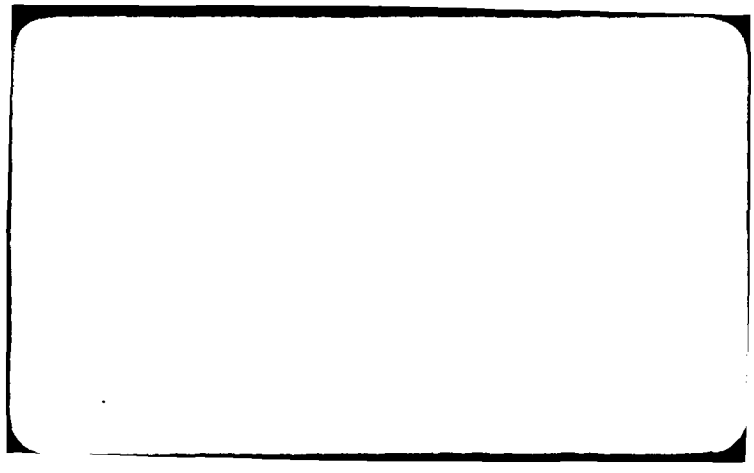
AD-A097 825 CORNELL UNIV ITHACA N Y DEPT OF COMPUTER SCIENCE F/G 12/1
POLYNOMIAL-TIME ALGORITHMS FOR PERMUTATION GROUPS.(U)
OCT 80 M FURST, J HOPCROFT, E LUKS
UNCLASSIFIED CU-CSD-TR-80-442 N00014-76-C-0018 NL

END
DATE FILMED
5 81
DTIC



(12)

(9) Technical rept.

(6) Polynomial-Time Algorithms for Permutation Groups.

(12) 29

(10) Merrick/Furst[1]
John/Hopcroft[2]
Eugene/Luks[3]

(11) Oct 80

(14) CU-CSD-TR-80-442

(15) N00014-76-C-0018,
NSF-SPI79-14927

Department of Computer Science
Cornell University
Ithaca, New York 14853

[1] Department of Computer Science, Carnegie-Mellon University, Pittsburgh Pennsylvania
[2] Department of Computer Science, Cornell University, Ithaca, New York
[3] Department of Mathematics, Bucknell University, Lewisburg, Pennsylvania

This research was supported in part by ONR contract N00014-76-C-0018 and NSF grant SPI-7914927.

407072

Polynomial-Time Algorithms for Permutation Groups

Merrick Furst[1]

John Hopcroft[2]

Eugene Luks[3]

Abstract

A permutation group on n letters may always be represented by a small set of generators, even though its size may be exponential in n. We show that it is practical to use such a representation since many problems such as membership testing, equality testing, and inclusion testing are decidable in polynomial time. In addition, we demonstrate that the normal closure of a subgroup can be computed in polynomial time, and that this procedure can be used to test a group for solvability. We also describe an approach to computing the intersection of two groups. The procedures and techniques have wide applicability and have recently been used to improve many graph isomorphism algorithms.

[1] Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania

[2] Department of Computer Science, Cornell University, Ithaca, New York

[3] Department of Mathematics, Bucknell University, Lewisburg, Pennsylvania

This research was supported in part by ONR contract N00014-76-C-0018, and NSF grant SPI-7914927.

Introduction

Let $g_1,\ldots,g_k$ be permutations of the set $\{1,\ldots,n\}$. The collection of permutations expressible as finite compositions of these g_i forms a group; a group whose size may easily be exponential in n and k. For example, take the set S_n of all n! permutations. It can be generated by just two permutations, a cyclic shift of the n letters, and a transposition of the letters 1 and 2. In fact, every subgroup G of S_n can be represented succinctly by $O(\log|G|)$ generators.

It is natural to ask, from a computational perspective, whether using such a short representation of such a large collection is practical. More to the point, is it possible to answer basic questions about a group G that is defined by a list of generators? For example, can membership be tested? can the size be determined? can two groups be tested for equality? can one be shown to include the other? We provide positive answers to each of these questions in the form of polynomial-time algorithms. Classical algorithms to solve these problems have been known to computational group theorists for some time, but without accurate analyses of running times [3,7].

