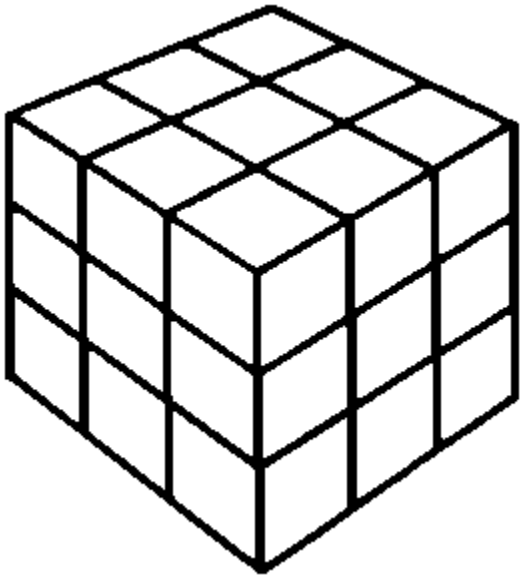
COMPUTER ORIENTED SOLUTIONS TO THE RUBIK'S CUBE

IBM Version 1.0



USER'S GUIDE

Douglas E. Woolley
University of South Florida

September 1988

Introduction

The Rubik's Cube, invented by Erno Rubik, started the historical puzzle craze in the early 1980's. Without help, solving the Rubik's Cube could take several months. For skilled mathematicians it could take several weeks to restore the cube to its original state. The most advanced computer would take 1.4 million years to examine the cube's 43,252,003,274,489,856,000 possible combinations. The solution to the Rubik's Cube, along with many other puzzles, is found by the use of abstract algebra, or group theory.

Being fascinated by computers and having the math skills necessary to take on the task, I embarked upon proving my philosophy: Anything that a person can logically solve or develop an algorithm for, a programmer can instruct a computer to do the same. With this thought in mind, I worked on manually solving the Rubik's Cube with the aid of some elementary group theory and a solution guide. Once an algorithm was developed for solving the puzzle from any given combination, I then coded the solution into a program written in Turbo Pascal.

This program entitled, ***Computer Oriented Solutions to the Rubik's Cube***, allows the user to enter the appearance of the puzzle into the computer. The program will then display a step-by-step solution for the user to restore the puzzle to its original state. After each move of the solution is put on the screen, either a graphical or a textual sketch of the cube is displayed along with the colors on the recently moved cube.

The program named, **RUBECUBE.COM**, can be run on any IBM PC or compatible. It is recommended that a color monitor be used in order to appreciate the color displays. When the computer is in DOS (Disk Operation System), the program can be started by typing RUBECUBE and then pressing the RETURN key (ENTER key). At this point a title page will appear. To continue, press any key, and then the program menu will appear:

1. Graphical Solution

2. Text Solution

3. Instructions

4. Set Valid Colors

5. Test Program

6. Exit

Choose:_

Simply press the number of the option you desire to choose.

Table of Contents

Introduction ii

Table of Contents iii

Input for Graphical and Text Solutions 1

Graphical Solution 3

Text Solution 6

Instructions 8

Set Valid Colors 9

Test Program 10

Research - History 11

Research - Competitions 13

Research - Solution Manuals 14

Research - Group Theory 15

Research - Group Theory Solutions 16

Description of Program Solution Strategy 18

Partial Pascal Solution Listing 19

Name and Description of Procedures in Program 28

Appendix - Variations of the Rubik's Cube 32

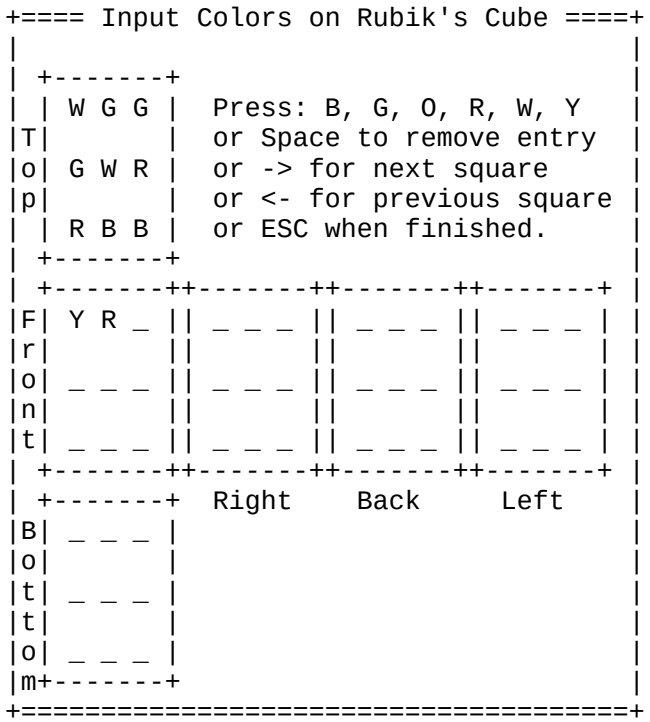
Bibliography 33

Author's Biography 34

Input for Graphical and Text Solutions

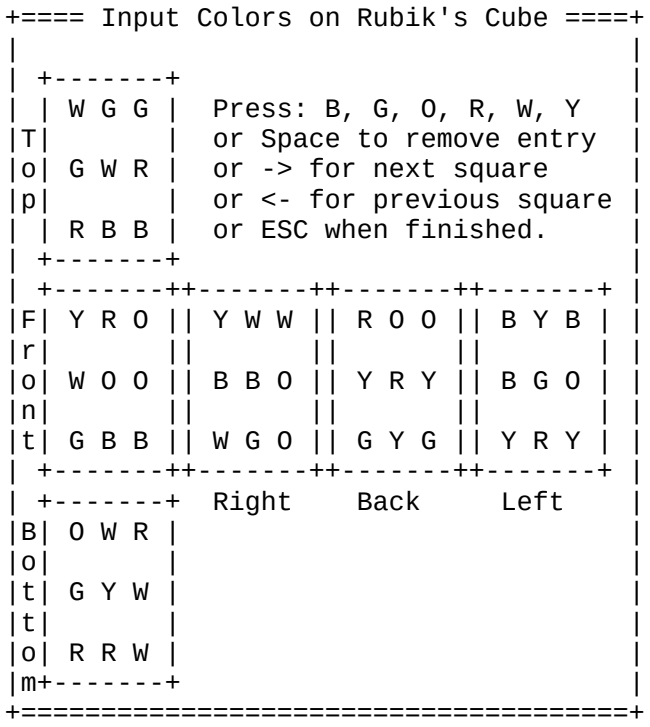
In order to obtain a solution (Graphical or Text), the user must describe the appearance of the puzzle to the computer. This is done by entering the color of the squares on each of the six sides of the cube into the computer. Each side of the Rubik's Cube contains 9 squares--a total of 54 squares. The original Rubik's Cube, produced by Ideal Toy Corporation, along with other similar brands, will have the following six unique colors on its squares: (B)lue, (G)reen, (O)range, (R)ed, (W)hite, and (Y)ellow. If your cube does not have all of these colors, then you first must choose option 4 from the main menu to "set valid colors" for the cube before you can input the cube's colors.

The input routine is the same for the graphic solution option and the text solution option. To begin, place the cube in front of you and designate one side to be the front--it will face you. To input colors on the Rubik's Cube, you will enter the color symbol pertaining to each of the squares, beginning with those on the top, then front, right, back, left, and bottom. For a graphic illustration of these sides, you may want to view the instructions in the program, option 3. If you have set your own valid color symbols through option 4, then the first line of instructions below will differ slightly after displaying "Press". The colors on each side are to be input from top to bottom, left to right by pressing the valid color symbol. If you make a mistake, press the left arrow and then press the correct color symbol. The input screen will look like the following after you have entered the colors on the top and part of the front:



If you press a valid color symbol and it does not display on the screen, then there are already 9 squares entered with that color. You probably mistyped one of the other squares. To make a correction, press the left or right arrow keys until the cursor appears at the square that needs to be corrected; then press the new color.

After entering the colors for the top, front, right, back, and left sides, then place the cube so that the **front is facing you** and the top side is facing up. To enter the color symbols on the bottom, **tilt the cube so that the front becomes the top and the bottom becomes the front**, temporarily. Now, enter the colors from top to bottom, left to right as you see them on the front. After doing so, tilt the cube back so that it has its original front and top. You may want to check to make sure that everything was entered correctly. If you encounter an error, then you will need to move the cursor to the positions that need to be changed. First clear them by pressing the space bar; then enter the correct color symbols. When you have finished entering, the screen will look similar to the following:



At this point, you may press the ESC key. If you press the ESC key before all 54 squares have been filled, then the following message is displayed, "Input is incomplete. Try again? (Y/N)". If you enter 'N', then the program returns to the menu. You may continue inputting by pressing 'Y'. If you entered a side in an improper manner, especially the bottom, then this message will display, "A side is incorrect. Try again? (Y/N)". To correct your mistakes, press 'Y' and then clear the squares that are incorrect and then re-enter them.

