Teaching Tools for Data Structures and Algorithms

Martin J. Biernat†
AT&T
1200 East Warrenville Road
Naperville, Illinois 60565
martin.j.biernat@att.com

1. ABSTRACT

This paper describes several teaching tools used in an Analysis of Algorithms course. The tools aid in reviewing data structures and explaining various algorithms while actively involving the student with the algorithms. These tools have helped students not only understand and retain the concepts behind the algorithms, but has also raised their level of interest in the class.
Keywords: Algorithms, Analysis of Algorithms, Data Structures

2. INTRODUCTION

My experience in both corporate training and academia indicates that most students enjoy personal involvement with hands-on in-class exercises. With this idea I searched for ways to transform Analysis of Algorithms from simply lecturing about algorithms contained in a program, textbook, or chalkboard computer to an exciting hands-on interactive learning process where all students use games and real-life examples to increase their understanding of basic algorithms.

3. TEACHING TOOLS

The following teaching tools have been divided into general categories of algorithms. These teaching tools are used to aid in the explanation of the various algorithms and not as a replacement for programming, homework assignments, and examinations.

3.1 REVIEWING DATA STRUCTURES

A prerequisite for the Analysis of Algorithms course is that each student be well-versed in data structures. Data structures will play an important role in the analysis of algorithms so the course begins reviewing data structures to insure that all students have a common knowledge base of data structures from which to start. I do this by employing an academically competitive game of Data Structures Jeopardy.®[1] Several 5-member teams are formed(class size approximately 25). A question is asked by the instructor and each team must submit a written response within a given time period. Time periods are predetermined and length of time allowed depends on question difficulty. Each team receives one point for a correct answer, and no points for an incorrect answer. Two questions have "Daily Double" opportunities at which time each team can wager up to their current score on each of these questions. Any team can ask for an explanation of any question or answer during the game. Correct answers to all the Data Structure Jeopardy questions are distributed after the review. To make the game a little more interesting and fun, small prizes are awarded to all students with the prizes distributed on a highest scoring team choses prizes first distribution scheme.

3.2 DIVIDE-AND-CONQUER

The basic strategy for Divide-And-Conquer is given a function with n inputs, divide the input into k distinct set, $1 < k \leq n$, creating k subproblems. Continue to divide until the subproblems can be easily solved. Then determine a method to combine the subsolutions into the solution of the whole.

A tool that works extremely well for teaching sorting and searching techniques involves using a set of variable length pieces of wood. I use nine pieces of wood, each a different length from one inch to nine inches, and each labeled with its' corresponding length[See Figure 1]. I mix up the pieces of wood and the students apply the MERGESORT or QUICKSORT algorithms to the blocks by moving the blocks accordingly to each algorithm.

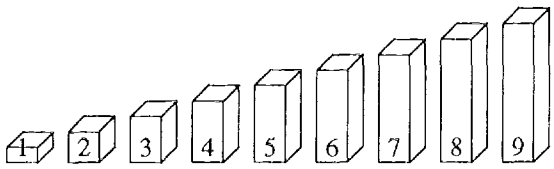


Figure 1. Sorting Blocks

† Adjunct Faculty Member
Department of Computer Science
Illinois Institute of Technology
Rice Campus
201 East Loop Road
Wheaton, Illinois 60187

1. Jeopardy is a registered trademark of the Milton Bradley Company.

3.3 GREEDY METHOD

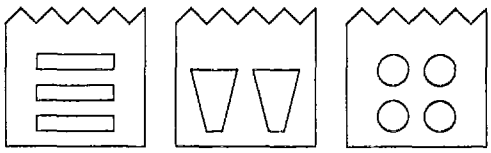
The basic strategy for the GREEDY METHOD is to take a problem and find a solution, normally a subset of the input, that satisfies some predefined constraints. A solution that satisfies the constraints is called a *feasible solution*. A feasible solution that maximizes or minimizes a given *objective function* is called an *optimal solution*.

One way to the students' mind is through his/her stomach and I take full advantage of this for the group of Greedy Method algorithms. I place one pound of M&M's®[2] Brand Chocolate Candies in a bowl. Have one small paper cup per student. Select one student to distribute the M&M's from the bowl using the paper cups. The directions for distribution are: "Make one pass through the class. Each student must have a cup filled with some amount of M&M's and no M&M's can remain in the bowl when you complete the one pass through the class." Students see a form of the Greedy Method in practice immediately if the student distributing the M&M's likes M&M's and fills his/her cup to the brim and then attempts to equally distribute the remaining M&M's. After the M&M's are distributed, I give the student that distributed the M&M's a moment to describe their distribution algorithm in terms of the constraints and objective function and whether they produced a feasible or optimal solution.

Before the students eat their M&M's I use the M&M's as an introduction to an example of the Knapsack problem. Each student separates his/her M&M's by color and assigns a numerical value to each color. Each student then chooses a value for their knapsack(their small paper cup) such that all the M&M's with their associated numerical values cannot be used. I ask the students to fill their knapsacks. Upon completion I discuss with them what "algorithm" they used by having them define their constraints and objective function. After all that work the student then can enjoy their M&M's.

