CS 810: Complexity Theory	3/1/2007
Lecture 19: Counting	
Instructor: Dieter van Melkebeek	Scribe: Chi Man Liu

Last time we introduced extractors and discussed two methods to construct them. In the first part of this lecture, we present an explicit construction of extractors based on finite fields [1]. This construction gives extractors with the same parameters as the second construction discussed last time, except that it removes the constraint on the amount of randomness in the weak source.

In the second part of this lecture, we introduce the class #P of counting problems. Problems we have seen in class so far are decision problems where the output is a single bit. On the contrary, counting problems require integers (in the form of binary strings) as output. We look at some properties of #P, as well as relations between #P and other complexity classes.

1 Extractor from Condenser Overview

Recall that in the previous lecture we presented two constructions of polynomial time computable extractors. The first construction is based on pseudorandom generators secure against circuits. This construction gives $m = k^{\Omega(1)}$ and $\ell = O(\log n)$ for sources with $k = n^{\Omega(1)}$. The second construction is based on expander graphs. For any constant $\alpha > 0$, this construction gives a $\delta > 0$ such that for $k = (1-\delta)n$ it extracts $m = (1-\alpha)k$ bits while using $\ell = O(\log n)$ perfect random bits. In this section, we give a construction based on finite fields [1]. This construction uses the second construction as a building block, and achieves the same parameters except that it works for any value of k.

We restate the definition of an extractor here for reference.

Definition 1 (Extractor)**.** *A* (k, ϵ)-extractor *is a function* $E : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^m$ *such that for every source* X *over* $\{0,1\}^n$ *with* $H_\infty(X) \geq k$, $\|E(X, U_\ell) - U_m\|_1 < 2\epsilon$.

Recall that the second construction requires $k = (1-\delta)n$ for some $\delta > 0$. Given any weak random source, we want to apply a procedure that converts the random source into one suitable for applying the second construction. For an arbitrary source on n bits with min-entropy k, we want to condense randomness of the source, i.e., we want to make the ratio k/n close to 1 while keeping a high min-entropy. This intuitive notion is formalized in the following definition.

Definition 2 (Condenser)**.** *A* (k, k', ϵ)-condenser *is a function* $C : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^m$ *such that for every source* X *over* $\{0,1\}^n$ *with* $H_\infty(X) \geq k$, *there exists a distribution* Y *over* $\{0,1\}^m$ *with* $H_\infty(Y) \geq k'$ *and* $\|C(X, U_\ell) - Y\|_1 < 2\epsilon$.

Condensers are a natural generalization of extractors. While we require a near-uniform output from an extractor, we only require the output of a condenser to be close to any source with a certain amount of randomness. Note that in the definition of a condenser if we set $k' = m$, we get back an extractor. We give a construction of a condenser with good enough parameters so that the output distribution can be used as the weak random source of the second construction from last lecture.

Theorem 1. *For any constants* $\alpha, \delta, \epsilon > 0$, *and for all* $k \leq n$, *there exists a* $\mathrm{poly}(n)$ *time computable* (k, k', ϵ)*-condenser with* $k'/k \geq 1-\delta$, $k'/m \geq 1-\alpha$ *and* $\ell = O(\log n)$.

Before proving this theorem, we see how it can be used in combination with the expander based extractor. Let X be a source on n bits with $H_\infty(X) \geq k$ for any value k. For any $\alpha > 0$, pick some constant $\alpha' \in (0, \alpha)$. Suppose that extracting $(1-\alpha')m$ bits using the expander based extractor requires $k = (1-\delta')n$ where $\delta' > 0$ is a constant depending on α'. Applying the condenser on X with $\alpha = \delta'$ produces an output Y on $m' \geq (1-\delta)k$ bits that is $O(\epsilon)$ close to a distribution with min-entropy $k' \geq (1-\delta')m'$ while using a seed of length $O(\log n)$. We can now apply the extractor to get a distrbution Z over $m \geq (1-\alpha')(1-\delta)k$ bits which is $O(\epsilon)$ close to uniform while using an additional seed of length $O(\log k)$. To achieve $m = (1-\alpha)k$, we just need to pick $\delta = 1 - (1-\alpha)/(1-\alpha')$. Overall, a seed of $O(\log n)$ has been used, and composing the two constructions takes $\mathrm{poly}(n)$ time.

