30/09/2024

Computational Complexity

Lec 1

Exercises (every one or two weeks) → 20%
Exam 1 — 23% (Sept 3)
Exam 2 — 23% (Oct 15)
Exam 3 — 34%

Goal:- To understand the easiness/difficulty of computational problems.

In terms of resources used —

Time : P, NP
Space : PSPACE

other resources:
- Randomness
- No of gates
- Interaction

→ complexity class : Collection of problems classified based on resources needed.

P → Polynomial time
NP — Nondeterministic Polynomial Time

→ Time : P, NP, Polynomial Hierarchy
→ Space : PSPACE, Log Space, Non Det. log Space
→ Oracle
→ Randomized Complexity : RP, ZPP, BPP
→ Circuit Complexity : P/poly, AC, NC

Till Chapter 7 of Arora.

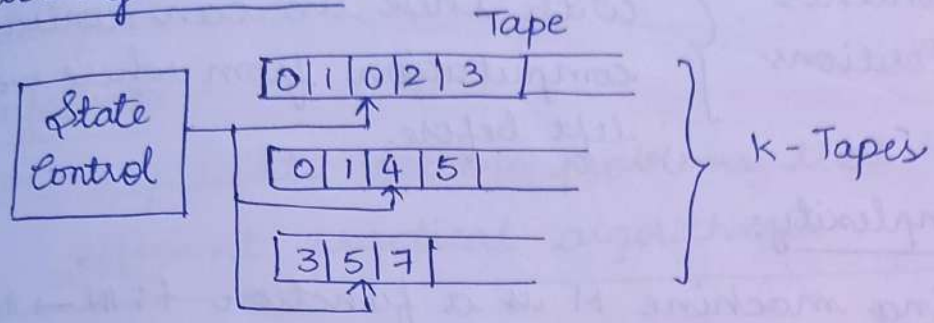
Extra :

→ Interactive Proofs (8? Arora)
→ Hardness of Approx (11? Arora)

Decision Problems :-

→ which have YES/NO answers.
→ We usually work on them, not functional problems.

Turing Machine



k is fixed : Machine is fixed

→ Power of k-tapes vs Power of 1-tape?
Equal.

Transition Functions

$\delta(q_3, 2, 4, 1) = (q_6, 1, 3, 7, L, R, S)$

Replace 2,4,1 with 1,3,7
move head 1 : 1 step left 3 : stay there
of take 2 : 1 step right

0+L rule → Just stay there
Tapes are infinite till end (right end)

→ Modifications : Infinite both ends, 2D tapes etc.

TM : Start with input in tape 1.
Other tapes are blank.
All heads are at left most positions.

Non determinism and NP

Defn: A non-deterministic TM (NTM) accepts an input if there is any valid sequence of moves that leads to acceptance.

Running time of an NTM is a function $f: \mathbb{N} \to \mathbb{N}$ where $f(n)$ is the maximum among all inputs of length n, and maximum length of computation among those input.

Defn:- NTIME $(t(n))$

Class of all languages that are decided in time $O(t(n))$ by a non-deterministic T.M

$\{L \mid L$ is decided in $O(t(n))$ time by an NTM$\}$

Defn:- $NP = \bigcup_{k=1}^{\infty} NTIME(n^k)$

SUBSET SUM

$\{\langle S,t\rangle \mid S = \{x_1, x_2, \ldots, x_k\} \exists\ T \subseteq \{1,2,\ldots,k\}$ such that $\sum_{i \in T} x_i = t\} \in NP.$

On an input $\langle S,t\rangle$ non deterministically select or reject each of $x_1, x_2, \ldots, x_k$.

- Add the selected x_i's and verify if they sum to t.
- If sum $= t \rightarrow$ accept else reject.

3-COLOURABLE :

$\{\langle G \rangle \mid G$ is 3-colourable$\}$

Go through vertices 1...n, assign R, G, B in non-deterministic fashion and once the colouring is finalised, go through each edge, checking the colour of the two end points.

If colouring is proper, accept, else, reject.

$DTIME(n^k) \subseteq NTIME(n^k)$

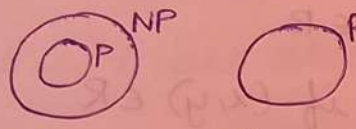
→ We can view the DTM as an NTM.

$$\bigcup_k DTIME(n^k) \subseteq \bigcup_k NTIME(n^k)$$

$$P \subseteq NP$$

P vs NP Question

Is this containment proper?



Defn:- NP is the class of languages that have det. polynomial time verifiers.

SAT: (satisfiability)

Is the given boolean formula satisfiable (be true)

SAT = $\{\langle \phi \rangle \mid \phi$ is a boolean formula, and is satisfiable$\}$

Are there any which are exp. in non determinism?

A. Succint circuits. They accept two vertex numbers are input and output whether there is an edge between them.

5/08/2024 # Computational Complexity Lec 3

SAT $\in$ NP

CNF : Conjunctive normal form

$C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge \ldots$

$C_i = (x_{i_1} \vee x_{i_2} \vee \overline{x}_{i_3} \vee \ldots)$

i.e. AND of OR's.

3-CNF form : where each clause has exactly 3 literals.

CNF-SAT = $\{\langle \phi \rangle$ is a boolean CNF formula that is satisfiable $\}$

3-CNFSAT $\rightsquigarrow$ similar

CNF-SAT, 3-CNFSAT $\in$ NP

Theorem :- NP is a class of languages that have polynomial time verifiers.

(Recall. We define $NP = \bigcup_{k=1}^{\infty} NTIME(n^k)$)

A verifier for L is a det T.M V such that

$L = \{x \mid \exists y, V \text{ accepts } \langle x, y \rangle\}$

For x to be in L, there should exist a string (y) which can be used to verify the membership of x in L.

Example :- 3-colourable graph

y can be a valid 3-colouring.

y is called proof / witness / certificate.

$L \in NP \iff$ { there is a polytime verifier alg V such that
$x \in L \iff \exists y, |y| = poly(|x|)$
$V(x,y) = 1$

$L \in P \iff$ { there is a det. poly time alg. A such that
$x \in L \iff A(x) = 1$

Proof:- NP => Polytime verifier.

Suppose $L \in NP$. So L has a NTM, N that runs in poly-time. If $x \in L$, there is a computation path that leads to x getting accepted.

Verifier V will run one computation path of N. V will ~~use~~ be guided by the encoding ~~path~~ of an accepted path, which is given as y leading to accept.

If $x \in L$, then $\exists$ a path, and hence $\exists y$.

Max no of branches at any config of N will be a constant, independent of input x.

V simply simulates N. But when it has to make a choice among multiple next configurations, it will be guided by y.

If $x \notin L$, then there is no such path and hence no such y.

Polytime verifier => NP

Suppose ∃ a polytime verifier V for L. The existence of V implicitly fixes a "format" for the certificate and hence a bound on its length.

NTM N now guesses a certificate y of the format/size and runs V on $\langle x, y \rangle$.
It can be checked that this is a valid NTM for L (Exercise. Check this!)

Max no. of paths = $b^{|y|}$ OR no. of possible certificate strings

where b is the alphabet size of the certificate alphabet.

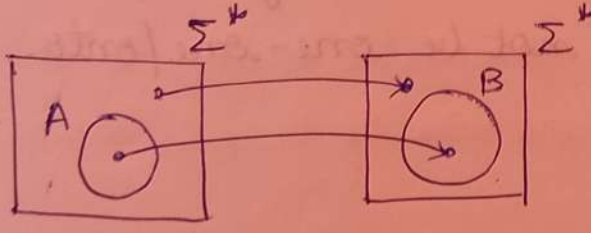
Poly time reductions

Lang A is poly time reducible to Lang B if there is a poly time computable function $f: \Sigma^* \to \Sigma^*$, such that

$w \in A \iff f(w) \in B$.

denoted $A \leq_P B$.

DOUBT: Why backward arrow?



Note:- f itself should be polytime computable.

If you know how to decide B, we can convert or reduce A-instance to B-instance and then decide B-instance.

8th Aug '24 CS5110 / Computational Complexity (Lec 4)

$A \leq_p B$ if there is f, computable in poly time s.t $\forall x$

$$x \in A \iff f(x) \in B$$

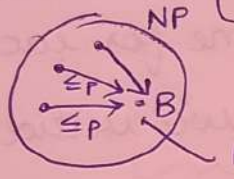
3-SAT $\leq_p$ CLIQUE

NP-Completeness

A language B is NP-complete if

(1) $B \in NP$

(2) $\forall A \in NP,\ A \leq_p B$ → NP-hard



NP-complete.

1) Suppose

If B is an NP-Complete problem.

If B has a polytime algorithm, then

$\forall A \in NP,\ A \in P \Rightarrow P = NP$

This is because $A \leq_p B$ and $B \in P$.

- NP complete problems are considered to be the "hardest problems in NP".

2) Suppose B is NP-complete. Suppose there is another language $C \in NP$. If $B \leq_p C$, then C is NP-complete.

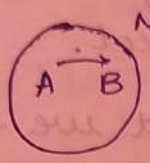
We know $C \in NP$

$\forall A \in NP$, we know $A \leq_p B$

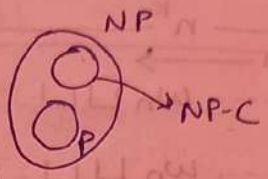
$A \leq_p B$ & $B \leq_p C \Rightarrow A \leq_p C$

Q: Are there languages in NP, but not NP-complete?

→ Suppose P=NP. Then all non-trivial languages (not ϕ, Σ^*) in P=NP are NP-complete.



→ Suppose $P \neq NP$. In this case, we know $NP\text{-comp} \cap P = \phi$.



Yes, since $P \neq$ NP-complete.

But, are there languages in NP that are neither in P nor NP-complete?

→ Yes. Ladner's theorem.

FACTOR = $\{\langle N, q\rangle \mid$ Does N have a factor between 2 and $q\}$

Cook-Levin Theorem

- SAT is NP-complete.

(1) $SAT \in NP$

(2) $\forall A \in NP$, $A \leq_p SAT$

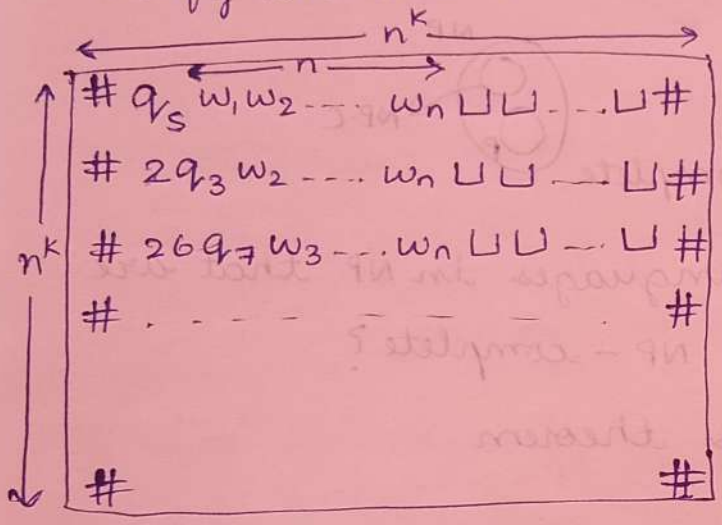
We will prove (2) Given $A \in NP$ we will give f such that $f(w)$ is a SAT instance such that

$$w \in A \iff f(w) \in SAT.$$

$A \in NP$ $\Rightarrow$ there is an NTM N that decides on A instance in n^k time.

Given w, if $w \in A$, there is a sequence of configurations of N that lead to w being accepted. If $w \notin A$, all legal sequence of configurations lead to reject.

The Boolean formula that we construct, will verify the above statements.



→ This computation table has $n^k \times n^k = n^{2k}$ cells.

We will write a formula ϕ that checks this.

→ Does the T.M start ~~colle~~ correctly?

→ Does it end correctly?

→ Does it move correctly?

→ Do the variables form a proper encoding of the table?

$$\phi = \phi_{start} \wedge \phi_{accept} \wedge \phi_{move} \wedge \phi_{cell}$$

Boolean variables $x_{i,j,s}$

↙ True if $(i,j)^{th}$ cell contains symbol s ↘ False otherwise.

$1 \le i,j \le n^k$

$s \in \Delta = \Gamma \cup Q \cup \{\#\}$ → Tape Alphabet → Contains input symbols and ($\sqcup$) and possibly more.

↓ States

No. of variables $= n^{2k} * |\Delta|$

x_{i,j,a_1} x_{i,j,a_2} x_{i,j,a_3} x_{i,j,a_4} ....

Among all $x_{i,j,s}$ for a fixed i,j, exactly one should be true.

$$\phi_{cell} = \bigwedge_{1 \le i,j \le n^k} \left[\left(\bigvee_{s \in \Delta} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,s' \in \Delta \\ s \neq s'}} [\bar{x}_{i,j,s} \vee \bar{x}_{i,j,s'}] \right) \right]$$

$$\text{size} = (\Delta + 2\Delta^2) n^{2k}$$

$$\phi_{start} = x_{1,1,\#} \wedge x_{1,2,q_{start}} \wedge x_{1,3,w_1} \wedge \cdots$$

$$\wedge x_{1,n+2,w_n} \wedge x_{1,n+3,\sqcup} \cdots \wedge x_{1,n^k,\#}$$

$$\text{size} = n^k$$

$$\phi_{accept} = \bigvee_{1 \le i,j \le n^k} x_{i,j,q_{accept}}$$

$$\text{size} = n^{2k}$$

ϕ_{move} is the hardest, and ensures that each configuration in the table is a valid successor of the previous config.

We cannot brute force through all the possible exponential configs of the NTM.

We note that the TM changes tape contents in a "local" manner.

It is enough to check 2×3 windows of the table. We tile the computation table with 2×3 windows and ensure that each such window is legal.

legal: It can legally appear in an actual computation of N.

$\frac{a\ b\ c}{a\ b\ c}$ where a, b, c are all part of Γ.

$\frac{a\ q\ b}{a\ c\ r}$ if $\delta(q, b)$ contains (r, c, R) as an option.

$\phi_{move} = \bigwedge_{i,j}$ [formula that checks that $(i,j)^{th}$ window is a valid window]

22/08/2024
Thursday

Computation Complexity Lec 6

COOK-LEVIN Theorem

SAT is NP-complete.

→ $\forall A \in NP, A \leq_p SAT$

→ Computation tables

→ list of all configurations of the NTM

Let $A \in NP$ Then there is an NTM N

Consider $w \in A$, then N has an accepting series of config. on w.

ϕ: Checks if N has a sequence of configurations leading to accept of w.

$\phi = \phi_{start} \wedge \phi_{accept} \wedge \phi_{cell} \wedge \phi_{move}$

$$x_{ijl} = \begin{cases} \text{TRUE if } (i,j)^{th} \text{ cell of the table contains} \\ \text{symbol } l \in \Delta. \\ \text{FALSE otherwise} \end{cases}$$

$1 \leq i,j \leq n^k$

$l \in \Delta$

ϕ_{move}: Is each row a valid successor of the previous row?

We check all the 2×3 windows and see if they are valid

$\frac{b\ q\ a}{b\ d\ r}$ ✓ This is valid if $(r,d,R) \in \delta(q,a)$

$\frac{b\ q\ a}{r\ b\ d}$ ✓ $\frac{b\ c\ q}{b\ c\ d}$ ✓ is valid if there is a move that takes the head right from the state q and writes d.

~~b q a~~
~~r b d~~

$$\phi_{move} = \bigwedge_{1 \le i,j \le n^k} \overbrace{(i,j)^{th} \text{ window is valid}}$$

$$\bigvee_{(a_1,a_2,a_3,a_4,a_5,a_6) \text{ is valid } 2\times 3 \text{ window}} \left(x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6} \right)$$

Claim: If all 2×3 windows are valid, then all configurations are valid successors of the previous one.

Proof:- ① If a symbol is not adjacent to the state symbol, then it is unchanged.

② If symbol is adjacent to the state symbol, then the 2×3 window where top middle is state captures the change.

