(1.)

Input : Bipartite graph $G = (V, E)$

Output : If G has odd number of perfect Matching.

As discussed in the class,

If B_G is the biadjacency matrix of a bipartite graph G, then $\text{Permanent}(B_G) = \#\text{Perfect Maching}(G)$ → (1)

The question ask for an Algorithm to decide if #Perfect Matching is odd or not.

Eqn① implies

$$\text{Parity}(\#\text{PerfectMatching}(G)) = \text{Perm}(B_G) \pmod 2$$

Note: $\text{Parity}(x) = 0$ iff x is even.

Lemma-1: $\text{Permanent}(B_G) \pmod 2 = \text{Determinant}(B_G) \pmod 2$

Proof : (see next page)

Lemma-2: There exists a polynomial time algorithm to compute determinant of a matrix $M_{n \times n}$.

Proof : Use Gaussian elimination method. (class P)

Or, use Csanky's algorithm, a parallel algorithm. (class NC)

Since, $NC \subseteq P$, we can choose either method to compute the determinant within polynomial time.

Algorithm :

Input : Biadjacency matrix of graph, say B_G

step-I : Compute determinant of B_G

step-II : Take modulo 2 of the step-I result

step-III : If step-II outputs 0, declare the graph is not member of set ⊕PerfectMatching

O.W, declare the graph is in ⊕PerfectMatching

Proof of Correctness:

Since, parity of a number could be either even or odd.

function parity(N) = even iff $N \pmod 2 = 0$

Thus algorithm output, the Graph is in ⊕PerfectMatching iff it has odd number of perfect matching.

Runtime : $\det(B_G)$ takes poly(n) time

mod 2 operation can be implemented by division. Hence, poly(n) time.

Thus, this is a poly-time process.

Proof of Lemma-1:

Main idea : We inspect the below formula to Compute the det(A) and perm(A). The main difference is, in determinant, we put plus or minus sign based on even or odd permutation of the product terms. While, it is kept always plus irrespective of the cases (for permanent).

$$\det(A) = \sum_{\sigma \in S_n} \text{sign}(\sigma) \prod_{j=1}^{n} a_{j\sigma(j)}$$

$$\text{perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i\sigma(i)}$$

using the known result from modular arithematic

$$(x+y) \bmod 2 = (x-y) \bmod 2$$

Justification: $x+y = x-y+2y$

$$\Rightarrow (x+y) \bmod 2 = (x-y+2y) \bmod 2$$
$$= (x-y) \bmod 2$$

Observation: Binary addition and subtraction with modulo 2 operation yeild same result

Note:

$$(A + B + C + D + \cdots) \bmod 2 = \left(\left((A+B) \bmod 2 + C \right) \bmod 2 + \cdots \right.$$

Thus,

$$\det(A) \bmod 2 = \operatorname{perm}(A) \bmod 2$$

■

2.

Lemma 1: Below equation I and II are equivalent

(Ryser formula) $$\text{perm}(A) = \sum_{S \subseteq [n]} (-1)^{n-|S|} \prod_{i \in [n]} \left(\sum_{j \in S} a_{ij} \right) \quad \text{— I}$$

(Definition) $$\text{perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i,\sigma(i)} \quad \text{— II}$$

Proof: (see next page)

→ Algorithm to compute perm(A) via Ryser formula.

Key points and notation

$[n] = \{1, 2, \ldots, n\}$

$S \subseteq [N]$, also power set of $[n] = \{\{\phi\}, \{1\}, \ldots, \{1,2,\ldots,n\}\}$

$|[n]| = 2^n$ (Cardinality)

Term-wise analysis of Ryser formula

$$(-1)^n \sum (-1)^{|S|} \prod_{i \in [n]} \underbrace{\left(\sum_{j \in S} a_{ij} \right)}_{\text{(sum-term, say)}}$$

For a certain $S \subseteq N$, $\prod_{i \in [n]} \left(\sum_{j \in S} a_{ij} \right)$ reduces to product of sums. Each sum contains $|S|$ elements for each values of $i \in [n]$. Thus it has n products.

$$\prod_{i \in [n]} \left(\sum_{j \in S} a_{ij} \right) = \left(\sum_{j \in S} a_{1j} \right) \left(\sum_{j \in S} a_{2j} \right) \cdots \left(\sum_{j \in S} a_{nj} \right) \rightarrow \begin{bmatrix} \text{Say} \\ \text{operation} \\ \text{Ops}(S) \end{bmatrix}$$

any 'sum-term' requires atmost n-addition

→ Each of n terms can be evaluated in poly-time [as addition of n terms scales as poly(n)].

