Chapter 3. List of Elements

This chapter develops Dimino's algorithm for constructing a list of elements of a group given by a set of generators. At each stage in the development, we will analyse the algorithms, and indicate the knowledge that is used by the algorithm.

Obvious Approach from First Principles

Suppose we are given a set of generators S of a group G. How do we determine a list of the elements of G, presuming this list is small enough to store in our computer? To answer this, let us consider what we know about groups and generators. We know that

(a) the group has an identity element;
(b) the set of generators S is a subset of the group G;
(c) a group is closed under multiplication, so that if we know two elements g and h in G then we must ensure their product $g \times h$ is in G; and
(d) the group G is the smallest set containing S which is closed under multiplication.

This suggests Algorithm 1.

Algorithm 1 : Closing under multiplication of elements

Input : a set S of generators of a group G;

Output : a list of the elements of the group G;

```
begin
  list := id ∪ elements of S;
  for each pair (g, h) of elements in list do
    if g × h is not in list then
      append g × h to list;
    end if;
  end for;
end;
```

To analyse the algorithm, there are $|G|^2$ pairs of elements (g, h), and for each pair we form one multiplication and perform one search of the list. The cost of appending an element to the list is insignificant, and occurs only $|G|$ times. Thus the total cost is

$|G|^2$ multiplications, and

$|G|^2$ searches of the list of elements.

The order of the group - and the length of the list - may be as large as 10 000. Therefore the searches must be performed as efficiently as possible. Today's implementations use a hash list of elements, so the average cost of a search is between 1.2 and 2 comparisons of elements. For permutations, we can assume that a comparison is equal in cost to a multiplication. In our analysis, we will assume the cost of a search is the same cost as two multiplications.

A Trivial Improvement

The algorithm may be improved slightly by observing one extra piece of information, namely

(e) for all elements g, $g \times id = id \times g = g$.

Hence, we should not consider the identity element as one of the pair of elements when we are closing the list of elements under multiplication. Alas, on analysis, we only reduce the $|G|^2$ factor to $(|G|-1)^2$.

Although this is not a major improvement, it does illustrate how additional knowledge can lead to improvement of algorithms. Our next example is more worthwhile.

Induction on the Length of Products

The next piece of knowledge we will draw upon comes from the definition of a set of generators. So far, we have only used the fact that the generators form a subset, and from them closure determines all the elements. The definition, however, says more. It says that

(f) each element of the group can be expressed as a product

$$s_{i_1} \times s_{i_2} \times \cdots \times s_{i_m}$$

of generators.

If we decompose the product into the first m-1 terms and the last term, then we may proceed by induction on the length of the product. Since (f) says that every element expressible as a product of length m is a product $g \times s$ of a generator s and an element g expressible as a product of length m-1, we modify our previous algorithm to only consider pairs of elements (g, h) where the second element is a generator. The new algorithm is Algorithm 2.

Algorithm 2 : Closing under multiplication with generators

Input : a set S of generators of a group G;

Output : a list of elements of the group G;

```
begin
  list := {id} ∪ elements of S;
  for each element g in list do
    for each generator s do
      if g × s is not in list then
        append g × s to list;
      end if;
    end for;
  end for;
end;
```

The analysis reveals that the algorithm requires

$|G| \times |S|$ multiplications, and

$|G| \times |S|$ searches.

If we work through the example of the symmetries of the square generated by $\{elt[2], elt[5]\}$ we obtain the following list of elements. We note the product that first indicated that the element is in the group.

$elt[1]$, identity
$elt[2]$, generator
$elt[5]$, generator
$elt[3] = elt[2] \times elt[2]$
$elt[6] = elt[2] \times elt[5]$
$elt[8] = elt[5] \times elt[2]$
$elt[4] = elt[3] \times elt[2]$
$elt[7] = elt[3] \times elt[5]$

The number of multiplications performed is $16 = 8 \times 2$, as against 64 for Algorithm 1.