All 2×3 windows without state on top row have middle symbols unchanged.

$$w \text{ accepted by } N \iff \{ \text{the formula } \phi \text{ is satisfiable}$$

We also need to show that ϕ can be constructed in poly time.

$\phi_{cell} \rightarrow$ length $n^{2k} \times |\Delta|^2$

$\phi_{start} \rightarrow$ length n^k

$\phi_{accept} \rightarrow$ length n^{2k}

$\phi_{move} \rightarrow$ length $n^{2k} \times$ no. of valid windows $\times 6$

} Poly. in n.

All the formulas can be generated by a poly time algo. $\Rightarrow A \le_p SAT$.

Polynomial Hierarchy

→ Hierarchy of complexity classes.

P, NP

$L \in P \iff$ there is det. poly. time V s.t

$x \in L \iff V(x) = 1$

$L \in NP \iff$ there is det poly. time V s.t

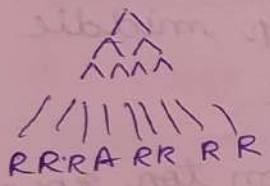
$x \in L \iff \exists y, |y|, |y| = poly(|x|)$

$V(x, y) = 1$

$L \in$ co-NP $\iff$ there is a det. poly. time V

s.t $x \in L \iff \forall y, |y| = poly(|x|)$

$V(x, y) = 1$



L ∈ NP: x is accepted iff there is atleast one accept computation

L ∈ coNP : x is accepted iff all comp. paths accept.

Def:- co-NP = $\{ L \mid \bar{L} \in NP \}$

↳ Not the complement of NP.

→ $\overline{3\text{-COL}}$: We can give a valid NO certificate for $\overline{3\text{-COL}}$

Consider the NP verifier of 3-COL. By flipping accept/reject, we get the co-NP verifier of $\overline{3\text{-COL}}$.

Same construction works for any NP language

Examples of languages in co-NP:

$\overline{3\text{-COL}}$, $\overline{SAT}$, CONNECTED

Note: $P \subseteq$ co-NP.

Consider the verifier V to be independent of y.

TAUTOLOGY: Is the given Boolean formula always true?

$\phi \in$ TAUTOLOGY $\Rightarrow \overline{\phi}$ should always be false

$\Rightarrow \overline{\phi} \notin SAT$.

Co-NP completeness:

B is co-NP Complete if (1) $B \in$ co-NP

(2) $\forall A \in$ co-NP, $A \leq_p B$

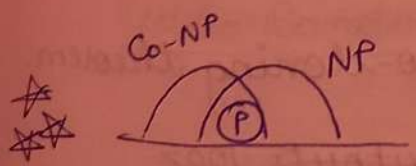
Examples: $\overline{SAT}$, $\overline{3\text{-COL}}$

If B is NP-complete, then $\overline{B}$ is co-NP complete

$\forall C \in$ co-NP, $\overline{C} \in NP$,

$\overline{C} \leq_p B$

$\Rightarrow C \leq_p \overline{B}$



If $P = NP$, then $P = NP = co\text{-}NP$

→ Taking contrapositive,

$NP \neq co\text{-}NP \Rightarrow P \neq NP$

26/08/2024
Monday.

Computation Complexity Lec 7

$L \in NP$ if there is a det. poly. time V such that

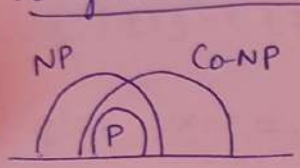
$$x \in L \iff \exists y, |y| = poly(|x|), V(x,y) = 1$$

$L \in$ co-NP if there is a det. poly time V such that

$$x \in L \iff \forall y, |y| = poly(|x|), V(x,y) = 1.$$

Also $Co\text{-}NP = \{L \mid \bar{L} \in NP\}$

Polynomial Hierarchy



if $NP \subseteq co\text{-}NP \Rightarrow NP = co\text{-}NP$

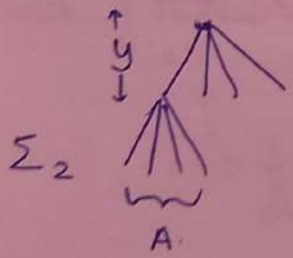
~~need~~ not necessarily = P.

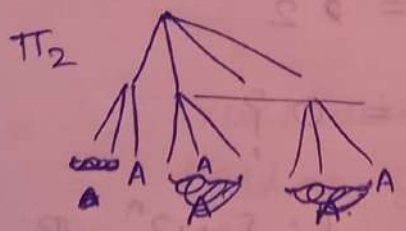
$L \in \Sigma_2$: $x \in L \iff \exists y, \forall z$, where $|y|, |z|$ are $poly(|x|)$ and $V(x,y,z) = 1$.

$(\Sigma_2^P)^P$

$L \in \Pi_2$: $x \in L \iff \forall y, \exists z$, where $|y|, |z|$ are $poly(|x|)$ and $V(x,y,z) = 1$

(Π_2^P)





Any y-subtree has atleast one accept computation.

Example: NOT-SMALLEST-FORMULA

$NSF = \{ \langle \phi \rangle \mid \phi$ is a Boolean formula, $\exists \psi, |\psi| < |\phi|$, and $\psi \equiv \phi \}$

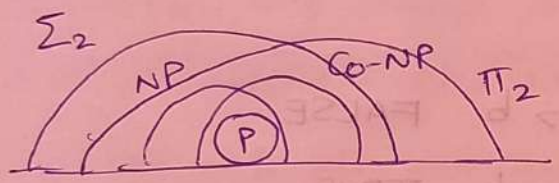
$\phi \in NSF \iff \exists \psi$, s.t $|\psi| < |\phi|$ and $\forall z$, $\psi(z) = \phi(z)$

$\iff \exists \psi, \forall z,$ { $|\psi| < |\phi|$ and $\psi(z) = \phi(z)$

↑ existential Quantifier

↑ Universal Quantifier

$NSF \in \Sigma_2$.



$L \in \Sigma_3^P$: $x \in L \iff \exists y \forall z \exists w$ s.t $|y|, |z|, |w|$ are poly$(|x|)$ and $V(x, y, z, w) = 1$.

Π_3^P : $x \in L \iff \forall y \exists z \forall w$, s.t $|y|, |z|, |w|$ are poly$(|x|)$ and $V(x, y, z, w) = 1$.

*
**

EXACT-CLIQUE : $\{ \langle G, k \rangle \mid$ largest clique in G is of size k $\}$

Try this

EXACTLY-CLIQUE $\in \Sigma_2$ and Π_2

$\Pi_2 = \{ L \mid \bar{L} \in \Sigma_2 \}$

Proof:- Consider $L \in \Sigma_2$. Then there is det. poly time V such that

$$x \in L \iff \exists y_0 \forall z, \ |y|, |z| \in \text{poly}(|x|), \ V(x,y,z) = 1$$

$$x \notin L \iff \forall y \exists z, \ |y|, |z| \in \text{poly}(|x|), \ V(x,y,z) = 0$$

$$x \in \bar{L} \iff \forall y \exists z, \ |y|, |z| \in \text{poly}(|x|), \ V'(x,y,z) = 1$$

$$\bar{L} \in \Pi_2$$

Consider rewriting this by setting $V' = V$ with output flipped.

$a, b \in \mathbb{Z}$ $\quad \exists a \ \forall b \ a > b$ FALSE

$\forall b \ \exists a \ a > b$ TRUE

Order of quantifiers are important.

$\forall a \ \exists b \ a > b$, TRUE

The relation between Σ_2 and Π_2 is not known. But we ~~have~~ know

$$P, NP, co\text{-}NP \subseteq \Sigma_2$$

$$P, NP, co\text{-}NP \subseteq \Pi_2$$

To show $NP \subseteq \Pi_2$

Let $L \in NP$, then there is a det. poly time V s.t

$$x \in L \iff \exists y \quad V(x,y) = 1.$$

Now we create $V'(x,y,z) = V(x,y)$

$x \in L \iff \exists y \; V'(x,y,z) = 1$

$x \in L \iff \forall z \; \exists y \; V'(x,y,z) = 1$

(since V' is independent of z)

So $L \in \Pi_2$.

$\phi \in$ SMALLEST-FORMULA

$\iff \forall \psi$ s.t $|\psi| < |\phi|$, $\exists z$ s.t $\psi(z) \neq \phi(z)$

$\phi \in SAT$: $\exists x$ s.t $\phi(x) = TRUE$

ΣSAT_k: We need a Boolean formula with the variables seperated into k sets.

$\phi \in \Sigma SAT_k$: $\exists x^{(1)} \forall x^{(2)} \exists x^{(3)} \ldots Q_t x^{(k)}$

$\phi(x^1, x^2, \ldots, x^k) = TRUE$

Theorem:- Suppose $P = NP$. Then $P = Co\text{-}NP = NP = \Sigma_2 = \Pi_2$.

Proof:- If $P = NP$. Consider $L \in Co\text{-}NP$

$\bar{L} \in NP = P$. So $L \in P \Rightarrow CoNP \subseteq P \Rightarrow CoNP = P$.

Consider $L \in \Sigma_2$. Then

$x \in L \iff \exists y, \forall z, V(x,y,z) = 1$.

Consider a language L' that consists of pairs

(x,y) as defined below

$$(x,y) \in L' \iff \forall z,\ V'((x,y),z) = 1$$
$$\| $$
$$V(x,y,z)$$

Clearly, by defn., $L' \in$ Co-NP $\Rightarrow L' \in P$.

There is a det. poly. time alg $\hat{V}$ for L'

$$(x,y) \in L' \iff \hat{V}(x,y) = 1$$

Now we can rewrite L as follows.

$$x \in L \iff \exists y\ \hat{V}(x,y) = 1.$$

This implies that $L \in NP = P$

So $\Sigma_2 \subseteq P \Rightarrow \Sigma_2 = P$.

Similarly, we can show $\Pi_2 = P$ if $P = NP$.

Exercise: Try to work out the details of $\Pi_2 = P$.

Similar ideas can be used to show $P = NP \Rightarrow$
$\Sigma_3 = \Pi_3 = P$.

Theorem: Suppose NP = Co-NP. This implies $\Sigma_2 = \Pi_2 = NP$.

Proof:- Let $L \in \Sigma_2$. Follow the same proof till we get L'. $L' \in$ Co-NP $\Rightarrow L' =$ NP

Then there is a det. poly. time $\hat{V}$ such that

$$(x,y) \in L' \iff \exists w,\ \hat{V}((x,y),w) = 1.$$

Rewriting L as follows

$$x \in L \iff \exists y\ \exists w\ \ \hat{V}((x,y),w) = 1$$

$$\iff \exists (y,w)\ \ V''(x,(y,w)) = 1$$

$\Rightarrow L \in NP$. Also, $\Sigma_2 \subseteq NP \Rightarrow \Sigma_2 = NP$.

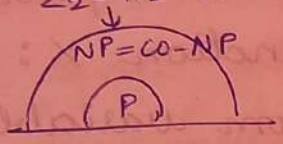
To see that $\Pi_2 = NP$, consider

$L \in \Pi_2 \Rightarrow \bar{L} \in \Sigma_2 = NP = co\text{-}NP$

$\Rightarrow L \in NP$

$\Pi_2 \subseteq NP \Rightarrow \Pi_2 = NP$

$\Sigma_2 = \Pi_2$ also collapses here



also any $\Sigma_i = \Pi_i$ collapses here.

29/08/2024
Thursday

Computation Complexity Lec 8

Σ_k – Complete Language

Σ_kSAT

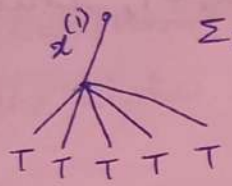
$$\exists x^{(1)} \forall x^{(2)} \exists x^{(3)} \ldots Q_k x^{(k)} \phi$$

where Q_k is $\exists/\forall$ depending on parity of k

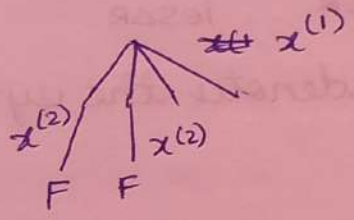
This is a Σ_k-complete language.

Each $x^{(i)}$ is a set of variables

Σ_2 $x^{(1)}$ Σ_2 SAT – Yes instance



Σ_2 SAT – No instance



Σ_2 language A : Then there is det. poly V

$$x \in A \iff \exists y \forall z \quad V(x,y,z) = 1.$$

Since $V(x,y,z)$ is deterministic, we can encode the entire operation into a polynomial sized Boolean formula without needing additional variables (Compute the additional variables completely)

Time Hierarchy Theorem

Q: If we have more time, can we really compute more?

Yes

Theorem :-

Let $t_1(n)$, $t_2(n)$ be "time-constructible" functions such that $(t_1(n) \log t_1(n)) \leq O(t_2(n))$ then

$$DTIME(t_1) \subsetneq DTIME(t_2)$$

Time Constructible: We should be able to compute the function $f(n) : \mathbb{N} \to \mathbb{N}$ in $f(n)$ time. That is That is, given 1^n as input, we should be able to write $f(n)$ ~~in bi~~ is binary, in $f(n)$ time.

Diagonalization

$$A_{TM} = \{ \langle M, w \rangle \mid M \text{ accepts } w \}$$

Theorem: A_{TM} is undecidable.

Proof: Let M' be a machine that decides A_{TM}.

D: Given input $\langle M \rangle$
Runs M' on $\langle M, \langle M \rangle \rangle$
Accept if M' rejects.
Reject if M' accepts.

What happens when we give $\langle D \rangle$ as input to D?

CONTRADICTION

Proof of T.H Theorem:

We will construct L such that $DTIME(t_1(n)) \subsetneq DTIME(t_2(n))$.

For simplicity, we will use $t_1(n) = n$, $t_2(n) = n^2$

L is going to be the set of strings accepted

by D.

D: input <M>
Compute n = length <M>
Simulate M on <M> for $n^{1.9}$ steps of simulation.
Reject <M> if M accepts <M>.
Accept <M> if M does not accept <M>

$L \in DTIME(n^2)$. Since we need to run only $n^{1.9}$ steps.

$L \notin DTIME(n)$

Suppose the contrary is true. Let there be a DTM R that runs in linear time and recognizes L.

Let us feed R as input to D. Choose a description of R, call it X, of length n.
X → <R> We need $n^{1.9} > cn\log n$

A TM that runs in t(n) time can be simulated by a universal Turing machine in $O(t(n)\log(t(n)))$ time. (Hennie - Stearns - '66)

What happens when we feed x (description of R) as input to D?
- D runs R on X for $n^{1.9}$ steps.
Simulates
(by this time R decides on X, since R is linear time).

D accepts x if R rejects x.
D rejects x if R accepts x.

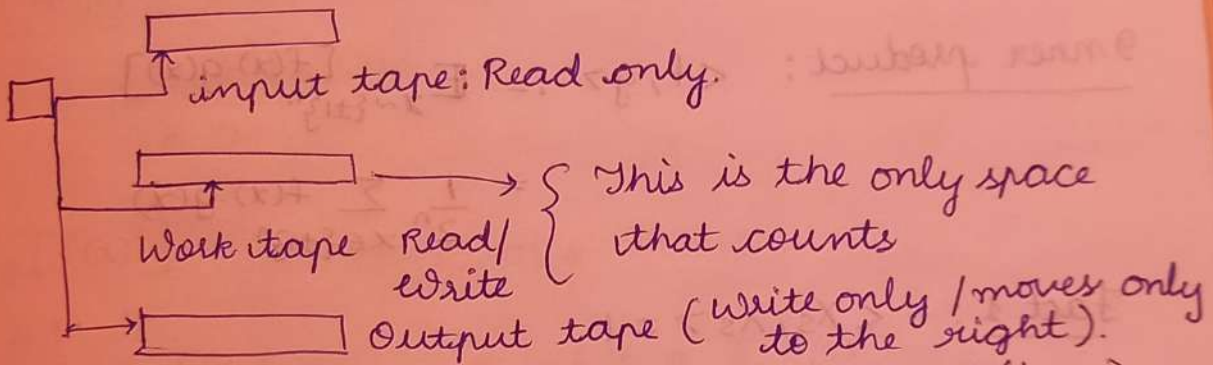
Now $L(D) \neq L(R)$. Contradiction.