Graphical Solution

After ESC is pressed during the input routine, the computer will then generate the entire solution within the next 1 to 2 seconds, if the puzzle can be solved. It is possible to have a Rubik's Cube that is impossible to solve. This can happen if a person takes apart the pieces of the puzzle and puts them back in an improper order. If a person randomly puts back the pieces, then there is only a one in twelve chance that the puzzle has a solution. If no solution is possible the following message is displayed:

```
An error has been detected.
Possibly, you have an im-
possible cube.
     Press any key.
```

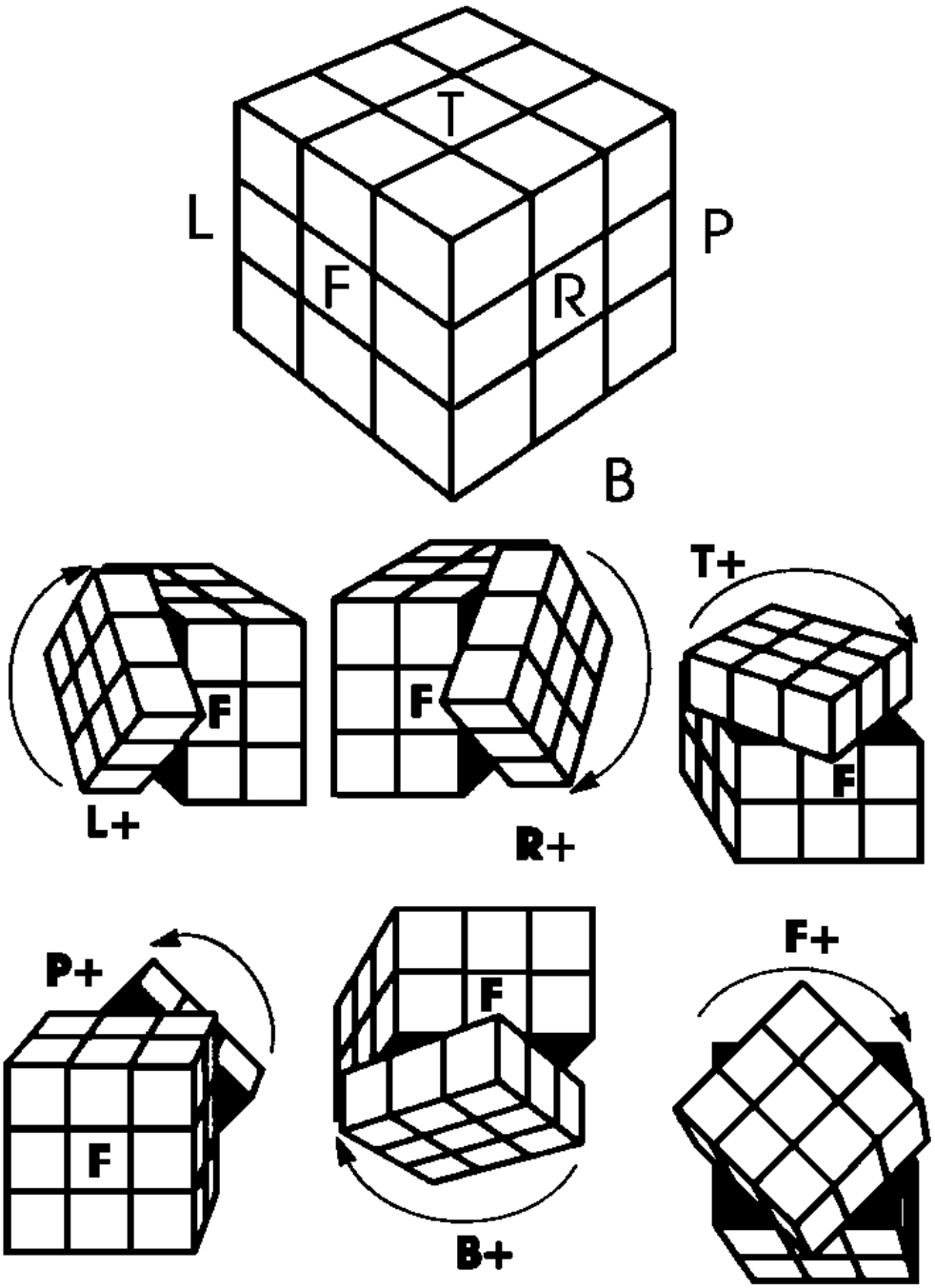
If a solution is possible, then the screen will clear and a graphical sketch of the Rubik's Cube will appear along with a new set of instructions (shown below). To see the first move, you must press either the space bar, the right arrow, or the Enter key. One move of the solution will appear in large symbols, underneath the phrase, "Do the move:". The number of moves made will appear followed by the total number of moves required in the complete solution. The screen is formatted similarly as show below:

```
Graphical Solution

Do the move:

    T+
------------    A graphical Rubik's
                Cube will appear here
  1 move        with the colors currently
out of 134      on the front side after
                the move has been made.

----------------------------------------
Press: Space, ->, or Enter and perform
move shown above, then compare side(s);
or (T, F, R, P, L, B) to view sides;
or <- for previous move; or ESC to quit.
```

Each move will either: 1) turn one side independently of the others, or 2) rotate the entire cube so that a new side becomes the front.

First, most moves will consist of a sides' symbol (T, F, R, L, or B) and a rotation (+, 2, or -). This move means to turn the corresponding side clockwise (if a + follows), or counter- clockwise (if a - follows), or 2 times either way (if a 2 follows). Each move is made as if you are viewing it face to face. For Example, T+ means "turn the Top side clockwise." Furthermore, B- means "turn the Bottom side counter-clockwise" as if you are looking at it from the bottom. (See illustrations on next page)

ILLUSTRATIONS: The six faces are indicated by their initials: top (T), right (R), front (F), left (L), posterior (P), and bottom (B).



Keeping the front face oriented toward you—all clockwise rotations are shown. To make counter-clockwise turns, simply rotate the faces in the opposite directions. All rotations should be made as though you were viewing each face from the front. A clockwise turn would always be a twist to the right as if viewed from the front. See illustrations above.
(Text/Graphic illustrations taken from ***The Simple Solution to Cubic Puzzles***, p. 52)

Another move might be to Rotate the Cube (RC). RC+ means to rotate the cube clockwise as if you were looking at the top. Therefore, the right side becomes the new front, the front becomes the left side, and the top remains the same. RC- and RC2 mean to rotate the cube counter-clockwise and twice, respectively, as if you are viewing the cube from the top.

After completing the move, compare the colors on the side displayed with the colors on the corresponding side of your cube. To view the colors on the other sides, press T, F, R, P, L, or B.

To see the next move, press either the space bar, right arrow, or Enter key. The colors on the new front side will appear, as a result of making the next move. Compare this side with yours and continue to make moves until your cube is solved. If you make a mistake, then undo the move you made. You may then need to either make the proper move or have to press the left arrow key to view the previous move.

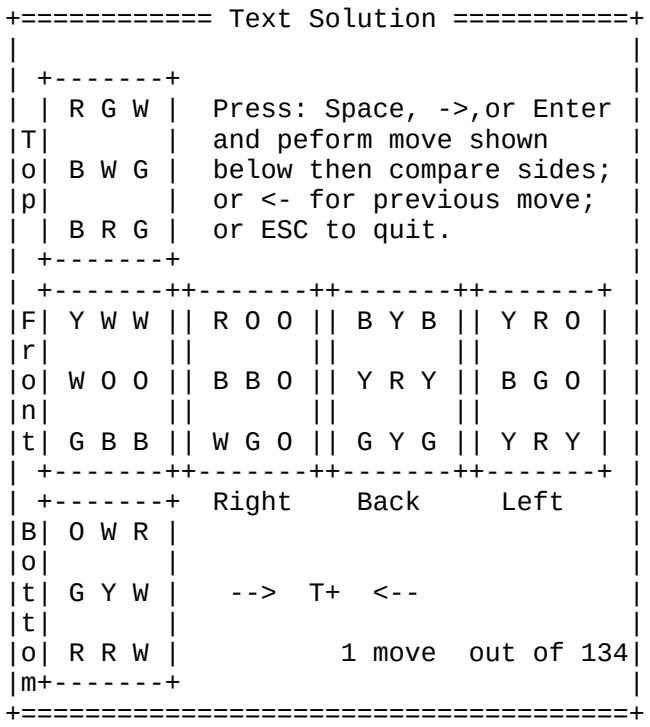
After you have completed all the moves, the computer will "congratulate" you. If you want to return to the main menu without completing all the moves, then press the ESC key.

Text Solution

After ESC is pressed during the input routine, the computer will then generate the entire solution within the next 1 to 2 seconds, if the puzzle can be solved. It is possible to have a Rubik's Cube that is impossible to solve. This can happen if a person takes apart the pieces of the puzzle and puts them back in an improper order. If a person randomly puts back the pieces, then there is only a one in twelve chance that the puzzle has a solution. If no solution is possible the following message is displayed:

```
An error has been detected.
Possibly, you have an im-
possible cube.
   Press any key.
```

If a solution is possible, then a new set of instructions will be displayed. To see the first move, you must press either the space bar, the right arrow, or the Enter key. One move of the solution will appear between two arrows as displayed below, underneath the Right and Back sides. The number of moves made will appear followed by "move out of" and then the total number of moves required in the complete solution, as show below:



Each move will either: 1) turn one side independently of the others, or 2) rotate the entire cube so that a new side becomes the front.

First, most moves will consist of a sides' symbol (T, F, R, L, or B) and a rotation (+, 2, or -). This move means to turn the corresponding side clockwise (if a + follows), or counter-clockwise (if a - follows), or 2 times either way (if a 2 follows). Each move is made as if you are viewing it face to face. For Example, T+ means "turn the Top side clockwise." Furthermore, B- means "turn the Bottom side counter-clockwise" as if you are looking at it from the bottom. (See illustrations of moves under the topic ***Graphical Solution***)

Another move might be to Rotate the Cube (RC). RC+ means to rotate the cube clockwise as if you were looking at the top. Therefore, the right side becomes the new front, the front becomes the left side, and the top remains the same. RC- and RC2 mean to rotate the cube counter-clockwise and twice, respectively, as if you are viewing the cube from the top.

After completing the move, compare the colors on the sides of your cube with the colors on the corresponding sides displayed.

To see the next move, press either the space bar, right arrow, or Enter key. The colors on the new sides will appear, as a result of making the next move. Compare these sides with yours and continue to make moves until your cube is solved. If you make a mistake, then undo the move you made. You may then need to either make the proper move or have to press the left arrow key to view the previous move.

