CSCI338301

# Visualization of Problems Solved by Dynamic Programming and Traditional Recursion

Jay Agrawal, Lurein Perera, Eric Wang, and Brian Ward

## Overview

Our goal was to develop a visualization to show the differences between traditional recursion and dynamic programming algorithms to solve the same problems. Our project does this by taking four popular problems that can be implemented with both approaches and displaying the differences in structure and efficiency.

We created a visualization tool to help display the fundamental differences between the traditionally recursive approach and the dynamic programming approach using four common problems. Our target audience for this visualization tool is current Computer Science II students as they would understand core concepts such as time complexity and recursion, and they could gain a visual understanding of some of the benefits that dynamic programming could bring. The primary differences that we want to display is the different in execution style. We also decided to use Fibonacci, binomial coefficients, the longest increasing subsequence problem, and edit distance as our four problems due to the fact they could all offer slightly different styles of visualizations.

## Problem Statement

Dynamic Programming is a powerful tool that could be applied to many problems with recursive solutions to yield dramatic increases in efficiency. However, it is often not introduced until high level courses and is easy to misunderstand how or why they are improved solutions to these problems. We were interested in showing these differences between the dynamic programming approach and a more traditional recursion through an educational visualization of several problems that can be solved both ways. For our project, we selected four problems of varying popularity and complexity. They are calculating the nth term of the Fibonnaci sequence, calculating the Binomial Coefficient of n and k, finding the length of the Longest Increasing Subsequence of a list, and finding the Edit Distance or alignment cost between two strings.

We have created a tool that shows and explains both forms of the algorithm to solve each chosen problem, and allows users to enter their desired inputs and watch as

they are solved by each approach. This includes visuals such as a recursion tree for recursion, or the filling of the table for the Dynamic Programming approach.

We hope that a few well-explained and visualized examples, along with the infinite potential for self-learning granted by an interactive tool, will help many more people appreciate this important problem-solving technique. The tool also includes brief descriptions of the problems and links to online resources to learn more.

Finally, we have created and used tools in a way that allows for extensibility if others would wish to add more problems to our visualization. Reasonable extensions to our program should be very easy because of the use of our own `dpviz.py` library and dynamic coding practices.

## Documentation

### Installed Libraries

Our program used many libraries to achieve its goal. Many are shipped with Python 3 installations. These include:

- `tkinter`, Python's default GUI library
- `inspect`, Python's library for getting information about a function from its signature
- `time`, Python's library for getting current times and timing function calls
- `threading`, Python's library for multithreading, used for our display loops
- `webbrowser`, Python's library which allows programs to open web links in the system web browser.
- `sys`, Python's library for setting system variables and program traces
- `functools`, Python's library with various utilities for function wrapping
- `math`, Python's default library containing many mathematical functions
- `io`, Python's library for handling system in/out operations

Our program also used several common libraries which must be installed by the user using the `pip` command. These include:

- `numpy`, a popular package for scientific computing and numerical manipulation.
- `pygments`, a package for advanced syntax highlighting
- `pillow`, a Python 3 fork of the popular Python Imaging Library (PIL), used to support more image types than TkInter allows by default.
- `graphviz`, a Python wrapper for GraphViz.

**Auxiliarily Files**

Our project contains two files not primarily written by our group. They are used as libraries for the main visualizer.

1. `zoom_advanced3.py` is a recipe created by GitHub user foobar167 for creating zoomable images in TkInter. It is used by our project for display of recursive trees, with only aesthetic edits done by our group.
2. `rcviz.py` is a module which provides a wrapper which can be placed around a function to later visualize it's call graph, with special attention payed to recursive functions. Originally created by GitHub user carlsborg, it required many modifications in several catergories.
    a. Dependency. RCViz uses a free program called GraphViz to actually render its images. By default, RCViz uses a Python wrapper on GraphViz called pygraphviz. However, this dependency is very outdated and incompatible with modern 64-bit installations of Python or GraphViz. As a result, we needed to replace this with the more modern Python wrapper, also called graphviz.
    b. Functionality. RCViz was designed to render out images to a file. We were able to modify it using our newer dependency to instead return an encoded string of the image, instead of rendering it to a file. This improved functionality and meant the program would not spam the directory it was run in. RCViz also did not properly wrap functions, leading to issues when we tried to dynamically grab the source code of a function, so we fixed this.

c. Aesthetic. RCViz has some odd defaults for appearance of the final graph. Some significant tweaking was done for purposes of fitting in the rest of our program

**Our Library**

We knew it would be necessary to create our own module for the viewing and analysis of Dynamic Programming algorithms. We wanted to make this as general as possible, so we created our own module separate from the main program.

