Computational Complexity - Lecture 18.

Recap: - Promise problems

- SAT $\leq_{RP}$ Unique-SAT (Valiant-Vazirani Lemma).

Next few lectures : the power of counting

$$\#P = \left\{ f : \Sigma^* \to \mathbb{N} \; : \; \begin{array}{l} \text{There is a polytime machine } M(\cdot,\cdot) \\ \text{s.t. } f(x) = |\{w : M(x,w) = 1\}| \end{array} \right\}$$

Not a decision problem but rather a function

$$FP = \left\{ f : \Sigma^* \to \Sigma^* \; ; \; \begin{array}{l} \text{There is a det. polytime machine} \\ M \text{ that outputs } f(x) \text{ on inp } x. \end{array} \right\}$$

What all can you do if $\#P = FP$?

- $P = NP = PH = RP = coRP = BPP$ everything collapses

Comparing with other classes:

$\#P$ - how many witnesses?

RP - Is there $\geq \frac{1}{2} \cdot 2^n$ witnesses, or none?

BPP - Is there $\geq \frac{2}{3} 2^n$ witnesses, or $\leq \frac{1}{3} \cdot 2^n$ witnesses?

Examples of problems in $\#P$:

- Given ϕ - 3CNF, count $\#SAT(\phi)$.

- Given a graph G, count # spanning trees.
  Actually in FP! Look up Kirchoff's tree thm.

- Counting VCs of size k.

Remark: Counting can be hard eventhough detection is easy!

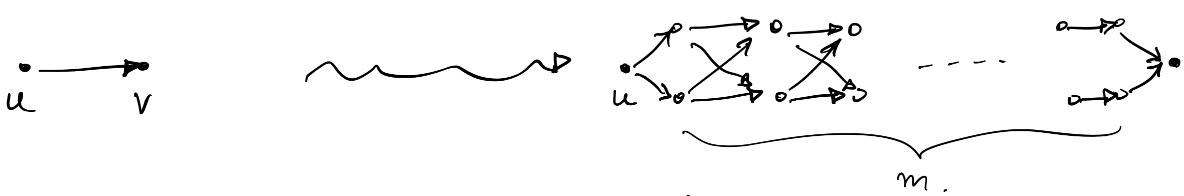
$\#CYCLE(G) = \#$ simple cycles in G.

ie $(v_{i_1}, \ldots, v_{i_r})$ s.t $v_{i_1} = v_{i_r}$ and distinct otherwise and $(v_{i_j}, v_{i_{j+1}}) \in G$.

Lemma: If $\#CYCLE \in FP$, then HamiltonCycle $\in P$.

Pf: Idea: $G \longrightarrow H$

has an n-cycle $\rightsquigarrow$ lots! of different cycles.



2^m possible paths from $u \rightsquigarrow v$.

$\therefore$ Any cycle of length $k \rightsquigarrow 2^{mk}$ many distinct cycles.

Set $m = n^3$.

Claim: There is a Hamiltonian cycle in G. $\Longleftrightarrow$ $\#CYCLE(H) \geq 2^{n^4}$

Pf: $\Rightarrow$: obvious

$\Leftarrow$: Every cycle in H is related to some cycle in G.

If no length n-cycle in G, how many cycles can we have in H?
(#cycles in G). $2^{m\cdot(n-1)}$

$$n! \cdot 2^{m(n-1)} = 2^{n^4 - n^3 + O(n\log n)} \ll 2^{n^4}. \quad \square.$$

#P- completeness .

Defn: (#P-hardness) $f: \Sigma^* \to \mathbb{N}$ is #P-hard if for any $g \in$ #P, we have $g \in FP^{f}$.

f is #P-complete if f is #P-hard & $f \in$ #P.

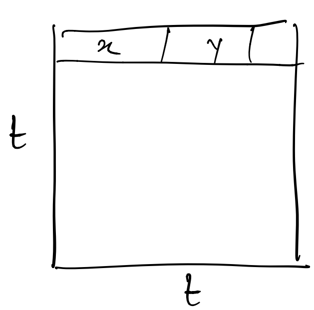
Some candidate #P- complete languages.

▷ The usual example: $f: \langle M, x, 1^t \rangle \mapsto$ #witnesses for M when run on x for just t steps.

▷ Prop: #SAT is #P-complete.

