Great Theoretical Ideas

Assignment 9 Due: April 6, 1999

The program you will write for this assignment will be able to answer questions like these.

Question 1 How many four-stone necklaces can we build using rubies and diamonds so that no two necklaces are symmetric? (Two necklaces are symmetric if we can reach one through a sequence of rotations and flips of the other.)

Although there are 16 ways to choose stones for each of 4 locations, many of them are equivalent; there are only 6 distinct necklaces. (See Figure 1.)

Question 2 Painting each face of a cube red or green, how many cubes can we paint so that no two cubes are three-dimensional rotations of each other? (Think about this one for a while; the answer is 10.)

Question 3 What if we allow 3 colors? (There are 57 colorings.) If we additionally say two are symmetric if they are mirror images (that is, if swapping one opposite pair of faces yields another), there are 56 distinct colorings.

Question 4 Consider a $2 \times 2 \times 2$ Rubik's Cube. We can twist a face of the cube, or we can rotate the entire cube. Choosing one of 6 colors for each exterior face of a small cube, how many colorings are there so that for any two colorings, we cannot reach one from another through a series of twists and three-dimensional rotations? (This problem is quite large; not even our best program has solved it.)

To generalize these problems and others (including many chemistry problems, like counting isomers of a molecule), we need to formalize the notions of colorings and symmetries. Let n be the number of locations (faces, beads, whatever) being colored and let k be the number of colors available. We can represent a coloring as a function $t : \{1, \ldots, n\} \to \{1, \ldots, k\}$ assigning colors to locations.

To formalize *symmetries*, we use permutations. For example, we want to say that the necklace $t = \binom{1\ 2\ 3\ 4}{2\ 1\ 2\ 1}$ (upper-right necklace of Figure 1) is symmetric to the necklace $t' = \binom{1\ 2\ 3\ 4}{1\ 2\ 1\ 2}$ (lower-right) because we can rotate t once counterclockwise to get t'. That is, t and t' are symmetric in this example because if we permute the colors of t so that location 1 gets the color t assigns to 2, location 2 gets the color t assigns to 3, location 3 gets the color t assigns to 4, and location 4 gets the color t assigns to 1, then we get the color assignments of t'. Thus we can represent this permutation of colors with the permutation $\Pi = \binom{1\ 2\ 3\ 4}{2\ 3\ 4\ 1}$.

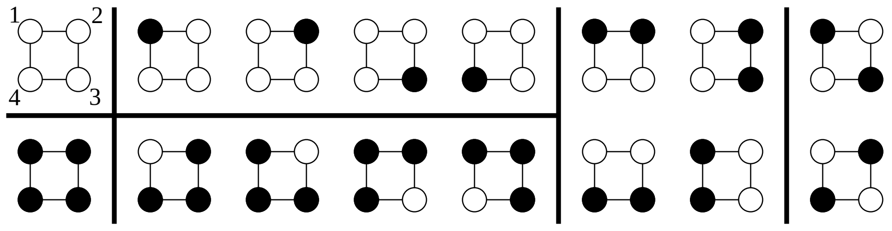

Figure 1: Ways to build a necklace.

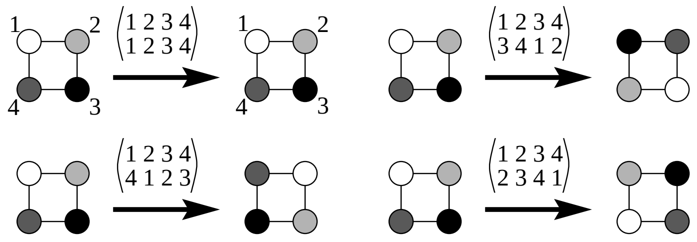
1
2
4
3
$\begin{pmatrix}1&2&3&4\\1&2&3&4\end{pmatrix}$
$\begin{pmatrix}1&2&3&4\\3&4&1&2\end{pmatrix}$
$\begin{pmatrix}1&2&3&4\\4&1&2&3\end{pmatrix}$
$\begin{pmatrix}1&2&3&4\\2&3&4&1\end{pmatrix}$

Figure 2: Rotations as permutations.

