Algebra and Computation	Course Instructor: V. Arvind
Lecture 2	
Lecturer: V. Arvind	*Scribe: Ramprasad Saptharishi*

1 Motivation

Last lecture we had a brush-up of group theory to set up the arsenal required to study Graph Isomorphism. This lecture we shall see how group theory motivates graph isomorphism, and some more theorems on group theory that we will require for later lectures.

2 Graph Isomorphism and Automorphism Groups

Recall that two graphs G_1 and G_2 are isomorphic if there is a re-numbering of vertices of one graph to get the other, or in other words, there is an automorphism of one graph that sends it to the other.

And clearly, $Aut(G) \leq S_n$, the symmetric group on n objects, which represent the permutation group on the vertices. And since it is a subgroup of the permutation group, $|Aut(G)| \leq n!$

Of course, providing the entire automorphism group as output would take exponential time but what about a small generating set? Which then leads us to, does there exist a small generating set?

Lemma 1. *For any group H of size n, there exists a generating set of size $\log n$.*

Proof. Let $H_0 = \{e\}$. If $H_0 = H$ we are done. Otherwise, let $x = H \setminus H_0$ and $H_1 = \langle H_0, x\rangle$. In general, let $x \in H \setminus H_i$ and $H = \langle H_i, x\rangle$. Since x forms at least two distinct cosets of H_i, $|H_{i+1}| \geq 2|H_i|$. And hence, in at most $\log n$ steps we will hit H. □

Now we can ask the question: can we output a small generating set of the automorphism group of a graph G? We shall refer to this problem as Graph-Aut. We shall now show that Graph-Iso and Graph-Aut are polynomial time equivalent.

Theorem 2. *With* Graph-Iso *as an oracle, there is a polynomial time algorithm for* Graph-Aut *and vice-versa.*

Proof. First we shall show that we can solve Graph-Iso with Graph-Aut as an oracle. We are given two graphs G_1 and G_2 and we need to create a graph G using the two such that the generating set of the automorphism group should tell us if they are isomorphic or not.

Let $G = G_1 \cup G_2$. Suppose additionally we knew that G_1 and G_2 are connected, then a single oracle query would be sufficient: if any of the generators of $Aut(G)$ interchanged a vertex in G_1 with one in G_2, then connnectivity should force $G_1 \cong G_2$.

But what if they are not connected? We then have this very neat trick: $G_1 \cong G_2 \iff \overline{G}_1 \cong \overline{G}_2$. As either G_1 or $\overline{G}_1$ has to be connected, one can check for connectivity and then ask the appropriate query.

The other direction is a bit more involved. The idea is to see that any group is a union of cosets. Suppose

$$H = a_1K \cup a_2K \cup \cdots a_nK$$

then $\{a_1, a_2, \ldots, a_n\}$ along with a generating set for K form a generating set for H. Hence once we have a subgroup K with small index, we can then recurse on K.

Hence we are looking for a tower of subgroups

$$Aut(G) = H \geq H_1 \geq H_2 \geq \cdots \geq H_m = \{e\}$$

such that $[H_i : H_{i+1}]$ is polynomially bounded.

For our graph G, let $Aut(G) = H \leq S_n$. We shall use Weilandt's notation where i^π denotes the image of i under π. In this notation, composition becomes simpler: $(i^\pi)^\tau = i^{\pi\cdot\tau}$.

Define $H_i = \{\pi \in H : 1^\pi = 1, 2^\pi = 2, \cdots i^\pi = i\}$. And this gives the tower

$$H_0 = H \geq H_1 \geq H_2 \geq \cdots \geq H_{n-1} = \{e\}$$

with the additional property that $[H_i : H_{i+1}] \leq n - i$ since there are at most $n - i$ places that $i + 1$ can go to when the first i are fixed.

We need to find to find the coset representatives.

Look at the tableau

Picture supposed to come here, needs to be completed

As H is $Aut(G)$, we can find the coset representatives using queries to the Graph-Iso subroutine: to find a representative for $\left[H^{(i)} : H^{(i+1)}\right]$, make two copies of G, force the first i vertices to be fixed (by putting identical

gadgets on them in each copy), and for each place j' that $i+1$ might go to, force $i+1$ to go to j', test if a graph isomorphism exists, and continue till an isomorphism is found.

□

3 The Set Stabilizer Problem

The Problem Statement: $H \leq S_n$, given by a small generating set. Also given is a subset $\Delta \subseteq [n]$. Find the *stabilizer of* Δ *in* H, defined as

$$stab_\Delta(H) = \{\pi \in H : \Delta^\pi = \Delta\}$$

Though this problem has nothing to do with graphs directly, graph isomorphism reduces to this problem (which we shall call Set-Stab).

Theorem 3.

$$\text{Graph-Iso} \leq_P \text{Set-Stab}$$

Proof. By our earlier theorem, it is enough to show that Graph-Aut reduces to Set-Stab.

One simply needs to note that an automorphism can be thought of as acting on the edges as well. Given a graph $G = (V, E)$, a permutation of the vertices induces a permutation of the edges. Hence,

$$\phi : Sym(V) \to Sym(\binom{V}{2})$$

is injective.

Thus, all we need to do is find the set of elements in $Sym(V)$ that stabilizes E. Taking the set H to be $\phi(Sym(V)) \subseteq Sym(\binom{V}{2})$ and $\Delta = E \subseteq \left[\binom{n}{2}\right]$, the automorphism group is precisely $stab_\Delta(H)$. □

4 More group theory: Sylow theorems

We will need some more tools for the lectures that follow, the Sylow theorems in particular. Before that, we need the Orbit-Stabilizer theorem.

Definition 4. *Let G act on a set S. Let $s \in S$*

- *The orbit of s (s^G), is the set of all possible images of s under the action of G.*

$$s^G = \{t \in S : \exists g \in G, gs = t\}$$

- *The stabilizer of s (G_s) is the set of all elements of G that fix s.*

$$G_s = \{g \in G : gs = s\}$$

Theorem 5 (Orbit-Stabilizer theorem)**.** *For any finite group G that acts on a set S, for every $s \in S$,*

$$|G| = |s^G| \cdot |G_s|$$

Proof. This is just Lagrange's theorem; all we need to see is that the stabilizer G_s is a subgroup of G and that $[G : G_s] = |s^G|$. □

Theorem 6 (Sylow theorems)**.** *Let G be a group, $|G| = p^m r$, where p is a prime and $gcd(r, p) = 1$. Then*

1. *there exists a subgroup P such that $|P| = p^m$ (p-Sylow subgroup)*
2. *for any p-subgroup[1] H of G, one of its conjugates is contained in P*
3. *the number of p-sylow subgroups of G is $1 \pmod p$*

Proof. We shall prove the subdivisions one after another.

Subdivision 1:

Let Ω be the set of subsets of G of size p^m. Note that $|\Omega| = \binom{p^m r}{p^m}$. Lucas' theorem tells us that $\binom{p^m r}{p^m}$ is not divisible by p by our choice of r. Let G act on Ω by left multiplication.

The action decomposes Ω into orbits, and since $p \nmid |\Omega|$, there exists $A \in \Omega$ such that $p \nmid |A^G|$. And since $p^m r|G| = |A^G||G_A|$, and by the choice of A, $p^m \mid |G^A|$.

And since the elements $ga \in A$ for $a \in A$ are distinct under the action of G_A, it follows that $|A| \geq |G_A|$ and hence forcing $|G_A| = p^m$. Hence G_A is our desired p-sylow subgroup.

Subdivision 2:

Let Ω be the set of left cosets of P, our p-sylow subgroup. And let H be a p-subgroup of G, which induces an action on Ω by left multiplication.

Since $|H| = p^a$, every non-trivial orbit of Ω is has cardinality a multiple of p. Hence, the number of points of Ω that are fixed by H is modulo p is the same as $|\Omega|$. Hence in particular, since $p \nmid |\Omega|$, the set of points fixed by H is non-zero. Hence, there exists a gP that is fixed by H.

[1] subgroup of order p^a for some a

Hence $hgP \subseteq gP$ or $g^{-1}hgP \subseteq P \implies g^{-1}hg \in P$ for all $h \in H$. Thus $g^{-1}Hg \subseteq P$

Note that this also tells us that all p-sylow subgroups are conjugates of each other.

Subdivision 3*:*

Let P be a p-sylow subgroup and let Ω be the set of p-sylow subgroups of G on which P acts by conjugation. For any $Q \in \Omega$, the stabilizer of Q under conjugation is called the normalizer of Q, denoted by $N_G(Q)$.

Suppose $Q \in \Omega$ is fixed by P on conjugation, then $P \leq N_G(Q)$. But subdivision 2 tells us that P and Q are conjugate to each other in $N_G(Q)$, which then forces $P = Q$. Hence the only fixed point is P itself and hence $|\Omega| = 1 \pmod{p}$. □

We also did the proof of Lucas' Theorem, has to be TEX-ed out

Algebra and Computation Course Instructor: V. Arvind

Lecture 3: Divide-and-Conquer on Groups

Lecturer: V. Arvind *Scribe: Ramprasad Saptharishi*

5 Overview

Last lecture we studied automorphism groups motivated through graph isomorphism. This lecture we shall examine other techniques to study group theoretic properties; we shall implement a 'divide-and-conquer' approach to study groups.

6 Orbit Computation

Computing the orbit of an element is one of the basic questions in group-theoretic algorithms.

We are given a group G that acts on a finite set Ω, and hence G can be thought of as a subgroup of $Sym(\Omega)$. And of course since G could be very large, a small generating set A of G is given as input. Given an element $\alpha \in \Omega$, we would like to compute the G-orbit of α, denoted by α^G. Recall that

$$\alpha^G = \{\beta \in \Omega \ : \ \exists g \in G, \beta = \alpha^g\}$$

The naive way is to try and 'reach' every element in the orbit using elements of A acting on α, which is as follows:

```
1: Δ = {α}
2: while Δ grows do
3:    for each a ∈ A and each δ ∈ Δ do
4:       Δ = Δ ∪ {δ^a}
5:    end for
6: end while
```

The running time is at least quadratic in $|\Omega|$. A reachability approach is much better.

Algorithm: Define a graph $X = (V, E)$ where $V = \Omega$ and $(\alpha, \beta) \in E$ if $\alpha^a = \beta$ for some a in $A \cup A^{-1}$. The connected components of this graph correspond to the orbits.

This algorithm takes time $n\,|A|$ where $n = |\Omega|$. Note that this algorithm in fact gives more: given any two elements of the same orbit, one can also obtain a group element that takes one to another by looking at the path from one to another and multiplying the edge labels.

This gives us a step forward towards a divide-and-conquer approach. Once we have the orbit decomposition of Ω, we can study the action of the group on each orbit separately. We would then have the additional property of the action of G being transitive[2] on the set.

7 Decomposition of Transitive Groups

In order to break down transitive groups we shall study "blocks". In this section, we shall assume that the action of G is transitive.

Definition 7. *$\Delta \subseteq \Omega$ is called a block if for all $g \in G$, either $\Delta^g = \Delta$ or $\Delta^g \cap \Delta = \phi$*

Of course for any action, Ω and singletons of Ω are blocks, and are called the *trivial blocks*.

Definition 8. *G is said to be primitive if it has only trivial blocks.*

Let us look at some examples.

1. When G is in fact the entire $Sym(\Omega)$, it's easily seen that G is primitive.

2. Even when G is the set of even permutations over Ω, denoted by $A_{|\Omega|}$, it has enough permutations to still remain primitive. This also can be easily checked.

3. Suppose G acts on itself, say by left multiplication, $G \leq Sym(G)$. Every subgroup of G is a block since the cosets are either identical or disjoint. In fact, any coset is also a block.

   Thus, we have a tiling of Ω using such blocks. We shall refer to these as a *block system*.

   Definition 9. *A block system is a partition of Ω such that each part in the partition is a block with respect to the action.*

[2] any element of the set can be pushed to any other by some element of the group

Note that if Δ is a block, so is Δ^g for every $g \in G$. And the tiling gives a theorem very similar to Lagrange's theorem on subgroups of a group.[3]

Theorem 10. *Let G act transitively on a set Ω and let $\Delta \subseteq \Omega$ be any block. Then $|\Delta|$ divides $|\Omega|$*

And this immediately leads to the following corollary.

Corollary 11. *For any group G that acts on Ω transitively, if $|\Omega|$ is a prime, then G is primitive.*

4. Let X be the graph, a collection of some k triangles and look at its automorphism group acting on it. First note that this action is transitive, and further, each triangle would be a block.

   Thus the automorphism group is imprimitive.

   This can of course be extended to any collection of k identical graphs such that the automorphism group of the piece is transitive.

5. Look at the leaves of a complete binary tree of depth k, and let the group be the automorphism group acting on them.

   What are the blocks?

   Take any internal node in the tree, and look at the set of all descendant leaves of it, and this set of leaves form a block. In fact, all blocks are precisely such sets of leaves.

The last example in fact gives a great motivation to a divide-and-conquer approach.

If G is transitive and imprimitive, let Δ be the smallest block. Now group elements of Ω corresponding to the block-system generated by Δ. Now notice that G in fact acts on this block system since G moves the blocks in the system around. And let the block system be the new set Ω_1 and the group being the projected version of G and we can now ask the question "Is G' primitive/transitive ?" with respect to the smaller set Ω.

Checking if the action is transitive can be done by orbit computation, but we need to check if a group is primitive.

[3] note that you require the action to be transitive, otherwise such Δ^g blocks needn't cover all of Ω

8 Blocks and Subgroups

Observation: If Δ_1 and Δ_2 are G-blocks, then so is $\Delta_1 \cap \Delta_2$.

With this observation, we can now talk about the smallest block containing a bunch of elements of Ω.

Lemma 12. *$G \leq Sym(\Omega)$ acting transitively on Ω, is primitive if and only if G_α is a maximal subgroup of G.*

Proof. Note that α needn't be specified since G_α and G_β are conjugates of each other when G is transitive. It is easy to check that of $g \in G$ such that $g\alpha = \beta$, then $gG_\alpha g^{-1} = G_\beta$.

Suppose $\{\alpha\} < \Delta < \Omega$, a non-trivial block. Let $H = \{g \in G \ : \ \Delta^g = \Delta\}$. We will now show that $G_\alpha < H < G$, thus proving one direction of the lemma.

Clearly, since G acts transitively and $\Delta < \Omega$, H has to be a proper subgroup of G. Also, if $g \in G_\alpha$, then $\alpha \in \Delta \cap \Delta^g \neq \phi$. Since Δ is a block, this forces $\Delta = \Delta^g$ and thus $g \in H$. Since $\{\alpha\} < \Delta$ there exists a $\beta \in \Delta$ different from α. Let g be the element of the group that takes α to β. Then since $\beta \in \Delta \cap \Delta^g$, $g \in H$ but $g \notin G_\alpha$. Thus $G_\alpha < H$.

As for the other direction, let $G_\alpha < H < G$. We shall show that $\alpha^H = \Delta$ is our non-trivial block. Since $G_\alpha < H$, $\Delta \neq \{\alpha\}$. Showing $\Delta < \Omega$ is a bit more involved. Since G_α is a subgroup of G, G_α and its cosets partition G:

$$G = \bigcup_{\beta \in \Omega \ , \ g_\beta : \alpha \mapsto \beta} G_\alpha g_\beta$$

And note that if any $g_\beta \in H$, the the entire coset of $G_\alpha g_\beta$ is contained in H. Hence since $\Delta = \Omega$ would imply $H = G$, our assumption $H < G$ forces $\Delta < \Omega$.

All that's left to show is that Δ is a block. Suppose $\Delta^g \cap \Delta \neq \phi$, then for some $h, h' \in H$, $\alpha^{hg} = \alpha^{h'}$ which then forces $h'gh^{-1} \in G_\alpha < H$. Hence $g \in H$ and therefore $\Delta^g = \Delta$. □

What the above lemma established is a one-to-one correspondence between subgroups of G and blocks of Ω.

One could also think of G as acting on $\{G_\alpha g \ : \ \alpha \in \Omega\}$, by identifying each point $\alpha \in \Omega$ by the subgroup G_α and its cosets.

Lemma 13. *Let $N \lhd G$, a normal transitive subgroup of G. Then the orbits of N form a block system.*

Proof. We want to show that α^N is a block. Suppose $\alpha^{Ng} \cap \alpha^N \neq \phi$, then for some $n_1, n_2 \in N$, $\alpha^{n_1 g} = \alpha^{n_2}$ and hence $n_1 g n_2^{-1} \in G_\alpha$. By normality of N, the above terms can be written as $n_3 g$ for some $n_3 \in N$. From this, $n_1 g n_2^{-1} \in Ng$ and hence $n_2 \in Ng$ which collapses Ng and N, thus forcing $\alpha^{Ng} = \alpha^N$. □

Corollary 14. *If G is primitive, all its normal subgroups are transitive.*

9 Finding Blocks

Problem: Given $\langle A \rangle = G \leq Sym(\Omega)$ a transitive group. Find a non-trivial block system or report PRIMITIVE.

Observe that if G is not primitive, for every $\alpha \in \Omega$, there exists a $\beta \neq \alpha$ such that G has a non-trivial block containing $\{\alpha, \beta\}$. And hence it is enough to solve the following MINBLOCK problem efficiently.
MINBLOCK: Given $\{\alpha, \beta\} \subseteq \Omega$, find the minimum block containing α and β.

The algorithm is very clever and neat. Define an undirected graph $X = (V, E)$ such that $V = \Omega$ and $E = \{(\alpha, \beta)\}^G = \{(\alpha^g, \beta^g) \ : \ g \in G\}$.

Claim 15. *The connected component C containing α is the minimum block.*

Proof. Note that $G \leq Aut(X)$ and also G is transitive on Ω. Hence connected components have to move as a whole, and thus connected components are blocks and hence C is a block.

Suppose C was not minimal, let $C_1 \subsetneq C$ be a block. Since the containment is strict, there exists an edge $(\gamma, \delta) = (\alpha^g, \beta^g)$ such that $\gamma \in C_1$ and $\delta \in C \setminus C_1$. Now $\gamma \in C_1^g \cap C_1$ but $\delta \in C_1^g \setminus C_1$ which contradicts that C_1 is a block. Hence C has to be the minimal block containing α and β. □

One can now run over all β for a given α to find solve the non-trivial block problem.

10 Membership Testing

Problem: Given $\langle A \rangle = G \leq Sym(\Omega)$ and $g \in Sym(\Omega)$, check if $g \in G$.

This problem clearly reduces to the problem of computing the order of the group given by a set of generators (to check for membership of g, throw

g into the generating set and check if the order changes). We are looking for a divide and conquer approach to solve membership.

A promising avenue is the orbit-stabilizer formula $|G| = |G_\alpha||\alpha^G|$. We know to compute α^G efficiently, and hence can recurse on the smaller subgroup G_α. But how do we get hold of a small generating set for G_α? The following marvellous idea of Schreir gives the answer.

Theorem 16 (Schreir)**.** *Let $\langle A\rangle = G$ and $H \leq G$. Let R be the set of distinct coset representatives of H, (i.e)*

$$G = \bigsqcup_{r\in R} Hr$$

Then

$$B = \left\{r_1 a r_2^{-1} \ : \ a \in A; r_1, r_2 \in R\right\} \cap H$$

generates H

Proof. For any given $r_1 \in R$ and $a \in A$, there exists a unique r_2 such that $r_1 a r_2^{-1} \in H$ (because $Hr_1 a = Hr_2$).

$$\begin{aligned}
RA &\subseteq BR \quad (r_1 a = (r_1 a r_2^{-1})r_2) \\
&\subseteq \langle B\rangle R \\
\implies RAA &\subseteq \langle B\rangle RA \\
&\subseteq \langle B\rangle\langle B\rangle R = \langle B\rangle R \\
\therefore \forall t \geq 0,\ RA^t &\subseteq \langle B\rangle R \\
\implies G &= \langle B\rangle R
\end{aligned}$$

Now since G can be partitioned into cosets of H and there are distinct representatives of every coset in R, unless $\langle B\rangle = H$, $\langle B\rangle R$ cannot cover the group. Hence $\langle B\rangle = H$. □

One could then look at $H = G_\alpha$ and the coset representatives are $g_\beta : \alpha \mapsto \beta$ and can be found in the orbit computation itself. But this still doesn't quite solve the problem since the size of the generating set is growing rapidly ($|B| = |R||H|$) and would get to exponential size in n steps.

This can also be tackled in a very clever way. We shall just see the sketch here, the details will be worked out next class.

Idea: For any two elements π and ψ in B, if $1^\pi = 1^\psi$ (both π and ψ map 1 to the same element), replace $\{\pi, \psi\}$ by $\{\pi, \pi^{-1}\psi\}$. This would then ensure that the two elements map 1 to different images now. Repeating this

replacement process, we can ensure that the elements of B are never larger than n^2.

The details shall be worked out in the next lecture, the interested reader could take this as an easy exercise though.

Algebra and Computation Course Instructor: V. Arvind

Lecture 4: Membership Testing in Permutation Groups

Lecturer: V. Arvind *Scribe: Shreevatsa R and Ramprasad Saptharishi*

11 Overview

In the previous lecture, we began studying the problem of testing membership in permutation groups. In this lecture, we describe its solution and explore related problems.

12 The problem

Given a subgroup G (given by a small generating set, say $G = \langle A \rangle$) of the permutation group S_n, and a permutation $g \in S_n$, the **membership testing** problem is to decide whether $g \in G$. If the answer is yes, we might also want a representation of g in terms of the generators.

If the output has to be a product $g = a_{i_1} a_{i_2} \dots a_{i_N}$, how large might N have to be, in the worst case? It is easy to see that we can ensure N is at most by $|G| - 1$: consider the prefix products $s_1 = a_{i_1}, s_2 = a_{i_1} a_{i_2}, \dots, s_N = a_{i_1} a_{i_2} \dots a_{i_N}$. If $N > |G| - 1$, either some s_j is the identity, in which case we can write $g = a_{i_{j+1}} \dots a_{i_N}$, or $s_j = s_k$ for some j and k, in which case we can write $g = a_{i_1} \dots a_{i_j} a_{i_{k+1}} \dots a_{i_N}$, so in either case N can be made smaller.

However, N might have to be very large. As an example, write n as $n = p_1 + p_2 + \dots + p_m$ where the p_is are distinct primes, and look at the cyclic group generated by

$$a = (1\ 2\ \dots\ p_1)(p_1 + 1\ \dots\ p_1 + p_2)(\dots)\quad \dots\quad (\dots) = C_1 C_2 \dots C_m.$$

The C_is are disjoint cycles with lengths p_i, so the order of a is $M = p_1 p_2 \dots p_m$. The group is $G = \langle \{a\} \rangle = 1, a, a^2, \dots, a^{M-1}$, in which the element a^{M-1} can be written only as the product of $M - 1$ as. So here, $N = p_1 p_2 \dots p_m - 1 \geq 2^m - 1$, and we know by the prime number theorem that $m = \Omega(\frac{p_m}{\ln p_m})$. Further, as $n = p_1 + p_2 + \dots + p_m \leq 1 + 2 + 3 + \dots + p_m \leq {p_m}^2$, we have that $p_m \geq \sqrt{n}$, so $N \geq 2^{\frac{c\sqrt{n}}{\log n}} - 1$ for some c. This is not polynomial in n.

Thus, we cannot hope to solve the problem in polynomial time if the required output is an explicit product; the "representation" has to be a circuit on A. In the example above, a^{M-1} can easily be expressed (or computed) in terms of the a through repeated squaring. Our algorithm will give a similar good representation.

13 Idea

Membership testing reduces to the problem of **order computation**, as $g \in G \iff |G| = |G \cup g|$. The idea for solving the latter, as we saw in the previous lecture, is to find a tower of subgroups

$$G = G^{(0)} \geq G^{(1)} \geq \cdots \geq G^{(n-1)} = \{1\}$$

such that the index $\left[G^{(i-1)} : G^{(i)}\right]$ is easy to calculate, for each i. Then, as $|G| = \prod_{i=1}^{n-1} \left[G^{(i-1)} : G^{(i)}\right]$, we can easily compute it.

Consider the pointwise stabilisers, and look at the tower of subgroups

$$G^{(i)} = \{g \in G \ : \ j^g = j \text{ for all } j,\ 1 \leq j \leq i\}$$

Here, $\left[G^{(i-1)} : G^{(i)}\right] \leq n - i$ for every i(why? since fixing $i-1$ leaves $n-i$ choices for the ith index), and our algorithm will compute it by computing a system of coset representatives for this. The algorithm will also give us a *new* generating set for G; thus it can be used to decide many membership queries without recomputing it.

14 Algorithm

14.1 Finding the cosets at each level of the tower

How do we find the coset representatives of $G^{(1)}$ in G? Let X_1 be the orbit of 1 under the action of G, and for any $k \in X_1$, let g_{1k} be the element that maps 1 to it (i.e., $1^{g_{1k}} = k$). Then

$$G = \bigcup_{k \in X_1} G^{(1)} g_{1k}$$

We shall see shortly (in subsection 14.3) how to find a small generating set for $G^{(1)}$. Assuming that we can do it, we can similarly find X_2, the orbit of 2 under the action of $G^{(1)}$, find the g_{2k}s such that $k = 2^{g_{2k}}$, and similarly find representatives for the cosets of $G^{(2)}$ in $G^{(1)}$.

