MCT- ASSIGNMENT 2

— PRISHA
— 202110107 5

Q1) Suppose a given X is NP complete — — —

— — — — — — time complexity of problem X?

Solution

We have been provided following information,

(1) Reduction function $f: \Sigma_1^* \rightarrow \Sigma_2^*$ works in polynomial time. hence it means we have polynomial p such that reduction can be performed in $p(x)$, we have x as input

(2) Problem X can be reduced to SAT in polynomial time and due to this reduction size of instance of X after reduction is X^3

(3) It is ~~required~~ known SAT requires $\Omega(c^n)$ time for some constant $c>1$. This implies that solving SAT instances grow exponentially with respect to input size

now we need to analyze time complexity of X, denoted by $T(x)$

$T(x)$ = Time Complexity of Reduction + time complexity of SAT after reduction

poly(x) $\qquad$ c^{x^3}

hence $T(x) = poly(x) + c^{x^3}$
(polynomial term) (Exponential term)

note now when input is large then exponential term dominates

$T(x) \in \Omega(c^{(x^3)})$

This means that Problem X requires exponential-time to solve if reduced to SAT!

(P2) An oracle to a language $A \subseteq \{0,1\}^*$ _____ NP = CONP.

1) Proof of Existence of an Infinite Sequence

We are assuming $(\leq)$ to be $\{0,1\}$. Without any loss of generality we will represent any 3-SAT instance in binary $(0,1)$.

First, we prove that if $P \neq NP$, then $P \cap NP$ would have no complete problem. If there exist an NP-complete problem α, that is also in P, we can karp reduce NP problem to α (since α is NP-complete). Since α is also in P, it can be solved in P!

Since karp reduction also works in polynomial time, any problem in NP can be solved in polynomial time which implies P = NP (contradiction!)

Hence all NP-complete problem lie in (NP-P). — ①

Since 3SAT is NP-complete, it will belong to (NP-P) from ①.

Let S be the infinite set.

Let S_1 = 3-SAT, accept all binary string that have satisfying assignment.

$$S_i = S_{i-1} \cup \{x \mid x \text{ is multiple of } i \text{ in binary}\}$$

now S_i can be karp reduced to S_{i-1} because we can check if x is multiple of i in polynomial time (polynomial time in length of input). If yes we map to "yes" instance of S_{i-1}, otherwise we use identity function and map x of S_i to x of S_{i-1}. Hence polynomial time monotone (karp reduction) from S_i to S_{i-1}!

Si can not be coak reduced to Si+1. Consider input to turing machine for Si assuming it's a multiple of (i+1), but not i!. There would exist infinite number of such binary string. Since it's a multiple of (i+1) Si+1 will accept the string even if it is not 3-SATISFIABLE.

However we know that it is not multiple of i so Si should accept it and only if it is 3-SAT!

Hence input to Si and Si+1. We know there exist infinite numbers

Note that because 3SAT is NP-complete and since P≠NP, hence no polynomial time algorithm for it!

→ Thus we could not find a mapping. Existence of mapping would imply 3SAT is in P that would be contradiction to P≠NP!

2.) Let S is karp reducible to some set in NP if and only if S is in NP

Proof: let us assume S is karp reducible to S' in NP.

⇒

assume S is karp-reducible to some set in NP, which means there exist polynomial time function f that map 'yes' instance of S to yes instance of problem in NP! and no-instance of S to no instance of problem in NP.

To prove this we can construct a non-deterministic turing machine for S as follows,

→ Given an input x for S, compute f(x). Existence of f(x) is guaranteed by karp reduction from S to S'.

→ Simulate the NDTM for the S on input f(x). Existence of polytime NDTM is guaranteed because it belong to NP!

Thus we constructed NDTM for the set S. Since first part runs in polynomial time and second one works in polynomial time hence we can decide S in polynomial time using NDTM hence S is in NP.

⟸

We know that every problem in NP can be reduced to NP-complete problem. If a set is in NP, it is always reducible to some other set in NP. Eg say 3SAT is NP-complete problem, K-clique problem is also one example! . We can always reduce S' in NP to some set in NP. (Eg we can reduce S' to 3-SAT).

