Lecture Notes on

Algorithmic Algebra

Jayalal Sarma
Department of Computer Science and Engineering
IIT Madras, Chennai 600036

Draft—August 25, 2015 and forever

Preface

This lecture notes are produced as a part of the course *CS6842: Algorithmic Algebra* which was a course offered during August to November semester in 2013 and 2015 at the CSE Department of IIT Madras.

Acknowledgements

Thanks to Alexander Shrestov and Markus Blaser for creating nice templates for lecture notes which are being combined and in this document.

Scribe status

Each lecture has a field called **status**. It tells which stage of the edit pipeline is the document currently. Each scribe will be set to choose to either Monday-Tuesday (MT) or Wednesday-Friday (WF) pair of lectures. There are four versions.

1. Alpha (α) (deadline $X + 2$): Rough draft. The MT Scribe is expected to turn-in the rough notes by X+2 (Thursday night 10pm) and WF scribe is expected to turn this in by Sunday night 10pm.

2. Beta (β) (deadline $X+4$) : Final draft. The MT scribe is expected to turn this in by Saturday night 10pm and WF scribe is expected to turn this in by Tuesday night 10pm.

3. Gamma (γ) (deadline $X + 5$) : Corrected draft. This is done by the TAs and given to the instructor.

4. Delta (δ) (deadline $X + 6$) : Final notes. The instructor puts out this final version.

Even after these edits, it is possible that there are still errors in the draft, which may not get noticed. If you find errors still, please report at the git hub itself, or send emails to the instructor or the TAs.

Todo list

List of Scribes

Table of Contents

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: K Dinesh
Date: Aug 3, 2015
Status: δ

Lecture
1

Introduction, Motivation and the Language

1.1 Overview of the course. Administrative, Academic policies

1.2 Introduction and Motivation

Main theme of this course is to use algebra to solve computational problems. Let us consider the following two problems :

Plagarism check Given two C programs P_1 and P_2, check if they are the same under renaming of variables.

Molecule detection Given two chemical molecules check if they have the same structure.

Consider the following simpler variant of plagarism checking where the program submitted has just its variables renamed and all other portions of the program are the same. In this case, given the two versions of the program, we need to check if there is a way to rename the variables of one program to get the other one.

In the second case one could view the given molecules as graphs. The problem is again similar problem, where we want to see if there is way to rename the vertex labels of the graph so that under the relabelling both the graphs are the same.

Our aim in both cases is to check whether the two structures are same under renaming. We are interested in solving this problem on graphs.

Definition 1.1 (Graph Isomorphism)**.** *Two graphs $X_1(V_1, E_1)$, $X_2(V_2, E_2)$ are said to be isomorphic if there is a bijective map $\sigma : V_1 \to V_2$ such that $\forall (u, v) \in V_1 \times V_1$,*

$$(u, v) \in E_1 \iff (\sigma(u), \sigma(v)) \in E_2$$

Problem 1.2. *The* graph isomorphism problem *is the decision problem of checking if given two graphs X_1, X_2 are isomorphic.*

We are also interested in the following special case of the above problem called graph automorphism problem.

Definition 1.3 (Graph Automorphism)**.** *For a graph $X(V, E)$, an automorphism of X is a renaming of the vertices of X given by a bijective map $\sigma : V \to V$ such that $\forall (u, v) \in V \times V$,*

$$(u, v) \in E \iff (\sigma(u), \sigma(v)) \in E$$

We are interested in the set of all bijections such that they are automorphisms of X. We denote this by $Aut(X)$.

Definition 1.4. *For a graph X on n vertices, $Aut(X) = \{\sigma \mid \sigma : [n] \to [n], \sigma$ is an automorphism of $X\}$*

Note that an identity map which takes a vertex to itself always belongs to $Aut(X)$ for all graphs X. Hence the question is : are there any bijections other than the identity map as automorphism of X.

Problem 1.5 (Graph Automorphism Problem)**.** *Given a graph X does $Aut(X)$ has any element other than the identity element.*

One way to see bijections is via permutations. This is because, every bijection define a permutation and vice versa.

Let X be an n vertex graph. Denote S_n to be the set of all permutations on n elements. Hence $Aut(X)$ can be defined as $\{\sigma \mid \sigma \in S_n \text{ and } \sigma \text{ is an automorphism of } G\}$.

We now show that the set $Aut(X)$ has some nice properties. Given $\sigma_1, \sigma_2 \in Aut(X)$, we can compose two permutations as $\sigma_1 \circ \sigma_2 = (\sigma_1(\sigma_2(1)), \sigma_1(\sigma_2(2)), \ldots, \sigma_1(\sigma_2(n)))$. This is same as applying σ_2 on identity permutation and then applying σ_1 to the result. We show that $Aut(X)$ along with the composition operation $\circ$ gives us many nice properties.

- If $\sigma_1, \sigma_2 \in Aut(X)$, then $\sigma_1 \circ \sigma_2$ is also an automorphism of X. The reason is that for any $(u, v) \in X \times X$, $(u, v) \in E \iff (\sigma_2(u), \sigma_2(v)) \in E$ Now applying σ_1 on the previous tuple, we get that $(\sigma_2(u), \sigma_2(v)) \iff (\sigma_1 \circ \sigma_2(u), \sigma_1 \circ \sigma_2(v)) \in E$. Hence $(u, v) \in E \iff (\sigma_1 \circ \sigma_2(u), \sigma_1 \circ \sigma_2(v)) \in E$. This tells that $Aut(X)$ is *closed* under $\circ$.
- The composition operation is also *associative*.
- *Identity* permutation belongs to $Aut(X)$ as observed before.
- Since we are considering bijections, it is natural to consider *inverse* for a permutation σ denoted σ^{-1} with the property that $\sigma \circ \sigma^{-1}$ is identity permutation.

We gave a definition of inverse for an arbitrary permutation. But then a natural question is : given $\sigma \in Aut(X)$, is it true that σ^{-1} also belongs to $Aut(X)$. It turns out that it is true.

Claim 1.6. *For the graph $X(V, E)$ $\sigma \in Aut(X) \iff \sigma^{-1} \in Aut(X)$*

Proof. Recall that $\sigma \in Aut(G)$ iff $\forall (u, v) \in V \times V$, $(u, v) \in E \iff (\sigma(u), \sigma(v)) \in E$. In particular, this must be true for $(\sigma^{-1}(u), \sigma^{-1}(v))$ also. This means,

$$(\sigma^{-1}(u), \sigma^{-1}(v)) \in E \iff (\sigma(\sigma^{-1}(u)), \sigma(\sigma^{-1}(v))) \in E$$

By definition of σ^{-1} we get that $(\sigma^{-1}(u), \sigma^{-1}(v)) \in E \iff (u, v) \in E$. This shows that $\sigma^{-1} \in Aut(X)$ □

Objects which satisfy these kind of properties are called groups.

1.3 Overview of the course

There are two major themes.

- Algorithms for permutation groups.
- Algorithms for polynomials.

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: K Dinesh
Date: Aug 4, 2015
Status: δ

Lecture
2

Algebraic Approach to Primality Testing

In this lecture, we will see an algebraic approach to solving a fundamental problem in Number Theory.

2.1 Application to Number Theory

Following is an algorithmic question that we are interested.

Problem 2.1. *Given a number n in its binary representation, check if it is a prime or not in time $O(poly(\log N))$.*

Note 2.2. *The trivial algorithms that we can think of will depend on n and hence takes time exponential in its input representation.*

Consider the following property about prime number proved by Fermat which is of interest in this context.

Theorem 2.3 (Fermat's Little Theorem)**.** *If N is a prime, then $\forall\ a, 1 \leq a \leq N-1$,*

$$a^N = a \mod N$$

Proof. Fix an $a \in \{1, 2, \ldots N-1\}$. Now consider the sequence $a, 2a, \ldots, (N-1)a$. The question we ask is : can any two of the numbers in this sequence be the same modulo N. We claim that this cannot happen. We give a proof by contradiction : suppose that there are two distinct r, s with $1 \leq r < s \leq N-1$ and $sa = ra \mod N$. Then clearly $N|(s-r)a$ which means $N|a$ or $N|(s-r)$. But both cannot happen as $a, s-r$ are strictly smaller than N.

This gives that all the N numbers in our list modulo N are distinct. Hence all numbers from $1, 2, \ldots, N-1$ appear in the list when we go modulo N. Taking product of the list and the list modulo N, we get

$$(N-1)!a^{N-1} = (N-1)! \mod N$$

By cancelling $(N-1)!$, we get that $a^{N-1} = 1 \mod N$. □

This tells that the above condition is necessary for a number to be prime. If this test is also sufficient (i.e, if the converse of the above theorem is true), we have a test for checking primality of a number. But it turns out that this is not true due to the existance of Carmichael numbers which are not prime numbers but satisfy the above test.

So one want a necessary and sufficient condition which can be used for primality testing. For this, we need the notion of polynomials.

Definition 2.4. *A polynomial $p(x) = \sum_{i=0}^{d} a_i x^i$ with $a_d \neq 0$ denotes a polynomial in one variable x of degree d. Here a_is are called coefficients and each term excluding the coefficient is called a monomial.* ***A polynomial is said to be identically zero, if all the coefficients are zero.***

A polynomial time algorithm for this problem has been found in 2002 by Manindra Agarwal, Neeraj Kayal and Nitin Saxena (called as the AKS algorithm). In their result, they used the following polynomial characterisation for a prime number.

Theorem 2.5 (Polynomial formulation (Agarwal-Biswas 1999))**.** *Let $N \geq 1$ be an integer. Define a polynomial $p_N(z) = (1+z)^N - 1 - z^N$. Then*

$$p_N(z) \equiv 0 (mod\ N) \iff N \text{ is prime}$$

Hence checking if N is prime or not boils down to checking if $p_N(z)$ is identically 0 or not except for the fact that the underlying operations are done modulo N.

Proof of the theorem is as follows.

Proof. Note that $p_N(z) = \sum_{i=1}^{N-1} \binom{N}{i} z^i$ and $\binom{N}{i} = \frac{N(N-1)\ldots(N-i+1)}{1\cdot 2\ldots i}$ with $1 \leq i \leq N-1$.

If N is prime, then $\binom{N}{i} = N \times k_i$ for some integer k_i as none of $1, 2, \ldots i$ divides N. Hence $\binom{N}{i} \mod N = 0$ for every i and $p_N(z) \equiv 0 \mod N$ since the polynomial has all coefficients as zero.