Regarding Π as a function from $\{1,\ldots,n\}$ onto $\{1,\ldots,n\}$, so that $\Pi(1)=2$, $\Pi(2)=3$, $\Pi(3)=4$, and $\Pi(4)=1$, we can succinctly express that t is symmetric to t' because there is a symmetry permutation Π so that $t\circ\Pi=t'$. (Here $\circ$ represents function composition.) It's important to think of the permutations as permuting the *colors* of locations, not the locations themselves.

Thus, to abstractly talk of symmetries, we can talk about a set of permutations. As the previous paragraph discusses, $\binom{1\ 2\ 3\ 4}{2\ 3\ 4\ 1}$ is one symmetry permutation for the necklace problem. Figure 2 illustrates all rotations of a necklace and their corresponding permutations. If we want to say that all rotations are symmetric, then, we use the set of permutations

$$G=\left\{\begin{pmatrix}1&2&3&4\\1&2&3&4\end{pmatrix},\begin{pmatrix}1&2&3&4\\2&3&4&1\end{pmatrix},\begin{pmatrix}1&2&3&4\\3&4&1&2\end{pmatrix},\begin{pmatrix}1&2&3&4\\4&1&2&3\end{pmatrix}\right\}.$$

These permutations represent the ways to rotate a necklace. If we also want to include flipping the necklace, we add four more permutations to G.

$$\begin{pmatrix}1&2&3&4\\4&3&2&1\end{pmatrix},\begin{pmatrix}1&2&3&4\\2&1&4&3\end{pmatrix},\begin{pmatrix}1&2&3&4\\1&4&3&2\end{pmatrix},\begin{pmatrix}1&2&3&4\\3&2&1&4\end{pmatrix}$$

(With or without flips, the answer is 6.) Notice that, for any "natural" notion of symmetry (including all the questions asked at the beginning of this handout), the corresponding set of permutations will be a group under composition. Also notice that, for any permutations Δ and Π and for any coloring t, $t\circ(\Delta\circ\Pi)$ is $(t\circ\Delta)\circ\Pi$.

Given a group of permutations G, two colorings t and t' are **symmetric** (written $t\sim t'$) if we can reach t' from t using some permutation of G — that is, if there is some permutation $\Delta\in G$ so that $t\circ\Delta=t'$. Convince yourself that $\sim$ is an equivalence relation on colorings.

The general question, then, is the following.

Question A Given a collection of n elements and k colors, and given a notion of symmetry represented by a permutation group G, how many ways can we color the elements with k colors so that no two ways are symmetric according to G?

We could ask Question A even more succinctly: For n, k, and G, how many equivalence classes does $\sim$ have? Notice that we get different counting problems for different symmetry groups G.

Your program will work with groups of permutations. These groups can be quite large; a user shouldn't have to type all the permutations of a group. A simpler way to describe

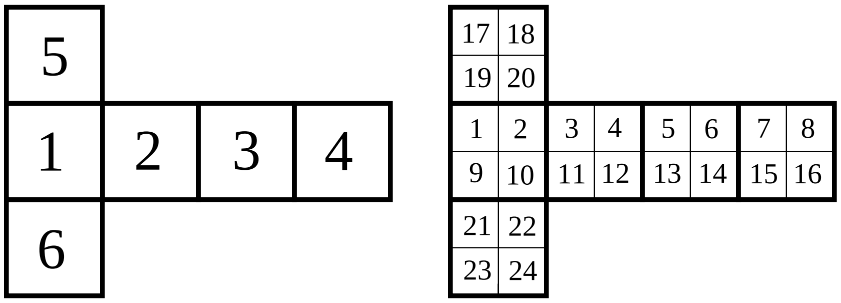

Figure 3: Numbering faces (Problems 1 and 2).