→ Also, we evaluate each 'sum-term' at the same time as they don't depend on each other. [Parallel work-load]

At the end of this step, we have n-terms to product. Since product of 'n' number scales as poly(n), This step is also 'efficient'.

Above calculation is for a certain subset $S \subseteq [n]$. There are 2^n such subsets. Each requires poly(n) steps. Also evaluating $(-1)^{|S|}$ requires poly$(\log |S|)$ time. We simply apply modulo 2 operation and declare 1 and -1 accordingly.

Psuedo Code:

```
Res = 0
for each subset S ⊆ [n] do:
    Res = Res + (-1)^|S| ops[S]
output = (-1)^n Res
```

Runtime: There are 2^n loops, each takes poly(n) time

$$T[n] = 2^n \text{poly}(n)$$

Proof of correctness:

Since we implemented Ryser formula as per its prescription, the algorithm output per(A) due to correctness of Lemma-1.

Proof of Lemma-1: (via principal of Inclusion and Exclusion)

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & & & & \\ \vdots & & & & \\ a_{n1} & & & & a_{nn} \end{bmatrix}$$

Key idea: We start with sometrial sum and product of the matrix element. Then we see how well it represent the permanent. If something extra (or, less) is present remove (or, add) to adjust the expression to match exactly with the permanent.

Steps: start with adding the rows together and multiply them.

$$S_1 = \prod_{j=1}^{n} \left(\sum_{i=1}^{n} a_{ij} \right)$$

This contains permanent plus some extra term. The extra terms are those that contains more than one element from same row. Ex: $a_{11}\, a_{12} \cdots$

Cases-I: a term containing all element from same row.
Case-II: " " " only from any two row.
Case-III: " " " only from any three row
⋮
Case-last: " " only from any $(n-1)$ row

Method to remove such pathological cases:

We start with the last case.

→ $(n-1)$ elements can be picked in several ways from n elements. For each such combination produce the below sum of product.

$$S_2 = (a_{11} + a_{12} + \cdots)(\quad) \cdots$$
$$+ (a_{11} + a_{13} + \cdots (\quad)$$

Get the result $S_2' = S_1 - S_2$

As it turns out it remove some the terms twice.
i.e terms coming from the case just before the last case. (say case 'last-1')

Thus we need to add the case 'last-1' to compensate.

Now, we see a pattern of inclusion and exclusion.

We keep adding and removing the terms from S, in the same fashion. Finally this induction leads to the desired sum that exactly equal perm(A).

3.

Result: 0/1 permanent is #P complete. [Valiant's theorem]

Lemma: Let $A_{n\times n}$ matrix, define $p_A(x_1,\cdots,x_n) = \prod_{i=1}^{n}\sum_{j=1}^{n} x_j \cdot a_{ij}$

$$\text{perm}(A) = \left(\frac{\partial^n}{\partial x_1 \cdots \partial x_n}\right) p_A$$

→ Say operator O_1.

proof: (Next page)

Theorem: Computing $\left(\frac{\partial^n}{\partial x_1 \cdots \partial x_n}\right) p_A$ is equivalent to computing a special instonce of $\left(\frac{\partial^n}{\partial x_1^{e_1} \cdots \partial x_n^{e_n}}\right) p_A$

→ Say. Operator O_e

Proof: given, $e_1 + e_2 + \cdots + e_n = n$

if $e_i = 1 \ \forall i \in [n]$, the desired reduction follow.

Conclusion: Since $O_1 p_A = \text{perm}(A)$, and it reduces to $O_e p_A$. Thus all problem in #P reduces to $O_e p_A$. Hence, $O_e p_A$ is #P-Hard.

Proof of Lemma:

High level idea: we look at expansion of $p_A(x)$ and notice the pattern in the coefficient of the monomial $\prod_{i=1}^{n} x_i \; (= x_1 x_2 \cdots x_n)$. Then we argue it is equivalent to perm(A).

$$p_A(x_1, \ldots, x_n) = \prod_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_j = \prod_{i=1}^{n} (a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n)$$

$$= (a_{11}x_1 + \cdots a_{1n}x_n)(a_{21}x_1 + a_{22}x_2 + \cdots +) \cdots (a_{n1}x_1 + \cdots + a_{nn}x_n)$$

Notice, there are n^n terms in the polynomial.