If N is composite, we need to show that $p_N(z) \not\equiv 0 (mod\ N)$ which means that there is at least one non-zero coefficient $p_N(z) \mod N$[1]. Since N is composite, there exists a prime p such that $p \mid N$. Let $p^k | N$ where $k \geq 1$ is the largest exponent of p in the prime factorisation of n. Hence $p^{k+1} \nmid N$.

We first show that $p_N(z)$ is a non zero polynomial by showing that the coefficient of z^i for $i = p$, which is $\binom{N}{p}$, is non zero modulo p^k showing[2] $p_N(z)$ is not a zero polynomial modulo N. Let $N = g \times p^k$. In $\binom{N}{p}$, p of the denominator divides N. Note that p cannot divide g, for if it does, then $p^{k+1} | N$ which is not possible by choice of p and k.

$$\binom{N}{p} = \frac{g \times p^k (N-1)\ldots(N-p+1)}{1\cdot 2\ldots p} = \frac{g \times p^{k-1}(N-1)\ldots(N-p+1)}{1\cdot 2\ldots p-1} \tag{2.1}$$

Hence it must be that p^{k-1} divides $\binom{N}{p}$. This is because, there is no p left now in the denominator to divide N. Now $p^k \nmid \binom{N}{p}$ because, none of the terms $(N-1), \ldots, (N-p+1)$ can have p as a

[1] In class we asked the following question of finding an $a \in \{1, 2, \ldots, N-1\}$ such that $p_N(a) \neq 0 \mod N$. This has a counter example. Consider the case where N is a Carmicahel number. By definition, Carmichael numbers are composite numbers that satisfy Fermat's Litte theorem. Hence if N is Carmichael, $\forall\ a \in \{1, 2, \ldots N-1\}$, we get $a^N = a \mod N$. This gives that $\forall\ a \in \{1, 2, \ldots, N-1\}$

$$p_N(a) = ((a+1)^N - a^N - 1) \mod N = ((a+1) - a - 1) \mod N$$

which is zero modulo N.

[2] Note that if $N | \binom{N}{i}$ then $p^k | \binom{N}{i}$. Taking contrapositive, we get that $\binom{N}{i} \neq 0 \mod p^k$ implies $\binom{N}{i} \neq 0 \mod N$

factor since all these terms are obtained by subtracting at most $p-1$ times from N. Hence $\binom{N}{p} \neq 0 \mod p^k$. This completes the proof. □

Note that Fermat's Little Theorem is a special case of Agarwal-Biswas formulation of primality testing.

Claim 2.6. *Fermat's Little Theorem is a special case of Agarwal-Biswas theorem.*

Proof. To prove Fermat's Little theorem, we need one direction of implication of Agarwal-Biswas theorem :

"If N is prime, then $(1+z)^N = (1+z^N) \mod N$".

Note that Fermat's Little theorem asks about $a^N \mod N$ for $a \in \{1, 2, \ldots, N-1\}$. Since the implication talks about $z^N \mod N$ and $(1+z)^N \mod N$, this suggests an induction strategy on a.

Let N be prime. We check the base case : for $a = 1$, the $1^N = 1 \mod N$. Hence the base case is true. By induction, suppose that $a^N = a \mod N$ for $a \in \{1, 2, \ldots, N-1\}$. Now,

$$\begin{aligned} (a+1)^N &= (a^N + 1) \mod N && \text{[By Agarwal-Biswas as } N \text{ is prime]} \\ &= (a+1) \mod N && \text{[By inducive hypothesis]} \end{aligned}$$

This completes the inductive case. □

This shows that checking if the polynomial $p_N(z)$ is a zero polynomial is an if and only if check for primality of N. So checking primality of a number now boils down to checking if a polynomial is identically zero or not. This is a fundamental problem of polynomial identity testing. We will be discussing about polynomial identity testing and AKS algorithm in our next theme.

Remark 2.7. *One could consider two possible definitions of a polynomial being identically zero and they are not equivalent. Indeed, if all coefficients of a polynomial are zeros then it evaluates to zero on all substitutions of the variables. However, the converse need not be true in all underlying algebraic structures. For example, consider the polynomials $x^p - x$ in a field $\mathbb{Z}_p$ (operations are $+$ and $\times$ modulo p). This is indeed a non-zero polynomial but all evaluations modulo p are zero.*

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: K Dinesh
Date: Aug 5, 2015
Status: δ

Lecture
3

Algebraic Approach to finding Perfect Matchings in Graphs

3.1 Application to Graph algorithms

Consider the following problem of finding perfect matching.

Definition 3.1 (Finding Perfect Matching)**.** *Given a bipartite graph $G(V_1, V_2, E)$, we need to come up with an $E' \subseteq E$ such that $\forall u \in V_1 \cup V_2$, there is exactly one edge incident to it in E'.*

We shall give a polynomial formulation for the problem. Given $G(V_1, V_2, E)$ with vertex sets $|V_1| = |V_2| = n$. Define an $n \times n$ matrix A where $A(i, j) = 1$ if $(i, j) \in E$ and is 0 otherwise for all $(i, j) \in V_1 \times V_2$. Recall the determinant of A given by

$$det(A) = \sum_{\sigma \in S_n} sign(\sigma) \prod_{i=1}^{n} A_{i,\sigma(i)}$$

where

$$sign(\sigma) = \begin{cases} -1 & \text{if } inv(\sigma) \text{ is even} \\ 1 & \text{if } inv(\sigma) \text{ is odd} \end{cases}$$

and $inv(\sigma)$ is defined as $|\{(i, j) \mid i < j \text{ and } \sigma(i) > \sigma(j), 1 \leq i < j \leq n\}|$. We denote $f(x) \equiv 0$ to denote that polynomial $f(x)$ is the zero polynomial.

Lemma 3.2. *For the matrix A as defined as before, $det(A) \not\equiv 0 \Rightarrow G$ has a perfect matching*

Proof. Let $det(A) \not\equiv 0$. Hence there exists a $\sigma \in S_n$ such that $\prod_{i=1}^{n} A_{i,\sigma(i)} \neq 0$. Hence the edge set $E' = \{(i, \sigma(i)) \mid 1 \leq i \leq n\}$ exists in G and since $\sigma(i) = \sigma(j)$ iff $i = j$ for every i, j, E' form a perfect matching.

□

Note that converse of this statement is not true. For example, consider the bipartite graph whose A matrix is $A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. It can be verified that $det(A) = 0$ but the bipartite graph associated has a perfect matching.

Hence the next natural question, similar to our primality testing problem, is to ask for some kind of modification so that converse of previous lemma (lemma 3.2) is true. In the example considered, there were two perfect matchings in G having opposite sign due to which the determinant became 0. So the modification should ensure that perfect matchings of opposite signs does not cancel off in the determinant.

One way to achieve this is as follows. Define a matrix T as

$$T(i,j) = \begin{cases} x_{ij} & (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

where $(i,j) \in V_1 \times V_2$. This matrix is called as the Tutte matrix. Now, if we consider determinant of this matrix, we can see that the monomials corresponding a $\sigma \in S_n$ can be a product of at most n variables. Hence $det(T)$ is a polynomial in n^2 variables with degree at most n.

Also in the expansion of determinant, we can observe that each term picks exactly one entry from every row and column meaning each of the entries picked is from a distinct row and column. Hence each of the them is a bijection and hence is a permutation on n elements. Also corresponding to any permutation on n elements, we can get a term. This gives us the following observation.

Observation 3.3. *Set of monomials in $det(T)$ is in one-one correspondence with set of all permutations on n.*

We now give polynomial formulation for the problem of checking perfect matching in a bipartite graph.

Claim 3.4. *For the matrix T as defined before, $det(T) \not\equiv 0 \iff G$ has a perfect matching*

Proof. By $det(T) \equiv 0$, we mean that the polynomial $det(T)$ has all coefficients as zero. ($\Rightarrow$) Since $det(T) \not\equiv 0$, there exists a $\sigma \in S_n$ such that the monomial corresponding is non-zero and by definition of determinant it must be expressed as product of some n set of variables $\prod_i T(i, \sigma(i))$. The variable indices gives a matching and since there are n variables this is a perfect matching.

($\Leftarrow$) Suppose G has a perfect matching given by a M. Let τ denote the permutation corresponding to M. We now need to show that $det(T) \not\equiv 0$. To prove this, it suffices to show that there is a substitution to $det(T)$ which evaluates to a non-zero value.

Consider the following assignment, $\forall\ i, j$

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in M \\ 0 & \text{otherwise} \end{cases}$$

From the formula of determinant,

$$\begin{aligned} det(T) &= \sum_{\sigma \in S_n} sign(\sigma) \prod_{i=1}^{n} T_{i,\sigma(i)} \\ &= sign(\tau) \prod_{i=1}^{n} A_{i,\tau(i)} + \sum_{\sigma \in S_n \setminus \{\tau\}} sign(\sigma) \prod_{i=1}^{n} T_{i,\sigma(i)} \end{aligned}$$

Now substituting $x_{ij} = a_{ij}$, we get that the first term evaluates to $sign(\tau)$ since all the entires are 1. The second term evaluates to 0 since for all $\sigma \neq \tau$, there must be a j such that $\sigma(j) \neq \tau(j)$. Hence it must be that $a_{j,\sigma(j)} = 0$ and hence the product corresponding to σ goes to 0. □

Hence to check if the bipartite graph G has a perfect matching or not, it suffices to check if the polynomial $det(T)$ is identically zero or not. Checking if a polynomial is identically zero or not is one of the fundamental question in this area.

Note that this problem becomes easy if the polynomial is given as sum of monomial form. In most of the cases, the polynomial will not be given this way. For example if we consider our problem, we are just given the T matrix and $det(T)$ is the required polynomial. Trying to expand $det(T)$ and simplifying will involve dealing with $n!$ monomials which is not feasible.

Hence the computational question again boils down to checking if a polynomial is identically zero or not.

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma M.N.
Scribe: Ramya C
Date: Aug 7, 2015
Status: δ

Lecture
4

Graphs, Groups and Generators

In this lecture we will pose three graph theoretic questions and find answers using approaches in algebra.

4.1 Graph Isomorphism, Automorphism and Rigidity

Definition 4.1. *(Graph Isomorphism.) Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be graphs. We say $G_1 \stackrel{\sim}{=} G_2$ (read as G_1 is* isomorphic to G_2) if there exists a bijection $\sigma : V_1 \to V_2$ such that $\forall (u, v) \in V_1 \times V_2$ we have

$$(u, v) \in E_1 \iff (\sigma(u), \sigma(v)) \in E_2$$

In other words, we say a graph G_1 is isomorphic to G_2 if there exists a relabeling of the vertices in G_1 such that the the adjacency and non-adjacency relationships in G_2 is preserved.