I also teach the knapsack problem by using a real life venue. As a homework assignment, each student must bring to class a paper bag from a fast food restaurant. Students have no idea what this is all about until I show them a form of the knapsack problem by looking at the bottom of the bag. Nearly all fast food restaurants now have at least item counts or illustrated packing pictures for every bag[See Figure 2].

To illustrate PRIM'S and KRUSKAL'S minimum spanning tree algorithms I use a Tinkertoy[3] Building Set where the connector spools are nodes of the tree and the rods are the weighted branches of the trees. I step through the algorithm and use the Tinkertoys to construct the minimum spanning tree using each technique. Not only does the student understand each algorithm but, it easily illustrates the differences between each algorithm. I find students coming up on a class break to run through the algorithms and construct the minimum spanning trees themselves using the Tinkertoys.



ITEM COUNT: 2-4

Figure 2. Bottom of Fast Food Bag

3.4 BACKTRACKING

Backtracking is one of the most fundamental algorithm design techniques for problem searching for a optimal solution or a set of solutions which satisfy some constraints. The solution is expressed as a n-tuple $(x_1, x_2, \ldots, x_n)$ where x_i is some value from a finite set of possible values. A state space tree can be created where a value is chosen for one element of the tuple at each level of the tree.

I use an illustration of the entire state space tree for the 4-queens problem to show how backtracking finds a solution by traversing the state space tree discarding subtrees and moving toward a solution. Although this picture can become large and cumbersome its use is well justified because it allows the students to easily see the size of the state space tree for a given problem and how much of the state space tree is not traversed using backtracking.

I have also created for the 4-Queens problem a large felt chess board with accompanying four felt queen crowns. In real-time as I hand compute the 4-Queens problem the students learn what the program does by seeing how the board appears after each value of the tuple is selected[See Figure 3]. The added value of using the 4-Queens Board is that students can see how a certain board configuration relates to the state space tree.

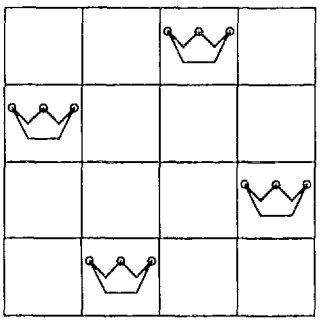


Figure 3. 4-Queens Board

2. M&M's is a registered trademark of Mars Inc.
3. Tinker Toy is a registered trademark of Playskool Inc.

3.5 BRANCH-AND-BOUND

Branch-And-Bound refers to the state space traversal technique which all children of the *E*-node[4] are generated before any other *live nodes*[5] can become the *E*-node.

A real life game that makes an interesting programming problem for Branch-And-Bound and that students can easily access and enjoy playing is the 15-Puzzle Game[See Figure 4].[6]

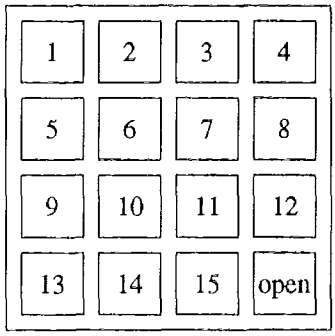


Figure 4. 15-Puzzle Game

A more challenging game to play and to program a Branch-And-Bound Solution is Top Spin™[See Figure 5].[7] Here the student is to arrange the balls in order by moving the balls forward or backward and changing the order of the four balls in the "round house".

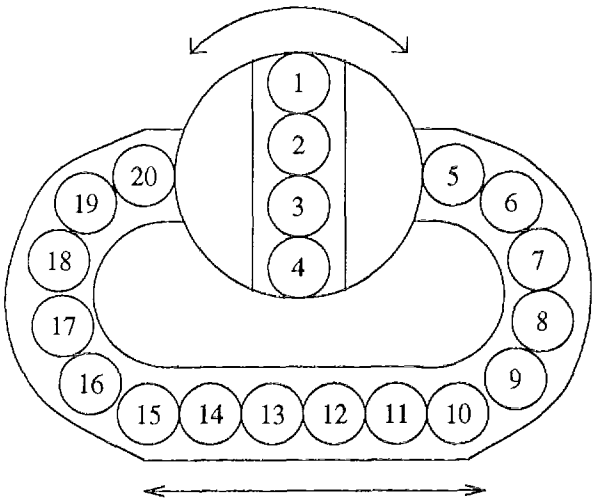


Figure 5. Top Spin Game

Another challenging programming assignment for Branch-And-Bound is to have students write a maze traversal program for the Amazing Micro-Mouse Contest[See Figure 6].[8] The Objective is to reach "home", usually located in the center of the board, in the least number of moves. One Mouse move is restricted to moving forward one square in the direction the mouse is facing or turning in place left or right 90 degrees.

