Lecture Notes on

Algorithmic Algebra

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

Draft—August 18, 2015 and forever

Todo list

List of Scribes

Table of Contents

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.

CS6842 – Algorithmic Algebra

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

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

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 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.

[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 $\binom{N}{i} \neq 0 \mod N$

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 :

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

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. There are two naive ways to do it. One is to evaluate it at all the numbers from 1 to N or to expand out the polynomial and see if it is identically zero. But both operations are costly.

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.

CS6842 – Algorithmic Algebra

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

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

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 $G_1 \cong G_2$ (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 $G_1 \cong G_2$ 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 S_3 is not cyclic.

1: JS says: reviewed this lecture's notes until here

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

Having observed that $Aut(X)$ could have potentially be of exponenetial size, it is natural to look for generating sets of small size.

Given a graph $X = (V, E)$ where $|V| = n$, does $Aut(X)$ have a generating set of size $\mathsf{poly}(n)$?

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 $g = h_1 g_1$. Similarly $g \in Hg_2$ suggests that there exists a $h_2 \in H$ such that $g = h_2 g_2$.

$$h_1 g_1 = h_2 g_2 = g$$

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

$$g_1 = h_1^{-1} h_2 g_2 \tag{4.2}$$

$$g_2 = h_2^{-1} h_1 g_1 \tag{4.3}$$

(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,

$$\begin{aligned} g' &= h' g_1 \\ g' &= h'(h_1^{-1} h_2 g_2) \qquad \text{[By equation (4.2)]} \end{aligned}$$

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,

$$\begin{aligned} g' &= h' g_2 \\ g' &= h'(h_2^{-1} h_1 g_1) \qquad \text{[By equation (4.3)]} \end{aligned}$$

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 $|Hg| \leq |H|$.

Let us show : $|H| \leq |Hg|$.Suppose not, $|Hg| < |H|$. Then there exists $h_1, h_2 \in H, h_1 \neq h_2$ such that $h_1 g = h_2 g$. Since $(G, *)$ is a group, g^{-1} exists. We have $h_1 g g^{-1} = h_2 g g^{-1}$ 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 $\log |S_n| = log(n!) \in \mathcal{O}(n \log n)$ by Stirling's approximation.*

CS6842 – Algorithmic Algebra

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

LECTURE 5

Orbit Stabilizer Lemma

In this lecture we will understand and prove the Orbit Stabilizer Lemma, while defining the various terms associated with it.

Definition 5.1. *(Order of a Group) Order of a group G is number of elements in the group, that is* $\|G\|$

Definition 5.2. *(Right Co-set) Let* $H \leq G$, $g \in G$ *Right co-set of* H *in* G *is defined as*

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

Definition 5.3. *(Left Co-set) Let* $H \leq G$, $g \in G$ *Right co-set of* H *in* G *is defined as*

$$gH = \{gh | h \in H\}$$

Note 5.4. *In general, it is not necessary that the left and right co-sets are the same.*

Definition 5.5. *(Normal SubGroup) Let* $H \leq G$, *if we have*

$$\forall g \in G, Hg = gH$$

then H *is called a* Normal SubGroup

Let H be a Normal SubGroup of G. Divide G into co-sets of H and take one element from each of these as a representative of the set.

Claim 5.6. *These elements have a group structure among them.*

This will be proved in later classes.

5.1 Group Action and Orbits

Let G be a SubGroup of S_n. Let $\alpha \in [n]$ and $g \in G$.

Note 5.7. α^g *is the image of* α *in the permutation* g.

Definition 5.8. *(Orbit)*

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

5.1.1 An Equivalence Realtion

Consider the following relation,

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

Claim 5.9. *The above realtion is an Equivalence Relation.*

Reflexive: $e \in G$, where e is the identity element. Hence, $\alpha \backsim \alpha$
Symmetric: Let $\alpha \backsim \beta$. Thus $\exists g \in G, \alpha^g = \beta$. Hence, $\alpha = \beta^{g^{-1}}$. Thus, $\beta \backsim \alpha$
Transitive: Let $\alpha \backsim \beta, \beta \backsim \gamma$. Thus $\exists g_1, g_2 \alpha^{g_1} = \beta, \beta^{g_2} = \gamma$. By composition of permnutations, $(\alpha^{g_1})^{g_2} = \gamma$. Hence, $\alpha \backsim \gamma$

Definition 5.10. *(Stabilizer of α)*

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

Observation 5.11. *G_α is a SubGroup of G.*

5.1.2 Orbit Stabilizer Lemma

$$\forall \alpha \in [n] \| \alpha^G \| * \| G_\alpha \| = \| G \|$$

Claim 5.12. *There exists a bijection from α^G to Co-sets of G_α in G.*

Corollary 5.13. *Number of Co-sets of $G_\alpha = \| \alpha^G \|$.*

And, since G_α forms a SubGroup, applying Lagrange's Theorem, we get the Orbit Stabilizer Lemma.