Observation 4.2. *If $|V_1| \neq |V_2|$,then G_1 is not isomorphic to G_2.*

The graph isomorphism problem is stated as follows.

Problem 4.3 (Graph Isomorphism Problem). *Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, test if $G_1 \stackrel{\sim}{=} G_2$ or not.*

A natural question to ask in this setting is that if there is an isomorphism from a graph G to itself.

Let $[n] = \{1, 2, \ldots, n\}$. Let S_n denote the set of all permutations from the set $[n]$ to $[n]$. Let $G = (V, E)$ be a graph. An automorphism of G is a bijection $\sigma : V \to V$ such that $\sigma(G) = G$. Let

$$Aut(G) = \{\sigma | \sigma \in S_n \text{ and } \sigma(G) = G\}$$

be the set of all automorphisms of G. Ideally, we would like to compute the set of automorphism of a graph to itself.

Problem 4.4 (**Graph Automorphism Problem**). *Given a graph G, list the elements of $Aut(G)$.*

The above problem can be expected to be solved in polynomial time only if the output expected is polynomial in length. This brings up the size of the $Aut(G)$ into question. Unfortunately, if G is the complete graph on n vertices. Then $|Aut(G)| = n!$. Hence the above question is not well-formulated.

Since identity permutation is trivially an automorphism for any graph, the $Aut(G)$ is always a non-empty subset of S_n where n is the number of vertices. Hence, one can ask a natural decision variant of the above problem, namely the graph rigidity problem.

Formally, the graph rigidity problem is stated as follows.

Problem 4.5 (**Graph Rigidity Problem**). *Given a graph G, test if $Aut(G)$ is trivial. That is, whether $Aut(G)$ contains only the identity permutation or not.*

More than the size, in the first lecture of this course, we have seen that $Aut(G)$ forms a *subgroup* of S_n. To utilize this structure, we first refresh the definition of an abstract group. From now on, we will use the letter X to denote a graph and G to denote a group.

4.2 Groups and Generators

Definition 4.6. *(Groups.) A set G together with a binary operation $*$ is said to be a group if the following four consitions are met*

- ***Closure*** *: $a, b \in G$, the element $a * b \in G$.*
- ***Associative*** *: For any $a, b, c \in G$, we have $(a * b) * c = a * (b * c)$.*
- ***Existence of Identity*** *: For any $a \in G$ there exists a unique element $e \in G$ such that $a * e = e * a = a$.*
- ***Existence of Inverse*** *: For any $a \in G$ there exists a unique element $b \in G$(denoted by a^{-1}) such that $a * b = b * a = e$.*

Example 4.7.
- *S_n forms a group under composition.*
- *$(\mathbb{Z}_5, +)$ is a group.*

From now on, we will use the letter X to denote a graph and G to denote a group.

Remark 4.8. *Let $(G, *)$ be a group. Let $H \subseteq G$ such that $(H, *)$ also forms a group. We say H is a* subgroup *of G and denote by $H \leq G$.*

Exercise 4.9. *For any graph X, the set $Aut(X)$ forms a group under the composition operation. That is, $Aut(G) \leq M$ where $M = (S_n, \circ)$ is the permutation group.*

Let $(G, *)$ be a group. Let $H \subseteq G$ such that $(H, *)$ also forms a group. We say H is a *subgroup* of G and denote by $H \leq G$. We showed in the first lecture of the course that, for any graph X, the set $Aut(X)$ forms a group under the composition operation. That is, $Aut(G) \leq M$ where $M = (S_n, \circ)$ is the symmetric group.

Let $(G, +)$ be a finite group and $g \in G$ be an element. Let $g^2 = g * g, g^3 = g * g * g$. Similarly $g^k = \underbrace{g * g * \cdots * g}_{\text{k times}}$. Now consider the set $H = \{g, g^2, g^3, \ldots\}$. Since $(G, *)$ is a finite group there must exist a k such that $g^k = g$ in H.

Lemma 4.10. *Let $(G, +)$ be a finite group and $g \in G$ be an element. Let $H = \{g, g^2, g^3, \ldots\}$ be a set of elements. The unique identity e of G is in H.*

Proof. Since $(G, *)$ is a finite group there must exist a k such that $g^k = g$ in H. By definition, $g^k = g^{k-1} * g = g$. As $(G, *)$ is a group, g^{-1} exists in G.
Therefore,

$$g^{k-1} * g * g^{-1} = g * g^{-1} = e$$

□

Definition 4.11 (Generator). *Let $(G, +)$ be a finite group and $g \in G$ be an element. We say an element $g \in G$ is a generator of the set H if for every element $h \in H$ there exists a m such that $h = g^m$. (denoted by $H = \langle g \rangle$).*

Observation 4.12. *$H = \langle g \rangle$ is a subgroup of G. That is, $H \leq G$.*

A quick example is that 1 is a generator for $(\mathbb{Z}_5, +)$

Definition 4.13. *An group $(G, *)$ that can be generated by a single element is called a* cyclic group. *For instance, $(\mathbb{Z}_5, +)$.*

Not every group is cyclic. For instance, one can verify that S_3 (with 6 elements in it) is not cyclic.

Definition 4.14 (Generating Set). *Let $S \subseteq G$ be the set $\{u_1, \ldots, u_k\}$. S is said to be* generating *if $\langle S \rangle = G$.*

Having observed that $Aut(X)$ could have potentially be of exponential size, a reasonable way to formulate the graph automorphism problem is in terms of the generating set of the group. For this, we establish first that $Aut(X)$ have a generating set of size $\mathsf{poly}(n)$? In fact any group has !.

4.2.1 Lagrange's theorem

Let $(G, *)$ be a group. Let $H \leq G$. For any $g \in G$, define the right coset of H in G to be

$$Hg = \{hg \mid h \in H\}$$

Let $g_1, g_2 \in G$ and Hg_1, Hg_2 be the corresponding right cosets. Are there elements that belong to more than one coset of H in G? Is $Hg_1 \cap Hg_2 \neq \phi$? If yes, then the set of right cosets of H in G form a partition of the ground set of G. In that case, how many such cosets are required to cover the entire set G ? Let us answer these two questions.

Lemma 4.15. *Let $g_1, g_2 \in G$ and $Hg_1 = \{hg_1 \mid h \in H\}, Hg_2 = \{hg_2 \mid h \in H\}$. Then*

$$Hg_1 = Hg_2 \text{ or } Hg_1 \cap Hg_2 = \phi.$$

Proof. If $g_1 = g_2$, then by definition $Hg_1 = Hg_2$. Therefore let $g_1 \neq g_2$. We will prove : If $Hg_1 \cap Hg_2 \neq \phi$ then $Hg_1 = Hg_2$. Let $Hg_1 \cap Hg_2 \neq \phi, g \in Hg_1 \cap Hg_2$ we will show

(i) $Hg_1 \subseteq Hg_2$; and

(ii) $Hg_2 \subseteq Hg_1$.

Since $g \in Hg_1$ we know that there exists a $h_1 \in H$ such that $g = h_1g_1$. Similarly $g \in Hg_2$ suggests that there exists a $h_2 \in H$ such that $g = h_2g_2$.

$$h_1g_1 = h_2g_2 = g$$

As $(H, *)$ is a group by itself, h_1^{-1} and h_2^{-1} exists.

$$g_1 = h_1^{-1}h_2g_2 \tag{4.2}$$

$$g_2 = h_2^{-1}h_1g_1 \tag{4.3}$$

(i) $Hg_1 \subseteq Hg_2$
Let $g' \in Hg_1$. This implies there exists a $h' \in H$ such that $g' = h'g_1$. Therefore,

$$g' = h'g_1$$
$$g' = h'(h_1^{-1}h_2g_2) \qquad \text{[By equation (4.2)]}$$

By closure property in $(H, *)$, we have $h'' = h'h_1^{-1}h_2 \in H$. Therefore $g' = h''g_2$, $g' \in Hg_2$.

(ii) $Hg_2 \subseteq Hg_1$
Let $g' \in Hg_2$. This implies there exists a $h' \in H$ such that $g' = h'g_2$. Therefore,

$$g' = h'g_2$$
$$g' = h'(h_2^{-1}h_1g_1) \qquad \text{[By equation (4.3)]}$$

By closure property in $(H, *)$, we have $h'' = h'h_2^{-1}h_1 \in H$. Therefore $g' = h''g_1$, $g' \in Hg_1$.

□

Lemma 4.16. *For every $g \in G$, $|Hg| = |H|$.*

Proof. By construction, for every element in H there exists an element in Hg. So $|Hg| \leq |H|$. We first argue that $|H| \leq |Hg|$. Suppose not. Let $|Hg| < |H|$. Then there exists $h_1, h_2 \in H, h_1 \neq h_2$ such that $h_1g = h_2g$. Since $(G, *)$ is a group, g^{-1} exists. We have $h_1gg^{-1} = h_2gg^{-1}$ which implies $h_1 = h_2$, a contradiction. □

Theorem 4.17 (**Lagrange's Theorem**). *Let $(G, *)$ be a group and $H \leq G$. Then $|H|$ divides $|G|$.*

Proof. Direct consequence of Lemmas 4.15 and 4.16 □

Observation 4.18. *Let $H =< g >$ and $H' =< H, g' >$ where $g' \in G \backslash H$. Then $H \leq H' \leq G$. We have $g \in H' \backslash H$, therefore $|H'| > |H|$. $H' \leq H$. By Theorem 4.17, $|H'| \geq 2|H|$. This shows that every group has a generating set of size $\log |G|$.*

Remark 4.19. *We know that $Aut(X) \leq S_n$ for any graph $X = (V, E)$. By Observation 4.18 $Aut(X)$ has a generating set of size $\log |S_n| = log(n!) \in \mathcal{O}(n \log n)$.*

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Ameya Panse
Date: August, 10 2015
Status: δ

Lecture
5

Orbit-Stabilizer Lemma

We first start with some notations and definitions. Order of a group G is number of elements in the group, that is $|G|$. Let $H \leq G$ and $g \in G$. The right coset of H in G is defined as $Hg = \{hg \mid h \in H\}$. If the mutliplication is on the left, we call it the left coset. That is, $gH = \{gh \mid h \in H\}$. In general, it is not necessary that the left and right cosets are the same. If H is a group such that left and right cosets are the same element-wise, (that is, $\forall g \in G, Hg = gH$), H is a said to be a normal subgroup of G.

Let H be a normal subgroup of G. Choose one element from each of these as a representative of the set (say $[g]$ denote the coset representative of the coset $Hg = gH$). These elements have a group structure among them. This requires a proof, which we will come to in the next few lectures.