Permutation groups, depending upon how they are represented, either have polynomial-time membership tests, or they don't. Those that do we call polynomial-time recognizable. Any group represented by generators is polynomial-time recognizable. The automorphism group of a graph is also polynomial-time recognizable, even if generators for it are unknown, since testing whether a permutation is an automorphism is easy. As an important corollary we will prove that given generators for a group G, generators for any polynomial-time recognizable subgroup with

small index can be found in polynomial time.

Using these techniques, in [4] L. Babai's probabilistic polynomial-time isomorphism test for graphs of bounded color multiplicity [2] was improved to deterministic polynomial time, C. Hoffmann's probabilistic $O(n^{c\log n})$ isomorphism test for cone graphs was improved to deterministic $O(n^{c\log n})$ time, and a subexponential algorithm for trivalent graph isomorphism was described. In these proceedings E. Luks [6] shows that the last two problems are decidable in deterministic polynomial time. Many other problems of permutation groups seem to have polynomial-time solutions. We give two examples, an algorithm for computing the normal closure of a subgroup, and an algorithm to test a group for solvability. In addition, we describe an approach to the problem of computing group intersections.

Preliminaries

A permutation group on $\{1,\ldots,n\}$ is a collection of 1-1 maps from $\{1,\ldots,n\}$ onto itself that forms a group under composition. Let G be a group. The order of G, $|G|$, is the number of permutations in G. Let H be a subgroup of G, written $G \supset H$. The quotient symbol G/H stands for the collection of cosets of H in G, i.e., the collection of equivalence classes of elements of G in which $x \equiv y$ if and only if $x^{-1}y$ is in H. From Lagrange's theorem we know that every coset of G/H has the same size and, therefore, $|G| = |G/H|\,|H|$. The size of G/H is called the index of H in G.

If $g_1,\ldots,g_k$ are permutations, then the group $\langle g_1,\ldots,g_k\rangle$, generated by the g_i, is the group of all permutations formed by products of

the g_i. We use the symbol I to mean the unique group generated by the identity permutation.

Testing Membership and Determining the Order of a Group

Let G be a group of permutations on $\{1,\ldots,n\}$ generated by $g_1,\ldots,g_k$. There is a descending chain of subgroups, $G = G_0 \supset G_1 \supset \cdots \supset G_n = I$, from G to the identity group, in which G_i is the subgroup of G fixing $1,\ldots,i$. Consider the quotients G_i/G_{i+1}, for $i=0,\ldots,n-1$. The group G can be expressed as

$$G = G_0 = (G_0/G_1)G_1,$$
$$\text{or, } G = (G_0/G_1)(G_1/G_2)\ldots(G_{n-1}/G_n).$$

Therefore, any element g in G can be written in the form $g = a_0a_1\cdots a_{n-1}$, where a_i is an element of G_i/G_{i+1}. Intuitively this says that any permutation in G can be realized as a permutation that moves 1 to the correct place, followed by a permutation that fixes 1 and moves 2 to the correct place, followed by a permutation that fixes 1 and 2 and moves 3 to the correct place, etc. The collection of elements in G_i/G_{i+1}, $i=0,\ldots,n-1$, are called strong generators for G. Our first theorem yields an algorithm, like the one proposed by Sims [7], that computes coset representatives for each of the quotients in polynomial time.

Theorem 1: Let G be a group generated by $g_1,\ldots,g_k$. Let G_i be the subgroup of G fixing letters $1,\ldots,i$. Coset representatives for G_i/G_{i+1}, $i=0,\ldots,n-1$ can be determined in polynomial time.

Proof: The maximum value that $|G_i/G_{i+1}|$ can have is n since cosets only

differ by where they map the letter i+1. We will construct a table T with n rows, labelled 0 to n-1, and n columns, labelled 1 to n, whose ith row is a set of right coset representatives for G_i/G_{i+1}. The table will be organized in such a way that the permutation in the i,jth position fixes letters 1,...,i-1 and maps letter i to position j. Thus the entries in T will only lie on or above the diagonal.

When we are finished, the table should have the property that g is in G if and only if g can be expressed as $a_0a_1\cdots a_{r-1}$, where a_i is a member of the ith row. This we will call the canonical representation of g.

