

①

Defined,  
 $\text{PolyL} := \bigcup_{c>0} \text{SPACE}(\log^c n)$

$\text{SC} := \text{polytime and poly log space}$

Is  $\text{PATH} \in \text{SC}$ ?

Part-I: Savitch's theorem gives a constructive argument to show  $\text{NSPACE}(scn) \subseteq \text{DSPACE}(s^2(n))$ .

But the algorithm suggested in the theorem has time complexity  $2^{\Omega(s^2(n))}$ .

③

It is known,  $\text{PATH} \in \text{NL} = \text{NSPACE}(\log n)$   
 $\Rightarrow \text{PATH} \in \text{DSPACE}(\log^2 n)$  [Savitch]

As per Savitch, Runtime is  $2^{\Omega(\log^2 n)}$

This is Quasi-polynomial.

Hence, not sufficient to show  $\text{PATH} \in \text{SC}$ .

Part-II: No,  $\text{SC} \subseteq \text{PolyL} \cap \text{P}$  [By definition]

But  $\text{PolyL} \cap \text{P}$  contains those languages also that is decided by two separate Turing machine  $\text{TM}_1$  and  $\text{TM}_2$ , where  $\text{TM}_1$  is polytime and  $\text{TM}_2$  is poly log space machine.

$\text{SC}$  requires a single TM having both feature.

polytime

(2)

Certificate definition of NL.

$L \in NL$  if there is a log-space verifier 'M'

& a poly-function 'q' s.t.

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{q(|x|)} \text{ s.t. } M(x, u) = 1$$

Constrain: where 'u' is on read-only tape.

claim: if 'u' is on regular tape where back and forth motion for head is allowed. Then class defined is equal to NP.

Strategy: • We show 3-SAT is a member of class defined in the above way.

This proves  $NP \subseteq \text{new-NL}$  {  
① → Every problem in NP is log-space reducible  
② → to 3-SAT.

You also need

to show  $\text{new-NL} \subseteq NP$ .

Part ①: Let,  $x \in 3SAT$

example:  $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_5 \vee x_6) \wedge (\dots)$

assume certificate: 10101...

Verification by log-space TM

→ It starts with first clause, look at the first literal

You also need to store its location on log sized pointer. Go to certificate tape. Read the value of corresponding literal on certificate tape.

index.

→ Come back to work tape and note down its value. Do the same for other literals in the current clause. Finally evaluate the value of the clause.

→ Repeat the process for next clause. Evaluate its value using log-space like before.

Keep adding the output of current clause to the previous. In this way, we don't need to store all outputs of clauses.

5.5

→ This shows 3-SAT is in the newly defined class.

Part (II): 3-SAT is NP-Complete under log-space reduction.

$$L \in NL \Rightarrow L \leq_L 3\text{-SAT}$$

→ We recall in case of polytime reduction, local nature of computation was utilized. Previous three cells were used to compute the current cell.

This operation was replaced by a CNF (circuit).

→ The main challenge here is to come-up with log-space computable function  $f : x \rightarrow \phi_x$  s.t.

$$x \in L \Leftrightarrow \phi_x \in 3\text{-SAT}.$$

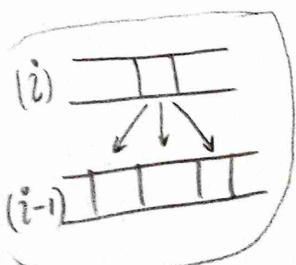
More precisely,

$$\exists u \in \{0,1\}^{p(|x|)}$$

$$\text{s.t. } M(x, u) = 1 \Leftrightarrow \phi_x \text{ is satisfiable}$$

→ Given 'x' will be hardwired in the constructed circuit. 'u' will become input to circuit.

→ The function 'f' is log-space computable as it needs to keep track of a finite number of pointers to do the local computation. Each pointer takes log-space.



