RICH INTERNSHIP REPORT

NICOLAS NAING

1. Introduction

My research with Professor Chinta this summer focused on the topic of turning problems in mathematics into simple games or puzzles. There is a long history of using mathematics to construct, solve and study puzzles and games, both for the purposes of education and recreation. For example, the famous Rubik's cube was invented by an architecture professor as a project for his students. Both this and the classic "15 puzzle" can be used to illustrate ideas in group theory and abstract algebra. More recently, Sudoku and Ken-Ken are examples of mainstream games which have attracted the attention of mathematicians.

My work this summer consisted of constructing programs in Java to implement to classes of games. First, I wrote a program which took geometric realizations of groups of small order and turned it into a game in which the player is asked to write an arbitrary element in the group in terms of a fixed set of generators. The second program takes a blog post of John Baez [1] as inspiration. In the post, Baez explains earlier papers [2, 3] and shows how the polynomial T is equivalent to T^7 in the semiring of polynomials $\mathbb{N}[T]$ with nonnegative integer coefficients, modulo the relation $T = 1 + T^2$.

2. A puzzle arising from bijections between trees

We describe a result from Blass [2] This result is somewhat strange as smaller numbers such as 2, 3, and 5 in place of 7 all do not work, while all other numbers congruent to 1 modulo 6 do. Blass starts off the second section of the paper with a few insights into trees and the theorem above.

The first result is the fact that all trees are either a root or can be split into two separate trees. Letting T denote the set of all trees, this observation leads to the bijection

$$T \to 1 + T^2. \tag{2.1}$$

This can be extended to $T^n \to T^{n-1} + T^{n+1}$ and $T^{n-1} + T^{n+1} \to T^n$. The second result is the fact that when viewed as a polynomial, the equation $T = 1 + T^2$ shares roots with $T^7 = T$, in particular any sixth root of unity. While this is not an actual proof of the first result as T is a set of trees and not a complex number, it offers some reason as to why 7 and other numbers congruent to 1 modulo 6 work in the theorem.

To implement the first result, I made a program that forms a "game" where the user is given nine slots, which represent $1, T, T^2, \ldots, T^8$. These represent the nine different powers of T that were used in a solution given by John Baez [1] to pass from T to T^7 using the relation (2.1). This solutions last move was $T^6 + T^8 \to T^7$, so the slots stop at T^8. The starting position has all slots empty except for the T slot, which has coefficient 1, and the objective of the game is to get from the starting configuration to the goal configuration. There are two types of moves that the user can make: "expand" (e) and "contract" (c). The expand moves take a nonnegative slot, subtracts its coefficient by 1,

and increases the coefficients in the two adjacent slots by 1. Expand moves represent the $T^n \to T^{n-1} + T^{n+1}$ bijection. The contract moves are basically the opposite of expand as they take two nonnegative slots that are exactly two apart, subtract 1 from each, and add 1 to the slot in between. This represents the $T^{n-1} + T^{n+1} \to T^n$ bijection. After the user picks one of (e) or (c), they then pick the "middle" number that the type of move uses. For example, e4 "expands" the T^4 slot to yield $T^3 + T^5$, while c6 takes $T^5 + T^7$ and "contracts" them to T^6. Below are some specific examples of sequences of moves. The first picture is a solution for a random configuration, while the second shows the moves for the theorem in question ($T \to T^7$).

```
[HON-2017489:resurch nnaing$ java Game
Curr: 0 1 0 0 0 0 0 0 0
Goal: 0 2 0 0 1 0 0 0 0
e1
Curr: 1 0 1 0 0 0 0 0 0
e2
Curr: 1 1 0 1 0 0 0 0 0
e3
Curr: 1 1 1 0 1 0 0 0 0
c1
Curr: 0 2 0 0 1 0 0 0 0
e1 c1 e1 e2 e3 c3 c2 e2 e3 c3 e1 c1 e3 c1
14
Curr: 0 1 0 0 0 0 0 0 0
Goal: 1 1 2 3 0 4 1 1 1
e1 e2 e3 c1 e4 c2 e5 c3 c2 e6 e5 c3 e6 c4 e7 c5 c6 c7
Curr: 1 0 1 0 0 0 0 0 0
Curr: 1 1 0 1 0 0 0 0 0
Curr: 1 1 1 0 1 0 0 0 0
Curr: 0 2 0 0 1 0 0 0 0
Curr: 0 2 0 1 0 1 0 0 0
Curr: 0 1 1 0 0 1 0 0 0
Curr: 0 1 1 0 1 0 1 0 0
Curr: 0 1 0 1 0 0 1 0 0
Curr: 0 0 1 0 0 0 1 0 0
Curr: 0 0 1 0 0 1 0 1 0
Curr: 0 0 1 0 1 0 1 1 0
Curr: 0 0 0 1 0 0 1 1 0
Curr: 0 0 0 1 0 1 0 2 0
Curr: 0 0 0 0 1 0 0 2 0
Curr: 0 0 0 0 1 0 1 1 1
Curr: 0 0 0 0 0 1 0 1 1
Curr: 0 0 0 0 0 0 1 0 1
Curr: 0 0 0 0 0 0 0 1 0
```