To start, initialize T with the diagonal elements equal to the identity and all others empty. The procedure sift(x), defined below, modifies the table by inserting at most one new coset representative in such a way that x can be written in canonical form.

```
sift(x):
    i ← 0
    while (i ≠ n-1 and
        there is a y in row i such that
        y and x map i+1 to the same letter)
    do
        i ← i+1
        x ← y⁻¹x
    if x is not a member if row i
    then insert x in row i
```

As an example of how sift works, suppose the table for the tower $G_0 \supset G_1 \supset G_2 \supset G_3 = I$ at some point looks like Figure 1.

$(G_0=)$ |__1__| G_1 + |__a__| G_1 + |_____| $G_1^{\sim}$

$(G_1=)$ |__1__| G_2 + |_____| G_2 + |_____| $G_2^{\sim}$

$(G_2=)$ |__1__| G_3 + |_____| G_3 + |_____| $G_3^{\sim}$

Figure 1

Consider the call sift(b) for some b in G_0 but not in G_1. If $a^{-1}b$ is in G_1, and $a^{-1}b$ is not in G_2, then after the call sift(b) the table would look like Figure 2. In this table b is expressible in canonical form as $a(a^{-1}b)1$.

1	a	
1	$a^{-1}b$	
1		

Figure 2

At this point we make a key observation: all of the coset representatives have been found if and only if

(1) each generator can be written in canonical form, and

(2) each product of a pair of representatives in the table can be written in canonical form.

Since we will only sift elements from G we need only verify that when (1) and (2) are satisfied any g in G can be written in the canonical form. Let g be an element of G. Write g as a product of generators and write each generator in canonical form. If this product is not in canonical form, use (2) to rewrite it.

By using (2) we can take an adjacent pair of representatives x,y in the string representing g and, if x comes from a higher numbered row than y, rewrite xy in canonical form. This has the effect of moving an

element from a lower numbered row past an element of a higher numbered row to the left in the string representing g. Moving all the row 0 representatives to the left, then all the row 1 representatives, and so on, we can put the string in canonical form. It is important to note here that writing xy in canonical form does not require the introduction of any elements from lower numbered rows than the one y comes from.

The whole algorithm can be described as

Step 1. Sift all the generators.

Step 2. Close the table such that the product xy, for every pair (x,y) in the table, can be written in canonical form.

To perform Step 2 simply run through all pairs (x,y) from the table and sift their product. The number of coset representatives in the table is at most n^2. The number of calls to sift is at most $(n^2)(n^2)$ and each call to sift takes roughly n^2 time. Therefore, the running time is $O(n^6)$, a polynomial in n.

□

Once the coset representatives have been found, testing membership and computing the size of G is not hard. To determine if x is an element of G, run sift with argument x. If x can be written in canonical form without the introduction of new elements into the table, then x has can be written as a product of generators. If x cannot be written in canonical form, then x is not in G. The order of G is the product of the sizes of G_i/G_{i+1}.

The group $G = \langle g_1,\ldots,g_k \rangle$ contains $H = \langle h_1,\ldots,h_s \rangle$ if and only if

each h_i is a member of G. Two groups are equal if and only if each contains the other. Therefore, the polynomial-time membership test gives polynomial-time inclusion and equality tests for permutation groups.

The following important corollary to Theorem 1 is used by E. Luks [6] in his polynomial-time, bounded-valence graph isomorphism test.

Corollary 1: Let G be a permutation group on $\{1,\ldots,n\}$ generated by $g_1,\ldots,g_k$. Let H be a polynomial-time recognizable subgroup of G, whose index in G is at most a polynomial in n. Generators for H can be found in polynomial time.

Proof: The sequence of groups $G \supset H \supset H_1 \supset \cdots \supset H_n = I$, where H_i is the subgroup of H that fixes the letters $1,\ldots,i$, can be used just as the sequence $G \supset G_1 \supset \cdots \supset G_n$ was used in Theorem 1, with two changes. The first is that row 0 may have a polynomial number of entries instead of at most n. The second is that to test whether two elements x and y of G are in different cosets of G/H, the permutation $x^{-1}y$ has to be tested for membership in H.