In general, for each i from 1 to n, we find the orbit X_i of i under the action of $G^{(i-1)}$, and also, for each element $k \in X_i$, the element g_{ik} such that $i^{g_{ik}} = k$. Then $G^{(i-1)} = \bigcup_{k \in X} G^{(i)} g_{ik}$

Taking the union of the sets of coset representatives we get for each $\left[G^{(i-1)} : G^{(i)}\right]$ gives us a generating set for G. What we have just found has a name:

Definition 17. *Let Ω be an ordered set $\{\omega_1, \omega_2, \ldots, \omega_n\}$ and let $G \leq \mathrm{Sym}(\Omega)$. A **strong generating set** with respect to this ordering is the union of the sets of right-coset representatives (or the right-transversals) for $G^{(i)}$ in $G^{(i-1)}$, for $0 \leq i \leq n-1$.*

14.2 Testing membership of a given g

Given a $g \in G$, let $k_1 = 1^g$. If $k \notin X_1$, we can immediately conclude that $g \notin G$. Else, let $g^{(1)} = g g_{1k}^{-1}$. By construction, $g^{(1)} \in G^{(1)}$. Next, letting $k_2 = 2^{g^{(1)}}$, if $k_2 \notin X_2$, we can abort and report that $g \notin G$; else we let $g^{(2)} = g g_{1k_1}^{-1} g_{2k_2}^{-1}$ and continue similarly. In other words, for each i from 1 to n, we let $k_i = i^{g^{(i-1)}}$, abort if $k_i \notin X_i$, and set $g^{(i)} = g^{(i-1)} g_{ik_i}^{-1}$ otherwise, in which case it is guaranteed to be in $G^{(i)}$. If at any time $g^{(r)} = g g_{1k_1}^{-1} g_{2k_2}^{-1} \ldots g_{rk_r}^{-1} = 1$ then we have g as a product of elements of G. If $g \in G$, it is guaranteed that this will eventually happen, for if we have not aborted by the time i takes the value of n, then it is guaranteed that $g^{(n-1)} \in G^{(n-1)} = 1$.

14.3 Finding generating sets for the subgroups

Our algorithm above for finding the strong generating set depends on being able to find a generating set for $G^{(i)}$ when we know one for $G^{(i-1)}$. This is made possible by Schreier's lemma, as we saw in the previous lecture.

Theorem 1 (Schreier's lemma)**.** *If $G = \langle A \rangle$ and R is a set of distinct right coset representatives for the subgroup H in G, then*

$$B = \left\{ r_1 a r_2^{-1} \ : \ a \in A, r_1, r_2 \in R \right\} \cap H$$

generates H.

For every r_1 and a, there is a unique r_2 such that $r_1 a r_2^{-1} = h$, for any h in H, so we know that $|B| \leq |R|\,|A|$. However, this alone is not sufficient for us to complete our algorithm, as it does not ensure polynomial time — the size of the generating set might increase by $\Theta(n)$ each time. To avoid

this, we need to make sure we can keep the generating set small. We now show that it can be done.

14.4 The REDUCE Algorithm

Theorem 2. *Given a group $G \leq S_n$, we can find a generating set for it of size at most n^2.*

The idea is to remove collisions as you see them. Suppose π and ψ are two elements of your generating set such that they map 1 to the same element, (i.e) $1\pi = 1\psi$. What we do then is replace $\{\pi, \psi\}$ by $\{\pi, \psi\pi^{-1}\}$. This then ensures that one of the elements fix 1, and is taken care in the following levels. A better way to look at it would be to think of a table where row i represents $G^{(i)}$.

Start with the bottom row, representing $G^{(0)} = G$. For each element $\pi \in B$, if $1\pi = 1$, move it to the row $G^{(1)}$, else place it in column 1^{π}. Whenever you have collisions, do the pruning process and send the element that fixes 1 and send it to the next row. Once row 1 is done (no more collisions), move to the next and repeat this process.

The complete algorithms for REDUCE and MEMBERSHIP can be found at the end of this file.

15 Other Problems

Using the technique of using group towers, there is a whole pool of problems we could inspect. We shall see a few of them now, which will be very useful for the lectures to follow.

SUBGROUP: Given $H = \langle B \rangle$ and $G = \langle A \rangle$, subgroups of Sym_n. Check if $H \leq G$.

The solution is extremely simple, for every element $b \in B$, check if $b \in G$ using the MEMBERSHIP algorithm.

NORMAL: Given $H = \langle B \rangle$ and $G = \langle A \rangle$, subgroups of Sym_n. Check if H is a normal subgroup of G, denoted by $H \lhd G$.

This is also easy, for each $a \in A$ and each $b \in B$, check if $aba^{-1} \in H$ using MEMBERSHIP.

Definition 18. *For any subgroup H of G, normal closure of H is defined as the smallest normal subgroup of G that contains H. It is denoted by $\left\langle H^G \right\rangle$*

$$H \leq \left\langle H^G \right\rangle \lhd G$$

The normalizer of H, denoted by $N_G(H)$, is the largest subgroup of G in which H is normal.

$$H \lhd N_G(H) \leq G$$

NORMAL CLOSURE: Given $H = \langle B \rangle$ and $G = \langle A \rangle$, subgroups of Sym_n, find $\left\langle H^G \right\rangle$

This is easy too, look at $\left\{aba^{-1} \ : \ a \in A, b \in B\right\}$. If something of this set is not in H, throw it into B and repeat. Everytime, the size of the group doubles and hence you will definitely hit the normal closure quickly.

15.1 The Group-Intersection Problem

GROUP-INTER: Given $G = \langle A \rangle$ and $H = \langle B \rangle$, subgroups of Sym_n. Find $G \cap H$.

There is no known polynomial time algorithm for this problem in general, and we don't expect one to exists even because of the following theorem.

Theorem 19. SET-STAB *is polynomial time equivalent to* GROUP-INTER

Proof. SET-STAB $\leq_P$ GROUP-INTER:

Suppose Δ is the set we want to stabilize, all we need to do is to compute $G \cap \left\{\mathrm{Sym}_\Delta \times \mathrm{Sym}_{\Omega \setminus \Delta}\right\}$. One could just choose transpositions as a generating set for the product.

The intersection of the two sets precisely yields $stab_\Delta(G)$.

GROUP-INTER $\leq_P$ SET-STAB:

Look at the product $G \times H \leq \mathrm{Sym}_\Omega \times \mathrm{Sym}_\Omega \leq Sym_{\Omega \times \Omega}$ and let the action be just the coordinate wise action, $(g,h)(i,j) = (i^g, j^h)$. All we need to do now is stabilize the diagonal, $\Omega \times \Omega = \{(i,i) \ : \ i \in \Omega\}$ and this would yield $G \cap H$. □

And since we say that graph isomorphism reduced to SET-STAB, it is unlikely that we have a polynomial time algorithm for the intersection problem.

However, for a special case when G normalizes H (i.e $G \leq N_{\mathrm{Sym}_\Omega}(H)$), we can solve the intersection problem in polynomials time.

Claim 20. *If G normalizes H, then we can compute $G \cap H$ in polynomial time.*

Proof. Again, we are looking for a tower of subgroups with "nice" properties. And since G normalizes H, the central idea is that $GH = \{gh \ : \ g \in G, h \in H\}$ is a subgroup of Sym_n. And hence, for all i, $G^{(i)}H$ is also a subgroup. Further, G normalizes $H \implies H \lhd GH$.

The tower we are looking for is

$$G \cap H = G \cap G^{(n-1)}H \leq G \cap G^{(n-2)}H \leq \cdots \leq G \cap GH = G$$

And since the generating set for $G^{(i)}H$ is just the union of the generating set for $G^{(i)}$ and H, we can check for membership in $G \cap G^{(i)}H$. Thus, using Schreier's lemma and the REDUCE algorithm, we can descend the tower computing generating sets.

An additional property we need is that the index between consecutive elements of the tower is small. We leave it as an exercise to show that $\left[G \cap G^{(i-1)}H : G \cap G^{(i)}H\right] \leq n - i$. □

Algorithm 1 REDUCE

$B_0 = B$
$A[\,][\,]$, an empty $n \times n$ array
for $i = 0$ to $n - 1$ **do**
  for all $\psi \in B_i$ **do**
    $j = i^{\psi}$
    if $A[i][j]$ is empty **then**
      if $j = i$ **then**
        $B_{i+1} = B_{i+1} \cup \{\psi\}$
      else
        $A[i][j] = \psi$
      end if
    else
      $\pi = A[i][j]$
      $B_{i+1} = B_{i+1} \cup \left\{\pi \cdot \psi^{-1}\right\}$
    end if
  end for
end for
discard all trivial elements from $\bigcup B_i$
return $\bigcup B_i$

Algorithm 2 MEMBERSHIP

Input: $g \in \mathrm{Sym}_n$ and a generating set A for $G^{(i)} \leq \mathrm{Sym}_n$ and the index i

if $g = id$ **then**
  return true
end if
$X_i = (i+1)^{G^{(i)}}$ {use the ORBIT algorithm}
compute the set R of distinct coset representatives of $G^{(i+1)}$ in $G^{(i)}$
$k = (i+1)^g$ {image of $i+1$ under the action of g}
if $k \notin X_i$ **then**
  return false
else
  compute generating set B for $G^{(i+1)}$ using Schreier's lemma
  REDUCE B
  pick g_{ik} from R, the coset representative of $G^{(i)}$
  $g' = g \cdot g_{ik}^{-1}$
  return MEMBERSHIP(g',B, $i+1$)
end if

Algebra and Computation Course Instructor: V. Arvind

Lecture 5: More on Subgroups and GROUP-INTER

Lecturer: V. Arvind *Scribe: Ramprasad Saptharishi*

16 Overview

Last lecture we saw that membership in a permutation subgroup can be done efficiently and inspected the recognizability of some group properties. This lecture we shall look at other properties of groups that can be detected efficiently.

17 The General Setting

Over all examples that we have seen, there is a central structure to all recursive algorithms that we are doing; we want small generating sets for recursive calls. In general, we are provided a group $G = \langle A \rangle$ and a subgroup H that is only given as a black box, and we wish to find a generating set for H to inspect its properties. The general idea was to find a tower of subgroups between G and H, like

$$G = G_0 \geq G_1 \geq G_2 \geq \cdots \geq G_r = H$$

and we want certain "nice" properties to be satisfied. This can infact be generalized to non-permutation groups as well. Assume that our groups are embedded into a larger group, where multiplication, inverses etc are known.

1. Given a generating set for G_i, we should be able to obtain a generating set for G_{i+1} quickly
2. The index $[G_i : G_{i+1}] \leq m$ is not too large
3. Quick membership algorithms in G_i
4. A REDUCE procedure to bring down the generating set.

The *quick membership algorithm* requirement has a definition by its own.

Definition 21. *A group H is said to be polynomial time recognizable if there exists a polynomial time algorithm that solves membership in H.*

Though H is not explicitly given, this is a very reasonable definition. For example, look at H being the automorphism group of some graph. Finding what H is very unlikely, but certainly given a permutation of vertices, one can easily check if it is infact an automorphism or not.

The properties stated before are precisely what we required for the recursion to go through.

Claim 22. *With the four properties, we can find a generating set for H quickly*

Proof. The first step is to find coset representatives of H in G.

$$G = Hx_1 \cup Hx_2 \cup \cdots \cup Hx_r \quad r \leq m$$

In the permutation group case, just the orbit of $\{1\}$ would have finished it, but what about this case? Note that G acts transitively on the set of right cosets! Hence one can find a set of unique coset representatives by makign G act on the $id \in G$ and checking of two elements x and y of the orbit belong to the same coset of H (this happens if and only if $xy^{-1} \in H$).

Once we have the coset representatives, Schreier's lemma gives a generating set for the next element of the tower and recursion happens. Crowding of generators around H can be prevented by the REDUCE algorithm. Putting them all together, we would have a small generating set for H. □

18 Subnormality and GROUP-INTER

Definition 23. *A subgroup H of G is said to be subnormal in G if there exists a tower of subgroups chained by normality (i.e)*

$$H = G_r \lhd G_{r-1} \lhd \cdots \lhd G_0 = G$$

This is denoted by $H \lhd\lhd G$.

SUBNORMAL: Given $H = \langle B \rangle$ and $G = \langle A \rangle$. Check if $H \lhd\lhd G$.

The following claim is the core of the algorithm to detect subnormality.

Claim 24. $H \lhd\lhd G \iff H \lhd\lhd \left\langle H^G \right\rangle$

Proof. One way is obvious, if $H \lhd\lhd \left\langle H^G \right\rangle$, then immediately we have the tower $H \lhd\lhd \left\langle H^G \right\rangle \lhd G$.

The other direction is also fairly straightforward. Suppose we had a tower for $H \lhd\lhd G$,

$$H = G_r \lhd G_{r-1} \lhd \cdots \lhd G_0 = G$$

then clearly, just intersecting the entire tower with the normal closure $\langle H^G \rangle$, we get

$$H = H \cap \langle H^G \rangle \lhd G_{r-1} \cap \langle H^G \rangle \lhd \cdots \lhd G \cap \langle H^G \rangle = \langle H^G \rangle$$

which is a tower for $H \lhd\lhd \langle H^G \rangle$. □

Hence if H were subnormal, you are guaranteed to find a series through the normal closure. Hence the algorithm is then immediate:

Compute the normal closure $\langle H^G \rangle$. If $\langle H^G \rangle = H$, then we are done since $H \lhd G$ is our tower. If $\langle H^G \rangle = G$, then we know that H cannot be subnormal in G since there is no way by which you can obtain the tower if $\langle H^G \rangle = G$. Hence if $\langle H^G \rangle$ was a strict subgroup of G, recurse on this.

1: $K_1 = G$;
2: **while** $\langle H^{K_1} \rangle \neq H$ **do**
3: **if** $\langle H^{K_1} \rangle = K_1$ **then**
4: **output false**
5: **else**
6: $K_1 = \langle H^{K_1} \rangle$
7: **end if**
8: **end while**
9: **output true**

We will now go on to show that if $H \lhd\lhd G$, then we can compute $G \cap H$ efficiently.

Theorem 25. *Let $H = \langle B \rangle$ and $G = \langle A \rangle$ be subgroups of Sym_n. Given the promise that $H \lhd\lhd \langle G, H \rangle$, we can compute a small generating set for $G \cap H$ in polynomial time.*

Proof. Using the subnormality algorithm, we can infact compute the normal series of H. Let

$$\begin{aligned} H = G_r \lhd G_{r-1} \lhd \quad \cdots \quad \lhd G_0 = \langle G, H \rangle \\ G \cap H \lhd G \cap G_{r-1} \lhd \quad \cdots \quad \lhd G \cap G_0 = G \cap \langle G, H \rangle = G \end{aligned}$$

Now there are two questions to ask before we apply the recursion procedure, is the index between adjacent indices small? Given a generating set for $G \cap G_i$, can we find a generating set for $G \cap G_{i+1}$?

As for the second question, note that $G \cap G_i$ normalizes G_{i+1}! And hence we can use the previous algorithm to get a generating set for the intersection $(G \cap G_i) \cap G_{i+1} = G \cap G_{i+1}$

As for the first question, exercise! □

18.1 Recognizing Solvability

Definition 26. *For any group G, the commutator subgroup G' of the group, also denoted by $[G, G]$, is defined as*

$$[G, G] = \left\{ g_1 g_2 g_1^{-1} g_2^{-1} \ : \ g_1, g_2 \in G \right\}$$

Also, it's easy to see that $G' \lhd G$ and G/G' is abelian, infact it's the smallest group that satisfies this property!

Definition 27. *A group G is said to be solvable if we can find a tower*

$$G \rhd G_1 \rhd G_2 \rhd \cdots \rhd \{1\}$$

such that for each i, G_i/G_{i+1} is abelian.

Equivalently, a group G is solvable if the you can hit $\{1\}$ by looking at successive commutator subgroups.

SOLVABLE: Given a group $G = \langle A \rangle$, check if G is solvable.

Here's the solution: Take $[A, A] = \left\{ a_1 a_2 a_1^{-1} a_2^{-1} \ : \ a_1, a_2 \in A \right\}$ and consider its normal closure. The claim is that, this is is the commutator subgroup of G! Once we prove this, we can descend by computing successive normal closures, and if we get stuck, then G can't be solvable.

We leave the proof of the claim and details of this algorithm as an exercise.

19 Jerrum's Filter

Recall the MEMBERSHIP algorithm, the REDUCE procedure was crucial in keeping the generating set in control through the recursion. Our algorithm made sure that the generating set does not grow beyond n^2. However, there are much stronger results, and reduce algorithms.

Theorem 28 (Neumann)**.** *For $n > 3$, every subgroup of Sym_n can be generated by atmost $\left\lfloor \frac{n}{2} \right\rfloor$ elements.*

Infact, the above theorem is tight. Consider $\Omega = \{a_1, a_2, \cdots, a_m, b_1, \cdots, b_m\}$ and let G be the group generated by the transpositions $\{(a_i, b_i) \ : \ 1 \leq i \leq m\}$. This is a generating set for G and has size $\frac{n}{2}$. And since G is isomorphic to $\mathbb{F}_2^m$, this matches the lower bound as well.

However, to algorithmically compute the generators, we need slightly weaker bound.

Theorem 29 (Jerrum)**.** *Any subgroup G of Sym_n has a generating set of size $(n-1)$. Infact, given $G = \langle A \rangle$, we can compute an $(n-1)$ sized generating set in polynomial time.*

Proof. Define the graph $X_A = ([n], E_A)$ as follows: For each $g \in A$, let $i_g \in [n]$ be the least point moved by g. Add the edge $e_g = (i_g, i_g^g)$ to E_A.

We now have an undirected graph on n vertices. We shall go on to show that we can keep modifying A to get to get to a graph $X_{A'}$ that is acyclic (and of course, $G = \langle A' \rangle$), and hence the theorem follows.

Before we go into the algorithm, we need one more terminology. For any $T \in \mathrm{Sym}_n$, the weight of the graph X_T (denoted by $w(T)$) is defined as $\sum_{g \in T} i_g$. An obvious upper bound on $w(T)$ is $|T|\, n$.

The algorithm is an online algorithm, it maintains a set A such that X_A is acyclic and as a new element g comes in, it changes A a little to accomodate g and still maintain an acyclic graph.

At any stage assume that we have a set S such that X_S is acyclic; enter g. If even on addition of e_g, the graph remains acyclic, there is nothing to be done, just add that edge and add g to the set S.

Otherwise, in $X_{S \cup \{g\}}$, there exists a unique cycle containing the edge e_g. Now each edge of the cycle is labelled with an element of S (the direction gives an ambiguity of whether it is g_k or g_k^{-1}, this will be usually denoted by g_k^ϵ where ϵ can be 1 or -1).

Let i be the least point on this cycle. Since i is a part of the cycle, one of the two edges incident on i in the cycle must be labelled as g_0 such that $i = i_{g_0}$. In that direction, walk from i round the complete cycle back to i; this naturally corresponds to a product of the form $g_0 g_1^{\epsilon_1} g_2^{\epsilon_2} \cdots g_k^{\epsilon_k} = h$. Replace g_0 by h in the set S (unless h is trivial, in which case just discard it).

Note that this transformation doesn't change the group generated by the set, but on the other hand, the choice of h demands that $j^h = h$ for all $1 \leq j \leq i$ and hence fixes sometime beyond i. And since we chose i to be the least element of the cycle, the weight of the graph increases when we replace g_0 by h.

The size of our set remained the same, but the choice of h forced the weight to go up. Hence in a polynomially many steps, we are sure to hit the upper bound and hence will end up with an acyclic graph (with some h becoming trivial in the process).

Repeating the process with all elements of A, we have a generating set A' with an acyclic graph $X_{A'}$, and hence $|A| \leq n - 1$. □

Let us revisit the original REDUCE algorithm that restricted our generating set to n^2, infact that can also be seen in this setting. In that algorithm, we were looking at collisions and doing modifications based on that, those precisely correspond to 2 cycles in X_A.

Hence Jerrum's filter can also be thought of as the original REDUCE algorithm but being more mindful about the entire graph rather than just 2 cycles.

Algebra and Computation Course Instructor: V. Arvind

Lecture 6,7: Special Cases of GRAPH-ISO

Lecturer: V. Arvind *Scribe: Ramprasad Saptharishi*

20 Overview

It is unlikely that we have an efficient algorithm for GRAPH-ISO, but certain special cases of it can be solved in polynomial time. In this lecture, we shall inspect two such cases, coloured graphs with bounded colour classes and trivalent graph. [4]

21 Bounded Colour Multiplicity GRAPH-ISO: $BCGI_b$

Instead of general graph isomorphism instances, $BCGI_b$ adds an extra structure to the graphs by associating with each vertex a colour. Further, assume that each colour class[5] has its size bounded by a constant b. Given two graphs X_1 and X_2 with such a promise, can check if they are isomorphic efficiently?

Without loss of generality, we can assume that X_1 and X_2 are connected (otherwise consider their complements). Consider the graph $(V, E) = X = X_1 \cup X_2$ and from our earlier lectures, checking isomorphism can be reduced to checking the automorphism group of X. The additional colour structure imposes the constraint that $Aut(X) \leq \bigotimes \mathrm{Sym}(C_i)$ where C_i is the colour class for colour i. The idea is that we use the SET-STAB reduction in this scenario and show that we can compute the necessary stabilizer efficiently.

Let $E_i = E \cap \binom{C_i}{2}$ for each colour i (the set of intercolour edges) and $E_{ij} = E \cap (C_i \times C_j)$ for colour pairs $i \neq j$ (cross edges). Clearly, any $\pi \in \bigotimes \mathrm{Sym}(C_i)$ is a colour preserving automorphism *if and only if*, $E_i^\pi = E_i$ and $E_{ij}^\pi = E_{ij}$. Thus we now have got it to a SET-STAB form.

Let $G = \bigotimes \mathrm{Sym}\, C_i = \langle A \rangle$, by choosing transpositions as generators. Now define $\Omega = (\bigcup C_i) \cup \left(\bigcup 2^{\binom{C_i}{2}}\right) \cup \left(\bigcup 2^{C_i \times C_j}\right)$, which is just identifying the subsets E_i and E_{ij} as points. The bound on the colour classes tell us

[4] We also discussed a partial solution to the solvability exercise he gave in the last class, but I'm not putting it here

[5] set of vertices of a given colour

that $|\Omega| \leq n + r2^{\binom{b}{2}} + \binom{r}{2}2^{b^2}$, if r is the number of colour classes, and since b is a constant $|\Omega| = poly(n)$.

G can be naturally extended to act on Ω (extension in the most obvious sense, if i is sent to something and j is sent to something, E_ij should go to the right thing). And now, $Aut(X)$ is just finding the subgroup that pointwise stabilizes the E_i clusters and the E_{ij} cllusters and this can be done using our *strong generating set* tower discussed in lecture 3.

The running time is $poly(|\Omega|)$ and hence $poly(|X|)$.

22 Bounded Degree GRAPH-ISO

Another restriction that we can impose on graphs is the degree of each vertex, suppose we are given the promise that the degree of each vertex is bounded by a constant d, can we solve GRAPH-ISO?

The case when $d = 1$ is trivial, just a bunch of independant edges and so is the case when $d = 2$. The first interesting case is when $d = 3$, which is also called *trivalent graph isomorphism.*

We are given two graphs X_1 and X_2 with the promise that their degrees are bounded by 3, and we need to check if they are isomorphic. Firstly note that we can assume that both of them are connected, since if not we can just look at the connected components and work with all possible pairs[6]. Checking if they are isomorphic is equivalent to computing their automorphism group.

Further, suppose had a distinguished edge, and we are only intereseted in automorphisms that fix that edge, is that good enough? Yes it is. Fix some edge $e_{uv} \in X_1$, for each $e_{pq} \in X_2$, add a new edge that 'connects' e_{uv} and e_{pq} (add the midpoints as another vertex and join the two midpoints). Thus if an isomorphism swapped e_{uv} and e_{pq}, that isomorphism will fix e. And since we are doing this over all edges e_{pq}, the two problem are clearly equivalent. Hence we shall restrict ourselves to finding $Aut_e(X)$ for some graph X where e is a distinguished edge we want to fix.

The algorithm works by building the automorphism groups by approximations, $Aut_e(X_r)$ where each X_r is a subgraph of X. For each r, define X_r to be the consisting of all vertices and edges that appear in paths of length $\leq r$ passing through e. This layers X with respect to distance from e.

picture needed

[6] note that the complement idea won't work since graph will no longer be trivalent

And clearly, since e is a distinguished edge, any $\pi \in Aut_e(X)$ must preserve layers. And infact we have the following crucial theorem by Tutte.

Theorem 30 (Tutte)**.** *If X is a connected trivalent graph and e is any edge in X, then $Aut_e(X)$ is a 2-group (a group of order 2^m).*

Proof. The basic idea is that the automorphism groups are successive approximations and each expansion is through a 2-group.

The proof will be an induction on i, assume that $Aut_e(X_i)$ is a 2-group. Since any automorphism that preserves e has to respect the layers, we have a natural homomorphism $\phi : Aut_e(X_{i+1}) \longrightarrow Aut_e(X_i)$, which is just the projection function ($Aut_e(X_{i+1})$ preserves layers till X_{i+1}).

Hence $|Aut_e(X_{i+1})| = |\phi(Aut_e(X_{i+1}))| \cdot |\ker \phi|$ and since $\phi(Aut_e(X_{i+1}))$ is a subgroup of $Aut_e(X_i)$, it is a 2-group. Hence all that's left to do is to check that $\ker \phi$ is a 2-group as well.

We are interested in counting $\pi \in Aut_e(X_{i+1})$ that fixes X_i pointwise. If $V = V(X_{i+1}) \setminus V(X_i)$, then any non-trivial π have to do something on V alone. But note that the graph X is trivalent, and hence any $u \in V(X_i)$ can be connected to atmost 2 vertices in V (since degree of u is bounded by 3) and hence any automorphism of X_{i+1} that fixes X_i can atmost swap the two neighbours of u. Thus any $\pi \in \ker \phi$ satisfies the constraint that $\pi^2 = id$ and hence $\ker \phi$ is also a 2-group. □

22.1 A Road Map

We want to compute $Aut_e(X)$, and we shall do it using the tower induced by the different layers X_r. The general philosophy is the following:

If $\phi : G \longrightarrow H$ is a group homomorphism and if we had a generating set for $\ker \phi = \{k_1, k_2, \cdots, k_n\}$ and $\phi(H) = \{\phi(g_1), \phi(g_2), \cdots, \phi(g_m)\}$, then we can find a generating set for G efficiently.