Space Complexity

$L, NL = O(\log n)$ space

PSPACE = poly space

What is the model?
We want to not be penalized for input length.



Exercise: Decide PALINDROME in $O(\log n)$ space.

2/09/2024
Monday

Computational Complexity Lec 9

input tape : Read only

Work tape : Read / Write

Output tape : Write only (if needed)

Space Complexity : ~~Space~~ Space used in the work tape as a function of input length, n.

In case of NTM, space complexity is the maximum space used among all computation paths.

$$\left.\begin{matrix} SPACE(f(n)) \\ DSPACE(f(n)) \end{matrix}\right\} = \{L \mid L \text{ is decided by an } O(f(n)) \text{ space DTM}\}$$

$$NSPACE(f(n)) = \{L \mid L \text{ is decided by an } O(f(n)) \text{ space NTM}\}$$

→ PALINDROME $\in$ DSPACE(log n)

→ 3-SAT $\in$ DSPACE(n)

Work space to check if satisfies ϕ.

Def :- $L = LOGSPACE = DSPACE(\log n)$

$NL = NSPACE(\log n)$

$$PSPACE = \bigcup_{k=1}^{\infty} DSPACE(n^k)$$

Theorem:- $DSPACE(f(n)) \subseteq NSPACE(f(n))$

$L \subseteq NL$

Theorem:- $DTIME(f(n)) \subseteq DSPACE(f(n))$. ord(4)

Proof sketch:- In $f(n)$ time, you can cover atmost $f(n)$ space.

Theorem:- Suppose $f(n) \geq \log n$. Then.

$NTIME(f(n)) \subseteq DSPACE(f(n))$.

No. of computation paths $\leq b^{cf(n)}$.

All we need is a counter that stores which cell we are in, and the space for simulating that branch of computation

Counter requires $\leq cf(n)$ symbols.

Extra space for simulation $\leq cf(n)$.

Theorem:- $DSPACE(f(n)) \subseteq DTIME(2^{O(f(n))})$

Proof:- Time bounded TM could just simulate the space bounded TM. But how do we bound the time?

Bound the no. of config of the space bounded machine $= |\Sigma|^{cf(n)} \times n \times cf(n) \times |Q|$

$2^{c_1 f(n)} \times 2^{\log n} \times 2^{\log(cf(n))}$

Assuming $f(n) \geq \log n$

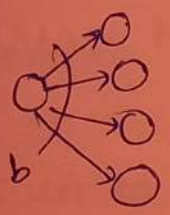
$\leq 2^{c_2 f(n)}$

We run the time bounded TM for $2^{c_2 f(n)}$ steps by using a counter to keep track of the no. of steps. If not accepted till then, we reject space required.

$$= \left.\begin{array}{l}\text{space} \\ \text{space bounded} \\ \text{TM}\end{array}\right\} + \text{Extra space for counter}$$

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

Theorem :- $NSPACE(f(n)) \subseteq DTIME(2^{O(f(n))})$



No. of paths $\leq b^{2^{cf(n)}}$!!

But no. of configurations is $\leq 2^{cf(n)}$.

In case of DSPACE machine config graph had out degree ≤ 1.

For NSPACE config graph has outdegree $\leq b$.

But total no. of vertices is $\leq 2^{cf(n)}$

Note: We can convert any TM to have unique accepting config.

We need to check if there is a path from $C_{start,w}$ to C_{accept}. We can do this in $O(|V|^2)$ time $= O((2^{cf(n)})^2) = O(2^{2c\,f(n)})$.

$NL \subseteq P$

PATH = $\{ \langle G, s, t \rangle \mid$ G is a directed graph where there is a path from vertex s to vertex t $\}$

Exc:- Show that PATH $\in$ NL

Guess the next vertex and keep a counter of it.

Can be done at most 'n' times
↓
no. of vertices.

Guess the path and verify.

NL-Completeness

B is NL-complete if

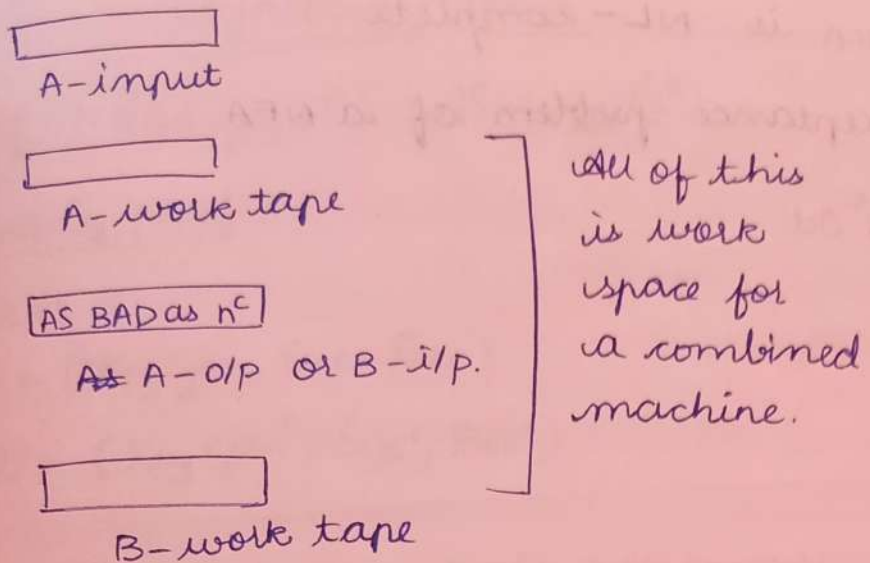
(1) $B \in NL$, and

(2) $\forall A \in NL$, $A \leq_L B$.
↓
log space reductions.

Logspace Reductions: We say $A \leq_L B$; or A is logspace reducible to B if there is a det. log space bounded M, that computes f such that $w \in A \iff f(w) \in B$

Suppose $A \leq_L B$. If $|w| = n$, then what is an upper bound on $|f(w)|$?
↓
can be as bad as n^c

Want to say $A \leq_L B$ and $B \in L \Rightarrow A \in L$



* Let M - decider for B.
  R - reduction machine from A to B.

→ Start M. Suppose the input A instance is w.

→ When M needs the j^{th} symbol of f(w)

→ Remember j. (~~log~~ $\log(n^c)$ at most)

→ Run R till the j^{th} symbol of f(w) is produced.

→ Use the j^{th} symbol.

We repeat this process whenever we need a symbol from f(w). So we end up re-running R many times but only write down the j^{th} symbol. So we get

$$A \leq_L B \text{ and } B \in L \Rightarrow A \in L$$

Theorem :- PATH is NL-complete

Part (2) involves looking at an NL language as a configuration graph.

12/09/2024
Thursday

CS5510 Computational Complexity

Lec10

Savitch's Theorem ('70)

For $f(n) \geq \log n$

$NSPACE(f(n)) \subseteq DSPACE((f(n))^2)$

→ Simulating an NSPACE machine can be seen as checking if there is a path in config graph.

Exercise :- PATH is NL-complete. Go through notes / video.

Watch out for the details. We need $O(\log n)$ space bound.

→ Let $A \in NSPACE(f(n))$. There is an $O(f(n))$ NSPACE TM that decides A.

How many configurations does this NTM have? $\leq 2^{df(n)}$

The configuration graph has $2^{df(n)}$ vertices. We need to check if there is a path from C_{start} to C_{accept}.

PATH (v_1, v_2, t) : (Is there a path from v_1 to v_2 of length $\leq t$)

If $t=1$, ACCEPT if $v_1 = v_2$ or (v_1, v_2) is an edge.

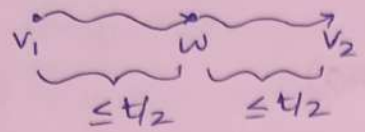
Else for all vertices w

Run PATH $(v_1, w, t/2)$

Run PATH $(w, v_2, t/2)$

ACCEPT if both accept.

REJECT if not accepted yet.



Analysis of space needed

Initial call $v_1 = C_{start}$ $v_2 = C_{accept}$ $t = 2^{df(n)}$

During the recursive call, we need to store v_1, v_2, w, t.

↳ Each one needs $df(n)$ space.

Depth is $\leq \log t \approx df(n)$ space

Total space $= O(f(n)) \cdot df(n) = O((f(n))^2)$.

Consequences

(1) $NL \subseteq DSPACE((\log n)^2)$

(2) $NSPACE(n) \subseteq DSPACE(n^2)$

(3) $NSPACE(n^k) \subseteq DSPACE(n^{2k})$

(4) Non-det. poly space

$$NSPACE = \bigcup_{k=1}^{\infty} NSPACE(n^k)$$

$$\subseteq \bigcup_{k=1}^{\infty} DSPACE(n^{2k})$$

$$\subseteq \bigcup_{k=1}^{\infty} DSPACE(n^k)$$

$$= PSPACE$$

$\Rightarrow$ NSPACE = PSPACE

NL = co-NL (Immerman - Szelepcsenyi)

co-NL = $\{L \mid \bar{L} \in NL\}$.

It is enough to show $\overline{PATH} \in NL$.

PATH = $\{\langle G, s, T\rangle \mid G$ is a directed graph and there is a path in G from s to t$\}$.

PATH is NL-complete.

So, $\overline{PATH}$ is co-NL complete

For all $L \in$ co-NL, $L \leq_L \overline{PATH}$

If $A \leq_L B$ and $B \in L \Rightarrow A \in L$

$x \quad f(x)$.

If $\overline{PATH} \in NL$, we get co-NL $\subseteq$ NL.

~~since $\overline{PATH}$ is co-NL complete~~

We know PATH is NL complete.

$\overline{PATH} \in NL \Rightarrow PATH \in$ co-NL

So $\forall L \in NL$, we get ~~$L \leq$~~ $L \subseteq$ co-NL

So NL $\subseteq$ co-NL.

co-NL $\subseteq$ NL. Consider $A \in NL \Rightarrow \bar{A} \in$ co-NL.

We have $\bar{A} \in NL \Rightarrow A \in$ co-NL

So NL $\subseteq$ co-NL.

$\overline{PATH}$ = $\{\langle G, s, t\rangle$ | There is no path from s to t $\}$.

We have to non-det verify the non-existence of a path.

If there is an NTM for language A, then flipping Accept/Reject of the NTM need not give the NTM for $\bar{A}$.

Given s, t and a count c of the no of vertices reachable from s, can we verify the non-existence of s-t path?

One possibility: We guess c vertices (such that none of them are t) and verify that each one of them is reachable from s.

- This is OK when c is small. But not when $c \geq n/2$.

→ count = 0
→ Run a counter from i=1 to |V|
→ For each i, guess if i is reachable or not from s. (If i is guessed to be not reachable, we move to the next i)
- If i is guessed to be reachable, verify that it is by guessing a path (similar to PATH ∈ NL proof)
- If correctly verified, increment count.
- If we cannot verify, reject.
- If t is guessed to be reachable, reject.

→ If count = c, we accept Else reject.

Space needed: Count, i, and verifying that a vertex is reachable.

→ O(log n) space

How do we get c, no. of reachable vertices from s?

→ We need a non-det log space computation that either rejects, or correctly computes c. We need at least one correct path and no path should give incorrect count.

Let A_i = set of vertices reachable from s in $\leq i$ steps.

$c_i = |A_i|$

$A_0 = \{s\}$ $\quad c_0 = 1.$

$c = c_{|V|}$

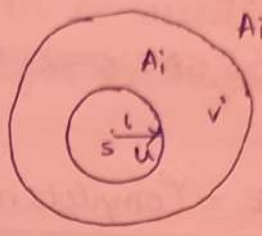
We compute $c_0, c_1, c_2, \ldots, c_{|V|}$.

19/09/2024
Thursday

CS5100: Computational Complexity Lec 11

NL = co-NL

Let $c_0 = 1$
For $i = 0$ to $|V|-1$
  let $c_{i+1} = 1$
  For each node $V \neq S$
    let $d = 0$
    For each node $u \in G$
      Non-det do or skip
        CHECK ~~PATH~~ PATH (G, s, u, i) → Rejects if not correct
        $d = d+1$
        If $(u,v) \in$ Edge or $u = v$
          ~~$c_i = c_{i+1}$~~ $c_{i+1} = c_{i+1} + 1$
    If $d \neq c_i$ Reject
  Return $c_{|V|}$



Memory needed: i, c_i, c_{i+1}, d, u, v.
(whatever is needed for CHECK PATH)

Each one needs $\log|V|$ space.

$L \longrightarrow NL = co\text{-}NL \longrightarrow P$ → NP, co-NP → Σ_2, Π_2 → Σ_3, Π_3 ... → PSPACE

This means $L \subseteq NL$.

PSPACE = N-SPACE

The only seperation known here ~~is that~~ are $L \neq$ PSPACE.
$NL \neq$ PSPACE

True

Fully Quantified Boolean formula (TQBF)
↓
Everything is quantified and no restrictions on no. of alternations (flips of $\forall$ and $\exists$).

SAT : $\exists x_1, x_2, \ldots, x_n \ \phi(x_1, x_2, \ldots, x_n)$

Tautology : $\forall x_1, x_2, \ldots, x_n \ \phi(x_1, x_2, \ldots, x_n)$

$\phi \in SAT \iff \bar{\phi} \notin TAUT$

PSPACE - Completeness

B is PSPACE complete if

(1) $B \in PSPACE$

(2) $\forall A \in PSPACE, A \leq_p B.$

Polytime

There is P-completeness which uses a reduction "weaker" than P.

$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \ \psi(x_1, x_2, \ldots, x_n)$ where each $Q_i \in \{\exists, \forall\}$

TQBF = $\{\langle \phi \rangle \mid \phi$ is a true fully quantified Boolean formula$\}$

TQBF is P-SPACE complete.

(1) $TQBF \in PSPACE$

- No quantifiers - formula only has constants.
- First quantifier is $\exists$: $\psi(x_1, x_2, \ldots, x_n)$
- First quantifier is $\forall$:

Recursively check $\psi(T, x_2, x_3, \ldots)$ and $\psi(F, x_2, x_3, \ldots)$ and ACCEPT iff one of them

accepts.

Second one, ACCEPT iff both accept.

PADDING : Padding argument can be used to "inherit" relations applicable for a higher value / function.

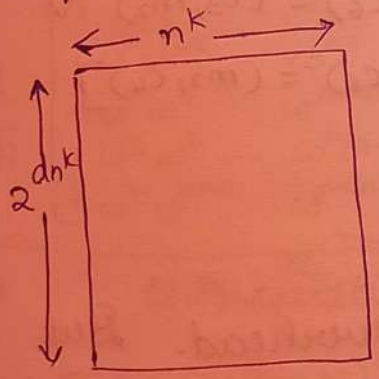
(2) $\forall A \in PSPACE$, $A \leq_P TQBF$

There is a PSPACE machine M that decides A.

We construct, given w, a TQBF instance ϕ such that

$$w \in A \iff \phi \in TQBF$$

Computation table of M



No. of rows $\leq$ no. of configurations

$= 2^{O(n^k)} = 2^{dn^k}$

We need to encode the decision of M into ϕ.

$\phi_{c_1, c_2, t}$: Needs to check if we can reach c_2 from c_1 in $\leq t$ steps ?

ϕ_{cstart}, $\phi_{caccept}$, 2^{dn^k}

$$\phi_{c_1,c_2,t} = \exists m_1 \left[\phi_{c_1, m_1, t/2} \wedge \phi_{m_1, c_2, t/2} \right].$$

$$= \exists m_1 \left[\exists m_2 \left(\phi_{c_1, m_2, t/4} \wedge \phi_{m_2, m_1, t/4} \right) \wedge \exists m_3 \left(\phi_{m_1, m_3, t/4} \wedge \phi_{m_3, c_2, t/4} \right) \right]$$

This is not yielding any reduction in size

We use a trick called "folding"

$$\phi_{c_1,c_2,t} = \exists m_1, \forall (c_3,c_4) \in \{(c_1,m_1),(m_1,c_2)\} \; [\phi_{c_3,c_4,t/2}]$$

$$= \exists m_1 \; \forall c_3,c_4 \; [(c_3,c_4) = (c_1,m_1) \vee (c_3,c_4) = (m_1,c_2)]$$

$$\Downarrow$$

$$\phi_{c_3,c_4,t/2}$$

Exercise:

Equality (=) and implication ($\Rightarrow$) can be converted into Boolean formula.

$$\phi_{c_3,c_4,t/2} = \exists m_2 \forall_{c_5,c_6} [(c_5,c_6) = (c_3,m_2) \vee (c_5,c_6) = (m_2,c_4)]$$

$$\Downarrow$$

$$\phi_{c_3,c_6,t/4}$$

At each "step", we use $O(n^k)$ overhead. But we cut down the t by half.

$$\phi_{c_1,c_2,t} = \exists m_1 \forall_{c_3,c_4} \exists m_2 \forall c_5,c_6 \exists m_3 \cdots \cdots [\phi_{c_i,c_{i+1},1}]$$

We can write a Boolean formula of size $O(n^k)$.