They can be succinctly represented as,

$$\text{term}_{(\beta)} \{ p_A(x_1, \ldots x_n) \} = \prod_{i=1}^{n} x_{\beta(i)} \, a_{i,\beta(i)}$$

where β is all possible map from set $\{1, 2, \ldots, n\} \longrightarrow \{1, 2, \ldots, n\}$

[There are n^n possible maps]

Let us take the all permutation map. There are $n(n-1) \cdots 1 = n!$ of such maps. [say, σ_n map]

Then the corresponding terms are monomial (as each variable x_i appear exactly once)

since scalar multiplication commute, $(x_1 x_2 \cdots = x_2 x_1 \cdots)$

We sum all the terms of the monomial $(x_1 \cdots x_n)$ to get

$$\sum_{\beta \in \sigma_n} \prod_{i=1}^{n} a_{i\beta(i)}$$

This is equivalent to standard definition of perm(A). ∎

4.

Lemma-1 : $ZPP \subseteq (RP \cap Co\text{-}RP)$

Proof : (Next Page)

Lemma-2 : $(RP \cap Co\text{-}RP) \subseteq ZPP$

Proof : (Next Page)

Theorem : $ZPP = (RP \cap Co\text{-}RP)$

Proof : If set A is contained in set B, also set B is contained in set B. This is possible iff both set-A and B are equal. Thus Lemma-1 and Lemma-2 implies

$ZPP = (RP \cap Co\text{-}RP)$ ∎

Pooof of Lemma-1 :

→ Defn (ZPP): If a language $L \in ZPP$, $\exists M_Z$ (Turing machine) that runs in polynomial time (say, $p(|x|)$) such that for every input 'x', whenever M_Z halts on x, the output $M(x) = L(x)$.

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

→ Simulation of a language $L \in ZPP$ by a PTM machine M_R.

M_R simulate M_Z for, say, $3p(|x|)$ steps. There could be two possibilities :

Scenario:1. M_Z halts and output '1'. (In this case M_R also output '1')

Scenario:2. M_Z doesn't halt. (In this case, M_R output 0)

Now, we do output analysis of M_R.

(Case-I) If $x \in L$,

$Pr[M_R(x) = 1] = Pr[M_Z$ halts within $3p(|x|)$ time and output $1]$

Since, M_Z always give correct result if it halts.

So, $M_Z(x) = M_R(x) = 1$

Analysis of $Pr[M_Z$ halts within $3p(|x|)$ time$]$

$$Pr[M_2 \text{ halts with } 3p(|x|) \text{ time}] = pr[T_{M_2} \leq 3p(|x|)]$$
$$= 1 - pr[T_{M_2} \geq 3p(|x|)]$$

Markov's inequality asserts:

$$pr[X > \ell E[X]] < \frac{1}{\ell}$$

Thus,

$$pr[M_R(x) = 1] \geq 1 - \frac{1}{3} = \frac{2}{3}$$

$1 - \frac{1}{3} = \frac{2}{3}$

(Case-II) If $x \notin L$

$$Pr[M_R(x) = 0] = pr[M_2 \text{ halts within } 3p(|x|) \text{ and output } 0]$$
$$+ pr[M_2 \text{ doesn't halt within } 3p(|x|)]$$

for $x \notin L$, $M_2(x) = 0 = M_R(x)$

Thus,

$$Pr[M_R(x) = 0] = pr[M_2 \text{ halts within } 3p(|x|)] + pr[M_2 \text{ doesn't halt within } 3p(|x|)]$$

$= 1$ (as they are mutually exclusive and exhaustive)

Thus M_R is RP machine that decides any language in ZPP.

To show ZPP $\subseteq$ co-RP, the structure of the proof remain same. But the machine M_C (say) output 1 if M_2 doesn't halt within $3p(|x|)$ time [i.e change in scenario:2 on the last page].

The immediate consequence is -

If $x \in L$; $pr[M_C(x) = 1] = 1$

while $x \notin L$; $pr[M_C(x) = 0] \geq 2/3$.

Thus, M_c is a co-RP machine deciding $\forall L \in ZPP$.

$\Rightarrow ZPP \subseteq$ co-RP

Proof of Lemma-2 : $RP \cap$ co-$RP \subseteq ZPP$

If $L \in (RP \cap$ co-$RP)$, then there exists two PTM, say, M_R and M_c that halts in atmost poly$(|x|)$ steps such that :

$$\left.\begin{array}{l} \text{if } x \in L \Rightarrow Pr[M_R(x) = 1] \geq 2/3 \\ \qquad \text{and } Pr[M_c(x) = 0] = 1 \\ \text{if } x \notin L \Rightarrow Pr[M_R(x) = 0] = 1 \\ \qquad \text{and } Pr[M_c(x) = 1] \geq 2/3 \end{array}\right\} \text{— (A)}^{*}$$

Note: M_R decides L while M_c decide $\bar{L}$.
[This is due to definition of RP and co-RP]
If $M_R(x) = 1 \Rightarrow x \in L$
$M_c(x) = 1 \Rightarrow x \notin L$
We rely on above observation to simulate it by ZPP machine.