Pf: The Cook-Levin reduction is a parsimonious redn!

$$x \mapsto \varphi_{M,x}(y)$$



$$\varphi_M(z_{1,1}, \dots, z_{tt})$$

$$= \bigwedge (z_{1,*} = \text{start state})$$

$$\wedge \ (z_{t*} = \text{accepting})$$

$$\bigwedge_{ij} \left(z_{ij} = \text{whatever local check says}\right)$$

Every acc y for $M(x,\cdot)$ leads to a unique z.
and vice-versa

$\therefore$ #witnesses for M on x = $\#SAT(\varphi_{M,x}(y))$ $\square$

Fact: #CNF-SAT is also #P-complete.
The counting version of all NP-complete problems we encountered (VC, ind-set, clique) all are #P-hard.
$\because$ the reduction we did were actually parsimonious.
(or can be made so with little effort).

The Permanent:

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

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

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

Claim: Perm of a 0/1-matrix $\in$ #P.

Pf: M on A:

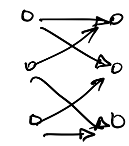
Guess σ. Acc if σ is a permutation and all $a_{i\sigma(i)} = 1$. $\square$.

Thm [Valiant]: 0/1 - Permanent is #P-complete.

We'll actually prove a weaker result, which will show that Perm with entries $\{-2,-1,0,1,2\}$ is #P-hard. Going from here to standard 0/1-Perm is a short step.

Graph theoretic interpretations.

▷ A - bipartite adjacency matrix: $\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$



$\prod a_{i\sigma(i)} = 1 \iff \sigma$ corresponds to a perfect matching.

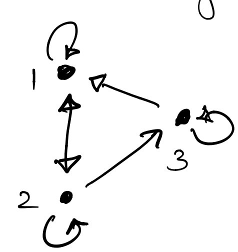
$\therefore$ Perm(A) = # perfect matchings.

For weighted graphs,

Perm(A) = sum of weighted perfect matchings where weight(M) = $\prod$ edge weights.

▷ A - adjacency matrix of a general directed graph.

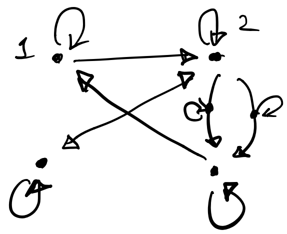
$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$



Defn: A cycle cover of G is a union of disjoint directed cycles that cover all vertices.
wt(cycle cover) = $\prod$ edge weights.

Obs: If A interpreted as the adj. matrix of a directed graph, then Perm(A) = sum of weighted cycle covers.

$$\begin{bmatrix} 1 & 1 & & \\ & 1 & 2 & 1 \\ 1 & & 1 & \\ & 1 & & 1 \end{bmatrix}$$



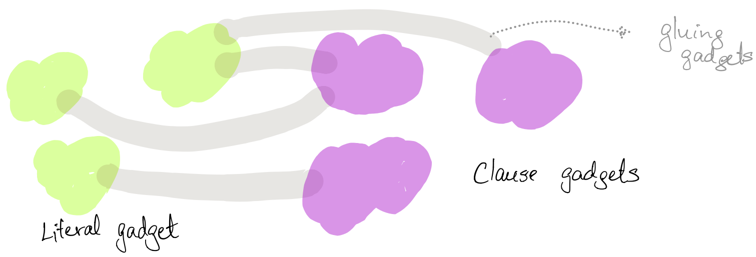
$$\begin{bmatrix} 1 & 1 & 0 & 0 & & \\ & 1 & & 1 & 1 & 1 \\ 1 & & 1 & & & \\ & 1 & & 1 & & \\ & & 1 & & 1 & \\ & & 1 & & & 1 \end{bmatrix}$$

Thm: [Valiant] Perm is #P-hard.

Pf: #3CNF-SAT $\leq$ Perm.

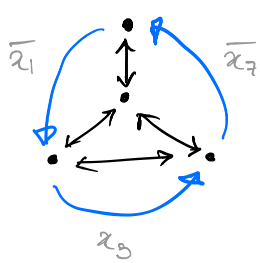
$\varphi \longrightarrow G_\varphi$ - directed graph

satisfying assignments $\longleftrightarrow$ cycle cover in G_φ.