For the initial step, we need to have $t = 2^{dn^k}$. The depth (no. of folds) is $\log t = dn^k = O(n^k)$

So the total space needed $= O(n^k) * O(n^k)$

$$= O(n^{2k})$$

23/09/2024
Monday

Computational Complexity Lec 12

GAMES & PSPACE COMPLETENESS

→ PSPACE COMPLETE

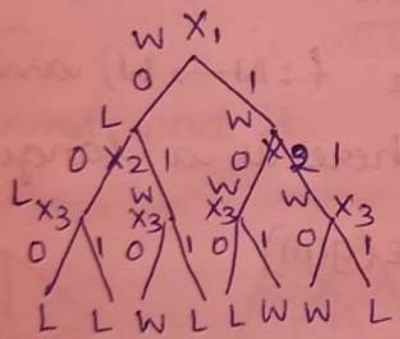
FORMULA-GAME: $\{ \langle \phi \rangle \mid$ Player 1 has a winning strategy $\}$.

$\phi = \exists x_1 \forall x_2 \exists x_3 \forall x_4 \cdots (\psi)$ → Unquantified

P1: sets the odd x_i → Wants to make ψ true.

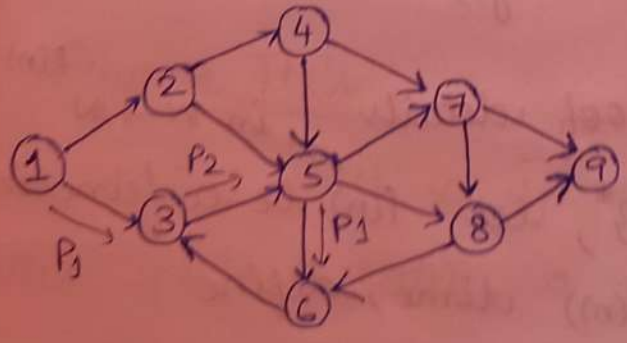
P2: sets the even x_i → Wants to make ψ false.

$\phi = (x_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3)$

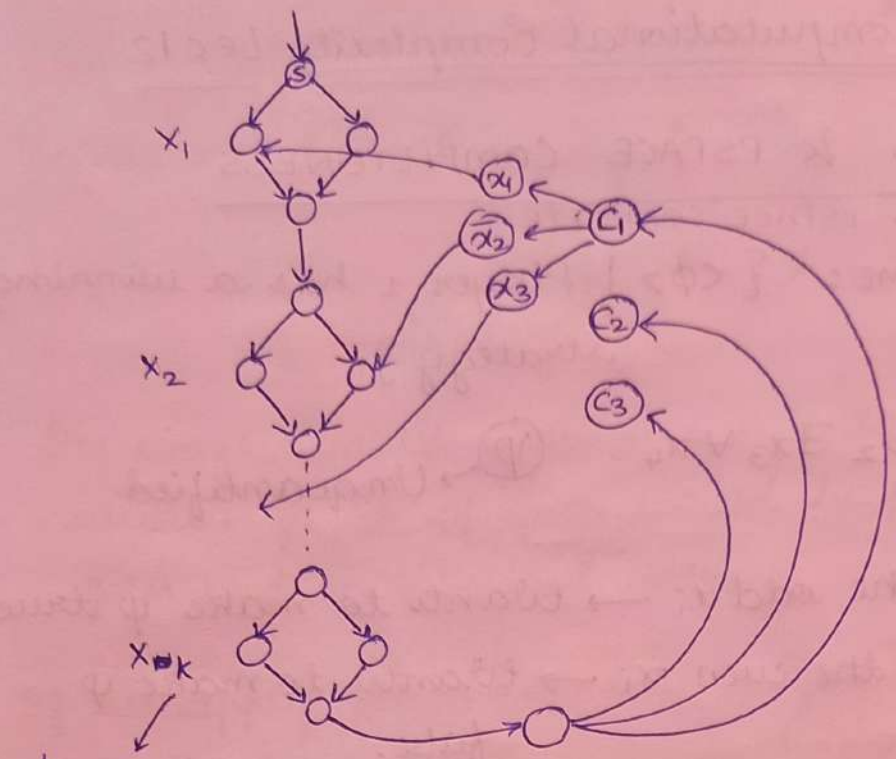


GENERALIZED-GEOGRAPHY

$GG = \{ \langle G, b \rangle \mid$ P1 has a winning strategy starting from point b in G $\}$



GG is PSPACE complete. Reduction from TQBF.



Space Hierarchy Theorem

For any space constructible $f: \mathbb{N} \to \mathbb{N}$, and $g(n)$ such that $g(n) = O(f(n))$ there is a language

$A \in DSPACE(f(n))$ $\quad A \notin DSPACE(g(n))$

$$DSPACE(g(n)) \subsetneq DSPACE(f(n))$$

In Time Hierarchy, we needed

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

Here (in space hierarchy),

Exercise: Read the proof details in $\mathbb{N} \to \mathbb{N}^{f(n) \geq n}$

Exercise 2: Let $L \subseteq \{0,1\}^*$, let $f(n)$ be a function computable in $O(f(n))$ time, define

$$L_f = \{ x \#^{f(|x|)} \mid x \in L \}$$

$L_f \subseteq \{0, 1, \#\}^*$

* $L \in DTIME(f(n)) \implies L_f \in DTIME(n)$

* If f is polynomial.

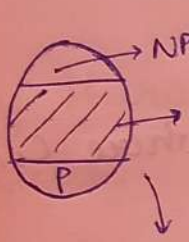
$L \in P \iff L_f \in P$.

Ladner's Theorem

If $P \neq NP$, are there languages in NP\P that are not NP-complete?

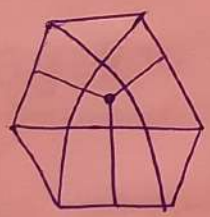
Ladner's theorem says "Yes".

"NP intermediate"



One proof is using "padding". Take SAT and pad it.

Petersen Graph

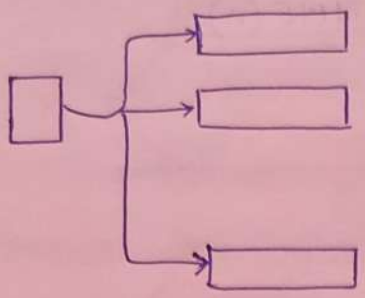


These two graphs are isomorphic.

FACTORING = $\{\langle N, d\rangle \mid N$ has a prime factor $\leq d\}$

Oracle TM's:

It is a TM with a special query tape and three special states $q_?$, q_{YES}, q_{NO}.



Oracle TM can ask the oracle questions by writing queries in the oracle tape. The oracle answers in the next step.

An oracle TM M with access to oracle A is denoted by M^A

can be generalized to C_1^A or $C_2^{C_1}$ where C_1, C_2 are complexity classes.

For language A, P^A denotes all polytime det TM with access to A oracle. Similarly we can define NP^A.

Turing Reductions

Defn: A – Turing reduces to B ($A \leq_T B$) if there is an oracle TM M such that

$A = L(M^B)$ (we need some resource restriction)

Example:- $\overline{SAT} \leq_T SAT$

Ask the SAT oracle if input ϕ is satisfiable and flip the response.

$\overline{SAT} \in P^{SAT}$

→ $NP \cup co\text{-}NP \subseteq P^{SAT}$

→ If $A \in P$, then $P^A = P$ (Exercise)

But $P^{SAT} \neq NP$, if $NP \neq co\text{-}NP$.

→ $NP^{SAT} = \Sigma_2^P$ } (We'll see now)

→ $NP^{\mathcal{O}SAT_i} = \Sigma_{i+1}^P$

$co\text{-}NP^{\mathcal{O}SAT_i} = \Pi_{i+1}^P$

A not NP-complete.

Assuming $P \neq NP$, is there $A \in NP$ such that $NP^A \neq NP$?

Poly. Hierarchy can be defined in terms of oracles.

$CoNP^{SAT} = \Pi_2$

Theorem: $NP^{SAT} = \Sigma_2$

Proof: Let $L \in \Sigma_2$ (To prove $\Sigma_2 \subseteq NP^{SAT}$)

$x \in L \iff \exists y \; \forall z$, we have $V(x, y, z) = 1$.

$V_{x,y}(z) = 1$.

base

The 1 NP Machine can handle "$\exists y$".

$\forall z \; V(x, y, z) = 1 \iff \forall z \; V_{x,y}(z) = 1$.

$\iff$ NOT $(\exists z, V_{x,y}(z) = 0)$

We can define ϕ that is true when $V_{x,y} = 0$

$\iff$ NOT $(\exists z \; \phi(z) = TRUE)$

↓

can be answered by SAT oracle.

This completes $\Sigma_2 \subseteq NP^{SAT}$

26/09/2024
~~Tuesday~~ Thursday.

Computational Complexity Lec 13

Randomized Complexity Classes

(RP, co-RP, BPP, ZPP)

Problem: I/P: $A, B, C \in \mathbb{R}^{n \times n}$ ($n \times n$ matrices)

O/P: Answer Yes, if $AB = C$

No otherwise.

Naive Algo: Multiply A and B $\rightarrow O(n^3)$ / $O(n^\omega)$

Compare it to C entrywise

$O(n^2)$

ω = matrix Mult const ≈ 2.7 < 3

Randomized Algorithms

- Uses random coins (Execution of algorithm depends upon outcome of random coin tosses)
- Pr [answer is correct] > 0.99.

for ANY input.

Freivald's Algorithm:

- Pick $x \in_R \{0,1\}^*$ uniformly at random.
- Check if $ABx = Cx$. $[A(Bx) = Cx]$
  - Output YES, if ans is Yes
  - Output NO, if ans is No.

Running time = $O(n^2)$

Theorem: If $AB \neq C$, then $Pr[ABx = Cx] \leq \frac{1}{2}$.

Proof: Suppose $(AB-C)[ij] \neq 0$

Let $AB - C \triangleq D$., $D_{ij} \neq 0$.

$(Dx)[i] = \sum_j D_{ij} x_j$.

$= \sum_k D_{ik} x_k$

$= D_{ij} x_j + \underbrace{\sum_{k \neq j} D_{ik} x_k}_{\alpha}$

$\Pr[(Dx)_i = 0] = \Pr[\alpha = 0] \cdot \Pr[\underbrace{D_{ij} x_j}_{1/2} = 0 \mid \alpha = 0] +$

$\Pr[\alpha \neq 0] \cdot \Pr[\underbrace{D_{ij} x_j = -\alpha}_{\leq 1/2} \mid \alpha \neq 0]$

$\leq 1/2$

8)

Guarantee: If $AB = C$, algo o/ps the right answer (YES)

If $AB \neq C$, algo o/ps wrong answer w.p $\leq 1/2$

One-sided error.

- Repeat Freivald's algorithm 100 times.
- If it answers NO in any run, o/p NO

  Else o/p YES.

Analysis: When $AB = C$, $\Pr[\text{o/p is YES}] = 1$.

When $AB \neq C$, $\Pr[\text{o/p is YES}]$

$= \Pr[\text{all runs o/p YES}]$

$\leq \left(\frac{1}{2}\right)^{100}$.

$= \frac{1}{2^{100}} << 0.01$.

Run time = $100n^2$

Why do we use randomized Algorithms?

— (May) save on running time.

— Easier to state / implement.

— Practical.

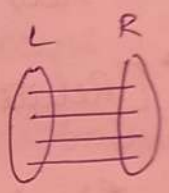
— Complexity Classes for problems that admit randomized algorithms.

Perfect Matchings in Bipartite graphs

I/p: $G = (L \uplus R, E)$, $|L| = |R| = n$, bipartite.

O/p: 1 if G has a perfect matching.

0 o/w.

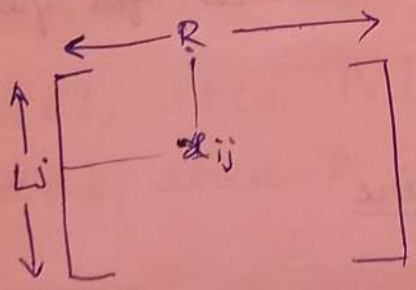


Deterministic: ~~Min~~ Max flow / ~~Max~~ Min cut

Linear Programming based

Edmond's Algo.

Randomized:



$M[i,j] = x_{ij}$ if $(i,j) \in E$.

$= 0$ o/w.

(x_{ij}'s are variables)

Claim: det(M) is the zero polynomial

$\Leftrightarrow$ G does not have a perfect matching.

$\det(M)$ is a multivariate polynomial in x_{ij}'s.

degree $(\det(M)) \leq n$.

$$\det(M) = \sum_{\sigma \in S_n} (-1)^{sgn(\sigma)} \prod_{i \in [n]} M_{i\sigma(i)}$$

σ → Permutation of $[n]$

S_n → Set of all permutations.

i/p: $n = 4$: $(2\ 1\ 3\ 4)$ $\quad (-1)^{sgn(\sigma)} M_{12} M_{21} M_{33} M_{44}$

Checking if a polynomial $P(x_1, \ldots x_r)$ is zero.

- Has an efficient randomized algorithm.

[Related to PIT (Polynomial Identity Testing)]

- Check if $P(x_1, x_2, \ldots, x_r) = 0$ at some randomly chosen point $(x_1, x_2, \ldots, x_r)$

↳ Analyze using "Schwatz - Zippel lemma".

- Faster than other det. methods for perfect matching.

Probabilistic Turing Machine

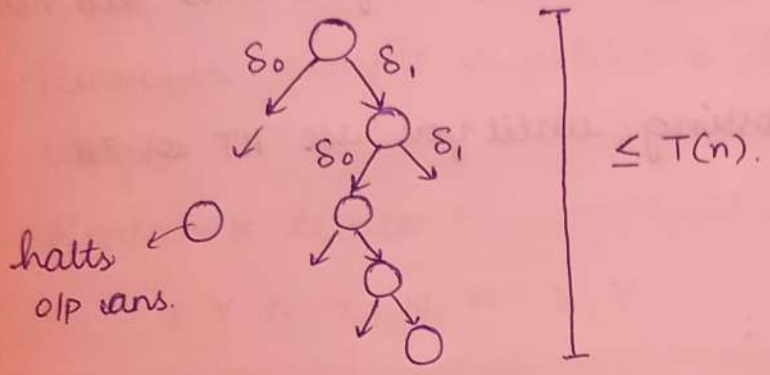
~~At every step~~

Has two transition functions δ_0, δ_1

- At every step chooses to follow δ_0 w.p $1/2$ and δ_1 w.p $1/2$.
- Choice at any step is independent of previous

steps.

- O/P: 1 or 0.
- M(x) runs in time T(n), if it halts within T(n) steps regardless of random choice.



NP: Accepts if ∃ a path leading to accept.

RP: Languages L for which there is a probabilistic polytime T.M s.t

$x \in L$, $Pr[M(x) = Accept] \geq \frac{1}{2}$

$x \notin L$ $Pr[M(x) = Accept] = 0$.

$L = \{(A, B, C) : AB = C\}$

If $x \in L$, then $Pr[$Frievald's algo accepts$] = 1$.

$x \notin L$, then $Pr[$Frievald's algo accepts$] \leq \frac{1}{2}$

Co-RP: Languages L for which there is a probabilistic polytime T.M s.t.