After you have completed all the moves, the computer will "congratulate" you. If you want to return to the main menu without completing all the moves, then press the ESC key.

Instructions

Choosing option 3 from the main menu allows the user to view a brief description of each of the menu options. It will also emphasize that the first option that should be chosen (after reading the instructions) is number 4, **Set Valid Colors**. The program must know the exact colors used on your cube in order to provide a solution.

After option 4 is completed, you may then enter the color symbols on the cube into the computer by either choosing option 1 or 2. Both the **Graphical Solution** and the **Text Solution** use the same input routine. After describing the cube to the computer, the program will either remain in text mode (if a text solution was desired) or will enter graphics mode (if a graphical solution was desired).

Set Valid Colors

The cube has six sides, each with a unique color in its original state. However, not all versions of the cube puzzle have the same six unique colors. The original Rubik's Cube, produced by Ideal Toy Corporation, has as its colors, blue, green, orange, red, white, and yellow. Other cubes produced by different corporations have colors such as purple, green, orange, pink, white, and yellow. There many other variations.

The ***Computer Oriented Solutions to the Rubik's Cube*** program solves the puzzle by placing each of the colors on its own side. Thus, the computer needs to know the exact colors used on your cube. Instead of entering the actual name of the color, the computer will accept a one letter color symbol, such as the first letter in the name of the color. Six unique color symbols must be entered in order to properly describe the appearance of the cube.

If your cube has the same six colors that are on the original Rubik's Cube, then you can press the Enter key six times so that the program defaults to the first letter of the following colors: (B)lue, (G)reen, (O)range, (R)ed, (W)hite, (Y)ellow. **If your cube has colors other than these, then you must enter the six color symbols that are on your cube before you can obtain a Graphical or Text Solution.** To do so, press a one letter color symbol to replace the default values that are shown that differ. If some of the color symbols match, then press the Enter key to select the default value shown.

Test Program

Choosing option 5 brings you to the routine that tests the program ***Computer Oriented Solutions to the Rubik's Cube***. Furthermore, this option provides statistical facts pertaining to this computer oriented solution. The first command requiring a response is:

```
Enter # of cubes to solve (1-14): --
```

You must enter a number between 1 to 14 indicating the number of **imaginary cubes** that you want the computer to solve. Suppose you enter 10. The computer will simulate 10 solved cubes within its memory. The next command displayed is:

```
Enter # of random turns (1-30): --
```

You must enter a number between 1 to 30 indicating the number of random turns (or moves) to make to each of the 10 solved puzzles. If you choose a relatively small number, less than 15, then the puzzle will not be very mixed up. However, you may want to gather statistics for solutions to cubes that are not mixed up well. For our sake, let's choose 30 random turns. Each turn is randomly chosen from the following moves:

```
T+, T2, T-, F+, F2, F-, R+,  R2,  R-,
L+, L2, L-, B+, B2, B-, RC+, RC2, RC-
```

The notation of moves are explained in the ***Graphical Solution*** section. With 10 cubes each randomly scrambled, the computer then attempts to solve each one. The program stores the number of moves it took to solve each cube, and then displays the cube number followed by the number of moves it took to solve the puzzle. The program then displays the "Average # of moves", "Most moves", and "Least moves".

Statistically speaking, the program will solve a randomly mixed up cube in an average of 140 moves (including Rotating Cube moves). Without counting Rotation moves, the program will solve a cube in an average of 125 moves (assuming 15 rotations). Other authors of algorithm solutions do not include Rotating Cube moves in their statistics, but only include those moves that permute the colors on the cube. The program solution seems to have an upper limit of approximately 170 moves (including RC's). The least number of moves needed to solve the cube is trivially 0, if the random moves leave the cube unchanged.

At this point in time, the program will always solve a valid Rubik's Cube. However, in the developmental stages of this program, the computer did not always solve the puzzle correctly. In such cases, this **Test Program** routine displays the set of random moves the computer made to its puzzle in order for the user to recreate the error and detect the part of the program needing correction.

Research - History

The Rubik's Cube, which is also called the Magic Cube, is a three by three by three inch cube. A cube has six sides or faces which contain a different color. Each face is divided up into nine squares. A brilliant inner mechanism of a spring loaded spindle allows sides of the cube to be rotated either vertically or horizontally and independent of the other sides. In the cube's pristine condition it has a solid color on each face, but is mixed up by just a few random moves. The cube comes in six different colors which can then be mixed up. The original Rubik's Cube has these six colors: white, yellow, orange, red, green, and blue. The object is to get the cube back to its original positions so that all six sides have nine identical squares with respect to their colors.

This great puzzle, the Rubik's Cube, was invented by Erno Rubik, who has held degrees in architectural engineering and interior design and teaches at the Academy of Applied Arts in Budapest, Hungary. From this job of teaching he only earned an estimated $1,820 a year. Although he was licensed the right to the cube at a modest fee, he made $800,000 for 1980-81.

The summer of 1974 was when Rubik had thought about the laws of geometry and had started playing with the ideas of a three dimensional object that could rotate around any of the three axes. He kept turning the object in his mind till he could turn the theoretical construction gloating in his head into something real. He had a problem and wanted to make a mechanical devise capable of converting all of the complicated movements into simple twists and turns that would change the arrangement of squares on the faces of an actual cube. After working hard on his model he gave a few turns and then found out that he made a puzzle to solve.

Weeks went by and he finally got some of the different sections of the cube back to their starting point. What had started to be an aid for his students to give greater experience in dealing with three dimensional objects turned out to be much more.

In 1975, he wrote up the details of the cube's construction and obtained a patent. Rubik then interested a small toy-manufacturing co-operative, Polutechnika, in his invention and soon it was in mass production. The original production run of 5000 was shipped to Hungarian stores just before Christmas 1977. Within a few days they had sold out, and 7000 more cubes were rushed off the production line.

In 1980, cubes were bought at prices from $5 to $15. A total of 30 million of the cubes were purchased worldwide by 1983. These cubes were sold in all kinds of shops. In Rubik's homeland, Hungary, a controlled economy had not kept up with demand for manufacture of the cube, and they tended to buy three or four at a time when they were available. In the United States, Hong Kong, and the Caribbean, by arrangement with Komsumex, one of Hungary's import/export agencies, the Ideal Toy Corp. manufactured 1.5 million cubes every month. Logical Games Inc., of Haymarket, Va., which introduced the puzzle to the United States under the name Magic Cube in 1979, has manufactured the Cube.

One reason why Ideal Toy Corporation was so successful with the selling of the Rubik's Cube was because they advertised. Helfgart, Towne & Silverstein, were the company's agency for 16 years. The Rubik's Cube showed up at a Toy Fair for the first time in February 1980, but its reception was less than spectacular. Beginning the 4th of July weekend, advertising started with two weeks of television exposure using the 30-second commercial made up of three vignettes, and then four weeks using three 10-second spots, each of which was one of the vignettes. The television advertising was used in four major markets

along with small-space newspaper ads. After a respite during the summer, advertising started again in September 1981, and was in 100 spot markets by December 1981.

Ideal Toy Corporation agreed to be acquired by the toy division of CBS Inc. in a transaction valued at about $58 million. Under terms of the proper acquisition, CBS would pay $14.85 a share for the 3.9 million outstanding common shares of Ideal, which is based in Hollis, Queens. For the fiscal year 1981, the first full year to include results from sales of the cube, Ideal reported a net income of $19.1 million or $4.94 a share versus a loss of $15.45 million in 1980 and income of $3.7 million in 1979.

Magic Puzzler, Wonderful Puzzler, and Le Cube, are all variations of Rubik's Cubes except that they do not bear the trademark of the Rubik's Cube and are made by companies other than Ideal. These puzzles retail one-third the price of Rubik's Cube and were selling very well. Ideal had filed dozens of civil suits against cube distributors and retailers across the country. In Federal District Court of Manhattan alone there were seven suits pending, one against the largest distributors, the John Hansen Company of Milbrae California. In May of 1981, Ideal charged Hansen with packaging its cube so that customers would mistake the puzzle for Rubik's Cube. Hansen then sold his cube under the name of Le Cube. Ideal Toy Corporation also filed a suit against United Supply Corporation, claiming unfair competition and trade dress copying of Ideals popular Rubik's Cube. Ideal brought Macy's to court in April of 1981, for displaying hundreds of nameless cubes piled right next to a video screen continuously showing an advertisement for Rubik's Cube. Macy's agreed not to sell nameless cubes in that way anymore, and in return Ideal agreed not to discuss the case publicly.

The privately held Moleculon Research Corporation of Cambridge filed a $60 million patent infringement suit against Ideal in Federal District Court in Delaware. Moleculon contends that Mr. Nichol's 1972 American patent for a cubic "pattern-forming puzzle" which closely resembles the 30 million cubes that Ideal sold worldwide since 1980, beat Rubik's invention by 2 years. Rubik holds a 1974 patent for his devise in Hungary but never applied for one in the United States or elsewhere. Mr. Nichol's said he has been trying to sell his puzzle since 1969 but was rejected because Moleculon did not want to show a prototype until it received a patent protection. Morris & Safford, of Ideal say Moleculon's 1972 patent was invalid because the ideas contained in it were not original and that Mr. Nichol's patent, which primarily describes a cube held together by magnets, did not cover the mechanical devise that enables puzzle mechanical solvers to move each piece of Rubik's Cube independently.

Research - Competitions

The first organized cube competition began on January 4, 1980. Kate Fried organized the competition in Budapest, Hungary at the Youth Mathematical Circle and at a Magic Cube Fans Club. Each competitor had to supply his own cube and put it in a cardboard box with his name on it. The judge(s) then scrambled the cubes identically and put them back in their boxes. Viktor Toth, a student, won this first competition in 55 seconds. After this competition, the winner and the runner-up had a play-off in which each scrambled the other's cube. Toth won this competition also.

