Use Cases For Algorithm Visualization

By group 10

Algorithm Visualization - Group 10

Group Members: Joshua Swaida, Jason Tralongo, Aidan Shepston, Harleigh Kyle Chua, Bach Luu

Manager: Yona Voss-Andreae

Project Idea:

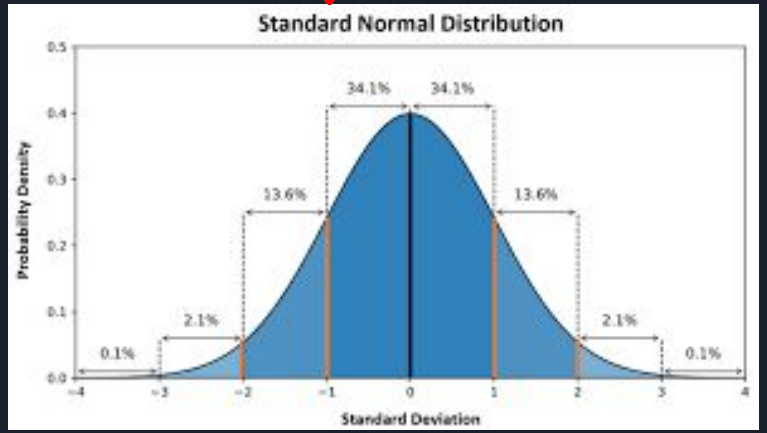
- Purpose: Visualize search algorithms (BFS, DFS) on node graphs and sorting algorithms (quicksort) on an array representation
- Goal: Enhance User understanding and learning of algorithms
- Features:
    - Modify or create custom graphs/arrays
    - Interactive web app interface

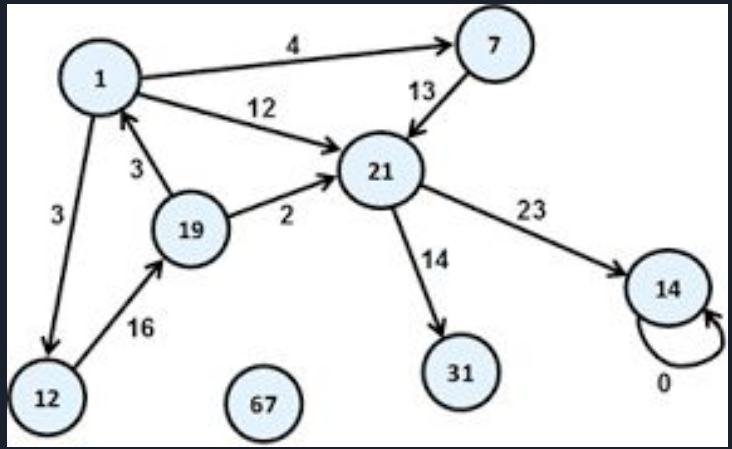
Creating Graph

A graph is a visual representation of data showing relationships between variables. It consists of points (nodes) connected by lines (edges).

Goal	User creates graph with nodes and edges
Primary Actor	Student/User
Precondition	App is on the main screen
Success End Condition	Graph is created and ready for an algorithm to be executed
Failure end condition	Graph cannot be created (nodes do not appear after shift click or edges do not appear after dragging)
Secondary actors	N/A
Trigger	The student has selected graph from the graph/sorting toggle switch and has chosen between directed or undirected

Type of Graph





Creating Graph Scenarios

Main Success Scenario	1. Student opens app 2. Student selects graph from toggle switch 3. Student selects directed or undirected from another toggle switch 4. Student shift-clicks to create nodes 5. Student clicks and drags between nodes to add edges 6. Graph is created
Variations (Error Scenarios)	3a. The student attempts to place a node on top of another node (or close enough to cause overlapping). Visual feedback guides the user to indicate that a node would be overlapping. If the node was placed overlapping another node then the second node snaps into place outside of the existing node to avoid overlapping.
Variations (Alternative Scenarios)	- Graph can have directed or undirected edges - Graph can have weighted edges or unweighted edges

Search Algorithm Visualization

This app will allow the user to run search algorithms on a graph. After creating a graph, the user will be able to select a starting node and a traversal algorithm (BFS or DFS). The program will visualize this traversal.
For the visualization, the current node being search will turn green (nodes will be gray by default).

Goal	Run Algorithm on Graph
Primary Actor	Student/User
Precondition	Student has built graph
Success End Condition	Visualize the graph and execute the algorithm correctly
Failure end condition	Algorithm fails to run properly on graph
Secondary actors	N/A
Trigger	User selects run button

Search Algorithm Visualization Scenarios

Main Success Scenario	1. Student opens app 2. Student places nodes and edges 3. Student selects an algorithm from the dropdown box 4. Student selects a starting node 5. Student clicks the run button 6. Algorithm runs successfully on graph with a visualization (node is grey by default, green while being visited and red after completely explored, this updates as the algorithm continues. Edges are colored as they are used)
Variations (Error Scenarios)	4a. The student places zero nodes. The student clicks the run button. The run button does nothing. Error message displays 4b. The student doesn't select a node for the algorithm to start with. The run button does nothing. Error message displays.
Variations (Alternative Scenarios)	- BFS, DFS - Directed, Undirected - For weighted graphs: Dijkstra's Algorithm, MST

Sorting Algorithm Visualization

A sorting algorithm visualization is a graphical representation that demonstrates how a sorting algorithm organizes a set of data. It typically uses visual elements like bars or dots to represent data points, and as the algorithm progresses, these elements move and change color to illustrate the sorting process.
Note: arrays can only contain numbers (no strings, objects, etc)

Goal	Users can visualize a chosen sorting algorithm on an array
Primary Actor	Student/User
Precondition	Student inputs an array or chooses a randomized array and chooses a sorting algorithm from a grouping of “Divide-and-Conquer” or “Other”
Success end condition	Sorting algorithm is correctly visualized and the array is sorted
Failure end condition	Sorting algorithm is incorrectly visualized and/or the array is not sorted
Secondary actors	N/A
Trigger	User selects run button

Sorting Algorithm Visualization Scenarios

Main Success Scenario	1. Student opens app 2. Student inputs array 3. Student selects an algorithm from the dropdown box 4. Student clicks the run button 5. Algorithm runs successfully on the array and array is sorted with a visualization
Variations (Error Scenarios)	4a. The Input array is empty. The student clicks the run button. Nothing happens.
Variations (Alternative Scenarios)	Bubble Sort, Quick Sort, Merge Sort, Selection Sort, Insertion Sort, Size variations

Array Searching Algorithm Visualization

Our program will provide a visualization of common array search algorithms, like linear search and binary search.
Note: arrays can only contain numbers (no strings, objects, etc)

Goal	Users can visualize a chosen array search algorithm
Primary Actor	Student/User
Precondition	Student inputs an array or chooses a randomized array, enters the target value, and chooses search algorithm
Success end condition	Search algorithm is correctly visualized
Failure end condition	Search algorithm is incorrectly visualized
Secondary actors	N/A
Trigger	User selects run button

Array Searching Algorithm Visualization

Main Success Scenario	1. Student opens app 2. Student inputs array 3. Student selects an search algorithm from the dropdown box 4. Student enters the target value 5. Student clicks the run button 6. Algorithm runs successfully on the array and target value is either found or not in the array
Variations (Error Scenarios)	4a. The Input array is empty. The student clicks the run button. Nothing happens. 5a. Target value is not inputted. The student clicks the run button. Nothing happens.
Variations (Alternative Scenarios)	Linear Search, Binary Search, Size variations

Thank you for listening!