Some things that I noticed were:

(1) The types of moves are reflections in that the first move is expand while the last is compress, second is expand + 2nd last compress, etc. For example, e1 and c7 are opposites because 1 is the second from the left and 7 is the second from the right.

(2) The "middle" state (#9) has a 2 and a 6, interesting as they are both reflections of each otherabout the number 4 and it is the only nontrivial state that only has 2 slots filled
(3) The first few moves of the given solution can be rearranged so the first five moves go e1 e2 e3 e4 e5. This can occur while preserving symmetry as long as the c moves at the end go c3 c4 c5 c6 c7. The middle state in this new solution has the slots at 1 1 1 1 2 1 1 1 1, which may mean that the state of 1 1 1 1 1 1 1 1 1 (9 1s) is not attainable from 0 1 0 0 0 0 0 0 0.

Aside from the rearrangement in #3 above, I did not find any methodology to the given solution. I tried to find a mathematical invariant for the possible states of the puzzle from the starting state 0 1 0 0 0 0 0 0 0, but I did not make any progress there either. My current scrambling function uses Math.random in Java to generate a random decimal from 0 to 1, and then modify this value to return integers from 0 to 13 by scalar multiplication and flooring. This is an inefficient way to make random goal states, but it works well enough to get goals that are $\sim 15-20$ moves away, which, in my opinion, is a reasonable number of steps for this kind of game.

From the point of view of puzzles, the main attraction of the game above is that it is possible to play with no knowledge of any advanced mathematics. This has the potential for broad appeal. Mathematically, the interest stems from the fact that arithmetic in the semiring $\mathbb{N}[T]/(T = 1 + T^2)$ is more subtle than in the ring $\mathbb{Z}[T]/(1 - T + T^2)$. Indeed in the latter polynomial quotient ring, one may easily go from T to T^7 in 18 steps:

$$\begin{aligned}
B &\cong 1+B^2 \cong 1+B+B^3 \cong 1+B+B^2+B^4 \cong B+B+B^4 \cong B+B+B^3+B^5 \\
&\cong B+B^2+B^5 \cong B+B^2+B^4+B^6 \cong B+B^3+B^6 \cong B^2+B^6 \cong B^2+B^5+B^7 \\
&\cong B^2+B^4+B^6+B^7 \cong B^3+B^6+B^7 \cong B^3+B^5+B^7+B^7 \cong B^4+B^7+B^7 \cong B^4+B^6+B^7+B^8 \\
&\cong B^5+B^7+B^8 \cong B^6+B^8 \cong B^7
\end{aligned}$$

This is an illustration of the Euclidean algorithm, which is lacking in $\mathbb{N}[T]/(T = 1 + T^2)$.

3. Groups of small order

I also made a small group theory program in Java and Java Graphics2D that navigates through the dihedral groups of a square, triangle, and tetrahedron. While this program is very basic since it only has two operations (reflect over a set altitude, shift a group of vertices), these two commands are sufficient to reach all possible permutations of the numbered vertices. Essentially, an instantiation of each java file (DrawTri, Draw, DrawTet, one for each shape) contains variables that each stores the value at a specific vertex of the shape. Then the user types in a command, "ref" (reflection) or "shift", which swaps the necessary values in the instance of each class. Finally, the main java program, Groups.java, takes this updated instance and changes the values at each vertex on the graphics screen. The terminal also contains a copy of the instance's values to make sure the graphic is correct.

4. Programs

Both of these programs can be found at github.com/p2pcmlp/Groupss and instructions for each program can be found in the README.md. I implemented both programs in Java and used the Graphics2D class that can be called with import javax.swing.*. I also used a Keyboard "class" from my AP Computer Science class that takes the user's keyboard input and stores them into String/int to allow the user to make commands in the terminal.

References

1. John Baez, *This week's finds in mathematical physics (week 202)*, `http://math.ucr.edu/home/baez/week202.html`.
2. Andreas Blass, *Seven trees in one*, J. Pure Appl. Algebra **103** (1995), no. 1, 1–21. MR 1354064
3. Marcelo Fiore, *Isomorphisms of generic recursive polynomial types*, Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (New York, NY, USA), POPL '04, ACM, 2004, pp. 77–88.