Since we proved either direction hence proved

---

3. If every set in NP can be karp reduced to some set in NP∩coNP then NP=coNP!

now the statement that " if every set in NP can be reduced to some set in NP∩coNP" is defination of completeness. First we have that there can't exist NP complete problem (defined wrt to karp reduction) in NP∩coNP unless NP=coNP,

We will use fact that since every karp reduction if is cook reduction by defination then don't exist NPC problem in NP∩coNP unless NP=coNP.

Claim : If some NP complete problem is in coNP, then NP = coNP!

Proof : Suppose L is some NP-complete problem which is in coNP. Let A be problem in NP, then since L is NP complete problem hence there is polynomial time reduction from A to L! which means $x \in A$ iff $f(x) \in L$.

Since L (NP-complete problem) is also in coNP this implies A is in coNP. because we would have poly reduction from L to problem in coNP for yes/no instances.

Since we can generalize L to any problem in NP. Hence $NP \subseteq coNP$ — (1)

→ Now because L is NP-complete, $\bar{L}$ is coNP-complete. Consider any problem in coNP, say C. Since C is in co-NP, $\bar{C}$ is in NP, thus there exist poly reduction from $\bar{C}$ to L, and this reduction maps yes instances of $\bar{C}$ to yes instance of L, and no instance of $\bar{C}$ to no instance of L. We apply the same reduction function from C to $\bar{L}$ (since it preserves that 'yes' instances are marked to yes and 'no' to no).

Thus, by providing existence of poly reduction function, we proved that coNP problem C can be poly reduced to $\bar{L}$, hence $\bar{L}$ is co-NP complete problem.

Since L and $\bar{L}$ both lie in NP ∩ co-NP. This can be seen by using fact that if any Y on NP, then $\bar{Y}$ will be in coNP. Applying same for L and $\bar{L}$, give us result that it lie in intersection!

Now to let B be any problem in coNP. Then there is a polynomial time reduction f such that $x \in B$ iff $f(x) \in \bar{L}$ (Proved that $\bar{L}$ is co-NP complete). Since $\bar{L}$ is in NP as well, this shows that B can be reduced to NP-problem, proving B is in NP.

Since we can generalise B to be any coNP problem thus $coNP \subseteq NP$ (iii)

Using conclusion from (ii) that $NP \subseteq coNP$ and $coNP \subseteq NP$ hence $NP = coNP$

hence proved!

Q3) If NP≠coNP, then show that NP∩coNP \P is subset of NPI where NPI is class of problem which are not P nor NP complete.

Solution

Given: If NP≠coNP.

To show: NP∩coNP/P is subset of NPI

PROOF:

We would use 'proof by contrapositive': if some NP-complete problem is in coNP, then NP=coNP.

→ Suppose L is some NP-complete problem which is in coNP. Let A be any problem in NP.

→ There would exist polynomial time reduction from A→L (since L is NP complete) i.e $A \leq_p L$ (A is polynomial time reducible to L)

$A \leq_p L \implies \forall x \in A, f(x) \in L$ —— (I)

Since L is in coNP this shows that A is in coNP (from (I))

hence NP is a subset of coNP! —— (II)

Now let B be any problem in coNP. Then there is polynomial time reduction f such that $x \in B$ iff $f(x) \in \bar{L}$ (since L is NP-complete and $\bar{B}$ is in NP). —— (III)

Since $\bar{L}$ is in NP hence this shows B is in NP from (III) hence coNP is subset of NP! —— (III)

Using (II) and (III) we can conclude that NP=coNP!

hence we did proof by contrapositive

Way-2 Proof by contradiction

Let us assume $NP \cap coNP \setminus P \nsubseteq NPI$ (that mean it either belong to P or NP-complete)

$\Rightarrow$ $NP \cap coNP \setminus P \subseteq P$ (CI) or $NP \cap coNP \setminus P \subseteq$ NP-Complete (CII)

Let us consider element $x \in NP \cap coNP \setminus P$ that means,

$x \in NP$ and $x \in coNP$ and $x \notin P$ —①

from ①

$\Rightarrow NP \cap coNP \setminus P \nsubseteq P$ (hence CI is discarded)

now let's consider for the NP-complete set, according to our assumption

if $x \in NP \cap coNP \setminus P$ then $x \in$ NP-complete —⑪

$\therefore \forall$ from ⑪ if $x \in NP$ & $x \in coNP$, then $x \in$ NP-complete.