One Generator Groups

An important special case of the above algorithm occurs when there is only one generator. In this case we know the elements of the group are the powers of the generator s. This allows us to avoid searches in this case, by performing comparisons with the identity instead. One generator groups occur infrequently, so it would not pay to make any modifications to the above algorithm, if it was our general purpose algorithm for constructing lists of elements. However, one generator groups always play a role in Dimino's algorithm, and this special case is the best we can do for one generator groups.

One generator groups are usually called *cyclic* groups.

Induction on the Generators

Algorithm 2 formed the elements of the group, expressible as products

$$s_{i_1} \times s_{i_2} \times \cdots \times s_{i_m}$$

of the generators, in order of increasing length of the product. Dimino's algorithm forms the elements in order of the generators involved in the product. Thus, if $S = \{s_1, s_2, \cdots, s_t\}$ is the set of generators, all the elements involving the first i-1 generators are formed before any element that must involve the i-th generator s_i. The usefulness of this approach stems from the fact that the elements that involve only the first i generators form a subgroup. Now we can use our knowledge of subgroups and, in particular, cosets!

Some notation is required. Let $S_i = \{s_1, s_2, \cdots, s_i\}$, and let $H_i = \langle S_i \rangle$. For convenience, we let S_0 be the empty set, and H_0 be the identity subgroup. The typical inductive step of Dimino's algorithm is, given a list of elements for H_{i-1}, determine the list of elements for H_i.

Our knowledge of subgroups includes the fact that

(g) a subgroup is a group, and therefore is closed under multiplication.

In terms of the inductive step, this means that we know the product $g \times s$ is in the list, for all g in H_{i-1} and s in S_{i-1}. We could reorganise Algorithm 2 slightly so that, during the step from H_{i-1} to H_i, only the products $g \times s_i$, for g in H_{i-1}, and $g \times s$, for g not in H_{i-1} and s in S_i are considered, but this reorganisation would not save us anything over Algorithm 2.

The important facts about cosets are that

(h) the cosets of a subgroup partition a group, and

(i) all cosets have the same size as the subgroup. A coset can be formed by multiplying the elements of the subgroup by a coset representative.

Thus, if we know the elements of H_{i-1}, then whenever an element g is added to the list of elements for H_i we can in fact add the whole coset $H_{i-1} \times g$ to the list. If we extend the list of elements one coset at a time, then we know that all the elements of the coset $H_{i-1} \times g$ are not in the list once we know g is not in the list. This opens up the possibility of adding elements to the list while avoiding a lot of searching. However, the searches in Algorithm 2 come about in considering products $g \times s$ and determining if they are new elements. If we add cosets rather than elements we are still left with considering products $g \times s$ to find new coset representatives, so we have not avoided any searching. For the addition of cosets to be useful we need a better method of finding coset representatives than considering all products of elements and generators.

A new fact about cosets is needed. We have not met this fact before but it is easy to prove.

Lemma

Let H be a subgroup of G. Let $H \times g$ be a coset of H in G and let $s \in G$. Then all the elements of $(H \times g) \times s$ lie in the coset of H with representative $g \times s$.

The immediate consequence of this lemma is that we can find the coset representatives of H_{i-1} by considering the products $g \times s$ where s is in S_i and g is a previously found coset representative, rather than an arbitrary element. This is a major reduction in the number of products to be considered.

Taking the identity to be the coset representative of H_{i-1} gives Algorithm 3 for the inductive step of Dimino's algorithm.

Algorithm 3 : Inductive Step of Dimino's Algorithm

Input : a set S of generators of a group G;
a list of elements of H_{i-1};

Output : a list of elements of H_i;
begin
coset_reps := { *id* };
for each g in *coset_reps* **do**
for each generator s in S_i **do**
if $g \times s$ is not in *list* **then**
append $g \times s$ to *coset_reps*;
append $H_{i-1} \times (g \times s)$ to *list*;
end if;
end for;
end for;
end;

To analyse Algorithm 3, we note that there are i generators in S_i; there are $|H_i:H_{i-1}|$ cosets representatives; and there are $|H_i| - |H_{i-1}|$ elements added to the list. Thus there are $i \times |H_i:H_{i-1}|$ products formed when finding coset representatives, and $|H_i| - |H_{i-1}|$ products formed when appending elements to the list. Hence, there are a total of

$i \times |H_i:H_{i-1}| + (|H_i| - |H_{i-1}|)$ multiplications, and

$i \times |H_i:H_{i-1}|$ searches.