$x \in L$: $Pr[x \text{ accepts}] = 1$

$x \notin L$: $Pr[x \text{ accepts}] \leq \frac{1}{2}$.

eg:- (Checking matrix mult. is in Co-RP).

- Const $(\frac{1}{2})$ of error can be made to any $\frac{1}{2^k}$ for $k = poly(n)$ [Boosting] by repetition.

- $L \in RP \iff \bar{L} \in co\text{-}RP$. [Exercise]

- Can also use biased coins, definitions do not change.

  Idea: Keep on tossing until you see HT or TH.

  (Look up!)

Primality Testing:

I/p: Integer n. [I/P : $(\log n)$ - bits]

O/P: "PRIME" if n is prime

"Non-Prime" if n is composite.

Want: algo that runs in poly time $\equiv (\log n)^c$ for some constant c.

[1980] Miller Rabin Test: Primality testing $\in$ co-RP.

[~1992] Adleman Huang: Also in RP.

[~2000] AKS: in P.

BPP: (Bounded-Error Probabilistic Polytime)

Language L is in BPP if there is a probabilistic polytime machine M s.t.

$$\forall x \in L : Pr[M(x) = Acc] \geq 2/3$$

$$\forall x \notin L : Pr[M(x) = Acc] \leq 1/3$$

OR: $[\forall x : Pr[M(x) = L(x)] \geq 2/3]$

- Error constants don't matter. Only need

$$Pr[M(x) = L(x)] \geq \frac{1}{2} + \frac{1}{2^{p(|x|)}}$$

$\frac{1}{2} + \frac{1}{2^{n^{100}}}$ correctness can be boosted to $2/3$.

- However simple repetitions does not work.

Why?

Repeat k times

Y Y N Y N N Y Y

→ need to aggregate the answers properly.

→ Take majority of k outcomes.

→ Analyze using Chernoff bounds.

30/09/2024
Monday

Computational Complexity Lec 14

$\Sigma_2^P = NP^{SAT}$

We saw, $\Sigma_2^P \subseteq NP^{SAT}$

$\exists x \; \forall y \; V(x, y, z)$

↓

Nested query to SAT oracle.

$NP^{SAT} \subseteq \Sigma_2^P$

Given ~~$L \in \Sigma_2^P$~~ $L \in NP^{SAT}$: L is ~~divided~~ decided by an NTM, say N with access to a SAT oracle.

Given input x, L makes non-det choices, and queries to SAT oracle.

Given x, the NTM makes,

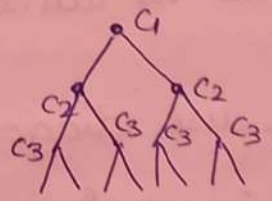
non-det choices : $c_1, c_2, \ldots, c_m$.

Queries to SAT oracle } $\phi_1, \phi_2, \ldots, \phi_k$.

Answers from oracle } $a_1, a_2, \ldots, a_k$.

If $a_i = 1$, then $\phi_i \in SAT$

If $a_i = 0$, then $\phi_i \notin SAT$



↓

$\forall v_i$, we need $\phi_i(v_i) = FALSE$

$x \in L \iff \exists \; c_1, c_2, c_3 \ldots c_m$ that leads N to accept.

$\iff \exists \; [c_1, c_2, \ldots c_m, a_1, a_2, \ldots, a_k, u_1, u_2, \ldots, u_k]$
$\forall \; [v_1, v_2, \ldots, v_k]$

~~$a_i = 1 \implies \phi_i(u_i) = TRUE$~~

such that N accepts with the non-det choices $c_1, c_2, \ldots c_m$ and oracle responds $a_1, \ldots, a_k$. and

$$[a_i = 1 \Rightarrow \phi_i(u_i) = \text{TRUE}] \wedge$$

$$[a_i = 0 \Rightarrow \phi_i(v_i) = \text{FALSE}]$$

Baker Gill Solovay Theorem (⇉5)

P vs NP cannot be resolved using proofs that relativize.

↳ that are valid with oracles.

↳ Mainly, diagonalization.

Theorem:

(1) There is an oracle A (→ Powerful A) such that $P^A = NP^A$

You cannot show $P \neq NP$ using techniques that relativize.

(2) There is an oracle B such that $P^B \neq NP^B$.

↓

We use diagonalization. We show a language U_B in NP^B, and use diag. to show $U_B \notin P^B$.

$A = \{ \langle M, x, 1^n \rangle \mid$ M outputs 1 on input x within 2^n steps $\}$ → M accepts x.

↳ n in unary

$A \in EXP = \bigcup_{k=1}^{\infty} DTIME(2^{n^k})$

Why? → We can run M on x for 2^n steps

Input length = $|M| + |x| + n \geq n$.

Claim 1: $EXP \subseteq P^A$

Given $L \in EXP$, we can decide it in P^A.

~~Suppose L decides on x in $2^{|x|^k}$ time.~~

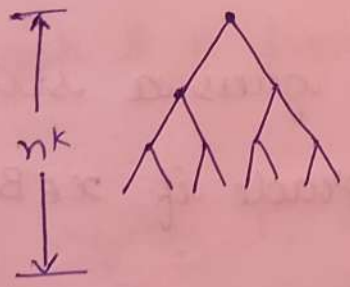
Suppose M decides if $x \in L$ in $2^{|x|^k}$ time.

We ask the A oracle $\langle M, x, 1^{|x|^k} \rangle$ as query.

Accept $\iff$ oracle says yes.

Claim : $NP^A \subseteq EXP$.

Proof: Suppose $L \in NP^A$. This means there is an NTM N that runs in poly. time with access to A oracle.



Comp. time of N.

We simply go through all possible computation paths of N. If N runs in n^k time, there will be atmost b^{n^k} paths

$= 2^{\log b \cdot n^k}$

where b is the max. branching factor.

We also need to simulate the oracle, which is in EXP TIME. (Multiplying two exponential functions gives exp function).

Together, we infer $P^A = NP^A = EXP$

Oracle B such that $NP^B \neq P^B$

— We will build B as part of the proof.

— We will construct U_B such that $U_B \notin P^B$, and $U_B \in NP^B$

$U_B = \{ 1^n \mid B \text{ contains a string of length } n \}$

$B = \{0, 1, 000, 101, 111, 00001, 11111111\}$

$U_B = \{1, 111, 11111, 11111111\}$

$B \in P^B$. So we need a trickier language U_B such that $U_B \notin P^B$.

Why is $U_B \in NP^B$?

Given 1^n, we simply guess a string $x \in \{0,1\}^n$ and ask oracle if $x \in B$.

— Let $M_1, M_2, M_3, \ldots$ be a listing of oracle TM's. We will construct B such that none of the M_i's can decide U_B in poly. time.

B is constructed in stages. After, stage i, we ensure that M_i (with access to B oracle) does not decide U_B.

→ In each stage, we will decide on the membership of some strings in B.

Stage i (Want to fool M_i)

→ Let the longest string whose status has been determined have length l.

→ Set $n = l+1$.

→ Run M_i^B on 1^n for $\frac{2^n}{10}$ steps.

- When M_i queries the oracle if $x \in B$ If the status of x has already been determined, then answer consistently.

If x has not been decided, then answer $x \notin B$.

→ If M_i accepts 1^n, we exclude all strings of length n from B. For all $x \in \{0,1\}^n$, $x \notin B$.

→ If M_i rejects 1^n, then we include a string of length n (whose status is so far not determined) in B. There will be such a string available, since all the strings in $\{0,1\}^n$ were available before stage i. During stage i, we decided on at most $2^n/10$ strings.

In stage i, we ensure that M_i^B does not decide v_B. ~~$\phi\theta$, $v_B \notin$~~ ~~not decide~~

$\phi\theta$, $v_B \notin P^B$.

Next class : ZPP

8/10/2024
Thursday

Computational Complexity Lec 15

RP – Set of languages L s.t there is a P.P.T machine M

$x \in L \Rightarrow Pr(M(x) = acc) \geq 1/2$ $\geq 1 - e^{-q(|x|)}$

$x \notin L \Rightarrow Pr(M(x) = acc) = 0.$

$L \in$ co-RP $\Leftrightarrow \bar{L} \in RP.$

$L \in$ co-RP if there is M s.t

$x \in L \Rightarrow Pr(M(x) = acc) = 1$

$x \notin L \Rightarrow Pr(M(x) = acc) \leq \frac{1}{2}.$

$P \subseteq RP \subseteq NP$

RP has false positives but not false negatives.

$L \in BPP$ if there exists M

$x \in L \Rightarrow Pr(M(x) = acc) \geq \frac{2}{3}.$ $\left(\frac{1}{2} + \frac{1}{p(|x|)}\right)$

$x \notin L \Rightarrow Pr(M(x) = acc) \leq \frac{1}{3}$ $\left(\frac{1}{2} - \frac{1}{p(|x|)}\right)$

RP, co-RP $\subseteq$ BPP

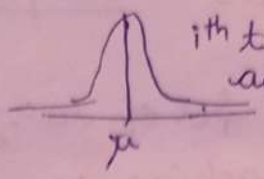
Chernoff Bound

Let $X_1, X_2, \ldots, X_n$ be independent 0/1 r.v's with $P[X_i = 1] = p \ \forall i.$

Let $X = \sum_{i=1}^{n} X_i.$ Let $\mu = np = E[X].$

For $0 \le \varepsilon \le 1$

$X_i = 1$ if i^{th} trial accepts.



$$Pr(X \ge (1+\varepsilon)\mu) \le e^{-\frac{\varepsilon^2}{3}\mu}$$

$$Pr(X \le (1-\varepsilon)\mu) \le e^{-\frac{\varepsilon^2}{2}\mu}.$$

If we have prob. alg that has $Pr[acc] \ge \frac{1}{2} + \frac{1}{P(x)}$ in YES instance and $Pr[accept] \le \frac{1}{2} - \frac{1}{P(x)}$ in NO instance, then repeat

$t(x) = p.\ 2p^2(x) \ln 4$ and choose majority.

Suppose we have a YES instance.

Then $Pr(M^*(x) \ne acc) \le Pr\left[\sum_{i=1}^{t(x)} X_i < \frac{t(x)}{2}\right]$

$$= Pr\left[\Sigma X_i \le p\,t(x) - \frac{t(x)}{P(x)}\right]$$

$$= Pr\left[\Sigma X_i \le p\,t(x)\left[1 - \frac{1}{pP(x)}\right]\right]$$

Applying Chernoff's we get

$$\le e^{-\frac{\varepsilon^2}{2}\mu}$$

$$= e^{-\frac{1}{p^2p^2(x).2} \times p\,t(x)} \to p\,2\,p^2(x).\ln 4$$

$$= \frac{1}{4}$$

Monte Carlo and Las Vegas Algorithms

↓ Running time is fixed but there could be error.

RP, co-RP, BPP.

↘ Running time may vary but no error.

Running time will be a random variable.

ZPP

(Zero Error probabilistic poly time)

$L \in ZPP$ if there exists prob. polynomial time M s.t M can say YES, NO or DON'T KNOW, s.t

(1) $\forall x$ when M says YES or NO, it is the correct answer.

$$\forall x \quad Pr[M(x) = \text{DON'T KNOW}] \leq \frac{1}{2}.$$

(2) Another defn:

ZPP is a class of languages for which there is probabilistic poly. time M that runs in expected time $O(t(n))$ where $t(n)$ is a polynomial such that when M halts, it gives the correct answer.

Polynomial Identity Testing (PIT)

Is a given polynomial identically equal to zero?

↓

Is the poly. equal to zero polynomial?

$$\underbrace{(x-1)(x+3)(x-6)}_{F(X)} \stackrel{?}{=} \underbrace{x^3+4x^2-12x+18}_{G(x)}$$

Are the above two the same polynomial.

In other words, is $F(X) - G(X) \equiv 0$?

↳ co-RP algorithm exists using Schwartz-Zippel.

(1) to (2) $\Rightarrow$ Repeat till we get YES / NO.

(2) to (1) $\Rightarrow$ Run for $t(n)$, time, and if program has not terminated till then,

Output DON'T KNOW.

What's the prob. that we would not have halted in $ct(n)$ time? $\leq \frac{1}{c}$ (use Markov's)

~~Using~~ Markov's inequality

If X is a non-negative R.V.

$$P(X \geq \alpha E(X)) \leq \frac{1}{\alpha}.$$

ZPP $\subseteq$ RP.

Let M be a ZPP machine.

We construct M' such that when M says DON'T KNOW $\longrightarrow$ M' says NO.

When $x \in L$, $Pr(M'(x) = \overset{YES}{\cancel{acc}}) \geq \frac{1}{2}$.

When $x \notin L$, $Pr(M'(x) = YES) = 0$.

M' is an RP machine.

ZPP $\subseteq$ co-RP:

Say YES when M says DON'T KNOW.

ZPP $\subseteq$ RP $\cap$ co-RP.

Theorem: ZPP = RP $\cap$ co-RP.

Proof: We have seen ZPP $\subseteq$ RP $\cap$ co-RP.

We need to show RP $\cap$ co-RP $\subseteq$ ZPP.

Suppose ~~$L \in$~~ $L \in$ RP $\cap$ co-RP

There is M_1, RP machine for L.

M_2, co-RP machine for L

We need to build a ZPP machine for L.

M_1 : No false positives

M_2 : NO false negatives

Run both M_1 and M_2 on the input.

If M_1 accepts, we accept.

If M_2 rejects, we reject.

If M_1 rejects and M_2 accepts, say DON'T KNOW.

This is a ZPP machine.

Monday
14/10/2024

Computational Complexity Lec 16

$BPP \subset \Sigma_2$

BPP, ZPP are closed under complement.

$L \in BPP$ The above implies $L \in \Sigma_2$

$\bar{L} \in BPP \Rightarrow \bar{L} \in \Sigma_2 \Rightarrow L \in \Pi_2$

$\therefore BPP \subset \Sigma_2 \cap \Pi_2$

Σ_2: $x \in L \Leftrightarrow \exists y \, \forall z$, $|y|, |z|$ are poly$(|x|)$
$V(x,y,z) = 1$

Π_2: $x \in L \Leftrightarrow \forall y \, \exists z, \ldots V(x,y,z) = 1$.

To prove $BPP \subseteq \Sigma_2$.

We look at the randomized alg. as a det alg. that takes x, r as inputs

(x: actual input; r: random bits)

$M(x,r)$ $\quad 1 - \frac{1}{2^n}$

If $x \in L$, then for $\geq \frac{2}{3}$ choices of r, $M(x,r)$ accepts.

$\quad \frac{1}{2^n}$

If $x \notin L$, then for $\leq \frac{1}{3}$ choices of r $M(x,r)$ ~~rejects~~ accepts.

where $n = |x|$ length of x.

For a fixed x, let S_x denote the set of r such that M accepts.

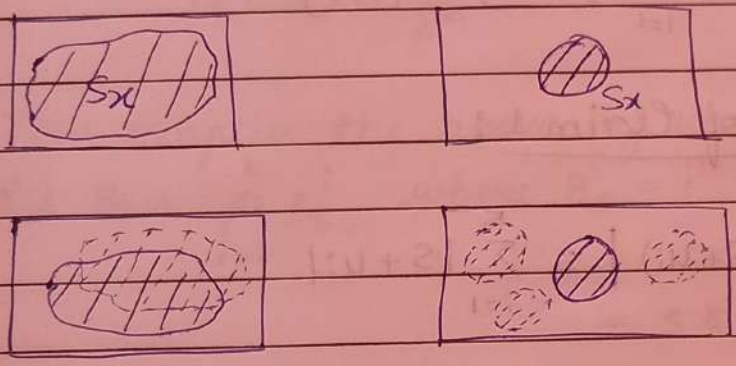
$$S_x = \{ r \mid M(x,r) = 1 \}$$

Let us assume the alg. takes m bits random string on input x. → m = poly(|x|).
That means $S_x \subseteq \{0,1\}^m$

If $x \in L$, then $|S_x| \geq 2^m\left(1 - \frac{1}{2^n}\right)$

If $x \notin L$, then $|S_x| \leq \frac{2^m}{2^n}$

$x \in L$ $\qquad$ $x \notin L$



Q: Can we cover the entire universe ($\{0,1\}^m$) by k shifted copies of S_x?