Let M_z be a PTM. It acts as follow on an input 'x':

step 1. M_z simulate M_R on input x. If $M_R(x) = 1$, declare $x \in L$. otherwise, Go to step-2.

step 2. M_z simulate M_c on input x. If $M_c(x) = 1$ declare $x \notin L$. otherwise, Go to step-1.

[Note: Whenever M_z declares a result, it halts.]

Now, we need to show the expectation value of the runtime is bounded by poly$(|x|)$.

Expected runtime of machine M_Z.

Based on working of M_Z (PTM), step-1 and step-2 implies M_Z might get stuck in infinite loop in worst case. Thus we analyse average runtime.

Let's define an event 'Bad' if both M_R and M_C output 0. Based on description (A)* on the last page,

$$pr[Bad] \leq \frac{1}{3} \quad [\text{irrespective of } x \in L \text{ or } x \notin L]$$

Let T_x be a random variable denoting the runtime of M_Z on input 'x'.

Since, (conditional expectation)

$$E[T_x \mid \neg Bad] \leq \text{poly}(|x|)$$

also,

$$E[T_x \mid Bad] \leq \text{poly}'(|x|)$$

Using,

$$E[T_x] = E[T_x \mid Bad]\, pr[Bad] + E[T_x \mid \neg Bad]\, pr[\neg Bad]$$

$$E[T_x] \leq \frac{2}{3}\,\text{poly}(|x|) + \frac{1}{3}\,\text{poly}'(|x|)$$

$$E[T_x] \leq \text{poly}''(|x|) \; ; \text{ Bound remains polynomial}$$

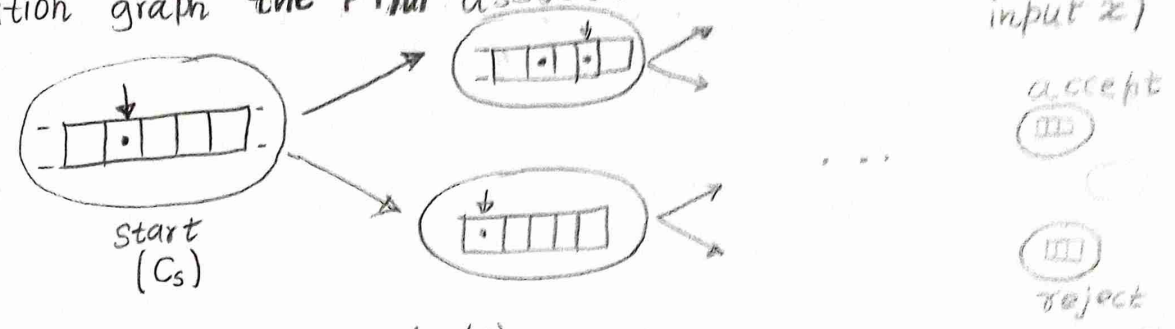
Thus M_Z is a ZPP machine deciding 'L'.

5.

$L \in BPL$ if there is an $O(\log(n))$ space probabilistic Turing Machine M such that

$$Pr[M(x) = L(x)] \geq 2/3 .$$

→ Configuration graph the PTM associated with BPL. (on certain input x)



$\#$ nodes/vertices $= 2^{c \cdot \log(n)} = n^c$

→ If M outputs 1, it reaches accept state. O.W reject state

→ Let the adjacency matrix of the conf. graph $\equiv A_G$

If node i and j are adjucent, then $(A_G)_{ij} = \frac{1}{2}$.

→ Let the runtime of the machine (for x) be 'm'.

Lemma-1 : #path of length 'k' from vertex i to j is equal to the element a_{ij} in A_G^k matrix.

Proof : (Next page)

Lemma-2 : $x \in L$ iff (# path from start to accept) $\geq 2 \cdot$ (# path from start to reject)

Since, Lemma-1 implies

path from start to accept of length atmost 'm'

$$= \sum_{k=1}^{m} (A^k)_{sa} \quad ; \quad s \rightarrow \text{index of start node}, \; a \rightarrow \text{index of accept node}$$

similarly,

path from start to reject node of atmost 'm' length

$$= \sum_{k=1}^{m} (A^k)_{sr} \quad ; \quad r \rightarrow \text{index of reject node}$$

then, $x \in L$ iff $\sum (A^k)_{sa} \geq 2 \sum (A^k)_{sr}$

Proof: (next page)

A polytime algorithm to decide any $L \in BPL$

Input : A PTM (say, M) on input x.

