Lecture 23

CSE 431
Intro to Theory of Computation

$TIME(f(n)) \subseteq NTIME(f(n)) \subseteq SPACE(f(n)) \subseteq NSPACE(f(n)) \subseteq TIME(2^{O(f(n))})$

for $f(n) \geq \log_2 n$

$TIME(2^{O(f(n))}) = \bigcup_k TIME(2^{k f(n)})$

$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXP \subseteq NEXP$

$P \neq EXP$, $NL \neq PSPACE$

$\Phi \quad \exists x_1 \forall x_2 \exists x_3 \ldots \exists x_n \, \varphi(x_1 \ldots x_n)$

Formula Game for Φ: Played on formula Φ:

$\exists$ player wants to make φ have value 1
$\forall$ player wants to make φ have value 0.

Players take turns setting values of x_i vars.
$\exists$ player sets all $\exists x_i$ variables
$\forall$ player sets all $\forall x_i$ variables

$\exists$ player has a winning strategy $\Leftrightarrow$ Φ is true

note: $\neg\Phi$ is true $\Leftrightarrow$ $\neg \exists x_1 \forall x_2 \exists x_3 \ldots \exists x_n \varphi(x_1 \ldots x_n)$
$\Leftrightarrow \forall x_1 \exists x_2 \forall x_3 \ldots \forall x_n \neg\varphi(x_1 \ldots x_n)$
$\Leftrightarrow$ $\exists$ player for $\neg\Phi$ has winning strategy
$\Leftrightarrow$ $\forall$ player for Φ has a winning strategy

Above is the basic idea for how to start showing that deciding guaranteed win in 2-player games are PSPACE-complete ($n \times n$ checkers etc.)

Polynomial Hierarchy:

sets of languages Σ_k^P, Π_k^P for integer k

$$\Sigma_k^P = \{A : A \leq_m^P \{\text{true formulas in } \exists QBF_k\}\}$$

where $\exists QBF_k = \{$fully quantified Boolean formulas with $\leq k$ alternating blocks of quantifiers beginning with $\exists\}$

$$\Pi_k^P = \{A : A \leq_m^P \{\text{true formulas in } \forall QBF_k\}\}$$

where $\forall QBF_k = \{$fully quantified Boolean formulas with $\leq k$ alternating blocks of quantifiers beginning with $\forall\}$

$\Sigma_1^P = NP \qquad \exists x_1 \ldots \exists x_n \, \varphi(x_1 \ldots x_n)$

$\Pi_1^P = coNP \qquad \forall x_1 \ldots \forall x_n \, \varphi(x_1 \ldots x_n)$

Polynomial time hierarchy:

$$PH = \bigcup_k \Sigma_k^P = \bigcup_k \Pi_k^P$$

There are other natural problems with practical importance that are as hard as any problem in classes like Σ_2^P

e.g. Minimum equivalent DNF (sum of products)
$\{\langle F \rangle : F$ is a DNF formula that cannot be shortened$\}$

Thm: If $P = NP$ then $PH = P$

Proof idea: if $P = NP$ then we can replace
$\exists \vec{x}\, \varphi(\vec{x})$ by some φ' of polysize.
or $\forall \vec{x}\, \varphi(\vec{x})$ by some φ' of polysize
repeat this k times starting on the inside ∎
Blow-up is fixed exponent depending on k.

Note: key difference between PH and PSPACE
is that # of alternations is bounded by
a constant than being dependent
on length of the input

Logarithmic Space

Consider the following non-regular language

$$A = \{0^n 1^n : n \geq 0\}$$

TM deciding A: On input x
Count # of 0's at start before first 1
Count # of 1's next.
If counts differ or there are
more characters before 1st
blank reject
else accept.

Space
two counters
up to length
of input

$O(\log n)$ bits

Let $L = SPACE(\log n)$ $\quad \therefore A \in L$
$NL = NSPACE(\log n)$

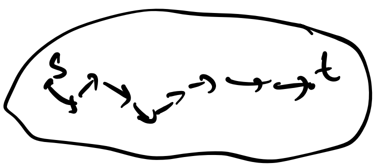
$$L \subseteq NL \subseteq TIME(2^{O(\log n)}) = P$$

Recall PATH = {<G, s, t> : G is a directed graph with a path from s to t}

Thm PATH ∈ NL

Proof idea "Guess and verify a path of length ≤ n (# vertices) from s to t, one vertex at a time"

not enough space to actually write down the path.