$\therefore \forall y \in NP, y \leq_P x$ —⑪

Since we have said $x \in coNP$ hence from ⑪ $y \in coNP$ hence

$\Rightarrow \boxed{NP \subseteq coNP}$

$y \in NP$ —⑫

but from ⑪ since $y \in NP$, $\bar{y} \in coNP$.

hence $\boxed{coNP \subseteq NP}$

from drawn conclusions that are $NP \subseteq coNP$, $coNP \subseteq NP$ we have $NP = coNP$ this is a contradiction to given statement

hence our assumption

$NP \cap coNP \setminus P \nsubseteq NPI$ is false indeed

$NP \cap coNP \setminus P \subseteq NPI$ hence proved

Q4) Given $NEXP = \bigcup_{k \in N} NTIME(2^{n^k})$. Show that following problem is NEXP-complete. Given $\langle M, x, N \rangle$ consisting of description of NTM M input x, integer n in binary, does M have an accepting computation of x on n steps.

Solution

To prove: Step I: We will demonstrate that A is in NEXP where

$A = \{ \langle M, x, N \rangle \mid$ NTM M accepts input x in n steps $\}$

Proof:

1) We can simulate NTM M on input x for n steps, in polynomial time in size of input

2) Time complexity of this simulation it atmost polynomial in $|M|$ (description of M), $|x|$ (length of input x), $\log(n)$ (binary represent of n)

3) Time complexity is at worst exponential in input string $(|M|+|x|+\log n)$

Since it runs in exponential time in input size, so

A is in NEXP

Step 2: Reduction to A

To prove that A is NEXP-complete, we need to show that any language L in NEXP can be reduced to A.

$L \leq_p A$

Let L be language in NEXP, by defination there exist a non deterministic TM M^* running in time $2^{\wedge |x|^{\wedge k}}$ where k is some constant

Now we want to decide whether given input $x \in L$, we can reduce it to A

→ construct the input $\langle M^*, x, 2^{|x|^k} \rangle$ for A

→ we are determining whether M^* accept input x in $2^{|x|^k}$ steps.

Hence we establish claim,

Claim: x belong to L if and only if $\langle M^*, x, 2^{|x|^k} \rangle$ belong to A

To prove the claim :

(→ if $x \in L$, it means NTM M^* exist that accept x in $2^{|x|^k}$ steps hence it means $\langle M^*, x, 2^{|x|^k} \rangle$ belongs to A. and vice versa)

Hence from step ① and step ② we have shown that A is in NEXP and any language in NEXP can be reduced to A hence A is NEXP-complete.

Q5) Show that for every ~~two~~ time constructible $t: N \to N$, if $L \in TIME(t(n))$, then there is oblivious TM that decide L in $O(t(n) \log t(n))$ time.

Solution

Given: A time constructible $t: N \to N$

$L \in TIME(t(n))$.

To show: There is an oblivious turing machine that decide L in $O(t(n) \log(t(n)))$ time

We know that function $f(n)$ is called time constructible if there is a deterministic turing machine that, on input 1^n, can compute $f(n)$ in $O(f(n))$ time.

- Oblivious Turing machine add constraint on how the tape is used during computation. In oblivious turing machine, head of machine moves in predetermined manner regardless of input state of ~~matter~~ machine.

It does not have ability to adapt head movement based on content of tape!

1) We would try to describe simpler version that proves time bound in $O(t^2(n))$ time. We take turing machine O with tape alphabet Γ.

Let M be another turing machine such that O simulate M on input X.

The algorithm proceeds by

(1) sweeping across the occupied region of the work tape left to right to find the symbol currently scanned, and hence to moves to execute using [M].

