Code Implementation Guideline for Algorithm Teams

URL Format

Examples:

- With parameter input: http://localhost:3000/?alg=heapSort&mode=sort&list=1,2,3,41
- Without parameter input: http://localhost:3000/?alg=heapSort&mode=sort

Parameter input names:

list, value, xyCoords, edgeWeights, size, start, end, string, pattern, union, heuristic, min, max

Algorithms URL Table

Algorithm	mode	alg	Parameter Input Name	Example URL	Notes
Heapsort	sort	heapSort	list	`http://localhost:3000/?alg=heapSort&mode=sort&list=1,3,5,2,8`	Cannot accept negative nodes
Quicksort	sort	quickSort	list	`http://localhost:3000/?alg=quickSort&mode=sort&list=1,3,5,2,8`	Cannot accept negative nodes
Quicksort M3	sort	quickSortM3	list	`http://localhost:3000/?alg=quickSortM3&mode=sort&list=1,3,5,2,8`	Cannot accept negative nodes
Mergesort	sort	`msort_arr_td`	list	`http://localhost:3000/?alg=msort_arr_td&mode=sort&list=1,3,5,2,8`	Cannot accept negative nodes
Mergesort (list)	sort	`msort_lista_td`	list	`http://localhost:3000/?alg=msort_lista_td&mode=sort&list=1,3,5,2,8`	Cannot accept negative nodes
Binary Search Tree	search insertion	`binarySearchTree`	list value	`http://localhost:3000/?alg=binarySearchTree&mode=search&list=1,5,2,6,6&value=5`	Algorithm takes string input and converts into numbers for algorithm.
2-3-4 Tree	search insertion	`TTFTree`	list value	`http://localhost:3000/?alg=TTFTree&mode=search&list=1,5,2,6&value=5`	`cannot accept duplicate values in the list`

Depth First Search	find	DFSrec	`xyCoord`
			`edgeWeight`
			`size`
			`start`
			`end`
			`heuristic`
			`min`
			`max`
				`http://localhost:3000/?alg=DFSrec&mode=find&size=4&start=1&end=4&xyCoords=1-10,2-2,3-1,8-2&edgeWeights=1-2-1,1-4-3,2-3-1,2-4-2&heuristic=Euclidean&min=0&max=10`	

DFS (Iterative)	find	DFS	xyCoord edgeWeight size start end heuristic min max	http://localhost:3000/?alg=DFS&mode=find&size=4&start=1&end=4&xyCoords=1-10,2-2,3-1,8-2&edgeWeights=1-2-1,1-4-3,2-3-1,2-4-2&heuristic=Euclidean&min=0&max=10	
Breadth First Search	find	BFS	xyCoord edgeWeight size start end heuristic min max	http://localhost:3000/?alg=BFS&mode=find&size=4&start=1&end=4&xyCoords=1-1,2-2,3-1,4-2&edgeWeights=1-2-1,1-4-3,2-3-1,2-4-2&heuristic=Euclidean	
Dijkstra's (shortest path)	find	dijkstra	xyCoord edgeWeight size start end heuristic min max	http://localhost:3000/?alg=dijkstra&mode=find&size=4&start=1&end=4&xyCoords=1-10,2-2,3-1,8-2&edgeWeights=1-2-1,1-4-3,2-3-1,2-4-2&heuristic=Euclidean&min=0&max=10	
A* (heuristic search)	find	aStar	xyCoord edgeWeight size start end heuristic min max	http://localhost:3000/?alg=aStar&mode=find&size=4&start=1&end=4&min=1&max=30&xyCoords=1-1,2-2,3-1,4-2&edgeWeights=1-2-1,1-3-2,1-4-3,2-3-1,2-4-2&heuristic=Euclidean	
Prim's (min. spanning tree)	find	prim	xyCoord edgeWeight size start end heuristic min max	http://localhost:3000/?alg=prim&mode=find&size=4&start=1&end=4&xyCoords=1-1,2-2,3-1,4-2&edgeWeights=1-2-1,1-4-3,2-3-1,2-4-2&heuristic=Euclidean&min=1&max=30	