□

Straight-Line Programs

By a straight line program we mean a sequence of instructions $h_1,\ldots,h_m$ in which

$$h_i = \begin{cases} h_j h_k \text{ with } j,k < i, \text{ or} \\ g_j, \text{ where } g_j \text{ is a generator.} \end{cases}$$

Not every permutation x in G can necessarily be expressed by a

polynomial-length word in the g_i, however, our next theorem states that every x can be expressed as the result of a short straight-line program.

Theorem 2: Let $G = \langle g_1,\ldots,g_k \rangle$ be a group of permutations on $\{1,\ldots,n\}$. There is a polynomial p such that each permutation x in G can be computed by a straight-line program $h_1,\ldots,h_m = x$, where $m \leq p(n)$.

Proof: Using the polynomial-time procedure of Theorem 1, compute a table of strong generators for G. Before the algorithm begins, form a directed acyclic graph with k leaves, one for each of the generators, and no edges. As the table is built, new products are formed. Each time a product is formed, add a new node to the dag in such a way that the sons of this node correspond to the factors of the product.

When the procedure terminates, a polynomial-size dag will have been formed and every permutation represented by a node in the graph will have an obvious polynomial-length straight-line program to compute it. Since each permutation in G is the product of exactly n strong generators, each can be computed by a polynomial-length straight-line program. □

Group-Theoretic Applications

Let H be a subgroup of a permutation group G on $\{1,\ldots,n\}$. H is normal in G if $g^{-1}Hg = H$ for any g in G. The normal closure of H in G is the smallest normal subgroup, K, of G that contains H. To illustrate the usefulness of the sift and close operations we present a polynomial-time algorithm to compute generators for the normal closure of H in G.

Theorem 3: Let $H = \langle h_1,\ldots,h_r \rangle$ be a subgroup of $G = \langle g_1,\ldots,g_k \rangle$. Generators for K, the normal closure of H in G, can be computed in polynomial time.

Proof: Form a table T of strong generators for H using the algorithm of Theorem 1. In order to get generators for K we will modify T until it is a table of strong generators for K.

Since H is a subset of K, and K is normal in G, each product of the form $g_i^{-1}hg_i$, where g_i is a generator of G and h is a coset representative from T, should be in K. In order to achieve this we will take every such product and sift it into T. The following program takes T and augments it using the sift and close steps until T has the property that for all h in T, and for all generators g_i of G, the product $g_i^{-1}hg_i$ is expressible in canonical form using representatives of T.

```
while there is an x in T
      not processed by this loop
do
      for each g_i: sift(g_i^-1 x g_i)
      close T
```

The main loop is executed at most a polynomial number of times. Since sifting and closing are polynomial-time operations, the whole program runs in polynomial time.

Let Group(T) stand for the group generated by the permutations in the table T. Certainly Group(T) contains H. A simple induction proves that the T produced by the above algorithm contains only permutations that are generated from products of the form x or $g_i^{-1}xg_i$ where x is an element of the normal closure of H in G. Therefore, if Group(T) is normal in G, then it is the smallest normal subgroup of G that contains H.

To see that Group(T) is normal in G, observe that for each generator g_i of G, and each generator x of Group(T), the product $g_i^{-1}xg_i$ is in Group(T). Let $x_1,\ldots,x_m$ be some of the generators of Group(T). For each generator g_i of G, define x_j' to be $g_i^{-1}x_jg_i$, an element of Group(T). If $y = x_1x_2\cdots x_m$, then

$$\begin{aligned} g_i^{-1}yg_i &= g_i^{-1}x_1\cdots x_mg_i \\ &= x_1'g_i^{-1}x_2\cdots x_mg_i \\ &= x_1'\cdots x_m'g_i^{-1}g_i \\ &= x_1'\cdots x_m', \text{ an element of Group(T).} \end{aligned}$$

It is not hard to see from this that if g is any element of G and y is any element of Group(T), then $g^{-1}yg$ is an element of Group(T). Therefore, Group(T) is the normal closure of H in G.