YES $\Leftrightarrow$ $x \in L$.

$x \in L \Leftrightarrow \exists$ k shifts of S_x s.t $\forall\, r \in \{0,1\}^m$
"there is one of the k shifts that lead to $M(x,r) = 1$"

A shift is simply XOR with a specific m-bit string.

Given $S \subseteq \{0,1\}^m$ and $u \in \{0,1\}^m$

$S + u = \{w + u : w \in S\}$

↓

bit-wise XOR.

Claim 1:- If $|S| \leq 2^{m-n}$, then for any choice of $u_1, u_2, \ldots, u_k \in \{0,1\}^m$, we have

$$\bigcup_{i=1}^{k} (S+u_i) \neq \{0,1\}^m$$

Claim 2:- If $|S| \geq \left(1-\frac{1}{2^n}\right) \cdot 2^m$, there exists $u_1, u_2, \ldots, u_k \in \{0,1\}^m$ such that

$$\bigcup_{i=1}^{k} (S+u_i) = \{0,1\}^m$$

Proof of Claim 1:

$$\left| \bigcup_{i=1}^{k} (S+u_i) \right| \leq \sum_{i=1}^{k} |S+u_i|$$

$$= k|S|$$

$$\leq k \cdot \frac{2^m}{2^n} < 2^m \text{ since } \frac{k}{2^n} < 1 \text{ for } n \text{ large.}$$

Proof of Claim 2:

We will show that if we pick k random u_i's from $\{0,1\}^m$, then with positive probability, we will be able to cover $\{0,1\}^m$.

(Probabilistic Method)

$\Rightarrow$ The existence of $u_1, u_2, \ldots, u_k$ such that $\bigcup_{i=1}^{k} (S+u_i) = \{0,1\}^m$.

For $s \in \{0,1\}^m$, let B_s be the event that

$$s \notin \bigcup_{i=1}^{k} (S+u_i)$$

Prob [all $s \in \{0,1\}^m$ being covered] = $\bigwedge_{s \in \{0,1\}^m} \bar{B}_s$

[$\exists$ an $s \in \{0,1\}^m$ that is not covered] = $\bigcup_{s \in \{0,1\}^m} B_s$

$Pr[\bigcup B_s] < 1$

In fact we will show for an arbitrary s,

$Pr(B_s) < \frac{1}{2^m}$

This implies the above by union bound.

Let $B_s = \bigwedge_{i=1}^{k} B_s^i$, where B_s^i = i^{th} shift does not contain s.

$= s \notin S + u_i$.

$Pr[s \notin S+u_i] = Pr(S+u_i \notin S) \leq \frac{2^m}{2^n} \frac{1}{2^n}$

$S+u_i$ is a random string.

$Pr(B_s) = \prod_{i=1}^{k} Pr(B_s^i)$: since each u_i is independently chosen.

$\leq \left(\frac{1}{2^n}\right)^k = \frac{1}{2^{nk}} \rightarrow < \frac{1}{2^m}$

If $x \in L$, then $|S_x| \geq \left(1 - \frac{1}{2^n}\right) \cdot 2^m$.

By claim 2, we have $\exists u_1, u_2, \ldots, u_k$ such that $\forall r \in \{0,1\}^m$ $r \in \bigcup_{i=1}^{k} (S+u_i)$

$M(x, r+u_i)$ must accept must accept for atleast one i.

$x \in L \Leftrightarrow \exists\, u_1, u_2, \ldots, u_k \in \{0,1\}^m$
$\forall\, r \in \{0,1\}^m \; [\vee M[x, r+u_i]]$ accepts.

When $x \notin L$, $|S_x| \leq 2^m/2^n$, then by claim 1,
$\forall u_1, u_2, \ldots, u_k$, $\exists\, r \in \{0,1\}^m$ such that
$\vee M(x, r+u_i)$ rejects.

21st Oct '24
Monday

Computational Complexity Lec 17

P/poly:

We saw $P \subset P/poly$.

Reverse containment does not hold since P/poly contains undecidable languages.

$L \subseteq 1^*$

Theorem: Suppose L is a unary language. Then $L \in P/poly$.

$L = \{1, 11, 111, 1111\}$.

$$C_k = \begin{cases} \bigwedge_{i=1}^{k} x_i & \text{if } 1^k \in L \\ 0 & \text{if } 1^k \notin L. \end{cases}$$

Theorem:- P/poly contains undecidable languages.

Proof:- Consider an undecidable language, say

$HALT_{TM} = \{ \langle M, w \rangle \mid M \text{ halts on } w \}$

↓

represented as some string 100110011

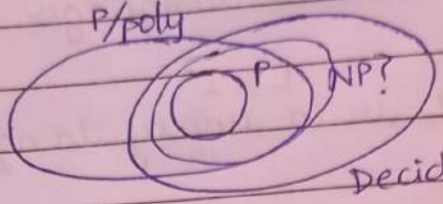
We can convert $HALT_{TM}$ into a unary version of the problem. If the binary string of a specific instance equals the integer k, then we use 1^k to denote instance.

$U_{HALT} = \{ 1^k \mid k \text{ corresponds to YES instance } \langle M, w \rangle \text{ of } HALT_{TM} \}$.

So, $U_{HALT} \in P/poly$ since it is unary.

Uniform circuit families: The circuit for input size k should be computable in P/logspace.

Ex: P-uniform P/poly = P (try to show this)



Where does NP fit in the above?
In other words, does NP have poly-sized circuits?

Karp-Lipton (80): If SAT has poly-sized circuits, then PH collapses to Σ_2.

Claim 1: If NP $\subseteq$ P/poly, then there exists a polynomial p, and $p(n)$ sized circuit family C_n such that for every Boolean formula ϕ with n variables

$C_n(\phi) = 1 \iff \exists\, v \in \{0,1\}^n$, s.t $V(\phi) = 1$.

Claim 2: If NP $\subseteq$ P/poly, then there is a polynomial q, and $q(n)$ sized circuit family $\{C'_n\}_{n \in \mathbb{N}}$ such that for every Boolean formula ϕ,

$$C'_n(\phi) = \begin{cases} v : \phi(v) = 1 \text{ if } \exists \text{ such a } v \\ 0^n : \text{otherwise.} \end{cases}$$

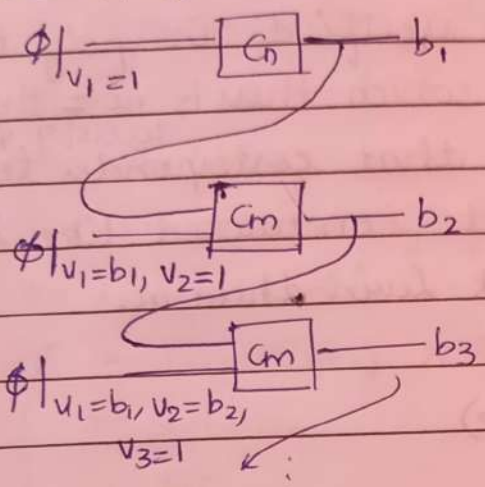
Self-Reducibility of SAT:
By Claim 1, we know C_n exists.

Given ϕ, construct $C_n(\phi)$

If $C_n(\phi) = 0$, then output 0^n.

Else, do the following:

Let $v_1, v_2, \ldots$ be the variables in ϕ



(Note that these will be a satisfying assignment for ϕ where $v_1 = b_1$)

At the end, the string $b_1, b_2, \ldots, b_n$ will be a satisfying assignment of ϕ.

Proof of Karp-Lipton Theorem:

Assuming SAT has poly-sized circuits, we will show ~~$\Sigma_2 \subseteq \Pi_2 \Rightarrow \Sigma_2 = \Pi_2$~~

$$\Pi_2 \subseteq \Sigma_2 \Rightarrow \Pi_2 = \Sigma_2.$$

$$L \in \Sigma_2 \Rightarrow \bar{L} \in \Pi_2 \Rightarrow \bar{L} \in \Sigma_2 \Rightarrow L \in \Pi_2$$

Given $L \in \Pi_2$

$x \in L \iff \forall y, \exists z$ s.t $M(x,y,z) = 1$

(y, z are poly$(|x|)$ and M is in det poly time)

Consider $L' = \{(x,y) \mid \exists z, M(x,y,z) = 1\}$

$L' \in NP$

$\Rightarrow$ there is a polynomial sized circuit family C'_m that takes as input (x,y) and

outputs z such that $M(x,y,z) = 1$.

$x \in L \iff \exists C_n \; \forall y \; M(x, y, C_n(x,y)) = 1.$

can be verified in poly time

If $x \notin L$, $\exists y$ for which there is no z that will satisfy $\phi(x,y,z)$

ϕ is the formula that corresponds to the DTM M. This can be constructed like in the proof of Cook-Levin theorem.

$$\phi_{x,y}(z) = M(x,y,z)$$

→ This is a Σ_2 characterization of L.

$\Rightarrow L \in \Sigma_2$

Theorem (Kannan): Fix $k > 0$. Then $\Sigma_4 \not\subseteq$ SIZE(n^k)

Proof:- Consider the lexicographic ordering of all the functions.

Consider the first "hard" function, that cannot be computed using SIZE(n^k) circuit.

This can be expressed as Σ_4 language.

$f \in L$ ⇔ ∃ input x, s.t.

$\exists f, \forall C_1, |C_1| < n^k, C_1 \neq f \quad C_1(x) \neq f(x)$

$\forall g, g \leq_{lex} f, \exists C_2, |C_2| \leq n^4$

$\forall$ inputs $x \quad C_2(x) = g(x)$

$L \in \Pi_3$

Can be easily improved to Σ_2 using Karp-Lipton.

5/10/2024
Monday

classmate
Date
Page

Computation Complexity Lec18

Theorem (Kannan):
Fix $k>0$. Then
Σ_4 contains languages that need size $> n^k$.
→ go through notes shared.

Using Karp Lipton, this Σ_4 can be replaced by Σ_2.

If SAT has poly sized circuits, then by Karp Lipton $\Sigma_4 = \Sigma_2$
If SAT does not have poly sized circuits, then SAT needs $> n^k$ size

Addleman's Theorem:
$BPP \subset P/poly$

Proof:- Suppose $L \in BPP$. Then $\exists$ a polytime T.M M such that

$$\Pr_r [M(x,r) = \text{correct answer}] \geq 1 - \frac{1}{2^{n+1}}$$

and $\Pr_r [M(x,r) = \text{incorrect}] \leq \frac{1}{2^{n+1}}$ where $n=|x|$

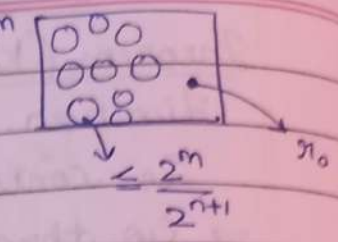
Given x, what is the no of r (suppose $|r|=m$) that leads to correct answer

$$\geq 2^m \left(1 - \frac{1}{2^{n+1}}\right)$$

For all x of length n, we say that r is bad if it gives the wrong answer for some x of length n.

No. of bad r's $\leq \frac{2^m}{2^{n+1}} \times 2^n$

$= \frac{2^m}{2}$

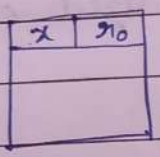


$\Rightarrow \geq$ half of the r's are good

correctly answer for all x of length n.

There exists r_0 such that

$\forall x$ of length n, $M(x, r_0)$ = correct



We can hard code r_0 in the T.M and get $M_{r_0}(\cdot)$.

By arguments similar to Cook-Levin Theorem, we can make it into a poly-sized circuit.

But, finding such an r_0 is hard. If we could find such an r in poly time, this would imply BPP = P.

$\{C_i\}$ is a P-uniform circuit family if the function $1^i \rightarrow C_i$ can be computed

(1^i: i is unary)

in poly time.

We can replace P with logspace to get definition of logspace-uniform circuit family.

Ex: P-uniform P/poly = P.
log-space-uniform P/poly = P.

T.M that take advice:

$DTIME\left(\frac{t(n)}{a(n)}\right)$.

The class of languages that is accepted by DTIME(t(n)) T.M given advice of length a(n).

$L \in DTIME\left(\frac{t(n)}{a(n)}\right)$ if there is a DTM M,

$M \in DTIME(t(n))$ and $\exists \{\alpha_n\}_{n \in \mathbb{N}}$ such that $|\alpha_n| \leq a(n)$ and

$x \in L \iff M(x, \alpha_{|x|}) = 1$.

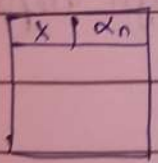
Eg:- UHALT $\in P/_1$
Needs one bit just to know whether k is in L or not

$P/poly = \bigcup_{c,d} DTIME(n^c)/n^d$

DOUBT: Why $DTIME(n^c)$?

$\subseteq$: The advice string can encode the circuit.

$\supseteq$: Given x, which is of length n, then we get a T.M which runs in poly time in x and α_n.



We can construct C_n, which is circuit representation of the TM with α_n hard-coded as input.

Classes NC and AC: Circuit classes which are more fine-grained in their definition.

Defn: $L \in NC^i$ if there is a family of circuits $\{C_n\}$ that decide L such that

(1) C_n uses only AND, OR, NOT gates

(2) AND/OR gates have fan-in 2.

(3) Depth of C_n is $O(\log n)^i$

(4) poly size

Defn: $L \in AC^i$ if there is a circuit family C_n that decides L, satisfying (1), (3) and (4) above. Unbounded fan-in.

Nick's Class $\rightarrow NC = \bigcup_{i=0} NC^i$ Alternate Class $\rightarrow AC = \bigcup_{i=0} AC^i$

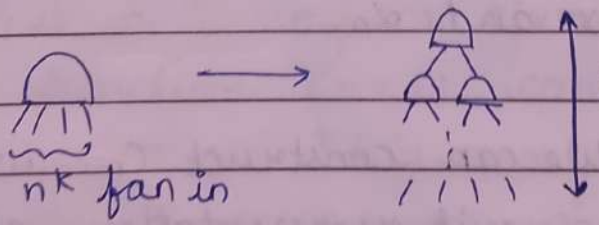
$NC^i \subseteq NC^{i+1}$
$AC^i \subseteq AC^{i+1}$
} ~~Since more depth~~

$NC^i \subseteq AC^i$ } No fan in constraint.

$AC^i \subseteq NC^{i+1}$

$NC = AC$

We can replace poly(n) fan in gate with a circuit of depth $O(\log n)$



So, when we replace unbounded fan-in gates with gates of fan in 2, we get an $O(\log n)$ blow up in the depth.

$NC^0 \subsetneq AC^0 \subsetneq NC^1 \subseteq AC^1 \subseteq NC^2 \subseteq AC^2 \subseteq \ldots$

NC^0 : No. of input bits that define the output is at most 2^d. Hence this class is not so interesting.

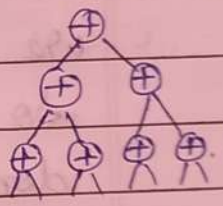
For example OR_n & ~~AC^0~~ NC^0
But $OR_n \in AC^0$

① $PARITY_n = \oplus_n \in NC^1$

$PARITY_n(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \text{if odd no of inputs is 1} \\ 0 & \text{otherwise} \end{cases}$

$\oplus$ → 2 input / PARITY gate
↳ can be realized using const. depth AND, OR, NOT gates.

$PARITY_n \in NC^1$. We can build
But $PARITY_n \notin AC^0$
(We will see later)

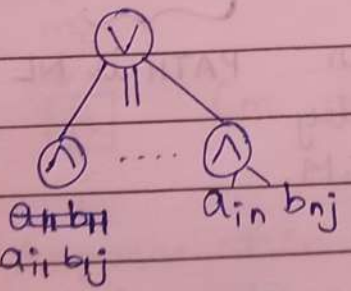


② Boolean Matrix Multiplication $\in AC^0 \subseteq NC^1$

A and B are $n \times n$ $\{0,1\}$ matrices

$$(AB)_{ij} = \bigvee_{k=1}^{n} (a_{ik} \wedge b_{kj})$$

i - [ ] [ ] j
A B



$BMM = \{(A,B,C)\}$ $A, B, C \in \{0,1\}^{m \times n}$ s.t $A \times B = C\}$

③ GRAPH REACHABILITY

PATH $= \{ \langle G, s, t \rangle \mid \exists$ in G, t is reachable from $s \}$.