Proof of Claim: The idea is that the set of permnutations that send α to β, form a Co-set of permutations that send α to α.

Let $\beta \in \alpha^G$ and $h \in G, \alpha^h = \beta$.

$$\begin{aligned}
\text{Consider} \{g \in G | \beta = \alpha^g\} \text{ We have to show that this is a Co-set} \\
&= \{g \in G | \alpha^h = \alpha^g\} \\
&= \{g \in G | \alpha^{gh^{-1}} = \alpha\} \\
\text{Thus } gh^{-1} \in G_\alpha \\
&= \{g \in G | gh_{-1} \in G_\alpha\} \\
&= \{g \in G | g \in G_\alpha h\}
\end{aligned}$$

Exercise 5.14. *Complete the above proof by showing a bijection between β's and the Co-sets of G_α.*

5.2 Graph Automorphism and Graph Isomorphism

We will now define and analyse various problems related to GI.
Graph Isomorphism [GI]: Given graphs G_1, G_2,
Output 1
if $\exists \sigma : V_1 \to V_2, \forall (u,v) \in E_1, (\sigma(u), \sigma(v)) \in V_2$
else Output 0.

Definition 5.15 (Automorphism)**.** *A Graph G is said to be automorphic if it is non-trivially isomorphic to itself.*

Definition 5.16 ($Aut(G)$)**.**

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

PROBLEM : GRAPH AUTOMORPHISM
Input : A graph $G = (V, E)$
Output : A Generating Set for $Aut(G)$.

PROBLEM : GRAPH RIGIDITY
Input : A graph $G = (V, E)$
Output : Decide if $Aut(G)$ is trivial or not.

PROBLEM : NUMBER OF ISOMORPHISMS
Input : A graph $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$
Output : The number of Isomorphisms from G_1 to G_2.

PROBLEM : NUMBER OF AUTOMORPHISMS
Input : A graph $G = (V, E)$
Output : $\|Aut(G)\|$.

PROBLEM : COMPUTING ISOMORPHISM
Input : A graph $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$
Output : A permutation that 'morphs' G_1 to G_2.

PROBLEM : COMPUTING AUTOMORPHISM
Input : A graph $G = (V, E)$
Output : A non-trivial element of $Aut(G)$.

5.3 Informal Reduction

Given two problems A, B, we say that $A \leq B$ (A reduces to B), if given a polytime algorithm for B, we can give out a polytimeytime algorithm for A.

In the next lecture we will talk about the relation among the above defined problems.

CS6842 – Algorithmic Algebra

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

LECTURE
6

A Closer Look at Graph Isomorphism and Automorphism

6.1 Another related Problem

Here the vertex set is divided into c color classes by the function

$$\Psi : V(X) \to [c],$$
where $i \in [c]$ denotes a color and
the i^{th} color class is $\Psi^{-1}(i)$.

Colored Graph Isomorphism [CGI]: Given two C-colored Graphs (X_1, Ψ_1) and (X_2, Ψ_2)
Output 1
if $\exists \sigma : V(X_1) \to V(X_2)$ such that $\forall (u, v) \in V(X_1) \times V(X_2), (u, v) \in E(X_1)$ if and only if $(\sigma(u), \sigma(v)) \in E(X_2)$
and $\forall u \in V(X_1), \Psi_1(u) = \Psi_2(\sigma(u))$.

6.2 Relations Among the Problems

6.2.1 GI $\leq$ CGI

Set $c = 1$, and color all the vertices with the same color.

6.2.2 CGI $\leq$ GI

Given (X_1, Ψ_1) and (X_2, Ψ_2).
[Gadget] : $\forall u \in V(X_1)$ 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 to get the graph X_1'.

Do the same for (X_2, Ψ_2) to get X_2'.
Now run GI on X_1', X_2'.

Correctness :

Forward Direction:

Let (X_1, Ψ_1) and $(X_2, \Psi_2) \in CGI$. To show that $X_1' \cong X_2'$.