a group is to indicate a small set A of permutations and then to find all permutations we can reach through repeated applications of permutations from A. You can convince yourself that this is a subgroup; we call it the **subgroup generated by** A. (The items from A are called **generators**.) For example, if $A = \left\{\binom{1\ 2\ 3\ 4}{2\ 3\ 4\ 1}, \binom{1\ 2\ 3\ 4}{4\ 3\ 2\ 1}\right\}$, then the subgroup generated by A is

$$\left\{\begin{pmatrix}1&2&3&4\\1&2&3&4\end{pmatrix},\begin{pmatrix}1&2&3&4\\2&3&4&1\end{pmatrix},\begin{pmatrix}1&2&3&4\\3&4&1&2\end{pmatrix},\begin{pmatrix}1&2&3&4\\4&1&2&3\end{pmatrix}\right.$$
$$\left.\begin{pmatrix}1&2&3&4\\4&3&2&1\end{pmatrix},\begin{pmatrix}1&2&3&4\\2&1&4&3\end{pmatrix},\begin{pmatrix}1&2&3&4\\1&4&3&2\end{pmatrix},\begin{pmatrix}1&2&3&4\\3&2&1&4\end{pmatrix}\right\}.$$

Problems

This assignment has 5 problems. The first 2 are written, the last 3 are programming. All will be automatically graded. The following section ("About your program") describes administrative points about the assignment.

Problem 1 (5 points) What input would you use to ask Question 2 at the beginning of this handout? Number the faces as in Figure 3 and place your answer in the file `cube.in` in your handin directory using the format described in "About your program" below. A full-credit answer will use as few generators as possible. (The `check` script will not check your answer.)

Problem 2 (10 points) What input would you use for Question 4? Number the faces as in Figure 3 and place your answer in the file `rubik.in` in your handin directory using the format described in "About your program" below. A full-credit answer will use as few generators as possible. (The `check` script will not check your answer.)

Problem 3 (20 points) The first type of request your program should handle is for the subgroup generated by a set of permutations. To do this, you should complete the following function in `Enum.cc`.

```
PermList* problem3(int n, PermList *gen);
```

This function takes n, the number of elements being colored, and gen, a list of generators. The function should compute and return all permutations in the subgroup generated by the generators listed in gen.

The grading program always represents a permutation as an array of integers. For example, the permutation $\binom{1\ 2\ 3\ 4}{2\ 3\ 4\ 1}$ is represented by the grading program as the array `{1,2,3,0}`. (Subtracting 1 from each element is convenient for C.)

The grading on this question will be all-or-nothing. For credit, you should consistently generate the correct answer in time at most 10 times benchmark.

Hint: Our implementation takes $O(jmn)$ time, where j is the number of generators and m is the number of permutations in the generated subgroup.

Problem 4 (55 points) The second request type your program will encounter is Question A.

```
int problem4(int n, int k, PermList *gen);
```

Besides n and *gen* as with `problem3()`, this function takes k, the number of colors available for use. The function should return the number of ways we can color the elements so that no two are symmetric according to the subgroup generated by *gen*.

The grading on this problem is similar to the grading for Assignment 3. We will give your program a total of 5 minutes to solve Problem-4 requests. Your score will be

$$\max\left\{0, \min\left\{55, 45 - \frac{3}{2}\log_2 \frac{you}{benchmark}\right\} - 10(wrong)\right\},$$

where *benchmark* is the amount of time used to get through the last question your program attempted in the 5 minutes. (Any of the first 6 questions that your program fails to answer is counted as wrong.)

Although the grading scale is similar to that of Assignment 3, the benchmark is not. To get full credit for this problem you will need to be more than 100 times faster than benchmark! The benchmark uses an exhaustive-search technique; our current best solution, which uses a different technique, is 200 times faster than benchmark. The appendix to this assignment includes proofs to two lemmas that you may find useful in developing the fastest possible solution.

Problem 5 (10 points) The final type of request your program will encounter restricts the set of colorings to colorings with a_1 items colored the first color, a_2 items colored the second color, $\ldots$, a_k items colored the kth color. For example, if we ask how many necklaces of two rubies and two diamonds there are, the answer is 2 (Figure 1).

```
int problem5(int n, int k, PermList *gen, int *num_color);
```

As in Problem 4, we have n, k, and *gen*. The *num_color* array has k elements and specifies how many of each color is available; the total number of colors available will be exactly n. The function should return the number of ways to color n items consistently with *num_color* so that no two ways are symmetric according to the subgroup generated by *gen*.