③

Given,

$$\text{set } S = \{S_i\}_{i=1}^m \subset U$$

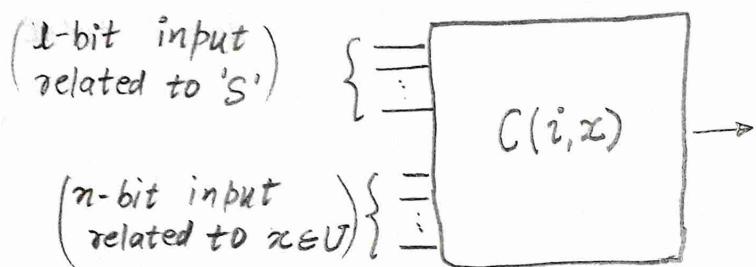
$\text{VC}(S)$  is an integer equal to cardinality of set  $X \subseteq U$

s.t. for every subset  $X' \subseteq X$  there is 'i'

$$\text{s.t. } S_i \cap X = X'$$

⑤

Given Boolean Circuit C



$C(i, x) = 1$  only for set of elements  $x \in U$  that is member of set  $S_i$ .

So, we are given access to this circuit and integer 'k'.

Language VC-DIMENSIONS =

$$\{ \langle C, k \rangle : C \text{ represents a collection } S \text{ such that } \text{VC}(S) \geq k \}$$

Proof strategy : we will analyze an explicit polytime computable predicate with alternating quantifier of form:  $\exists \cdot \forall \cdot \exists \cdot \forall \cdot \dots$  We show the logical statement evaluates to 1 iff the input belongs to the language.

Logical statement:

$$VC(S) > K \Leftrightarrow (\exists x_1, \dots, x_K \in \{0,1\}^n)$$

①

$$(\forall s \in \{0,1\}^K)$$

②

$$(\exists i \in \{0,1\}^2)$$

$$(\forall j \in \{1, \dots, K\})$$

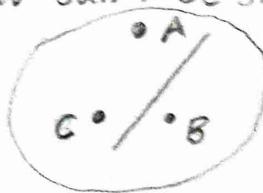
$$\underline{\textcircled{3} \leftarrow M[C(i, x) = s[j]] = 1}$$

Explanation:

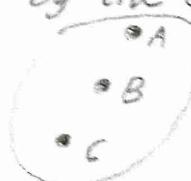
part ① demands if there exists a set  $\{x_1, \dots, x_K\} \subseteq U$

It is not required that all set of size 'k' shall fulfill upcoming requirements. [shattering by S]

For example VC-DIM of Binary classifier in plane is 3 even though there exist a set that can't be shattered by the classifier.



It is possible to classify  $\{A, C\}$  and  $\{B\}$  by line



No line can classify  $\{A, C\}$  and  $\{B\}$

part ②: Once we have the set  $\{x_1, \dots, x_K\}$ , we check shattering condition over all its subsets  $x'$ .

It has  $2^K$  subsets. We index them as

$$\{x_1\} = 1000$$

$$\{x_1, x_2\} = 1100 \dots \text{and so on}$$

Hence,  $\{0,1\}^K$  contain all the strings indexing the subsets.

part (c): Now,

for each subset  $x'$  we search if there exists  $S_i$  s.t  $S_i \cap X = x'$

$\exists i \in \{0,1\}^k$  capture this requirement.

part (d)

as mentioned in part (b),

a given string  $ss \in \{0,1\}^k$  that index subset of  $X$ . we check via turing Machine if  $j$ th bit of  $ss$  matches with Circuit output  $C(i, x_j)$ .

Example:

Suppose,  $ss = 100\dots0 = \{x_1\} \subseteq X$

assume  $S_2 \cap X = \{x_1\}$

then,

$$C(2, x_1) = ss[1] = 1$$

$$C(2, x_2) = ss[2] = 0 \text{ and so on.}$$

Any mismatch implies the circuit doesn't represent collection 'S'.

→ Runtime of this check is polynomial in input size.

→ Using the fact that  $VC(S)$  has an upper bound. This is logarithm of (number of different classifier it could return).

Say in a line why this  $VC(S) \leq \log |S| = l$

is true. → Hence, TM need to check over those ' $k$ ' where  $k < l$ .