If we are careful about forming the elements of the coset then we do not form the coset representative twice. That is, when appending the coset to the end of the list, do not form the product of the identity and the coset representative. This reduces the number of multiplications by $|H_i:H_{i-1}| - 1$. Furthermore, only the product $id \times s_i$, that is s_i, has to be considered for the initial coset representative *id*. This reduces the number of multiplications by i.

Let us work through the second step for the example where G is the symmetries of the square generated by {*elt*[2], *elt*[5]}. Initially the list is

elt[1] *elt*[2] *elt*[3] *elt*[4]

and the coset representatives are

elt[1]

The generator $s_2 = elt[5]$ is not in the list, so it is the next coset representative. This appends the remainder of the elements to the list. The algorithm forms the products *elt*[5] × *elt*[2] and *elt*[5] × *elt*[5] and verifies that they are in the list. The algorithm terminates.

Let us consider the same group, but with generators {*elt*[5], *elt*[2]}. Again we perform the second inductive step. The list initially is

$elt[1]\ elt[5]$

and the coset representatives are

$elt[1]$

The generator $elt[2]$ is the next coset representative, and its coset is added to the list. Now the list is

$elt[1]\ elt[5]\ elt[2]\ elt[6]$

and the coset representatives are

$elt[1]\ elt[2]$

The product $elt[2] \times elt[5]$ is $elt[6]$, which is in the list. The product $elt[2] \times elt[2]$ is $elt[3]$, which is the next coset representative. Its coset is added to the list giving

$elt[1]\ elt[5]\ elt[2]\ elt[6]\ elt[3]\ elt[7]$

and the coset representatives are

$elt[1]\ elt[2]\ elt[3]$

The product $elt[3] \times elt[5]$ is $elt[7]$, which is in the list. The product $elt[3] \times elt[2]$ is $elt[4]$, which is the next coset representative. Its coset is added to the list completing the group. The products $elt[4] \times elt[5]$ and $elt[4] \times elt[2]$ are formed and found to be in the list. The algorithm terminates.

Simple Algorithm of Dimino

We now incorporate the inductive step with a few modifications to obtain the simple version of Dimino's algorithm. The first inductive step involves a one generator group, so we treat this as a special case and do not use Algorithm 3 at this step. Instead we use the modified Algorithm 2 discussed in the previous section. If the first element in the list is the identity then the first element of a coset will always be the coset representative. Each coset has size $|H_{i-1}|$ during the i-th step. Hence, we can easily locate the coset representatives in the list, and so we do not store them separately. We also incorporate a check on the redundancy of the generators. There is no inductive step if a generator is already in the list. The simple algorithm of Dimino is Algorithm 4.

Algorithm 4 : Simple Dimino's Algorithm

Input : a set $S = \{s_1, s_2, \ldots, s_t\}$ of generators of a group G;

Output : a list of elements of the group G;

```
begin
  (*Treat the special case <s1>*)
  order := 1; elements[1] := id;
  g := s1;
  while not (g = id) do
    order := order + 1; element[order] := g;
    g := g × s1;
  end while;

  (*Treat remaining inductive levels*)
  for i := 2 to t do
    if not si in elements then (*next generator is not redundant*)
      previous_order := order; (*i.e. |Hi-1|*)
      (*first useful coset representative is si - add a coset*)
      order := order + 1; elements[order] := si;
      for j := 2 to previous_order do
        order := order + 1; elements[order] := elements[j] × si;
      end for;

      (*get coset representative's position*)
      rep_pos := previous_order + 1;
      repeat
        for each s in Si do
          elt := elements[rep_pos] × s;
          if not elt in elements then (*add coset*)
            order := order + 1; elements[order] := elt;
            for j := 2 to previous_order do
              order := order + 1; elements[order] := elements[j] × elt;
            end for;
          end if;
        end for;
        (*get position of next coset representative*)
        rep_pos := rep_pos + previous_order;
      until rep_pos > order;

    end if; (* not si ... *)
  end for; (* i := 2 ... *)
end;
```