We are going to use the homomorphisms $\phi : Aut_e(X_{r+1}) \longrightarrow Aut_e(X_r)$ to ascend the tower and finally get to a generating set for $Aut_e(X)$. We need to find

1. A generating set for $\ker \phi$
2. A generating set for $\phi(Aut_e(X_{r-1}))$

22.2 A generating set for $\ker \phi$

The proof of Tutte's theorem infact gave us the algorithm. Observe that for every vertex $v \in X_r$ is attached to atmost two vertices of $X_{r+1} \setminus X_r$ and an automorphism could possible swap these two neighbours of v.

When would this not be possible? Precisely when the neighbourhoods of the two vertices are different! Thus if $w_1, w_2 \in X_{r+1} \setminus X_r$ are neighbours of $v \in X_r$, a transposition (w_1, w_2) would be a valid automorphism fixing X_r *if and only if* $\Gamma(w_1) = \Gamma(w_2)$. And this can be easily checked by inspection. Thus in linear time, we can infact get a generating set (a set of disjoint transpositions) for $\ker \phi$.

22.3 A generating set for $\phi(Aut_e(X_{r+1}))$

This is the harder part. Since we'll be referring to vertices of $X_{r+1} \setminus X_r$, we shall refer to this set as V_r. Note that every vertex $v \in V_r$ is connected to 1 or 2 or 3 neighbours of X_r. Hence, let A be the collection of all subsets of X_r of size 1 or 2 or 3. Then we have the following neighbourhood map $\Gamma_r : V_r \to A$ which takes each vertex to the set of neighbours in the graph X_{r+1}.

Note that for every automorphism $\sigma \in Aut_e(X_{r+1}), \Gamma_r(\sigma(v)) = \sigma(\Gamma_r(v))$, and further if it were in the kernel, then $\Gamma_r(v) = \sigma(\Gamma_r(v))$. Call two vertices $v_1, v_2 \in V_r$ as *twins* if $\Gamma_r(v_1) = \Gamma_r(v_2)$.

Hence, any $\sigma \in \phi(Aut_e(X_{r+1}))$ has to stabilize the set of fathers with just 1 son.

$$A_1 = \{a \in A \ : \ a = \Gamma_r(v) \text{ for some unique } v \in V_r\}$$

σ must also stabilize the set of fathers of twins,

$$A_2 = \{a \in A \ : \ a = \Gamma_r(v_1) = \Gamma_r(v_2), v_i \neq v_j\}$$

And apart from reaching out to vertices of V_r, the next layer also induces edges between vertices of X_r, and any automorphism must certainly preserve these as well.

$$A_3 = \{\{w_1, w_2\} \in A \ : \ (w_1, w_2) \in X_{r+1}\}$$

Infact, these are all we need to ensure so that $\sigma \in Aut_e(X_r)$ is infact in $\phi(Aut_e(X_{r+1}))$.

Claim 31. *The image is precisely those automorphisms $\sigma \in Aut_e(X_r)$ which stabilize the sets A_1, A_2 and A_3.*

Proof. We need to show that if σ stabilizes the three sets, then we can extend it to an automorphism of X_{r+1}. The extension is built as follows:

- For each single child v, $\Gamma_r(v) \in A_1$, since $\sigma(\Gamma_r(v)) \in A_1$, map v to the only child of $\sigma(\Gamma_r(v))$.

- For each pair of twins v_1, v_2, $\Gamma_r(v_1) = \Gamma_r(v_2) \in A_2$, since $\sigma(\Gamma_r(v)) \in A_2$, map $\{v_1, v_2\}$ to the sons of $\sigma(\Gamma_r(v_1))$ in any order.

The constructions enforces that it respects edges between X_r and V_r. And since it also stabilizes A_3, σ also respects the edges between vertices of X_r that were newly formed. Hence σ is indeed can be extended to an automorphism of X_{r+1}. □

Now we have reduced the isomorphism problem to a set-stabilizer problem for 2-groups. We shall discuss how to deal with it in the next class.

Algebra and Computation Course Instructor: V. Arvind

Lecture 8,9: General Bounded Degree GRAPH-ISO

Lecturer: V. Arvind *Scribe: Ramprasad Saptharishi*

23 Recap

In the last few classes, we saw that we could reduce trivalent graph isomorphism to SET-STAB for 2-groups. The algorithm is identical to a more general problem, which could be used to solve GRAPH-ISO with degree of each vertex bounded by a constant d.

The idea is more or less the same, we have two graphs with max degree bounded by a constant d. We shall, in polynomial time, reduce this problem to a restricted SET-STAB problem that could be solved in polynomial time.

24 Generalization to $(\leq d)$-degree graphs

Given two graphs X_1 and X_2 with the additional promise that the max degree in both of them is bounded by a constant d.

Just as in the trivalent case, add a new bridge between two edges of the graph. Then problem the reduces to finding the automorphism group of the graph that fixes this distinguished edge e. Let X_i be the subgraph consisting of edges that are reachable by paths of length atmost i through e. As before, any e-automorphism must preserve these layers and hence we have a natural map $\pi_i : Aut_e(X_{i+1}) \longrightarrow Aut_e(X_i)$. Using these maps, we are going to find a generating set for $Aut_e(X_n) = Aut_e(X)$.

The claim is that the groups in discussion are special, all their composition factors are $\leq d$. We shall prove this by induction just as in the trivalent case (where each of the factors were atmost 2, thus giving us a 2-group).

24.1 $\ker \pi_i$ has small factors

The kernel is the set of e-automorphisms of X_{i+1} that fix X_i. And since the degree is bounded by d, every vertex in X_i can reach out to atmost d vertices in X_{i+1}, and all automorphisms in the kernel can only permute the vertices with common parents.

Let A be the set of subsets of $V(X_i)$ of size atmost d. Then we have the neighbourhood map $\Gamma : V(X_{i+1}) \setminus V(X_i) \longrightarrow A$ such that $\Gamma(u) = a$ if a

is the set of neighbours of u. Note that $\left\{\Gamma^{-1}(a)\right\}$ partitions the vertices in $X_{i+1} \setminus X_i$ and $|\Gamma^{-1}(a)| \leq d$.

Thus clearly, $\ker \pi_i = \bigotimes \mathrm{Sym}\left(\Gamma^{-1}(a)\right)$ and thus has composition factors atmost d.

This hence proves the claim that each of the automorphism has small factors. The above argument also yields a generating set for the kernel. Once we have a generating set for the image as well, we can construct the automorphism group of X_{i+1} from X_i.

24.2 Image of π_i reduces to restricted SET-STAB

Similar to the trivalent case, the image is just the set of all automorphisms that stabilize the following sets.

Edges within X_i:

$$A' = \{\text{2-subsets of } A \text{ that are new edges inside } X_i\}$$

Parent with same number of siblings:

$$A_s = \left\{a \in A \ : \ |\Gamma^{-1}(a)| = s\right\} \qquad 1 \leq s \leq d$$

Claim 32. *The image is precisely the subgroup of automorphisms of $Aut_e(X_i)$ that stabilize each of the A_s and A'.*

The proof is identical to the trivalent case, we shall hence skip it.

25 The restricted SET-STAB

Given a group $G = \langle A \rangle \leq \mathrm{Sym}(\Omega)$ and $\Delta \subseteq \Omega$. We are interested in finding $G_\Delta = \{g \in G \ : \ \Delta^g = \Delta\}$.

We do not hope to solve the general problem in polynomial time. However, a restricted version where G is a group with small compositional factors can be done efficiently. Divide-and-conquer is the method adopted in the algorithm.

Definition 33. *A finite group $G \in \mathcal{B}_d$ if every composition factor is isomorphic to a subgroup of S_d.*

From the earlier sections, we saw that $Aut_e(X) \in \mathcal{B}_d$ if the degree of X is bounded by d. Hence, we need to solve the SET-STAB problem for groups in $\mathcal{B}_d$.

Firstly, if $\Omega_1, \Omega_2, \cdots, \Omega_m$ are the G orbits then it is equivalent compute the stabilizer of $\Omega_i \cap \Delta$ for all i. Thus, we can assume that G acts transitively on Ω.

How do we further divide? Consider the block structure! Recall that blocks (Δ) are subsets of Ω such that they move as a whole ($\Delta^g \cap \Delta \neq \phi \Rightarrow \Delta = \Delta^g$). A block system naturally induces a tree structure on Ω. We shall call this the structure forest of G. And since we've already broken down Ω into G-orbits, we infact have just a structure tree.

picture might be useful

Each node is labelled by a block, that is the union of its children. The leaves are the trivial singleton blocks.

Now consider the top-most level, the root is labelled by Ω, which is the union of its children say $\Gamma_1, \cdots, \Gamma_m$. Now, considering each Γ_i as a point, G's action on them is primitive (there are no non-trivial blocks). The kernel of this action is the group $H = \{g \in G \ : \ \Gamma_i^g = \Gamma_i \quad \forall i\}$.

Let $G = \bigcup_{i=1}^{r} H\tau_i$, then $G_\Delta = \bigcup (H\tau_i)_\Delta$. But in order to talk about a stabilizer in a coset, we need to generalize the stabilizer problem.

25.1 Generalized SET-STAB

One could think of Δ as a 2-colouring of the set Ω. Thus a natural generalization would be the following.

Given a colouring C of Ω, we wish to find

$$G_C = \{g \in G \ : \ a^g \text{ has the same colour as } a \quad \forall a \in \Omega\}$$

Let us extend this a little further to capture the coset structure as well.

Given a coloured Ω , $G \leq \mathrm{Sym}(\Omega)$, $\sigma \in \mathrm{Sym}(\Omega)$ and $\Omega' \subseteq \Omega$ that is G-stable[7], we wish to find

$$stab(\Omega', G\sigma) = \left\{g \in G\sigma \ : \ \omega^g \sim \omega \quad \forall \omega \in \Omega'\right\}$$

Then we have the following easy claim, the proof is left to the reader (simple though)

Claim 34. *$stab(\Omega', G\sigma)$ is a right coset of $stab(\Omega', G)$.*

We can hence ask the following question: Given $G = \langle A \rangle \leq \mathrm{Sym}(\Omega)$ that is coloured and a Ω' that is a G-stable set and a $\sigma \in \mathrm{Sym}(\Omega)$. How do we compute $stab(\Omega', G\sigma)$?

With this generalization, we can do the divide and conquer.

[7] stable under action of G, union of orbits

26 The Divide and Conquer

Given a group $G \in \mathcal{B}_d$, we want to solve the stabilizer problem. The first step is to break Ω into orbits and work on each of them separately. Once we do that, we look at the block structure of G and at the topmost level.

We have a single tree, with Ω at the root with $\Gamma_1, \Gamma_2, \cdots, \Gamma_m$ as the primitive blocks on which G acts. The kernel of this action is $H = \{g \in G \ : \ \Gamma_i^g = \Gamma_i \quad \forall i\}$. Let

$$G\sigma = \bigcup_{i=1}^{r} H\tau_i\sigma$$

Now all that we need to do is compute $stab(\delta \cap \Gamma_i, H\tau_j\sigma)$ for all i and j and we have a recursive algorithm.

The catch here is that we need to know how many recursive calls to make. It would be completely useless if r (the index of H in G) was exponential; we need a decent bound on r. Fortunately, we do have such a bound.

Theorem 35 (Babai, Cameron, Pálfy)**.** *If $G \leq S_n$ belongs to $\mathcal{B}_d$ and is primitive, then $|G| \leq n^{cd}$ where c is an absolute constant.*

Thus, $r \leq m^{cd}$ and the algorithm would then go through. It can be shown that we then have an $O(n^d)$ algorithm for the restricted SET-STAB, and hence a $O(n^{d^2})$ algorithm for bounded degree graph isomorphism.

Algebra and Computation Course Instructor: V. Arvind

Lecture 10: General GRAPH-ISO

Lecturer: V. Arvind *Scribe: Ramprasad Saptharishi*

27 Overview

In the last few classes, we solved some special cases of the graph isomorphism problem. Now we shall use those ideas to give an algorithm for the general GRAPH-ISO problem. This, of course, is not a polynomial time algorithm but is interesting nevertheless.

Since any graph on n vertices has degree bounded by n, just a naive simulation of the bounded degree GRAPH-ISO would only give us a n^{n^2} which is worse than the brute-force approach (which is a $O(n!) = (cn)^n$ algorithm). We shall see a $O(n^{n^{2/3}})$ algorithm for GRAPH-ISO

28 Colourings of Graphs

A *colouring* of a graph is just associating a colour to every vertex of the graph. Formally, it is a map $f : V \longrightarrow \{1, 2, \cdots, |V|\}$. And we can further assume that the range of f is an initial segment of $\{1, 2, \cdots, |V|\}$.

A colouring f_1 is said to be a *refinement* of f, denoted by $f \leq f_1$, if $f_1(x) \leq f_1(y) \implies f(x) \leq f(y)$ for all vertices x, y. A refinement f' of f is said to be *proper* if $f' \neq f$.

28.1 Colour-Degree Refinement

For this lecture, this is the refinement that we would be dealing with. Let (X, f) be a coloured graph. The new refinement is defined as follows:

Define $g(x) = \big(f(x), k_1(x), k_2(x), \cdots, k_{|V|}(x)\big)$ where $k_i(x)$ is the number of neighbours of x that are coloured i. Since we need to map these tuples to $\{1, 2, \cdots, |V|\}$, lexicographically sort the tuples and map them to $\{1, 2, \cdots, |V|\}$. Let us call this induced colouring as f'.

The choice of the tuple, whose first coordinate is $f(x)$, it is clear that f' is a refinement of f.

One can continuously keep refining a colouring, and it spreads out vertices with more refinements. Hence, we can properly refine at most n times.

A colouring that cannot be properly refined by the colour-degree refinement is said to be a *stable colouring.*

28.2 Propagating Refinements

The central idea is to keep propagate refinements as much as possible. The problem is that the colour-degree refinements may not be able to spread the vertices enough, it might get stuck in a stable colouring much earlier.

But how does refinements help? Where are we heading?

Proposition 36. *Let (X_1, f_1) and (X_2, f_2) are two coloured graphs with f_1 and f_2 their corresponding refinements. Then*

$$(X_1, f_1) \equiv (X_2, f_2) \iff (X_1, f_1') \equiv (X_2, f_2')$$

Proof. Of course! □

The idea is to get to a stage where we can effeciently solve the problem. But, as remarked earlier, what do we do when we get stuck up at a stable colouring? We *individualize* a vertex.

- Keep refining until stable refinement
- Pick some vertex, and give it a colour that has not been assigned to any vertex.
- Repeat

The trick is to find a good a good vertex to individualize in order to propagate the refinements. We shall see that there is a small sequence of vertices, which on the 'refine until stable, individualize, repeat' gets us where we want.

29 Colour Valence and GRAPH-ISO

Definition 37. *A coloured graph (X, f) is said to have colour valence d if for all vertices x and colours i, either x is adjacent to at most d neighbours of colour i (valence of x is bounded by d) or it is not adjacent to atmost d vertices of colour i (covalence of x is bounded by d).*

The following theorem shows that there exists a small sequence of vertices that can get us to a constant colour valence.

Theorem 38. *If (X, f) has colour valence d, then there exists a sequence of nodes $\{x_1, x_2, \cdots, x_k\}$ with $k \leq 2n/d$ such that (X, f'), (colouring obtained by stabilizing and individualizing these vertices) has colour valence $d/2$.*

Proof. The proof is a greedy algorithm to pick up the vertices. Let $S_i = \{x_1, \cdots, x_{i-1}\}$. If (X, S_i) has colour valence $\leq d/2$, then we are already done, hence stop.

Else, there exists an $x \in V(X)$ and a colour m such that both valence and covalence of x in $f^{-1}(m)$ is $> d/2$. Let $N(x)$ be the neighborhood or co-neighbourhood of x (whichever violates the valence bound). Hence we have $d/2 < |N(x)| \leq d$. Pick x as x_i and continue.

Claim: The sets $\{N(x_i)\}$ are pairwise disjoint.

Once we prove the claim,

$$\frac{kd}{2} \leq \sum_{i=1}^{k} |N(x_i)| \leq n$$

which then forces $k \leq dn/2$.

Proof of claim: Suppose $N(x_j) \cap N(x_i) \neq \phi$ for some $j < i$. If m is the colour class that x_i violates, then $N(x_j) \cap f_{S_i}^{-1}(m) \neq 0$.

Now, i is a refinement further away from j, and hence would have further spread the colours of the vertices. Therefore, $N(x_j)$ has to be a union of colour classes in S_i.

Therefore, if $N(x_j) \cap f_{S_i}^{-1}(m) \neq \phi \implies f_{S_i}^{-1}(m) \subseteq N(x_j)$.

Now, since the entire colour class is contained inside $N(x_j)$, both the valence and covalence is contained in $N(x_j)$. And since each of them is atleast $d/2$, forces $|N(x_j)| > d$ contradicting the colour valence of X being bounded by d.

Hence, $\{N(x_j)\}$ are pairwise disjoint, thus proving the claim and the theorem. □

We can now start with a trivial colouring (every vertex given the same colour, and with $k \leq 8n/d$ get to (X, f') with colour valence bounded by d.

29.1 Enter Luks

Let $T(n)$ be the worst case time bound for GRAPH-ISO and $T(n, d)$ be the worst case time bound for d-colourvalence-GRAPH-ISO.

The natural algorithm is the following:

1. Put trivial colouring on both graphs

2. Refine until stable.

3. Pick the set of vertices to individualize on one graph. Try all possibilities on the other.

4. If at any try, if x was the vertex picked with valence greater than $d/2$, but valence $\leq d/2$ on the other graph, stop that try.

5. Once you get to bounded colour valence, do the corresponding algorithm.

Hence, clearly $T(n) \leq n^{8n/d}T(n,d)$.

The claim is that, Luks' algorithm for bounded degree graphs work here!

Let (X, f) be the graph with a stable colouring and colour valence d. Let $C_1, \cdots, C_r$ be the colour classes. Define $X(C_i)$ be the induced subgraph on this colour class (only edges within that class) and $X(C_i, C_j)$ as the induced bipartite graph (only edges from one class to another).

The choice of the stable colouring then forces $X(C_i)$ to be a regular graph (since the colour-degree-refinement would give a proper refinement) and also $X(C_i, C_j)$ is a semi-regular[8] graph.

Now in $X(C_i)$, either the degree is bounded by d or codegree. We can assume that it is the degree, by complementing otherwise (and checking if the other graph also has the same property, they necessarily have to if they are isomorphic). And similarly for $X(C_i, C_j)$, complement if $|E(X(C_i, C_j))| > |C_i||C_j|/2$.

(X, f) now has the property that for all vertices x and colours i, degree of x in C_i is bounded above by d. Hence we now have two graphs (X, f) and (Y, g) with the above property. As in the Luks algorithm, add a distinguished edge, and additionally a vertex on the edge with a new colour. Now layer the graphs based on edges obtained by paths from the new vertex.

Here, the kernel is again $\bigotimes \mathrm{Sym}(\cdot)$ since the colour classes can't move! And each element of the product is bounded since the degree is bounded! Hence the automorphism group is infact in $\mathcal{B}_d$ and hence can be solved by a $O(n^{d^2})$ algorithm.

Now $T(n) \leq n^{8n/d+d^2}$ and an optimal choice for $d$would give us a $O(n^{n^{2/3}})$ algorithm for GRAPH-ISO.

[8] degree of all vertices on left are the same, the same on the right

Algebra and Computation Course Instructor: V. Arvind

Lecture 11: Factorisation over finite fields

Lecturer: V. Arvind *Scribe: Kazim Bhojani and Shreevatsa R*

30 Crash course in Field theory

Definition, size is prime power, the generating function counting, cyclic,

$$X^{q^n} - X = \prod_{\substack{\deg f | n \\ f \text{ monic and irreducible in } \mathbb{F}_q[X]}} f(X) \tag{1}$$

the equality with the product of irreducible polynomials, uniqueness

31 Algorithms

We now have enough understanding of fields to look at various problems.

In general, we will be given as input a field $\mathbb{F}_q$, where $q = p^m$ for some prime number p. It is unreasonable to expect to be given $\mathbb{F}_q$ as a list of elements and addition/multiplication tables. As $\mathbb{F}_q = \mathbb{F}_p[X]/(h(X))$ for some irreducible polynomial $h(X)$ of degree m, it can be specified by p and the coefficients of $h(X)$, and that is what we are given. (This contributes $m \log p = \log q$ to the input size, and a particular element of $\mathbb{F}_q$ can be written down using $\log q$ bits).

31.1 Testing Irreducibility

Given a polynomial $f(X)$ of degree n over $\mathbb{F}_q$, we want to test whether it is irreducible.

The input needs to specify the coefficients of $f(X)$, each of which is an element of $\mathbb{F}_q$, so the input size is $(\deg f)(\log q)$.

We observe that by Equation 1, if $f(X)$ is irreducible, it must divide $X^{q^n} - X$, i.e., $\gcd(f(X), X^{q^n} - X) = f(X)$. Conversely, if $f(X)$ is not irreducible, there exists some d less than n such that $\gcd(f(X), X^{q^d} - X) \neq 1$.

Thus we have reduced testing irreducibility to finding the gcd of two polynomials, which is simply Euclid's algorithm. Note here that although the polynomials $X^{q^d} - X$ are of exponentially large degree, we only need them modulo $f(X)$, and we can easily compute X^r modulo $f(X)$ for exponential r in polynomial time by using the repeating squaring algorithm for powering.

31.2 Factorisation: Cantor–Zassenhaus algorithm

The next thing we would like to do is to actually factorise $f(X)$ into its irreducible factors. In this subsection, we describe an algorithm due to Cantor and Zassenhaus which is randomised and is in Las Vegas polytime.

Firstly, note that any repeated factors of f are factors of $\gcd(f, f')$ as well. In fact, if $f = g_1^{l_1} g_2^{l_2} \dots g_r^{l_r}$, then $\frac{f}{\gcd(f,f')} = g_1 g_2 \dots g_r$, and once we have the factorisation of the latter, we can easily find each l_i as the highest power of g_i that is present in f. So we can assume f is square-free.

Further, we can use the gcd idea (Equation 1) to separate out the irreducible factors of degree d, for every d. That is, let

$$f_1 = \gcd(f, X^q - X) \qquad f \leftarrow \frac{f}{f_1}$$

$$f_2 = \gcd(f, f, X^{q^2} - X) \qquad f \leftarrow \frac{f}{f_2}$$

and so on, then $f_1(X)$ is the product of all the linear factors, $f_2(X)$ is the product of all the quadratic factors, and in general, $f_d(X)$ is the product of all the irreducible factors of $f(X)$ of degree d. We can deal with each the f_ds separately. So we can assume $f = g_1 g_2 \dots g_r$ where all the g_i are irreducible and of (known) degree d.

By the Chinese Remainder Theorem,

$$\begin{aligned} \frac{\mathbb{F}_q[X]}{(f(X))} &\cong \frac{\mathbb{F}_q[X]}{(g_1(X))} \times \dots \times \frac{\mathbb{F}_q[X]}{(g_r(X))} \\ &\cong \mathbb{F}_{q^d} \times \dots \times F_{q^d} \end{aligned}$$

Proof of the CRT. Consider the map

$$\phi : a(X) \mapsto (a(X) \bmod g_1, \dots, a(X) \bmod g_r)$$

This is a homomorphism, is injective (the kernel is only the zero polynomial), and is surjective (the left- and right-hand sides have the same size), hence is an isomorphism. To invert it, let $G_i = \prod_{j \neq i} g_j$. There exist (u_i, v_i) such that $u_i g_i + v_i G_i = 1$, and $v_i G_i a_i \equiv a_i \pmod{g}_i$, so $a = \sum_{i=1}^r v_i g_i a_i$ is the inverse image of $(a_1, \dots, a_r)$. □

An element $(a_1, a_2, \dots, a_r)$ is a unit in $\frac{\mathbb{F}_q[X]}{(f(X))}$ (has gcd 1 with $f(X)$) iff each of the a_is is nonzero. There are a large number — $(q^d - 1)^r$ — of units, out of the $q^n = q^{dr}$ total. What we would like to get, for factorisation, are the (nonzero) zero divisors (those with nontrivial gcd with $f(X)$).

Depending on whether the characteristic of the field is odd or even, we will use different tricks to get some.

31.2.1 If q is odd

For each $x \in \mathbb{F}_{q^d}^*$, $x^{q^d-1} = 1$, and $x^{\frac{q^d-1}{2}}$ is $+1$ or -1 with probability $\frac{1}{2}$ each. This means that the map $a(X) \mapsto a(X)^{\frac{q^d-1}{2}} - 1$ takes $a(X)$ to $(\pm 1 - 1, \pm 1 - 1, \ldots, \pm 1 - 1)$, which is zero only when every "co-ordinate" is 0 and a unit only when each of them is -2, each of which happens with probability $\frac{1}{2^r}$. Thus, with probability $1 - \frac{2}{2^r}$, we get a zero divisor, whose gcd with $f(X)$ gives us a factor. (And as $1 - \frac{2}{2^r} > \frac{1}{2}$, we can repeat this until we get such a factor; this runs in Las Vegas polytime.) We can now remove this factor, test for irreducibility, and recurse.

31.2.2 If q is even

When q is even (the characteristic is 2), the above does not work as 1 and -1 are the same. q is a power of 2, say $q = 2^k$.

The mth trace polynomial is defined as

$$T_m(X) = X + X^2 + X^{2^2} + X^{2^3} + \cdots + X^{2^{m-1}}$$

Consider

$$\begin{aligned} T_m(X)(T_m(X)+1) &= T_m(X)^2 + T_m(X) \\ &= T_m(X^2) + T_m(X) && \text{characteristic 2} \\ &= X^{2^m} + X && \text{everything else occurs twice} \end{aligned}$$

Thus in $\mathbb{F}_{2^m}$, $T_m(T_m+1)$ is 0 for every element, and hence splits as $\prod_{\alpha \in \mathbb{F}_{2^m}} (x - \alpha)$. For a random element $\alpha \in \mathbb{F}_{2^m}$,

$$\Pr[T_m(\alpha) = 0] = \Pr[T_m(\alpha) = 1] = \frac{1}{2}.$$

We have

$$\frac{\mathbb{F}_q[X]}{(f(X))} \cong \mathbb{F}_{2^{kd}} \times \cdots \times \mathbb{F}_{2^{kd}}$$

so for $m = kd$, $T_m(a(X))$ is a zero divisor with probability $1 - \frac{2}{2^r}$. As before, we can get a factor, remove it, and recurse.