Hence $\exists \sigma : V(X_1) \to V(X_2)$ such that $\forall (u,v) \in V(X_1) \times V(X_2), (u,v) \in E(X_1)$ if and only if $(\sigma(u), \sigma(v)) \in E(X_2)$ and $\forall u \in V(X_1), \Psi_1(u) = \Psi_2(\sigma(u))$. Additionally map the extra vertices added correspondingly. Thus $X_1' \cong X_2'$.

Backward Direction:

Let $X_1' \cong X_2'$. To show that $(X_1, \Psi_1) \cong (X_2, \Psi_2)$.
$X_1' \cong X_2'$, hence $\exists \sigma : V(X_1') \to V(X_2'), \forall (u,v) \in E(X_1'), (\sigma(u), \sigma(v)) \in E(X_2')$.
If possible let there exist $u \in v(X_1), u \in \Psi^{-1}(i)$ such that $\sigma(u) \notin V(X_2)$. That is, u is mapped to one of the extra vertices. But $u \in X_1'$ and $deg(u) \geq ni$, whereas degree of any extra vertex is 1. Hence, we have a contradiction.

Now, if possible let $u \notin \Psi_2^{-1}(i)$. Hence $\sigma(u) \in \Psi_2^{-1}(j)$ and $j \neq i$. Note that $ni + n > deg(u) \geq ni \Rightarrow n(i+1) > deg(u) \geq ni$.

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, $\sigma(u) \in V(X_2), \sigma(u) \in \Psi_2^{-1}(i)$. Thus, $(X_1, \Psi_1) \cong (X_2, \Psi_2)$.

Time Complexity : We have made only one query to GI. Hence, the reduction is polytime.

Hence Proved.

6.2.3 Computing Isomorphism $\leq GI$

Given X_1, X_2 Output a permutation that morphs 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 $i \in V_1$ Color i with color C_i.
   - Color $j \in V_2 -$ [Already Colored Vertices] with C_i, temporarily.
   - Query to CGI.
   - Repeat on j untill you get a yes answer. Fix color of j as C_i.

   Ouput the permutation.

Here each vertex is colored with a different color, and hence we have a permutation.
Also, if the graphs are isomorphic, then there will exists an $j \in V_2$, such that we get a yes answer.

Time Complexity: We make atmost $O(n^2)$ queries to CGI which in turn makes a single query to GI.
Thus the reduction is polytime.

6.2.4 GI $\leq$ GA

Take $X = X_1 \cup X_2$.
Let S be the generating set of $Aut(G)$.

Claim 6.1. *$X_1 \cong X_2$ if and only if $\exists \sigma \in S$ such that σ maps atleast one vertex in X_1 to a vertex in X_2.*

For the time being assume that the two graphs are connected graphs.
Forward Direction Assume that $X_1 \cong X_2$.
$\exists \tau$ which is an isomorphism between X_1, X_2. $\tau \in Aut(X)$. Hence, there is a σ that maps a vertex

in X_1 to a vertex in X_2.
Backward Direction : $\exists \sigma$ that maps $u \in X_1$ to $\sigma(u) \in X_2$. 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. Hence, we have a contradiction. Thus, σ maps all vertices in X_1 to X_2. Thus $X_1 \cong X_2$.

In case the two are not connected, add an extra vertex to both the graphs that is adjeacent to all the vertices in the correcponding graphs. Since, the new vertices hace degree n, they can be mapped only to each other.
Hence Proved.

CS6842 – Algorithmic Algebra

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

LECTURE 7

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

7.1 Recap

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 $\mathcal{GI} \leq \mathcal{GA}$. It was also proved that $\mathcal{COMPUTE} - \mathcal{GI} \leq \mathcal{GI}$. In this lecture, we show that $\mathcal{GA} \leq \mathcal{GI}$. This shows that the graph isomorphism and graph automorphism problems are equivalent.

7.2 Towers of Sub-Groups of G

The discussion in this section sets up the notation and concepts related to towers of sub-groups of any given group G. The concept can be applied to any group, but we will use this specifically in the case of $\mathcal{AUT}(X)$, for a given graph X. $\mathcal{AUT}(X)$ is defined to be the group of all automorphisms of a given graph X.
Given a group G, we define tower of sub-groups of G to be a sequence of groups $\{G^0, \cdots, G^k\}$ such that $G^i \leq G^{i-1}$ for all i and $G^k = \{e\}$ is the group which contains only the identity element. That is, G^i is a sub-group of G^{i-1}. Hence the sequence of sub-groups looks like $G^k = \{e\} \leq G^1 \leq \cdots \leq G = G^0$.