Let $N_i = |H_i{:}H_{i-1}|$, so that $N_1 = |H_1|$ and $|G| = \prod_{i=1}^{t} N_i$. If we assume there are no redundant generators (and if we omit the t-1 searches associated with the redundancy checking), then the total number of multiplications for Algorithm 4 is N_1 for the initial

special case plus the sum of the inductive steps. The inductive steps are careful to avoid the multiplications mentioned in the previous section, so the total number of multiplications is

$$N_1 + \sum_{i=2}^{t} \left[i \times (N_i - 1) + (N_i - 1) \times (|H_{i-1}| - 1) \right]$$
$$= N_1 + \sum_{i=2}^{t} \left[(i-1) \times (N_i - 1) \right] + \sum_{i=2}^{t} \left[|H_i| - |H_{i-1}| \right].$$

The last summation evaluates to $|G| - |H_1|$, so the total is

$$|G| + \sum_{i=2}^{t} \left[(i-1) \times (N_i - 1) \right].$$

For the common case, where $t = 2$, the number of multiplications is maximised when $N_1 = 2$, and $N_2 = |G|/2$. The maximum is $1.5 \times |G| - 2$ multiplications.

An analysis of the number of searches gives N_1 comparisons with the identity, for the special case, plus $\sum_{i=2}^{t} \left[i \times (N_i - 1) \right]$ searches for the inductive steps. This is maximised as above for the case when $t = 2$ to be approximately $|G|$ searches.

As an example, let us consider the symmetric group on {1,2,3,4}. The group has order 24. The generators we choose are (1,2), (3,4), (1,3,2,4), and (2,3,4). This gives values of $N_1 = 2$, $N_2 = 2$, $N_3 = 2$, and $N_4 = 3$. The subgroups form a chain of subgroups

$$id < H_1 < H_2 < H_3 < H_4 = G$$

which provides the basis of the induction. The aim of Dimino's algorithm is to make the cost dependent on the sum of the sizes of the steps from H_{i-1} to H_i - that is, on $\sum_{i=1}^{4} N_i$ - instead on the total size of the group - that is, on $\prod_{i=1}^{4} N_i$. The cost of Algorithm 2 for this example is 96 multiplications and 96 searches, which is equivalent to a total of 288 multiplications. The cost of Algorithm 4 is 33 multiplications, 2 comparisons, and 13 searches, which is equivalent to a total of 61 multiplications.

Complete Algorithm of Dimino

There is still a minor improvement that can be made to the algorithm. However, we need the concept of a normal subgroup. A subgroup H of G is *normal* in G if $H \times g = g \times H$, for all elements g of G. A consequence of this is that if $G = \langle H, g \rangle$ then the coset representatives of H in G can be taken to be

$$id, g, g^2, \ldots, g^m$$

where m is the smallest integer such that $g^{m+1} \in H$. We can test if H is normal in $\langle H, g \rangle$ by testing if all the elements $g^{-1} \times s \times g$, for s a generator of H, are in H. If so, then H is normal.

The cost of testing if H_{i-1} is normal in H_i is $2 \times (i\text{-}1)$ multiplications to form the elements above, and $(i-1)$ searches to test if they are in the subgroup. The cost of the inductive step if H_{i-1} is normal in H_i is $(N_i - 1) \times (|H_{i-1}| - 1)$ multiplications to add the cosets, $N_i - 1$ multiplications to form the powers of s_i as coset representatives, and $N_i - 1$ searches to see if

the powers are in the subgroup. This saves $(i-1) \times (N_i - 1)$ multiplications and $(i-1) \times (N_i - 1)$ searches over the typical inductive step (ignoring the cost of normality testing).