31.3 Factorisation: Berlekamp's algorithm

The Cantor–Zassenhaus algorithm is randomised. Berlekamp's algorithm is a deterministic algorithm, which runs in polynomial time when the size q of the field is small.

As before, we can remove repeated factors of f, so we can assume that $f = g_1 g_2 \dots g_r$ where all the g_is are distinct irreducible factors.

Now consider the map

$$\phi : \frac{\mathbb{F}_q[X]}{(f(X))} \to \frac{\mathbb{F}_q[X]}{(f(X))}$$

defined as $a \mapsto a^q - a$. This is a linear map (check).

Let $\mathcal{B} = \ker(\phi) = \left\{ a \in \frac{\mathbb{F}_q[X]}{(f(X))} \ : \ a^q = a \right\}$.

Let $\psi : \frac{\mathbb{F}_q[X]}{f} \to \frac{\mathbb{F}_q[X]}{g_1} \times \dots \times \frac{\mathbb{F}_q[X]}{g_r}$ be the isomorphism given by the Chinese Remainder Theorem. $\psi((B)) = \mathbb{F}_q \times \dots \times \mathbb{F}_q$.

We want to find $\mathcal{B}$ – that is, find a basis for it. This is easy; we can find out for each X^j its image $X^{qj} - X^j \bmod f$, and hence write down the matrix for the linear map ϕ.

Note that the elements of $\mathcal{B}$ are precisely the zero divisors. So we can sample from $\mathcal{B}$, and use the zero divisors to get factors of f, remove them and recurse, as before.

Algebra and Computation Course Instructor: V. Arvind

Lecture 12: The AKS Primality Test

Lecturer: V. Arvind *Scribe: Ramprasad Saptharishi*

32 Overview

We shall take a small detour before we go into factorising polynomials. In this class we shall look at the AKS primality test, an unconditional, deterministic polynomial time algorithm for primality testing.

33 Preliminaries

Given a number $n \in \mathbb{N}$, we wish to test whether the number is prime or not. And since n is given in binary (hence input size is $O(\log n)$), the running time is to be polynomial in $\log n$.

The AKS algorithm uses the following proposition to distinguish between primes and composites.

Proposition 39. *If* $(a, n) = 1$*, then* $(X + a)^n = X^n + a \pmod n$ *if and only if* n *is prime.*

Proof. If n is a prime, then we have already seen in the earlier class that $(X + a)^n = X^n + a^n = X^n + a \pmod n$.

Suppose n was composite and p was a prime divisor of n. Let the largest power of p that divides n be p^k. The coefficient of x^p in $(X+a)^n$ is $\binom{n}{p}a^{n-p}$. a anyway is coprime to n, and hence can be ignored. But it is easy to see that p^k does not evenly divide $\binom{n}{p}$ (since a power of p is knocked off from the n) and hence that term would survive mod$n$. □

There are however two problem with this, firstly being computing $(X + a)^n$ efficiently, but that we saw last class (using repeated squaring and the binary representation of n). A more serious problem is that n is exponential in the input size, and the polynomial would be too large to compare and check if it is $X^n + a$.

Getting around this difficulty was gave the AKS test.

34 The Primality Test

The idea to get around the difficulty of exponential degree is to check it modulo polynomials of "small" degree, a "small" number of times.

We shall give the algorithm first and then show that it is infact correct and that it runs in time polynomial in $\log n$.

Algorithm 3 AKS Primality Test:

Input: n in binary

1: Check if n is of the form a^b for $b \geq 2$. If yes, **output** COMPOSITE
2: Find the least r such that the order of n modulo r (denoted by $O_r(n)$)is at least $4\log^2 n$
3: If $(a, r) \neq 1$ for $1 \leq a \leq r$, **output** COMPOSITE
4: If $n < r$, **output** PRIME
5: **for** $a = 1$ to $\left\lfloor 2\sqrt{\phi(r)}\log n \right\rfloor$ **do**
6: **if** $(X + a)^n \neq X^n + a \pmod{n, X^r - 1}$ **then**
7: **output** COMPOSITE
8: **end if**
9: **end for**
10: **output** PRIME

Apart from step 2, it is clear that the algorithm will run in time polyomial in the input length. As for step 2, the following lemma would tell us that we can indeed find such an r quickly.

Lemma 40. *If m is an odd number, then the lcm of $1, 2, \cdots, m$ is atleast 2^{m-1}.*

Proof. Let $m = 2n + 1$. Consider the following integral:

$$\int_0^1 x^n(1-x)^n dx$$

Since $x(1-x) < 1/4$, this integral is upper bounded by 2^{2n}. But if we were to expand $(1-x)^n$ using the binomial theorem, we have

$$\begin{aligned} 2^{-2n} \leq \int_0^1 x^n(1-x)^n dx &= \int_0^1 \sum_{k=0}^{n} (-1)^k \binom{n}{k} x^{n+k} \\ &= \sum_{k=0}^{n} (-1)^k \binom{n}{k} \frac{1}{n+k+1} = \frac{M}{N} \end{aligned}$$

Clearly, the denomiantor is atmost the lcm (L) of the numbers $1, 2, \cdots 2n+1$ and M is atleast 1. Hence $L2^{-2n} > N2^{-2n} \geq 1$ and hence $L \geq 2^{2n}$. $\square$

Lemma 41. *There exists an $r \leq 16 \log^5 n$ such that $O_r(n) \geq 4 \log^2 n$.*

Proof. Suppose all r's till T were bad, that is, for all $1 \leq r \leq T$ the order of n modulo r was less than $4 \log^2 n$. Then, for every r there exists a $j < 4 \log^2 n$ such that r divided $n^j - 1$. Therefore, each $1 \leq r \leq T$ divides the product $\prod_{j=1}^{4 \log^2 n}(n^j - 1) < n^{16 \log^4 n} = 2^{16 \log^5 n}$. And therefore, the lcm of the first T numbers divide the product. The bound from the earlier lemma will now force $T < 16 \log^5 n$. $\square$

It is now clear that the algorithm runs in time polynomail in $\log n$, each step is clearly a polylog operation. With some work, one can see that this is roughly a $O(\log^{11} n)$ algorithm.

35 Proof Correctness

One way is clear, if the number was a prime then the algorithm would certainly output PRIME. We need to show that if the algorithm outputs PRIME, then it is indeed prime.

Suppose not, let p be a prime divisor of n. And from our initial tests, we know that $p > r$. And further for $a = 1, 2, \cdots l$ where $l = \left\lfloor 2\sqrt{\phi(r)} \log n \right\rfloor$,

$$(X + a)^n = X^n + a \pmod{n, X^r - 1} = X^n + a \pmod{p, X^r - 1}$$

Note that from Fermat's little theorem, we have

$$(X + a)^p = X^p + a \pmod{p, X^r - 1}$$

We shall use a small notation here.

Definition 42. *For any function f, a number m is called introspective for f if $f(X)^m = f(X^m) \pmod{p, X^r - 1}$.*

We then have the two simple lemmas.

Lemma 43. *If m and m' are introspective for f, so is mm'.*

Proof.

$$\begin{aligned}
f(X)^m &= f(X^m) \pmod{p, X^r-1} \\
\implies f(X^{m'})^m &= f(X^{mm'}) \pmod{p, X^{m'r-1}} \quad (\text{substitute } X^{m'} \text{ for } X) \\
&= f(X)^{mm'} \pmod{p, X^r-1} \quad (X^r-1 \text{ divides } X^{rm'}-1) \\
\implies f(X)^{mm'} &= f(X^{mm'}) \pmod{p, X^r-1}
\end{aligned}$$

□

Lemma 44. *If m is introspective for f and g, then m is introspective for fg.*

Proof. Obvious! □

Let $I = \left\{n^i p^j \ : \ i \geq 0, j \geq 0\right\}$ and $P = \left\{\prod_{a=1}^{l}(X+a)^{e_a} \ : \ e_a \geq 0\right\}$. Then, from the two lemmas, every element of I is introspective for every element in P.

Let G be the subgroup of $\mathbb{Z}_r^\star$, of size t, generated by n and p. Since $O_r(n) \geq 4\log^2 n$, $|G| = t \geq 4\log^2 n$. To do a similar thing for the polynomials, we first need to move from the ring $(\mathbb{F}_p/(X^r-1))$ to a field. Let $X^r-1 = h_1(X)h_2(X)\cdots h_k(X)$ be the factorization into irreducible factors. Since the primitive r-th root of unity generates all the roots of unity (since we are going mod X^r-1), the primitive root of unity is a root of some irreducible polynomial. Hence we shall look at $\mathbb{F}_r/(h(x))$, which is essentially $\mathbb{F}_r[\eta]$ where η is a primitive r-th root of unity.

Now in the field $\mathbb{F}_r/(h(x))$, look at the multiplicative group. Let $\mathcal{G}$ be the subgroup generated by $\{(X+a)\}_{1\leq a\leq l}$, the set of polynomials in P that are non-zero in $\mathbb{F}_p/(h(x))$.

Lemma 45. $|\mathcal{G}| \geq \binom{t+l-2}{t-1}$

Proof. Since $l = \left\lfloor 2\sqrt{\phi(r)}\log n \right\rfloor < 2\sqrt{r}\log n < r < p$, each of $\{(X+a)\}_{1\leq a\leq l}$ are distinct in $\mathbb{F}_p/(h(x))$. At worst $h(x)$ can be equal to one of them, hence at least $l-1$ of them are non-zero and distinct.

Let f and g be two polynomials from P of degree less than t. Suppose $f(x) = g(x)$ in $\mathbb{F}_p/(h(x))$, then we also have $f^m(x) = g^m(x)$ in the field. If we choose $m \in G$, then since m is introspective for f and g, we have $f(X^m) = g(X^m)$ in the field.

Since we can identify X with a primitive r-th root of unity, each of the X^m are distinct. And infact, each of them is a root of the polynomial $H(Y) = f(Y) - g(Y)$. The size of the group G is t but the degree of f and

g is less than t, which gives an absurd situation of the polynomial $f - g$ having more roots than its degree in the field.

Hence, if two polynomials of degree less than t are chosen from P, they are mapped to different elements in $\mathbb{F}_p/(h(x))$.

$$|\mathcal{G}| \geq \left| \left\{ \prod_{1 \leq a \leq l} (X+a)^{e_a} \ : \ \sum e_a \leq t \right\} \right|$$

As we remarked earlier, there can be at most one $X + a_0$ that becomes zero in the field. So essentially, we just need to find the different integer solutions to $\sum e_a \leq t$, summing over all $a \neq a_0$, and this is equal to $\binom{t+l-2}{t-1}$. Hence $|\mathcal{G}| \geq \binom{t+l-2}{t-1}$. □

Lemma 46. *If n is not a power of a prime, $|\mathcal{G}| < n^{2\sqrt{t}}$*

Proof. Consider the subset $I' = \left\{ n^i p^j \ : \ 0 \leq i, j \leq \sqrt{t} \right\}$. Hence each element in I' is bounded by $n^{\sqrt{t}} p^{\sqrt{t}} < n^{2\sqrt{t}}$. And further, if n is not a power of a prime, $|I'| = (1+\sqrt{t})^2 > t$. Since $|G| = t$, there exists two distinct m, m' of I' such that $m = m' \pmod{r}$, and hence $X^m = X^{m'} \pmod{r, X^r - 1}$. Also, for any $f \in \mathcal{G}$, $f(X)^m = f(X)^{m'}$. Therefore, if you consider the polynomial $Y^m - Y^{m'}$, every element of $\mathcal{G}$ is a root. And since the degree is bounded by $n^{2\sqrt{t}}$, this forces $|\mathcal{G}| < n^{2\sqrt{t}}$. □

With the choice of l, it is easy to see that the above two lemmas give conflicting bounds (lower bound greater than upper bound), which gives us the desired contradiction to the assumption that n is composite.

Hence, summarizing it in a theorem:

Theorem 47. *The algorithm returns* PRIME *if and only if the input is a prime.* □

36 A short note on identity testing

I haven't taken notes here... would be nice if someone could fill this part.

Algebra and Computation Course Instructor: V. Arvind

Lecture 13: More on Univariate Factorization over $\mathbb{F}_q$

Lecturer: V. Arvind *Scribe: Ramprasad Saptharishi*

37 Overview

(the initial part of this lecture has been appended to lecture 11 *for continuity.)*

In the earlier lectures we saw univariate polynomial factoring when the characteristic of the field is small. Before we get into bivariate polynomial factoring (which has all the essentials needed for general multivariate factoring), we shall look at some problems closely related to the things we have seen.

38 Factoring $\leq_P$ Root-Finding

In the earlier lectures, we saw factorization of univariate polynomials over a finite field of small characteristic. What about the case when p is large, polynomially many bits long? In this section, we shall show that the problem can be reduced that of finding a root of a polynomial over the prime field $\mathbb{F}_p$.

The polynomial has an additional promise that it splits over $\mathbb{F}_p$ (all its roots are contained in $\mathbb{F}_p$). We will show that finding a non-trivial factor of a polynomial reduces to finding a root of such a polynomial over the prime field. The rest of the section will be a proof of this theorem.

Theorem 48. *Given $f \in \mathbb{F}_q[x], q = p^m$, we can, in polynomial time, reduce it to the problem of finding a root of a polynomial $g \in \mathbb{F}_p[x]$ with the promise that all its roots are in $\mathbb{F}_p$.*

The following simple proposition is the core of the reduction.

Proposition 49. *For any $f, g \in F[x]$, polynomials over some field,* $\gcd(f, g) \neq 1$ *if and only if there exists $s, t \in F[x]$ such that* $\deg s < \deg f, \deg t < \deg g$ *and $fs + gt = 0$.*

Proof. Obvious! (take $s = f/\gcd(f, g)$ and $t = -g/\gcd(f, g)$) □

$P_m = \{s \in F[x] \ : \ \deg s < m\}$ and $P_n = \{t \in F[x] \ : \ \deg t < n\}$, the set of all possible s and t for the Let the $\deg f = n$ and $\deg g = m$. Look at the following sets proposition; they form a vector space over F. Once we fix f and g, we have the following linear map:

$$\begin{array}{rcl} \theta : P_m \times P_n & \longrightarrow & P_{n+m} \\ (s,t) & \mapsto & sf + gt \end{array}$$

The proposition tells us that the above linear map is invertible if and only if $\gcd(f,g) = 1$. Let us fix a basis for the two spaces so that we can find the matrix for the linear map.

The natural basis for $P_m \times P_n$ is

$$\left\{(X^{m-1},0),(X^{m-2},0),\cdots,(1,0)\right\} \cup \left\{(0,X^{n-1}),(0,X^{n-2}),\cdots,(0,1)\right\}$$

and that for P_{m+n} as just

$$\left\{X^{m+n-1}, X^{m+n-2}, \cdots, 1\right\}$$

Suppose $f = f_0 + f_1 x + f_2 x^2 + \cdots f_n x^n$ and $g = g_0 + g_1 x + \cdots g_m x^m$, it is easy to see that the matrix for θ is the following:

$$\mathcal{S} = \begin{bmatrix} f_n & 0 & \cdots & 0 & g_m & \cdots & 0 \\ f_{n-1} & f_n & \cdots & & g_{m-1} & \ddots & \vdots \\ f_{n-2} & f_{n-1} & \ddots & \vdots & \vdots & \vdots & g_m \\ \vdots & \vdots & \cdots & f_n & \vdots & \ddots & \vdots \\ \vdots & \vdots & \cdots & \vdots & g_0 & \vdots & \vdots \\ f_0 & f_1 & \cdots & \vdots & 0 & \vdots & \vdots \\ 0 & f_0 & \cdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & f_0 & 0 & \cdots & g_0 \end{bmatrix}_{(m+n)\times(m+n)}$$

The matrix is called the *Sylvester Matrix*, named after the mathematician. The determinant of the matrix is called the resultant of f, g (denoted by $Res(f,g)$). The proposition tells us that $Res(f,g) = 0$ if and only if $\gcd(f,g) \neq 1$.

From Berlekamp's algorithm, if $a \in \ker \phi$ (the Berlekamp kernel), then $f = \prod_{\alpha \in \mathbb{F}_p} \gcd(f, a - \alpha)$. We want to convert this to a polynomial with roots

whenever $\gcd(f, a-\alpha) \neq 1$. The idea is to look at α as an indeterminate. Now $Res_x(f, a-Y)$ will now be a polynomial in Y with a root α whenever $\gcd(f, a-\alpha) \neq 1$.

The roots we are looking for are certainly in $\mathbb{F}_p$, but we need the extra property that it infact splits in $\mathbb{F}_p$. But that can be done easily, all that we need to do is take $P(Y) = \gcd(Y^p - Y, Res_x(f, a-Y))$ since the polynomial $Y^p - Y$ as $\prod_{\alpha \in \mathbb{F}_p}(Y-\alpha)$.

Clearly, whatever we have done is a polynomial time computation (determinant by just gaussian elimination, gcd trick as done in the earlier lectures, finding an element in berlekamp kernel by gaussian elimination, etc) and reduces finding a non-trivial factor to finding a root of a special polynomial.

39 How to Share a Secret

This problem arises in the context of secure multiparty computations. Informally, the problem is the following: You have a secret x chosen randomly from a set $\mathcal{S}$ of secrets and it is to be split up into n parts and given to n people (one piece each). You want the pieces to satisfy the following property that the secret can be found if and only if all of them get together.

Formally, $\mathcal{S} \subseteq \mathbb{F}_q$. An element χ chosen from $\mathcal{S}$ uniformly at random. There are n people ($n < q$), and the problem is to assign each of them pieces such that

- For any proper subset of shares, no information about χ is lost.
- If all of them are together, χ can be found.

The following scheme, Shamir's Secret Sharing Scheme, is a beautiful application of the chinese remaindering theorem.

Choose a polynomial $a(x) = a_0 + a_1 x + \cdots a_{n-1}x^{n-1}$ such that $a_0 = \chi$ and every other a_i is chosen at random from $\mathbb{F}_q$. Pick a polynomial $f(x) = (x-\alpha_1)(x-\alpha_2)\cdots(x-\alpha_n)$ where the α_is are distinct non-zero elements of $\mathbb{F}_q$. By the chinese remaindering theorem, we the isomorphism

$$\psi : \frac{\mathbb{F}_q[x]}{(f(x))} \longrightarrow \frac{\mathbb{F}_q[x]}{(x-\alpha_1)} \times \cdots \frac{\mathbb{F}_q[x]}{(x-\alpha_n)}$$

Thus ψ would send $a(x)$ to the tuple $\langle a(\alpha_1), \cdots, a(\alpha_n)\rangle$. Give the i^{th} person $a(\alpha_i)$.

It is clear that when everyone gets together, they can invert the map ψ and recover a and hence $a(0) = \chi$. We need to argue that no proper subset can be able to recover any information about χ. Without loss of generality, we can assume that the first $n-1$ get together, and say they recover $\langle a_1, \cdots, a_{n-1}, ? \rangle$.

The set of possible polynomials is

$$A = \left\{ a(x) \in \frac{\mathbb{F}_q[x]}{(f)} \ : \ a(x) \mapsto \langle a_1, a_2, \cdots, a_{n-1}, ? \rangle \right\}$$

If $a(x)$ was the actual polynomial, it is clear that $A = a(x) + S$ where

$$S = \left\{ c(x) \in \frac{\mathbb{F}_q[x]}{(f)} \ : \ c(x) \mapsto \langle 0, \cdots, 0, ? \rangle \right\}$$

The set S is precisely the set of polynomials $c = \alpha \prod_{i=1}^{n-1}(x - \alpha_i)$ where α can be any scalar. And it is easy to see that $c(0) = \alpha \prod_{i=1}^{n-1}(-\alpha_i)$ which can again take any possible value in $\mathbb{F}_q$ and hence $a(0) + c(0)$ also takes all possible values in $\mathbb{F}_q$, thus reveals nothing about χ.

40 Towards Bivariate Factoring

We shall next look at bivariate factoring. This does not mean we would look at trivariate factoring next; bivariate factoring contains all the essentials of multivariate factoring.

Before we go into it, we need a crash course in ring theory, definitions and useful theorems at least.

40.1 A Crash Course in Ring Theory

1. Ring: A set R with two operations $\star, +$ such that $(R, +)$ is an abelian group, and $\star$ distributes over $+$. We'll denote $a \star b$ by just ab.

2. Zero divisors: Elements $x \in R$ such that there exists a y such that $xy = 0$.

3. Integral Domains: Commutative rings with a multiplicative identity element called 1 that does not have any non-trivial zero divisors.

4. Irreducible elements in an integral domain: An element x is said to be irreducible if $x = yz$ implies either y or z is a unit.

5. Prime elements in an integral domain: $p \mid ab \implies p \mid a$ or $p \mid b$.

6. The field of fractions of an integral domain: The field of formal fractions, ordered pairs (x, y) interpretted as x/y with addition and multiplication defined naturally.

7. Unique Factorization Domain: An integral domain where factorization into irreducible factors is unique (up to units and rearrangements)

8. Ideal: A subset $I \in R$ such that it is a subgroup under $+$, and satisfies the property that for all $x \in I$ and $r \in R$, $rx \in R$. They are useful for defining homomorphism (kernel of any ring homomorphism is an idea) and quotient rings.

9. Ideal generated by a $a_1, \cdots, a_n$: The smallest ideal containing $a_1, \cdots, a_n$. This is just the set of all elements of the form $\sum_{r_i \in R} r_i a_.$

10. Prime Ideal: An ideal satisfying the property that $xy \in I \implies x \in I$ or $y \in I$.

Some properties:

Fact 1. *prime* $\implies$ *irreducible*

The converse, however, is not true in general. For example, consider $\mathbb{Q}[X^2, X^3]$. There X^2 is irreducible, but not prime since $X^2 \mid X^3 \cdot X^3$. Nevertheless, they are the same over an UFD.

Fact 2. *If R is a UFD, prime $\Leftrightarrow$ irreducible.*

Fact 3. *In an integral domain, if I is a maximal ideal, the quotient R/I is a field.*

Fact 4. *If I is a prime ideal of R, then R/I is an integral domain.*

Fact 5. *If R is a UFD, then so is $R[x]$.*

The last fact is an important theorem. In particular, since any field is a UFD, $F[X_1, \cdots, X_n]$ is a UFD.

40.2 Towards Bivariate Factorization

The idea is to look at $F[x, y]$ as $F[x][y]$, thinking of $F[x, y]$ as a univariate polynomial extension over $F[x]$ through the variable y. The question is that, can we somehow use the univariate factorization in this context? If we can factorize efficiently in $F[x]$, can we do that in $F[x][y]$ through some bootstrapping?

One possibility is trying to substitute values and factorize, but that would case factors that were initially irreducible to split after substitution. Is there a way by which we can get around this difficulty? We shall explore these problems in the following lecture.

Algebra and Computation	Course Instructor: V. Arvind
Lecture 14: Bivariate Factorization	
Lecturer: V. Arvind	*Scribe: Ramprasad Saptharishi*

41 Overview

In the next two lectures, we shall discuss bivariate factorization. We shall look at the major parts of the algorithm, and fill up the missing ends next class.

42 The Idea

We saw last time that for any field F, $F[x, y]$ is a unique factorization domain. And since we know how to factorize univariate polynomials, thinking of $F[x, y]$ as $F(y)[x]$ might be useful.

Assume that f is square-free and that $f(x, 0)$ is square-free as well. Since $f(x, 0)$ is a polynomial in x alone, we know to factorize it. Suppose, $f(x, 0) = g_0(x, 0)h_0(x, 0)$, this can be thought of as a factorization $\pmod y$, i.e

$$f(x, y) = g_0(x, 0)h_0(x, 0) \pmod y$$

The questions now are, can we lift this to a factorization modulo higher powers of y? After we lift it sufficiently, would be able to clean up to get the actual factor of f?

43 Hensel Lifting

The following lemmas form the core of the algorithm. The following notation would make things simpler.

Definition 50. *For two elements $f, g \in R$, and I an ideal of R, we say the pseudo-gcd of f, g is $1 \pmod I$ if there exists $a, b \in R$ such that*

$$af + bg = 1 \pmod I$$

Lemma 51 (Hensel's Lifting Lemma)**.** *Let R be an arbitrary commutative ring with identity with an ideal I. If $f \in R$ can be written as $f = gh \pmod I$*

such the pseudo-gcd of g and h is 1 modulo I, we can lift this factorization in the following sense: There exists g' and h' and a', b' such that

$$\begin{aligned} f &= g'h' \pmod{I^2} \\ a'g' + b'h' &= 1 \pmod{I^2} \\ g' &= g \pmod{I} \\ h' &= h \pmod{I} \end{aligned}$$

And further, the following also holds

- *Given a, b, g, h, we can easily compute a', b', g', h'.*
- *The solution g', h' is unique in the sense that if g'' and h'' also satisfy the equations, then*

$$\begin{aligned} g'' &= g'(1+u) \pmod{I^2} \\ h'' &= h'(1-u) \pmod{I^2} \end{aligned}$$

  for some $u \in I$.

