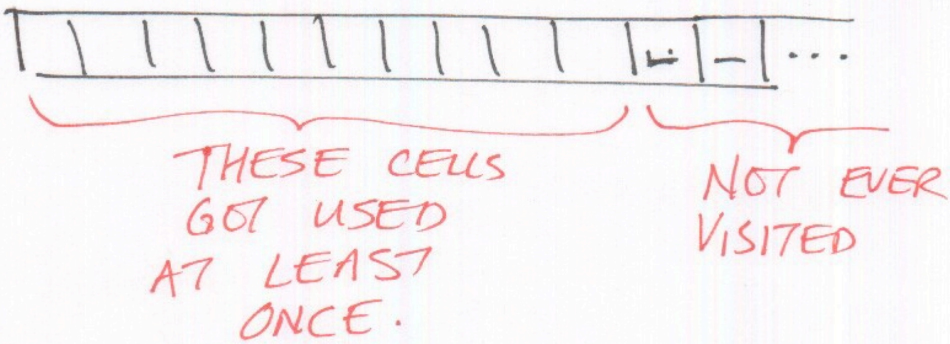
$$\phi_{MOVE} = \bigwedge_{1 \le i,j \le n^k} \left(\bigvee_{\substack{\text{ALL 592}\\\text{LEGAL}\\\text{WINDOWS}}} \left(\begin{matrix} x \wedge x \wedge x \wedge \\ x \wedge x \wedge x \end{matrix} \right) \right)$$

SPACE COMPLEXITY

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 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 ~~a~~ 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, ~~does~~ 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.

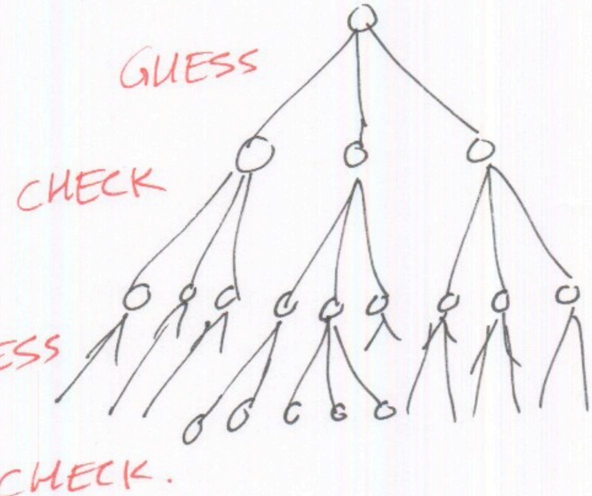
- Guess a good move for me.
- Check all his possible moves.
- Guess another good move.
- Check all his possible moves.

⋮

Nondeterminism helps here

But does not help here.

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

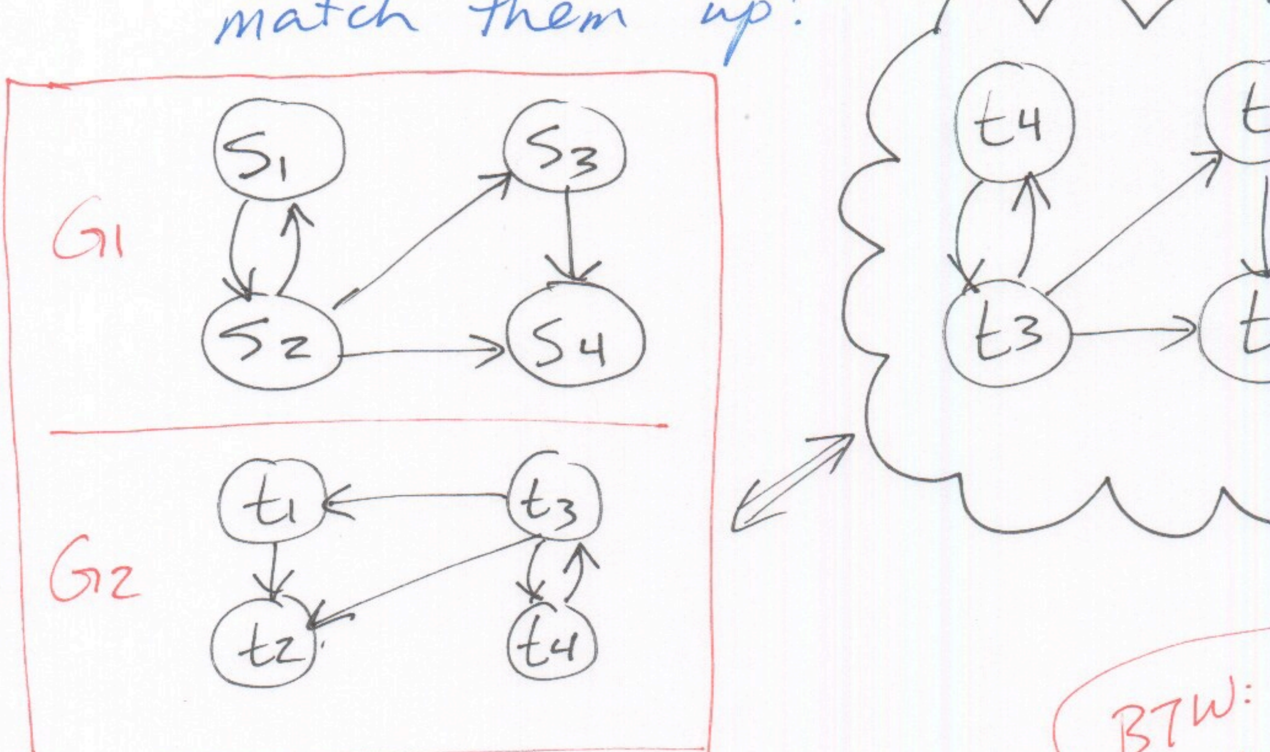


- This is a Search of a tree.
- The tree is exponential in size.
  $\Rightarrow$ We cannot store the tree.
- Do a depth first search of this tree.
- Time taken to search the tree: EXPONENTIAL.

GRAPH ISOMORPHISM

s_1	s_2	s_3	s_4	
t_1	t_2	t_3	t_4	← NO
4	3	1	2	← YES

Given two graphs, can you match them up?



BTW: This problem is NOT 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 NOT isomorphic?

This problem is NOT in NP.

There are N! different possible correspondences.

You have to check each of them.

P Det. TM- in Poly time

NP NonDet. TM in Poly time

PSPACE Det/Non Det TM- Poly space

EXPTIME Det. TM in exponential time

EXPSPACE Det. TM - exponential space.

WE KNOW:

$$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME \subseteq EXPSPACE$$

$$P \subset EXPTIME$$

$$PSPACE \subset EXPSPACE$$