`dpviz.py` is a module which contains a function-wrapper object, also called `dpviz`. Annotating a function with `@dpviz` marks it for review by our module, which will do the following on each step of the computation of that function: If the function has a field called `dp_table`, it will save a copy of that object to a function attribute `.tables`. This grants access to the data structure used by a dynamic programming algorithm after it has completed computation, and has one for each step of the computation.

For our project, the data structures in question were all array-like, meaning we could add the extra feature of eliminating steps that do not change the arrays in question. This was particularly useful later when we sought to animate the table, as it led to far fewer frames being necessary. This feature can be disabled if a different data structure is needed by annotating the function with `@dpviz(arraylike=False)`.

A standard use of this module is as follows:

1. Write a dynamic programming function, with the memoization data structure used named `dp_table`
2. Call the function with your desired arguments
3. Access the function's `.tables` attribute to view the stored copies of the data structure from the computation

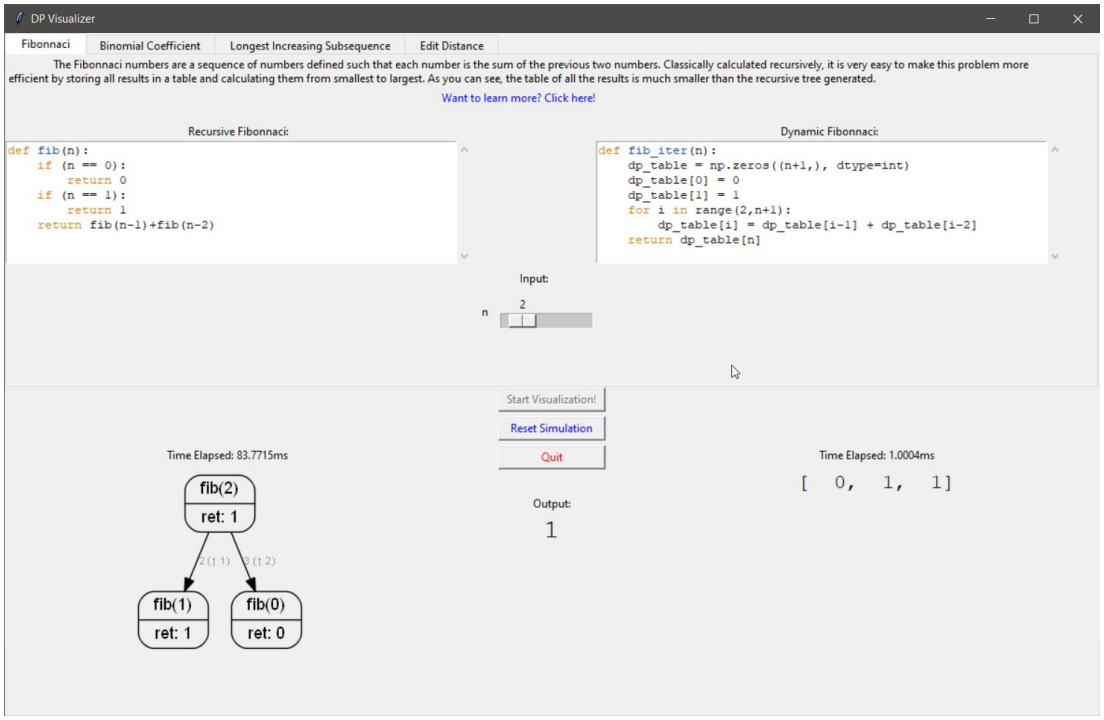
**Problem Files**

We created a file for each of our four problems to house both a recursive version and dynamic programming version of the problem. Each of these was respectively annotated with the @rcviz or @dpviz wrapper.

1. `fib.py` contains fairly standard versions of code for calculating the nth Fibonacci number. NumPy is used to create an empty array, but other than that the code is all standard Python and fits the standard algorithmic approach
2. `binom.py` is very similar, also using NumPy but otherwise being rather standard approaches to the Binomial Coefficient calculation.
3. `lis.py` contains not two but three functions, as the Longest Increasing Subsequence problem is calculated through the use of a more general helper function called LIS_Smaller. The dynamic programming algorithm implemented is the one which uses a 2-dimensional array, as we believe this is the easiest to understand. Adapting this algorithm also meant adjusting it to 0-indexed arrays found in Python, so the details of the for loop and body are slightly adjusted in obvious ways.
4. `edit_distance.py` contains the standard algorithms for Edit Distance/String Alignment, implemented in Python. Very little was needed to be changed from standard pseudocode for these algorithms, besides, again, the use of NumPy and Python syntactic sugar for string prefixes.