Proof. Define $g' = g + bm$ and $h' = h + am$, where $f - gh = m \pmod{I^2}$. Now,

$$\begin{aligned} f - g'h' &= f - (g + bm)(h + am) \\ &= f - gh + m(ag + bh) \pmod{I^2} \\ &= m(1 - (ag + bh)) \pmod{I^2} \\ &= 0 \pmod{I^2} \end{aligned}$$

As for the pseudo-gcd, let $a' = a + am'$ and $b = b + bm'$ where $m' = 1 - (ag' + bh') \in I$.

$$\begin{aligned} a'g' + b'h' &= (a + am')g' + (b + bm')h' \pmod{I^2} \\ &= (ag' + bh') + m'(ag' + bh') \pmod{I^2} \\ &= 1 - m' + m'(1 - m') \pmod{I^2} \\ &= 1 - m' + m' - m'^2 \pmod{I^2} \\ &= 1 \pmod{I^2} \end{aligned}$$

Now for the uniqueness, suppose g'' and h'' also satisfy the equations, and hence let $g'' - g' = m_1 \in I$ and $h'' - h' = m_2 \in I$. Let $u = m_1 a' - m_2 b' \in I$ since both m_1 and m_2 are in I.

$$\begin{aligned}
f = g''h'' &= g'h' \pmod{I^2} \\
(g' + m_1)(h' + m_2) &= g'h' \pmod{I^2} \\
\implies m_1h' + m_2g' &= 0 \pmod{I^2} \\
\implies m_2g' &= -m_1h' \pmod{I^2} \\
a'm_2g' &= -a'm_1h' \pmod{I^2} \\
m_2(1 - b'h') &= -m_1a'h' \pmod{I^2} \\
m_2 &= h'(m_2b' - m_1a') \pmod{I^2} \\
m_2 &= h'(-u) \\
\implies h'' &= h'(1 - u)
\end{aligned}$$

and similarly for g''. □

In our context, when $R = F[x, y]$ and $I = \langle y^k \rangle$, if we force g to be monic in x, then we can force the u in the above lemma to be zero.

Lemma 52. *When $R = F[x, y]$ and $I = \langle y^k \rangle$, if $f = gh \pmod{y^k}$ such that g is monic in x. Then we can hensel lift this to g', h' such that the conditions hold and that g' is monic. Infact, g' is unique modulo y^{2k}.*

Proof. By the earlier lemma, there exists a lifting $f = g'h' \pmod{y^{2k}}$. And since $g' - g$ is a multiple of y^k, let $a = (g' - g)/y^k$. Since g is monic in x, we can apply the division algorithm to divide a by g to obtain

$$a = gq + r$$

with $\deg_x r < \deg_x g$. Now let $g_0 = g + ry^k$, another monic polynomial. It is easy to see that $g_0 = g'(1 + u)$ where $u = -y^k qg' \in I$ and hence $g_0, h_0 = h'(1 - u)$ is a solution wiht a monic g_0.

As for uniqueness, any solution looks like $g'' = g'(1 + vy^k)$ for some v, and hence the only way g'' can be monic is when v is zero, and hence g' is unique. □

44 The Factoring Algorithm

1. Preprocess such that f and $f(x, 0)$ are square free. Let the total degree of f be d.

2. Using the univariate factoring algorithm, factorize

$$f(x,y) = g_0(x,y)h_0(x,y) \pmod{y}$$

where g_0 is monic in x and irreducible.

3. Do a hensel lift k times where k is chosen such that $2^k > 2d^2$. Let $f(x,y) = g_k(x,y)h_k(x,y) \pmod{y^{2^k}}$

4. Solve, as a system of linear equations, for

$$g' = g_k(x,y)l_k(x,y) \pmod{y^{2^k}}$$

where $\deg_x g' < \deg_x f$ and $\deg_y l_k, \deg_y g' \leq \deg_y f$.

5. Compute $\gcd_x(f, g')$, as polynomials in $F(y)[x]$, and find a non-trivial factor using Gauss's Lemma if the gcd is non-trivial.

We need to argue that step 4 will have a non-trival solution, and also that hence 5 will happen.

44.1 Step 4 will have a non-trivial solution

Note that when we start with $f = g_0h_0 \pmod{y}$, g_0 need not correspond to a factor of f but will certainly divide a factor modulo y. Let this irreducible factor was called g, and $f = gh$ in $F[x,y]$. Then $g = g_0l_0 \pmod{y}$, where g_0 is monic in x. Hensel lift this k times to obtain $g = g'_k l'_k \pmod{y^{2^k}}$ with $g_0 = g'_k \pmod{y^{2^k}}$ and g'_k monic. We will show that $g'_k = g_k$, the polynomial got by hensel lifting $f = g_0h_0$ for k times.

$$f = gh = g'_k l'_k h = g'_k h' \pmod{y^{2^k}}$$

where $h' = l'_k h \pmod{y^{2^k}}$. But since g'_k is also monic, the hensel lifting is unique and hence $g'_k = g_k$. Thus, $g' = g$ and $l_k = l'_k$ form a non-trivial solution to step 4. □

44.2 Step 5 will happen

Suppose $\gcd_x(f, g') = 1$, then there exists polynomials u, v in $F(x)$ such that

$$uf + vg' = 1$$

Note that the u, v are from $F(x)$, elements from the fraction field. We shall now see how to clear the denominators. Recall that the Sylvester

matrix is the transformation that takes (s,t) to $sf+tg'$. Hence The sylvester matrix would take (u,v) to $uf+vg'=1$. Now, we can use Cramer's rule to hence solve for $\mathcal{S}(u,v)^T=(0,0,\cdots,0,1)^T$. And notice that the denominator for each coordinate in (u,v) would be $Res_x(f,g')$ and hence multiplying each coordinate by that value would completely clear the denominators.

Hence, we can get

$$u'f+v'g=Res_x(f,g')$$

where $u',v'\in F[x,y]$. Going mod y^{2^k}, we have

$$Res_x(f,g')=u'g_kl_k+v'g_kh_k=g_k(u'l_k+v'h_k)$$

Note that the left hand side is of degree atmost $2d^2$ by the choice of k and would remain unchanged when we go modulo y^{2^k}. But the right hand side on the other hand, g_k is monic. The resultant is a polynomial in y alone, and hence the only way the top x in g_k can be killed is when $u'l_k+v'h_k=0$ but that would force $Res_x(f,g')=0$, which contradicts the assumption that $\gcd(f,g')=1$.

Hence step 5 would give us a non-trivial factor of f. □

45 Missing Pieces

The algorithm relies on the assumption that $f=gh$ where g and h are coprime, similarly $f=g_0h_0 \pmod{y}$. Hence, we need the assumption that f and $f(x,0)$ are square free.

We shall fill in these missing ends in the next lecture, and also some interpretations of Hensel Lifting to Newton's method for finding roots.

Algebra and Computation Course Instructor: V. Arvind

Lecture 15: Bivariate Factorization: Missing Pieces

Lecturer: V. Arvind *Scribe: Ramprasad Saptharishi*

46 Overview

Last class we did bivariate factorization, but we made some assumptions in the beginning. The hope was that with some preprocessing, the assumptions can be guarenteed. This class we shall see what those preprocessing steps are.

After that, we shall discuass a Hensel Lifting take on Newton's root finding algorithm.

47 The Missing Pieces

The algorithm relies on the assumption that the factorization of f and $f(x,0)$ is square free since we want the pseodo-gcd of factors to be 1. We need to make sure that we can pull out repeated factors in the beginning.

47.1 f is square free

In the univariate case, this was trivial since we just had to take the derivative and do it. Multivariate cases are a little tricky. The first step is to remove the *content* of each variable from the polynomial.

Think of the polynomial f as one over $F[y][x]$, a univariate polynomial with coefficients coming from $F[y]$. The y-content of f is defined as the gcd of the coefficients of the polynomial when considered as one in $F[y][x]$.

The x-content and y-content are clearly factors of f and hence we can factorize them using univariate factorization. Hence we can assume that

$$f = f_1^{e_1} f_2^{e_2} \cdots f_k^{e_k}$$

where each f_i is an irreducible factor with x-content and y-content being 1. Let us look at this as $f = f_1^e h$. Then,

$$\frac{\partial f}{\partial x} = e f_1^{e-1} h \frac{\partial f_1}{\partial x} + f_1^e \frac{\partial h}{\partial x}$$

Suppose both $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ are zero, then the only way this can happen if each power of x and y is a multiple of p. Hence $f(x,y) = g(x^p, y^p)$ and this can be checked easily and it now just amounts to factorizing g.

We can now assume without loss of generality that $\frac{\partial f}{\partial x}$ is non-zero. Now suppose that $\frac{\partial f_1}{\partial x}$ was non-zero, then clearly from the above equation the largest power of f_1 that divides $\frac{\partial f}{\partial x}$ is f^{e-1}.

Let $u = \frac{\partial f}{\partial x}$, $v = \frac{\partial f}{\partial y}$, $u' = f/\gcd(f,u)$ and $v' = f/\gcd(f,v)$, whenever they are non-zero. The bad news is that, since some of the $\frac{\partial f_i}{\partial x}$ could be zero, it misses out the factors that are x^p polynomials. The good news is that, these are the only things that u' and v' would miss.

Hence, factorize u' and v', then divide f by the collection of factors. We are then assured that the remaining factors have to be polynomials of x^p and y^p. We can them make the transformation and recurse.

Thus, we can ensure that f does not have any repeated factors.

47.2 $f(x,0)$ is square free

Though we have $f(x,y)$ to be square free, substituting 0 for y would case certain factors to collapse; $f(x,0)$ could have repeated roots. The trick is to make a small change of variables to ensure that it is square free.

Replace $f(x,y)$ by $f_\beta(x,y) = f(x, y+\beta)$. We need to show that there exists a β such that $f(x,\beta) = f_\beta(x,0)$ is square free.

If $f' = 0$, then reverse the roles of x and y (if both are zero, then it is a polynomial of x^p and y^p). Note that $\gcd(f,f') \neq 1$ if and only if $Res_x(f,f') = 0$. And since the resultant is a polynomial of degree $2d^2$, this can have atmost $2d^2$ roots of F. Hence if $|F| > 2d$, we can just substitute $2d^2+1$ values for y and we would get a polynomial where the residue is non-zero, and thus $f_\beta(x,0)$ would be square free.

Hence, all that's left to do is the case when $|F| \leq 2d^2$. The trick is to go to a larger field and work there. Suppose $F = \mathbb{F}_q$, choose a prime t such that $q^t > 2d^2$ and $t > \deg f$. Replace F by $\mathbb{F}_{q^t}$ (just find an irreducible polynomial of degree t and work in $\mathbb{F}_q$ modulo that polynomial). In this larger field, the irreducible factors could split even further.

$$f = f_1' f_2' \cdots f_{k'}'$$

where bunches of these factors correspond to the original factors. To study these bunches, we need an important map known as the *Frobenius map.*

$$\begin{aligned} \sigma : \mathbb{F}_{q^t} &\longrightarrow \mathbb{F}_{q^t} \\ a &\mapsto a^q \end{aligned}$$

Note that the map fixes every element of $\mathbb{F}_q$ pointwise, and is an automorphism. This can be naturally extended to the ring $\mathbb{F}_q[x, y]$.

And since $f_1 \in \mathbb{F}_q[x, y]$, the frobenius map will fix it. We are interested in finding the bunch of f_i' that correspond to f_1. Suppose $f_1' \mid f_1$, then by the automorphism, $\sigma(f_1') \mid f_1$, $\sigma^2(f_1') \mid f_1$ and so on.

Since $\sigma^t(f_1') = f_1'$, for any r such that $\sigma^r(f_1') = f_1'$ will force r to divide t. Since t is chosen to be a prime, either $r = t$ or $r = 1$. If $r = t$, then each of the t elements of the form $\sigma^i(f_1')$ would be a factor of f_1. But since $t > \deg f$, all of them cannot fit inside f.

Hence $r = 1$, and thus the factorization does not fit further in $\mathbb{F}_{q^t}$. We can now hunt for a β here to make it square free.

48 Hensel Lifting and Newton Rhapson

Suppose we are given a polynomial $f(x) \in \mathbb{Z}[x]$, we want to find a root of f efficiently by successive approximations. We shall do this using Hensel lifting.

Pick a small prime p such that $f(x)$ is square free.

$$\begin{aligned} f(x) &= f_0 + f_1 x + f_2 x^2 + \cdots f_n x^n \\ f(x+h) &= \sum_{i=1}^{n} f_i (x+h)^i \\ &= \sum_{i=1}^{n} f_i (x^i + ihx^i - 1 + \cdots) \\ &= f(x) + hf'(x) + h^2 P(x, h) \end{aligned}$$

Now using Berlekamp's algorithm, find an x such that $f(x) = 0 \pmod p$. Suppose there exists an $\hat{x}$ such that $\hat{x} = x \pmod p$ and $f(\hat{x}) = 0$ then $\hat{x} = x + ap$. And hence

$$\begin{aligned} f(\hat{x}) &= f(x) + apf'(x) + a^2 p^2 P(x, ap) \\ \implies 0 = f(\hat{x}) &= f(x) + apf'(x) \pmod{p^2} \end{aligned}$$

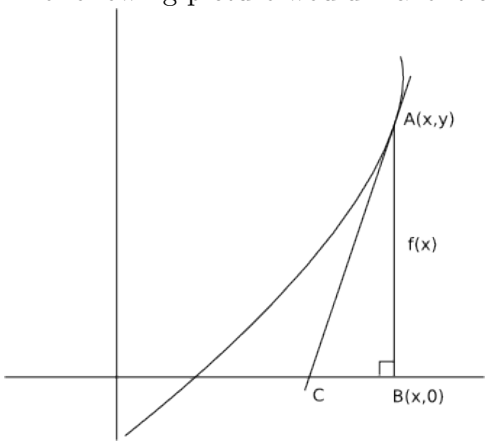
Since $f(x) = 0 \pmod{p}$, it makes sense to talk about $(f(x)/p)$. Thus, if we were to choose $a = (-f(x)/p)\left[f'(x)\right]^{-1}$, the above equation would be satisfied.

$$\begin{aligned} a &= \left(\frac{-f(x)}{p}\right)\left[f'(x)\right]^{-1} \pmod{p} \\ \implies \hat{x} &= x - f(x)\left[f'(x)\right]^{-1} \pmod{p} \end{aligned}$$

Thus, from a factorization modulo p, we have gone up to p^2 with $\hat{x}$ as our next approximation.

Newton-Rhapson also has the similar expression. You are given a function f, you choose a random point x. The next approximation is given by drawing the tangent to the curve f at $(x, f(x))$ and taking the point where this tangent meets the x-axis as its next approximation.

The following picture would make it clear.



If the coordinate of C was $\hat{x}$, our next approximation,

$$\begin{aligned} f'(x) &= \frac{f(x)}{x - \hat{x}} \\ \implies \hat{x} &= x - \frac{f(x)}{f'(x)} \end{aligned}$$

which is exactly what we got in the Hensel Lifting method.

Newton's method however require floating point arithmetic (since division by $f'(x)$ is actual division, unlike inverse modulo p in the hensel lifting case), while it enjoys the ease of not having to find the inverse modulo a number.

Algebra and Computation Course Instructor: V. Arvind

Lecture 16 and 17: Linear Diophantine Equations

Lecturer: V. Arvind *Scribe: Ramprasad Saptharishi*

49 Overview

Our route now is towards factorization of polynomials over $\mathbb{Q}$. This require a lot of machinery to be built and we shall do it over the next few lectures.

In this class, we shall look at solving a system linear diophantine equations and its connection to lattices.

50 Linear Diophantine Equations

A linear diophantine equation is of the form $a_1x_1 + a_2x_2 + \cdots a_nx_n = b$ and we are interested in integer solutions $\{x_i\}$. A system of linear diophantine equations is a bunch of such equations. This can be written in a matrix notation as follows:

Given a rational $m \times n$ matrix (matrix with rational entries) A, and a rational m-vector b, we are looking for integral vectors x that satisfy $Ax = b$.

We are looking for a polynomial time algorithm to give us all possible solutions to this equation. Getting all solutions is simple once we have a single solution $\hat{x}$. All we need to do get the solution space $\mathcal{S}$ to $Ax = 0$ and and the solutions to the diophantine system is just $\hat{x} + \mathcal{S}$.

Firstly, we can assume that A is of full rank (row rank is equal to m) since even otherwise we can drop the other rows since they are linear combinations of the independent rows. Another thing we can assume is that the entries are integral (we can just scale the matrix up by the LCM of the denominators and rescale it in the end).

The *hermite normal form* is the key to finding solutions to the diophantine equations.

51 Hermite Normal Form

A full rank matrix A is said to be in *hermite normal form* if

- The matrix A is of the form $[B \ 0]$ where B is a $m \times m$ matrix that is invertible.
- B is lower triangular.
- The diagonal entries of B are strictly greater than zero.
- Other entries are non-negative.
- For every row, the unique maximum of that row is attained at the diagonal entry.

An example is the following:

$$\begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 1 & 3 & 0 & 0 & 0 \\ 2 & 1 & 3 & 0 & 0 \end{bmatrix}$$

We will now see that every matrix can be converted into one in HNF with simple operations.

51.1 Converting to HNF

We want to start with a full rank matrix A and convert it to one in HNF using simple operations called *modular column operations*. These are operations of the form

- exchange two columns
- multiply a column by -1
- Replace a column C_i by $C_i + kC_j$ where $j \neq i$ and $k \in \mathbb{Z}$.

Note that each of the above operation just amounts to post multiplying by a matrix of determinant ± 1. And any sequence of of modular column operations would just be multiplying A by a single unitary matrix U.

Theorem 53. *Every full rank rational matrix can be converted into a matrix in HNF using modular column operations*

Proof. The proces will be row-wise. Assume we have got it to the form

$$\begin{bmatrix} B & 0 \\ C & D \end{bmatrix}$$

where $[B \ 0]$ is already in HNF.

First, multiply the columns of D by -1 to make the top row of D with just non-negative entries. Then, rearrange the columns to make sure that the entries are non-decreasing down the row, that is, $\delta_1 \geq \delta_2 \geq \cdots \geq 0$. Note that all of them can't be zero since we have assumed that A is of full rank (thus forces D to also be full rank).

Suppose the gcd of the δ_i was d, then implementing euclid's algorithm using modular column operations, one of the δ_i can be made equal to d. Once this is done, since every other element is a multiple of d, they can be killed. Thus we can make sure that $\delta_1 = d$ and $\delta_i = 0$ for all $i > 1$.

Now to ensure the unique maximum property, let the first row of C be $c_1, c_2, \cdots, c_k$. Use the division algorithm to write $c_i = md + r$ where $r < d$ and replace C_i by $C_i - mD_1$ to change every c_i to its positive reminder modulo d. Thus δ_1 would be the unique maximum in that row and every entry to its left is non-negative.

Proceeding this way, we can convert the matrix to one in HNF. □

The following characterizition is extremely powerful since it takes us from an existential quantifier to a universal quantifier.

Theorem 54. *Let A be a full rank rational matrix and b be a rational vector. The following are equivalent:*

1. *$\exists$ an integral x such that $Ax = b$.*
2. *$\forall$ rational y, yA is integral implies yb is rational.*

Proof. One direction is clear, if $Ax = b$ has an integral solution, then $yAx = yb$. Since x is integral, yaA integral would clearly force yb to be integral.

The converse is slightly tricky. We know that there exists a unimodular matrix that converts A to the HNF. And more over, the conversion preserves the equivalence in the theorem and hence we'll work with that.

$$[B \;\; 0]\, x = b$$

Clearly, $x = \binom{B^{-1}b}{0}$ is a solution to the above equation; the only trouble we could have is that this matrix need not be integral since B^{-1} could have fractional entries.

Now consider the matrix $B^{-1}\,[B \;\; 0]$ row-wise. $B_i^{-1}\,[B \;\; 0]$ is integral would imply $B_i^{-1}b$ is integral. But $B^{-1}\,[B \;\; 0] = [I \;\; 0]$ and hence forces $B^{-1}b$ to be integral. Hence the solution $x = \binom{B^{-1}b}{0}$ is indeed an integral solution. □

52 HNF and Lattices

Let $a_1, a_2, \cdots, a_m$ be a spanning set for R^n. The lattice created by them is the set of all integral combinations of these vectors.

$$L(a_1, a_2, \cdots, a_m) = \left\{ \sum \lambda_i a_i \ : \ \lambda_i \in \mathbb{Z} \right\}$$

When each a_i has only rational entries, this will form a discrete subset in R^n. The HNF of a matrix A completely determines the lattice generated by the columns of A. An immediate corollary is that the HNF is unique.

Theorem 55. *For any two matrices A and A', their columns generate the same lattice if and only if the non-zero part of their HNFs are identical.*

Proof. Of course, if the HNFs are identical we can just invert the unitary tranformation and hence the lattices are equal. Suppose A and A' give the same lattice, let the invertible of the HNF be B and B' respectively. Let i be the first row where B and B' differ and let the column index be j.

Without loss of generality, assume that $0 \geq b_{ij} < b'_{ij} \leq b'_{ii}$. Now look at the vector $b'_j - b_j$, this is cleary in the lattice formed by B' since we are assuming tht both lattices are the same. And more over, the first $i-1$ coordinates of this vector is zero since we have chosen i to be the first row where it differs. Hence if $b'_j - b_j = \sum \lambda_k b'_k$, then $\lambda_1, \lambda_2, \cdots, \lambda_{i-1} = 0$. But since $b'_{ij} - b_{ij} < b'_{ii}$, an integer sum of $\{b_k\}_{k \geq i}$ can never create the coordinate $b'_{ij} - b_{ij}$, giving us the contradiction.

Hence the invertible parts of the HNFs are identical. □

53 Bounding Sizes

In order to talk about efficient algorithms for the linear diophantine equations, we first need to see if HNFs help at all. What if the entries of the HNF are huge? What if the unitary matrix has massive numbers in it? Do we have good bounds for them?

The answer is yes!

53.1 Bounds on HNF size

The following fact makes the bound possible.

Fact 6. $\det B$ *is equal to the gcd of all $m \times m$ subdeterminants of A.*

Proof. Pretty simple, just need to show that the gcd is unaltered in the conversion process. We leave it to the reader to complete the proof. □

A geometric way to look at this is through the lattice. The determinant gives the volume of the principle parallelopiped in the lattice and that is the gcd of m-parallelopipeds by the columns.

With this, we then have $\det A \leq n!(\max |A_{ij}|)^n$ but this is still polynomially many bits. And since our final matrix is upper triangular with integer entries, each of the diagonal entries is bounded by this and hence the entire matrix.

Thus, the size of B is polynomially bounded.

53.2 Bounds on U

Without loss of generality, we can assume that the first m columns of A are linearly independent. Hence let $A = [A' \;\; A'']$ where A' is an invertible $m \times m$ matrix.

Just replace A by the invertible matrix

$$\hat{A} = \begin{pmatrix} A' & A'' \\ 0 & I \end{pmatrix}$$

Suppose this had its HNF as

$$\mathcal{B} = \begin{pmatrix} B & 0 \\ B_1 & B_2 \end{pmatrix}$$

then clearly $U = \hat{A}\mathcal{B}$ and is polynomially bounded.

Hence, the size of U is not too large.

54 Keeping Numbers Small

Though we know that the numbers at the end would be small, we need to make sure that they do not blow up in any intermediate step. The following really clever trick was given by Bachem-Kannan.

Assume that $A = [A' \;\; A'']$ where A' is a non-singular square matrix. Let $|\det A'| = M$. Replace A by the matrix

$$\bar{A} = \left[\begin{array}{c|cccc} & M & & & \\ A & & M & & \\ & & & \ddots & \\ & & & & M \end{array}\right]$$

Claim 56. *The matrices A and $\bar{A}$ generate the same lattice.*

Proof. We know that $A' \cdot adjA' = \det A' \cdot I$ and since $adjA'$ is an integer matrix, $M \cdot I$ is generated by the columns of A'. And hence, both matrices generate the same lattice. □

Since the lattices are the same, computing the HNF of this matrix would be the same as computing the HNF of A. The good thing in $\bar{A}$ is that you can use its columns to make sure that numbers don't blow up; whenever they do, just use the appropriate column to drive it smaller than M.

But notice that these columns help only till the triangulation of the matrix. How do we drive the diagonal to the unique maximum? What if numbers blow up there?

The good news is that it won't. We have made sure that every entry of the matrix is atmost M. The operation of converting every non-diagonal entry to it's remainder modulo M can atmost blow indices by a factor of $(M+1)$. Hence, at the end of it, we would have atmost $M(M+1)^m$ and this is still not large in terms of bit complexity!

Therefore we are in good shape. Since we now know to get the HNF, this solve the linear diophantine equation.

55 Arithmetic circuits with bounds on final answer

Suppose we have an arithmetic circuit, a circuit with multiplication and addition gates, with inputs provided and the circuit evaluating some polynomial.

Suppose we are given the promise that the final answer is upper bounded by some M, we could try the following thing:

- replace every multiplication gate by a multiplication $\pmod M$ gate
- replace every addition gate by an addition $\pmod M$ gate

This modification will not change the output of the circuit at all! The modification ensures that the numbers never get too large in the middle.

This howeever cannot be directly used in the HNF setting since it is not just a circuit we are looking at. There are branches based on comparisons and $\pmod M$ gates need not preserve them.

Nevertheless, this is a great trick.

Algebra and Computation	Course Instructor: V. Arvind
Lecture 18 and 19: LLL and Factorization over $\mathbb{Q}$	
Lecturer: V. Arvind	*Scribe: Ramprasad Saptharishi*

56 Overview

Another problem very essential for factoring univariate polynomials over $\mathbb{Q}$ is the shortest vector problem. Of course, finding the optimum solution is NP-hard and we only want an approximation algorithm to this.

We shall discuss the LLL algorithm for the shortest vector and then give the algorithm for factorizing univariate polynomials over $\mathbb{Q}$.

57 The Shortest Vector Problem

We are given a basis $\{b_i\}_{0\leq i\leq n}$ in R^n and we want to find a vector $v = \sum a_i b_i$, where $a_i \in \mathbb{Z}$, whose norm (the usual euclidian norm) is minimum.

Solving this problem in full generality is NP-hard and we do not expect to find the optimal solution. LLL however allows us to find an approximate solution, the approximation factor depending only on the dimension.

The basic idea is in mimicking the Gram-Schmidt orthogonalization method on a lattice.

57.1 The Gram-Schmidt Orthogonalization

We are given a basis $\{b_1, b_2, \cdots, b_n\}$ and we want to convert it into a new orthogonal basis $\{b_1^\star, b_2^\star, \cdots, b_n^\star\}$.