Let a polytime DTM be D that act as following on the input.

step-I : Compute the adjacency matrix for configuration states of PTM M on input x. Let 'A' be the matrix.

step-II : Compute $A^2, A^3, \ldots, A^m$ via matrix multiplication

step-III : Compute $\sum_{k=1}^{m} (A^k) = A^*$ (say)

step IV : If $A^*_{sa} \geq 2\, A^*_{sr}$, declare $x \in L$

otherwise, $x \notin L$

Proof of Correctness: The algorithm is checking the criteria given by Lemma-2. Together with Lemma-1, it implies algorithm decides membership of 'x' in L.

Runtime analysis :

(For step I) since the number of nodes/vertices are poly(n), it requires poly(n) checks (and making adjacency matrix A_G.

(For step-II) Matrix multiplication requires poly(n) operation. [Assume the entries of the matrix have some bound on its value]

(For step-III) Matrix addition and multiplication of a scalar factor to a matrix requires poly(n) operation.

(For step-IV) Element of a matrix can be accessed within poly(n) steps.

Thus, DTM runs for poly(n) steps.

6.

Def: $BP.NP = \{ L : L \leq_r SAT \}$

Lemma-1: If $L \in BP.NP$ then there is a poly-time PTM (say, M_p) such that for every input x,

$$pr\left[L(x) = SAT(M_p(x))\right] \geq 1 - 2^{-|x|}$$

Proof: This is called boosting the success probability. (see next page for formal proof)

Theorem: $BP.NP \subseteq \Sigma_3$

Proof:

→ We use proof strategy similar to Sipser-Gacs-Lautemann(83)

(It introduces the idea of set size A_x that depends on membership of 'x' in L.)

The PTM M_p of Lemma-1, can be equivalently described as a DTM (say, M_d) with access to poly(|x|) size random number generator. [pulling out randomness trick]

Formally, $\exists M_d$ and a polynomial $q(\cdot)$ such that

$$pr_{r \in_R \{0,1\}^{q(|x|)}}\left[L(x) = SAT(M_d(x,r))\right] \geq 1 - 2^{-|x|} \quad \text{— (1)}$$

[By using lemma-1]

[W.L.O.G assume $M_d(x,r)$ is a 3-CNF with input size $p(|x|)$; where $p(\cdot)$ is polynomial.]

Introducing variable for carrying out the proof:

$$n = |x|, \quad m = q(|x|)$$

$$A_x = \{ r \in \{0,1\}^m : SAT(M_d(x,r)) = 1 \} \quad \text{— (2)}$$

Also called the set of all 'good' r for x.

Expression (1) and (2) implies

$$x \in L \Rightarrow |A_x| \geq (1 - 2^{-n}) \cdot 2^m$$

$$x \notin L \Rightarrow |A_x| \leq 2^{-n} \cdot 2^m$$

Take $\frac{m}{n}+1=k$, it implies

Claim(1): If $|A_x| \leq 2^{-n}\cdot 2^m$, then for any collection $u_1,\dots,u_k \in \{0,1\}^m$, $\bigcup_{i\in[k]}(A_x \oplus u_i) \subsetneq \{0,1\}^m$

Reason:
If $|A_x| \leq 2^{m-n}$, $\left|\bigcup_{i\in[k]}(A_x \oplus u_i)\right| \leq k\cdot 2^{-n}2^m < 2^m$
for sufficiently large n.

Claim (2): $|A_x| \geq (1-2^{-n})\cdot 2^m$, then $\exists (u_1,\dots,u_k) \in \{0,1\}^m$ such that $\bigcup_{i\in[k]}(A_x \oplus u_i) = \{0,1\}^m$

Reason: (As mentioned in the lecture note)

High level idea:
- Basically, we pick $\{u_i\}_{i=1}^k$ independently and uniformly at random from $\{0,1\}^m$.
- Fix a certain $r \in \{0,1\}^m$ and $i \in [k]$ case: evaluate the probability bound for this case.
- Apply union bound to get the upper bound on the probability which implies existence of $\{u_i\}$ set in claim(2).