→ ALSO,  $n, k, l$  used in the logical statement are also bounded.

Finally we have three alternating quantifiers given by  $\textcircled{a}$ ,  $\textcircled{b}$  and  $\textcircled{c}$ . There is a Turing Machine that runs in poly time and carry out this "Verification-like" process.

This shows  $\text{VC-DIM} \in \sum_3$  in PH.

④

Definition of  $\text{NC}' \Rightarrow$  poly-size and log-depth circuit family  $\{C_n\}$ ;  $C_n$  is Boolean circuit [Definition 6.24, Arora, Barak]

6.5

In question to show

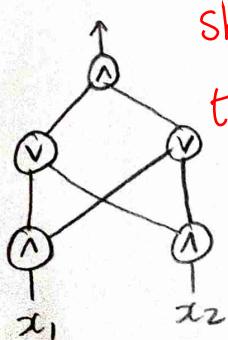
$L \in \text{NC}' \Leftrightarrow L$  is decided by a poly-size circuit family  $\{C_n\}$ ;  $C_n$  is Boolean function.

→ First we show a poly-size and log-depth circuit can be converted to a poly-size Boolean formula.

since, formula is a special circuit with fan-out of one.

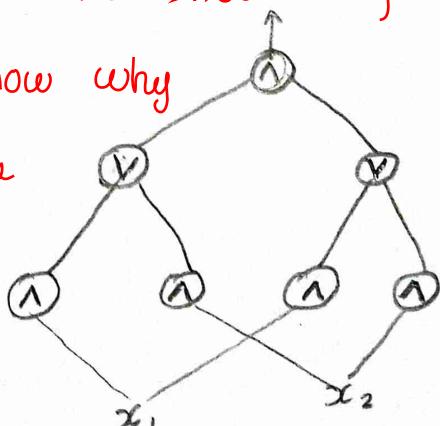
→ Method to change a circuit to formula:  
"Unwinding" the circuit graph into a tree:  
check how many paths exists between a certain gate 'g' (say) to the output gate. Now duplicate that many gate 'g'.

Example: Argue why this works. Instead of



circuit

Showing an eg. show why  
this works in the  
general case.



Formula: Tree

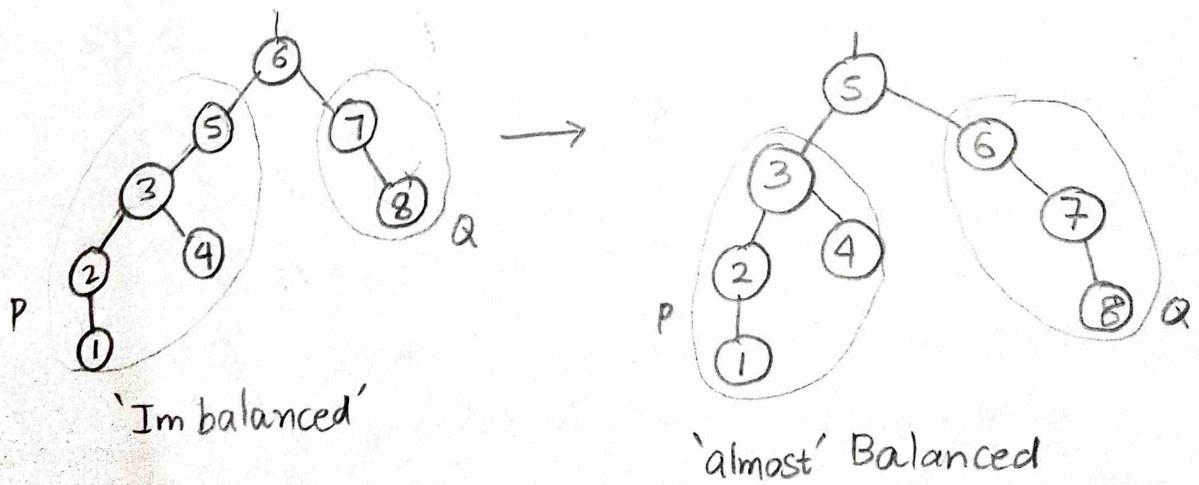
Note: The depth remain conserved in this process which implies the blow up in size is atmost by a polynomial factor. Justify this in a line.