The GS algorithm is as follows:

$$\begin{aligned} & b_1^\star &=& \ b_1 \\ \forall 1 < i \leq n \qquad & b_i^\star &=& \ b_i - \sum_{j<i} \mu_{ij} b_j^\star \qquad \text{where } \mu_{ij} = \frac{\left\langle b_i, b_j^\star \right\rangle}{\left\| b_j^\star \right\|^2} \end{aligned}$$

The GS basis satisfies the following properties, which are easy to check:

- Different ordering of the basis vectors could give different GS orthogonal bases.

- The basis vectors are mutually orthogonal.
- For each i, $\|b_i^\star\| \leq \|b_i\|$.
- Fore each i, the span of $\{b_1, \cdots, b_i\}$ is the same as the span $\{b_1^\star, \cdots, b_i^\star\}$.
- If B is the matrix whose columns are the vectors $\{b_i\}$ and if $B^\star$ is the matrix with columns $\{b_i^\star\}$, then the tranformation is given by the following unimodular triangular matrix:

$$\begin{bmatrix} 1 & & & \\ \mu_{21} & 1 & & \\ \vdots & & \ddots & \\ \mu_{n1} & \mu_{n2} & \cdots & 1 \end{bmatrix} B^\star = B$$

  and therefore $\det B = \det B^\star$, the volume of the fundamental parallelopiped of the lattice is preserved.
- If $L(B)$ is the lattice generated by B,

$$|\det L(B)| := |\det B| = \|b_1^\star\| \|b_2^\star\| \cdots \|b_n^\star\| \leq \|b_1\| \|b_2\| \cdots \|b_n\|$$

  And if $\mathcal{B}$ is the largest value in the matrix B, then we have the famous hadamard inequality

$$|\det B| \leq n^{n/2} \mathcal{B}^n$$

57.2 Reduced Basis

The following observation is the key to LLL.

Observation 57. *Let $b = \lambda_i b_i$ be the shortest vector in the lattice. Then*

$$\|b\| \geq min \|b_i^\star\|$$

Proof. Let k be the largest index such that $\lambda_k \neq 0$. Since the GS tranformation matrix has 1s on the diagonal, even after we write $b = \sum \lambda_i' b_i^\star$, $\lambda_k = \lambda' k$.

Hence,

$$\|b\| = \sum |\lambda_i'|^2 \|b_i^\star\| \geq |\lambda_k| \|b_k^\star\| \geq \|b_k^\star\| \geq min \|b_i^\star\|$$

□

With this as the motivation, we have the following concept of a reduced basis.

Definition 58. *A basis $\{b_i\}$ is said to be a reduced basis if it satisfies the condition that for all i, $\|b_i\|^2 \leq 2\|b_{i+1}\|^2$.*

From the earlier observation, it is clear that once we have a reduced basis,

$$\|b_1\| \leq 2^{\frac{n-1}{2}} \|\text{opt}\|$$

And hence, if we can find a reduced basis for the lattice, then we have achieved our goal of finding a constant factor (only a function of degree) approximation of the shortest vector problem.

58 LLL Algorithm

The *LLL* algorithm finds a reduced basis for the lattice. The idea is to mimic GS but tranform vectors to those within the lattice. At the same time, we need to keep in mind that vectors don't become too short (to form a reduced basis). The algorithm is very mysterious, we shall first present he algorithm then argue that it halts quickly and also that it works correctly.

Input: A basis $\{b_i\}$.

1: Find the GS basis $\{b_i^\star\}$.
(Reduction Step)
2: **for** $i = 2$ to n **do**
3: **for** $j = i - 1$ to 1 **do**
4: $b_i = b_i - \alpha_{ij} b_j$ where $\alpha_{ij} = \left\lfloor \frac{\langle b_i, b_j^\star \rangle}{\|b_j^\star\|^2} \right\rceil$
5: **end for**
6: **end for**
7: *(Swap Step)* If there exists an i such that

$$\frac{3}{4}\|b_i^\star\|^2 > \left\|b_{i+1}^\star + \mu_{i+1,i} b_i^\star\right\|^2$$

then swap b_i and b_{i+1} and go to step 1.
8: **output** $b_1, b_2, \cdots, b_n$.

58.1 The Reduction Step

The reduction step is basically an approximation to the GS orthogonalization, but staying on the lattice. We shall show that we actually get pretty close to the orthogonal basis.

For the basis $\{b_i\}$, let the GS basis is $\{b_i^\star\}$. If we were to consider the matrix with columns as $\{b_i\}$ as vectors over the GS basis as the standard basis, then B would look like an upper triangular matrix with 1s on the diagonal.

The reduction step makes the other non-diagonal entries small (bounded by 1/2). We shall see how this is achieved.

The two *for* loops are designed cleverly so that you never undo something that you have already done. The key point to note is that in the reduction step, the GS basis is maintained. Look at an intermediate step, say at i, j. By induction, assume that all columns whose index is less than i has already been taken care of.

And since we have gone up to j, the i-th column is fixed from bottom to top. Since the GS basis is fixed, if we had removed the roundoff in α_{ij} when we did $b_i = b_i - \alpha_{ij} b_j$ we would have actually got a vector orthogonal to $b_j^\star$ and hence $b_i j$ would have become 0. But since we are just rounding off, we will atleast reduce that value to 1/2. Note that this works only because the GS basis stays the same throughout.

Now we have fixed the index b_{ij} and we can go on to $b_{i,j-1}$. Thus by induction, we have proved that at the end of the reduction step, we have an uppertriangular matrix with 1s on the diagonal and every non-zero entry is bounded by 1/2.

58.2 The Swap Step

The swap step is like a 'check if reduced basis, else rectify' step. The crucial point is that this step will happen for atmost polynomially many steps. To show this, we will develop a certain value (exponential sized) and show that decreases by a constant factor (3/4) and hence can happen atmost polynomially many times.

For a basis B, define

$$\begin{aligned} D_{B,i} &= \prod_{j=1}^{i} \|b_i^\star\|^2 \\ D_B &= \prod_{i=1}^{n} D_{B,i} \end{aligned}$$

It is easy to see that D_B is a value that is at most exponential. We will show that it goes down by 3/4 each time we swap.

Recall that

$$MB^\star = \begin{bmatrix} 1 & & & \\ \mu_{21} & 1 & & \\ \vdots & & \ddots & \\ \mu_{n1} & \mu_{n2} & \cdots & 1 \end{bmatrix} B^\star = B$$

We could do the same by restricting the above equation to just the first i rows and columns. As a notation, we will write this as

$$B_i = M_i B_i^\star$$

Since M_i is a unimodular matrix,

$$\det(B_i B_i^T) = \det(B_i^\star (B_i^\star)^T) = D_{B_i}$$

Consider the case when you are to do the swap operation between i and $i+1$. Then the basis $B = \{b_1, \cdots, b_{i-1}, b_i, b_{i+1}, \cdots, b_n\}$ will now change to $\hat{B} = \{b_1, \cdots, b_{i-1}, b_{i+1}, b_i, b_{i+2}, \cdots, b_n\}$. The only place where the GS basis will differ will be at the i-th index.

In the original basis B, we would have just $b_i^\star$. But in the other basis $\hat{B}$, it is easy to check that $\hat{b}_i^\star = b_{i+1}^\star + \mu_{i+1,i} b_i^\star$. The other vectors would be the same in both cases.

Thus, clearly,

$$\frac{D_{\hat{B}}}{D_B} = \frac{D_{\hat{B},i}}{D_{B,i}} = \frac{\left\|b_{i+1}^\star + \mu_{i+1,i} b_i^\star\right\|^2}{\|b_i^\star\|^2} \leq \frac{3}{4}$$

And hence the swap step is executed only polynomially many times.

58.3 Correctness

The next thing we need to show is that at the end of the algorithm, we do have a reduced basis. This is an easy observation. Since for all indices

$$\begin{aligned} \frac{3}{4}\left\|b_i^\star\right\|^2 &\leq \left\|b_{i+1}^\star + \mu_{i+1,i} b_i^\star\right\|^2 \\ &= \left\|b_{i+1}^\star\right\|^2 + \mu_{i+1,i}^2 \left\|b_i^\star\right\|^2 \\ &\leq \left\|b_{i+1}^\star\right\|^2 + \frac{1}{4}\left\|b_i^\star\right\|^2 \\ \implies \left\|b_i^\star\right\| &\leq 2\left\|b_{i+1}^\star\right\| \end{aligned}$$

And therefore, we indeed have a reduced basis, and hence solves the approximation of the shortest vector problem.

58.4 Sizes of Numbers

Using the matrices that appeared in the reduction step section, we can show using cramer's rule that the numbers do not become very large.

We leave this as an exercise.

59 Factoring over $\mathbb{Q}$

We will see a sketch of the factoring algorithm, and gaps are left as an assignment. The working is very similar to the bivariate hensel lifting.

The algorithm is as follows:

1. Assume $f(x) \in \mathbb{Z}[x]$ is square free.
2. Pick a small ($O(\log n)$ bits long) prime such that $f(x)$ is square free modulo p.
3. Factor $f = gh \pmod p$. where g is irreducible and monic.
4. Hensel lift the factorization k times to obtain $f = g_k h_k \pmod{p^k}$.
5. Solve the linear equation $\tilde{g} = g_k l_k \pmod{p^k}$ for polynomials $\tilde{g}$ and l_k such that their degree is less than $\deg f$.
6. Output $\gcd(f, \tilde{g})$, if trivial output irreducible.

First catch is the following, does a polynomial necessarily have only small factors? Can there be factors with huge numbers in them? The following bound tells us that we are safe in this area.

Lemma 59 (Mignotte's Bound)**.** *If* $f(x) = a_0 + a_1 x + \cdots + a_n x^n$*, then any root* α *of* f *is such that* $|\alpha| < n \max |a_i|$.

Since all coefficients are symmetric polynomials over the roots, we are in good shape.

For the proof of correctness, we need a suitable bound on k to push the proof of the bivariate case through the same this. But the issue is that, we do not have any bounds on the coefficients of $l_k, \tilde{g}$ to make it work. How do we make sure that the solution to the system of equations is small? Enter LLL.

Look at $\tilde{g} = g_k l_k + p^k r_k$ for any polynomial r. We can easily induce a lattice structure on this by choosing a natural basis. Over this lattice, we can now ask for a short vector. Note that LLL will not give us the shortest vector but a $2^{\frac{n-1}{2}}$ is good enough!

Using that, a bound on k can be fixed and the same proof of bivariate factorization will go through. The gaps are left to the readers to fill in.

Algebra and Computation Course Instructor: V. Arvind

Lecture 22: Simon's Problem and towards Shor

Lecturer: V. Arvind *Scribe: Ramprasad Saptharishi*

60 Overview

Last lecture we saw a toy problem that showed that the quantum model could be more efficient than the turing machine model, query complexity in the last example.

In this lecture, we shall inspect one more problem, which in a way is a true separation from the classical and quantum model since it beats even randomized algorithms on classical turing machines. The techniques used here will be essential in Shor's algorithms for integer factoring and discrete logarithm, whci hwe shall see in the following lectures.

61 Simon's Problem

We are given a function $f : \{0,1\}^n \to X$ with the promise that f is a 2-to-1 function (exactly two strings in $\{0,1\}^n$ mapping to an element of X). Further, f satisfies the additional property that there exists a $\lambda \neq 0^n$ such that $f(x \oplus \lambda) = f(x)$ for all x. The goal is to find the period λ with as few probes as possible.

We could make X canonical by enforcing an order on X. We can think of strings in $\{0,1\}^n$ as ordered pairs $\{(x, x \oplus \lambda)\}$ where x is lexicographically smaller than $x \oplus \lambda$. These pairs can now be identified with $\{0,1\}^{n-1}$ by just sorting the pairs, and hence fixes the set on the right. Thus, the number of such functions is exactly equal to the number of possible λs which is $2^n - 1$.

61.1 Lower Bounds on Classical Deterministic Computation

An adversial argument showsn an exponential lower bound for the query complexity in the determistic model. As an adversary, the point is to keep giving different answers for queries as long as it is possible. If k strings have been queried, the number of λs that we eliminate by giving different answers to each is $\binom{k}{2}$. Thus the adversary can go on giving different answers till $2^{n/2}$ queries have made.

Thus, the deterministic query complexity is atleast $2^{n/2}$.

61.2 Lower Bounds on Classical Randomized Computation

Suppose there was a procedure such that with just $q(n)$ queries the procedure will find λ with error probability atmost $1/3$. We can amplify this by repeating this experiment suitably many times and ge the error probability down to less than 2^{-n}.

Now we can use the same idea as we did in showing $\mathsf{BPP} \in \mathsf{P/poly}$. There are only $2^n - 1$ possible λs and the error probability is less than 2^{-n}. The randomized algorithms makes some random choices in the case of each λ. Suppose there was atleast 1 error in every possible λ, then the overall error probability would be more than 2^{-n}. Hence there has to be a sequence of choices such that it works for all possible values of λ.

But we have just shown earlier that the determistic computation has an exponential lowerbound, which gives a contradiction. Hence there is no probabilistic algorithm in the classical model that can find λ error bounded by a constant.

61.3 A Quantum Algorithm

Again, the hadamard code plays the major role here. Start with the uniform superposition

$$\begin{aligned} H_n |x\rangle &= \frac{1}{2^{n/2}} \sum_{u \in \{0,1\}^n} (-1)^{u \cdot x} |u\rangle \\ \implies H_n |0^m\rangle &= \frac{1}{2^{n/2}} \sum_{u \in \{0,1\}^n} |u\rangle \end{aligned}$$

By giving it input x and 0^m, it would return x and $0^m \oplus f(x) = f(x)$ on the other side.

Hence, if we make x the uniform superposition, on applying f we get:

$$|\psi\rangle = \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} |x\rangle \otimes |0^m\rangle \longrightarrow \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} |x\rangle \otimes |f(x)\rangle$$

Now on just measuring the qubit $|f(x)\rangle$, it would collapse to a uniformly random element in the image say $f(x_0)$. And on a measurement, since every other coordinate dies, the state would now collapse to:

$$|\psi\rangle = \frac{1}{\sqrt{2}} \left(|x_0\rangle + |x_0 + \lambda\rangle \right)$$

On applying H_n to this, we get

$$\begin{aligned}
|\psi\rangle &\rightarrow c\left(\sum_{u\in\{0,1\}^n}(-1)^{x_0\cdot u}|u\rangle+\sum_{u\in\{0,1\}^n}(-1)^{(x_0\oplus\lambda)\cdot u}|u\rangle\right)\\
&= c\sum_{u\in\{0,1\}^n}\left((-1)^{x_0\cdot u}+(-1)^{x_0\cdot u\oplus\lambda\cdot u}\right)|u\rangle\\
&= \sum_{u\cdot\lambda=0}\alpha_u|u\rangle \quad , \quad \alpha_u=\frac{1}{2^{(n-1)/2}}
\end{aligned}$$

But this is just a uniform superposition on the orthogonal space of the span of α, and thus measuring u would now give a random sampling in the orthogonal space. We will now show that once we have a random sampling of the space, we can get a basis.

Lemma 60. *Let G be a finite group. The probability that a uniform sample of size $4\log|G|$ generates G is at least $\frac{1}{3}$.*

Proof. Let $g_1, g_2, \cdots, g_m$ be the sample. Define G_i to be the group generated by $\{g_j\}_{j\le i}$. Let X be the indicator random variable that something bad happens:

$$X_i=\begin{cases}0 & \text{if } G_{i-1}=G \text{ or } g_i\notin G_{i-1}\\ 1 & \text{otherwise}\end{cases}$$

and let $X=\sum_{i=1}^m X_i$ then

$$\begin{aligned}
\Pr[X_i=0] &= \Pr[G_{i-1}=G]+\Pr[G_i\neq G]\Pr[g_i\notin G_{i-1}|G_{i-1}\neq G]\\
&= p_i+(1-p_i)\frac{1}{2}\\
&= \frac{1}{2}+\frac{p_i}{2}\geq\frac{1}{2}\\
\implies E[X_i] &= \Pr[X_i=1]\leq\frac{1}{2}\\
\implies E[X] &= \sum_{i=1}^m E[X_i]\leq\frac{m}{2}
\end{aligned}$$

By Markov's inequality,

$$\Pr[X\geq a]\leq\frac{E[x]}{a}\leq\frac{m}{2a}$$

Choosing $a=\frac{3m}{4}$ would get give $\Pr[X\geq\frac{3m}{4}]\leq\frac{2}{3}$.

But notice that if we haven't got a generating set already, $X_i = 0$ can happen atmost $\log |G|$ many times. Thus if we were to choose $m = 4 \log |G|$, at most $\log |G|$ of the X_i can be zero and hence will force $G_m = G$ with probability atleast $\frac{1}{3}$. □

Now that we have a generating set for the orthogonal set, all we need to do right now is solve a system of linear equations to get the orthogonal space of this, which is the space of λ. Thus we would find λ with just polynomially many queries since $\log |G| \leq n$.

The same idea can be used to solve a general problem as well. We are given a function f such that there is a subspace such that $f(x \oplus H) = f(x)$. The goal is to find a subspace. The same ideas can be used to do this as well. (exercise to the reader)

62 Towards Shor's Algorithms

The techniques used in the quantum solution to Simon's problem is essential for Shor's algorithms for integer factoring and the discrete logarithm problem. Shor's algorithm is a quantum algorithm for order finding (given a numbers a and n, find $ord_n(a)$). But the following lemma would show that this is good enough.

Lemma 61. *Order finding is as hard as integer factoring.*

Proof. We will show a probabilistic reduction from factoring to order finding. Without loss of generality, we can assume that n is odd. Pick an x at random from $\mathbb{Z}_n \setminus \{0\}$. We would be extremely lucky if $\gcd(x, n) \neq 1$, we then immediately have a non-trivial factor of n. Hence, we can assume that x is randomly picked from $\mathbb{Z}_n^\star$.

Suppose $ord_n(x)$ is even, and it further satisfies the property that $x^{ord_n(x)/2} \neq -1 \pmod{n}$, then we have non-trivial square root of unity and hence $\gcd(n, x^{ord_n(x)/2} - 1)$ or $\gcd(n, x^{ord_n(x)/2} + 1)$ will be non-trivial. This is the reduction. Once we show that we find a good x with high probability, we are done.

Assume that $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$. By the chinese remaindering theorem,

$$\mathbb{Z}_n^\star = \mathbb{Z}_{p_1^{\alpha_1}}^\star \times \mathbb{Z}_{p_2^{\alpha_2}}^\star \times \cdots \mathbb{Z}_{p_k^{\alpha_k}}^\star$$

This map would take $x \mapsto (x_1, x_2, \cdots, x_k)$. Let $r = s2^t = ord_n(x)$ and $r_i = s_i 2^{t_i} = ord_{p_i^{\alpha_i}}(x_i)$. Clearly, $s = \text{lcm } s_i$ and $t = \max t_i$. Thus if r is odd, then $t = 0$ which means $t_1 = t_2 = \cdots = t_k = 0$.

Suppose r was even, we shall analyze the probability that $x^{r/2} = -1 \pmod{n}$. By the chinese remaindering theorem, $x^{r/2} = -1 \mapsto (-1, -1, \cdots, -1)$. But suppose some $t_i \neq t$, then $t_i \leq t - 1$ and hence $r_i \mid r/2$ and therefore the i-th coordinate in the tuple will have to be a 1 instead of a -1.

Therefore, the probability that r is odd and $x^{r/2} = -1 \pmod{n}$ is bounded above by $\Pr[t_1 = t_2 = \cdots = t_k] \leq 2^{k-1}$ since each t_i is independent.

Therefore $\Pr[r \text{ is even and } x^{r/2} \neq -1 \pmod{n}] \geq 1 - 2^{k-1}$. □

Over the next few lectures, we will see how we can find the order of an element using a quantum algorithm, and this would solve the integer factoring problem.

Algebra and Computation Course Instructor: V. Arvind

Lecture 20: Introduction to quantum computation

Lecturer: V. Arvind *Scribe: Vipul Naik*

63 Physical systems and computational models

63.1 Computers as physical systems

A computer program takes certain input data, manipulates it using certain rules, and produces some output. If we assume that this manipulation is subject to "physical laws" we can consider the loose analogy:

- The computer is a physical system, or lab apparatus
- Running the program is like conducting an experiment
- The output is like the observations made from the experiment

For any computational model to be of practical interest to us, it should be implementable as a physical system. The interesting question is the reverse one: given any physical system, can we turn it into a useful computational model?

63.2 Feynman's question

Feynman wanted to know if quantum mechanics could be used to provide a useful computational model. There are the following questions:

- How can we describe an abstract computational model whose corresponding physical system is subject to the laws of quantum mechanics?
- How does the computational power of such a model compare with that of physical systems subject to the laws of classical mechanics?

63.3 The situation before quantum mechanics

Turing and Church had considered various computational models, such as Turing machines, random-access machines, and so on. All these computational models could be implemented through physical systems subject to the laws of classical mechanics. While studying many such computational models, computer scientists came up with the following Holy Grails:

1. **Church-Turing thesis**: This states that any computational model is as powerful as the Turing machine. In other words, given any computational model, we can simulate computations on that model using the Turing machine. The simulation may of course involve a blow-up in time taken as well as in space used.

2. **Strong Church-Turing thesis**: This states that for any computational model, a polynomial-time algorithm for a decision problem in that computational model can be simulated by a polynomial-time algorithm in the Turing machine model. In looser language, if we think of polynomial time as the notion of *tractability*, then tractability in any computational model is equivalent to tractability in the Turing machine model.

3. **Strong Church-Turing thesis** (randomized version): This states that for any computational model, a bounded-error probabilistic polynomial time algorithm for a decision problem in that computational model can be simulated by a bounded-error probabilistic polynomial time algorithm for the problem in the Turing machine model. In looser language, if we think of BPP as the notion of *tractability*, then BPP is any computational model is equivalent to tractability in the Turing machine model.

While (1) remains unchallenged, quantum computation challenges (2) and (3) – if we can think of the quantum computation model as sufficiently *reasonable*.

64 The two-slit experiment

64.1 The two-slit experiment with particles

Suppose a gun is placed behind a wall with two slits – with the gun firing bullets uniformly in all directions. There is a screen behind the wall that "picks up" those bullets which pass through the slits.

Now, we have the following intuitively clear fact: Suppose p denotes the total number of particles hitting per unit time at a point on the screen when both slits are open, p_1 denotes the number when one slit is open, and p_2 denotes the number when the other slit is open. Then $p = p_1 + p_2$.

In other words, every particular passes either through one slit or the other.

64.2 The two-slit experiment with waves

In the waves version of the two-slit experiment, the "source" is a light source rather than a gun, and light is radiated uniformly in all directions. Now, if A denotes the amplitude of light received at a point on the screen behind when both slits are open, and A_1 denotes the amplitude when only the first slit is open, and A_2 denotes the amplitude when only the second slit is open, then:

$$A^2 = A_1^2 + A_2^2$$

In other words, it is *not* true that $A = A_1 + A_2$ – there is a *cancellation* due to interference. What gets added is the total *energy* and not the amplitudes.

64.3 The dual nature of matter and waves

The surprising thing about quantum theory is that the same thing could behave both as particle and as wave – it behaves as a particle when the amplitudes simply add, and it behaves like a wave when the squares of the amplitudes add.

65 The setup of quantum theory

65.1 Basic axioms

In quantum theory, we have the following regarding the state of a physical system:

1. The possible outcomes form a basis of a $\mathbb{C}$-vector space, called the state space.
2. The current state of the system is an element in the state space, viz a $\mathbb{C}$-linear combination of the outcomes. If ψ_i denotes the component of this state along outcome i, then we have $\sum_i |\alpha_i|^2 = 1$

   The current state of the system is termed a quantum superposition of the states i for which $\alpha_i \neq 0$.
3. Given the current state of the system, if we try to "measure" the outcome, we will get outcome i with probability $|\alpha_i|^2$.

We can express the state vector as a column vector with the i^{th} entry being α_i.

65.2 Hermitian inner product

Let V be a $\mathbb{C}$-vector space. An inner product is a map $\langle | \rangle : V \times V \to \mathbb{C}$ satisfying the following conditions:

1. It is conjugate-linear in the first variable, viz:

$$\langle a + b | c \rangle = \langle a | c \rangle + \langle b | c \rangle$$
$$\langle \alpha a | b \rangle = \overline{\alpha} \langle a | b \rangle$$

2. It is linear in the second variable, viz:

$$\langle a | b + c \rangle = \langle a | b \rangle + \langle a | c \rangle$$
$$\langle a | \alpha b \rangle = \alpha \langle a | b \rangle$$

3. It is Hermitian-symmetric, viz:

$$\langle a | b \rangle = \overline{\langle b | a \rangle}$$

4. It is positive definite, viz:

$$\langle a | a \rangle > 0$$

   whenever $a \neq 0$

We will follow Dirac's notation. The basis vector corresponding to outcome i will be denoted as $|i\rangle$. This is also called the *ket* vector.

Given any state A we denote by $\langle A | \psi \rangle$ the Hermitian inner product of A and ψ, and we also call this the probability amplitude of ψ in A.

65.3 The way quantum states evolve

A unitary operator is an invertible linear operator from the state space to itself under which the Hermitian inner product evolves. When we apply a unitary operator, we essentially switch from the original orthonormal basis to a new orthonormal basis. This means that when we now make a measurement in the new basis, we will get one of the new basis vectors, with probability equalling the square of the modulus of its amplitude.

The power of quantum theory lies in the following fact: in discrete time, the evolution of the quantum state of a system is given by a unitary operator. That is, there is a unitary operator U on the state space that maps the initial state to the final state.

In matrix terms, we can view U as a unitary matrix which takes a state written as a column vector in the original basis, and outputs the column vector for it in the new basis.

65.4 Different quantum states giving the same probability

Note that if $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ and $(\beta_1, \beta_2, \ldots, \beta_n)$ are two different states such that β_i/α_i has norm 1 for every i, then they give rise to the same probability distribution.