2 Condenser

To prove Theorem 1, we need to find a condenser that outputs a distribution with (or close to) high min-entropy for any input source with a guaranteed amount of randomness. For an arbitrary function $C : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^m$, its output distribution may contain elements with large weights, which is a hindrance to achieving high min-entropy. The following lemma shows that if the set of inputs generating these heavy elements is small enough, the output distribution of C will be close to a distribution with high min-entropy.

Lemma 1. *Let X be a source with $H_\infty(X) \geq k$, and C be a function from $\{0,1\}^n \times \{0,1\}^\ell$ to $\{0,1\}^m$. Define the set of k'-*heavy elements *as*

$$H = \{z \mid \Pr_{\substack{x \leftarrow X \\ y \leftarrow U_\ell}}[C(x,y) = z] > 2^{-k'}\}.$$

Let the set of bad inputs *be*

$$BAD = \{x \mid \Pr_{y \leftarrow U_\ell}[C(x,y) \in H] > \frac{\epsilon}{2}\}.$$

If $|BAD| \leq \epsilon \cdot 2^{k-1}$, then there exists a distribution Z over $\{0,1\}^m$ with $H_\infty(Z) \geq k'$ such that $\|C(X,U_\ell) - Z\|_1 < 2\epsilon$.

Proof. Each element in X has weight at most 2^{-k} since X has min-entropy at least k. Thus $\Pr[X \in BAD] \leq |BAD| \cdot 2^{-k} \leq \epsilon/2$. Hence

$$\begin{aligned}
\Pr_{\substack{x \leftarrow X \\ y \leftarrow U_\ell}}[C(x,y) \in H] &\leq \Pr[X \in BAD] + \Pr[X \notin BAD] \cdot \frac{\epsilon}{2} \\
&\leq \frac{\epsilon}{2} + (1 - \frac{\epsilon}{2})(\frac{\epsilon}{2}) \\
&< \epsilon.
\end{aligned}$$

Note that the weight of any element in $\{0,1\}^m \backslash H$ is at most $2^{-k'}$. We can redistribute the weights on H to other elements in $\{0,1\}^m$, resulting in a new distribution Z in which no element has weight greater than $2^{-k'}$, i.e. $H_\infty(Z) \geq k'$. Since the weight needed to redistribute is $\Pr[C(x,y) \in H]$, which is less than ϵ, therefore the difference in probability between $C(X,U_\ell)$ and Y on any set is less than ϵ, so $\|C(X,U_\ell) - Z\|_1 < 2\epsilon$. $\square$

If C is a function such that $|BAD| \leq \epsilon \cdot 2^{k-1}$ for all sources X with min-entropy at least k, then C is a (k, k', ϵ)-condenser. Note that although H depends on the source X, H always satisfies $|H| \leq 2^{k'}$. Thus, if we can show that there are few bad inputs with respect to *any* H satisfying $|H| \leq 2^{k'}$, then we can as well conclude that C is a (k, k', ϵ)-condenser. We have the following lemma.

Lemma 2. *Let H be any subset of $\{0,1\}^m$. Define the* bad set with respect to H *as*

$$BAD_H = \{x \mid \Pr_{y \leftarrow U_\ell}[C(x,y) \in H] > \frac{\epsilon}{2}\}.$$

If $|BAD_H| \leq \epsilon \cdot 2^{k-1}$ for all H with $|H| \leq 2^{k'}$, then C is a (k, k', ϵ)-condenser.

We can now prove our main theorem.

Proof of Theorem 1. We view the sample from the weak random source as specifying a polynomial over a finite field. The extractor evaluates powers of this polynomial at a point specified by the perfect random input and outputs these. We will use the condition given in Lemma 2 to show the construction is a condenser. For any small set in the range, we will construct a non-zero univariate polynomial over samples from the weak random source that evaluates to 0 on samples that hit the small set with high probability. If we can do this while keeping the degree of this polynomial small, this will show that not too many samples can hit the small set in the range with high probability. Previous constructions often use multivariate polyonmials instead of univariate polynomials. While multivariate polynomials generally give lower degrees, univariate polynomials provide us with a better handle on the number of zeroes, which turns out to be useful in our construction.