- Now, it remains to show if any poly-size formula without any depth restriction can be equivalently converted to a poly-size and log-depth formula.
- This necessarily implies the class of language computed by poly-size formula remains same with and without log-depth restriction.
- Indeed there is a technique to convert a poly-size formula (without log-depth) to a poly-size formula with log-depth. [Claim-I]

→ Proof of Claim-I by formula "Re-Balancing" method.

(Re-Balance step-I): first a vertex that splits the computation tree into "almost" equal part.  
[meaning of 'almost' discussed later]

Ex:



Meaning of word 'almost': If the size of formula is ' $s$ ' then we choose that vertex which splits the tree into two subtree ' $P$ ' and ' $Q$ ' such that  $|P|, |Q|$  lies  $[ \frac{s}{3}, \frac{2s}{3} ]$

→ The fact is there always exists such a vertex if we follow the below algorithm.

(1.) Take any vertex and check the sizes of its subtrees.

(2.) If they lie between  $\left[\frac{S}{3}, \frac{2S}{3}\right]$ ; then exit

(3.) Otherwise go into the subtree having size  $> \frac{2S}{3}$ . Repeat the step ① and ② until success.

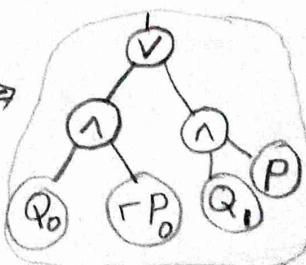
The algorithm must converge/stop, as we are decreasing the size of larger subtrees in each iterations.

A more detailed proof of correctness is required.

'Re-Balance' step-II): Step-I gives two almost balanced subtree 'P' and 'Q'. Now the crucial idea is to separately run computation of 'P' and 'Q' sub-parts to simulate the entire compilation of original tree. Without loss of generality assume 'Q' depends on 'P'. (But not vice-versa). There could be two possible outcome of 'P'. Let, if  $P=0$ ,  $Q$  evaluates to  $Q_0$ . Other  $Q$  evaluates to  $Q_1$ . Now, we build a selector formula that selects the appropriate copy of  $Q$  based on the computation of  $P$ .

Or,

$$f_{\text{selector}} = (Q_0 \wedge \neg P) \vee (Q_1 \wedge P)$$



It creates extra copy of 'Q', but also ensures depth reduction. This is fine, if blow in size is polynomial.

'Re-Balance' step-III: Now, we recursively re-balance P and Q<sub>i</sub>. Then we plug the result to the tree which further increase depth of the tree.

Analyse: We calculate maximum number of such recursive call. Since on each call the size of subparts to be considered next decreases by  $\frac{2}{3}$  (atmost). Hence atmost  $\log_{\frac{3}{2}}(s)$  recursive call required.

We notice that while building selector formula, we add a constant number of depth to the tree/formula. Since this goes for  $\Theta(\log(s))$  times. Hence, it has log depth.

You also need to argue why the size is polynomial.

Rough

$$s, s \cdot \frac{2}{3}, s \cdot \left(\frac{2}{3}\right)^2, \dots, s \left(\frac{2}{3}\right)^i$$

$$s \left(\frac{2}{3}\right)^i \sim 1 \Rightarrow i = \log_{\frac{3}{2}}(s)$$

⑤

## Linear Programming is P-complete

Strategy: We take any language  $L \in P$ , then show its log space reduction to linear programming (LP).

5.5

More details: we define LP and explain the meaning of log space reduction.

Linear program (LP): (over variable  $x$ )

maximize ( $c^T x$ ) → objective

s.t.  $Ax \geq b$  → Constraints

The que. defines LP as a feasibility prob. You are defining it as an optimisation prob.  
where,  $c$ ,  $x$  and  $b$  are vectors  
 $A$  is a matrix

Elements of  $c$ ,  $b$  and  $A$  belongs to  $\{-1, 0, 1\}$

Note: This is not an Integer program. We put no restriction on elements of vector  $x$ . Due to construction of LP, it will output integers.