□

The <u>derived subgroup</u> G' of a group G is defined to be the group generated by all products of the form $a^{-1}b^{-1}ab$, where a,b are elements of G. The group G is called <u>solvable</u> if the sequence $G \supset G' \supset\cdots\supset G^{(i)}\supset\cdots$ terminates at I. Solvable groups play an important role in the study of field extensions and ultimately relate to the conditions under which a polynomial equation has a solution in radicals. Using the normal closure algorithm we can get a polynomial-time test for solvability.

<u>Theorem 4</u>: Let $G = \langle g_1,\ldots,g_k\rangle$ be a group of permutations on $\{1,\ldots,n\}$. In polynomial time G can be tested for solvability.

<u>Proof</u>: It is a fact, which we don't prove here, that the derived subgroup of G is equal to the normal closure in G of the subgroup generated

by all products of the form $g_i^{-1}g_j^{-1}g_ig_j$. By forming all such products, and computing the normal closure, the derived subgroup of G can be computed in polynomial time.

If the sequence $G \supset G' \supset \cdots$ converges to I, then it does so within a polynomial number of steps, since $|G^{(i)}/G^{(i+1)}| \geq 2$. If the sequence doesn't converge to I, then for some polynomially bounded i, $G^{(i)} = G^{(i+1)}$. The derived groups can be computed, and group equality can be tested in polynomial time. Therefore, it can be determined, in polynomial time, whether the sequence converges to I. Hence, G can be tested for solvability in polynomial time.

□

The Intersection Problem

In [4], [5], and [6] a relationship has been established between certain graph isomorphism problems and the problem of computing generators for the intersection of two groups. Given an arbitrary pair of groups, $G=\langle g_1,\ldots,g_k\rangle$, and $H=\langle h_1,\ldots,h_r\rangle$ we do not know whether it is possible to compute generators for their intersection quickly. There are, however, certain situations in which the intersection can be found in polynomial time.

Polynomially Accessible Towers

The theorem proved in section 1 that computes coset representatives for the quotient groups G_i/G_{i+1} relies on four properties of the tower $G_0 \supset \cdots \supset G_r = I$ for its polynomial running time. They are

(i)the number of groups is polynomial in n,

(ii)generators for G_0 are known,

(iii)the size of G_i/G_{i+1} is bounded by a polynomial in n, and

(iv)there is a polynomial-time test to determine if a and b from G_i are from the same coset of G_i/G_{i+1}.

Any tower that satisfies these four conditions we call polynomially accessible. Using this definition we can restate the first theorem.

Theorem 5: Let $G_0 \supset \cdots \supset G_r = I$ be a polynomially accessible tower of groups. Coset representatives for the quotients G_i/G_{i+1}, for $i=0,\ldots,r-1$, can be determined in polynomial time.

This allows us to prove the main theorem of this section.

Theorem 6: Let G and H be any two polynomial-time recognizable groups. Let S be a group for which generators are known. If S contains both G and H, and there are two polynomially accessible towers, one from S through G to I, and the other from S through H to I, then generators for $G \cap H$ can be found in polynomial time.

Proof: Let $S = H_0 \supset \cdots \supset H_r = H \supset H_{r+1} \supset \cdots \supset H_{s-1} = I$, and $S = G_0 \supset \cdots \supset G_p = G \supset G_{p+1} \supset \cdots \supset G_{q-1} = I$ be the two towers. Construct an $s \times q$ table whose i,jth entry is the group $G_i \cap H_j$. Each entry is a recognizable group since it is the intersection of recognizable groups.

Both $|H_j/H_{j+1}|$ and $|G_i/G_{i+1}|$ are bounded by some polynomial $p(n)$. Consider the two groups $G_i \cap H_j$ and $G_i \cap H_{j+1}$. Let a and b be distinct

coset representatives of $X = (G_i \cap H_j)/(G_i \cap H_{j+1})$. The elements a and b are both from the group H_j. Furthermore, if $a^{-1}b$ were an element of H_{j+1}, it would also be an element of $G_i \cap H_{j+1}$. Since a and b are from different cosets of X it follows that $a^{-1}b$ is not in H_{j+1}. Therefore, a and b are distinct coset representatives of H_j/H_{j+1}. Hence $|(G_i \cap H_j)/(G_i \cap H_{j+1})|$ is less than or equal to $|H_j/H_{j+1}| \leq p(n)$. Similarly, $|(G_i \cap H_j)/(G_{i+1} \cap H_j)| \leq p(n)$.