We now formalize this intuition. Let ℓ be the size of the perfect random seed, n the length of a sample from the weak random source, and m the length of our output. We view $\{0,1\}^\ell$ as $\mathrm{GF}(q)$ for $q = 2^\ell$. We view $\{0,1\}^n$ as $\mathrm{GF}(q)[Y]/g(Y)$ where $g(Y)$ is some irreducible polynomial of degree $n' = n/\ell$ over $\mathrm{GF}(q)$. We view m as being $m' = m/\ell$ elements of $\mathrm{GF}(q)$. Given this setup, we define the condenser C as

$$C(x,y) = (x(y), x^s(y), x^{s^2}(y), x^{s^3}(y), \ldots, x^{s^{m'-1}}(y))$$

for any $x \in \mathrm{GF}(q)[Y]/g(Y)$ and $y \in \mathrm{GF}(q)$, where s is a parameter to be set later. The above formula is interpreted as follows. $C(x,y)$ outputs an m'-vector over $\mathrm{GF}(q)$. The i-th element is computed by treating x as an element of $\mathrm{GF}(q)[Y]/g(Y)$, raising it to the s^{i-1}-th power, then evaluating the resulting polynomial at y.

Let H be any subset of $(\mathrm{GF}(q))^{m'}$ with $|H| \leq 2^{k'}$. Then there exists a nonzero formal m'-variate polynomial Q over $\mathrm{GF}(q)$ that vanishes at all points in H and has degree less than s in each variable. To see why, consider evaluating the polynomial at some point in H. This gives a homogeneous linear equation with the polynomial coefficients as unknowns. There are $|H| \leq 2^{k'}$ such equations and $s^{m'}$ (number of coefficients of Q) unknowns. If

$$s^{m'} > 2^{k'}, \tag{1}$$

such a homogeneous system always has a nontrivial solution, which gives the coefficients of Q. We will pick values at the end of this proof to satisfy Equation 1.

Let $x \in BAD_H$. We treat x as an element of $\mathrm{GF}(q)[Y]/g(Y)$. Consider the formal univariate polynomial $R_x(Y) = Q(x(Y), x^s(Y), \ldots, x^{s^{m'-1}}(Y))$. As all functions in $\mathrm{GF}(q)[Y]/g(Y)$ have degree at most n', $R_x(Y)$ has degree at most $(s-1)m'n'$. Let $y \in \mathrm{GF}(q)$. If $C(x,y) \in H$, then $R_x(y) = 0$ by the construction of C and Q. Hence, from the definition of BAD_H, we know that R_x vanishes on at least $\epsilon q/2$ points in $\mathrm{GF}(q)$. Here we make our second assumption:

$$\epsilon q/2 \geq sm'n'. \tag{2}$$

Then, R_x has more zeroes than its degree, and so it must be the zero formal polynomial. Thus, R_x gives the zero formal polynomial as remainder modulo $g(Y)$. This is equivalent to that R_x is the zero element in $\mathrm{GF}(q)[Y]/g(Y)$.

Let $S(X) = Q(X, X^s, \ldots, X^{s^{m'-1}}) \mod g(X)$ be a univariate polynomial in $\mathrm{GF}(q)[Y]/g(Y)$. From the above discussion, $S(x) = 0$ (the zero element in $\mathrm{GF}(q)[Y]/g(Y)$) for every $x \in BAD_H$. Because the individual degrees of Q were all less than s, S is a nonzero polynomial (because distinct monomials in Q map to distinct monomials in S). The degree of S is at most $(s-1)(1+s+s^2+\cdots+s^{m'-1}) = s^{m'}-1$. Therefore, S has at most $s^{m'}-1$ zeroes, and so $|BAD_H| < s^{m'}$. Our third assumption is

$$s^{m'} \leq \epsilon \cdot 2^{k-1}, \tag{3}$$

which leads to $|BAD_H| \leq \epsilon \cdot 2^{k-1}$, and so by Lemma 2 we can conclude that C is a (k, k', ϵ)-condenser.