Log space reduction:

Let a language  $L \in P$ . Hence, a polytime TM decides  $L$ .

If  $x^* \in L \Leftrightarrow x^* \rightarrow \boxed{\text{TM}} \rightarrow 1$

Runtime:  $t(n)$  (say)

→ Now we claim there is a log space computable function 'f' s.t

if

$x^* \in L \Leftrightarrow f(x^*) \rightarrow \boxed{\begin{array}{c} \text{Linear} \\ \text{Program} \\ \text{Solver} \end{array}} \rightarrow 1$

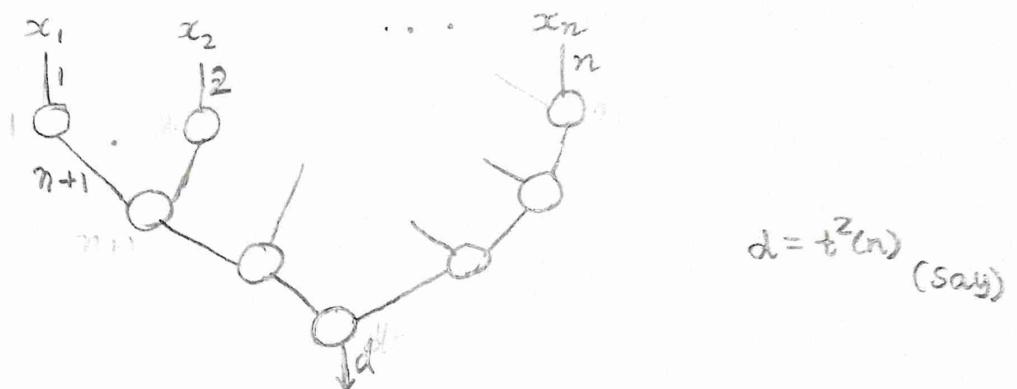
function 'f' takes  $x^*$  and output system of constraints  
TM corresponding to 'f' can use logarithmic amount of space.

Any TM running in  $t(n)$  time has an equivalent circuit of  $t^2(n)$  size. [size = no. of gates] [Cook Levin]

Without loss of generality lets take {OR, AND} gate set as they can represent any other gates with constant overhead.

Circuit (with input) to LP conversion :

Step-I: index each wire of the circuit



Step-II: Visit each gate at a time and output the constrain imposed by it to variable of wires  
 In the Work space, we need to store pointer to the current gate location, i.e.,  
 $\log(|d|)$  bit space required.

Step-III:

→ Let the output of  $i$ th gate be  $x_i$

$$0 \leq x_i \leq 1 \Rightarrow \begin{cases} x_i \geq 0, \text{ and} \\ x_i = 1 \text{ should} \end{cases}$$

be added.

→ At input wire, constrain is

$$x_i = x_i^* \Rightarrow \begin{cases} x_i \geq x_i^* \text{ and} \\ -x_i \geq -x_i^* \end{cases}$$

→ For every NOT Gate



$$x_i = 1 - x_j \Rightarrow \begin{cases} x_i + x_j \geq 1 \text{ and} \\ -x_i - x_j \geq -1 \end{cases}$$

→ For every AND gate

$$\text{AND gate} = \begin{cases} x_j - x_i \geq 0 \\ x_k - x_i \geq 0 \\ x_i - x_j - x_k \geq -1 \end{cases}$$

Consistent  
with AND  
truth table

We notice for each of the conversion, it need to store only current gate's location, output the constraint equation that requires information

You also need of local wire variable. Then move to next node and update pointer. It never need to store the circuit can any other information on work tape apart from be computed in as discussed above.  
log space.

Basically the system of constraints form a polyhedra.  
LP necessarily maximizes the value of objective:  $x_i^*$   
Over the polyhedra.

Due to construction of the Linear program,  $[x_i = x_i^* \in L]$   
No. The value of  $x_i$  in that element is 1 or 0.  
This polyhedra contain single element either 1 or 0,  
based on if input  $x_i^*$  is in language or not  
(respectively).