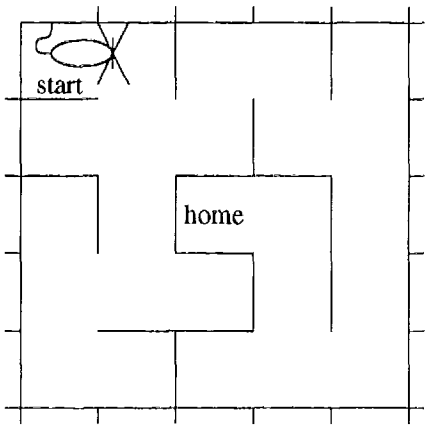


Figure 6. Amazing Micro-Mouse Maze

4. OBSERVATIONS

There are three distinct observations I have made when using these teaching tools.

1. Students comprehend the algorithms faster by playing with the teaching tool. With the hands-on approach the student can figure out what an algorithm is doing by physically manipulating the teaching tool. Many students will spend non-class time working with the teaching aid. Some students have even invented their own teaching aids.
2. Students remember the algorithms. Two years after the course, students can still tell me how a particular algorithm works. The students credit the use of the teaching tools to help them visualize an algorithm's behavior.
3. Students are excited to come to class to see what's in store. Students have told me they are eager to do the preclass reading assignment so they can clarify their understanding of the algorithms when the teaching tool is introduced. When these tools are employed in class it is obvious that most students have indeed read the assigned material prior to attending class.

4. A *E*-node or Expanding-node is a live node whose children are currently being generated.
5. A *live node* is a node whose children have not all been generated.
6. The 15-Puzzle Game was invented in 1878 by Sam Loyd.
7. Top Spin is a trademark of Binary Arts Corporation.
8. The Amazing Micro-Mouse Contest originally created by Donald Christiansen in May 1977.

5. CONCLUSION

This paper has described various teaching tools for an Analysis of Algorithms course. The teaching tools aid the students in learning, understanding, and remembering the algorithms. Secondary benefits derived from using these teaching tools is an increased level of interest by the student and an eagerness to prepare for classes that are educational as well as fun. These teaching tools can also be created at minimal expense to the instructor or institution.

6. REFERENCES

Baase, Sara. *Computer Algorithms Introduction to Design and Analysis.* Reading, MA: Addison-Wesley Publishing Company, 2nd edition, 1988.

Cormen, Thomas H. and Leiserson, Charles E. and Rivest, Ronald L. *Introduction to Algorithms.* Cambridge, MA: MIT Press, 1992.

ACM/IEEE-CS Joint Curriculum Task Force. *Computing Curricula 1991.* New York, NY: ACM Press, 1991.

Manber, Udi. *INTRODUCTION TO ALGORITHMS A creative Approach.* Reading, MA: Addison-Wesley Publishing Company, 1989.

Moret, B.M.E. and Shapiro, H.D. *Algorithms from P to NP Volume 1 Design & Efficiency.* Redwood, CA: The Benjamin/Cummings Publishing Company, Inc., 1991.

Silberman, Mel assisted by Auerbach, Carol. *ACTIVE TRAINING - A Handbook of Techniques, Designs, Case Examples, and Tips.* Lexington, KY: Lexington Books, 1990.

Smith, Jeffrey D. *Design and Analysis of Algorithms.* Boston, MA: PWS-KENT Publishing Company, 1989.

**********OBJECT-ORIENTED Continued From Page 8**********

```
matrix::matrix(int height, int width)
{
  mbody = new vector [height];
  for (int i=0; i<height; i++)
       mbody[i]= * new vector(width);
  for (i=0; i<height; i++)
       mbody[i][i]=1.0;
  m=height;
  n=width;
}

matrix::~matrix()
{
  delete [] mbody;
}

matrix& matrix::operator+(matrix& b)
{
  matrix *temp= new matrix(m,n);
  for (int i=0; i<m; i++)
       temp->mbody[i]=mbody[i]+b[i];
  return *temp;
}

matrix& matrix::operator*(matrix& b)
{
  matrix *temp= new matrix(m,b.n);
  for (int i=0; i<m; i++)
    for (int j=0; j<b.n; j++)
      temp->mbody[i][j]=mbody[i]*b.tr(j);
  return *temp;
}

matrix& matrix::operator=(matrix& b)
{
  mbody=new vector [b.m];
  m=b.m;  n=b.n;
  for (int i=0; i<m; i++) mbody[i]=b[i];
  return *this;
}

vector & matrix::operator[](int i)
{
  return(mbody[i]);
}

vector matrix::tr(int i)
{
  vector temp(m);
  for (int j=0; j<m; j++)
       temp[j]=mbody[j][i];
  return temp;
}

int matrix::sizeN() { return n; }
int matrix::sizeM() { return m; }

matrix& matrix::t()
{
  matrix *temp= new matrix(n,m);
  for (int i=0; i<n; i++)
       temp->mbody[i]=this->tr(i);
  return *temp;
}
-------- end of file   matrix.cc --------
```