For the example of the symmetric group on {1,2,3,4} given by four generators, the complete Dimino's algorithm works out to be more expensive. It requires 12 multiplications and 6 searches while testing normality. The first two inductive steps do involve normal subgroups. Their cost (outside of normality testing) is 6 multiplications and 2 searches. The initial special case for the cyclic group, and the last step cost 24 multiplications, 2 comparisons, and 8 searches, outside of normality testing. Hence, the total cost is equivalent to 76 multiplications.

Uses

A list of elements allows us to answer two vital questions about a group.

(1) What is the order of the group?

(2) Is a given permutation in the group?

A list of elements defines a bijection between the group and the integers from 1 to $|G|$. Representing an element by its position in the list is more space efficient and often more convenient than representing it as a permutation.

Using the bijection, a subset or subgroup of a group can be represented as a characteristic function (or bitstring) of length $|G|$ bits. The i-th bit is set if and only if the i-th element is in the subset.

The list of elements provides a simple means of running through the elements of the group without repetition. This is useful in many other algorithms, particularly those involving searches of the group.

Summary

Obtaining a list of elements is the first step in studying a small group, and Dimino's algorithm is the best known algorithm for obtaining the list. The summation terms in the analysis of Dimino's algorithm can be crudely bounded by the order of the group. Therefore, the total cost is never more than $2 \times |G|$ multiplications and $|G|$ searches.

Exercises

(1/Easy) For the small examples of Chapter 2, choose a suitable set of generators and construct a list of elements.

(2/Easy) The symmetries of the projective plane of order two may be generated by a=(4,6)(5,7), b=(4,5)(6,7), c=(2,6,4)(3,7,5), d=(2,4)(3,5), e=(1,2,4,5,7,3,6). The relevant indices are $N_1=2$, $N_2=2$, $N_3=3$, $N_4=2$, $N_5=7$. Noting that all steps except the last have H_{i-1} normal in H_i, what is the cost of the complete Dimino's algorithm?

How does it compare with the cost of the simple Dimino's algorithm?

Conclude that the index of a normal step must be greater than or equal to three for there to be any gain in using the complete algorithm.

Bibliographical Remarks

The earlier algorithms in this chapter have been suggested at various times in the literature. For example, an algorithm that multiplies every pair of elements is presented in Yu P. Vasil'ev, *"One method of computer-aided description of the lattice of subgroups of a finite group"*, Kibernetika **2** (1978) 131-133. There is also an algorithm that was the state of the art until Dimino's algorithm. It is due to Joachim Neubüser, *"Untersuchungen des Untergruppenverbandes endlicher Gruppen auf einer programmgesteuerten elektronisch Dualmaschine"*, Numerische Mathematik **2** (1960) 280-292.

Dimino's algorithm was developed by Lou Dimino of Bell Laboratories around 1970. He did not publish his algorithm, but through John Cannon it found its way into the GROUP system and into the folklore of the field. The first description in the literature is in Harald Steinmann, **Ein transportables Programm zur Bestimmung des Untergruppenverbandes von endlichen auflösbaren Gruppen**, Diplomarbeit, Aachen, 1974.

Details of hash searching, including the average number of comparisons, can be found in D. E. Knuth, **The Art of Computer Programming, volume 3 : Sorting and Searching**, Addison-Wesley, Reading, Massachusetts, 1973.

A variation on Dimino's algorithm to compute the list of elements in a double coset $H \times g \times K$, appears in G. Butler, *"Double cosets and searching small groups"*, **SYMSAC'81** (Proceedings of the 1981 ACM Symposium on Symbolic and Algebraic Computation, Snowbird, Utah, August 5-7, 1981) (P. Wang, ed.), ACM, 1981, 182-187.