Algorithm Visualization Automation

Michael Galliers

Eastern Kentucky University - Department of Computer Science

Introduction

Computer algorithms are the basis for how computers solve problems. Algorithms describe the series of steps a program must take to complete a certain task. This includes checking conditions and performing actions based on those conditions. Having the ability to visualize how algorithms operate while designing them can be extremely valuable. Automation of the visualization process can be even more useful in development, quickly showing ways in which algorithms could be improved to increase efficiency.

Sorting Algorithms

Sorting is a heavily studied topic in computer science. Many types of sorting algorithms have been developed and each has its advantages and disadvantages. ***Bubble Sort*** is one of the most basic. It works by comparing the first 2 elements in a list and swapping them if the first is greater than the second. This repeats with the second and third elements and so on until the greatest element is *bubbled* to the last position. The process is repeated to *bubble* the second greatest element to behind the greatest, and so on until the list is sorted.

Decision Trees

In algorithm analysis, a **pruned decision tree** is a tree that describes all possible execution paths a program can take, depending on the input, with any contradictory paths *pruned*. For sorting algorithms, this tree is **valid** if there is a path from the root node to a leaf node that sorts any permutation of an inputted list [1]. A **pruned-valid decision tree** can be interpreted as the different execution paths a program can take and the efficiency of each path. The fewer nodes in a path from the root to a leaf node, the less comparisons performed, and generally the more efficient the algorithm.

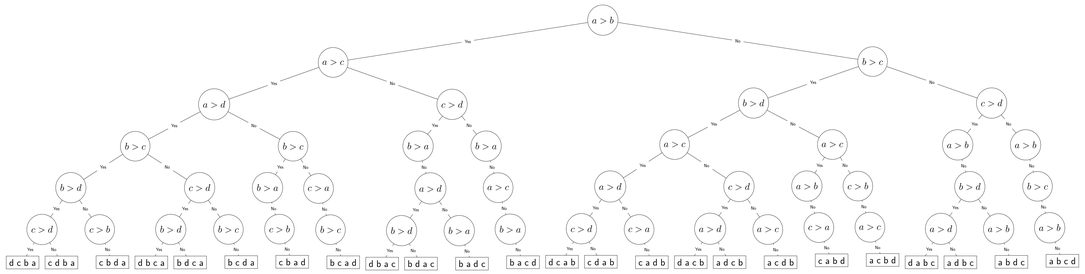

Figure 1: Bubble Sort on [a, b, c, d] - an inefficient sorting algorithm

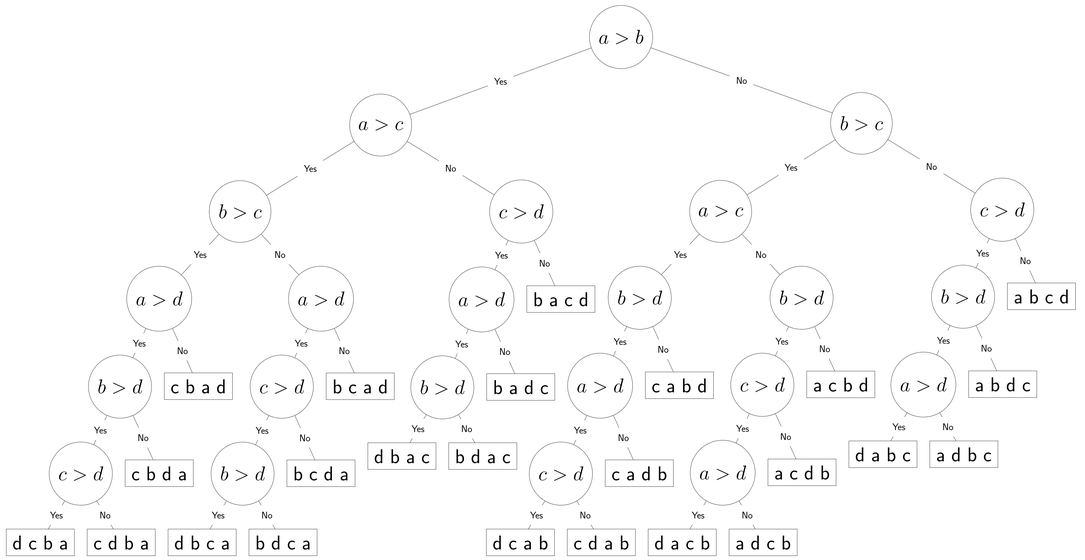

Figure 2: Insertion Sort on [a, b, c, d] - a more efficient sorting algorithm

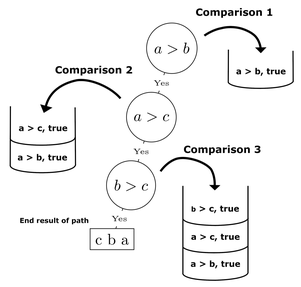

Figure 3: Generator Step 1 - Initial Traversal

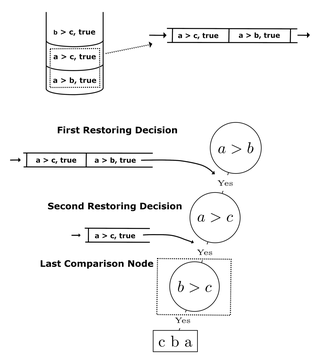

Figure 4: Generator Step 2 - Restoration Process

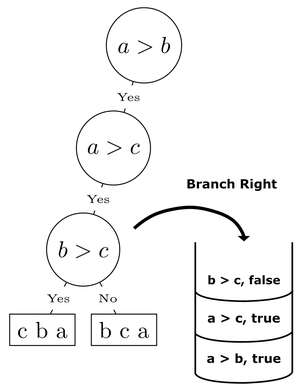

Figure 5: Generator Step 3 - Branching Right

Decision Tree Generator

To aid in development, I designed an automatic analysis code library, the ***Decision Tree Generator***. This library can take a modified version of any sorting algorithm and generate a pruned-valid decision tree for some arbitrary input variables. To build the tree, the generator must run the sorting algorithm through various situations, both observing and controlling its operation.

Generator Functionality

Below is a description of a key functionality of the generator: state restoration.

1. For each comparison of records, the generator takes the comparison and decision made and pushes them to the state stack. In this situation, the generator's decision is always *true*. Figure 3
2. Once the algorithm finishes running through, the generator must restore its execution state to the last node where it chose *true*, so it can then analyze the outcome of choosing *false*. It pulls prior decisions out of the state stack and places them in a restoration queue. With each comparison made, the next decision is dequeued and used until empty. Figure 4
3. The generator then chooses the decision *false* at that node, causing execution to branch right. The entire process repeats until the whole tree is traversed. Figure 5

Conclusion

Visual analysis is key to designing efficient algorithms. The ***Decision Tree Generator*** is just the beginning of visual analysis programs that could be implemented to analyze other algorithms.

References

[1] Richard E. Neapolitan.
Foundations of Algorithms.
Jones & Bartlett Learning, 5th edition, 2015.