Argue why  $x^* \in L \Leftrightarrow$  LP is feasible.

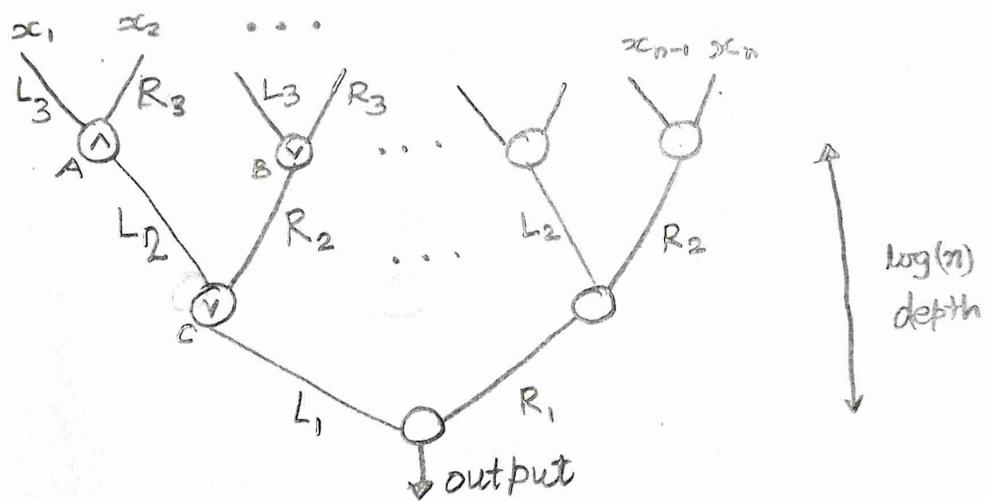
⑥ @ logspace Uniform NC<sup>1</sup>  $\subseteq L$

It consists of those languages that is decided by a circuit family  $\{C_n\}$ , each of size  $O(n^c)$  and depth  $O(\log n)$ .

4.5

Strategy: We show that on taking any circuit from above description, that circuit can be evaluated using a logspace TM.

This is sufficient to show  $NC^1 \subseteq L$



Above graph is an example to reinforce the algorithm to compute circuit value using logspace.

Take a path starting from the output node to input node. This is done via depth-First-Search. Path length is atmost  $\log(n)$ . We evaluate the gates in the path and store them in an another  $\log(n)$  bit space as:

$L_1 \underline{L_2} \underline{L_3} \dots$  (say)  
 $\underline{\log(n)}$  bit  
 for the path

$V_1 \underline{V_2} \underline{V_3} \dots$  (say)  
 $\underline{\log(n)}$  bit  
 for value of gate output in the path

→ The main goal is to optimize space rather than runtime of the algorithm. So, we might revisit the previously visit path if required.

→ While evaluating n gate output at some node, we require input value from both wire. If you need to value from one of the wire is unknown, we say what change path accordingly to estimate it. happens to

Example

→ Say input  $x_1=1, x_2=1$ , we get output at  $v_1, v_2, v_3, \dots$  of the current path when gate-A [see picture]. To get output at gate-C, we need output of gate-B, hence change path as:  $\underline{v_1} \underline{v_2} \underline{1}$  end up needing poly. space.

$(\underline{L_1} \underline{R_2} \underline{L_3})$   
Bottom to Up

$(\underline{v_1} \underline{v_2} \underline{1})$  end up needing  
poly. space.  
Value of  
gate-A

Once we evaluate gate value in new path, i.e., value of Gate-B, we feed it to Gate-C. This output is stored  $v_2$ . Now space for  $v_3$  can be reused.

→ This process goes recursively. We never need to store more space than what is described above to output whole circuit value for given input.

6 (b)

$$NL \subseteq NC$$

Argue why strategy: We take the hardest problem in NL.  
an NC ckt. for PATH PATH is NL-complete (under log-space red.)  
→ an NC we show  $PATH \in NC^2$   
ckt.  $\forall L \in NL \Rightarrow PATH \subseteq NC$  [as  $NC^2 \subseteq NC$ ]

→ In s-t path problem we have graph as adjacency matrix and two vertices 's' and 't'. we see an explicit algorithm to show s-t path can be decided by  $O(\log^2 n)$  depth circuit of poly(n) size.