**Visualizer Application**

All of the above code was then used in the payoff file which housed the final application our group created.  A screenshot of the final app running follows




`VisualizerApp.py` is ultimately a fairly standard, if large, Python/Tk application. We primarily used the grid placement manager to give us the most control. What follows is a global variable and method breakdown for the **major** features of the program:

1. The `problems`, `functions`, `descriptions`, and `links` objects. These global objects hold the information for each of the four problems we are tackling. They hold, respectively, the names (list), function names and argument types (dictionary), problem descriptions (dictionary), and read-more links (dictionary) for each problem. Our application is intelligent enough that adding a problem which is similar to one of our four should be as easy as writing the code for it and adding it to these four objects.
2. The `Application.create_tabs` function. This function populates the main portion of the application using a TTk Notebook object to create a tab for each problem. It is responsible for using the information from the above objects to dynamically create each tab and populate it with the descriptions, links, code, and input controls.

3. The `Application.create_viz` function. This populates the lower half of the application by creating the controls for the visualizer that are problem-independent: the Start, Reset, and Quit buttons, and the display areas for the outputs and visuals.
4. The `Application.begin_viz` function. This function does a lot of the real work of the program. Called when the begin button is pressed, it has the job of processing the inputs from the current tab and calling both the recursive and dynamic programming algorithms for the current problem. It then handles the visualization through RCViz and the `loop_tables` function covered later. It creates a zoomable image for the recursion tree and a seperate display thread for the dynamic programming table display. It also times the running of both algorithms and displays those times to the user.
5. The `Application.reset_viz` function. This function essentially undoes the above. Called when reset is pressed, it clears many of the fields such as the output frames and visuals, and it sets a global variable to let the display thread know it should terminate. It also clears the RCViz callgraph so the next visualization is started from a clean slate.
6. The `Application.loop_tables` function. This function uses the tables gathered by our DPViz module and loops them in an animation on the application window. It takes extra care to speed up if the number of frames to animate is very large, as to make sure the program feels consistent in its speeds. It also holds a bit longer on the final table than on the intermediary steps.

These major features are combined with a few small helper functions and some TkInter boilerplate to create the application.

**Full Code**

Our code is available on GitHub here**.**

**Milestones**

We are proud to say that we were able to achieve the milestones that we outlined in our initial report. In our initial report, we explained we wanted to develop a tool for the user that would explain both forms of the algorithm, select their own chosen inputs, and allow them to watch the process of solving the problem. We explained the forms of the algorithms through not only showing the code and having descriptions of each algorithm, but also we have the overall structures that each problem is solved with. We also have users choose their inputs to improve the interactive aspect of the tool. Furthermore, we display an animated version of a DP table and a complete recursive tree with each run which can be analyzed and understood. We have done this for our four selected problems, and have built an easily extensible framework for adding more.

## Learning Outcomes

This project definitely gave us much technical insight into real world situations such as working on a larger project with others. Other than finding out how to use collaboration tools such as Github, we gained experience for many issues that we could run into in the workplace. Differences in operating systems, packages, and schedules were all roadblocks that we needed to adapt and work around in order to complete the project. Aside from that, many of us also got the opportunity to have hands-on experiences with Python toolkits that we may not have gotten around to using. Tkinter, an interface some of us had never even heard of before, proved to be pivotal in our visualization and a nice option that some of us can apply again for future visualization projects.

## Statement of Code Release

We would be happy to share our code/report with other students, we believe it could be a good source of inspiration for future visualization projects. Furthermore, our tool is meant primarily for CSII students so we believe it would be very helpful if they use our tool to reinforce their understanding of recursion but also build some awareness for dynamic programming. For future Algorithms classes we believe it would

be helpful for others to get an idea of a feasible project to work towards for their end result. Our code is also well documented and commented throughout which makes it a very user friendly for future groups to follow. Aside from this, we also have the possibility of receiving feedback from future groups as to what we can improve or add to our code/report.

## Project Feedback

We believe that the final project was well structured overall with only a few minor comments to supplement it. In terms of help, we thought that although the examples were a good measure of what to expect, most were visualizations which influenced the class to pursue more visualization-oriented final presentations. A solution could be to show examples of research projects or empirical studies to influence more variety come final presentations next time. Another suggestion would be for groups to meet up and discuss the Initial Report. This would give both the group and professor time to discuss feasibility, suggestions, issues, etc. This would also help with another issue we found, which was the initial reports were due so early that we did not know exactly what we could expect to learn by the time we showed them off. That shows in the fact that many groups, including ours, did projects on topics from the first section of the course. Although we didn't do a poster session, we're glad that there wasn't as it wouldn't have been the most feasible way to show off certain projects such as ours. Our project was designed to be shown off in real time due to the various inputs and problems to be represented. A poster session would negate the work that we put towards implementing different problems in our code and force a more explanation based presentation.