We still have to pick the values for the parameters to satisfy our assumptions (Equations 1, 2 and 3 and achieve the claimed parameters of our condenser. By the statement of the theorem and the setup of the condenser, we already have the following: $k' \geq k(1-\delta), m \leq k'/(1-\alpha), q = 2^\ell, m' = m/\ell, n' = n/\ell$. We must show that we can choose s, q, and δ to ensure the assumptions are valid. We consider each in turn. For Equation 1, note that $s^{m'} = s^{m/\ell} = s^{k'/(\ell(1-\alpha))}$. Then Equation 1 is equivalent to $s^{k'/(\ell(1-\alpha))} > 2^{k'}$. By rearranging and using $q = 2^\ell$, this is equivalent to $s > q^{1-\alpha}$. We choose $s = q^{1-\alpha}$ to satisfy Equation 1.[1]

Now consider Equation 2. By substituting our value for s, this is $q\epsilon/2 \geq q^{1-\alpha}m'n'$, which can be rearranged as $q \geq \left(\frac{2m'n'}{\epsilon}\right)^{1/\alpha}$. Notice that $n > n' \geq m'$, so setting $q = \left(\frac{2n^2}{\epsilon}\right)^{1/\alpha}$ satisfies assumption 2. Note that $\ell = O(\log q) = O(\log n)$.

Now consider Equation 3. In fact, if we set $k' = k - 1 - \log(1/\epsilon)$, then with Equation 1 it is straightforward to see that Equation 3 holds.[2]

Finally, consider the efficiency of the construction. Finding an irreducible polynomial $g(Y)$ of degree n' over $\mathrm{GF}(2^\ell)$, as well as arithmetic operations over $\mathrm{GF}(2^\ell)$ and $\mathrm{GF}(2^\ell)[Y]/g(Y)$, can be done in time $\mathrm{poly}(n, \ell) = \mathrm{poly}(n)$. Hence the construction is polynomial-time computable.

□

3 Counting Class #P

All problems we have studied so far are decision problems — problems that require only "yes" or "no" answers. In this section, we introduce a new class of problems that require multiple-bit

[1] In fact, s has to be $q^{1-\alpha} + \zeta$ for some small constant $\zeta > 0$. But for simplicity, we just assume that $s = q^{1-\alpha}$. The rest of the argument still goes through if we increment s by ζ.

[2] Once again, we need to add a small constant to k' to get around the strict inequality.

outputs. In particular, we consider problems of the following form: Given an instance x, what is the number of solutions (or witnesses) to x? This leads us to exploring the class #P of counting problems.

3.1 Examples and Definitions

Before we give the precise definition of the complexity class #P, let us look at an example.

> #SAT: Given a Boolean formula ϕ, find the number of assignments satisfying ϕ.

The above problem is an example of a counting problem. Recall that the corresponding decision problem SAT, which asks whether there exists a satisfying assignment, is in NP. We can construct a NTM M which, on input ϕ, guesses an assignment and accepts if it satisfies ϕ. Note that the number of accepting computation paths of M on ϕ is exactly the number of satisfying assignments for ϕ. This leads us to defining #P in terms of NTMs.

Definition 3. #P $= \{f : \{0,1\}^* \to \mathbb{N} \mid$ *there exists a polynomial-time NTM M such that for all input x, $f(x)$ equals the number of accepting computation paths of M on x* $\}$.

We give some examples of problems in #P.

1. #SAT is in #P. This can be seen from the above discussion.

2. The problem #PM, which asks for the number of perfect matchings in a bipartite graph, is in #P. Note that the corresponding decision problem PM (existence of perfect matchings in bipartite graphs) can be solved in polynomial time. This problem arises from statistical physics.

3. For any polynomial-time decidable graph property (for example, connectivity and acyclicity), counting the number of graphs of a given size n that satisfy the property is a problem in #P. These problems appear very often in enumerative combinatorics.