The second competition was conducted in March, and over a thousand people came, packing into several rooms. This competition consisted of those people who had previously shown that they can solve the cube in less than 90 seconds. Nine people competed in the competition. The winner solved his cube in 40 seconds.

The first to try her skill publicly with the cube in America was Zsa Zsa Gabor, who was hired by the Ideal Toy Corporation, to promote the creation of her fellow Hungarian. The first regional competition of the title of the United States Rubik's Cube championship was held on July 25, 1981.

Research - Solution Manuals

Without help, solving the Rubik's Cube could have taken several months. For skilled mathematicians it would have taken several weeks to complete a cube. The most advanced computer would have taken 1.4 million years to examine all its 43,252,003,274,489,856,000 possible combinations. The solution to the Rubik's Cube, along with many other puzzles, was found by the use of abstract algebra, or group theory. A person did not have to know group theory to solve these puzzles. All a person had to do was learn or memorize a solution given by authors that have written books about the solutions of the puzzles. Once a person had read and learned the book well, that person could usually solve a puzzle with great ease.

James G. Nourse, a Stanford University Chemist who worked out his formula during his Christmas vacation, gets the Rubik's Cube back to its proper place in one minute. Nourse has sold over one million copies of his solution book, ***The Simple Solution to Rubik's Cube***. He is also the author of ***The Simple Solution to Cubic Puzzles***, which teaches the reader how to solve the Pyraminx, Missing Link, the Barrel, Picture Cube, Rubik's Magic Snake, and the Octagon.

The Hawaiian high school student, Minh Thai, wrote ***The Winning Solution***, which reveals his strategy and racing secrets for the Rubik's Cube. He also wrote a solution book for Rubik's Revenge, a four by four by four cube puzzle. Minh's book, The Winning Solution to Rubik's Revenge, also reveals his strategy and racing secrets for a puzzle. Minh became the U.S. National Champion at the Rubik's Cube-a-Thon aired on ABC's ***That's Incredible*** on December 7, 1981, by unscrambling Rubik's Cube in 26.04 seconds. He also restored seven scrambled cubes in under 30 seconds each in a public exhibition in Los Angeles.

Another author of a solution book was Patrick Bassert. Bassert was only 13 years old when he sold 750,000 copies of his solution book, ***You Can Do The Cube***, in London and earned more than $60,000. The appeal of his book is its relative clarity, and he accompanied his instructions with a system of diagrams and symbols showing how to rotate and move the squares. He and his cousin began charting basic moves on a tattered piece of graph paper that grew quickly into his book. He claims that "doing the cube depends not so much on math as on logic. You need to think logically and see what's happening to see the logic of the relationships between the various pieces, then it becomes clear."

Research - Group Theory

Most of the people who wrote solutions to various puzzles used group theory. The following is a brief introduction to the group theory used to solve puzzles. A group, G, is a set of elements with a well defined binary operation, (+), such that:

1) If a, b are elements of Group G, then so is a(+)b (closure under operation);
2) There exists an element e of Group G such that a(+)e= a =e(+)a for all a an element of G (identity);
3) If a is an element of G, there exists an element a' of G such that a(+)a'= e =a'(+)a (inverse);
4) If a, b, and c are elements of G, then (a(+)b)(+)c= a(+)(b(+)c) (associative law).

A ***permutation*** of the cube is a sequence of moves that rearrange the pieces and color squares of the puzzle. There are over 4.3 quintillion unique permutations for a standard Rubik's Cube. Each permutation is considered an element of Group G. The group consists of the collection of all unique permutations of the cube. "There are six basic moves or permutations of the cube:"

T+, F=, R+, P+, L+, B+,
Where each letter represents a side of the cube,
And + indicates a 90 degree turn clockwise.

By combining the above moves, all the other elements of the group can be generated. Some other elements include: R+F+, L+B+B+T+.

The binary operation, **(+)**, is defined as ***combining*** two elements in the group, which means to perform them in sequence. If this is a group, then the ***combination*** of any two members will result in a permutation that is a member of the collection.

The permutation that results in no change is called the ***identity*** permutation. The ***order*** of the permutation is the least number of times that the permutation needs to be performed to get the identity permutation. For example, the permutation element R2F2 has an order of 6 since the identity element is obtained by combining the moves R2F2 six times. No element has an order larger than 1260. The permutation which undoes a given permutation and gives the identity is called its ***inverse***. A group does not have to be commutative, but when the order in which two permutations are combined is not important, it is said to commute. The Group G, of permutations, is not commutative.

All permutations have cycles which switch certain pieces around; and if this permutation is done enough times, the original permutation shows up again. The length of a cycle is the number of elements in the cycle, and the order of a cycle is its length. A transposition is a cycle of length two. Two cycles are disjoint if there are no common positions which are formed in both. A permutation which is the product of an even number of transpositions is called even, and one which is the product of an odd number of transpositions is called odd. If every completed move of a puzzle is an even permutation and its eventual desired permutation is odd, then the puzzle is impossible to solve.

Research - Group Theory Solutions

The basic mathematical objective is to restore the cube from any scrambled pattern back to its original position, where each side has a solid color. In general, it is desirable to obtain a set of moves to obtain a particular pattern from any other valid pattern.

"The group of all possible permutations is as follows. The 8 corner pieces can be permuted among themselves in any way, giving 8! ways (! means factorial), and the 12 edge pieces can be permuted among themselves in any of 12! ways, except that the total permutation of corners and edges must be even. Further, independently of the movement of pieces, it is possible to flip the orientations of any two edge pieces and twist any two corner cubes in opposite directions. This means that it is possible to orientate all but one of the edges or corners--the orientation of the last one being forced. This means there are a total of

$$N = \frac{8!\ 12!}{2} \times 3^{8} \times 2^{12} \times \frac{1}{3} \times \frac{1}{2} = 43252003274489856000 \sim 4.3 \times 10^{19}$$

$$= 2^{27} \times 3^{14} \times 5^{3} \times 7^{2} \times 11$$

different patterns." The total number of ways a person can reassemble the cube after taking it apart is indicated by the numerator N. The denominator of 12 indicates that there are "12 distinct orbits of constructible patterns. (An orbit is the set of all patterns reachable from a given pattern by application of our group. It is impossible to get out of one orbit into another by use of the group of motions.)" If the cube is taken apart and randomly reassembled, then there is only a 1/12 chance of being able to solve the cube.

The following people are known for their developments of algorithms to solve the Rubik's Cube by using group theory. Morwen B. Thistlethwaite has done much work in the area of computing to find efficient algorithms. He had his algorithm down to 52 moves to solve the cube, but he was hoping to lower this to 50 by doing more computing. Thistlethwaite believes that it may be reducible to 45 with a lot of effort and researching.

In an article in the ***Journal of Recreational Mathematics,*** John Conway and Dave Benson showed how to always restore the cube in at most 100 moves. They did bottom edges, then bottom corners and middle edges together and then use their tables to first position and then orient the top layer, in at most 13 to 19 moves respectively. Roy Nelson solves the cube by doing the bottom and middle, then orienting top corners, then positioning them, then orienting top edges, then positioning them. Without having a cube, Hanke Bremer produced a solution. He gets corners in place, then oriented, then edges in place then oriented. Michel Daughpin, a sixth year student at a lycee in Luxembourg, has a method similar to Bremer's.

3-D Jackson has compiled "The Cube Dictionary" giving about 177 one layer processes. He claims that these allow the following algorithm to produce at most 92 moves: solve bottom edges (10 moves); bottom corners (22); middle edges (28); upper corners in place (8); then orient (14); upper edges (10).

Gerzson Keri says he can do two layers in at most 57 moves and that he believes any one layer process can be done in at most 18 moves. He claims to have over 100 one layer processes of at most 15 moves. He has published a lengthy two-part article in Hungary about his solution. One of his methods is entirely tabular--proceeding one piece at a time, seeing where it is and then

looking up a process to get it correctly in place. The table contains 24 + 21 + 18 + ... + 6 + 3 + 24 + 22 + 20 + ... + 4 + 2 = 264 entries of which 5 are impossible and 20 are trivial. There are at most 182 moves that are required.

Due to Adam Kertesz, Bill McKeeman has an algorithm which does two layers, then upper corners, then orients upper edges and then upper corners. Kersten Meier has a layer by layer algorithm but does not detail the working for the last layer. Dame Kathleen Ollerenshaw's method does the bottom face, then the top corners, then the middle slice edges, then top edges. She claims that the algorithm produces an average of 80 moves.

Based on Penrose’s algorithm, solving bottom edges, middle slice edges, top edges, then corners in place then oriented, Zoltan Perjes wrote out his own algorithm. He also outlined Rubik's original fast method: down corners, upper corners in place, then oriented, three down edges, three upper edges, the other down and upper edges, then the middle slice edges. Don Taylor does down edges, down corners, middle edges, upper corners in place, upper edges in place, upper corners oriented, then upper edges oriented.

Michael Vaughan-Lee has made a careful study of an edge first process: down edges (13), three middle edges (14), upper edges and last middle edge (17), corners in place (30) and then in orientation (38), but the last two steps can be reduced to 26 + 24, 32 + 22 or 28 + 22, giving a maximum of 98 moves.

David Sigmaster's algorithm looks like this:

1) Put all bottom edges correctly in place.
2) Put all bottom corners correctly in place.
3) Put middle slice edges correctly in place.
4) Flip top edges so all upper faces are up.
5) Make top orientation correct.
6) Put top edges correctly in place.
7) Put top corners in their right positions.
8) Twist top corners into their correct orientations.

Will there ever be an algorithm that is the shortest? Trivially, some positions of the cube can be unscrambled in 0, 1, 2, or 3 moves. Given any scrambled cube, what is the least number of moves required to restore the cube to its six sides of solid colors? There are several ways to show that at least 17 moves are required for the most scrambled cube. Can every cube be solved in at most 17 moves? The algorithm that obtains the shortest set of moves is know as God’s algorithm.