Theorem: claim(2) implies

$$x \in L \Leftrightarrow \exists\, u_1,\dots u_k \in \{0,1\}^m,\ \forall r \in \{0,1\}^h\ \exists w \in \{0,1\}^{p(|x|)}$$
$$M_d(x, u_1,\dots,u_k, r, w) = 1$$

Proof:

High level idea: We start with claim (2) and convert it into the desired expression of the theorem

Claim (2)

$$x \in L \text{ iff } \exists u_1, \ldots, u_k \in \{0,1\}^m, \bigcup_{i \in [k]} A_x \oplus u_i = \{0,1\}^m$$

introducing 'r'

$$x \in L \text{ iff } \exists u_1, \ldots, u_k \in \{0,1\}^m, \forall r \in \{0,1\}^m \; r \in \bigcup_{i \in [k]} (A_x \oplus u_i)$$

Equivalently, (due to property of bitwise XOR operation)

$$x \in L \text{ iff } \exists u_1, \ldots, u_k \in \{0,1\}^m, \forall r \in \{0,1\}^m \bigvee_{i \in [k]} [r \oplus u_i \in A_x]$$

introducing DTM (M_d)

$$x \in L \text{ iff } \exists u_1, \ldots, u_k \in \{0,1\}^m, \forall r \in \{0,1\}^m \; \#$$

$$\# = \bigvee_{i \in [k]} [SAT(M_d(x, r \oplus u_i)) = 1]$$

Using verifier definition of SAT

$$x \in L \text{ iff } \exists u_1, \ldots, u_k \in \{0,1\}^m, \forall r \in \{0,1\}^m \bigvee_{i \in [k]} \left[\exists w \in \{0,1\}^{p(|x|)}, M_d(x, r \oplus u_i)(w) = 1\right]$$

Bringing $\exists w$ outside the disjunction

$$x \in L \text{ iff } \exists u_1, \ldots, u_k \in \{0,1\}^m, \forall r \in \{0,1\}^m, \exists w \in \{0,1\}^{p(|x|)} \bigvee_{i \in [k]} [M_d(x, r \oplus u_i)(w) = 1]$$

Let another DTM (N) compute $\bigvee_{i \in [k]} [M(\cdot) = 1]$

$$x \in L \text{ iff } \exists u_1, \ldots, u_k \in \{0,1\}^m, \forall r \in \{0,1\}^m, \exists w \in \{0,1\}^{p(|x|)}, N(x, u_1, \ldots, u_k, r, w) = 1$$

Finally,

N is a polytime TM taking input as $x, u_1, \ldots, u_k, r, w$
and output 1 iff $M_d(x, r \oplus u_i) = 1$ for some $i \in [k]$

This implies

$$BP.NP \in \Sigma_3$$

7.

Lemma-1: $\Pi_3 \subseteq \Sigma_3 \Rightarrow PH = \Sigma_3$

(Done in class)

Lemma-2: $BP.NP \subseteq \Sigma_3$

(Last problem result)

Given (to assume), $\overline{3SAT} \in BP.NP$

Lemma-3: $BP.NP \subseteq NP/poly$

Proof: (see next page)

Thus, $\overline{3SAT} \subseteq NP/poly$

Our strategy is to show $\overline{3SAT} \subseteq NP/poly \Rightarrow \Pi_3 \subseteq \Sigma_3$

thus, $PH = \Sigma_3$

Theorem: $\overline{3SAT} \subseteq NP/poly \Rightarrow \Pi_3 \subseteq \Sigma_3$. Thus PH collapse to Σ_3.

proof: Strategy, we show the hardest problem in Π_3, i.e, Π_3-3SAT (a Π_3-complete) lies in Σ_3 using the result $\overline{3SAT} \subseteq NP/poly$ (assumption).

Def: Π_3-SAT is the set of boolean formulas ϕ such that $\forall a \exists b \forall c \; \phi(a,b,c) = 1$.

In the above definition, if we fix variables 'a' and 'b', then take negation of the formula.

This reduces $\forall a \exists b \forall c \; \phi(a,b,c) = 1$ to

$\forall c \neg \phi'(c) = 0$ [$\phi' \equiv$ input 'a' and 'b' hard coded to ϕ]

Note: $\forall c \neg \phi'(c) = 0$ is a $\overline{3SAT}$ problem.

Def: (NP/poly) A non-deterministic circuit takes two input x, y. we say 'c' accepts 'x' iff $\exists y$ s.t. $C(x,y) = 1$. The size of the circuit is measured as a function of $|x|$. Then NP/poly be the language that are decided by polynomial size non-deterministic circuit.
[Arora & Barak Ex. 7.7]

Since, $\overline{3SAT} \subseteq$ NP/poly. so, for $\overline{3SAT}$, there exists a poly-size circuit family that can decide $\overline{3SAT}$ with access to a (poly-size) witness 'y'.