5.1 Group Action and Orbits

Let G be a Subgroup of S_n. Let $\alpha \in [n]$ and $g \in G$. We denote by α^g is the image of α under the permutation g. Orbit of an element α in G is the set of elements it gets mapped to under permutations in G. More formally,

Definition 5.1 (**Orbit of α in G**). *The orbit of α in G is defined as*

$$\alpha^G = \{\alpha^g \mid g \in G\}$$

This defines a natural relation among elements of $[n]$.

$$\alpha \backsim \beta \leftrightarrow \exists g \in G, \alpha^g = \beta$$

We check that this is an equivalence relation. $e \in G$, where e is the identity element. Hence, $\alpha \backsim \alpha$. Let $\alpha \backsim \beta$. Thus $\exists g \in G, \alpha^g = \beta$. Hence, $\alpha = \beta^{g^{-1}}$. Thus, $\beta \backsim \alpha$. The relation $\backsim$ is an equivalence relation. For transitivity, let $\alpha \backsim \beta, \beta \backsim \gamma$. By definition, $\exists g_1, g_2 \alpha^{g_1} = \beta, \beta^{g_2} = \gamma$. By composition of permutations, $(\alpha^{g_1})^{g_2} = \gamma$. Hence, $\alpha \backsim \gamma$.

The stabilizer of α in G is

$$G_\alpha = \{g \mid \alpha^g = \alpha\}$$

. This is the set of elements in G which sends α to α itself.

5.1.1 Orbit-Stabilizer Lemma

Is there any connection between the number of permutations in G that fixes α and the number of elements to which α can be taken to? The orbit stabilizer lemma, which is an easy consequences of Lagrange's theorem gives a neat answer.

Lemma 5.2 (**Orbit-Stabilizer Lemma**). *Let $G \leq S_n$. Then for any $\alpha \in [n]$,*

$$|\alpha^G| * |G_\alpha| = |G|$$

Proof. We quickly observe that G_α is a Subgroup of G. Indeed, identity belongs to G_α trivially. If g and g' both fixes α, then so does gg' and $g'g$. If g fixes α, then so does g^{-1}. Since G_α forms a Subgroup of G, by Lagrange's Theorem,

$$\frac{|G|}{|G_\alpha|} = \text{number of distinct right cosets of } G_\alpha \text{ in } G$$

To complete the proof, it suffices to argue that the number if distinct cosets of G_α in G is the size of the orbit of α under the action of G. We now show the bijection.

Let $\beta \in \alpha^G$ via $h \in G$. That is,$\alpha^h = \beta$. Consider the map,

$$\Gamma : \beta \to \{g \in G | \alpha^g = \beta\}$$

We first show that this is well-defined map between the elements in the orbit of α to the cosets. For that, we first show that $\{g \in G | \alpha^g = \beta\}$ is indeed a right coset of G_α in G. To argue this, it suffices to show that

$$\begin{aligned}\Gamma(\beta) = \{g \in G \mid \beta = \alpha^g\} = \{g \in G \mid \alpha^h = \alpha^g\} = \{g \in G \mid \alpha^{gh^{-1}} = \alpha\} \\ = \{g \in G \mid gh^{-1} \in G_\alpha\} = \{g \in G \mid g \in G_\alpha h = G_\alpha h\end{aligned}$$

Thus Γ is a function.

We now show that the function Γ is injective. Let β and γ be two different elements of the orbit of α via the group elements h_β and h_γ. That is,

$$\Gamma(\beta) = \{g \in G \mid \alpha^g = \beta\}$$

$$\Gamma(\gamma) = \{g \in G \mid \alpha^g = \gamma\}$$

Indeed, these two sets cannot have an intersection since $\beta \neq \gamma$. Thus, $\Gamma(\beta) \neq \Gamma(\gamma)$.

We now argue that the function Γ is surjective. Consider any coset $C = G_\alpha g$ of G_α in G, where $g \in G$. We show that there is a $\beta \in \alpha^G$ such that $\Gamma(\beta) = C$. Indeed, define β to be α^g. Consider any $h \in G$ such that $\alpha^h = \beta$.

1: JS says: Surjectivity proof to be completed

□

5.2 Graph Automorphism and Graph Isomorphism

We define the problems that we are interested in, technically.

Problem 5.3 (Graph Isomorphism Problem (GI)). *Given a graph $X_1 = (V_1, E_1)$ and $X_2 = (V_2, E_2)$, decide if $X_1 \cong X_2$ or not.*

Problem 5.4 (Graph Automorphism Problem (GA)). *Given a graph $X = (V, E)$, compute a Generating Set for $Aut(X)$.*

Problem 5.5 (Graph Rigidity Problem (GR)). *Given a graph $X = (V, E)$, decide if $Aut(X)$ is trivial or not.*

Problem 5.6 (Counting Isomorphisms (#GI)). *Given a graph $X_1 = (V_1, E_1), X_2 = (V_2, E_2)$, output the number of Isomorphisms from X_1 to X_2, in binary.*

Problem 5.7 (Counting Automorphisms (#GA)). *Given a graph $X = (V, E)$, compute the size of the automorphism group, $|Aut(X)|$, in binary.*

Problem 5.8 (Computing the Isomorphism (ISO)). *Given a graph $X_1 = (V_1, E_1), X_2 = (V_2, E_2)$, output an isomorphism map between V_1 and V_2 if it exists.*

We use the notion of polynomial time reducibility between two problems. Given two problems A, B, we say that $A \leq B$ (A reduces to B), if given a polynomial time (in terms of input size n) algorithm for B, we can give out a polynomial time algorithm for A. We quickly observe some easy relationship among these problems. Indeed, if we can solve GA, we can solve GR as well. Given a graph X, to check if it is rigid, it is only a matter of checking if there is a nontrivial element in the generating set for the group $Aut(X)$. Hence we conclude that $GR \leq GA$. The same is the case for GR and $\#GA$ as well. That is, $GR \leq \#GA$. Similarly, if we can compute the isomorphism, we can decide it as well. Trivially, $GI \leq ISO$. It is interesting to think about whether $\#GA$ can be done using GA. That is, given the generating set of a permutation group (that is, subgroup of S_n), can we compute the size of the generated group?

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Ameya Panse
Date: August, 11 2015
Status: γ

Lecture
6

Reduction Between Variants of GI

Our aim is to understand the relationship between various problems related to graph isomorphism that we listed in the last lecture. To quote the names, we talked about, graph isomorphism problem (GI), graph automorphism problem (GA), graph rigidity problem (GR), counting isomorphisms (#GI), counting automorphisms (GA), and computing the isomorphism (Iso). As a main tool towards understanding these, we now introduced a colored version of graph isomorphism.

6.1 Colored Graph Isomorphism Problem

The driving thought is the following. Consider the following problem. Given two graphs X_1 and X_2, and consider a vertex $u \in V_1$ and $v \in V_2$. Can we test if there is an isomorphism between X_1 and X_2 that maps u to v itself? To create a terminology, the scenario can also be described as : we will color vertex u and v with a color (say *blue*), the rest of the vertices in both X_1 and X_2 as red, and ask if there is a *color preserving isomorphism* between the graphs.

More formally, let $X = (V, E)$ be a graph. Consider a coloring function function $\Psi : V(X) \to [c]$, where $i \in [c]$ denotes a color. Thus, for a vertex v, $\Psi(v)$ denotes its color. We call the set of vertices $\Psi^{-1}(i)$ as the i^{th} color class, and we call the graph as c-colored.

Problem 6.1 (Colored Graph Isomorphism (CGI))**.** *Given two c-colored graphs (X_1, Ψ_1) and (X_2, Ψ_2) decide if there exists $\sigma : V(X_1) \to V(X_2)$ such that*

- *for all $(u, v) \in V(X_1) \times V(X_2)$, $(u, v) \in E(X_1)$ if and only if $(\sigma(u), \sigma(v)) \in E(X_2)$*
- *for every $u \in V(X_1), \Psi_1(u) = \Psi_2(\sigma(u))$.*

How hard can colored graph isomorphism be? In general can there be an efficient algorithm which solves CGI for any c? Indeed, an easy observation is that GI is a special case when $c = 1$. That is, give all vertices in the graphs the same color so that any isomorphism preserves the colors. Thus, CGI seems harder when number of vertices in a color class is allowed to be large. Indeed, later in the course, when the number of vertices in any color class is bounded by a constant, we will

give a polynomial time algorithm for the problem. However, without any restrictions the problem is as hard as graph isomorphism. However, what about the cases when $c > 1$. at a first sight, they dont seem easier than GI. We start by showing that they are not harder for sure.

6.1.1 Gadget Trick : From Colored GI to Non-colored GI

We show that $CGI \leq GI$. That task is as follows, given (X_1, Ψ_1) and (X_2, Ψ_2) construct graphs X_1' and X_2' (uncolored graphs) such that

$$((X_1, \Psi_1), (X_2, \Psi_2)) \in CGI \iff (X_1', X_2') \in GI$$

The construction of X_1' and X_2' are as follows : For every vertex $u \in V(X_1)$ such that $u \in \Psi_1^{-1}(i)$:

1. Add ni extra vertices to X_1.
2. Add edges from each of the extra vertices to u.

Do the same for (X_2, Ψ_2) to get X_2'. This completes the reduction.

We denote by Y_u the extra vertices that we added for the vertex u. Notice that the degree of all extra vertices added is 1 each, and the degree of all original vertices in color class i ($i \geq 1$) is at least ni. The number of vertices in the new graph produced is at most $n + \sum_{i=1}^{c} ni \leq O(n^2)$. The reduction runs in polynomial time since we can construct the graphs X_1' and X_2' in polynomial time. Thus the only thing remaining is to prove the correctness of the reduction which we do by the following claim.

Claim 6.2. $((X_1, \Psi_1), (X_2, \Psi_2)) \in CGI \iff (X_1', X_2') \in GI$

Proof. ($\Longrightarrow$) Let $((X_1, \Psi_1), (X_2, \Psi_2)) \in CGI$. We need to show that $X_1' \cong X_2'$. From the assumption, we know that there exists $\sigma : V(X_1) \to V(X_2)$ such that :

1. $\forall (u, v) \in V(X_1) \times V(X_2), (u, v) \in E(X_1)$ if and only if $(\sigma(u), \sigma(v)) \in E(X_2)$.
2. $\forall u \in V(X_1), \Psi_1(u) = \Psi_2(\sigma(u))$.