Description of Program Solution Strategy

If a person is able to solve a puzzle himself in a logical fashion, then a person probably can write a program to solve the puzzle. With this thought as a basis, how does a person solve the Rubik's Cube? There are 12 corner cubes, 8 edge cubes, and 6 center cubes that need to be positioned and oriented correctly so that each of the 6 sides contain a solid color. First, you need an overall strategy or algorithm before considering the smaller details. The following is a brief sketch of the algorithm that I use to solve the Rubik's Cube:

1) Position and orient Top edges
2) Position and orient Top corners
3) Position and orient Vertical edges
4) Position Bottom corners
5) Orient Bottom corners
6) Position Bottom edges
7) Orient Bottom edges

The center cube on each side does not "move", and they determine the color to be restored on their sides'. For each of the above steps, particular cube edges or cube corners need to be located and then positioned and oriented in a certain spot. There are a finite number of places that this particular piece could be in. For each spot, group theory can aid in determining the set of moves to make to the cube to position and/or orient a particular piece without "messing up" too much of the rest of the puzzle. Once one piece is put in place, another piece needs to be located on this puzzle and then positioned and oriented.

With this manual algorithm, I could program the computer to solve the Rubik's Cube. The two biggest questions that I needed answers to were, "how do I let the computer know what the puzzle looks like?" and "how do I tell the computer to ***move*** the cube?"

The computer needs to know the colors on each of the 54 squares of the puzzle. The program accepts a letter as a color symbol for each of the squares and stores it in the designated array position. The color symbols on the top side are stored in A[1..9], the front side colors are stored in A[10..18], the right in A[19..27], the posterior (back) in A[28..36], the left in A[37..45], and the bottom in A[46..54].

With the appearance of the puzzle, the computer does a step of the program's overall algorithm and finds a spot that does not have the correct piece. The program then searches for the misplaced cube by asking itself logical "if-then" statements regarding the positions and colors of the squares. When it finds the correct piece, it reads a set of moves to position and/or orient this piece. The program then branches to a subroutine that "moves" the puzzle. To make a move, the program will actually exchange some of the variables in the array.

After all the moves have been made to place the desired piece, the computer will then know the exact appearance of the new puzzle as if a person had actually performed the moves. The program then does the next part of its algorithm and repeats the above process until it has completed each of the seven steps.