Let's denote the circuit description by 'w'
witness by 'y'
then $\overline{3SAT}$ can be expressed as

$$\exists w \forall c \exists y \neg \phi'(c) = 0$$

This is a Σ_3 problem, thus $\Pi_3 \subseteq \Sigma_3$.

$$\Rightarrow PH = \Sigma_3$$

---

which implies it is less likely that Co-NP complete can be randomly reduced to NP-complete problem.
[Due to PH Collapse Conjecture]

Proof of lemma-3 ($BP.NP \subseteq NP/poly$) :

Let $L \in BP.NP$, M be PTM (poly-time machine, assume).
Such that $\forall x$, $\Pr_{r \in_R \{0,1\}^{poly(|x|)}}[L(x) = 3SAT(M(x,r))] \geq 2/3$.

Basically, M is giving a randamized reduction from L to 3SAT.

By boosting technique, probability bound can be made exponentially large as,

$$\Pr_{r \in_R \{0,1\}^{poly(|x|)}}[L(x) = 3SAT(M(x,r))] \geq 1 - 2^{|x|-1}$$

$\Rightarrow$ for a given 'x', $\#r$ s.t $L(x) \neq 3SAT[M(x,r)]$ is atmost $2^m \times 2^{-|x|-1}$; $m = poly(|x|)$

Thus, over all input 'x' of length 'n', the number of such bad 'r' $= 2^n \cdot 2^m \times 2^{-|x|-1} = 2^{m-1}$ [since $|x|=n$]

This number is lesser than total number of 'r' of length 'm'.

$\Rightarrow$ there exists r^* such that

$$\forall x \quad L(x) = 3SAT[M(x,r^*)]$$

Using this fact, we construct a non-deterministic circuit C_n that takes input 'x' ($|x|=n$) and a certificate input 'y'.

$C_n(x,y)$ computes if 'y' is a satisfying assignment of 3SAT instance $M(x,r^*)$.

$C_n(\cdot)$ is a poly-sized circuit as the verification and computing $M(x,r^*)$ require poly-size circuit.

Note: We hard-cord the special r^* to the circuit.

Finally,

for any 'x' (of length 'n')

$x \in L \iff M(x, r^*) \in 3SAT$

$\iff \exists y \; C_n(x, y) = 1$. Thus $L \in NP/poly$ ∎

8.

Input: $A_{n\times n}$ and $B_{n\times n}$ with integer entries

Output: $C_{n\times n}$ s.t. $CA = BC$ (if it exist) where C has rational entries. o.w output 'C doesn't exist'.

Key observation: For given $A_{n\times n}$ and $B_{n\times n}$ matrices assume an unknown matrix $X_{n\times n}$ satisfy the below relationship.

$XA = BX$; where X has n^2 unknown element

Element wise comparison gives-

n^2 equation in n^2 unknown

We use Gaussian elimination method to estimate the unknowns.

Review of Gaussian elimination: $P\vec{s} = 0$

- Convert matrix P into row-echelon form.
- mark 'free' and 'pivot' variable
- Assign values to free variable. This fix the value of pivot variable. Calculate pivot value.
- Output the above assignment for $\vec{s}$ as solution vector.

Constructing X(=P) matrix for our case:

we have $A_{n\times n}$ and $B_{n\times n}$.

$$XA = BX$$

$$\sum_{l=1}^{n} x_{il} a_{lj} = \sum_{k=1}^{n} x_{ik} b_{kj} \qquad \forall\ i,j \in [n]$$

→ (1)

Writting it in $P\vec{s} = 0$ forms: ($\vec{s}$ encodes the elements of the 'X' matrix as vector form)

From equation ①

$$\sum_{1 \le k,\ell \le n} p_{k,\ell}^{(i,j)} x_{k\ell} = 0 \quad \forall\ i,j \in [n] \qquad \text{— ②}$$

where.

$$p_{k,\ell}^{(i,j)} = \begin{cases} a_{ij} & \text{if } k = i,\ \ell \neq j \\ -b_{ik} & \text{if } k \neq i,\ \ell = j \\ a_{ij} - b_{ij} & \text{if } k = i,\ \ell = j \\ 0 & \text{if } k \neq i,\ \ell \neq j \end{cases} \qquad \text{— ③}$$

Equation ② can be written as:

$$P\bar{x} = 0 \quad [P \text{ is } n^2 \times n^2 \text{ dimension matrix}]$$

$[\bar{x}$ is $n^2 \times 1$ dim vector $]$

Note: vector $\bar{x}$ vectorize the matrix X. The following (randamized) algorithm gives $\bar{x}$. Then we accordingly calculate matrix X from the solution vector $\bar{x}$.

step-I: Compute matrix $P_{n^2 \times n^2}$ from $A_{n \times n}$ and $B_{n \times n}$

step-II: Cast it into $P\bar{x} = 0$ template. Use Gauss elimination to change the matrix into row-echolean form.

step-III: Mark pivot and free variable.

step-IV: If there is no free variable, declare there exist no similar matrix C. Exit the algorithm.

step-V: If there are free variables, assign each of them a variable by picking a value from set $\{0, 1, \ldots, 2^n - 1\}$ independently and uniformly at random.