→ Some useful fact about graph adjacency matrix

Let  $A'$  be adjacency matrix

if entry  $a'_{ij} = 1 \Rightarrow$  There is a path between vertices  $v_i$  and  $v_j$  in graph  $G'$ .

Argue why this is the case in brief. If  $A^n = A'$  and entry  $a'_{ij} = 1 \Rightarrow$  there is a path of length 'n' between vertex  $v_i$  and  $v_j$ .  $\leq n$

→ Now we see how to estimate element  $a'_{ij}$  of  $A^n$  matrix using  $O(\log^2 n)$  depth circuit.

There are two key facts that need to be shown

claim-1: •  $A_{n \times n}^2$  can be computed using  $\log(n)$  depth circuit and poly(n)-size circuit.

claim-2: •  $A^n$  can be computed using repeated squaring using  $\log(n)$  steps.

Proof for claim-I :

Without loss of generality assume

$$P_{n \times n} = A_{n \times n} \cdot A_{n \times n}$$

hence,

$$P_{ij} = \sum_{k=1}^n a_{ik} a_{kj}.$$

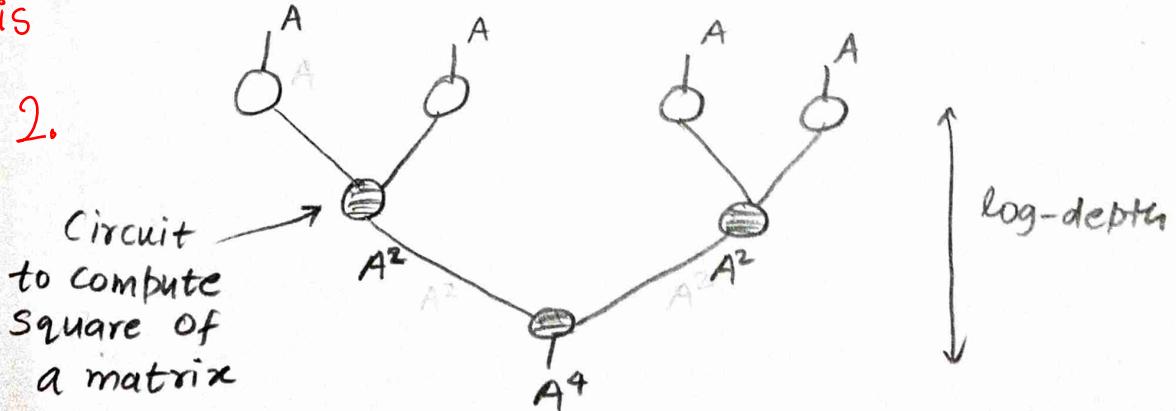
it requires  $O(n)$  multiplication and addition  
that can be done using  $\text{poly}(n)$  size and log depth  
circuit. [addition and multiplication is in  $NC^1$ ]

This holds true for all  $n^2$  entries of  $P_{ij}$ .

The depth remains  $O(\log n)$  just nodes in  
the graph increase by  $n^2$  factor. [Or, we  
still need  $O(\log n)$  time but  $O(n^3)$  processor]

Proof of claim-II :  $A^n$  in  $O(\log(n))$  steps using repeated  
squaring.

Only if  $n$  is  
a power of 2.



→ Thus above construction implies  $O(\log n)$   
circuit to compute  $A^n$ , where each node  
compute square of input matrices using  
 $O(\log n)$  depth circuit.

$$\text{Total depth} = O(\log^2 n)$$

Finally we use the fact that element  $a'_{st} = 1$  in  $A^n$  matrix implies a s-t path of length 'n' in the graph 'G'.

Also, s-t path length never be greater than number of vertices in the graph. This serves as stopping criteria.

The proofs here are much more well written compared to the proofs in the previous assignment.