Prim's (simpler code)	find	`prim_old`	`xyCoord`
			`edgeWeight`
			`size`
			`start`
			`end`
			`heuristic`
			`min`
			`max`
				`http://localhost:3000/?alg=prim_old&mode=find&size=4&start=1&end=4&xyCoords=1-1,2-2,3-1,4-2&edgeWeights=1-2-1,1-4-3,2-3-1,2-4-2&heuristic=Euclidean&min=1&max=30`	

Kruskal's (min. spanning tree)	find	`kruskal`	`xyCoord`
			`edgeWeight`
			`size`
			`start`
			`end`
			`heuristic`
			`min`
			`max`
				`http://localhost:3000/?alg=kruskal&mode=find&size=4&start=1&end=4&xyCoords=1-1,2-2,3-1,4-2&edgeWeights=1-2-1,1-4-3,2-3-1,2-4-2&heuristic=Euclidean&min=1&max=30`	

Warshall's (transitive closure)	`tc`	`transitiveClosure`	`size`
			`min`
			`max`
				`http://localhost:3000/?alg=transitiveClosure&mode=tc&size=5&min=0&max=1`	

Union Find	find	`unionFind`	`union` `value`	`http://localhost:3000/?alg=unionFind&mode=find&union=1-1,5-10,2-3,6-6&value=5`	
Brute Force	search	`bruteForceStringSearch`	`string` `pattern`	`http://localhost:3000/?alg=bruteForceStringSearch&mode=search&string=abcdef&pattern=def`	
Horspool's	search	`horspoolStringSearch`	`string` `pattern`	`http://localhost:3000/?alg=horspoolStringSearch&mode=search&string=abcdef&pattern=def`	

URL Input Code Implementation Requirement

Apply the following code changes to your <algorithm>Param.js file. E.g. ASTParam.js.

1. At the start of the file:

```
import PropTypes from 'prop-types'; // Import this for URL Param
import { withAlgorithmParams } from './helpers/urlHelpers' // Import this for URL Param
```

2. At the <algorithm>Param function definition:

`function <algorithm>Param({ mode, <parameter 1 input name>, <parameter 2 input name> }) {`

Parse in the parameters needed for the algorithm. For details of parameter names, check the Algorithm URL Table's "**Parameter Input Name**".

E.g.

```
function ASTParam( { mode, xyCoords, edgeWeights, size, start, end,
heuristic, min, max } ) {
```

3. Inside <algorithm>Param function

Use the parsed parameters as a prioritized alternative to all your algorithm's DEFAULT parameter values.

E.g. For 'sort' algorithms: `const [array, setArray] = useState(list || DEFAULT_ARR)`

If you are a Graph ('find') algorithm, additionally, define the graph_egs from the parsed parameters:

```
const graph_egs = [
      { name: 'URL Input Graph',
        size: size || GRAPH_EGS[0].size,
        coords: xyCoords || GRAPH_EGS[0].coords,
        edges: edgeWeights || GRAPH_EGS[0].edges
      }
      ]
```

```
return (
    <>
      {/* Matrix input */}
      <EuclideanMatrixParams
        name="aStar"
        mode="find"
        defaultSize={ size || DEFAULT_SIZE } // need this for URL
        defaultStart={ start || DEFAULT_START } // need this for URL
        defaultEnd={ end || DEFAULT_END } // need this for URL
        heuristic = { heuristic || DEFAULT_HEUR } // need this for URL
        min={ min || 1 } // need this for URL
        max={ max || 49 } // need this for URL
        symmetric
        graphEgs={ graph_egs || GRAPH_EGS } // need this for URL
        ALGORITHM_NAME={ASTAR}
        EXAMPLE={ASTAR_EXAMPLE}
        EXAMPLE2={ASTAR_EXAMPLE2}
        setMessage={setMessage}

      />

      {/* render success/error message */}
      {message}
    </>
  );
```

4. At the end of your <algorithm>Param.js file:

```
// Define the prop types for URL Params
QuicksortParam.propTypes = {
  alg: PropTypes.string.isRequired, // keep alg for all algorithms
  mode: PropTypes.string.isRequired, //keep mode for all algorithms
  <parameter 1 input name>: PropTypes.string.isRequired, // string
only. Don't define other PropTypes.
  <parameter 2 input name>: PropTypes.string.isRequired
};

export default withAlgorithmParams(QuicksortParam); // Export with the
wrapper for URL Params
```