Let P be any path in the table beginning at S, moving only one row down or one column across at a time, passing through $G \cap H$, and ending at I. This path P describes a polynomially accessible tower of groups from S through $G \cap H$ to I.

For example, the tower $S \supset H_1 \supset H_2 \supset \cdots \supset H \supset G_1 \cap H \supset \cdots \supset G_p \cap H \supset G \cap H \supset G_{p+1} \cap H \supset \cdots \supset I$ can be used to get generators for $G \cap H$ in polynomial time.

□

A natural conclusion that we can draw from this theorem is that many group intersection problems lie in $NP \cap coNP$. For example, this will be the case whenever we know that G and H

(1) lie under a common group S, and

(2) there exists a chain of groups between S and G, and a chain between S and H whose indices are polynomially bounded.

The reason computing $G \cap H$, for such G and H, is in $NP \cap coNP$ is that one can use nondeterminism to guess generators for S and for each of the groups in the chains. Towers from G to I, and from H to I can be

obtained by fixing letters, and then the algorithm in Theorem 5 can be used to verify the guesses.

References

[1] L. Babai, Isomorphism Testing and Symmetry of Graphs, unpublished manuscript.

[2] L. Babai, Monte-Carlo Algorithms in Graph Isomorphism Testing, Submitted to Siam J. of Computing, (1979).

[3] Curran, M.P.J., Topics on Group Theory and Computation, Academic Press (1977).

[4] M. Furst, J. Hopcroft, E. Luks, A subexponential Algorithm for Trivalent Graph Isomorphism, Proc. Eleventh Southeastern Conf. on Graph Theory and Computing, to appear.

[5] C. Hoffmann, Testing Isomorphism of Cone Graphs, Proc. Twelfth Annual ACM Symposium on the Theory of Computing (1980).

[6] E. Luks, Isomorphism of Graphs of Bounded Valence Can Be Tested in Polynomial Time, Proc. 21st FOCS (1980).

[7] C. Sims, Computational Problems in Abstract Algebra, Ed. John Leech, Pergamon Press (1970), pp.176-177.

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)
Cornell University
Department of Computer Science
Ithaca, New York 14853

2a. REPORT SECURITY CLASSIFICATION

2b. GROUP

3. REPORT TITLE

POLYNOMIAL-TIME ALGORITHMS FOR PERMUTATION GROUPS

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Technical Report #TR 80-442

5. AUTHOR(S) (First name, middle initial, last name)
Merrick Furst
John Hopcroft
Eugene Luks

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO OF REFS
October 1980	15	7

8a. CONTRACT OR GRANT NO	9a. ORIGINATOR'S REPORT NUMBER(S)
N00014-76-C-0018	TR 80-442
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. DISTRIBUTION STATEMENT
Distribution of manuscript is unlimited

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Office of Naval Research

13. ABSTRACT

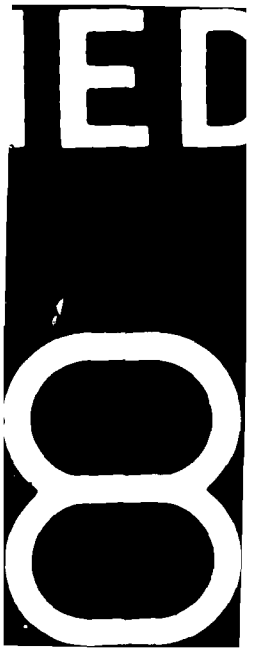
A permutation group on n letters may always be represented by a small set of generaters, even though its size may be exponential in n. We show that it is practical to use such a representation since many problems such as membership testing, equality testing, and inclusion testing are decidable in polynomial time. In addition, we demonstrate that the normal closure of a subgroup can be computed in polynomial time, and that this procedure can be used to test a group for solvability. We also describe an approach to computing the interesection of two groups. The procedures and techniques have wide applicability and have recently been used to improve many graph isomorphism algorithms.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6801

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
algorithms						
complexity						
permutation groups						
graph isomorphism						


ED
8