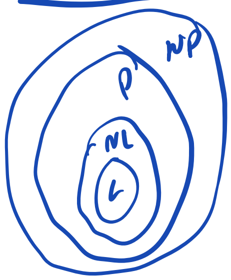


Keep track of : counter for the length — $\log_2 n$ bits
current vertex — $O(\log_2 n)$ bits
(and next vertex)

NTM: count ← 0
curr ← s
while count ≤ n and curr ≠ t do {
Guess next vertex (neighbour) v of curr
Check if (curr, v) is an edge
If not then reject
else curr ← v. }
If curr = t then accept
else reject

Question: Is L = NL?
L = P?
NL = P?
NL = NP?
L = NP?

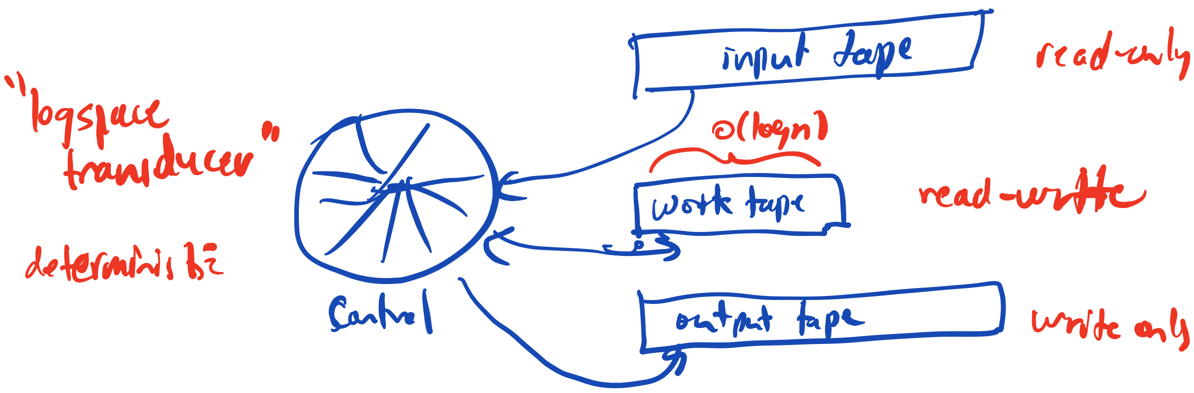
All of these are open
If P ≠ NP then L ≠ NP
which may be easier to show.



To study these questions we need a finer notion of reduction than $\leq_m^P$ which allows polynomial slack

For this we need a notion of log-space computable functions

We modify our 2-tape space bounded notion of TM to a 3-tape TM like this:



Def^n A function f is logspace computable

Iff it is computed by some logspace transducer.

$\Rightarrow$ f is polynomial-time computable (see below)

e.g. Consider the SORT function on n, $O(\log n)$-bit numbers

$\text{SORT}(x_1, \ldots, x_n) =$ "$x_1 \ldots x_n$ in sorted order"

i.e. $x_{\sigma(1)} \ldots x_{\sigma(n)}$ s.t. $x_{\sigma(1)} \leq x_{\sigma(2)} \leq \ldots \leq x_{\sigma(n)}$"

σ a permutation

Claim SORT is logspace computable:

Proof Fast sorts like quicksort, merge sort use too much space. Selection sort only needs counters and one or two x_i stored ∎

Note: This sort is slow $\geq n^2$ time but uses only $O(\log n)$ space

Fact: For any sorting algorithm
$$T \cdot S = \Omega(n^2)$$
time space

so this slowdown is unavoidable

Logspace reductions and completeness

Defn $A \leq_m^L B$ iff $A \leq_m B$ by a mapping function that is logspace computable

Thm If $A \leq_m^L B$ then $A \leq_m^P B$

Proof idea: # of configurations of logspace transducer (not counting the output tape which doesn't given how the TM moves) is $n^{O(1)}$ ∎

Defn B is **NL-hard** iff $\forall A \in NL$, $A \leq_m^L B$

Defn B is **NL-complete** iff
- $B \in NL$
- B is NL-hard

Thm PATH is NL-complete

Proof We already showed that PATH $\in$ NL we just need to show NL-hardness

Let $A \in NL$

Claim $A \leq_m^L PATH$

Proof $A \in NL$

$\therefore \exists$ NTM M deciding A using $O(\log n)$ space.

M accepts x

$\iff \exists$ path from C_0 to C_{accept} in directed graph $G_{M,x}$

so A PATH

$x \xmapsto{f} \langle G, s, t \rangle$

$G = G_{M,x}$, $s = C_0$, $t = C_{accept}$

$x \in A \iff \exists$ path in $G_{M,x}$ from C_0 to C_{accept}

Output is polysize since $2^{O(\log n)}$ nodes.
each edge takes only $O(\log n)$ bits to write down
and is easy to list through edges in order. $\blacksquare$