Partial Pascal Solution Listing

```
{ -- SOLUTION.INC}
procedure SolveTopEdges ({giving} var MoveError: Boolean);
{ -- This procedure will store the moves in Solution to restore top edges. }
{ -- Strategy: Work on front-top edge then rotate cube to next face. }

  var
    Face:        Integer;
    Tmid, Fmid:  Char;

begin
  for Face := 1 to 4 do begin
    Tmid := A[5];   { -- Has Color symbol of Top middle square. }
    Fmid := A[14];  { -- Has color symbol of Front middle square. }

    if (A[8] = Fmid) and (A[11] = Tmid) then  { -- Orient FT }
      AddToSolution ('F+ T- R+ T+ ')

    else if (A[6] = Fmid) and (A[20] = Tmid) then  { -- Move RT to FT }
      AddToSolution ('R- F- ')
    else if (A[6] = Tmid) and (A[20] = Fmid) then  { -- Move RT to FT }
      AddToSolution ('R- T- R+ T+ ')

    else if (A[2] = Fmid) and (A[29] = Tmid) then  { -- Move PT to FT }
      AddToSolution ('T+ R- T- F- ')
    else if (A[2] = Tmid) and (A[29] = Fmid) then  { -- Move PT to FT }
      AddToSolution ('T+ R2 T- B- F2 ')

    else if (A[4] = Fmid) and (A[38] = Tmid) then  { -- Move LT to FT }
      AddToSolution ('L+ F+ ')
    else if (A[4] = Tmid) and (A[38] = Fmid) then  { -- Move LT to FT }
      AddToSolution ('L2 B+ F2 ')

    else if (A[15] = Fmid) and (A[22] = Tmid) then  { -- Move FR to FT }
      AddToSolution ('F- ')
    else if (A[15] = Tmid) and (A[22] = Fmid) then  { -- Move FR to FT }
      AddToSolution ('R- B- R+ F2 ')

    else if (A[24] = Fmid) and (A[31] = Tmid) then  { -- Move PR to FT }
      AddToSolution ('R+ B- R- F2 ')
    else if (A[24] = Tmid) and (A[31] = Fmid) then  { -- Move PR to FT }
      AddToSolution ('R2 F- R2 ')

    else if (A[33] = Fmid) and (A[40] = Tmid) then  { -- Move LP to FT }
      AddToSolution ('L2 F+ L2 ')
    else if (A[33] = Tmid) and (A[40] = Fmid) then  { -- Move LP to FT }
      AddToSolution ('L- B+ L+ F2 ')

    else if (A[13] = Fmid) and (A[42] = Tmid) then  { -- Move FL to FT }
      AddToSolution ('F+ ')
    else if (A[13] = Tmid) and (A[42] = Fmid) then  { -- Move FL to FT }
      AddToSolution ('T+ L- T- ')
```

```
    else if (A[17] = Fmid) and (A[47] = Tmid) then  { -- Move BF to FT }
      AddToSolution ('F2 ')
    else if (A[17] = Tmid) and (A[47] = Fmid) then  { -- Move BF to FT }
      AddToSolution ('F+ T+ L- T- ')

    else if (A[26] = Fmid) and (A[51] = Tmid) then  { -- Move BR to FT }
      AddToSolution ('B- F2 ')
    else if (A[26] = Tmid) and (A[51] = Fmid) then  { -- Move BR to FT }
      AddToSolution ('R+ F- R- ')

    else if (A[35] = Fmid) and (A[53] = Tmid) then  { -- Move BP to FT }
      AddToSolution ('B2 F2 ')
    else if (A[35] = Tmid) and (A[53] = Fmid) then  { -- Move BP to FT }
      AddToSolution ('B+ L- F+ L+ ')

    else if (A[44] = Fmid) and (A[49] = Tmid) then  { -- Move BL to FT }
      AddToSolution ('B+ F2 ')
    else if (A[44] = Tmid) and (A[49] = Fmid) then  { -- Move BL to FT }
      AddToSolution ('L- F+ L+ ');

    if (A[8] = Tmid) and (A[11] = Fmid) and (Face < 4) then  { --Rotate Cube }
      AddToSolution ('RC+')
    else If (A[8]<>Tmid) or (A[11]<>Fmid) then begin  { -- Fatal error }
      ErrorInMoves;  MoveError := True;  Exit;  end;
  end;      { -- for Face }
end;
```

```
procedure SolveTopCorners ({giving} var MoveError: Boolean);
{ -- This procedure will store the moves in Solution to restore top corners. }
{ -- Strategy: Work on front-right-top corner then rotate cube to next face. }

  var
    Face:        Integer;
    Tmid, Fmid:  Char;

begin
  for Face := 1 to 4 do begin
    Tmid := A[5];   { -- Has Color symbol of Top middle square. }
    Fmid := A[14];  { -- Has color symbol of Front middle square. }

    if (A[9] = Fmid) and (A[19] = Tmid) then        { -- Orient FRT }
      AddToSolution ('R- B2 R+ F+ B2 F- ')
    else if (A[12] = Tmid) and (A[19] = Fmid) then  { -- Orient FRT }
      AddToSolution ('F+ B2 F- R- B2 R+ ')

    else if (A[3] = Tmid) and (A[21] = Fmid) then   { -- Move PRT to FRT }
      AddToSolution ('R+ B+ R- F+ B2 F- ')
    else if (A[3] = Fmid) and (A[28] = Tmid) then   { -- Move PRT to FRT }
      AddToSolution ('R+ B- R- F+ B- F- ')
    else if (A[21] = Tmid) and (A[28] = Fmid) then  { -- Move PRT to FRT }
      AddToSolution ('R+ B2 R2 B+ R+ ')

    else if (A[1] = Tmid) and (A[30] = Fmid) then   { -- Move PLT to FRT }
      AddToSolution ('L- B2 L+ B- R- B+ R+ ')
    else if (A[30] = Tmid) and (A[37] = Fmid) then  { -- Move PLT to FRT }
      AddToSolution ('L- B+ L+ R- B2 R+ ')
    else if (A[1] = Fmid) and (A[37] = Tmid) then   { -- Move PLT to FRT }
      AddToSolution ('L- B- L+ F+ B- F- ')

    else if (A[7] = Tmid) and (A[39] = Fmid) then   { -- Move FLT to FRT }
      AddToSolution ('L+ B2 L- F+ B- F- ')
    else if (A[10] = Fmid) and (A[39] = Tmid) then  { -- Move FLT to FRT }
      AddToSolution ('L+ R- B+ L- R+ ')
    else if (A[7] = Fmid) and (A[10] = Tmid) then   { -- Move FLT to FRT }
      AddToSolution ('F- B2 F2 B- F- ')

    else if (A[18] = Tmid) and (A[48] = Fmid) then  { -- Move FRB to FRT }
      AddToSolution ('F+ B+ F- ')
    else if (A[25] = Fmid) and (A[48] = Tmid) then  { -- Move FRB to FRT }
      AddToSolution ('R- B+ R+ F+ B2 F- ')
    else if (A[18] = Fmid) and (A[25] = Tmid) then  { -- Move FRB to FRT }
      AddToSolution ('R- B- R+ ')

    else if (A[27] = Fmid) and (A[34] = Tmid) then  { -- Move PRB to FRT }
      AddToSolution ('F+ B- F- ')
    else if (A[27] = Tmid) and (A[54] = Fmid) then  { -- Move PRB to FRT }
      AddToSolution ('B2 R- B+ R+ ')
    else if (A[34] = Fmid) and (A[54] = Tmid) then  { -- Move PRB to FRT }
      AddToSolution ('B- R- B+ R+ F+ B2 F- ')
```

```
    else if (A[36] = Tmid) and (A[52] = Fmid) then  { -- Move PLB to FRT }
      AddToSolution ('R- B2 R+ ')
    else if (A[36] = Fmid) and (A[43] = Tmid) then  { -- Move PLB to FRT }
      AddToSolution ('F+ B2 F- ')
    else if (A[43] = Fmid) and (A[52] = Tmid) then  { -- Move PLB to FRT }
      AddToSolution ('B2 R- B+ R+ F+ B2 F- ')

    else if (A[45] = Tmid) and (A[46] = Fmid) then  { -- Move FLB to FRT }
      AddToSolution ('R- B+ R+ ')
    else if (A[16] = Tmid) and (A[45] = Fmid) then  { -- Move FLB to FRT }
      AddToSolution ('B2 F+ B- F- ')
    else if (A[16] = Fmid) and (A[46] = Tmid) then  { -- Move FLB to FRT }
      AddToSolution ('B+ R- B+ R+ F+ B2 F- ');

    if (A[9] = Tmid) and (A[12] = Fmid) and (Face < 4) then { --Rotate Cube }
      AddToSolution ('RC+')
    else if (A[9]<>Tmid) or (A[12]<>Fmid) then begin  { -- Fatal error }
      ErrorInMoves;  MoveError := True;  Exit;  end;
  end;      { -- for Face }
end;
```

```
procedure SolveVerticalEdges ({giving} var MoveError: Boolean);
{ -- This procedure will store the moves in Solution to restore vert. edges. }
{ -- Strategy: Work on front-right edge then rotate cube to next face. }

  var
    Face:        Integer;
    Fmid, Rmid:  Char;

begin
  for Face := 1 to 4 do begin
    Fmid := A[14];  { -- Has color symbol of Front middle square. }
    Rmid := A[23];  { -- Has color symbol of Right middle square. }

    if (A[15] = Rmid) and (A[22] = Fmid) then        { -- Orient FR }
      AddToSolution ('R- B+ R+ B+ F+ B- F- B+ R- B+ R+ B+ F+ B- F- ')

    else if (A[24] = Fmid) and (A[31] = Rmid) then  { -- Move PR to FR }
      AddToSolution ('RC+R- B+ R+ B+ F+ B- F- RC-B- F+ B- F- B- R- B+ R+ ')
    else if (A[24] = Rmid) and (A[31] = Fmid) then  { -- Move PR to FR }
      AddToSolution ('RC+R- B+ R+ B+ F+ B- F- RC-R- B+ R+ B+ F+ B- F- ')

    else if (A[33] = Fmid) and (A[40] = Rmid) then  { -- Move PL to FR }
      AddToSolution ('RC2R- B+ R+ B+ F+ B- F- RC2B2 F+ B- F- B- R- B+ R+ ')
    else if (A[33] = Rmid) and (A[40] = Fmid) then  { -- Move PL to FR }
      AddToSolution ('RC2R- B+ R+ B+ F+ B- F- RC2B- R- B+ R+ B+ F+ B- F- ')

    else if (A[13] = Fmid) and (A[42] = Rmid) then  { -- Move FL to FR }
      AddToSolution ('RC-R- B+ R+ B+ F+ B- F- RC+B2 R- B+ R+ B+ F+ B- F- ')
    else if (A[13] = Rmid) and (A[42] = Fmid) then  { -- Move FL to FR }
      AddToSolution ('RC-R- B+ R+ B+ F+ B- F- RC+B+ F+ B- F- B- R- B+ R+ ')

    else if (A[17] = Fmid) and (A[47] = Rmid) then  { -- Move FB to FR }
      AddToSolution ('B- R- B+ R+ B+ F+ B- F- ')
    else if (A[17] = Rmid) and (A[47] = Fmid) then  { -- Move FB to FR }
      AddToSolution ('B2 F+ B- F- B- R- B+ R+ ')

    else if (A[26] = Rmid) and (A[51] = Fmid) then  { -- Move RB to FR }
      AddToSolution ('B+ F+ B- F- B- R- B+ R+ ')
    else if (A[26] = Fmid) and (A[51] = Rmid) then  { -- Move RB to FR }
      AddToSolution ('B2 R- B+ R+ B+ F+ B- F- ')

    else if (A[35] = Fmid) and (A[53] = Rmid) then  { -- Move PB to FR }
      AddToSolution ('B+ R- B+ R+ B+ F+ B- F- ')
    else if (A[35] = Rmid) and (A[53] = Fmid) then  { -- Move PB to FR }
      AddToSolution ('F+ B- F- B- R- B+ R+ ')

    else if (A[44] = Fmid) and (A[49] = Rmid) then  { -- Move LB to FR }
      AddToSolution ('R- B+ R+ B+ F+ B- F- ')
    else if (A[44] = Rmid) and (A[49] = Fmid) then  { -- Move LB to FR }
      AddToSolution ('B- F+ B- F- B- R- B+ R+ ');

    if (A[15] = Fmid) and (A[22] = Rmid) and (Face < 4) then { --Rotate Cube }
      AddToSolution ('RC+')
    else if (A[15]<>Fmid) or (A[22]<>Rmid) then begin  { -- Fatal error }
      ErrorInMoves;  MoveError := True;  Exit;  end;
  end;
end;
```

```
procedure SolveBottomCorners ({giving} var MoveError: Boolean);
{ -- This procedure will store the moves in Solution to restore bot corners. }
{ -- Strategy: First "position" all 4 corners, then "orient" 4 corners. }

  var
    Face:                          Integer;
    Fmid, Rmid, Pmid, Lmid, Bmid: Char;
    BFL,  BFR,  BPR,  BLP:         Boolean;  { -- True if this corner exists. }
    All4Positioned, All4Oriented: Boolean;

begin
  { -- ******************    Position all 4 corners    *************** }
  Face := 1;
  repeat
    If Face > 4 then begin  { -- Fatal error }
      ErrorInMoves;  MoveError := True;  Exit;  end;
    Fmid := A[14];  Rmid := A[23];  Pmid := A[32];  Lmid := A[41];

    { -- Determine which of the 4 Bottom corners exist (= True). }
    BFL := ((A[16] = Fmid) and (A[45] = Lmid)) or
           ((A[16] = Lmid) and (A[46] = Fmid)) or
           ((A[45] = Fmid) and (A[46] = Lmid));
    BFR := ((A[18] = Fmid) and (A[25] = Rmid)) or
           ((A[18] = Rmid) and (A[48] = Fmid)) or
           ((A[25] = Fmid) and (A[48] = Rmid));
    BPR := ((A[27] = Rmid) and (A[34] = Pmid)) or
           ((A[34] = Rmid) and (A[54] = Pmid)) or
           ((A[27] = Pmid) and (A[54] = Rmid));
    BLP := ((A[36] = Pmid) and (A[43] = Lmid)) or
           ((A[36] = Lmid) and (A[52] = Pmid)) or
           ((A[43] = Pmid) and (A[52] = Lmid));

    All4Positioned := True;  { -- Will be after this set, unless no matches. }
    if (BFL and BFR) or (BFR and BPR) or (BPR and BLP) or (BLP and BFL) then
      { -- Either all 4 match, or only 2 corners match. }
      if not (BFL and BFR and BPR and BLP) then  { -- only 2 corners match }
      begin
        if (BFL and BFR) then
          AddToSolution ('RC2')
        else if (BFR and BPR) then
          AddToSolution ('RC-')
        else if (BLP and BFL) then
          AddToSolution ('RC+');
        { -- Exchange adjacent sides BFL and BFR, which are out of place. }
        AddToSolution ('R- B- R+ F+ B+ F- R- B+ R+ B2 ');
      end
      else  { -- null else- since all 4 match, skip to next section. }
    else if (BFL and BPR) or (BFR and BLP) then  {--Pair of diagonals match. }
      begin
      { -- Exchange diagonals BFL and BPL}
      AddToSolution ('F- B- R- B+ R+ F+ ');
      { -- Turn Bottom one rotation left or right to match all 4 corners. }
      if (BFL and BPR) then
        AddToSolution ('B- ')
      else
        AddToSolution ('B+ ');
      end
```

```
    else  { -- No matches found with current Bottom rotation, try another. }
      begin
      AddToSolution ('B+ ');  All4Positioned := False;
      end;

    Face := Face + 1;
  until All4Positioned;


  { -- ******************  Orient 4 corners  ****************** }
  { -- Rotate Cube until 1 of 7 patterns appear, or all 4 are oriented. }
  { -- Perform set of moves.  If pattern is BC1 or BC2, cube is oriented. }

  All4Oriented := False;  Face := 1;
  repeat
    If Face > 4 then begin { -- Fatal error, Pattern not found after 4 turns.}
      ErrorInMoves;  MoveError := True;  Exit;  end;

   Bmid := A[50];
   if (A[46] = Bmid) and (A[48] = Bmid) and (A[52] = Bmid) and (A[54] = Bmid)
     then All4Oriented := True
   else
   if (A[46] = Bmid) and (A[43] = Bmid) and (A[34] = Bmid) and (A[25] = Bmid)
     then begin  { -- BC2 pattern }
       AddToSolution ('B2 R- B2 R+ B+ R- B+ R+ ');  Face := 1;
     end
   else  { -- 6 other patterns possible if the cube is at proper rotation. }
   if (A[46] = Bmid) and (A[36] = Bmid) and (A[27] = Bmid) and (A[18] = Bmid)
   or (A[45] = Bmid) and (A[36] = Bmid) and (A[34] = Bmid) and (A[25] = Bmid)
   or (A[16] = Bmid) and (A[36] = Bmid) and (A[54] = Bmid) and (A[48] = Bmid)
   or (A[16] = Bmid) and (A[52] = Bmid) and (A[54] = Bmid) and (A[18] = Bmid)
   or (A[16] = Bmid) and (A[52] = Bmid) and (A[27] = Bmid) and (A[48] = Bmid)
   or (A[45] = Bmid) and (A[43] = Bmid) and (A[27] = Bmid) and (A[25] = Bmid)
     then begin  { -- BC1 or BC3 or BC4 or BC5 or BC6 or BC7 pattern. }
       AddToSolution ('R- B- R+ B- R- B2 R+ B2 ');  Face := 1;
     end
   else begin  { -- Cube is not yet at proper rotation to match a pattern. }
       AddToSolution ('RC+');  Face := Face + 1;
     end;
  until All4Oriented;
end;
```

```
procedure SolveBottomEdges ({giving} var MoveError: Boolean);
{ -- This procedure will store the moves in Solution to restore bot edges. }
{ -- Strategy: First "position" all 4 edges, then "orient" the 4 edges. }

  var
    Face:                          Integer;
    Fmid, Rmid, Pmid, Lmid, Bmid: Char;
    BF,   BR,   BP,   BL:         Boolean;  { -- True if this corner exists. }
    All4Positioned, All4Oriented: Boolean;
    Pattern1, Pattern2, Pattern3: Boolean;

begin
  { -- ******************    Position all 4 edges    *************** }

  All4Positioned := False;  Face := 1;
  repeat
    If Face > 4 then begin  { -- Fatal error }
      ErrorInMoves;  MoveError := True;  Exit;  end;
    Fmid := A[14];  Rmid := A[23];  Pmid := A[32];  Lmid := A[41];

    { -- Determine which of the 4 Bottom edges are positioned (= True). }
    BF := (A[17] = Fmid) or (A[47] = Fmid);
    BR := (A[26] = Rmid) or (A[51] = Rmid);
    BP := (A[35] = Pmid) or (A[53] = Pmid);
    BL := (A[44] = Lmid) or (A[49] = Lmid);

    if BF and BR and BP and BL then  { -- All 4 edges are positioned. }
      All4Positioned := True
    else  { -- 0 or 1 egde is positioned.  There are 2 ways to position them.}
      begin
           if BR then AddToSolution ('RC+')
      else if BP then AddToSolution ('RC2')
      else if BL then AddToSolution ('RC-');
      { -- if 1 edge correctly positioned then it is in the BF position. }

      Lmid := A[41];    { -- Cube may have rotated and changed colors. }
      if (BF or BR or BP or BL) and ((A[51] = Lmid) or (A[26] = Lmid)) then
        { -- Permute 3 bottom edges:      BR -> BL -> BP -> BR }
        AddToSolution ('L- R+ F- L+ R- B2 L- R+ F- L+ R- ')

      else  { -- Either 0 edges positioned, or 3 non-positioned need this. }
        { -- Permute 3 bottom edges:      BR -> BP -> BL -> BR }
        AddToSolution ('L- R+ F+ L+ R- B2 L- R+ F+ L+ R- ');
      end;
    Face := Face + 1;
  until All4Positioned;
```

```
  { -- ******************  Orient 4 edges  ****************** }
  { -- Rotate Cube until 1 of 7 patterns appear, or all 4 are oriented. }
  { -- Perform set of moves.  If pattern is BC1 or BC2, cube is oriented. }

  Bmid := A[50];
  { -- Check if all edges are oriented. }
  All4Oriented := (A[47] + A[49] + A[51] + A[53]) = (Bmid+ Bmid+ Bmid+ Bmid);

  if not All4Oriented then begin  { -- Edges will be in 1 of 3 patterns. }
    Pattern1 := (A[47] <> Bmid) and (A[49] <> Bmid) and
                (A[51] <> Bmid) and (A[53] <> Bmid);
    Pattern2 := (A[47] =  Bmid) and (A[53] = Bmid)  or
                (A[49] =  Bmid) and (A[51] = Bmid);
    Pattern3 := (A[47] =  Bmid) and (A[49] = Bmid)  or
                (A[49] =  Bmid) and (A[53] = Bmid)  or
                (A[51] =  Bmid) and (A[53] = Bmid)  or
                (A[47] =  Bmid) and (A[51] = Bmid);

    if Pattern1 then  { -- Permute each edge (swap the colors on each) }
      AddToSolution('L- R+ F2 L+ R- B2 L- R+ F+ L+ R- B2 L- R+ F2 L+ R- B- ')

    else if Pattern2 then begin  { -- Permute edges across from each other. }
      if (A[47] =  Bmid) and (A[53] = Bmid) then  { -- Rotate Cube }
      AddToSolution ('RC+');
      AddToSolution('L- R+ F+ L+ R- B+ L- R+ F+ L+ R- B+ L- R+ F2 L+ R- B+ ');
      AddToSolution('L- R+ F+ L+ R- B+ L- R+ F+ L+ R- B2 ');
    end

    else if Pattern3 then begin  { -- 4 possible orientations, Rotate cube. }
      if (A[47] =  Bmid) and (A[49] = Bmid) then
      AddToSolution ('RC-')
      else if (A[49] =  Bmid) and (A[53] = Bmid) then
      AddToSolution ('RC2')
      else if (A[51] =  Bmid) and (A[53] = Bmid) then
      AddToSolution ('RC+');
      { -- Rotate BP -> BF -> BL -> BP, then orient 3 edges. }
      AddToSolution ('L- R+ F+ L+ R- B- L- R+ F- L+ R- B- L- R+ F2 L+ R- ');
      AddToSolution ('RC+L- R+ F+ L+ R- B2 L- R+ F+ L+ R- ');
    end

    else begin    { -- Fatal error, none of the patterns were detected. }
      ErrorInMoves;  MoveError := True;  Exit;  end;
  end; { -- If not All4Oriented }
end;
```

