Gabriel Dos Santos

Advisor: James Glenn

CPSC 490 Senior Project

01 February 2024

Project Proposal

Background:

Currently, there are a limited number of tools that easily enable students, teachers, and programmers to clearly visualize their data structures beyond a simple in-line print statement. One notable tool is jGRASP which was “created specifically to provide automatic generation of software visualizations to improve the comprehensibility of software” in Java environments. This tool, however, is outdated and not widely used in the computer science community. For other languages, there are even fewer tools available to accomplish this task.

Motivation:

From my experience as a Undergraduate Learning Assistant in the Computer Science department’s Data Structures course, I have come to notice that many students have a mismatch between what their code is doing and what they *think* their code is doing. This results in hours of frustration due to small bugs like updating the wrong reference pointer. By having more tools to accomplish this task of visualizing data structures in one’s program, software developers can more easily bridge this gap. As previously mentioned, the resources in this domain are limited and could definitely be improved and cover more languages.

Project Description:

This project will be a powerful tool designed for both debugging and educational purposes, offering an intuitive and interactive way to visualize various data structures. The project primarily focuses on graph-based structures such as trees, but also extends its functionality to visualize arrays, stacks, and queues in a simplified manner.

Key Features:

Graph Data Structure Visualization:

- Proficient in visualizing complex graph data structures, providing a clear representation of relationships, nodes, and edges.

Real-time Updates:

- Allows users to dynamically update the visual representation of data structures, providing a step-by-step view of changes.
- Facilitates a better understanding of how the data structure evolves during program execution.

Comparison and Backtracking:

- Enables users to compare different states of the data structure, supporting a back-and-forth navigation for effective debugging.

Inspired by jGRASP:

- Draws inspiration from the functionality of jGRASP, extending and enhancing it to support Python.
- Aims to provide a more flexible and user-friendly experience:
    - No need to specify the datatype of the structure (tree, stack, etc.)

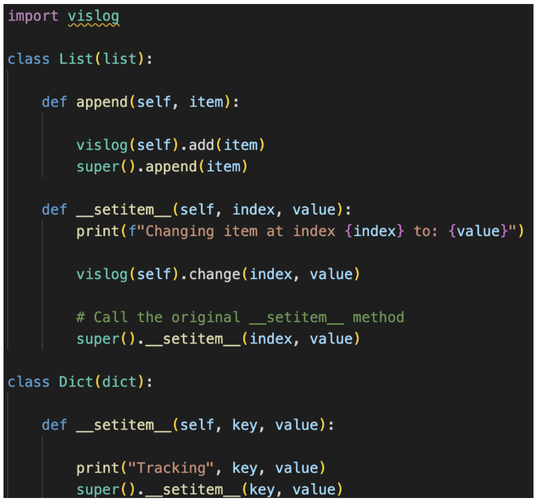
- Users control in-line how much they want logged/visualized.

Utilization of Graphviz and Matplotlib:

- Leverages powerful visualization tools such as Graphviz and Matplotlib to create visually appealing and informative representations.
- Enhances the overall aesthetic and comprehensibility of the visualized data structures compared to terminal debugging.

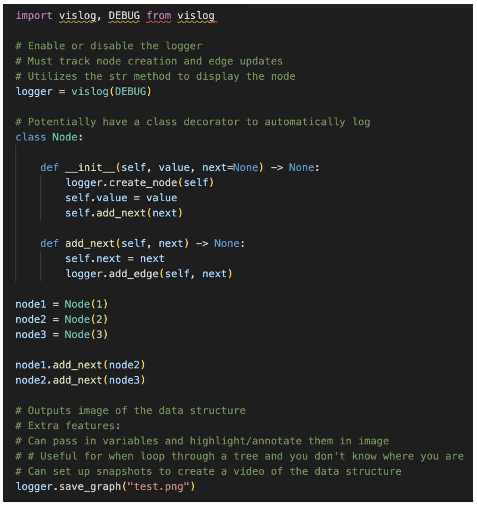
Stretch Goals:

- Small scale user tests: determine positive impacts and acquire feedback.
- Publish package: format and upload library to pypi.



```
import vislog

class List(list):

    def append(self, item):

        vislog(self).add(item)
        super().append(item)

    def __setitem__(self, index, value):
        print(f"Changing item at index {index} to: {value}")

        vislog(self).change(index, value)

        # Call the original __setitem__ method
        super().__setitem__(index, value)

class Dict(dict):

    def __setitem__(self, key, value):

        print("Tracking", key, value)
        super().__setitem__(key, value)
```

Potential way of adding logging to default classes in Python.



```
import vislog, DEBUG from vislog

# Enable or disable the logger
# Must track node creation and edge updates
# Utilizes the str method to display the node
logger = vislog(DEBUG)

# Potentially have a class decorator to automatically log
class Node:

    def __init__(self, value, next=None) -> None:
        logger.create_node(self)
        self.value = value
        self.add_next(next)

    def add_next(self, next) -> None:
        self.next = next
        logger.add_edge(self, next)

node1 = Node(1)
node2 = Node(2)
node3 = Node(3)

node1.add_next(node2)
node2.add_next(node3)

# Outputs image of the data structure
# Extra features:
# Can pass in variables and highlight/annotate them in image
# # Useful for when loop through a tree and you don't know where you are
# Can set up snapshots to create a video of the data structure
logger.save_graph("test.png")
```

Potential for users to configure their custom classes to be tracked.

Deliverables:

1. Project Proposal
    a. Meet with Advisor and agree on scope of project
2. Implement basic library tools for visualization of custom classes

    a. Utilize decorators, observers, etc. to monitor state
    b. Create an interface for users to log updates

3. Extend to built-in datatypes (lists & sets)
    a. Override basic operators to include logging
4. Time permitting, conduct tests on students
    a. Acquire training from Yale
    b. Build user study & analyze results
5. Final Report, Final Poster, and Codebase

Timeline:

Week	Date	Deliverable
1	Jan 19	Work on proposal
2	Jan 22	Study and explore tools: jGrasp, graphviz, networkx, matplotlib etc.
	Jan 29	Share first draft of proposal with advisor
3	Jan 31	Meet with advisor to finalize the project proposal
	Feb 2	**Submit project proposal** to CPSC 490
4	Feb 5	Begin work on implementing data structure visualization with graphviz
	Feb 9	**Project Proposal presentation**
5	Feb 16	Continue working on the visualization library with graphviz
6	Feb 23	Research how to mock builtin classes and their dunder methods
7	Mar 1	**Midterm Progress Check-in**
8	Mar 8	Complete minimum viable product
9	Mar 29	Implement addt'l features such as backtracking, visual customization, etc.
10	Apr 5	Time and resource permitting, conduct small scale user tests
11	Apr 12	Polish and clean the codebase. Draft up the final report

12	Apr 19	**Final Report due to Advisor**
13	Apr 22	**Submit Final Poster**
	Apr 26	**Poster Presentation**
14	May 2	**Final Report due to CPSC 490**
	May 3	Publish library to pypi

Relevant Resources:

- jGRASP documentation, video example
- Graphviz documentation
- Matplotlib documentation
- Pypi documentation