To extend this to an isomorphism σ' between X_1' and X_2', we need to define the images of the extra vertices that we added in the above construction. However, since the σ is color preserving, we know that for any $u \in V(X_1$, $|Y_u| = |Y_{\sigma(u)}|$. Extend σ to σ' by choosing any bijection between the vertices Y_u and $Y_{\sigma(u)}$.

We now argue that σ' is an isomorphism. Observe that, for any $u \in V(X_1)$, the only edges incident on Y_u are of the form (a, u) where $a \in Y_u$. Consider the pair $(\sigma'(a), \sigma'(u)$ which is same as $(a', \sigma(u)$ where $a' \in Y_{\sigma(u)}$. This is an edge in X_2' by construction. The converse of the above implications also hold, and hence $X_1' \cong X_2'$.
($\Longleftarrow$) Suppose $X_1' \cong X_2'$. We need to show that $(X_1, \Psi_1) \cong (X_2, \Psi_2)$. By assumption, there is a bijection $\sigma : V(X_1') \to V(X_2')$ such that $\forall (u, v) \in E(X_1'), (\sigma(u), \sigma(v)) \in E(X_2')$.

To begin with, we argue that σ must map elements of $V(X_1)$ to $V(X_2)$ itself. This is simply because the degrees of the vertices outside $V(X_2)$ are all 1 and the degree of all vertices in $V(X_1)$ in the graph X_1' are all at least n. Since σ is originally an isomorphism, σ restricted to $V(X_1)$ immediately gives an isomorphism between X_1 and X_2.

We now need to argue that the map σ preserves color. Suppose not. Let $u \notin \Psi_2^{-1}(i)$. Hence $\sigma(u) \in \Psi_2^{-1}(j)$ and $j \neq i$. Note that $ni + n > deg(u) \geq ni$. This implies $n(i+1) > deg(u) \geq ni$. And, $\sigma(u) \in \Psi_2^{-1}(j) \Rightarrow n(j+1) > deg(u) \geq nj$. Since both of these can not be true simultaneously, we have a contradiction. Hence, $\sigma(u) \in \Psi_2^{-1}(i)$. Thus, $(X_1, \Psi_1) \cong (X_2, \Psi_2)$. ∎

6.1.2 Computing Isomorphism $\leq$ GraphIso

We show that ISO $\leq CGI$. As $CGI \leq GI$ we have ISO $\leq GI$.

Given X_1, X_2 and oracle access to a blackbox which checks for isomorphism between any two given graphs, output a permutation that maps X_1 to x_2.

The Reduction is as follows:

1. Check if $X_1 \cong X_2$. If NO, end.
2. For each vertex vertex $i_1 \in [n] = V_1$
   Get X_1' by coloring i_1 with color i_1.
   - Get X_2' by coloring $i_2 \in V_2$ with color i_1.
   - Query CGI to check if $X_1' \cong X_2'$.
   - Remove colors from X_2' and repeat the last two steps for $i_2 \in V_2$ until we get a "yes" answer. For the i_2 on which we get a "yes" answer, fix the color as i_1 and do not reuse color i_1 again in the outer loop.
3. Ouput the permutation.

To see the correctness : suppose the graphs are isomorphic. That is the algorithm reaches step (2), 3rd subpart is guaranteed to find a j for each vertex i. Observing that we will color a vertex on the left and the right with some color i_1 on if we are sure that there is a color preserving isomorphism. Hence the isomorphism map, if exists, will be found. To see the running time, observe that we make atmost $O(n^2)$ queries to CGI which in turn makes a single query each to GI. Thus we can solve ISO using at most $O(n^2)$ queries to GI.

6.1.3 GraphIso $\leq$ GraphAuto

We need to construct a graph G such that the generating set S of $Aut(G)$ enables to decide if $X_1 \equiv X_2$.

Assume that the graphs are connected to begin with. The reduction is simply to take $X = X_1 \cup X_2$. Let S be the generating set of the reduction, check if there exists a $\sigma \in S$ such that σ maps atleast one vertex in X_1 to a vertex in X_2. We directly show the correctness of the reduction.

Claim 6.3. *$X_1 \cong X_2$ if and only if there exists a $\sigma \in S$ and $v \in V(X_1)$ such that $\sigma(v) \in V(X_2)$.*

Proof. ($\Rightarrow$) Suppose $X_1 \cong X_2$. Then there exists a τ which is an isomorphism from X_1, to X_2. $\tau \in Aut(X)$. Hence, there is a σ that maps a vertex in X_1 to a vertex in X_2.

($\Leftarrow$) Let there exist a $\sigma \in S$ such that σ maps $u \in X_1$ to a vertex in X_2. Let $v \in X_1$ be such that $\sigma(u) \in X_1$. Since X_1 is connected u, v are connected. But $\sigma(u) \in X_2, \sigma(v) \in X_1$ are not connected which is a contradiction. Therefore for $\sigma \in S$, σ maps all the vertices in X_1 to X_2. Thus $X_1 \cong X_2$. □

What do we do if the graphs are not connected? We simply add an extra vertex each to both the graphs that is adjacent to all the vertices in the corresponding graphs. Since the new vertices have degree n, they can only be mapped to each other by any isomorphism (or an automorphism of X). Thus GI $\leq$ GA.

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Sahil Sharma
Date: August, 12 2015
Status: δ

Lecture 7

Reduction of $\mathcal{GA}$ to $\mathcal{GI}$

7.1 Recap and Lecture overview

Let us denote by $\mathcal{COLOR} - \mathcal{GI}$ the problem of finding whether there is a coloring preserving isomorphism. In the previous few lectures we showed that $\mathcal{COLOR} - \mathcal{GI} \leq \mathcal{GI}$. We also showed that $\mathcal{GI} \leq \mathcal{GA}$. We also showed how to compute an isomorphism map between two graphs (COMPUTE-ISO) if we can check if two graphs are isomorphic. Recall that $Aut(X)$ is the group of all automorphisms of a given graph X and $\mathcal{GA}$ is the problem of obtaining a generator for $Aut(X)$. In this lecture, we show that $\mathcal{GA} \leq \mathcal{GI}$. That is, given a subroutine to solve $\mathcal{GI}$, we show how we can compute a generator set of $Aut(X)$ for an input graph X. Along with $\mathcal{GI} \leq \mathcal{GA}$ proved in the earlier class, this shows that the graph isomorphism and graph automorphism problems are equivalent in terms of hardness.

The main idea is that there is a unique way to express an element in G using Tower of subgroups of G.

7.2 Solving Graph Automorphism using Graph Isomorphism

7.2.1 Tower of Subgroups of Group

Definition 7.1 (Tower of Subgroups of G)**.** *Let $k \in \mathbb{N}$. For a group G, the k subgroups of G namely $G^{(1)}, G^{(2)}, \ldots, G^{(k)} = \{id\}$ is said to form a tower of subgroups of G if*

$$G = G^{(0)} \geq G^{(1)} \geq, \ldots, G^{(k)} = \{id\}$$

The above concept is defined for an arbitrary group, but we will use this specifically for understanding about Tower of subgroups of $Aut(X)$, for a given graph X.

Note that in the sequence, $G^{(i)}$ is a subgroup of $G^{(i-1)}$ for $1 \leq i \leq k$. Hence there is a coset structure that $G^{(i)}$ generates (or induces) in $G^{(i-1)}$ and we seek to exploit this structure to get a $O(n \log n)$ sized generating set of $Aut(X)$ efficiently, given a procedure for solving $\mathcal{GI}$.

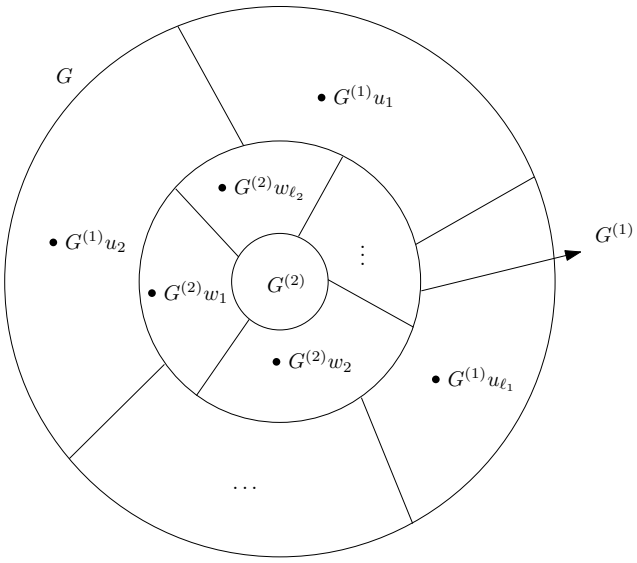

FIGURE 7.1: Tower of groups of G for two levels with cosets in the first level fixed as $u_1, u_2, \ldots, u_{\ell_1}$ and second level fixed as $w_1, w_2, \ldots, w_{\ell_2}$.

Definition 7.2. *For a group G and a subgroup H of G denote $H : G$ to denote the set of cosets of H in G and the number of cosets in $H : G$, denoted, $[H : G]$ is called as the index of H in G.*

For example, for $1 \leq i \leq k$, $[G^{(i+1)} : G^{(i)}]$ denotes the number of cosets of induced in $G^{(i)}$ by $G^{(i+1)}$. Denote this number by ℓ_i. Note that by Lagrange's theorem $\ell_i = \frac{|G^{(i)}|}{|G^{(i+1)}|}$. Hence $\prod_i \ell_i = |G^{(0)}| = |G|$.

Recall the notion of a *coset representative* of a coset, with respect a given group and a subgroup of the given group. For example, in the figure 7.1, consider the subgroup $G^{(1)}$ of G. Here we choose u_1 as the coset representative of the coset $G^{(1)}u_1$.

Note that any arbitrarily chosen element of that coset $G^{(1)}u_1$ can be made its representative due to the following reason. For any two elements in the same coset, say g and g' we have $G^{(1)}g = G^{(1)}g'$, by the definition of the coset (since it generates the same coset)[3]. Hence, there is nothing special about u_1, g, g' and any elements of the coset can be chosen as its representative.

7.2.2 Unique representation of a group in terms of coset representatives

Given this setup, we are now ready to give a unique way of representing group elements once we fix a tower of subgroups for the group and a coset representatives at each level.

Consider an element $g \in G^{(i-1)}$ for some $i \geq 1$ and $G^{(i+1)}$ be the subgroup in the next level in the tower of subgroups. Since the cosets of $G^{(i+1)}$ partitions $G^{(i)}$ it must be that g must lie in exactly one coset.