4. Let A be an $n \times n$ matrix with coefficients in $\mathbb{N}$. (Note: not $\mathbb{Z}$.) Define the *permanent* of A as
$$perm(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)},$$
where S_n is the set of all permutations on $\{1, \ldots, n\}$. Note that the permanent of a matrix is very similar to the determinant except that a factor of $(-1)^{sign(\sigma)}$ is multiplied to each permutation for the determinant, where $sign(\sigma)$ denotes the sign of the permutation σ. The problem of computing the permanent of a given matrix A is in #P. This might not be obvious at first sight. In fact we can construct a NTM M that generates computation paths according to the entries of A as follows. On input A, M guesses a permutation σ. For each entry b captured by σ, M creates b computation paths. (If $b = 0$, reject immediately.) Each of these computation paths keeps on expanding, until all n entries have been processed. The total number of computation paths generated is exactly the product of the entries captured by σ. Summing over all permutations, we get the permanent of A. We now argue that M runs in polynomial time. Guessing a permutation takes $O(n \log n)$ steps since we need to specify n numbers and each number has $O(\log n)$ bits. Let m be the largest integer entry

in A. Spawning m computation paths takes $O(\log m)$ steps since only a constant number of computation paths can be spawned in each step. Thus, the total running time of M is $O(n \log n + n \log m)$, which is polynomial in the input length $\Omega(n^2 + \log m)$. As an aside, the number of perfect matchings in a bipartite graph G is equal to the permanent of the "adjacency matrix" of G. [3]

In the above, we restricted our attention to matrices with non-negative entries. If we relaxed this constraint and let the matrices have negative entries, than computing their permanents would not be a problem in #P, simply because the permanents could be negative. In fact, there is another complexity class that captures this problem:

$$\mathrm{GapP} = \{f - g \mid f, g \in \#\mathrm{P}\}.$$

We can show that the unrestricted permanent problem lies in GapP by constructing a NTM M^+ that only accepts permutations giving positive products, and another NTM M^- that only accepts permutations giving negative products.

3.2 Properties of #P

The complexity class #P exhibits some algebraic properties:

1. #P is closed under addition, i.e. for any two functions f and g in #P, their sum $f + g$ also lies in #P. To show that this holds, let M and N be NTMs inducing f and g respectively. Let P be an NTM which, on input x, immediately creates two computation paths. One of the paths runs the computation of M on x; the other runs the computation of N on x. It is easy to see that the total number of accepting computation paths is $f(x) + g(x)$. More generally, uniform exponential sums of #P functions are also in #P, i.e., for any $g \in \#\mathrm{P}$ and constant c, the function
$$f(x) = \sum_{|y|=|x|^c} g(x, y)$$
is also in #P.

2. #P is closed under multiplication. This can be done by running the second NTM at the end of every accepting computation path of the first NTM. More generally, uniform polynomial products of #P functions are also in #P, i.e., for any $g \in \#\mathrm{P}$ and constant c, the function
$$f(x) = \prod_{|y|=c \cdot \log |x|} g(x, y)$$
is also in #P.

Remark. It follows directly from properties (1) and (2) that computing permanents over $\mathbb{N}$ is in #P.

The complexity class GapP also has the above two properties. Furthermore, it is closed under subtraction, a property which #P does not possess.

[3] The "adjacency matrix" here is the same as usual adjacency matrices except that the rows represent vertices in one partition and the columns represent vertices in the other.

3.3 #P and Other Complexity Classes

Most of the complexity classes we have encountered are classes of decision problems. In order to compare #P with them, we need to make #P an oracle. Denote by $P^{\#\mathrm{P}}$ the class of decision problems solvable using a polynomial-time DTM with access to a #P oracle, and $P^{\#\mathrm{P}[1]}$ the class of decision problems solvable using a polynomial-time DTM that makes at most one query to a #P oracle.

Proposition 1. *The following relations hold:*

(a) $\mathrm{NP} \subseteq \mathrm{P}^{\#\mathrm{P}[1]}$.

(b) $\mathrm{BPP} \subseteq \mathrm{P}^{\#\mathrm{P}[1]}$.

(c) $\mathrm{P}^{\#\mathrm{P}[1]} \subseteq \mathrm{P}^{\#\mathrm{P}} \subseteq \mathrm{PSPACE}$.