Given adjacency matrix

$$A = [a_{ij}] \qquad a_{ij} = \begin{cases} 1 & i = j \\ 1 & (i,j) \in E(G) \\ 0 & \text{otherwise} \end{cases}$$

$A = A^1$ tells us if there is a path of length ≤ 1 from i to j.

A^k tells us if there is a path of length $\leq k$.

↓

Want to know $(A^n)_{s,t}$ → can do $\lceil \log n \rceil$ squaring

We know matrix mult. $\in AC^0$.

So repeated squaring $\log n$ times can be done in AC^1.

Binary addition $\in AC^0$

→ Go through this yourself.

Below holds for logspace uniform circuit

$AC^0 \subseteq NC^1 \subseteq L = NL \subseteq AC^1 \subseteq \dots \subseteq NC \subseteq P$

logspace NC^1 can be simulated by a logspace T.M.

PATH is NL complete

need logspace uniform

From now on, we will only focus on logspace uniform NC^i and AC^i. Where in addition to the above defn, we also require the circuit family to be uniformly generated.

4/11/2024
Monday

Lec 19.

Computational Complexity → Randomized NC

Bipartite Perfect Matching in RNC

→ Polynomial identity testing

Matching is a subset of edges such that no two edges share the same endpoint.

Perfect Matching: Matching which covers all vertices.

PIT: Is poly $F \equiv G$?

$(x-1)(x+3)(x-6) \equiv x^3 + 4x^2 - 12x + 18$.

Random Algo:

Choose x randomly from a set S s.t $|S| = 100d$. If $F(x) \equiv G(x)$ at the chosen x, output $F \equiv G$.

Else, output $F \not\equiv G$.

For single variable, this gives $P(\text{error}) \leq \frac{1}{100}$

We get error only when we pick x to be a root of $F-G$. So $\text{Prob}(\text{error}) \leq \frac{1}{100}$.

→ Want to check if $P(x_1, x_2, \ldots, x_n) \equiv 0$?

→ Choose $(r_1, r_2, \ldots, r_n) \in S^n$, where $|S| = 100d$.

→ Evaluate $P(r_1, r_2, \ldots, r_n)$.

- If $P(r_1, r_2, \ldots, r_n) = 0$, say $P \equiv 0$.
- Else, declare $P \not\equiv 0$.

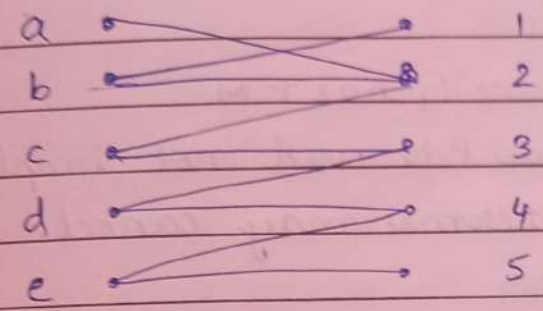
This is an RP-algorithm for

$\{P \mid P \not\equiv 0\}$.

Lemma

DeMillo - Lipton - Schwartz - Zippel :
If $P \neq 0$, then $Pr(P(x_1, x_2, \ldots, x_n) = 0) \leq \frac{d}{|S|}$

We don't know if PIT is in P.



	1	2	3	4	5
a	0	(1)	0	0	0
b	(1)	1	0	0	0
c	0	1	(1)	0	0
d	0	0	1	(1)	0
e	0	0	0	1	(1)

G has a perfect matching $\iff$ We can choose a set of 1's from the adj-matrix such that every row and col. has exactly one 1.

There has to be a permutation σ of the rows such that for all i, $A_{i,\sigma(i)} = 1$.

$\iff \sum_{\sigma} \prod_{i=1}^{n} A_{i,\sigma(i)} \neq 0$

Permanent of a matrix

Determinant:

$$\sum_{\sigma} (-1)^{\text{sign}(\sigma)} \prod_{i} A_{i,\sigma(i)}$$

Can I say that G has P.M $\Leftrightarrow$ det(A) is non-zero?

Ans:- NO.

Det $\neq 0 \Rightarrow$ G has P.M

But G has P.M need not imply det $\neq 0$.

Because terms may cancel.

Now, we will combine this with P.I.T.

$$T = \begin{bmatrix} 0 & x_{12} & 0 & 0 & 0 \\ x_{21} & x_{22} & 0 & 0 & 0 \\ 0 & x_{32} & x_{33} & 0 & 0 \\ 0 & 0 & x_{43} & x_{44} & 0 \\ 0 & 0 & 0 & x_{54} & x_{55} \end{bmatrix}$$

All variables are different.

G has a P.M $\Leftrightarrow$ det(T) $\not\equiv 0$.

↓

Not identically zero as a polynomial.

Fact:- Evaluating determinant can be done in NC^2.

We have now reduced problem of determining if the graph has a P.M. to the P.I.T problem.

We use the Schwartz - Zippel lemma and the randomized algorithm to determine if det(T) $\not\equiv 0$.

If G has P.M, we may make an error.
If G has no P.M, we will always correctly output that it does not.

Outputting a PM in parallel:

Isolation lemma: (Mulmuley, Vazirani)
Let $\mathcal{F}$ be a family of subsets of an n-element set X.
Let $w : X \to \{1, 2, \ldots, N\}$ independently and uniformly at random.
Then $\Pr\left[\text{There is a unique min. weight set } F \in \mathcal{F}\right] \geq 1 - \frac{n}{N}$

$X = \{x_1, x_2, x_3, x_4, x_5\}$

$\mathcal{F} = \{\{x_1, x_2\}, \{x_3\}, \{x_3, x_2\}, \{x_3, x_4, x_5\}\}$

(weights: 3, 3, 5, 12)

Proof:- For $x \in X$, define $\alpha(x) = \min_{\substack{F \in \mathcal{F} \\ x \notin F}} w(F)$

$$\alpha(x) = \min_{\substack{F \in \mathcal{F} \\ x \notin F}} w(F) - \min_{\substack{F \in \mathcal{F} \\ x \in F}} w(F \setminus \{x\})$$

$\alpha(x)$ is independent of $w(x)$.
In other words, $\alpha(x)$ only depends on weights of other elements.

$$\Pr[\alpha(x) = w(x)] \leq \frac{1}{N}$$

$$\Pr[\exists x \text{ s.t. } \alpha(x) = w(x)] \leq \frac{n}{N}$$

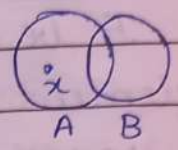
If there are two or more min weight sets, then there is x s.t. $\alpha(x) = w(x)$

This implies that

$$P\left(\begin{array}{c}\exists \text{ exists unique} \\ \text{min. weight set in } \mathcal{F}\end{array}\right) \geq 1 - \frac{n}{N}$$

Suppose sets A and B are distinct min. weight sets. Then pick x in A that is not in B.

What is $\alpha(x)$?



$$\min_{F \in \mathcal{F}} w(F) = w(B)$$

$x \notin F$

$$\min_{F \in \mathcal{F}} w(F \setminus \{x\}) = w(A) - w(x)$$

$x \in F$

$$\alpha(x) = w(B) - [w(A) - w(x)] = \underline{\underline{w(x)}}$$

1. Choose random values $w_e \in \{1, 2, \ldots, 2|E|\}$ for each $e \in E$

$$\rightarrow B = \begin{bmatrix} 0 & 2^{w_{12}} & 0 & 0 & 0 \\ 2^{w_{21}} & 2^{w_{22}} & 0 & 0 & 0 \\ 0 & 2^{w_{32}} & 2^{w_{33}} & 0 & 0 \\ 0 & 0 & 2^{w_{43}} & 2^{w_{44}} & 0 \\ 0 & 0 & 0 & 2^{w_{54}} & 2^{w_{55}} \end{bmatrix}$$

If σ is a matching then $\Pi_{A_{i,\sigma(i)}} = 2^{\sum w(e)}$

where e is in the matching.

2. Compute det(B).

   If det(B) = 0, then o/p no matching.

3. If det B ≠ 0. Compute R which is largest power of 2 that divides det(B).

4. In parallel, do
(a) For each edge, let B_e be the matrix obtained by setting entry of e to zero.
(b) Compute $\det(B_e)$. If $\det(B_e) = 0$, output e.
(c) Else compute largest power of 2 that divides $\det(B_e)$. Call it R_e.
(d) If $R_e > R$, output e.

7/11/2024
Thursday

Lec 20
Computational Complexity

~~m = |G|~~

Given a graph and matrix B,

(1) Find det(B). Let R be the largest power of 2 that divides B.

(2) Do in parallel for each edge e.

→ Replace 2^{w_e} with 0 to get matrix B_e.

→ Compute $\det(B_e)$.

→ If $\det(B_e) = 0$, output e.

→ Else compute R_e, the largest power of 2 that divides $\det(B_e)$.

→ Output e if $R_e > R$.

$$\det(B) = \sum_{M;\, M \text{ is a p.m of } G} (\pm)\, 2^{w(M)}$$

$$\pm 2^{w} \pm (2^{w_1} \pm 2^{w_2} \pm 2^{w_3} \pm \dots)$$

$$2^{w}(\pm 1 \pm 2(\dots))$$

So R will be the weight of the unique min weight p.m.

→ If there is only one p.m in G, then $\det(B_e) = 0$ when e is part of the matching

→ If there are multiple p.m, then remaining e necessarily increases the highest power of 2 that divides $\det(B_e)$. So $R_e > R$.

Theorem: PARITY $\notin AC^0$

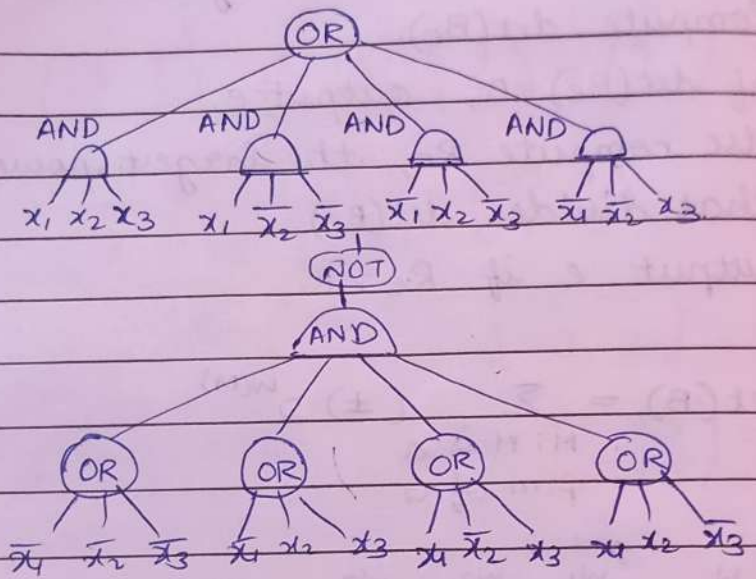
$NC^0 \subsetneq AC^0 \subsetneq NC^1$

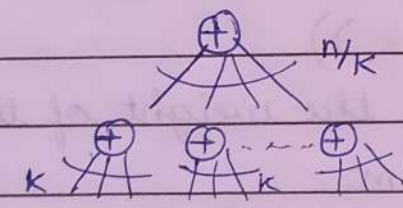
[Razborov '87] [Smolensky '87]

Any constant depth circuit that computes PARITY must have size $2^{\Omega(n^{1/d-1})}$

The proof we see has size $2^{\Omega(n^{1/d})}$ (stronger result)

PARITY$(x_1, x_2, \ldots, x_n)$ = 1, if odd no of inputs = 1
0, otherwise





The size will be minimized when $\min(k, n/k)$

Size is minimized when $k = \sqrt{n}$.

This gives us circuit of size $O(2^{\sqrt{n}})$

Ex: $2^{n^{1/d}}$.

Idea :- To represent the function using polynomials over $\mathbb{F}_3$.

→ Every AC^0 function has a "low" degree "approximation" over $\mathbb{F}_3$.

→ PARITY needs a large degree.

$\mathbb{F}_3 = \{0,1,2\}$ with addition, multiplication modulo 3.

$AND_n \rightarrow x_1, x_2, x_3 \ldots x_n$

When $x_1, x_2, \ldots, x_n \in \{0,1\}^n$, the above poly gives the same output as AND.

$x_1^2 x_3^3 x_2^6 \quad x_1^2 x_3^3 x_4^6 \rightarrow x_1 x_3 x_4$

When a boolean function is represented as a polynomial over $\mathbb{F}_3$, then each term can be "multilinear".

Mod p cannot be implemented in AC^0 circuit that uses AND, OR, NOT and mod q gates. (where p, q are distinct primes).

Def:- A random polynomial $p(x)$ chosen from dist. D, ε-approximates $f(x)$ (where $f(x)$ is a Boolean $f(n)$). if for each $x \in \{0,1\}^n$,

$\Pr_{p \sim D}[p(x) \neq f(x)] \leq \varepsilon$

$OR_n \rightarrow 1 - (1-x_1)(1-x_2)\cdots(1-x_n)$

Theorem: If $f(x)$ is computed by a circuit of size s and depth d, then there is a dist. D, from which you can choose poly $p(x) \in \mathbb{F}_3[x_1, x_2, \ldots, x_n]$ such that degree $(p(x)) \leq O(\log^d s)$ and $p(x)$, $1/4$-approximates $f(x)$.

If AC^0, $O(\log^d n)$

Consider OR.

$OR(x_1, x_2, \ldots, x_n) = 1$ This fails when $x = 0$?

$\alpha(x, r)$ = Dot product of x, r

$= x_1 r_1 + x_2 r_2 + \cdots + x_n r_n$ where r is a random vector from $\mathbb{F}_3^n$.

When $x = 0$, $\alpha(x, r) = 0$

When $x \neq 0$, what is $\Pr(\alpha(x, r) \neq 0)$?

~~When~~ $\alpha(x, r) = \sum_{i=1}^{n} x_i r_i$ ~~+ $x_j r_j$~~ $x_j r_j$

$\Pr(\alpha(x, r) \neq 0) \geq 2/3$

There is only one value of ~~r_j~~ r_j (upon fixing other r_i's) that results in $\alpha(x, r) = 0$.

Hence $p(\alpha(x, r) \neq 0) \geq 2/3$

If we take $\alpha^2(x, r)$, it will be 1 w.p $\geq 2/3$.

$(1 - (1 - \alpha(x, r_1)))(1 - (1 - \alpha(x, r_2)))$

11/11/2024
Monday

Lec 21

Computational Complexity

Theorem:- If $f(x)$ is computed by a circuit of size s and depth d, there is a distribution D from which you can choose poly $p(x)$ such that $\deg(p(x)) \leq O(\log^d s)$ and $p(x)$ $1/4$ - approximates $f(x)$.

$$\alpha(x, r) = \alpha_1 r_1 + \alpha_2 r_2 + \cdots + \alpha_n r_n$$

$$P(\alpha^2(x, r) = OR(x_1, x_2, \ldots, x_n))$$

Boosting:- $B(x, r) = 1 - (1 - \alpha^2(x, r^{(1)}))(1 - \alpha^2(x, r^{(2)})) \cdots (1 - \alpha^2(x, r^{(k)}))$

$$P(B(x, r) \neq OR_n(x)) \leq \frac{1}{3^k}$$

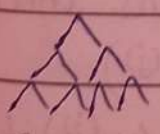
$$\text{Deg}(B()) = 2k$$

If we need error $\leq \varepsilon$, we need to set $k = O(\log 1/\varepsilon)$

$OR \rightarrow \varepsilon$-approximated with degree $O(\log 1/\varepsilon)$

$NOT(x) = 1 - x$

$AND_n(x) \rightarrow$ We can negate input variables, use OR and then negate output.



size s
depth d

We can approximate each gate with the corresponding poly.

→ Suppose each gate has an ε error.