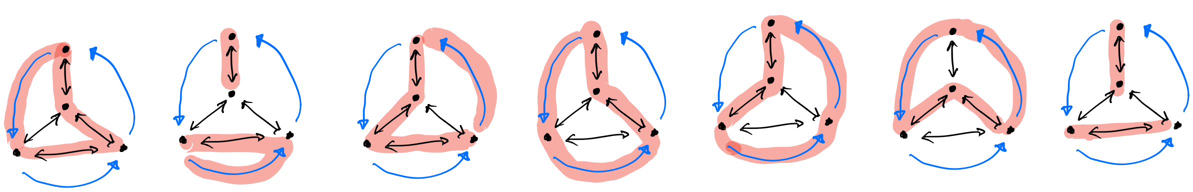
Clause gadget:

$x_1 \vee \bar{x}_3 \vee x_7$

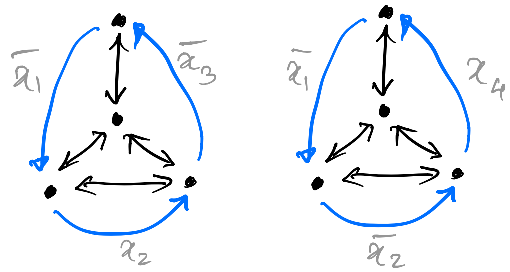


7 satisfying assignments.

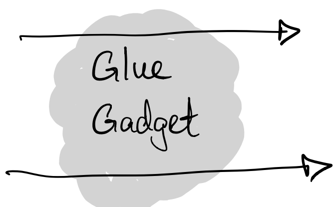
7 cycle covers.
(one for every proper subset of blue edges).



$$\varphi : \quad (x_1 \vee \overline{x_2} \vee x_3)$$
$$(x_1 \vee x_2 \vee \overline{x_4})$$



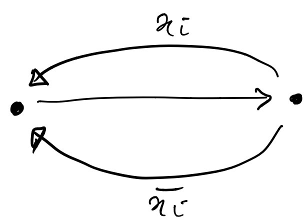
How do we enforce consistency?



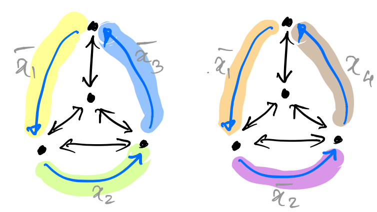
enforces that either

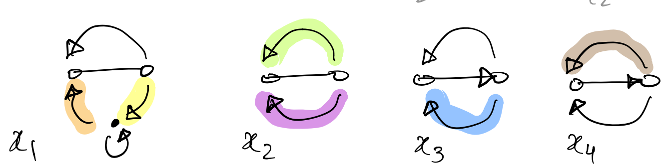
- cycle cover must use both the edges
- cycle cover uses neither.

Variable gadget:

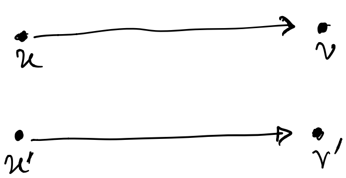


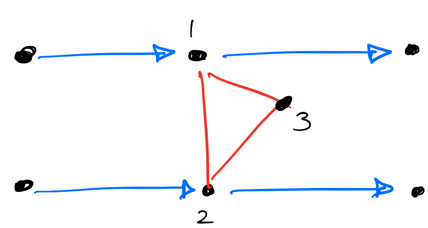
$$(x_1 \vee \overline{x_2} \vee x_3)$$
$$(x_1 \vee x_2 \vee \overline{x_4})$$





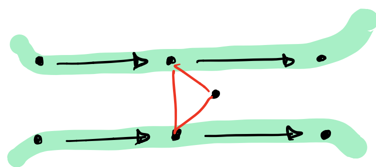
Glue gadget:





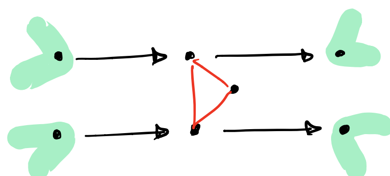
Gadget has adjacency matrix $A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$

What all do we want A to satisfy?

- 

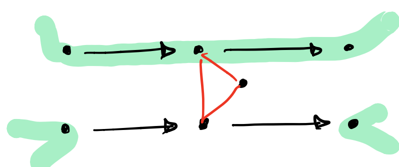
  Should contribute wt 1

  $\Rightarrow \; a_{33} = 1.$

- 

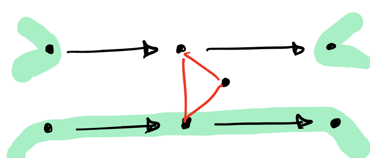
  Should contribute wt 1.

  $\Rightarrow$ Perm $(A) = 1.$

- 

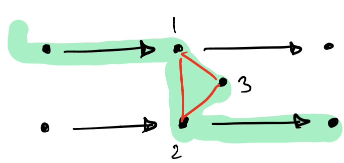
  Should contribute zero.