There is a coset structure that G^i generates in G^{i-1} and we seek to exploit this structure to get a $O(n \log n)$ sized generating set of $\mathcal{AUT}(X)$ efficiently, given a $\mathcal{GI}$ solving procedure.
The number of cosets of G^i in G^{i-1} is called the *index* of G^i in G^{i-1}. There is a notion of a *coset representative* of a coset, with respect a given group and it's sub-group. This is just any arbitrarily chosen element of that coset. Suppose the sub-group involved is G^i, then for any two elements in the same coset, say g and g' we have $G^i.g = G^i.g'$, by the definition of the coset (since it can generate the coset). Hence, any element can be chosen as the representative of the coset.

7.3 Unique representation of a group in terms of coset representatives

Claim 7.1. *Consider an element $g \in G^{i-1}$. Also, let the coset in which it lies be C_1 and the representative for that coset be u_1. $\exists$ a unique $h_i \in G^i$ such that*

$$g = h_i.u_1$$

.

Proof. $g \in C_1 \Rightarrow \exists h_i \in G^i$ such that $g = h_i.u_1$.
Such a h_i is unique since $h_i = g.u_1^{-1} = h_i'$. □

Theorem 7.2. *Suppose we fix the tower of sub-groups of G as well as the co-set representatives for each consequetive pair of {group, sub-group} in the tower of sub-groups. We claim that each element of the original group G can be represented as a product of some elements of this set of coset representatives.*

Proof. By the above claim, we know that such an h_1 exists for any element $g \in G$. Now we ask, which coset does h_1 belong to, in the set of cosets generated by G_2 in G_1. 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 $h_1 = h_2.u_2$, depending on the coset to which h_1 belongs. Continuing in this way, as we go down the tower of subgroups we end up with a unique representation for the element we started off with, that is, g. Note that the representation is unique since at each stage the h_i were found in a unique way, and since the coset-representatives are held fixed. □

7.4 Finding a generating set efficiently, for $\mathcal{AUT}(X)$

To find a generating set for $\mathcal{AUT}(X)$ it suffices to find a tower of sub-groups of $\mathcal{AUT}(X)$ and the coset representatives at each level. This is because according to the above theorem, we can represent each element of $\mathcal{AUT}(X)$ uniquely as a product (composition) of a sequence off coset representatives.
Let us define the tower of sub-groups in such a way that computing the coset-representatives becomes efficient (using $\mathcal{GI}$).

The sub-groups are defined as :

$$G^{i+1} := \{g \in G^i | 1^g = 1\}$$

That is, G^i is the sub-group of automorphisms which maps all the nodes of the graph in the range $\{1, \cdots, i\}$ 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 $\{1, \cdots, i\}$ 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 has the same properties as the inverse permutation in $\mathcal{AUT}(X)$. Hence, this is indeed a sub-group.

Let the number of cosets generated by G^{i+1} in G^i be l_i.

Claim 7.3.

$$l_i \leq n - i$$

Proof. Consider an automorphism $g \in G^i$. This maps the element $i+1$ to some element in the range $\{i+1, \cdots, n\}$. This is because the other elements are already fixed. Let $k = (i+1)^g$ be the element to which $i+1$ is mapped and l be the element such that $l^g = i+1$. Then consider the permutation g' which retains other mappings from g but changes the above two mappings to $(i+1)^{g'} = i+1$ and $l^{g'} = k$. Note that g' is also an automorphism since l being mapped to $i+1$ implies that if l is mapped to the image of i the automorphism would still be preserved (adjacency and non-adjacency is preserved). Also note that g' maps $\{1, \cdots, i+1\}$ 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. □

The algorithm for finding the coset representatives is now simple. For all i, look for automorphisms which map all of $\{1, \cdots, i\}$ to themselves and map $i+1$ to each element in the set $\{i+1, \cdots, n\}$, one by one and ask the question: Is there an automorphism which preserves these mappings. This is done by introducing an instance of the $\mathcal{COLOR} - \mathcal{GI}$ and solving it using $\mathcal{GI}$ as demonstrated in the previous lectures. Note that this can be done since the elements from $\{1, \cdots, i\}$ can be colored with the color which is the number itself, in both the graphs X_1 and X_2. $i+1$ in set X_1 can be colored with the color $i+1$ and j from X_2 is also colored with $i+1$. Rest of the elements are colored with a new color $i+2$. 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 $\mathcal{COMPUTE} - \mathcal{GI} \leq \mathcal{GI}$ 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 $\mathcal{AUT}(X)$ in polynomial time, given that we can solve $\mathcal{GI}$ in polynomial time.