[3]One way to show this is as follows : let $g, g' \in G^{(1)}u_1$ where u_1 is a coset representative. Hence $g = h_1u_1$, $g' = h_2u_2$ for $h_1, h_2 \in G^{(1)}$. For any $w \in G^{(1)}g$, $w = hg$ for some $h \in G^{(1)}$. Using the fact that $g = h_1u_1$, we get $w = hh_1u_1$. Hence $w \in G^{(1)}u_1$. Hence $G^{(1)}g \subseteq G^{(1)}u_1$. A similar argument shows that $G^{(1)}u_1 \subseteq G^{(1)}g$. This also shows that $G^{(1)}u_1 = G^{(1)}g'$. Hence the cosets formed by g and g' are the same as the original coset.

Observation 7.3. *Consider the groups $G^{(i)}$ and $G^{(i-1)}$ for some $i \geq 1$. Let $g \in G^{(i-1)}$ lie in the coset whose representative is u_1. Then there exists a unique $h_i \in G^{(i)}$ such that*

$$g = h_i u_1$$

Proof. By definition, $g \in G^{(i)}u_1 \Rightarrow \exists h_i \in G^{(i)}$ such that $g = h_i.u_1$. Such a h_i is unique since if there is another $h'_i \in G^{(i)}$ such that $h_i u_1 = h'_i u_1$, then $h_i = g.u_1^{-1} = h'_i$. □

This gives us a way to uniquely represent the elements of G.

Theorem 7.4. *Suppose we fix the tower of subgroups of G denoted $G = G^{(0)} \geq G^{(1)} \geq, \ldots, G^{(k)} = \{id\}$ as well as the coset representatives for $G^{(i+1)} : G^{(i)}$ for all $0 \leq i \leq k-1$ in the tower of subgroups. Then, each element of the group G can be represented as a unique product of coset representatives.*

Proof. Let $g \in G = G^{(0)}$. Let ℓ_1 be the number of cosets in $G^{(1)} : G^{(0)}$. Let $u_1, u_2, \ldots, u_{\ell_1}$ be the coset representatives. By the above observation 7.3 (setting $i = 1$), we know that there exists a unique $h_1 \in G^{(1)}$ such that $g = h_1 u_1$. Now consider the cosets $G^{(2)} : G^{(1)}$. Since they partition $G^{(1)}$, we ask, in which coset does h_1 belong to ? By the application of the same claim to h_1 and the pair of groups (G_1, G_2) we get a unique h_2 such that $h_1 = h_2 u_2$, where u_2 is the coset in which h_1 belongs. Hence g can be expressed as $h_2 u_2 u_1$.

Continuing in this way, as we go down the tower of subgroups while fixing the coset representations, in each stage we get a unique h_i and u_i whose product gives h_{i-1} and we end up with a unique representation for the element g. Note that this representation is unique since at each stage the h_i were found in a unique way, and the coset representatives are fixed. □

Note that this also gives us a way to solve $\#\mathcal{GA}$. If we can count the number of coset representatives at each level (which is ℓ_i by our notation), then $\prod_i \ell_i = |G|$.

7.2.3 Finding a generating set for $Aut(X)$

To find a generating set for $Aut(X)$ it suffices to find a tower of sub-groups of $Aut(X)$ and the coset representatives at each level.

Applying Theorem 7.4 for $Aut(X)$, we can represent each element of $Aut(X)$ uniquely as a product (which in our case is composition operation) of a sequence of coset representatives once we define a suitable tower of subgroups for $Aut(X)$ and compute the coset representatives efficiently. Hence it suffices to output these representatives to obtain a generating set.

We define the tower of subgroups in such a way that computing the coset representatives becomes efficient (using $\mathcal{GI}$). Denote $G^{(0)}$ as $Aut(X)$. The subgroups are defined as, for $0 \leq i \leq n-1$,

$$G^{(i+1)} := \{g \in G^{(i)} \mid i^g = i\}$$

That is, $G^{(i)}$ is the sub-group of automorphisms which maps all the nodes of the graph in the range $\{1, \ldots, i\}$ to themselves.

We can check that $G^{(i)}$ is indeed a group since the composition of two permutations which maps all the nodes of the graph in the range $\{1, \ldots, i\}$ to themselves also does the same. Moreover, the identity permutation is the identity for this sub-group too, and the inverse permutation is defined

in the standard way and satisfy the property of inverse element of a group. Hence, this is indeed a subgroup. By the definition of $G^{(i)}$ it can be verified that the subgroups defined indeed forms a tower of subgroups of $Aut(X)$.

Now the task is reduced to finding the coset representatives of $G^{(i+1)}$ in $G^{(i)}$.

Claim 7.5. *Let the number of cosets generated by $G^{(i+1)}$ in $G^{(i)}$ be ℓ_i. Then*

$$\ell_i \leq n - i$$

Proof. Consider $g \in G^{(i)}$. Note that g maps the element $i+1$ to an element in the range $\{i+1, \ldots, n\}$. This is because the other elements are already fixed. Let $k = (i+1)^g$ be the element to which $i+1$ is mapped and r be the element such that $r^g = i+1$. Then consider the permutation g' which retains other mappings from g but changes the above two mappings to $(i+1)^{g'} = i+1$ and $r^{g'} = k$. Note that g' is also an automorphism since r being mapped to $i+1$ implies that if r is mapped to the image of i the automorphism would still be preserved (adjacency and non-adjacency is preserved). Also note that g' maps $\{1, \ldots, i+1\}$ and hence is in G^{i+1}. This means that whatever is the number of automorphisms in G^i cannot exceed the number of automorphisms in G^{i+1} multiplied by $n-i$ since the $i+1$ can potentially map to only $n-i$ elements. □

This claims shows that at each level the number of representatives that we need to output is at most $n-i$ and hence the size of generating set collected over all levels is at most $\sum_{i=1}^{n}(n-i) = O(n^2)$. The algorithm for finding the coset representatives is the following. We explain how to get the representatives for a level i.

Look for automorphisms in $G^{(i)}$ which map all of $\{1, \ldots, i\}$ to themselves and map $i+1$ to each element in the set $\{i+1, \ldots, n\}$, one by one and ask the question: is there an automorphism which preserves these mappings. This gives us the cosets since the only freedom in the coset representatives is in where $(i+1)$ maps to.

We answer this using $\mathcal{COLOR} - \mathcal{GI}$ problem. Since we already have a solution to $\mathcal{GI}$, as demonstrated in the previous lectures, we can use this to also solve $\mathcal{COLOR} - \mathcal{GI}$. Now, this can done by using $\mathcal{COLOR} - \mathcal{GI}$ in the following way.

1. Make two copies of the input graph X and call it X_1 and X_2.
2. Colour the vertex $v_k \in \{v_1, \ldots, v_i\}$ by with the color k for both the graphs X_1 and X_2.
3. Colour the vertex v_{i+1} in X_1 and X_2 with the same colour $i+1$ and use a new colour $i+2$ and colour the rest of the vertices in both the graphs with the colour $i+2$.

In this way, we can cast the problem as a $\mathcal{COLOR} - \mathcal{GI}$ and solve it using $\mathcal{GI}$. Each time we get an answer as yes, we then use the reduction $\mathcal{COLOR} - \mathcal{GI} \leq \mathcal{GI}$ to find the actual permutation. This permutation is one of the coset representatives and hence in this manner we get each coset representative. Note that for each search for coset representative we make only polynomially many calls to $\mathcal{GI}$ and the number of such representatives is upper bounded by n^2 and hence our reduction is polynomial time and computes a generating set of $Aut(X)$ in polynomial time, given that we can solve $\mathcal{GI}$ in polynomial time.

Note that this procedure can me modified not only to compute an automorphism, but can also be used to compute $|Aut(X)|$. The reason is that the ℓ_i which corresponds to number of coset representatives in $G^{(i+1)} : G^{(i)}$ can be exactly computed in our setting. Our final task

is to obtain a permutation in $G^{(i)}$ that sends $i+1$ to $\{i+1,\ldots,n\}$. The set of permutations which satisfy this is nothing but the orbit of $i+1$ in $G^{(i)}$. Hence by Orbit stabilizer theorem, $|G^{(i)}| = \left|(i+1)^{G^{(i)}}\right| \ldots |G^{(i+1)}|$. This gives that $\left|(i+1)^{G^{(i)}}\right| = \frac{|G^{(i)}|}{|G^{(i+1)}|} = \ell_i$. This tells that to obtain ℓ_i it suffices to estimate the size of the orbit of $i+1$.

The generating set that we obtained by fixing a tower of subgroups and coset representatives have many nice properties. We will see more about them in subsequent lectures.

Definition 7.6 (Strong Generating Set)**.** *A generating set for $Aut(X)$ obtained in the manner above making use of coset representatives and tower of sub-groups is known as a strong generating set.*

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Sahil Sharma
Date: August, 14 2015
Status: δ

Lecture
8

Some Group-theoretic problems in Permutation Groups

8.1 Recap

In the previous lecture we showed that $\mathcal{GA} \leq \mathcal{GI}$. We also defined certain generic group theoretic concepts like *tower of sub-groups*, *coset representatives* and the notion of *strong generating sets*. Also note that the reduction $\#\mathcal{GA} \leq \mathcal{GI}$ can be done using the set of reductions $\#\mathcal{GA} \leq \mathcal{GA} \leq \mathcal{GI}$ where the first reduction follows from the fact that the generating set of $Aut(X)$ was in fact a strong generating set and hence each element of G could be obtained uniquely by composing elements of the generating set. This would imply that the size of the automorphism group is $\prod_i \ell_i$ where ℓ_i is the number of cosets in the i^{th} level of the tower of subgroups of G.

Our aim is to cast the graph theoretic problems in group theoretic terms and solve the problem using the machinery of group theory. Towards this aim, we abstract the various problems and questions which were encountered while doing the previous reductions and give a set of four related group-theoretic problems.

8.2 Some Computational Questions in Group theory

Following are two natural question to ask.

Problem 8.1 (**Order Computation**). *Given a group G via its generating set, can we compute the order of the group, that is, the number of elements in the group.*

Problem 8.2 (**Membership Testing**). *Given an element $g \in G$ and an $H \leq G$ expressed via a generating set S (i,e. $\langle S \rangle = H$), test if $g \in H$ or not.*

Note that problem 2 can be solved using problem 1. This is because we could just add g to the generating set S of H and use problem 1 to obtain the sizes of the groups generated by S and

$S \cup \{g\}$. If the sizes are unequal we conclude that g was not in H. This is because if g were in H, then the generating set S could have generated g as well and adding it to S could not have resulted in any new elements.