(2) sweeping right to left to execute those moves for which read/write head moves left or stay in place

(3) sweeping left to right to execute rightward head moves of M

(4) move back to left end of tape

This number of occupied cells on work tape of M is at most $T(|x|)$ so each step cost at most $O(T|x|)$ steps i.e $O(T^2|x|)$ in total

To improve time bound for $O(t(n)\log(t(n)))$, we modify the machine O so it has extra tape! Key idea would be to use dummy character which is ignored when we simulate tape content you need to then spread content of tape to have sufficient space (in form of dummy symbol) so that left and right shift only need to move small portion of tape content at each step.

→ We change the original tape of O to be a two way infinite tape with consecutive zones Z_0, Z_1 ——— where Z_i has 2^i cells and this would represent content of both left and right of read head (l_i, r_i)

→ This invariant ensures that 2^i values in $L_i \cup R_i$ will contain dummy symbol.

→ balance between L_i and R_i changes over time and after 2^i simulated steps it is possible that either of L_i or R_i will be packed so that there is no room for further shifting.

→ Every 2^i steps the algorithm will spread it's content so that each of $L_0, R_0, L_1, R_1 \text{ ——— } L_i, R_i$ have half the dummy symbol

↳ Requires shifting $\Theta(2^i)$ element upto $\Theta(2^i)$ positions. Single tape would have lead to $O(T^2(n))$ running time, however using second tape these shifts can be done in $O(2^i)$ time by copying content using second tape.

↳ Since algorithm run only for $T(n)$ steps and $T(n) \geq n$ hence $\log_2 T(n)$ zones are required.

↳ Total cost = $\sum_{i=0}^{\log_2(T(n))} c \cdot 2^i \cdot \frac{T(n)}{2^i} = O(T(n) \log(T(n)))$

hence proved.

Q6) For any real numbers x_1, x_2 ——— n^{x_1}.

Solution

for solving this question we would need to prove a theorem.

Theorem: if f and g are running time satisfying $f(n) = o(g(n))$ then $NTIME(f(n)) \subseteq NTIME(g(n))$

Proof

→ we would take the case $NTIME(n) \subseteq NTIME(n^2)$ to prove it

→ we use the concept of 'lazy diagonalization' because for a non deterministic machine that runs in $o(n)$ time it may have $2^{o(n)}$ branches hence its not clear how would we determine in $O(n^2)$ if it accepts/rejects hence we need lazy diagonalization

→ for pair of integers i, j we would define $f(i,j) = 2^{2^{2^{i} \cdot 3^{j}}}$ since function f is one to one function, hence $(i,j) < (k,l)$ if $f(i,j) < f(k,l)$.

→ define $<$ as linear ordering. We define that for any integer n there would be one largest pair (i,j) and smallest pair (k,l) such that $\boxed{f(i,j) < n \leq f(k,l)}$ say $d_n = f(i,j)$, $H_n = f(k,l)$

→ we can easily compute d_n, H_n for given n in $O(n^2)$ time, clearly $H_n > d(d_n)^2$ so interval $[d_n, H_n)$ is large

→ Diagonalization machine tries to flip the answer in such large interval

→ Let M_1, M_2, M_3 --- be enumeration of all NDTM's. We construct following NDTM D_1.

(1) On input x, if $x \in 1^*$, accept.

(2) if $x = 1^n$, then compute d_n. H_n. If $n = \frac{1}{H_n}$, accept 1^n iff M_i reject $1^{d_{n+1}}$ in $|d_n|^{1.5}$ time. (This means going through all possible $2^{|d_n|^{1.5}}$ non deterministic branches of M_i on input $1^{d_{n+1}}$)

(3) otherwise, simulate M_i on input 1^{n+1} using non determinism in $n^{1.5}$ time and output its answer

→ Clearly D_1 runs in $O(n^2)$ time. We know that D_1 accepts a different language from any NDTM M_i that runs ~~on input 1^n using non determinism in $n^{1.5}$ time and output the answer~~ $O(n)$ time. For contradiction sake assume M_i run in $cn+d$ time and accept the same language as D_1.