  $\Rightarrow$ Perm $\begin{bmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{bmatrix} = 0$

- 

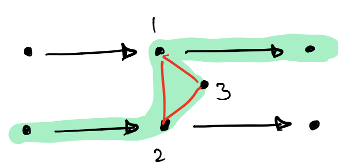
  Should contribute zero.

  $\Rightarrow$ Perm $\begin{bmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{bmatrix} = 0.$

- 

  $$\underbrace{a_{12} \cdot a_{33} + a_{13} a_{32}}_{\text{Perm} \begin{bmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{bmatrix}} = 0$$

  $$\text{Perm} \begin{bmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{bmatrix} = 0$$

- 

  $$\text{Perm} \begin{bmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{bmatrix} = 0.$$

Here is a matrix that works:
$$\begin{bmatrix} -1 & -1 & -1 \\ 1/2 & 1/2 & 1/2 \\ -1 & -1 & 1 \end{bmatrix}$$

Wt ½ is annoying. But if we scale all edge weights by 2, then all cycle covers get weight scaled by 2^m where m = # vertices.

$$\therefore \ \mathrm{Perm}(G_\varphi) = 2^m \cdot \#SAT(\varphi). \qquad \square.$$

Note: ▷ Matrix has entries $\{-2, -1, 0, 1, 2\}$.
With some additional work, we can get a 0/1-matrix.

▷ If you replace Perm by Det above, there is no way to satisfy those constraints!

To show hardness of 0/1 - Perm:

1) #SAT $\leq$ Perm with small, integer entries. → We just saw this.

2) Perm with small int $\leq$ Perm with non-neg. large entries.

3) Perm with non-neg large entries $\leq$ Perm with 0/1 entries.

Pf of (2): $A_{n \times n}$, say entries are just $\{-2,-1,0,1,2\}$.

How large can Perm A be? At most $2^n \cdot n! = M$.

Let $N = 2 \cdot M + 1$. and let $B = A \bmod N$

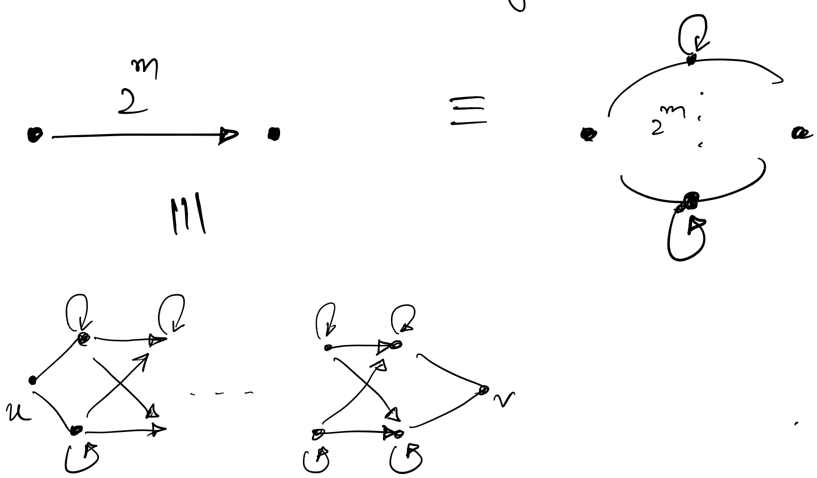
(replace any neg a_{ij} by $B + a_{ij}$)

Perm B = Perm A mod N.

If Perm B mod N < N/2, return that.

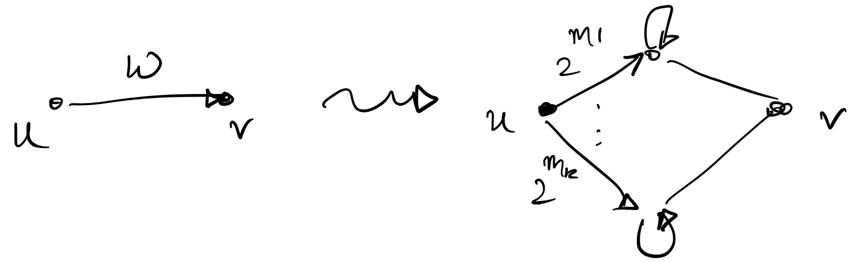
Else, return (Perm B mod N) − N. $\square$

Entries in B are as large as $2^n \cdot n!$ now...



too many.

And if wt on edge $= w = 2^{m_1} + 2^{m_2} + \dots + 2^{m_k}$



and now do the above trick.

This proves (3).