Name and Description of Procedures in Program

```
program RubeCube;
{ -- Project Name: RubeCube.Pas
  -- Author:       Douglas E. Woolley
  -- Date Started: 8/15/88
  -- Last Update:  9/27/88

  -- This program will accept the appearance of the Rubik's Cube as input,
  -- and output a step-by-step text/graphical solution to restore the puzzle.
}

procedure DisplayBorder ({using} Title: String30; TM: Integer);
{ -- This procedure will display special characters around the perimeter
  -- and centers the Title on the top line; TM=0 is width 80, = 1 width 40. }


procedure DisplayTitlePage;
{ -- This procedure will display program name, author, date, etc. }


procedure DisplayMenu ({giving} var Option: Integer);
{ -- This procedure will display Menu and accept valid option. }


procedure Beep;
{ -- This procedure makes a beep for errors. }


procedure InitValidColors;
{ -- This procedure will assign 6 common colors on the Rubik's Cube
  -- to the captial global variable ValidColor. }


procedure GetValidColors;
{ -- This procedure will accept 6 valid colors to use for the cube. }


procedure InitArrayOfColors;  { -- output is global array of colors. }
{ -- This procedure will set all array items to spaces for color. }


procedure DisplayBox ({using} Side: Integer);
{ -- This procedure displays a box for corresponding side at row,col. }


procedure DisplayColors ({using} Side: Integer);
{ -- This procedure displays the color contents of squares on the Side. }


procedure DisplayTextInstr;
{ -- This procedure will display commands needed for displaying the solution.}
```

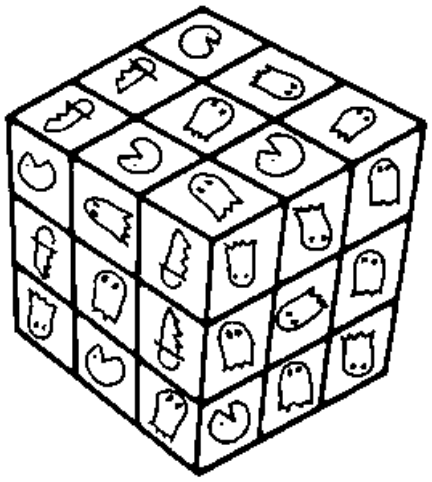
```
procedure DisplayInputInstr;
{ -- This procedure will display commands needed for inputting colors. }


procedure DisplayBoxes;
{ -- This procedure will display the 6 sides of the cube. }


function Match({using} Cor: String3;  A1, A2, A3: Char): {giving} Boolean;
{ -- This function returns True if Cor matches a permutation of A1,A2,A3. }


function InputCorrect: {giving} Boolean;
{ -- This function will output True if 12 unique edges and 8 unique corners
  -- are entered with 6 unique middle squares on the 6 sides. }


procedure GetColors;
{ -- This procedure gets the color of squares on each Side.
  -- Output: AInit[1..54] are assigned;  InputCorrect is True or False. }


procedure GetRCInput;
{ -- This procedure will assign colors to array variables for Rubik's Cube. }


{ -- ************  Computer Coded Solution  **************** }

procedure MoveSide ({given} SideToMove: String1;  NumOfRot: Integer);
{ -- This procedure will turn Side a total of NumOfRot rotations clockwise. }


procedure MakeMove ({using} NextMove: String3);
{ -- This procedure will process one move by swapping the A array
  -- corresponding to the Side being moved. }


procedure Combine ({using} FirstMove: String3);
{ -- This procedure will add the first move to the Solution array, compacting}


procedure AddToSolution ({using} SetOfMoves: String60);
{ -- This procedure will first internally move the cube as directed, then
  -- Add the set of moves to the entire Solution array, with compacting. }


procedure ErrorInMoves;
{ -- This procedure will display an error message about the solution. }


procedure SolveTopEdges ({giving} var MoveError: Boolean);
{ -- This procedure will store the moves in Solution to restore top edges. }
{ -- Strategy: Work on front-top edge then rotate cube to next face. }
```

```
procedure SolveTopCorners ({giving} var MoveError: Boolean);
{ -- This procedure will store the moves in Solution to restore top corners. }
{ -- Strategy: Work on front-right-top corner then rotate cube to next face. }


procedure SolveVerticalEdges ({giving} var MoveError: Boolean);
{ -- This procedure will store the moves in Solution to restore vert. edges. }
{ -- Strategy: Work on front-right edge then rotate cube to next face. }


procedure SolveBottomCorners ({giving} var MoveError: Boolean);
{ -- This procedure will store the moves in Solution to restore bot corners. }
{ -- Strategy: First "position" all 4 corners, then "orient" 4 corners. }


procedure SolveBottomEdges ({giving} var MoveError: Boolean);
{ -- This procedure will store the moves in Solution to restore bot edges. }
{ -- Strategy: First "position" all 4 edges, then "orient" the 4 edges. }


function CubeIsSolved: Boolean;
{ -- This function returns True if each side has 9 squares of the same color.}


procedure GetRCSolution ({using/giving} var MoveError: Boolean);
{ -- This procedure will store all the moves to restore the puzzle in the
  -- array Solution[1..200] of string[3], with Solution[LastSolIndex]
  -- having the last move of the solution. }


procedure DisplayTextSolution ({using} Option1or2: Integer);
{ -- This procedure will display step-by-step the moves to restore the puzzle
  -- in the array Solution[1..200] of string[3], with Solution[LastSolIndex]
  -- having the last move of the solution.
  -- If Option1or2 = 2 then display is text. }


procedure MagnifyLetter (L: Char; {by} M: Integer; {at} X, Y: Integer;
                                                   {using}  Co: Integer);
{ -- This procedure will magnify the letter L by M times at position X, Y.
  -- If the character L is not available (such as a space), then no display. }


procedure DisplayInstr;
{ -- This procedure will display a brief description of each menu option. }


procedure DisplayGraphicsInstr;
{ -- This procedure will display commands needed for displaying the solution.}
{ -- 4 colors available (Graphics and Text):
     0 = Background, 1 = LightCyan, 2 = LightMagenta, 3 = White. }


procedure DisplayGraphics;
{ -- This procedure will display the Graphical cube. }
```