→ from the construction of M_i, we know that for all input given such that $f(i,j) = d_n$ some j for some j and $d_n \neq H_n$, D_1 accept 1^n (and hence M_i) iff M_i accept 1^{n+1}!

→ now if M_i and D_1 accept same language, we conclude that M_i accepts 1^n iff M_i accept 1^{n+1}. This chain of implication lead to conclusion that M_i accept $1^{d_{n+1}}$ iff it accept 1^{H_n}.

→ But this leads to contradiction since when j is large enough (specifically large than $c+d$), machine D_1 accept 1^{4n} iff M_i reject 1^{2n+1}. Hence M_i and D_1 cannot accept the same language.

→ We have shown that for any j large enough ($j > c+d$), machine D_1 accepts 1^{4n} iff M_i reject 1^{2n+1}. We will consider implication of this now.

→ Since $f(n) = o(g(n))$, we can choose j sufficiently large such that $2^{2^{2^{2i} \cdot 3^i} \cdot f(n)}$ grows much faster than $g(n)^2$. Specifically we choose j° such that $2^{2^{2^{2i} \cdot 3^i} \cdot f(n)} > g(n)^2$ for all i.

→ Therefore when j is chosen large enough, machine D_1 will accept 1^{4n} but M_i will not accept 1^{2n+1}, leading to contradiction.

→ This establishes that there exist a language accepted by D_1 (and hence language in $NTIME(g(n))$ that is not accepted by any M_i running in $NTIME(f(n))$.

→ Since $f(n) = o(g(n))$ and we can choose sufficiently large j°, this argument hold for any $f(n)$ and $g(n)$ satisfying condition of theorem.

This completes proof of the nondeterministic time hierarchy theorem for general function $f(n)$, $g(n)$ where

$f(n) = o(g(n))$

Proof by limits

for any real number r_1, r_2 such that $1 \le r_1 \le r_2$ we aim to construct a set of strings with non deterministic time n^{r_2} but not n^{r_1}

Claim: There exist a set S such that $NTIME(n^{r_2}) \subseteq NTIME(n^{r_1})$

Proof:

(1) By the Time Hierarchy Theorem (proved above) we know if $f(n) = o(g(n))$, then $NTIME(f(n)) \subseteq NTIME(g(n))$.

(2) let's consider the function $f(n) = n^{r_1}$ and $g(n) = n^{r_2}$ where $1 \le r_1 \le r_2$

(3) we need to show that $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$ for this claim to be true

(4) taking the limit,

$$\lim_{n \to \infty} \frac{n^{r_1}}{n^{r_2}} = \lim_{n \to \infty} n^{r_1 - r_2}$$

(5) since $r_1 \le r_2$ we have $r_1 - r_2 \le 0$

(6) Therefore, $\lim_{n \to \infty} n^{r_1 - r_2} = 0$ which implies $f(n) = o(g(n))$

(7) Applying the time hierarchy theorem there exist a set S such that $NTIME(n^{r_2}) \subseteq NTIME(n^{r_1})$

hence we proved that for any real number r_1, r_2 such that $1 \le r_1 \le r_2$ there exist a set of string with non deterministic time n^{r_2} but not n^{r_1}.

Q7) Prove that if a unary language is NP-Complete then P=NP.

Solution Suppose a unary language L is NP-Complete. That is there exist polynomial time reduction f that when given a SAT formula ϕ as input produces $f(x)$ as output where $\phi \in SAT$ if $f(x) \in L$. We will show how we can solve SAT in polynomial time.

To show: SAT can be solved in polynomial time

Proof: Let L be an NP-complete unary language. Since L is NP-Complete there is polynomial time reduction f from SAT to L. Since f is polytime $|f(x)| \leq c|x|^d$ for some c, d. We will now describe polytime algorithm for SAT.

→ Denote input by ϕ that is formula on n variable $x_1, x_2 \ldots x_n$ (assuming $|\phi| \geq n$)

→ The algorithm proceed in n stages creating sequence of list $L_1, L_2 \ldots L_n$.