Step-VI: Calculate the value of pivot variable based on assigned free variable. [This uses the constrain imposed by row echelon $[P\bar{x}=0]$.

Step-V: Compute Matrix X from solution vector $\bar{x}$.

Algorithm Runtime:

→ Matrix $P_{n^2 \times n^2}$ requires checking the constrain imposed by equations ③. It scales polynomially to the size of matrices A and B.

→ Gaussian elimination on $n^2 \times n^2$ matrix requires poly(n) arithematic operations.

→ Assume there is a bound on the values of each entries of the matrix.

Thus, algorithm terminate in polynomial time

Probabilistic error bound estimation:

Case-I: if input A and B are similar.

→ Thus, there exists an invertible matrix 'C' s.t. $CAC^{-1} = B \Rightarrow \exists X$, s.t. $XA = BX$. — ⑤

This implies $P\bar{x}=0$ has non-trivial solution. — ⑥

Thus row echelon [P] must have atleast one free variable.

Since algorithm assigns values to free variable randomly, thus we estimate the probability that algorithm output a non-invertible matrix (i.e. det(Matrix)=0)

Let free variable be $x_{f_1}, x_{f_2}, \ldots, x_{f_t}$ (say $= x_{f_i}$)

pivot variable be $x_{p_1}, \ldots x_{p_l}$ ($= x_{p_i}$, say)

we "randomly" assign the value of x_{f_i}. This fixes the value of pivot variable as-

$$x_{p_i} = \sum_{j=1}^{t} c_j^{(i)} x_{f_j} \quad \text{[assume some constant } c_j^{(i)}\text{]} \quad \text{(WLOG)}$$

Now, we construct a matrix $\hat{X}$ that consider the above relationship between two types of variable

$$\hat{x}_{k,l} = \begin{cases} x_{f_i} & \text{if } x_{k,l} \text{ is a free variable} \\ \sum_{j=1}^{t} c_j^{(i)} x_{f_j} & \text{if } x_{k,l} \text{ is a pivot variable} \end{cases}$$

Thus, $\det(\hat{X})$ is function of free variables x_{f_i}.

or,

$$\det(\hat{X}) = f(x_{f_1}, x_{f_2}, \ldots, x_{f_t})$$

Key point:

(1.) f is not a zero polynomial, because we know it yeilds $\det(\hat{X}) = \det(C) \neq 0$ for certain values of $\{x_{f_i}\}$. Such assignment exists due to relation ⑤ and ⑥ on the last page holds. For example, assume (WLOG) there exists only one free variable C_{f_1} related to $CAC^{-1} = B \Rightarrow P\bar{c} = 0$; $\bar{c}$ has non-trivial

Set $C_{f_1} = x_{f_1}$. This is sufficient to show $f(\cdot)$ is non-zero polynomial.

Finally we have a non-zero polynomial $f(\cdot)$.

To estimate the probability that $f(\cdot)$ is equal to zero is quantified by Schwartz-zippel lemma.

Schwartz-zippel lemma: Let $P(r_1, r_2, \dots)$ be a non-zero polynomial of total degree 'd'. Let 'S' be a finite subset of an integral domain 'R'. Let $r_1, r_2, \dots, r_n$ be selected random independently and uniform from 'S'. Then,

$$\Pr[P(r_1, \dots, r_n) = 0] \leq \frac{d}{|S|}$$

In our case the polynomial is $f(\cdot)$.

$S = \{0, 1, \dots, 2^n - 1\} \Rightarrow |S| = 2^n$

Degree of $f(\cdot)$ is n^2

also, $r_1, r_2, \dots, r_t \in_r S$

Thus, (over random assignments)

$$\Pr[f(r_1, r_2, \dots, r_t)] \leq \frac{n^2}{2^n}$$

$$\Pr[\det(\hat{X}) = 0] \leq \frac{n^2}{2^n}$$

$$\Pr[\det(\hat{X}) \neq 0] \geq 1 - \frac{n^2}{2^n}$$

Finally the probability of the algorithm to output non-invertible matrix $\geq 1 - \frac{n^2}{2^n}$.