Proof. Let L be a problem in NP, and M be a polynomial-time NTM solving L. Then the $\mathrm{P}^{\#\mathrm{P}}$ oracle used in the reduction is simply the function f_M induced by M, and $x \in L$ if and only if $f_M(x) > 0$. This completes part (a).

Part (b) can be proved similarly. Given a polynomial-time probabilistic machine P, we construct a NTM P' that guesses a random bit string (which is polynomial in length) and simulates the computation of P. The oracle used in the reduction is $f_{P'}$, and the oracle machine accepts if and only if the number of accepting computation paths of P' is at least $2/3$ of the total number of computation paths.

Part (c) follows from the fact that the entire computation tree of an NTM can be traversed deterministically in polynomial space. ☐

Remark. The set containments in (a) and (b), as well as $\mathrm{P}^{\#\mathrm{P}} \subseteq \mathrm{PSPACE}$, are conjectured to be proper containments.

We introduce another counting-related complexity class here.

Definition 4. $\oplus\mathrm{P} = \{L \mid$ *there exists a polynomial-time NTM M such that for all x, $x \in L$ if and only if the number of accepting computation paths of M on x is odd* $\}$.

It is obvious that $\oplus\mathrm{P} \subseteq \mathrm{P}^{\#\mathrm{P}[1]}$. The following proposition shows a more interesting property of $\oplus$P, which is also seen in the classes P and BPP. The proof is left as an exercise for the reader.

Proposition 2. $(\oplus\mathrm{P})^{\oplus\mathrm{P}} = \oplus\mathrm{P}$.

3.4 #P-Complete Problems

In this section, we present some #P-complete problems.

Theorem 2. #SAT *is* #P*-complete under* $\leq_m^p$.

Proof. (Sketch) In Lecture 3 we proved that SAT is NP-complete. In that proof we took a polynomial-time NTM M and constructed a Boolean formula ϕ_x capturing the computation of M on some input x. It can be verified that the number of assignments satisfying ϕ_x is the same as the number of accepting computation paths of M on input x. ☐

Remark. The reduction used in the above proof is an example of a *parsimonious reduction*, which is a polynomial-time reduction preserving the number of solutions.

Theorem 3. *Computing the permanent of an integer matrix is* #P*-hard under* $\leq_o^p$.

Proof Idea: It can be shown that given a Boolean formula ϕ with ℓ occurrences of literals, we can construct in polynomial time an integer matrix A such that

$$perm(A) = 4^\ell \cdot \#(\phi), \tag{4}$$

where $\#(\phi)$ denotes the number of assignments satisfying ϕ. Given this construction, a single oracle call is needed to determine $\#(\phi)$. □

Theorem 4. #PM *is* #P*-complete under* $\leq_o^p$.

Proof Idea: This theorem can be proved by reducing integer matrix permanents to #PM, then applying Theorem 3. The reduction has two steps. In the first step, we get rid of all negative entries in the matrix using modular arithmetic. In the second step, we transform the nonnegative integer matrix into a 0/1-matrix (adjacency matrix). The complete proof is left as an exercise. □

Theorem 4 is quite a surprising result, as the corresponding decision problem PM is in P. The reduction from #SAT to #PM is not as simple as the one in Theorem 3. For if that was the case, SAT would be in P as follows: given a formula ϕ, reduce it to a graph G, determine whether G has a perfect matching, then apply Equation 4 above to conclude the satisfiability of ϕ. Each of these steps can be done in polynomial-time.

3.5 Reductions from Counting to Decision Problems

It is clear that any decision problem in NP can be reduced to its corresponding counting problem through a single oracle query. A more interesting question is: Can a counting problem be reduced to its corresponding decision problem through an oracle reduction? If the answer is positive for some #P-complete problem such as #SAT, this would imply that $\#\mathrm{P} \leq_o^p \mathrm{P}^{\mathrm{NP}}$ and so $\mathrm{P}^{\#\mathrm{P}} \subseteq \mathrm{P}^{\mathrm{P}^{\mathrm{NP}}} = \mathrm{P}^{\mathrm{NP}}$. By a theorem in the next lecture, this would give us $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{NP}}$, resulting in a collapse of PH to the second level. However, for some other problems in #P, we can give a positive answer to the above question. An example is the Graph Isomorphism problem (GI), which has not yet been shown to be either in P or NP-complete. This result is also considered as an evidence for GI not being NP-complete.