E.g. For Graph ('find') algorithms:

```
ASTParam.propTypes = {
  alg: PropTypes.string.isRequired,
  mode: PropTypes.string.isRequired,
  size: PropTypes.string.isRequired,
  start: PropTypes.string.isRequired,
  end: PropTypes.string.isRequired,
  heuristic: PropTypes.string.isRequired,
  xyCoords: PropTypes.string.isRequired,
  edgeWeights: PropTypes.string.isRequired,
  min: PropTypes.string.isRequired,
  max: PropTypes.string.isRequired,
};

export default withAlgorithmParams(ASTParam); // Export with the
wrapper for URL Params
```

Share Function Code Implementation Requirement

Extracted URLs are in the format described in the {URL Documentation}

Currently all new algorithms require the use `urlState.js` and useEffect function described below. The Graph algorithms that implement `EuclideanMatrixParams.js` and `MatrixParams.js` will already have the required imports and useEffect implemented, and will require not more additions to the code.

Inside your Param file, you will need to:

- import the URLContext for the extraction of values to be used in the URL generated for sharing, useContext and useEffect from the React framework are also required

```
import { URLContext } from '../../context/urlState.js';
import React, { useContext, useEffect } from 'react';
```

- Import the required set functions

```
const { requiredExtractions } = useContext(URLContext);

// HeapSort Example
const { setNodes } = useContext(URLContext);

// TTFTree Example
const { setNodes, setSearchValue } = useContext(URLContext);
```

- Implement the useEffect, this is done within the main param function

```
useEffect(() => {
    setValue(updatingValue);
}, [updatingValue]);

// HeapSort Example
useEffect(() => {
    setNodes(localNodes);
}, [localNodes]);

// TTFTree Example
useEffect(() => {
    setNodes(nodes);
    setSearchValue(localValue);
}, [nodes, localValue])
```

Inside midpanel/index.js:

- Currently this is what is constructing the URL displayed in the box after pressing the share button. If the new algorithm doesn't require a different structure of URL from other's in it's category, no modification is needed.
- If a different structure or a new category of algorithm is required, a new case inside the switch statement would be needed.

```
  useEffect(() => {
    // this creates the url of the current algorithm, with required
parameters
    if (share) {
      let url = `${window.location.origin}/?alg=${algorithmKey}
&mode=${mode}`

      switch (category) {
        case 'Sort':
          url += `&list=${nodes}`;
          break;

        case 'Insert/Search':
          url += `&list=${nodes}&value=${searchValue}`;
          break;

        case 'String Search':
          url += `&string=${nodes}&pattern=${searchValue}`;
          break;

        case 'Set':
          url += `&union=${nodes}&value=${searchValue}`;
          break;

        case 'Graph':
          if (algorithmKey == 'transitiveClosure') {
            url += `&size=${graphSize}&min=${graphMin}&max=${graphMax}`;
          } else {
            url += `&size=${graphSize}&start=${graphStart}
&end=${graphEnd}
                  &xyCoords=${nodes}&edgeWeights=${searchValue}
&heuristic=${heuristic}`;
          }
          break;

        default:
          break;
      }

      setCurrentUrl(url);
    }
  }, [share]);
```

Inside urlState.js:

- This is were new extractions would be defined and placed in the URLContext

```
  const [nodes, setNodes] = useState([]);
  const [searchValue, setSearchValue] = useState([]);
  const [graphSize, setGraphSize] = useState([]);
  const [graphStart, setGraphStart] = useState([]);
  const [graphEnd, setGraphEnd] = useState([]);
  const [heuristic, setHeuristic] = useState([]);
  const [graphMin, setGraphMin] = useState([]);
  const [graphMax, setGraphMax] = useState([]);
  const value = {
    nodes, setNodes,
    searchValue, setSearchValue,
    graphSize, setGraphSize,
    graphStart, setGraphStart,
    graphEnd, setGraphEnd,
    heuristic, setHeuristic,
    graphMin, setGraphMin,
    graphMax, setGraphMax,
  };
```