Pr (error of the entire poly) $\leq 3\varepsilon$ (Union bound)

$g_1(y_1, y_2) \equiv g_1(f(x_1), f(x_2))$
$\equiv g(f(x_1, x_2), f(x_3, x_4))$

deg of this poly $= \deg(g_1) * (\max\{\deg(f_1), \deg(f_2)\})$

$= d^2$ (assuming deg g, f_1, f_2 are all d)

For depth d circuit, degree $\leq O(\log 1/\varepsilon)^d$
Want $1/4$ - approximation i.e. $s \cdot \varepsilon \leq 1/4$.
$\varepsilon \leq 1/4s$.
So degree $\leq O(\log 4s)^d$

Theorem:- Suppose $f: \{1,0\}^n \to \{0,1\}$ is computed by circuit of size s and depth d, then there is a fixed poly $q(x)$ over $\mathbb{F}_3$ such that $\deg(q(x)) \leq O(\log s)^d$ and $\Pr_{x \in \{0,1\}^n}[f(x) = q(x)] \geq 3/4$ ($f(x) = q(x)$ on $\geq 3/4$ of the inputs x in $\{0,1\}^n$).

	P_1	P_2	P_3	...	P_j	...
$x^{(1)}$	1	1	0	...	1	...
$x^{(2)}$	1	0	1	...	0	...
$\vdots$						
$x^{(j)}$	0	0	1	...	1	...
$x^{(2^n)}$						

1 entry means that $P_i(x^{(j)}) = f(x^{(j)})$
0 means $P_i(x^{(j)}) \neq f(x^{(j)})$

First theorem states that atleast $3/4$ of the entries of a row are 1.

By looking at it column-wise, there must exist atleast one col. with $\geq 3/4$ entries = 1

⇓

There is a polynomial p_i that correctly computes $f(x)$ on $\geq 3/4$ of the inputs.

Theorem 3: For any $q(x)$ such that $q(x) = \text{PARITY}(x)$ on $\geq 3/4$ of inputs, we must have degree $(q(x)) \geq \frac{\sqrt{n}}{10}$

To note that PARITY $\notin AC^0$, we combine Theorems 2 and 3.

If PARITY $\in AC^0$, theorem 2 implies $q(x)$ with $O(\log n)^d$ degree, but theorem 3 says deg $(q(x)) \geq \frac{\sqrt{n}}{10}$.

If there is a circuit of size s and depth d, we get $O(\log s)^d \geq \frac{\sqrt{n}}{10}$ $\log s \geq \left(\frac{\sqrt{n}}{10}\right)^{1/d}$

$$s \geq 2^{\Omega(n^{1/2d})}$$

We have to prove theorem 3:

Suppose there is $q(x)$ with degree $= \frac{\sqrt{n}}{10}$ such that $q(x) = \text{PARITY}(x)$ on $\geq 3/4$ of inputs.

Let $A \subseteq \{0,1\}^n$ of inputs where $q(x) = \text{PARITY}(x)$.

Then we will show that we get a "low degree" representation of any polynomial $g(x)$ over A. We will

derive a contradiction between no. of such low degree representations and no. of g's

Claim:- Change of basis:

$$q'(x) = 1 - 2q\left(\frac{1-x_1}{2}, \frac{1-x_2}{2}, \ldots, \frac{1-x_n}{2}\right)$$

q computes PARITY of $\{0,1\}^n$ $\iff$ q' computes PARITY of $\{-1,+1\}^n$.

$0 \longrightarrow +1$

$1 \longrightarrow -1$.

Degree(q') $\leq$ Degree (q)

$$A = \{x \in \{\pm 1\}^n \mid q'(x) = \prod_{i=1}^{n} x_i \cdot y\}$$

We know $|A| \geq 3/4 \cdot 2^n$

Claim:- Consider any $g(x) \in \mathbb{F}_3(x)$. Then there is a poly(b(x) s.t.

(1) $\forall x \in A$, $h(x) = g(x)$.

(2) $\deg(h(x)) = \deg(q(x)) + \frac{n}{2} < \frac{\sqrt{n}}{10} + \frac{n}{2}$.

Proof:- We will modify g to obtain the degree bound.

Look at each monomial of g(x).

→ If there is a term with x_i^2, we replace it with 1. ($x_1^2 x_2 x_3 \rightarrow x_2 x_3$)

(We are only concerned with inputs from $\{-1,1\}^n$) ($x_1^3 x_2^4 x_3^7 \rightarrow x_1 x_2 x_3$)

So each monomial is multilinear.

$\prod_{i \in S} x_i$ → We have terms like this where $S \subseteq \{1, 2, \ldots, n\}$

→ If $|S| \leq n/2$, we retain the term without changes.

→ If $|S| > \frac{n}{2}$, $\prod_{i \in S} x_i = \left(\prod_{i \notin S} x_i\right)\left(\prod_{i=1}^{n} x_i\right)$

$\forall$ $\prod_{i \subseteq S} x_i = \left(\prod_{i \notin S} x_i\right) \cdot q'(x)$

since $q'(x) = \prod_i x_i$ when $x \in A$

Degree of this $\leq \frac{n}{2} + \deg(q'(x))$

$= \frac{n}{2} + \frac{\sqrt{n}}{10}$

We will see that no. of functions $g: A \to \mathbb{F}_3$ is much larger than the no. of ~~polynomials~~ multilinear polynomials of degree $\leq \frac{n}{2} + \frac{\sqrt{n}}{10}$

No. of functions $g: A \to \mathbb{F}_3$ $= 3^{|A|} \geq 3^{3/4 \cdot 2^n}$

No. of multilinear poly. of degree $\leq \frac{n}{2} + \frac{\sqrt{n}}{10}$

No. of monomials

$$\underbrace{\binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{n/2}}_{1/2 \cdot 2^n} + \underbrace{\binom{n}{n/2+1} + \cdots + \binom{n}{n/2+\frac{\sqrt{n}}{10}}}_{\sum_{i=1}^{\sqrt{n}/10} \binom{n}{\frac{n}{2}+i} \leq \binom{n}{n/2} \times \frac{\sqrt{n}}{10}}$$

$$\leq \frac{1}{\sqrt{\frac{n}{2}}} \cdot \frac{2^n}{10} \cdot \sqrt{\frac{\pi}{2}}$$

Sterling approx.

No. of such $g \leq 3^{0.6 \cdot 2^n}$

This proof also shows that PARITY $\notin AC^0$[3]

Output ~~0~~ with ← AC^0 with mod 3 gates

Output 1 iff inputs sum to 0 mod 3.

$$\sum_{i=1}^{\sqrt{n}/10} \binom{n}{n/2+i} \times \frac{\sqrt{n}}{10} \leq \binom{n}{n/2} \times \frac{\sqrt{n}}{10}$$

$$\approx \frac{2^n}{\sqrt{\pi n/2}} \times \frac{\sqrt{n}}{10} = \frac{1}{10\sqrt{\pi/2}} \times 2^n$$

Sterling's approx: $n! \sim \frac{n^n \sqrt{2\pi n}}{e^n}$

$$\binom{n}{n/2} = \frac{\frac{n^n}{e^n}\sqrt{2\pi n}}{\left(\frac{(n/2)^{n/2}}{e^{n/2}}\sqrt{2\pi n/2}\right)^2} = \frac{2^n}{\sqrt{\pi n/2}}$$

So no. of monomials $\leq \frac{1}{2} \cdot 2^n + \frac{1}{10\sqrt{\pi/2}} \cdot 2^n$

$$\leq 0.6\, 2^n$$

14/11/2024 Thursday

lec 22

Computational Complexity

18th Nov: No lecture

25th Nov: Final Exam

~~21st~~ 21st Nov: Final lecture.

#P: Counting Complexity

IP: Interactive Proofs

A way to generalize NP.

Verifier model of NP: We get a "proof" and need to verify.

What if ~~you~~ we can hold a conversation with the "prover"?

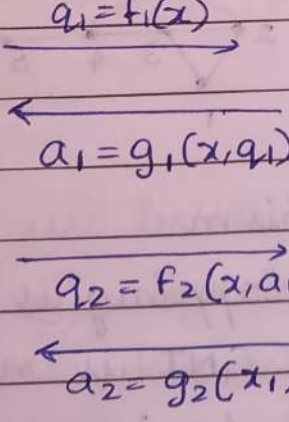
What can we accomplish under this setting?

→ All powerful prover

→ Prover may not be ~~trustable~~ trustworthy.

But the proof system/protocol needs to safeguard against this.

→ Verifier is limited to poly. time.



V accepts if the entire transcript can be used to verify that $x \in L$.

dIP: det IP

→ Verifier is restricted to det poly. time.

dIP: L has a 2k round det interactive proof system if there is a deterministic T.M. V on input $x, a_1, a_2, \ldots, a_k$, runs in time poly($|x|$) such that

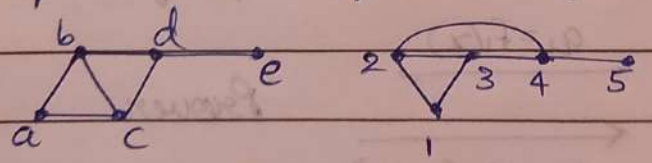
→ Completeness $x \in L \Rightarrow \exists$ a P s.t V accepts.

→ Soundness $x \notin L \Rightarrow \forall P$ V rejects x.

dIP = NP: The prover can provide the entire transcript of the ϕ x A that will happen with the verifier. Verifier can verify that the transcript is correct, and if it leads to accept.

Random Verifier:

Graph Non Isomorphism (GNI)



GI: Graph Isomorphism

GI $\in$ NP: Mapping is the certificate

GNI $\in$ co-NP: GNI is not known to be in NP.

GNI = $\{\langle G_1, G_2 \rangle \mid G_1$ is not isomorphic to $G_2\}$

1. Verifier picks $b \in \{0,1\}$ (or $\{1,2\}$) at random. Picks a permutation $\sigma : [n] \to [n]$ at random.

2. Permute the vertices of G_b as per σ, say we get H. Send H to prover.
3. Prover has to determine if H came from G_1 or G_2.
4. V accepts if prover correctly answers.

If $\langle G_1, G_2 \rangle \in GNI$, $Pr(\text{Verifier accepts}) = 1$.
If $\langle G_1, G_2 \rangle \notin GNI$, $Pr(\text{Verifier accepts}) = \frac{1}{2}$

Def:- $L \in IP[k]$, if there is a prob polytime verifier V, that has a k round interaction with a prover P such that

Completeness: $x \in L \Rightarrow \exists P, \underset{r}{Pr}[V^P(x) = acc] \geq 2/3$

Soundness: $x \notin L \Rightarrow \forall P, \underset{r}{Pr}[V^P(x) = acc] \leq 1/3$

Def:- $IP = IP[poly]$
We saw $GNI \in IP[4]$

→ $NP \subseteq IP$
→ $BPP \subseteq IP$
→ We can use better probabilities instead of $(2/3, 1/3)$, by boosting.
→ $IP \subseteq PSPACE$. We can replace the prover by a PSPACE machine. The entire transcript can be verified in poly space.
→ The other direction containment also holds, but it is non-trivial.

→ Using ideas from $BPP \subseteq \Sigma_2$, we can get perfect completeness. But we don't know of a way to get perfect soundness.

→ We can replace private coins with public coins, but with increasing no. of rounds.

↳ AM is the public coin Interactive Proofs complexity class.

#SAT $\in$ IP

(First Lec on Circuits)

Lec 7 Lec 23

12/11/2020 Tuesday

classmate Date Page

Computational Complexity

→ A circuit is a directed acyclic graph that has n inputs and a single output.

→ The non-input vertices are gates chosen from a basis (usually, AND, OR, NOT) ∧ ∨ ¬

→ A circuit computes a boolean function $f: \{0,1\}^n \to \{0,1\}$.

$L_i = \{x \mid |x| = i, x \in L\}$

→ To decide L, we will build a family of circuits C_i such that set of all strings are accepted by $C_i = L_i$

→ T.Ms — They have one T.M for all i/p lengths

→ Circuits — Varying i/p lengths have varying circuits under one family.

Resources:

→ SIZE, DEPTH, FAN-IN, NO. OF WIRES

→ We say $A \in SIZE(S(n))$ if there is a circuit C_i of size $S(n)$ that decides A.

$C_0, C_1, C_2, \ldots, C_n$

— Boolean Functions / Circuits

Symmetric Function:- Function value is unchanged by permuting of inputs.

$x_1 \wedge \bar{x}_2$ — not symmetric

$\bar{x}_1 \wedge x_2 \wedge x_3$ — not symmetric.

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

Monotone Functions:

$0000 \leq 1000$ Bitwise Comparison.

$001 \leq 100$

Monotone Circuits :- Circuits that use only AND and OR gates (no NOT gates)

Theorem :- f is monotone function $\Leftrightarrow$ f has a monotone circuit representation.

PROOF?

No. of functions from $\{0,1\}^n \to \{0,1\} = 2^{2^n}$

Symmetric functions: 2^{n+1}

Theorem :- For any language L, we have $L \in SIZE(O(2^n))$

Proof :- Let f be the function that corresponds to L_n.

$f(x_1, x_2, \ldots, x_n) = [x_1 \wedge f(1, x_2, \ldots, x_n)] \vee [\bar{x}_1 \wedge f(0, x_2, \ldots, x_n)]$

$S(n) = 2S(n-1) + 4$

$2S(n-1) = 2^2 S(n-2) + 4 \times 2 \quad S(1) = 1$

$2^{n-2} S(2) = 2^{n-1} S(1) + 4 \times 2^{n-2}$

$S(n) = 2^{n-1} + 4(1 + 2 + 2^2 + \ldots + 2^{n-2})$

$2^{n-1} + 4 \cdot 2^{n-1} - 4$

$S(n) = \frac{5 \cdot 2^{n-1}}{2} - 4$

Better bound: $L \in SIZE(o(2^n/n))$ for any L
$\Rightarrow$ LUPANOV's bound.

Theorem: (Shannon):
~~The~~ There are languages L such that
$L \notin SIZE\left(\frac{2^n}{10n}\right)$
By counting argument.
How many circuits can be made?
For each gate, we need — type of gate
inputs to the gate

Fix the circuit size to be s.
$n+s$ possibilities for each left and right input
We need $2\log(n+s)$ bits.
type $\leftarrow 2 + 2\log(n+s)$
Assuming $n \leq s$, $\leq 3\log s$.
If we have s gates, we need $3s\log s$ bits.
No. of circuits of size s $\Big\} \leq 2^{3s\log s}$.
No. of boolean functions $= 2^{2^n}$
If we set $s = \frac{2^n}{10n}$

$$3s\log s = \frac{3\cdot 2^n}{10n}\log\frac{2^n}{10n} = \frac{3\cdot 2^n}{10n}(n - \log 10n)$$

$$\leq \frac{2^n \cdot 3}{10n} < \frac{2^n}{3}$$

$$2^{3s\log s} < 2^{2^n/3} < 2^{2^n}$$

→ Since $2^{2^n/3} << 2^{2^n}$, any function on n bits is likely to require $\frac{2^n \cdot 3}{10n}$ gates

→ We donot know of an explicit function that requires even 6n gates.

Theorem:- For any f, g such that $n \leq 25nf(n) \leq g(n) \leq \frac{5}{2} \cdot 2^n$, we have $\text{SIZE}(f(n)) \subsetneq \text{SIZE}(g(n))$

Proof:- Not immediately by diagonalization.

We know that:

→ There are functions $h: \{0,1\}^l \to \{0,1\}$ that cannot be computed in size $\left(\frac{2^l}{10l}\right)$

→ Every function $h: \{0,1\}^l \to \{0,1\}$ can be computed in $\text{SIZE}\left(\frac{5}{2} \cdot 2^l\right)$

We choose l such that $f(n) \approx 2^l/10l$.

$$g(n) > 25nf(n)$$
$$\geq 25l \, 2^l/10l = \frac{5}{2} 2^l$$

So g can capture all functions

Choose a h & $\text{SIZE}(f(n)) \approx \text{SIZE}\left(\frac{2^l}{l}\right)$

Now lift $h: \{0,1\}^l \to \{0,1\}$ to $h': \{0,1\}^l \to \{0,1\}$

by $h'(x) = h(\text{first } l \text{ bits of } x)$