In the particular case where β_i/α_i is the same for all i, we say that the two quantum states differ by a *phase* of ϕ (where the common ratio is $e^{i\phi}$).

Here are two points:

- The quantum states that we are interested in are those on the unit sphere (that is, those of norm 1) upto phase. That is, we identify two quantum states if they differ by a multiplicative factor of a phase.

  In mathematical lingo, this is the projective complex space of $n-1$ dimensions.

  Note that if two quantum states differ only by phase, then applying the unitary operator to both of them again gives quantum states that differ by the same phase.

- It may be possible for two inequivalent quantum states to give the same probability distribution – this happens when the ratios for each coordinate are complex numbers.

  However, it is *not* true that if two quantum states give the same probability distribution, then applying any unitary operator to both of them also yields quantum states giving the same probability distribution. In other words, the quantum state carries *more* information than simply the associated probability distribution.

66 Quantum superposition versus random sampling

66.1 Probability distribution versus quantum superposition

A probability distribution over a set $\{1, 2, \cdots, n\}$ is an association of a nonnegative real number p_i to each i such that $\sum_i p_i = 1$. Suppose we sample randomly from this probability distribution, and associate a reward a_i to picking i. Then the expected reward is:

$$\sum_i a_i p_i$$

A quantum superposition over a set $\{1, 2, \cdots, n\}$ of states, on the other hand, is an association of a complex number ψ_i to each i such that the corresponding probability distribution associates, to each i, the value $|\psi_i|^2$. In other words, given a quantum superposition, the probability of measuring the value i from that superposition is $|\psi_i|^2$.

This immediately raises some questions:

- If two quantum superpositions give rise to the same probability distribution, how are they physically distinguishable?
- What are the ways in which we can transform one quantum superposition into another?

66.2 Transforming probability distributions

Suppose we are given a probability distribution. Then, to transform the probability distribution, we could do the following: consider a transition, which, if starting at state j, goes to state i with probability q_{ij}. Then if the current probability distribution vector is $p = (p_1, p_2, \ldots, p_n)^t$, and Q is the matrix of q_{ij}s, the new probability distribution vector is Qp.

The matrix Q here has the property that every column sum is exactly one; such a matrix is termed a stochastic matrix.

Note that since we are multiplying with a stochastic matrix, and all the entries of a stochastic matrix are nonnegative, it is not possible to make probabilities *cancel*, or *kill*, each other.

66.3 Transforming quantum states

The fundamental difference between the classical probabilistic model and the quantum model is that in the quantum model, we perform the operator, not

on the probability distribution, but on the underlying quantum state. That is, we pick on a unitary operator, and transform the quantum superposition according to the unitary operator. Here are some important points to note:

- The entries of a unitary operator can be both positive and negative (in fact, they can even be complex). Hence, it is possible to use a unitary operator to make terms *cancel* each other
- Note that we are making the unitary operator act on the underlying quantum state. Hence, two quantum superpositions that start off by giving the *same* probability distribution could end up giving separate probability distributions once we apply the unitary operator.

67 Quantum theory and Boolean circuits

67.1 Boolean circuits

Instead of looking at the Turing machine model (a model of variable-length computation) let us look at the Boolean circuit model (a model of fixed-length computation). The reason for choosing the Boolean circuit model to compare with quantum theory is that in quantum theory, we need to work in a state space of fixed dimension.

There are two aspects to the Boolean circuit:

- The values taken by a finite set of variables at any given time. These correspond to the classical "state" of the system.
- The gates themselves, which perform Boolean functions on some values and output the results.

At any stage in the evaluation of a Boolean function using a Boolean circuit, we have some Boolean variables and some values associated with those Boolean variables. To convert this to the quantum setting, we need to consider a state space where each possible assignment of values to the Boolean variables constitutes an outcome. In other words, if there are n Boolean variables, there are 2^n possible outcomes, and the state space is the space $\mathbb{C}^{2^n}$. The set of feasible states is the unit sphere in this space, and if we go upto phase, then the set of feasible states is the projective space.

Having converted the current state of the system to a quantum outcome, the next step is to view the gates in terms of unitary operators which can thus be simulated in a quantum system. There are the following immediate problems:

- The Boolean gates we have seen have more inputs than outputs, so they don't even preserve the number of states
- The Boolean functions for AND and OR are far from invertible, whereas any unitary operator must be invertible.

We shall see how to overcome both these problems at once, by associating to any Boolean function f a unitary operator U_f with approximately the same number of variables, such that computing f is classically the same as computing U_f.

67.2 Unitary operator for a Boolean function

Suppose $f : \{0,1\}^n \to \{0,1\}^m$ is a Boolean function. Then consider first the following Boolean function: it takes as input $(m+n)$ Boolean variables $(x_1, x_2, \ldots, x_n, z_1, z_2, \ldots, z_m)$ and outputs $(m+n)$ Boolean variables, namely $(x_1, x_2, \ldots, x_n, u_1, u_2, \ldots, u_m)$ where $u = z \oplus f(x)$.

First of all note that this Boolean function is involutive – it equals its own inverse. Thus, in particular, it is also invertible.

Now, this Boolean function is a permutation (in fact, an involutive permutation) on the set of all possible elements in $\{0,1\}^{m+n}$. Thus, it can be viewed as a permutation matrix sitting inside $\mathbb{C}^{\{0,1\}^{m+n}}$. Since permutation matrices are unitary matrices, we obtain a unitary operator U_f that computes this function.

67.3 Boolean circuit in the quantum language

In the classical picture of building a Boolean circuit, we start off with a Boolean function $\{0,1\}^n \to \{0,1\}^m$ and try to express it as a composite of the AND, OR and NOT functions in various ways.

Note that each of these AND, OR and NOT functions take in only a small number of variables and output only a small number of variables – they don't touch the other variables at all. Thus, even if they transform the state of the entire system, their actual effect is only in some small part of the system.

The parallel in the case of unitary operators would be: Write the unitary operator U_f as a short-length product of unitary operators each of which "affects" only a small number of variables. To make these notions rigourous, we need to introduce the notion of tensor products of operators.

67.4 Tensor product of vector spaces

Given two vector spaces V and W with bases e_i and f_j, the tensor product $V \otimes W$ is defined as the vector space with basis b_{ij}, with a map

$V \times W \to V \otimes W$

that sends $\left(\sum_i v_i e_i, \sum_j w_j e_j\right)$ to $\sum_{i,j} v_i w_j b_{ij}$.

The tensor product acquires a natural significance for state spaces. Namely, if $V = \mathbb{C}^m$ is the state space spanned by outcomes e_i of experiment I and $W = \mathbb{C}^n$ is the state space spanned by outcomes f_j of experiment J, then the tensor product $V \otimes W$ is the state space for possible outcomes of the combined experiment I, J.

In other words, each outcome for $V \otimes W$ gives both the outcome for experiment I and the outcome for experiment J.

Further, the amplitude of the outcome (i, j) for the combined experiment is the product of the amplitude of outcome i for experiment I and outcome j for experiment J.

Now, given a tensor product $V \otimes W$, it makes sense to talk of the tensor product of operators A and B where A is a linear operator on V and B is a linear operator on W. The idea is roughly to map each b_{ij} to the vector $Ae_i \otimes Bf_j$.

67.5 A quantum circuit

We can now see that if the "memory" stores n variables at any given time, then the state space is the n-fold tensor product of $\mathbb{C}^2$ where $\mathbb{C}^2$ is the state space for one quantum bit (or qubit).

Further, suppose the current state has n variables and there is a quantum gate that inputs r variables and outputs r of them (by applying a unitary operator). Then, if U_g denotes the unitary operator for that quantum gate (when acting only on those r variables), the overall unitary operator is:

$$U_g \otimes I$$

where U_g is viewed as acting on the space $\mathbb{C}^{2^r}$ of those r variables, and I is acting on the space of $\mathbb{C}^{2^{n-r}}$ for the remaining $n - r$ variables.

This helps tell us what the notion of a good quantum circuit should be:

A quantum circuit for U_f is an expression of U_f as a product of unitary operators, each of which can be expressed as the tensor product of a unitary operator acting on a small number of variables, with the identity map.

When the quantum circuit arises from a Boolean circuit, each of those small unitary operators will be the unitary operators corresponding to that Boolean circuit.

67.6 Particular cases: controlled NOT and controlled AND

The controlled NOT gate is obtained as a special case of the general construction.

67.7 Solovay's theorem

Algebra and Computation Course Instructor: V. Arvind

Lecture 21: Quantum simulation of Boolean circuits and more

Lecturer: V. Arvind *Scribe: Vipul Naik*

68 Postulates of quantum mechanics

68.1 The state space postulate

The state space of an isolated physical system is a $\mathbb{C}$-vector space (equipped with inner product) and the possible outcomes form an orthonormal basis for this space.

Note that we require the physical system to be *isolated*. In fact, one of the concrete problems with implementing quantum computers in reality is the inability to sufficiently isolate the quantum computer from the outside world.

68.2 The evolution postulate

The state space of an isolated physical system evolves under the action of a unitary operator.

In other words, if $|\psi\rangle$ is the state at time t_1 and $|\psi'\rangle$ is the state at time t_2, then there exists a unitary operator U_{t_1,t_2} that maps $|\psi\rangle$ to $|\psi'\rangle$.

The unitary operator can thus be viewed as acting in discrete time, according to a "clock" whose clock pulse is $t_2 - t_1$.

Continuous-time evolution, if we are interested in that, is governed by a Hermitian operator, called the Hamiltonian of the system. That is:

$$ih\frac{d}{dt}|\psi\rangle = H|\psi\rangle$$

This is obtained by differentiating the unitary operator with respect to time.

When we only assume the physical system to be closed and do *not* assume it to be isolated, then we get a time-varying Hamiltonian, and hence the evolution is not given by a unitary operator.

68.3 The measurement postulate

Definition 62. *A* measurement is a collection of linear operators M_m such that:

$$\sum_m M_m^* M_m = I$$

The measurement is said to be *measurement(projective) if it measures components with respect to an orthogonal direct sum decomposition.*

The measurements we have talked of so far are projective measurements where we look at a *complete* orthogonal direct sum decomposition, that is, a decomposition as a sum of pairwise orthogonal one-dimensional subspaces.

69 The Deutsch-Josza problem

69.1 Statement of the problem

The Deutsch-Josza problem is a somewhat artificial problem that illustrates that in the query model, deterministic quantum computation can be far faster than deterministic classical computation. By query model, we mean a model where the complexity is measured by the number of queries that need to be made to an oracle, to answer a question about something hidden within that oracle.

Here is the precise statement:

Problem:

$f : \{0,1\}^n \to \{0,1\}$ is a Boolean function with the "promise" that f is either constant (that is, $f(x) = f(y)$ for all $x, y \in \{0,1\}^n$) or balanced (that is, $\left|f^{-1}(0)\right| = \left|f^{-1}(1)\right|$). We have a query oracle for f, that can take in $x \in \{0,1\}^n$ and output $f(x)$. We need to use this query oracle to find where f is constant or balanced. The complexity of our procedure is determined by the number of queries (calls) made to the oracle.

69.2 Classical deterministic and randomized complexities

The deterministic complexity of the Deutsch-Josza problem is $2^{n-1}+1$. This is because if the function is actually constant, then we need to know its value at at least that many points to be sure that it is constant.

The randomized complexity of the Deutsch-Josza problem is constant, in the sense that we can, given any ϵ, make a constant number of queries

dependent only on ϵ such that the probability of error is bounded above by ϵ (note that in this case the error is one-sided).

69.3 Rules for the quantum algorithm

In the quantum algorithm, what we want to do is to use the fact that there are an equal number of 0s and 1s, to get the 0s and 1s to *cancel* one another. First, however, we need to be clear as to what exactly is *given* in the quantum algorithm. The quantum algorithm does not oracle-query f, rather it oracle-queries U_f, the unitary operator associated to f.

Further, the "input" that we send to U_f need not be a "pure" outcome, it could be a state with any mix of amplitudes of the various outcomes.

69.4 The Hadamard gate

Consider a qubit (state space is $\mathbb{C}^2$). The Hadamard gate takes this as input and outputs another qubit, and its action on the basis $|0\rangle, |1\rangle$ is defined as follows:

$$|0\rangle \mapsto \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$
$$|1\rangle \mapsto \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

The Hadamard gate (called H) can be thought of as a particular instance of what we will later see as a *rotation gate* – it rotates the basis by an angle of $\pi/4$.

The Hadamard gate acts on each qubit. Hence, if the state space has n qubits, we can consider the n^{th} tensor power of the Hadamard gate. This is a unitary operator that does the Hadamard on *each* gate. This tensor power is often denoted as $H^{\otimes n}$.

Let's see what happens if we apply $H^{\otimes n}$ to $|0\rangle^{\otimes n}$. We'll get:

$$\begin{aligned} H^{\otimes n}(|0\rangle^{\otimes n}) &= \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)^{\otimes n} \\ \implies H^{\otimes n}(|0\rangle^{\otimes n}) &= \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} |x\rangle \end{aligned}$$

Note that applying the inverse of the Hadamard gate again retrieves for us the origina $|0\rangle^{\otimes n}$.

69.5 The solution

The idea is to use the Hadamard gate to *cancel* the effect of the 0s and the 1s.

1. Start with a state where all qubits are $|0\rangle$
2. Apply the Hadamard transform $H^{\otimes n}$ to get a state where all qubits are $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$. By the calculation done above, the new state is $1/2^{n/2}$ times the sum of all possible outcomes (classical states) of the state space.
3. Tensor with the state $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$.
4. Now apply the operator U_f. Note that U_f applied to a pure outcome x is:

$$x \otimes \frac{|f(x)\rangle - |1 + f(x)\rangle}{2^{(n+1)/2}}$$

which simplifies to:

$$x \otimes (-1)^{f(x)} \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

Hence the effect of U_f on the current state is:

$$\sum_{x \in \{0,1\}^n} x \otimes (-1)^{f(x)} \frac{|0\rangle - |1\rangle}{2^{(n+1)/2}}$$

69.6 For a constant function

In the case that f is constant, the second term in the tensor product becomes constant, and pulling the $(-1)^{f(x)}$ out (which after all only controls the phase), we'll get:

$$(\sum_x \in \{0,1\}^n x) \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

Now, applying the inverse Hadamard transform to the first n qubits, we retrieve $|0\rangle^{\otimes n} \otimes \frac{|0\rangle-|1\rangle}{2^{(n+1)/2}}$. Thus, performing a measurement on the first n coordinates yields $|0\rangle^{\otimes n}$ with certainty.

69.7 For a balanced function

In the case that f is balanced, we get exactly half the x's added with a positive sign, and half the x's added with a negative sign. Now, when we apply the inverse Hadamard transform to this state, we will get a quantum state that will have nonzero coefficients for all the places where the function takes the value 1.

69.8 The upshot

The upshot is as follows:

- We use the Hadamard transform to obtain a uniform superposition of all the possible input states, and then apply the unitary operator to this, tensored with $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$
- We then again apply the inverse Hadamard transform to the output qubit and obtain the "aggregate" value of the function, hence any measurement gives us the answer.

Algebra and Computation Course Instructor: V. Arvind

Lecture 22: Simon's Problem and towards Shor

Lecturer: V. Arvind *Scribe: Ramprasad Saptharishi*

70 Overview

Last lecture we saw a toy problem that showed that the quantum model could be more efficient than the turing machine model, query complexity in the last example.

In this lecture, we shall inspect one more problem, which in a way is a true separation from the classical and quantum model since it beats even randomized algorithms on classical turing machines. The techniques used here will be essential in Shor's algorithms for integer factoring and discrete logarithm, whci hwe shall see in the following lectures.

71 Simon's Problem

We are given a function $f : \{0,1\}^n \to X$ with the promise that f is a 2-to-1 function (exactly two strings in $\{0,1\}^n$ mapping to an element of X). Further, f satisfies the additional property that there exists a $\lambda \neq 0^n$ such that $f(x \oplus \lambda) = f(x)$ for all x. The goal is to find the period λ with as few probes as possible.

We could make X canonical by enforcing an order on X. We can think of strings in $\{0,1\}^n$ as ordered pairs $\{(x, x \oplus \lambda)\}$ where x is lexicographically smaller than $x \oplus \lambda$. These pairs can now be identified with $\{0,1\}^{n-1}$ by just sorting the pairs, and hence fixes the set on the right. Thus, the number of such functions is exactly equal to the number of possible λs which is $2^n - 1$.

71.1 Lower Bounds on Classical Deterministic Computation

An adversial argument showsn an exponential lower bound for the query complexity in the determistic model. As an adversary, the point is to keep giving different answers for queries as long as it is possible. If k strings have been queried, the number of λs that we eliminate by giving different answers to each is $\binom{k}{2}$. Thus the adversary can go on giving different answers till $2^{n/2}$ queries have made.

Thus, the deterministic query complexity is atleast $2^{n/2}$.

71.2 Lower Bounds on Classical Randomized Computation

Suppose there was a procedure such that with just $q(n)$ queries the procedure will find λ with error probability atmost $1/3$. We can amplify this by repeating this experiment suitably many times and ge the error probability down to less than 2^{-n}.

Now we can use the same idea as we did in showing $\mathsf{BPP} \in \mathsf{P/poly}$. There are only $2^n - 1$ possible λs and the error probability is less than 2^{-n}. The randomized algorithms makes some random choices in the case of each λ. Suppose there was atleast 1 error in every possible λ, then the overall error probability would be more than 2^{-n}. Hence there has to be a sequence of choices such that it works for all possible values of λ.

But we have just shown earlier that the determistic computation has an exponential lowerbound, which gives a contradiction. Hence there is no probabilistic algorithm in the classical model that can find λ error bounded by a constant.

71.3 A Quantum Algorithm

Again, the hadamard code plays the major role here. Start with the uniform superposition

$$\begin{aligned} H_n |x\rangle &= \frac{1}{2^{n/2}} \sum_{u \in \{0,1\}^n} (-1)^{u \cdot x} |u\rangle \\ \implies H_n |0^m\rangle &= \frac{1}{2^{n/2}} \sum_{u \in \{0,1\}^n} |u\rangle \end{aligned}$$

By giving it input x and 0^m, it would return x and $0^m \oplus f(x) = f(x)$ on the other side.

Hence, if we make x the uniform superposition, on applying f we get:

$$|\psi\rangle = \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} |x\rangle \otimes |0^m\rangle \longrightarrow \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} |x\rangle \otimes |f(x)\rangle$$

Now on just measuring the qubit $|f(x)\rangle$, it would collapse to a uniformly random element in the image say $f(x_0)$. And on a measurement, since every other coordinate dies, the state would now collapse to:

$$|\psi\rangle = \frac{1}{\sqrt{2}} \left(|x_0\rangle + |x_0 + \lambda\rangle \right)$$

On applying H_n to this, we get

$$\begin{aligned}
|\psi\rangle &\rightarrow c\left(\sum_{u\in\{0,1\}^n}(-1)^{x_0\cdot u}|u\rangle+\sum_{u\in\{0,1\}^n}(-1)^{(x_0\oplus\lambda)\cdot u}|u\rangle\right)\\
&= c\sum_{u\in\{0,1\}^n}\left((-1)^{x_0\cdot u}+(-1)^{x_0\cdot u\oplus\lambda\cdot u}\right)|u\rangle\\
&= \sum_{u\cdot\lambda=0}\alpha_u|u\rangle \quad , \quad \alpha_u=\frac{1}{2^{(n-1)/2}}
\end{aligned}$$

But this is just a uniform superposition on the orthogonal space of the span of α, and thus measuring u would now give a random sampling in the orthogonal space. We will now show that once we have a random sampling of the space, we can get a basis.

Lemma 63. *Let G be a finite group. The probability that a uniform sample of size $4\log|G|$ generates G is at least $\frac{1}{3}$.*

Proof. Let $g_1, g_2, \cdots, g_m$ be the sample. Define G_i to be the group generated by $\{g_j\}_{j\le i}$. Let X be the indicator random variable that something bad happens:

$$X_i=\begin{cases}0 & \text{if } G_{i-1}=G \text{ or } g_i\notin G_{i-1}\\ 1 & \text{otherwise}\end{cases}$$

and let $X=\sum_{i=1}^m X_i$ then

$$\begin{aligned}
\Pr[X_i=0] &= \Pr[G_{i-1}=G]+\Pr[G_i\neq G]\Pr[g_i\notin G_{i-1}|G_{i-1}\neq G]\\
&= p_i+(1-p_i)\frac{1}{2}\\
&= \frac{1}{2}+\frac{p_i}{2}\geq\frac{1}{2}\\
\implies E[X_i] &= \Pr[X_i=1]\leq\frac{1}{2}\\
\implies E[X] &= \sum_{i=1}^m E[X_i]\leq\frac{m}{2}
\end{aligned}$$

By Markov's inequality,

$$\Pr[X\geq a]\leq\frac{E[x]}{a}\leq\frac{m}{2a}$$

Choosing $a=\frac{3m}{4}$ would get give $\Pr[X\geq\frac{3m}{4}]\leq\frac{2}{3}$.

But notice that if we haven't got a generating set already, $X_i = 0$ can happen atmost $\log |G|$ many times. Thus if we were to choose $m = 4 \log |G|$, at most $\log |G|$ of the X_i can be zero and hence will force $G_m = G$ with probability atleast $\frac{1}{3}$. □

Now that we have a generating set for the orthogonal set, all we need to do right now is solve a system of linear equations to get the orthogonal space of this, which is the space of λ. Thus we would find λ with just polynomially many queries since $\log |G| \leq n$.

The same idea can be used to solve a general problem as well. We are given a function f such that there is a subspace such that $f(x \oplus H) = f(x)$. The goal is to find a subspace. The same ideas can be used to do this as well. (exercise to the reader)

72 Towards Shor's Algorithms

The techniques used in the quantum solution to Simon's problem is essential for Shor's algorithms for integer factoring and the discrete logarithm problem. Shor's algorithm is a quantum algorithm for order finding (given a numbers a and n, find $ord_n(a)$). But the following lemma would show that this is good enough.

Lemma 64. *Order finding is as hard as integer factoring.*

Proof. We will show a probabilistic reduction from factoring to order finding. Without loss of generality, we can assume that n is odd. Pick an x at random from $\mathbb{Z}_n \setminus \{0\}$. We would be extremely lucky if $\gcd(x, n) \neq 1$, we then immediately have a non-trivial factor of n. Hence, we can assume that x is randomly picked from $\mathbb{Z}_n^\star$.

Suppose $ord_n(x)$ is even, and it further satisfies the property that $x^{ord_n(x)/2} \neq -1 \pmod{n}$, then we have non-trivial square root of unity and hence $\gcd(n, x^{ord_n(x)/2} - 1)$ or $\gcd(n, x^{ord_n(x)/2} + 1)$ will be non-trivial. This is the reduction. Once we show that we find a good x with high probability, we are done.

Assume that $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$. By the chinese remaindering theorem,

$$\mathbb{Z}_n^\star = \mathbb{Z}_{p_1^{\alpha_1}}^\star \times \mathbb{Z}_{p_2^{\alpha_2}}^\star \times \cdots \mathbb{Z}_{p_k^{\alpha_k}}^\star$$

This map would take $x \mapsto (x_1, x_2, \cdots, x_k)$. Let $r = s2^t = ord_n(x)$ and $r_i = s_i 2^{t_i} = ord_{p_i^{\alpha_i}}(x_i)$. Clearly, $s = \text{lcm } s_i$ and $t = \max t_i$. Thus if r is odd, then $t = 0$ which means $t_1 = t_2 = \cdots = t_k = 0$.

Suppose r was even, we shall analyze the probability that $x^{r/2} = -1 \pmod{n}$. By the chinese remaindering theorem, $x^{r/2} = -1 \mapsto (-1, -1, \cdots, -1)$. But suppose some $t_i \neq t$, then $t_i \leq t - 1$ and hence $r_i \mid r/2$ and therefore the i-th coordinate in the tuple will have to be a 1 instead of a -1.

Therefore, the probability that r is odd and $x^{r/2} = -1 \pmod{n}$ is bounded above by $\Pr[t_1 = t_2 = \cdots = t_k] \leq 2^{k-1}$ since each t_i is independent.

Therefore $\Pr[r \text{ is even and } x^{r/2} \neq -1 \pmod{n}] \geq 1 - 2^{k-1}$. □

Over the next few lectures, we will see how we can find the order of an element using a quantum algorithm, and this would solve the integer factoring problem.

Algebra and Computation	Course Instructor: V. Arvind
Lecture 23: Shor's Algorithm for Integer Factoring	
Lecturer: V. Arvind	*Scribe: Ramprasad Saptharishi*

73 Overview

In this lecture we shall see Shor's algorithm for order finding, and therefore for integer factoring.

74 The First Steps

We are given a number $a \in \mathbb{Z}_N^\star$ and we need to find the $r = ord_N(a)$.

Pick an L such that $N^2 \leq 2^L \leq 2N^2$ and let $q = 2^L$. Hence the group $\mathbb{Z}_q$ is the set of all L-bit binary strings. As in the Simon's problem, prepare the uniform superposition using the hadamard transform.

$$|\psi\rangle = \frac{1}{\sqrt{q}} \sum_{x=0}^{q-1} |x\rangle$$

And we shall assume that our function $f : x \mapsto a^x \pmod{n}$ is given as a unitary tranform where $U_f(x, y) = (x, y \oplus f(x))$. Thus applying this matrix to the uniform superposition padded with 0s, we get:

$$\frac{1}{\sqrt{q}} \sum_{x=0}^{q-1} |x\rangle \left|0^L\right\rangle \longrightarrow \frac{1}{\sqrt{q}} \sum_{x=0}^{q-1} |x\rangle |f(x)\rangle$$

Now measuring the second register would give us some $a^l \pmod{N}$ for some least l. And hence, the current state would then be:

$$\frac{1}{\sqrt{A+1}} \sum_{j=0}^{A} |l + jr\rangle \quad , \quad A = \left\lfloor \frac{q-l-1}{r} \right\rfloor$$

Our job is to retrieve the r and hence we need to get rid of the l. This is where the fourier tranform comes in.