→ The list L_k consist of pair $\{f(\psi_i), \psi_i\}$ where ψ_i is formula that we obtain by substituting value for $x_{k+1} \ldots x_n$ in ϕ.

We now maintain invariant that ϕ is satisfiable iff one of ψ_i is satisfiable!

Now we will see more on construction of list. Initial list L_n consist of pair $(f(\phi), \phi)$. Given the list L_k we can construct L_{k-1} in below way:

1) for each $((f(\psi), \psi) \in L_K)$, add to L_{K-1} the two pairs $(f(\psi|_{x_K=T}), \psi|_{x_K=T})$ and $(f(\psi|_{x_K=\perp}), \psi|_{x_K=\perp})$.

2) for each m, out of all pairs $(1^m, \psi)$ (if any) we retain only one.

It is now not hard to check that invariant is maintained. Notice that final set L_0 could potentially contain $(f(T), T)$ and $(f(\perp), \perp)$, formula is satisfiable if and only if it contain former!

Each list L_K would now have size atmost $c|\phi|^d$ hence we have solved SAT in polynomial time.

Hence proved, if unary language is NP-complete then $P = NP$.

Q8) Imagine you and your group ——— MINESWEEPER

NP Complete you would show your supervisor

Solution

Claim: Minesweeper consistency is NP Complete

Proof

→ Minesweeper Consistency is in NP

We begin by showing that Minesweeper Consistency is in NP. Suppose we are given a potential solution to an arbitrary Minesweeper grid (basically revealed board, no hidden squares)

→ In order to verify that the solution is consistent, Computer algorithm would step through grid, one square at time verifying that every number has exactly that number's indicated value of mines as neighbours

→ Since each square has a maximum of 8 adjacent squares to check hence given minesweeper grid with n nodes, runtime will be atmost 8t·n (t: amount of time necessary to check one square's adjacent partner!)

→ Since this verifying algorithm runs in polynomial time - Since this problem has polynomial time verifier hence this problem ~~has~~ is

! NP!

• Minesweeper consistency is NP hard

Proof: Assume we are given an arbitrary boolean circuit that entries the variables $x_1, x_2, \ldots x_n$ and output the value y.

→ This circuit will have series of AND, OR, NOT configuration which eventually assign either true/false to y depending on initial value of variables

→ We need to assign initial boolean values to each variable in such a way that results in true value of y (circuit satisfiability).

→ We can create wire, AND, OR, NOT gates then creating corresponding Minesweeper grid that represent our circuit. At end of minisweeper grid we place mine to represent output value y (ensuring y is true)

→ Assuming we have algorithm for determining Minesweeper grid consistency in polynomial time hence we can find boolean assignment for our initial variable that remain consistent with placing mine only!

→ Using Minesweeper consistency we can determine the circuit satisfiability provided we have polynomial time algo. Since circuit satisfiability is NP-hard hence we conclude by this relation Mine sweeper is also NP-hard

Since Minesweeper consistency is shown to be NP and NP-hard hence NP-complete!!

q9) The last mission was easy. This time thought ——————————
——————————————————— complete the mission.