For 7 points, your program should consistently compute the correct answer in time at most 10 times benchmark. For full credit, the program should consistently compute the correct answer in time at most 0.2 times benchmark. The benchmark is quite fast; you are unlikely to receive credit unless you have a full-credit solution to Problem 4.

About your program

To give the necklace problem to your program, we specify n, k, and G as follows.

```
4 4 2
1
2 3 4 1
0
```

Respectively, this tells your program to answer Problem 4 of this assignment with $n = 4$, $k = 2$, and G as the subgroup generated by $\{\binom{1\ 2\ 3\ 4}{2\ 3\ 4\ 1}\}$. The line `1` indicates that we use only one generator. The final `0` is a request to solve Problem 0, indicating that your program should terminate. The input may include many requests; the program will continue answering them until reaching a Problem-0 request.

If we wanted to allow flips in addition to rotations, we could add the permutation $\binom{1\ 2\ 3\ 4}{4\ 3\ 2\ 1}$ to the list of generators.

```
4 4 2
2
2 3 4 1
4 3 2 1
0
```

In your handin directory you will find several files. Your answers should appear only in the files `cube.in`, `rubik.in`, and `Enum.cc`; anything appearing in other files will be ignored. The other files are provided to help you concentrate on the primary issues; you can ignore or use these as you want, but your program should be compatible with them.

`cube.in`	your answer to Problem 1 should appear here
`rubik.in`	your answer to Problem 2 should appear here
`Enum.cc`	all of your code should appear in this single file
`Enum.h`	prototypes for the functions in `Enum.cc`
`Main.cc`	an implementation of `main()` that reads requests from the input file and uses your code in `Enum.cc` to compute answers
`PermList.{cc,h}`	implements a list of permutations with a connected hash table facilitating quick lookup within the list
`workload`	a set of input problems for testing
`Makefile`	compiles the program (see next paragraph)

(Neither `check` nor the grading program uses `Main.cc`; we provide it so that you can develop and test code where the `check` program is not available.)

To compile your program, you can run `gmake enum`, which will use the `main()` function in `Main.cc` to run your program. To run `enum`, direct to it an input file specifying a set of problems to solve (e.g., `./enum < workload`). The `enum` program should work on any machine. If you are on Andrew Sun machines, you may prefer compiling your program using `gmake enumch`. This will use a "checking" version of `main()` that checks your program's answers and reports performance relative to the benchmark. Like `enum`, to run `enumch` you should direct a problem file (e.g., `./enumch < workload`).

Once you have completed your program and copied your files into your handin directory, you should run a final check on your program using the `check` program,

```
/afs/andrew/scs/cs/15251/bin/check
```

This will verify that you have turned in the correct files and will report an estimate of your grade.

At no time should your program's memory usage exceed 2 megabytes; if this occurs, then the program will be considered to have crashed at this point. (Memory will probably not be an issue.) You may reuse data computed for previous requests if you would like.

Appendix

In this appendix we state and prove two lemmas as a hint that you may find useful in developing fast algorithms for Problems 4 and 5.

Lemma 1 *Given n, k, and permutation group G, we have our equivalence relation $\sim$. Consider an equivalence class C, and define for any two colorings $t, t' \in C$ the set $A_{t,t'}$ to be the set of permutations that permute t into t', $\{\Pi \in G : t \circ \Pi = t'\}$. There is a number m so that for all $t, t' \in C$, $A_{t,t'}$ has exactly m permutations.*