Here is a recursive strategy to solve Problem 1 (Order computation). The orbit stabilizer lemma that for any $\alpha \in G$,

$$|\alpha^G| \cdot |G_\alpha| = |G|$$

. Suppose we can estimate $|\alpha^G|$ which is the size of the orbit, then to compute $|G|$, we are left with the smaller recursive subproblem of estimating the size of $|G_\alpha|$.

To implement this strategy, we need to solve the two subproblems. (1) finding the size of the orbit and (2) finding the generating set of G_α. Moreover, in last lecture, we also wanted to count the number of permutations in $G^{(i)}$ which first i elements identically and maps the $i+1^{th}$ element to any of the $n-i$ elements. We saw that this is actually a question of obtaining the size of orbit. This motivates the following question.

Problem 8.3 (Orbit Computation). *Given a group $G \leq S_n$ (via its generating set S), compute the orbits of the action of G on $[n]$.*

Diversion: The following is a slight diversion to address a question raised in class. So far we were interested in groups which are subgroups of S_n. But how do we answer the same questions for general abstract groups? We show that an abstract group can nevertheless be viewed as a subgroup of a permutation group.

Claim 8.4. *A group G acts on itself.*

Proof. Consider an element $g \in G$. Consider any arbitrary ordering of the elements in G. Then the multiplication of G by g, $G.g$ sends the elements of G to a permutation of themselves[4]. Hence, with each element $g \in G$, we can associate a permutation of the elements of G in the following way : if $G = \{g_1, g_2, \ldots, g_k\}$ then for a $g_i \in G$, corresponding $\sigma_i \in S_k$ is defined as the one which satisfy $(g_i g_1, g_i g_2, \ldots, g_i g_n) = (g_{\sigma_i(1)}, g_{\sigma_i(2)}, \ldots, g_{\sigma_i(k)})$

Collect all the permutations associated with the group elements and call it H. Note that the resulting set of permutations is forms a group since multiplication in original group translates to composition in the new group[5] and the identity element in the original group is associated with the identity permutation and so on.

Hence if the order of the group is k, then the resulting group of permutations is a sub-group of S_k. This defines the notion of a group acting on itself. □

Note 8.5. *Hence from now on, we will assume that G acts on an arbitrary set Ω which we can think $[n]$.*

End of diversion.

[4]The reason is that the resulting operation acts bijectively on the elements. That is if $g_1 \neq g_2$ then $gg_1 \neq gg_2$. Also for any $h = g_i x$, there is a unique solution $x = hg_i^{-1}$ in G

[5] The statement amounts to showing that for any $g_1, g_2 \in G$ with $\sigma_1, \sigma_2 \in H$ as the corresponding elements defined by the map, $g_1 g_2 \in G \iff \sigma_1 \circ \sigma_2 \in H$. This is because,

$$\begin{aligned} g_1 g_2 (g_1, g_2, \ldots, g_k) =& (g_1 g_{\sigma_2(1)}, g_1 g_{\sigma_2(2)}, \ldots, g_1 g_{\sigma_2(k)}) \\ =& (g_{\sigma_1(\sigma_2(1))}, g_{\sigma_1(\sigma_2(2))}, \ldots, g_{\sigma_1(\sigma_2(k))}) \end{aligned}$$

8.3 Set Stabilizer Problem

We know that $Aut(X)$ acts on $[n]$. This can also be visualized as $Aut(X)$ acting on the set of potential edges in the graph X. What does it do to the actual edges in this action? Indeed, the automorphisms map edges to edges and non-edges to non-edges. Hence, any edge of the graph gets maped to another edge. To be more precise, we can also think of S_n as acting on the set of all potential edges given by $K = \{\{i, j\} | i, j \in [n] \text{ and } i \leq j\}$. In other words, the action of $Aut(X)$ on the set $E(X)$ does not changes $E(X)$ or it *fixes* $E(X)$. This leads us to the fourth problem called the *Set Stabilizer problem.*

Definition 8.6 (Set Stabilizer of Σ)**.** *Given a G and a subset $\Sigma \subseteq \Omega$,*

$$\mathsf{SETSTAB}(\Sigma) = \{g \in G \mid \Sigma^g = \Sigma\}$$

where $\Sigma^g = \{\alpha^g \mid \alpha \in \Sigma\}$

Given a group $G \leq S_n$ which acts on a set Ω, we define the Set Stabilizer of $\Sigma \subseteq \Omega$ as the set of all permutations in G which map elements of Σ to elements of Σ and all non elements of Σ to non-elements of Σ. Note that $\mathsf{SETSTAB}(\Sigma)$ is a sub-group of G. In the case of automorphism groups, the mapping is $G = S_n$, $\Omega = K$, $\Sigma = E(X)$ and $\mathsf{SETSTAB}(\Sigma)$=$Aut(X)$.

Having set up all the notation, let us state the computational question we seek to answer.

Problem 8.7 (**Set Stabilizer Problem**)**.** *Given a group G via its generating set S, a set Ω on which it acts and a subset $\Sigma \subseteq \Omega$, output a generating set for the group* $\mathsf{SETSTAB}(\Sigma)$.

By our previous discussion on $Aut(X)$ fixing the edges of X, we can conclude that $Aut(X)$ is the stabilizer of $E(X)$. Hence the problem $\mathcal{GA}$ reduces to the problem $\mathsf{SETSTAB}$.

8.4 Orbit Computation

For the sake of completeness, let us restate the question.

Orbit Computation : Given a group $G \leq S_n$ (via its generating set S), compute the orbits of the action of G on $[n]$.

We can see that the orbits of the action of G on $[n]$ partition the set into different subsets. Hence, we would want to compute the partitions. We cast this problem as a graph theoretic problem.

Let $\Omega = [n]$. We construct a directed graph X with $V(X) = \Omega$ and $E(X) = \{(\alpha, \beta) \mid \exists g \in G, \beta = \alpha^g\} \subseteq V(X) \times V(X)$. We can have the edge $(\alpha, \beta) \in E(X)$ labelled by $g \in G$ if $\alpha^g = \beta$. To check if two elements α, β lie in the same partition, it suffices to check if there is a directed path from α to β (or from β to α). Hence the connected components of the graph corresponds to the orbits. In the next lecture, we will give another algorithm for solving this problem.

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Aditi Raghunathan
Date: August, 17 2015
Status: δ

Lecture
9

Set Stabilizers and Point-wise Stabilisers

9.1 Recap and Exercise

In the previous class, we were looking at ways to solve $\mathcal{AUT}(\mathcal{X})$. We looked at 4 different problems in this context. Consider *Problem 3* of finding the orbit. We are interested in computing the size of the orbit of $\alpha \in G$. Since some of the students expressed difficulty in coming up with an algorithm, we decide to state the algorithm and leave the correctness proofs as an exercise.

9.1.1 Orbit Computation

The following is an algorithm to calculate the orbit of α

Algorithm 1 Algorithm for Orbit Computation

```
1: procedure ORBIT COMPUTATION( Input : Generating set S )
2:     Δ = {α}
3:     repeat
4:         Δ^1 = Δ
5:         for g ∈ S and δ ∈ Δ do
6:             Δ = Δ^1 ∪ {δ^g}
7:     until Δ^1 ≠ Δ
```

The algorithm was developed as step by step in the lecture which also developed its correctness. However, the formal proofs are left as an exercise. To be precise, it is left as an exercise to the reader to do the following : (1) Prove that Algorithm 1 eventually terminates. (2) Prove that Algorithm 1 terminates with $\Delta = \text{Orbit}(\alpha)$ (3) Calculate the running time of Algorithm 1.

The problem can be viewed in terms of the graph. Consider the following definition of a directed graph X. $V(X)$, the vertex set of the graph G is equal to the set Ω. $E(X) = \{(\alpha, \beta) \mid \exists g \in S \text{ such that } \alpha^g = \beta\}$

The key observation is that, if there is a path in X from α to γ then γ is in the orbit of α. Finding the orbit of α is equivalent to computing the transitive closure of the graph X. It is left as an exercise to the reader to complete the details of the above algorithm including rigorous proof of correctness.

9.2 Set Stabilisers Problem

We recall the problem that we defined in the last lecture. Given group $G \leq S_n$ and a set $\Sigma \subseteq \Omega$, where G acts on Ω and $|\Omega| = n$, we are interested in computing the generating set of the following group :

$$\mathcal{SETSTAB}(\Sigma) = \{g \in G \mid \Sigma^g = \Sigma\} \text{ where} \tag{9.4}$$

$$\Sigma^g = \{\alpha^g \mid \alpha \in \Sigma\} \tag{9.5}$$

As we saw in the last lecture, solving the set stabiliser problem gives an algorithm for obtaining $\mathcal{AUT}(X)$.

We now define a variant of the problem called the *Point Stabiliser Problem*

Point-wise Stabiliser Problem

$$\mathcal{POINTSTAB}(\Sigma) = \{g \in G \mid \forall \alpha \in \Sigma, \alpha^g = \alpha\} \tag{9.6}$$

$\mathcal{POINTSTAB}$ is *easier* to solve than $\mathcal{SETSTAB}$ The Point Stabilizer Problem is to obtain a generating set for the group $\mathcal{POINTSTAB}(\Sigma)$. It is easy to check that $\mathcal{POINTSTAB}(\Sigma)$ is indeed a group.

In the reduction from $\mathcal{GA}$ to $\mathcal{GI}$, we defined a tower of sub-groups where

$$G^{(i)} = \{g \in G \mid \forall 1 \leq j \leq i j^g = j\} \tag{9.7}$$

Observe that each tower here is a point-wise stabiliser of $\{1, 2, \cdots i\}$ For each $G^{(i)}$, we have $\Omega = \{1, 2, \cdots n\}$ and $\Sigma = \{1, 2, \cdots i\}$

Here is a general strategy for solving point-wise stabilizer problem. Without loss of generality, as above, assume that Σ is the first i elements of Ω. The problem to solve is precisely the following, given the generators S of a group G, find the generators of $G^{(i)}$. A natural attack on the problem is stepwise, given the generating set of G, find that of $G^{(1)}$, and then using that to find the generating set of $G^{(2)}$ and so on. In i levels of this recursion, we will be done.

The key problem that we identified for solving the order computation as well as point-stabilizer problem is the following. Given the generating set of $G^{(i-1)}$, find that of $G^{(i)}$. This is exactly what we will do in the next section.

9.2.1 Schreier's Lemma

Schrier gave a clever way of completing our task. If in addition to the generating set of G^{i-1}, we are also given the coset representatives of G^i as a subgroup in $G^{(i-1)}$.

In the following, we will call G to be the bigger group ($G^{(i-1)}$) and H to be the subgroup $G^{(i-1)}$. Let R be set of right coset representatives of the subgroup H in G. The following lemma gives a direct way of writing the generating set for H.