To prove that shuttle is NP complete we need to prove if

(i) Shuttle is NP

(ii) Shuttle is NP-hard

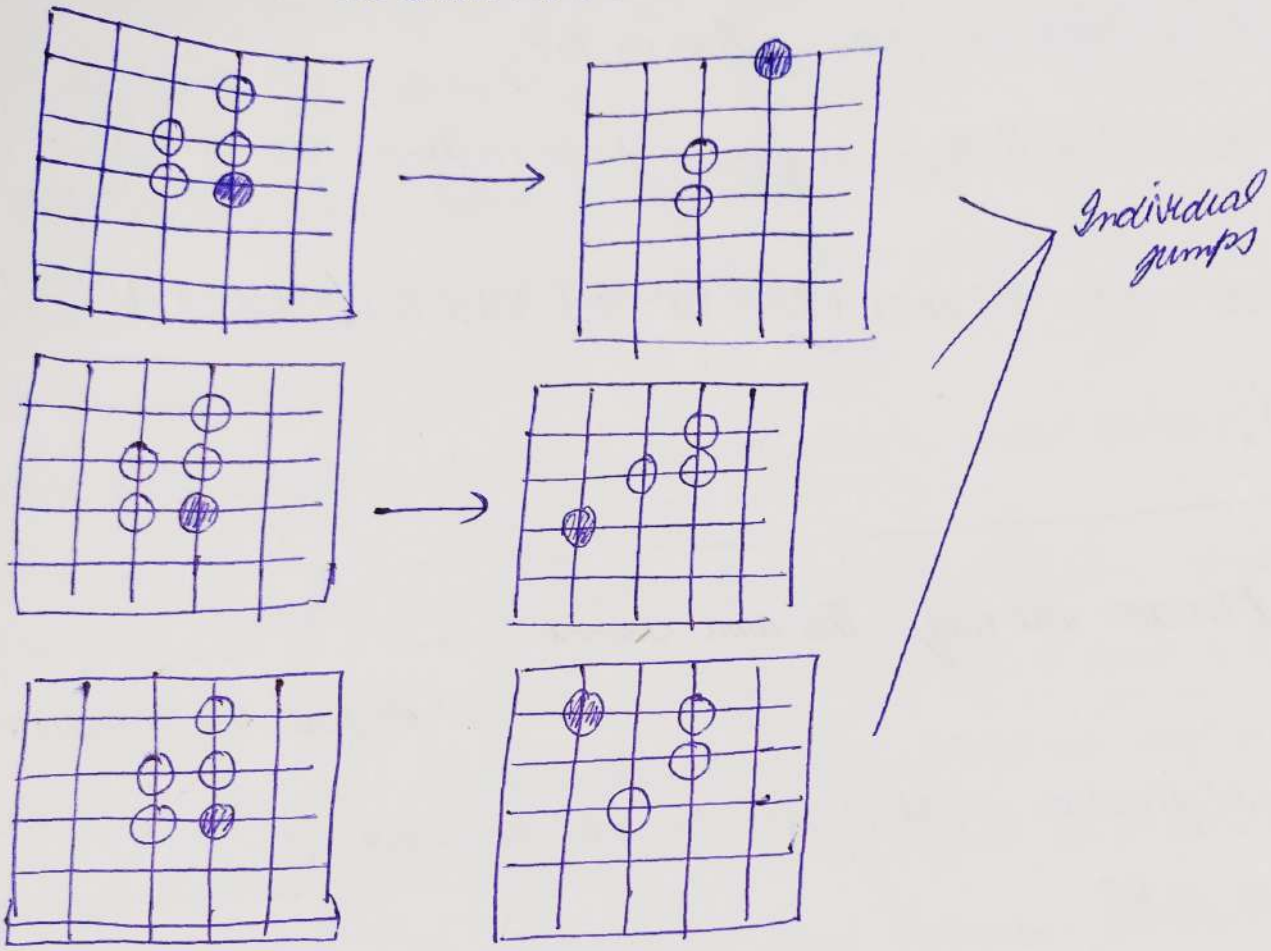
→ Shuttle is NP

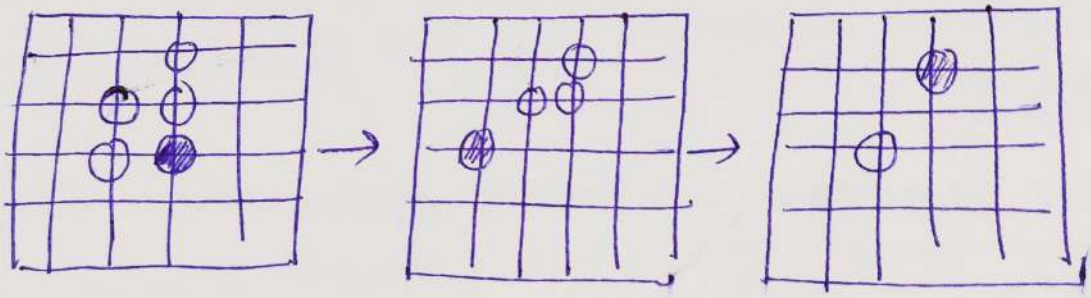
We know that players would take turns to perform action. During the turn player can:

- place new white stone on playing surface or execute move using using black stone
- moving black stone would be possible only if it is next to white stone. then it can jump over one/more white stone to reach first empty spot possible direction using horizontal/vertical/diagonal.
- After jump white stone that were jumped over are removed. If there

there are still viable jumps, current player can continue moving on the black stone.