Definition 7.4. *Strong Generating Set: A generating set for $\mathcal{AUT}(X)$ 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

LECTURE
8

Generalization of group-theoretic problems

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 $\mathcal{AUT}(X)$ 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 $\prod_i l_i$ where l_i was defined in the last lecture.

Abstracting the sub-problems which were encountered in doing the previous reduction, we get a set of 4 related group-theoretic problems listed below.

8.2 Problem 1

Given a generating set for G, can we compute the order of the group, that is, the number of elements in the group.

One of the things we will try to do, to solve this problem is to convert the given generating set into a strong generating set, since we already know how to solve the problem once that is done.

8.3 Problem 2

Given a group G and it's sub-group H through respective generating sets, we want to check whether an arbitrary $g \in G$ belongs to H.

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 compare 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.

8.4 Problem 3

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

We show a different way of viewing a group wherein it will be clear that any generic group can be looked at as a permutation group. Consider an element $g \in G$. Consider any arbitrary ordering of the elements in G. Then the multiplication $G.g$ sends the elements of G to a permutation of themselves. Hence, with each element $g \in G$, we can associate a permutation of the elements of G. If the order of the group, $|G| = k$, then the resulting group of permutations would be a sub-group of S_k. This defines the notion of a group acting on itself. Note that it is clear that the resulting set of permutations is a group since multiplication in original group translates to composition in the new group and the identity element in the original group is associated with the identity permutation and so on.

We know, due to the orbit stabilizer lemma that for any $\alpha \in G$

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

This equation alongwith Problem 3 can be used for solving Problem 1 since if we can copute the orbit size of α then we can recurse for the sub-group G_α.

8.5 Problem 4

We know that $\mathcal{AUT}(X)$ acts on $[n]$. This can also be visualized as $\mathcal{AUT}(X)$ acting on the set of edges $E(X)$. This is because the automorphisms map edges to edges and non-edges to non-edges. Hence, any edge of the graph gets maped to another edge. Hence we can think of $\mathcal{AUT}(X)$ as acting on the set of edges.

Extending this idea, we can also think of S_n which is a super-group of $\mathcal{AUT}(X)$ as acting on the set of all potential edges given by $K = \{\{i, j\} | i, j \in [n] \text{ and } i \leq j\}$.

Definition 8.1. *Set Stabilizer of Σ:*

$$\mathcal{SETSTAB}(\Sigma) := \{g \in G | \Sigma^g = \Sigma\}$$

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 $\mathcal{SETSTAB}(\Sigma)$ is a sub-group of G. In the case of automorphism groups, the mapping is $G = S_n$, $\Omega = K$, $\Sigma = E(X)$ and $\mathcal{SETSTAB}(\Sigma)$=$\mathcal{AUT}(X)$.

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

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 $\mathcal{SETSTAB}(\Sigma)$.

8.6 Solution to Problem 3

For completeness's sake, 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 sub-sets. Hence, what we would want to do is to compute the partition. We try to cast this problem as a graph-theoretic problem. The vertices are the numbers in the set $[n]$. If $j \in [n]$ lies in the orbit of $i \in [n]$, then there is an edge from i to j. We know due to the manner in which orbits are constructed and the fact that they induce partitions that the final graph looks like a set of cliques. Hence, we can try to compute the graph partitially (by finding some edges) and make use of transitive closures for finding the rest of the edges in the graph until we reach a fixed point, at which point we can say that we have computed the partition and hence the orbits.

CS6842 – Algorithmic Algebra

Instructor: Jayalal Sarma
Scribe: *Student name*
Date: *Month, XX 2015*

LECTURE
42

Title of Lecture

Note 42.1. *Enter date, lecture number, title and your name.*

2: JS says: This is a sample comment

CS6842 – Algorithmic Algebra

Instructor: jayalal sarma
Scribe: *student name*
Date: *month, xx 2015*

LECTURE
43

title of lecture

43.1 Add details of next lecture