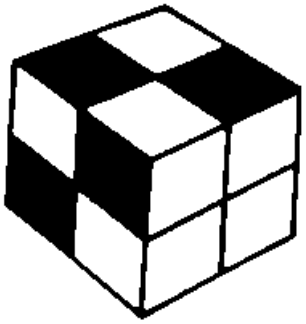
```
procedure DisplayGraphicsColors ({using} Side, Co: Integer);
{ -- This procedure displays the color contents of squares on the Side
     if Co is 1, 2, or 3.  If Co = 0 then the previous colors are erased. }


procedure DisplayGraphicalSolution;
{ -- This procedure will display step-by-step the moves to restore the puzzle
  -- in the array Solution[1..200] of string[3], with Solution[LastSolIndex]
  -- having the last move of the solution.  One side of a graphical cube is
  -- displayed. }


{ -- ****************    Test Program  routine   **************** }

procedure PutColorsOnCube;  { -- output is global array of colors. }
{ -- This procedure will set all array items to Valid colors for a cube. }


procedure GetNumber ({using} Col, Row: Integer; {giving} var Number: Integer);
{ -- This procedure will accept as input a 2 digit number. }


procedure TestProgram ({using} TestOption: Integer);
{ -- This procedure will display statistics for solving random cubes. }


{ -- ***********   Main Flow of Program   ************* }
begin
  DisplayTitlePage;
  InitValidColors;
  repeat
    DisplayMenu (Option);
    case Option of
      1, 2: begin
            GetRCInput;
            if CorrectInput then begin
            GetRCSolution (MoveError);
            if not MoveError then
              if Option = 1 then  { -- Graphical Solution }
                DisplayGraphicalSolution
              else      { -- Option = 2  Text Solution }
                DisplayTextSolution (Option)
            end;
          end;

       3: DisplayInstr;

       4: GetValidColors;

       5: TestProgram (Option);
    end;
  until Option = LastOption;

  ClrScr;
end.
```

Appendix - Variations of the Rubik's Cube

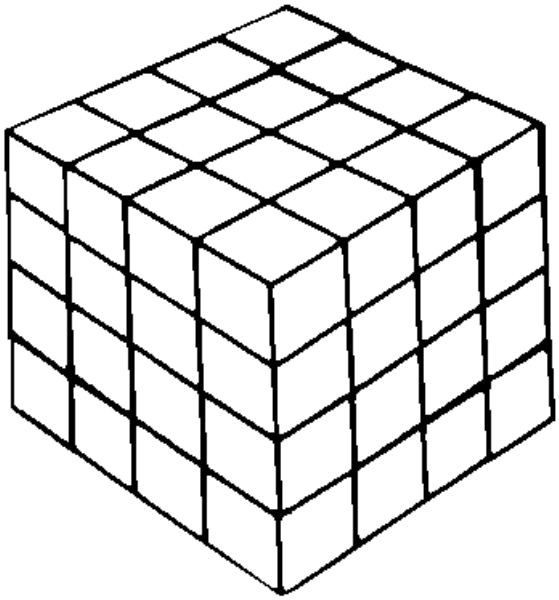
The **Picture Cube**, although mechanically similar to Rubik's Cube, has pictures on the six sides instead of solid colors. One such cube is called the Pac Man Picture Cube which has Pac Men, ghosts, and keys on the square patches. The difference between the Rubik's Cube and this Picture Cube is that the latter's center could be incorrectly oriented. Because of this variation, this Cube is harder to solve than the Rubik's Cube and has 88,580,102,706,155,225,088,000 possible combinations.



Rubik's Pocket Cube is a two by two by two inch cube that has six sides which contain a different color. The sides of this puzzle move independently of each other. Rubik's Pocket Cube has no center piece on any of its sides and has only 3,674,160 different arrangements.



Rubik's Revenge is a four by four by four cube. This is the hardest puzzle on the market because of its incredible number of arrangements: 3.7 x (10 to the 45th power). The Rubik's Cube has approximately 4.3 x (10 to the 19th power). Rubik's Revenge has 8 corner pieces, 24 middle edge pieces, and 24 center pieces.



Bibliography

Alexander, Ron, "A Cube Popular in all Circles", New York Times, July 21, 1981, III, 6:2.

Borders, William, "Best-Selling Author, 13, Thanks Rubik's Cube", New York Times, October 17, 1981, I, 3:5.

Carmichael, Jane, "Figure This One Out", Forbes, p. 34, May 1981.

Dougherty, Phillip H., "Ideal Toy's Son of Rubik Cube", New York Times, July 30, 1981, IV, 19:1.

Dubisch, Roy, Introduction to Abstract Algebra, p. 70, Seattle, Washington, John Wiley & Sons, Inc., 1965.

Editors of Discover, "Robot Cubist", Discover, June 1982.

Editors of Time, "Hot-Selling Hungarian Horror", Time, p. 83, March 23, 1981.

Ewing, John, and Kosniowski, Czes, Puzzle It Out, New York, NY, Cambridge University Press, 1982.

Faludi, Susan C., "CBS Unit Offers $58 Million for Makers of Rubik's Cube", New York Times, April 23, 1982, IV, 3:1.

Hauptfuhrer, Fred, "An Obscure Hungarian Professor Transforms America into a Nation of Cubic Rubes", People, p. 31, May 1981.

Nourse, James G., The Simple Solution to Rubik's Cube, New York, NY, Bantom Books 1981.

Nourse, James G., The Simple Solution to Cubic Puzzles, New York, NY, Bantom Books 1981.

Roman, Mark, "Rubik's Cube: Ideal Toy Takes on the Knock-Offs", New York Times, October 4, 1981, III, 21:1.

Sanger, David E., "Rubik's Rival", NY Times, June 6, 1982, III, 19:4.

Sigmaster, David, Notes on Rubik's Magic Cube, Hillside, NJ, Enslow Publishers, 1981.

Thai, Minh, The Winning Solution to Rubik's Revenge, pp. 6,7,9, Wayne, Pennsylvania, Banbury Books, Inc., 1982.

Warshofsky, Fred, "Rubik's Cube: Madness for Millions", Readers Digest, pp. 138-139, May 1981.

Author's Biography

Douglas E. Woolley is currently a senior at the University of South Florida in Tampa, majoring in Computer Science and minoring in Mathematics. Doug is a member of three national honor societies and is active in intramural sports. He is a teacher and leader in a Christian church and has been a guest speaker at high schools, banquets, churches, and conventions.

PROGRAMMING EXPERIENCE

1984-present
Florida Center for Instructional Computing, Tampa, Fl
Senior Computer Programmer

- Developed and maintained code for Instructional Management System which is designed for prescription learning and providing student and administrative reports. It is presently being used in Federally funded instructural environments.
- Author and official judge of the annual Florida High School Computer Contest items.
- Developed and maintained code for ***The Micro Goes to School*** (IBM and Apple versions). The four diskettes that accompany these books demonstrate the integration of instructional software in existing school curricula.
- Developed and maintained code for the "Computer Contest Database" which stores attributes of contest items and creates contests.
- Guest speaker at annual statewide computer conventions to demonstrate computer programs developed and maintained.

AWARDS AND HONORS

Key note speaker at State Mu Alpha Theta (MAT) convention (1985)
** Ranked number one in Florida in computer science (1984)
** Ranked number one in Florida in mathematics (1984)
Third at national (MAT) convention in mathematics (1984)
Fifth at national (MAT) convention in computer programming (1984)
Captain of first place Florida computer team (1984)
Recognized as one of the top 300 high school scientists in America by Westinghouse Science Talent Search (1984)

PUBLICATIONS

The Micro Goes to School: Instructional Applications of Microcomputer Technology, IBM and Apple Versions (Brooks/Cole Publishing Company)