- Repeat the sequence until no moves are available.





multiple jumps

ultimately leading to winning sequence!

To demonstrate that problem is NP, we can do that by establishing its verifiability in polynomial time. Given an certificate we can efficiently verify by simulating moves on provided board configuration.

Since each jump lead to removal of white stone so number of moves would. We almost count of white stone on table. Hence we can say that certificate can be verified in O(n) time complexity assuming problem is NP!

Solving this problem algorithmically:

Let V be the algorithm:

Given input <b,m>, where b represent pushball board with specific configuration and m is move order

1) for each jump in m, perform

(a) Check if it is a legal move.

(b) Execute the jump by updating position ~~or~~ of black stone

(c) Remove any white stone that black stone jumped over by eliminating them from board configuration!

2) If the board attain winning configuration (black stone reaches opponent goal line / removed from board), accept; otherwise reject!!

NP-Hard: proving that problem is NP-hard.

We organise horizontal line into pair and group. Vertical line into set of three. We interpret horizontal line as binary value that variable can assume while vertical line would serve as clauses! Make ~~three~~ precisely indicate which of three value within clause would have true value!

Construction:

To reduce 3SAT into Pushball we would need following gadgets:

Fan-In and Fan-out gadget :

This would help in movement at extremities of both horizontal & vertical lines. when stone reaches end, fan-in-fan-out gadget allows stone to alter direction and proceed to valid position!

Along horizontal axis, gadget interconnect two line within horizontal pair - On vertical axis, gadget interconnect the clauses!

Non Interacting line Crossing Element :

This would help crossing at the intersection of horizontal & vertical line without interference. It works by removing central segment allowing for jump in alternate direction!

Final Construction using gadgets :

- White stones are arranged in two horizontal line for each variable & three vertical line for each clause

→ for each occurence of variable in 3SAT formula :

- if i^{th} variable appear in j^{th} position for k^{th} clause we place intersection gadget where bottom line of i^{th} horizontal line pair interact with j^{th} line in k^{th} vertical set
- if negation of variable appear we place it intersection gadget on top line of horizontal pair
- for remaining crossing, non interacting line crossing gadget is used.
- Add extra white stone to fan-in gadget allowing back stone to exit after reaching that point

→ 3 unit of separation are required between gadget and variable except to adjacent gadget which require 4 unit of separation. Formula is rearranged such that last variable in clause is different from first variable in next clause!

→ Time Complexity: Resulting grid hence has $6n + O(1)$ rows and $9m + O(1)$ column (where n = variable, m = clauses). $O(1)$ = constant number of gadget used!. Grid hence has polynomially bounded size. Transformation can be executed in polynomial time by examining variable.

having constructed we will ~~first~~ now prove that: 3SAT is satisfiable if and only if Pushball instance created has winning move sequence.

⟹

- Black stone is initially placed to left of first variable ensuring why variable is assigned a value.
- We want a solution that can assign truth value to variable in horizontal line equivalent to assigning value to variable.
- When 3SAT is satisfiable atleast one variable in each clause would be making clause!. During vertical movement we select vertical line corresponding to that value since there will be a proper path up or down the board such that interaction gadget remain intact.

hence if formula is satisfiable a winning move sequence exist and path provides satisfying assignment to variable.

Conversely if there is a clause that evaluate to 0, all intersection gadget on vertical line will be open and a vertical pass won't be available.

If there are clauses that cannot be satisfied under any variable value assignment, interaction gadget will be open and solution to shutball won't exist indicating unsatisfiability.

Thus in all cases shutball problem can simulate 3-SAT instance hence since we proved that it is NP and also NP-hard (as 3SAT $\leq_P$ shutball) so it is NP-complete!

Thanks!