Recall that two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there exists a bijection $f : V_1 \to V_2$ such that for any $u, v \in V_1$, $(u, v) \in E_1$ if and only if $(f(u), f(v)) \in V_2$. We write $G_1 \cong G_2$ if G_1 and G_2 are isomorphic. Define $\mathrm{GI} = \{\langle G_1, G_2\rangle \mid G_1, G_2$ are graphs and $G_1 \cong G_2\}$. The corresponding counting problem is #GI which, given G_1 and G_2, asks for the number of different bijections satisfying the above edge-preserving condition. It is easy to verify that $\#\mathrm{GI} \in \#\mathrm{P}$. A related counting problem is the Graph Automorphism counting problem (#GA) which, given a graph $G = (V, E)$, asks for the number of *automorphisms* of G (edge-preserving permutations on V).

Theorem 5. $\#\mathrm{GI} \leq_o^p \mathrm{GI}$.

Proof. We only show that #GA $\leq_o^p$ GI. Showing #GI $\leq_o^p$ #GA is left as an exercise for the reader.

Let $G = (V, E)$ be a graph with $|V| = n$. The *automorphisms* of G forms a group $\text{Aut}(G)$. Pick a vertex z. Let F_z be the set of all automorphisms fixing z, i.e. $\mathcal{F}_z = \{\pi \in \text{Aut}(G) \mid \pi(z) = z\}$. Let C_z be the set of vertices that z can be mapped to by some automorphism, i.e. $C_z = \{\pi(z) \mid \pi \in \text{Aut}(G)\}$. For each w in C_z, associate with it an automorphism π_w such that $\pi_w(z) = w$. It follows that every automorphism in $\text{Aut}(G)$ can be uniquely decomposed as $\pi_w \circ \sigma$ where $w \in C_z$ and $\sigma \in \mathcal{F}_z$. Thus, $|\text{Aut}(G)| = |C_z||\mathcal{F}_z|$. This also follows from elementary group theory.

To compute $|C_z|$, we make use of the GI oracle. For each $v \in V$, we can decide if $v \in C_z$ using a "coloring" technique. Intuitively, we "color" v with some color to get a colored graph G_1. Likewise, we "color" z with the same color to get G_2. Then, we ask the oracle whether G_1 and G_2 have a color-preserving isomorphism. Since $v \in G_1$ can only be mapped to $z \in G_2$, $v \in C_z$ if and only if $G_1 \cong G_2$. However, GI does not answer queries regarding color-preserving isomorphisms. One way to achieve this coloring effect is by attaching soem rigid graph[4] of size at least $n + 1$ to v and z.

Computing $|\mathcal{F}_z|$ is in fact another instance of #GA: We can color z with a certain color, then count the number of color-preserving automorphisms of the new graph. Since z is the only vertex with that color, all color-preserving automorphisms of the new graph must fix z. Note that this graph is larger than the original graph G since we have attached a rigid graph to z. However, it is in fact an easier instance of #GA because one of the vertices of G has been fixed. When solving this instance, we can fix a vertex $z' \neq z$ which is not in the clique, and compute $|C_{z'}|$ and $|\mathcal{F}_{z'}|$ by coloring z' with another color. This goes on recursively for at most n steps. In each step, a rigid graph of polynomial size is added to the graph, so the final graph still has polynomial size. Each step takes polynomial time, hence the whole reduction is polynomial-time computable. □

4 Next Time

Next lecture we will discuss the relations between #P and the polynomial hierarchy PH. In particular, we will prove that $\text{PH} \subseteq \text{P}^{\#\text{P}[1]}$. We will also see that although it is unlikely that PH can handle exact counting, it can be shown that we can do approximate counting in the second level of PH.

References

[1] V. Guruswami, C. Umans and S. Vadhan. Extractors and condensers from univariate polynomials. *Electronic Colloquium on Computational Complexity*, Report TR06-134, October 2006.

[4] Rigid graphs are graphs that have only one automorphism — the trivial automorphism. These graphs do exist.