74.1 Fourier Tranform over $\mathbb{Z}_q$

The fourier transform over $\mathbb{Z}_q$ is the following map:

$$F_q : |y\rangle \to \frac{1}{\sqrt{q}} \sum_{c=0}^{q-1} e^{\frac{2\pi i}{q} yc} |c\rangle$$

Shor showed that there exists a polynomial sized quantum circuit for the fourier tranform. For the moment, let us take it for granted that there does exists a polynomial sized quantum circuit though we shall prove it later in the lecture.

The quantum state that we are in is

$$\frac{1}{\sqrt{A+1}} \sum_{j=0}^{A} |l + jr\rangle$$

Applying the fourier tranform to this, we have:

$$\frac{1}{\sqrt{q(A+1)}} \sum_{j=0}^{A} \sum_{c=0}^{q-1} e^{\frac{2\pi i}{2^L}(l+jr)c} |c\rangle = \frac{1}{\sqrt{q(A+1)}} \sum_{c=0}^{q-1} \alpha_c |c\rangle$$

The Easy case: $r \mid q$

Since r divides q and we chose the least l it follows that $A = \frac{q}{r} - 1$. Therefore,

$$\alpha_c = e^{\frac{2\pi i}{q} cl} \sum_{j=0}^{A} \left(e^{2\pi i \frac{cr}{q}} \right)^j$$

Suppose $q \nmid cr$ then $\omega = e^{2\pi i \frac{c}{q/r}} \neq 1$ is a q/r-th root of unity. Therefore,

$$\alpha_c = e^{\frac{2\pi i}{q} cl} \left(\sum_{j=0}^{\frac{q}{r}-1} \omega^j \right) = 0$$

Since the probability amplitude is 0 for the case when $q \nmid cr$, measuring c will give us a number that is a multiple of $\frac{q}{r}$ with uniform probability.

Hence we now will have a fraction $\frac{\lambda q}{r}$. How do we recover r from this? Suppose λ was coprime to r, we know $\frac{c}{q} = \frac{\lambda}{r}$. Since $\frac{\lambda}{r}$ is in its reduced form, just take $\frac{c}{q}$ and get it to the reduced form; the denominator would then give us the r.

How do we make sure that we get a fraction such that λ and r are coprime? With good probability we will. The probability that this happens is $\frac{\phi(r)}{r} > \frac{1}{\log n}$. Hence we are safe.

The general case:$r \nmid q$

For any c in the domain,

$$\begin{aligned}\Pr[c \text{ is measured}] &= \frac{1}{q(A+1)}\left|\sum_{j=0}^{A} e^{2\pi i \frac{c(l+rj)}{q}}\right|^2 \\ &= \frac{1}{q(A+1)}\left|\sum_{j=1}^{A} e^{2\pi i j(cr \bmod q)}\right|^2\end{aligned}$$

Let the event $E = \left\{c \ : \ \frac{-r}{2} \leq cr \bmod q \leq \frac{r}{2}\right\}$. To analyse the probability that this event will happen, for every $0 \leq \lambda \leq r$ look at the interval $\left[\lambda q - \frac{r}{2}, \lambda q + \frac{r}{2}\right]$. This interval will contain a multiple of r. There is a possibility of both end points of this interval being multiples of r. But the following argument shows that this is not possible.

$$\begin{aligned} cr &= \lambda q - \frac{r}{2} \\ (c+1)r &= \lambda q + \frac{r}{2} \\ \implies (2c+1)r &= 2\lambda q \end{aligned}$$

but the last line would force $r \geq 2q$ which is absurd by the choice of q.

Thus $|E| \geq r$ since there are r possible λ's. We now have

$$\begin{aligned} |cr - \lambda q| &\leq \frac{r}{2} \\ \left|\frac{c}{q} - \frac{\lambda}{r}\right| &\leq \frac{1}{2q} \leq \frac{1}{2N^2} \leq \frac{1}{2r^2} \end{aligned}$$

The following theorem then show how to recover r from this.

Theorem 65. *For any $\sigma \in \mathbb{Q}$, if $\left|\sigma - \frac{a}{b}\right| \leq \frac{1}{2b^2}$ then $\frac{a}{b}$ is one of the convergents*[9] *of σ.*

Thus all we need to do is get $\frac{c}{q}$, look at all its convergents and one of them will be $\frac{\lambda}{r}$. As in the earlier case, the probability that $\frac{\lambda}{r}$ will be in its reduced form will happen with good probability and thus the denominator of the fraction is the order of a.

Hence we just need to show that event E will happen with good probability.

[9] truncation of continued fractions

74.2 Bounding probability of event E

Needs to be done, didn't take good notes here.

75 Quantum Circuit for Fourier Transform

We want a circuit that transforms $|y\rangle$ to $\frac{1}{\sqrt{q}}\sum_{c=0}^{q-1} e^{\frac{2\pi i}{q}cy}|c\rangle$. Fix a c and lets its binary representation be $c_0c_1\cdots c_{L-1}$ and that of y be $y_0y_1\cdots y_{L-1}$.

$$\begin{aligned} e^{\frac{2\pi i}{2^L}cy}|c\rangle &= e^{\frac{2\pi i}{2^L}y(2^{L-1}c_0+\cdots+c_{L-1})}|c_0\rangle|c_1\rangle\cdots|c_{L-1}\rangle \\ &= \left(e^{2\pi i\frac{y}{2^L}(2^{L-1}c_0)}|c_0\rangle\right)\otimes\left(e^{2\pi i\frac{y}{2^L}(2^{L-2}c_1)}|c_1\rangle\right)\otimes\cdots\otimes\left(e^{2\pi i\frac{y}{2^L}(c_{L-1})}|c_{L-1}\rangle\right) \\ &= \left(e^{2\pi i(0.y_{L-1})c_0}|c_0\rangle\right)\otimes\left(e^{2\pi i(0.y_{L-2}y_{L-1})c_1}|c_1\rangle\right)\otimes\cdots\otimes\left(e^{2\pi i(0.y)c_{L-1}}|c_{L-1}\rangle\right) \end{aligned}$$

Thus, summing over the possible bits for each c_i we get the following tensor product.

$$\sum_{c=0}^{q-1} e^{\frac{2\pi i}{q}yc} = \left(\frac{|0\rangle+e^{2\pi i(0.y_{L-1})}|1\rangle}{\sqrt{2}}\right)\otimes\left(\frac{|0\rangle+e^{2\pi i(0.y_{L-2}y_{L-1})}|1\rangle}{\sqrt{2}}\right)\otimes\cdots\left(\frac{|0\rangle+e^{2\pi i(0.y)}|1\rangle}{\sqrt{2}}\right)$$

Define the following rotation gates:

$$R_k = \begin{pmatrix} 1 & 0 \\ 1 & e^{\frac{2\pi i}{2^k}} \end{pmatrix}$$

and its controlled version $CR_k = I\otimes R_k$, that takes in an extra bit and does the rotation only if that bit was 1. Our circuit will use the hadamard gates and these controlled rotation gates. For sake of notation $CR_k(x,y)$ will apply the rotation on x with y as the control bit.

Let us look at $\left(\frac{|0\rangle+e^{2\pi i(0.y_my_{m+1}\cdots y_{L-1})}|1\rangle}{\sqrt{2}}\right)$. Applying a hadamard transform on $|y_m\rangle$ would give us $\left(\frac{|0\rangle+e^{2\pi i(0.y_m)}}{\sqrt{2}}\right)$. Now there is a smaller rotation created by y_{m+1} only if it is equal to one. But this just amounts to rotating the present state by $2^{-(m+1)}$ controlled by y_{m+1}. And so on.
Writing it as an algorithm:

1: **for** $i = 0$ to $L-1$ **do**
2: $\quad y_i = H_1(y_i)$
3: $\quad j = 2$
4: $\quad$ **while** $(i+j-1 \le L-1)$ **do**

5: $(y_i, y_{i+j-1}) = CR_j(y_i, y_{i+j-1})$
6: **end while**
7: **end for**

(picture here would be much better)

Algebra and Computation Course Instructor: V. Arvind

Lecture 24: The Hidden Subgroup Problem

Lecturer: V. Arvind *Scribe: Ramprasad Saptharishi*

76 Overview

In this class we shall look at character theory and it's take on quantum computing. Once we have sufficient tools, we will get into the hidden subgroup problem, which can be used to solve a whole class of problems including the discrete logarithm.

77 The Hidden Subgroup Problem

The hidden subgroup problem is the natural generalization of the order finding problem.

The Problem: Let G be a finite group and $H \leq G$ be a subgroup of G. Let X be an arbitrary set and we are given a function $f : G \to X$ such that it is constant on every right coset of H ($f(x) = f(y)$ if and only if x and y belong to the same right coset of H) and is different for different right cosets.

Find a generating set for H.

77.1 Discrete Log as a hidden subgroup problem

Problem: p is a prime and g is a generator for $\mathbb{Z}_p^\star$. Given $a \in \mathbb{Z}_p^\star$ find x such that $g^x = a \pmod p$.

This can be easily converted to the HSP setting. Let G' be the additive group $(\mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}, +)$ and $f : G' \to \mathbb{Z}_p^\star$ such that it sends (α, β) to $g^\alpha a^{-\beta} \pmod p$.

It is easy to see that (α, β) goes to 1 if and only if $\alpha = x\beta$. Therefore, the hidden subgroup of this function is the subgroup generated by $(x, 1)$. Thus all we need to do is find a generator (α, β) and $\beta/\alpha = x$.

77.2 Graph Isomorphism as a hidden subgroup problem

We have seen earlier that graph isomorphism reduces to the problem of finding the automorphism group of the graph. Converting to the HSP setting is easy.

Let $\mathcal{G}_n$ be the set of all possible graphs on n nodes. The hidden function is

$$\begin{array}{rcl} f_X : S_n & \longrightarrow & \mathcal{G}_n \\ \pi & \mapsto & X^\pi \end{array}$$

that is, it takes a permutation and sends is to the graph obtained by permuting X by that permutation.

The hidden subgroup is precisely the automorphism group of the graph.

This however is a case of the HSP in a non-abelian group setting. We will just solve the problem for finite abelian groups.

78 Characters of a finite group

A character of a finite abelian group G is a homomorphism $\chi : G \to \mathbb{C}^\star$. That is, they satisfy properties like $\chi(1) = 1$,$\chi(g_1 g_2) = \chi(g_1)\chi(g_2)$,$\chi(g^{-1}) = \overline{\chi(g)}$ etc.

Define $\mathbb{C}[G]$ to be the $|G|$ dimensional vector space over $\mathbb{C}$, by just consider the elements of G as the standard basis elements of the vector space. It in fact also has a multiplicative structure and is called a group algebra. Note that the vector space for the quantum algorithms was $\mathbb{C}^{2^n} = \mathbb{C}[\mathbb{Z}_2^n]$ and at some points we even exploited the group structure of $\mathbb{Z}_2^n$. And the standard basis for the quantum setting were $\{|g\rangle \ : \ g \in G\}$ which is precisely $\mathsf{G}[G]$.

Now notice that $\mathbb{C}[G]$ can be thought of as a function from $\mathbb{C}$ to G, where every coordinate of the basis element can be thought of as the value of the function. Thus $\mathbb{C}[G] = \mathbb{C}^G$. In this setting, the characters, being functions from G to $\mathbb{C}$, can be thought of as vectors in $\mathbb{C}[G]$.

78.1 Properties of Characters

- It is easy to see that for every $g \in G$, $\chi(g)^{|G|} = 1$ since χ is a homomorphism. Hence, $\chi(g)$ is a $|G|$-th root of unity.

  Thus characters are vectors where each coordinate is a $|G|$-th root of unity. The vector $(1, 1, \cdots, 1)$ is referred to as the trivial character.

- As in the quantum setting, we shall normalize characters by writing them as
$$|\chi\rangle = \frac{1}{\sqrt{G}} \sum_g \chi(g) |g\rangle$$
By the usual hermitian inner product ($\langle a|b\rangle = \sum \overline{a_i} b_i$), it is clear that $\langle \chi|\chi\rangle = 1$.
Thus characters are vectors of norm 1.

- Suppose we have two distinct characters χ_1, χ_2, that is there exists an h such that $\chi_1(h) \neq \chi_2(h)$. Let us look at what happens to $\langle \chi_1|\chi_2\rangle$. Multiplying both sides by $\chi_1(h)$:
$$\begin{aligned}
\chi_1(h) \langle \chi_1|\chi_2\rangle &= \frac{1}{\sqrt{G}} \sum_g \chi_1(h)\chi_1(g^{-1})\chi_2(g) \\
&= \frac{1}{\sqrt{G}} \sum_g \chi_1(hg^{-1})\chi_2(g) \\
&= \frac{1}{\sqrt{G}} \sum_{\tilde{g}} \chi_1(\tilde{g}^{-1})\chi_2(\tilde{g}h) \quad , \quad \tilde{g} = gh^{-1} \\
&= \chi_2(h) \left(\frac{1}{\sqrt{G}} \sum_{\tilde{g}} \chi_1(\tilde{g}^{-1})\chi_2(\tilde{g}) \right) \\
&= \chi_2(h) \langle \chi_1|\chi_2\rangle
\end{aligned}$$
But since we assumed that $\chi_1(h) \neq \chi_2(h)$, this will force $\langle \chi_1|\chi_2\rangle = 0$. Thus the characters are mutually orthogonal to each other.
And hence, for any non-trivial character χ, $\langle (1, 1, \cdots, 1)|\chi\rangle = 0$ and hence $\sum_g \chi(g) = 0$.

Therefore it is clear that there are at most $|G|$ characters (a $|G|$ dimensional space can have at most that many mutually orthogonal vectors). For the finite abelian group setting, it is easy to show that there are in fact $|G|$ many characters.

Theorem 66 (Structure theorem for finite abelian groups)**.** *Any finite abelian group G is isomorphic to a direct product of cyclic groups.*

Thus
$$G \cong \mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2} \times \cdots \times \mathbb{Z}_{N_l}$$

For a cyclic group $\mathbb{Z}_N$, it is easy to show that we indeed have N characters:

$$\begin{aligned} \omega_N &= e^{\frac{2\pi i}{N}} \\ \chi_j : \mathbb{Z}_n &\longrightarrow \mathbb{C}^\star \\ 1 &\mapsto \omega_N^j \\ k &\mapsto \omega_N^{jk} \end{aligned}$$

And clearly these are distinct. In the same way, we have $|G|$ distinct characters by just saying:

$$\chi_{j_1,j_2,\cdots,j_l}(a_1, a_2, \cdots, a_l) \mapsto (\omega_{N_1})^{j_1 a_1} (\omega_{N_2})^{j_2 a_2} \cdots (\omega_{N_l})^{j_l a_l}$$

Thus, the characters indeed form an orthonormal basis for $\mathbb{C}[G]$.

78.2 The fourier transform

The fourier transform is just the change of basis from the standard to the characters. And the transform played an important role in the order finding algorithm due to the property of 'shift invariance that the character basis enjoys.

$$\begin{aligned} |\chi_g\rangle &= \frac{1}{\sqrt{G}} \sum_x \chi_g(x) |x\rangle \\ U_h \chi_g &= \frac{1}{\sqrt{G}} \sum_x \chi_g(x) |hx\rangle \\ &= \frac{1}{\sqrt{G}} \sum_x \chi_g(h^{-1}) \chi_g(hx) |hx\rangle \\ &= \chi_g(h^{-1}) \left(\frac{1}{\sqrt{G}} \sum_{x'} \chi_g(x') |x'\rangle \right) \quad , \quad x' = hx \\ &= \chi_g(h^{-1}) |\chi_g\rangle \end{aligned}$$

The fourier basis are all eigenvectors for all shift operators U_h, the eigenvalue being $\chi_g(h^{-1})$.

79 The Hidden Subgroup Problem for Finite Abelian Groups

The group is given to us as $\mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2} \times \cdots \times \mathbb{Z}_{N_l}$. And we need to find the hidden subgroup of G.

As in the Simon's problem and Shor's algorithm, first create the uniform superposition

$$|\psi\rangle = \frac{1}{\sqrt{G}} \sum_g |g\rangle$$

How we create this is a lovely trick that we shall see later in this lecture. Another thing we will assume that we can do a fourier transform (approximate at least) efficiently. Applying the function to the padded version, we get

$$\frac{1}{\sqrt{G}} \sum_g |g\rangle |f(g)\rangle$$

On measuring the second qubits, we would measure some $f(x)$ and thus would result in the state

$$\frac{1}{\sqrt{H}} \sum_h |xh\rangle$$

A fourier transform on this gives

$$\frac{1}{\sqrt{H}\sqrt{G}} \sum_h \sum_g \chi_{xh}(g) |g\rangle$$

Note that $\chi_a(b) = \chi_b(a)$. And hence

$$\begin{aligned} \frac{1}{\sqrt{H}\sqrt{G}} \sum_h \sum_g \chi_{xh}(g) |g\rangle &= \frac{1}{\sqrt{H}\sqrt{G}} \sum_h \sum_g \chi_g(xh) |g\rangle \\ &= \frac{\sqrt{H}\sqrt{G}}{\sum_g} \left(\sum_h \chi_g(h) \right) \chi_g(x) |g\rangle \end{aligned}$$

Now, since $\chi_g(h)$ is a character of H as well the summation inside the bracket will be zero for a lot of χs.

At this point, for any group G, define the dual group G' as the group of characters of G. $H^\perp = \{\chi \in G' \ : \ \chi(h) = 1 \forall h \in H\}$.

For all characters in $H^\perp$, the summation in the bracket will be $|H|$, and 0 otherwise. Hence the summation reduces to

$$\frac{\sqrt{H}}{\sqrt{G}} \sum_{g:\chi_g \in H^\perp} \chi_g(x) |g\rangle$$

Now measuring $|g\rangle$ will give us a random element in $H^\perp$. Thus using the sampling lemma in Simon's problem we can get a generating set for $H^\perp$. With this, how do we find a generating set for H?

Suppose we have our sample $g_1, g_2, \cdots, g_t$ where $t = 4 \log |G|$. By the structure of the group G, $g_i = \langle a_{i_1}, a_{i_2}, \cdots, a_{i_l} \rangle$. Thus for each $x_i \in H$ we know that

$$\omega_{N_1}^{x_1 a_{i_1}} \omega_{N_2}^{x_2 a_{i_2}} \cdots \omega_{N_l}^{x_1 a_{i_l}} = 1$$

But this is an exponential constraint, if we have a linear constraint we can solve it using the techniques discussed earlier.

Let $N = lcm(N_i)$ and let $M_i = N/N_i$. Then the constraint above is just finding solutions x_i to $\sum_j M_j a_{i_j} x_j = 0 \pmod N$. The mod can be removed as well by having an extra indeterminate y_i and writing it as

$$M_1 a_{i_1} x_1 + M_2 a_{i_2} x_2 + \cdots M_l a_{i_l} x_l + N y_i = 0$$

These constraints, for each i, is just a system of diophantine equations that can be solved using the hermite normal form. Thus, this would solve the hidden subgroup problem.

79.1 The Converse

Another important question is the following: suppose we have a way of solving the hidden subgroup problem for a finite abelian group, can we use that to find the structure of G?

One way is to take the generators of G (by random sampling), finding their orders and factorizing them. The factorization of the orders will decompose the group in to a direct product of p-groups. How do we find the cyclic product decomposition of the p-groups?

We shall discuss this in the next lecture.

79.2 Creating the uniform superposition

We want to create the state

$$|\psi\rangle = \frac{1}{\sqrt{G}} \sum_g |g\rangle$$

The idea is to find a binary encoding of the group and use that. Encode elements of G using binary strings of length m, m chosen such that $2^m \geq |G| \geq 2^m / poly(m)$ (a reasonably efficient encoding). Once we have an encoding function, we naturally have another checker functions U_f that takes a binary string and decides whether it is actually an encoding of an element of G.

Using the hadamard transform, we can create a uniform superposition over $\{0,1\}^m$:

$$|\psi\rangle = \frac{1}{2^m} \sum_{x \in \{0,1\}^m} |x\rangle$$

Applying the checker function to the padded version of this, we get

$$\frac{1}{2^m} \sum_{x \in \{0,1\}^m} |x\rangle \, |f(x)\rangle$$

Now since we assumed that the encoding is reasonably efficient, measuring $f(x)$ will give us a 1 with high probability. And hence, the rest of the state will collapse to

$$\frac{1}{\sqrt{G}} \sum_{g} |g\rangle$$

which is precisely what we want!

Algebra and Computation	Course Instructor: V. Arvind
Lecture 25: Needle in a Haystack: Grover Search	
Lecturer: V. Arvind	*Scribe: Ramprasad Saptharishi*

80 Overview

In this lecture, we shall look at another problem where quantum algorithms do better than classical algorithms. This, unlike the DJ problem, is deep and is truly one where the quantum model beats the classical model.

81 Grover's Search

The problem is the following. You are given a function $f : \{0,1\} \longrightarrow \{0,1\}$ as an oracle with the promise $f(x) = 1$ for precisely one x say x_0. The problem is to find the x_0. We want to do this with as few queries as possible.

This is equivalent to searching in an unordered list; the value x_0 is hidden in the list and you want to find it.

81.1 Lower bounds on classical models

It is clear that a deterministic algorithm will take $O(2^n)$ queries.

It is easy to show that even for randomized algorithms, it would need $O(2^n)$ queries to make the error probability bounded by a constant. We shall leave the proof to the interested reader.

81.2 The Quantum Model

Grover presented a quantum algorithms that makes $O\left(2^{n/2}poly(n)\right)$ queries and finds x_0 with error probability bounded by a constant.

82 The Algorithm

The basic idea is the following property:

Let M_1 and M_2 are two lines through the origin in the plane, and the angle separating them being α and let P is any point on the plane. If you reflect P about M_1 and then that reflection about M_2, the resultant point

is at an angle 2α from P in the direction M_1M_2.

Our setting is going to be like this. We will be working in $\mathbb{C}^{2^n}$ and there is one special coordinate axis labelled by x_0 that we want to find. Using the function provided as oracle, we can get the unitary transform of the rotation about x_0 eventhough we do not know what x_0 is. The idea is to take another vector whose angle with x_0 is known and use this rotation technique to get closer to x_0. Since we know the angles, we know precisely how many rotations need to be done and that will give us a vector with a very high probability amplitude on the x_0 coordinate at which point we can make a measurement.

82.1 The Reflection Maps

Consider the uniform superposition $|\psi\rangle = \frac{1}{N}\sum_x |x\rangle$. We know that it makes an angle of $\cos^{-1}(2^{-n/2})$ with each coordinate axis.

The unitary map given to us works as we have assumed always. And we have also seen that $|x\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)$ goes to $(-1)^{f(x)}|x\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)$. By just dropping the second coordinate, we can think of this as a map

$$I_{x_0} : |x\rangle \mapsto (-1)^{f(x)}|x\rangle$$

This is just the identity matrix with a -1 on the (x_0, x_0) entry. This can be compactly written as $I_{x_0} = I - 2|x_0\rangle\langle x_0|$. This is just reflection about the plan perpendicular to x_0.

Observation 67. *If U is any unitary operator, then $I_{U|0^n\rangle} = UI_{0^n}U^{-1}$.*

Proof. It follows from the observation that $I_x = I - 2|x\rangle\langle x|$. □

Observation 68. *For any states $|\phi\rangle$ and $|\psi\rangle$, I_ψ preserves the span of $|\phi\rangle, |\psi\rangle$.*

In particular, $I_{U|0^n\rangle}$ and I_{x_0} preserve the span S of x_0 and $U|0^n\rangle$.

Observation 69. *Let $|e_1\rangle = U|0^n\rangle$ and let e_2 be anything in S that is perpendicular to e_1. We can pull out the $e^{i\theta}$ factor out of e_i so that $\langle e_i, x_0\rangle$ is real. Hence we now have*

$$S = \{a|e_1\rangle + b|e_2\rangle \ : \ a, b \in \mathsf{R}\}$$

Observation 70. *Let $v \in S$ and let $v^\perp$ be another in S that is orthogonal to v in S. Then $I_v = -I_{v^\perp}$.*

Thus, we need not know what x_0 is but by just using I_{x_0} we can achieve the reflection about x_0. And let U be the hadamard transform and $U|0^n\rangle$ will now be the uniform superposition $|\psi\rangle$.

82.2 The Double Reflection

We know that $U0^n$ make an angle of $\cos^{-1}(2^{-n/2})$ with each coordinate axis, and this is almost a right angle. Thus instead of $U|0^n\rangle$, we shall look at the orthogonal vector $U|0^n\rangle^{\perp}$ in the span S and the reflection. The angle that x_0 makes with this $\sin^{-1}(2^{-n/2})$ which is very small. Now, the unitary transform that achieves the rotation about x_0 and then about $U|0^n\rangle^{\perp}$ is just $-UI_{0^n}UI_{x_0}$.

Now look at the uniform superposition $|\psi\rangle = \frac{1}{N}\sum_x |x\rangle$. This is in the plane and the double rotation will rotate it by an angle of $2\sin^{-1}(2^{-n/2})$. Thus with just $\frac{\pi}{4}\sqrt{2^n}$ such double rotations we would get very close to x_0.

At this point, we can safely make a measurement and obtain x_0 with high probability.

83 Implications on SAT

The problem can be transformed to a decision problem. We are given a function f as an oracle and we need to determine if f is 1 at atleast one x. This is certainly easier than determining the x_0 as we have seen.

A result of Valiant and Vazirani shows that any SAT instance ϕ can be reduced to another instance ϕ' in randomized polynomail time such that ϕ is satisfiable implies ϕ' has exactly one satisfiable instance, and ϕ is unsatisfiable implies ϕ' is also unsatisfiable.

Grover's algorithm shows that SAT can be solved in $O(\sqrt{2}^n)$ time using a quantum algorithm.

84 A glimpse into the finale

In the next class, we shall show that Grover's algorithm is infact tight, any quantum algorithm that has error probability bounded by a constant must take $O(2^{n/2})$ queries.

We shall prove this using two beautiful lower bound techniques in the oracle setting.