Lemma 9.1 (**Schreier's Lemma**). *Let S be the generating set of G and R be the set of coset representatives of H. $S' = \{r_1 g r_2^{-1} \mid r_1, r_2 \in R, g \in S\}$ $S' \cap H$ forms a generating set for H.*

Before we prove this, notice that R contains the identity of the groups. Hence the Set $S \subseteq S'$. By taking $S' \cap H$, we are extracting the elements in S' which are also in H. In order to do this, we do need a membership testing method for H. In our context, H is $G^{(i)}$ and has an easy membership test given an element $g \in G^{(i-1)}$.

Proof. Define,

$$RS = \{rs \mid r \in R, s \in S\}$$

Since each element in RS can be written as $(rsr_1^{-1})r_1$, we immediately get that,

$$RS \subseteq S'R \tag{9.8}$$

Let $\langle S' \rangle$ denotes the group generated by S'. By definition, $S'R \subseteq \langle S' \rangle R$. From 9.8, we have $RS \subseteq \langle S' \rangle R$ Therefore, $RSS \subseteq \langle S' \rangle RS$, and hence $RSS \subseteq \langle S' \rangle RS$. Repeating this process, we have $\forall t \geq 0, RS^t \subseteq \langle S' \rangle R$

Since R contains the identity, and S generates G, $G \subseteq \bigcup_{i=1}^{k} RS^t$ for some fixed finite k. In fact, we will see later that $k \leq |G|$, but we do not need this bound here.

We argue that $H \leq \langle S' \rangle$. Sinec $G \subseteq \langle S' \rangle R$, if $\langle S' \rangle$ does not contain H, $\langle S' \rangle R$ cannot cover G Hence proved by contradiction. Therefore, $S' \cap H$ generates H.

2: JS says: To be completed, why should it exactly generate H?

□

The following is a different view of the above proof. This was not done in the lecture, but we record this for completeness of this notes and for future use. We derive an equivalent statement to the above, by observing that for every r_1 and a, there is a unique r_2 such that $r_1 a r_2^{-1} \in H$. Call this r_2 as $\overline{r_1 a}$. Hence the following is an equivalent restatement.

Lemma 9.2 (**Schreier's Lemma restated**). *Let H be a subgroup of G, S be the generating set of G and R be the set of right coset representatives (containing identity), then,*

$$T = \left\{ rs(\overline{rs})^{-1} \mid r \in R, s \in S \right\}$$

is a generating set for H, where $\overline{g}$ denotes the coset representative corresponding to the element g (that is, the unique element in $Hg \cap R$).

Proof. First observe that the elements of T are in H (that is the way we redefined it to begin with). Hence, it cannot generate a larger group. It is enough to show that $T \cup T^{-1}$ generates all the elements of H.

Consider an arbitrary element $h \in H$. Since $H \leq G$, it can be written as the product of elements from $S \cup S^{-1}$. Let $h = s_1 s_2 \dots s_k$. We will incrementally keep modifying this product representations from left to right to obtain a product representation of h using elements of $T \cup T^{-1}$. We call this modification sequence to be $h_1, h_2 \dots h_k$ where each $h_i = h$. In general,

$$h_i = t_1 t_2 t_i r_{i+1} s_{i+1} s_{i+2} \dots s_k$$

where $t_i \in T \cup T^{-1}$ and $r_{i+1} \in R$.

The initial element $h_0 = \prod_{i=1}^{k} s_i$ with $r_1 = 1$, which is exactly the original representation, and the final element $h_k = (\prod_{i=1}^{k} t_i) r_{k+1}$. Since $h_k = h \in H$, it must be that r_{k+1} is the identity. This gives the representation for h in terms of $T \cup T^{-1}$.

Now we complete the proof by defining h_{i+1} from h_i. We just need to define t_{i+1} and r_{i+2}. Take, $t_{i+1} = r_{i+1} s_{i+1} \overline{r_{i+1} s_{i+1}}^{-1}$ and $r_{i+2} = \overline{r_{i+1} s_{i+1}}$. Clearly, the product h_{i+1} has the required form, $t_{i+1} \in T \cup T^{-1}$ and $r_{i+2} \in R$. □

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: Aditi Raghunathan
Date: August, 18 2015
Status: β

Lecture 10

Using Schreier's Lemma - Reduce Algorithm

Let G be a group generated by a set S. In the first part of this lecture, we went after a question from the previous lecture, about the value of k such that $G \subseteq \cup_{t=0}^{k} S^t$. This is an important question to be answered for the following variant of the membership problem. Given a group $G \leq S_n$ via the generating set S, and a $g \in S_n$, we asked the problem of testing whether $g \in G$. In the previous versions, we expected a yes or no answer to this question. However, it is also useful to ask the computational version of this problem. That is, if $g \in G$, then give a representation of g as the product of elements in S. However, for this, the first question to ask is a bound on the length of any such representation.

10.1 Bounds on length of generating sequence

We start with the following easy bound.

Proposition 10.1. $k \leq |G|$

Proof. We prove the statement by contradiction.

Let the smallest length of the generating sequence of some element $g_1 \in G$ be $|G| + m$, $m > 0$.

$$g_1 = \prod_{i=1}^{|G|+m} a_i \tag{10.9}$$

Consider partial products of the sequence above :

$$S_l = \prod_{i=1}^{l} a_i \tag{10.10}$$

There are $|G| + m$ partial products. There are only G possible values for S_i By Pigeon Hole Principle, we have the following :

$$\exists l_1, l_2, \quad l_1 < l_2 \quad S_{l_1} = S_{l_2} \tag{10.11}$$

This means, we can remove the elements of the generating sequence between l_1 and l_2, which will generate the same element g_1 (From Equation 10.11). Hence we get a shorter generating sequence for g_1 which contradicts the original assumption. Thus, $k \leq |G|$ □

The above bound is too weak for our purpose because, in our context, $|G|$ could be very large. Can we improve this bound in general? Unfortunately we end up proving that the bound is tight.

Proposition 10.2. *There is an $n \in N$, group $G \leq S_n$ and a generating set S of G, and a $g \in G$ such that the shortest product representation of sequence is of length $|G|$.*

Proof. Let $p_1, p_2, \cdots p_m$ be the first m distinct prime numbers. Choose $n = p_1 p_2 \cdots p_m$. We will define a $G \leq S_n$. The G that we define will be a cyclic group generated by the following element a.

$$a = (1, 2, \cdots p_1)(p_1 + 1, p_1 + 2 \cdots p_2) \cdots (\cdots) \tag{10.12}$$

Let M be the order of the element a. That is, M is the smallest M such that $a^M = id$. First observation is that, $M = \prod_{i=1}^{m} p_i$, which is the lcm of the length of the above cycles. It is easy to see that the element a^{M-1} cannot have a shorter representation than as the product of a, $M - 1$ times. Notice that $M \geq 2^m$.

From the prime number theorem, $m \geq \Omega(\frac{p_m}{\log p_m})$. Also notice that $n = \sum_{i=1}^{m} p_i \leq 1+2+3 \cdots + p_m \leq p_m^2$. Thus, $p_m \geq \sqrt{n}$.

Hence, $m \geq \Omega(\frac{\sqrt{n}}{\log \sqrt{n}})$. Thus, $M \geq 2^{\frac{c\sqrt{n}}{\log n}} - 1$. □

That is negative news. However, the above example does not rule out the existence of a polynomial length expression or computation for the element. For example, a^{M-1} can be expressed in terms of a through repeated squaring.

But then, how do we formulate our membership problem? This is where the strong generating strikes again. If we are allowed to change the generating set S to and S' (precomputation before seeing the g) we can always compute a generating set such that we can output a short representation.

10.2 Need of a Reduction Step

We now return to the algorithm to compute the generators of a sub group. Using Schreier's lemma, we concluded that it's sufficient to compute the set $S' \cap H$.

Before getting into the algorithm for the same, let us first look at the size of the set that is generated.

By the definition of the set S', we can only say

$$|S'| \leq |S||R|^2 \tag{10.13}$$

If we have to implement this recursively, we see that each level we are incurring a $|R|^2$ term. Hence

$$|S'| \leq |S| l_1^2 \cdot l_2^2 \cdots \tag{10.14}$$

We need to carefully control the size of the generating set S' at each level.

10.3 REDUCE algorithm

How do we reduce the generating set to control its size? Consider $\pi, \psi \in S'$ such that $1^\pi = 1^\psi = k$. Do we need to keep both of them? Yes, indeed, it may be that $2^\pi = k'$ and $2^\psi = k''$. We cannot afford to throw any of them away.

However, here is an observation. Consider replacing *psi* with $\pi^{-1}\psi$.

We can observe the following :

1. We can still obtain all the elements ψ can be obtained $\pi(\pi^{-1}\psi)$. Observe that we also do not add any new elements in the process as well. Hence by this change, the subgroup that is generated remains the same.

2. Doing the above process of replacement for every pair of elements that map 1 to the same element, we end up with elements that all map 1 to different elements (except those that fix 1).

3. $\pi^{-1}\psi$ fixes 1. Hence all the newly added elements fix 1.

4. We can repeat the same procedure for $2, 3, \ldots n$.

The idea seems neat. At the end of the above procedure, for every pair (i, j), we have exactly one π such that $i^\pi = j$. This immediately gives an upper bound of $O(n^2)$ in the size of the sets.

However, there is a catch. When we repeat the same procedure for $2, 3, \ldots n$, what is the guarantee that the property for 1 will not be violated?

The ideas is that once we fix the elements which are mapping 1 to k where $k \neq 1$, all the extra elements which are mapping now 1 to 1 are pushed to the stage 2. In stage 2, the operations of the form $\pi^{-1}\psi$ are done on elements which are stabilizing 1. Hence they will continue to stabilize 1. In the end, we may get several trivial elements, which we can simply discard.

To clarify this, we write this entire algorithm as a procedure.

Algorithm 2 REDUCE Algorithm. Input : Generating set S, Output: Reduced Generating Set

```
1: B_0 = B
2: A[ ][ ], an empty n × n array
3: for i = 0 to n − 1 do
4:     for all ψ ∈ B_i do
5:         j = i^ψ
6:         if A[i][j] is empty then
7:             if j = i then
8:                 B_{i+1} = B_{i+1} ∪ {ψ}
9:             else
10:                A[i][j] = ψ
11:        else
12:            π = A[i][j]
13:            B_{i+1} = B_{i+1} ∪ {ψ^{-1}π}
14: discard all trivial elements from ∪B_i
15: return ∪B_i
```

The correctness proof for the algorithm is left as an exercise.