Proof. Our first step is to show that, for any colorings t and t' from C, the number of permutations mapping t to itself is the same as the number of permutations mapping t to t'; that is, $|A_{t,t}| = |A_{t,t'}|$. We do this by setting up a bijective correspondence between $A_{t,t}$ and $A_{t,t'}$. Since $t \sim t'$, we can choose some permutation Δ so that $t \circ \Delta = t'$. Our correspondence maps $\Pi \in A_{t,t}$ to $\Pi \circ \Delta$. (Notice that $\Pi \circ \Delta \in A_{t,t'}$ since G is closed under composition and since $t \circ (\Pi \circ \Delta) = (t \circ \Pi) \circ \Delta = t \circ \Delta = t'$.) This correspondence is one-to-one because if any two permutations $\Pi, \Pi' \in A_{t,t}$ are mapped to the same place ($\Pi \circ \Delta = \Pi' \circ \Delta$) then they are the same ($\Pi = \Pi'$), since G (as a group) has cancellation. This correspondence is onto because for any $\Pi' \in A_{t,t'}$ we can find a permutation $\Pi \in A_{t,t}$ so that $\Pi \circ \Delta = \Pi'$. Namely, we can solve for Π by multiplying both sides by Δ^{-1} to get $\Pi = \Pi' \circ \Delta^{-1}$. This $\Pi' \circ \Delta^{-1}$ is in $A_{t,t}$ because $t \circ (\Pi' \circ \Delta^{-1}) = (t \circ \Pi') \circ \Delta^{-1} = t' \circ \Delta^{-1} = t$.

For every pair of colorings $t, t' \in C$, $A_{t,t'}$ has exactly $m = |G|/|C|$ permutations, satisfying the theorem. This follows from two observations. First, every permutation in G maps t to some coloring in C. Second, for any $t', t'' \in C$, we have $|A_{t,t'}| = |A_{t,t''}|$ (since the previous paragraph shows that both sides are equal to $|A_{t,t}|$). ■

The second lemma involves **cycle representations**, a useful way to represent permutations. A **cyclic permutation** is a permutation that rotates a subset of the range $\{1, \ldots, n\}$ and leaves the remainder of the range unchanged. Formally, a cyclic permutation Π has some sequence $(a_0\ a_1\ \cdots\ a_{n-1})$ so that

$$\Pi(x) = \begin{cases} a_{(i+1) \bmod n} & \text{if } x = a_i \text{ for some } i \in \{0, \ldots, n-1\} \\ x & \text{if } x \neq a_i \text{ for all } i \in \{0, \ldots, n-1\} \end{cases}$$

We abbreviate such a permutation as $(a_0\ a_1\ \cdots\ a_{n-1})$. For example, $(1\ 3\ 2)$ abbreviates the permutation $\Pi = \binom{1\ 2\ 3\ 4}{3\ 1\ 2\ 4}$.

x	$\Pi(x)$	reason
1	3	3 follows 1 in the sequence (1 3 2)
2	1	1 follows 2 in the sequence (wrap around)
3	2	2 follows 3 in the sequence
4	4	4 does not appear in the abbreviation, so it is unchanged

The cycle representation of a general permutation is a product (composition) of disjoint cyclic permutations' abbreviations.

example permutation	example as product of disjoint cyclic permutations	example's cycle rep.
$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{pmatrix}$	(2 3 4 1)
$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 1 & 4 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 3 & 2 \end{pmatrix}$	(3 1)(2 4)
$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 3 & 4 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 3 & 4 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix}$	(2 1)(3)(4)

Notice that if a permutation leaves an element fixed, then the cycle representation includes this element as its own cycle, as in the third example of the table.

Lemma 2 *Given $\Pi \in G$, let c be the number of disjoint cycles in Π's cycle representation. There are k^c colorings that Π leaves t fixed. That is, the number of colorings $t : \{1, \ldots, n\} \to \{1, \ldots, k\}$ so that $t \circ \Pi = t$ is k^c.*

Proof. Note that for a coloring t, $t \circ \Pi$ shifts each item's color once along the cycle in which the item appears. If t does not color a cycle monochromatically, then $t \circ \Pi$ will not be the same coloring as t, since the cycle's colors shifted once within the cycle will not be the same. On the other hand, if t colors every cycle monochromatically, then $t \circ \Pi$ will be the same coloring, since shifting the colors of a monochromatic cycle does not alter its colors. Thus for each of the c cycles we can choose one of k colors. ∎