Handover

As the product is delivered with a merged version of all 3 teams, we will give instructions on handover in one document here. **Indicative target, agreed with clients, is to have all this delivered before** 29 Oct 2021

When sent to client:

How sent to client:

Team Bluering

System architecture overview

See a rough partial ER Diagram below (click to enlarge)

Data Structures

There are several data structures able to be visualised, including 1D arrays, 2D arrays, and graphs (for trees, etc.). Each of these data structures will include their own Tracer and Renderer file.

Tracers:

Tracer files contain the class variables and methods that can be called upon the given data structure. For example, the Array2DTracer.js file contains an array of elements, which each have a value, sorted status, and any variables assigned to it. The state of these elements can be modified using the methods, like assignVariable() or sorted(), and these methods can be called within the algorithm file (e.g. quickSort.js).

Renderers:

The renderer files contain the CSS that will be rendered depending on the state of the elements mentioned in the Tracer file. For example, when sorted = 'true' for an array element, within Array1DRenderer.module.scss you can find that the background colour will be changed.

Algorithms

The algorithm files (e.g. quickSort.js) are where the steps of the algorithm are detailed, and where methods that will change the visualisation will be called. This is done by adding steps to the 'chunker'. When chunker.add(13, ...) is called, the code written within the '...' will be stored for when the pseudocode hits bookmark '13'. The location of these bookmarks in the pseudocode can be found in the REAL specification pseudocode, denoted by e.g. '\\B 13'.

Array Magnitude Heights

This is turned on for 1D arrays by passing the { arrayItemMagnitudes: true } property through to the visualiser constructor. See the following in the Heapsort algorithm controller file:

```
instance: new ArrayTracer('array', null, 'Array view', { arrayItemMagnitudes: true }),
```

Features added:

Pointers

The algorithm, throughout its execution, creates, modifies, and deletes different sets of variables that were not shown earlier in the animation. By developing this feature we are bringing the capability to add these variables to the animation. So the variables on the animation keep changing throughout the execution.

Standard methods like addVariable(), removeVariable(), assignVariable() and clearVariables() are added in the Array1DTracer.js to work with this feature. The rendering logic has been updated in the respective rendering files to render it on the animation by dynamically creating the HTML div elements.

Colour

The colour refers to the highlight of the array elements or tree nodes during the animation. The main changes is adding the colour highlight in quick sort and quick sort M3 algorithm. In code level, the functions controlling the highlight are encapsulated into two functions called highlight() and unhighlight().

Animation

Editing an Animation

This is for editing your first animation (tweening etc.), which was added in Sem 2 2021.

The library used for animation is called Framer Motion (see https://www.framer.com/motion/). Framer Motion makes use of 'motion components' which are DOM primitives optimised for 60fps animation and gestures. There's a motion component for every HTML element, for instance, motion.div or motion.li etc. which work like their static counterparts except that they offer props that allow you to:

- Declaratively or imperatively animate components.
- Add drag, pan, hover and tap gestures.
- Respond to gestures with animations.
- Deeply animate throughout React trees via variants.

Reference: https://www.framer.com/docs/component/

To add an animation

1. Navigate to the relevant 'Renderer' file (e.g. Array2DRenderer.js)
2. Ensure the file you're working on has imported the motion component, see https://www.framer.com/docs/introduction/##importing
3. Change the static HTML component you wish to animate to a motion component e.g. if the component is a , change this to <motion.div> (*see here in the code directly for an example* https://github.com/Melb-Uni/AA/blob/cad4ced964f0488cbc2d3ef8bf78acb80f8196e8/src/components/DataStructures/Array/Array2DRenderer/index.js#L99)
4. Add 'animation' props such as scale, rotate, or x and y values (for changing position) to animate the component, see https://www.framer.com/docs/animation/

Note: if you would like to animate the repositioning of a component which occurs without specifically changing the x and y position values (i.e. animating the the swap of array items - the repositioning occurs due to the array item's new index, not because we change it's x and y position values), you can utilise the 'layout' prop (see https://www.framer.com/docs/component/##layout-animation). If layout is true, the component will automatically animate to its new position when its layout changes. Ensure that such motion components have a **unique key** which doesn't change when the items position changes (i.e. an array item component's key should be different from it's index, as its index changes when its position changes but its key should not).

To edit an animation

1. Navigate to the relevant 'Renderer' file (e.g. Array2DRenderer.js) and find the motion component you wish to change the animation for (e.g. motion.td)
2. Add or change the value of the 'animation' or 'transition' props to edit the look of the animation, see https://www.framer.com/docs/animation/

Notes on Incomplete Features

Pseudocode Collapse

See explainer video for this highly important AiA Feature: Here (~10 minutes)

There are still issues with the pseudocode on the right being fully collapsed and still the algorithm stepping through stages which should be completed a single step.

To see an example of this behaviour with Heapsort (also shown in video linked above):

1. Open application to Heapsort
2. Expand Buildheap by one level
3. Observer that the loop executes once unless you expand the actual loop or its children.

To see an example of this behaviour with a recursive algorithm:

1. Open Application to Quicksort,
2. Do *not* expand any of the pseudocode

3. Execute the algorithm and observe that, due to entering the recursive calls, the call to 'Quicksort Firsthalf' occurs each time. This is because from the algorithms' execution perspective *each recursive call is a 'new' context*, so it doesn't know that it should just do all the recursive calls in one step!

To fix this, a possible approach was brainstormed, but nothing implemented:

1. Make the bookmarking / isBookmarkVisible code aware of the *depth* of a function call, i.e. if it has been called recursively, and then act accordingly. Quicksort already has a depth parameter for the Stack visualisation.
    a. The way this would work is that, if the isbookMarkVisible code encountered a depth > 1 for a *collapsed* section of pseudocode, it would skip it.

It should be noted that this (and the issues brought up in the explanation video) likely does **not** require a full rebuild of the system - some smart (re)design choices for a select few functions, as well as perhaps greater use of bookmarks for all lines of code, could result in the system working *exactly* as intended in all circumstances.

Team Redback

GraphRender

- making changes on the original file, /src/components/DataStructures/Graph/GraphRenderer. for the curved edges and resized arrows
- future work: possible solution:
    - 1. import the algorithm label to control different versions for implementation (could merge the team Boxjelly pointer).
    - 2. extract the transitive closure and prim's algorithm code out and make them inherit the parent class.

GitHub Repository

https://github.com/Melb-Uni/AA/commits/redback-warshall

Documents

- User story
- Test

Features Added

Deselect multiple edges in Prim

In Prim's algorithm, to show the detailed process of updating the Priority Queue, we loop over the edges ij connecting i to other vertices j. For each such edge, if j still belongs to Prioritg Queue and edge ij has a smaller weight than Cost[j], then update. In this process, we highlight all edges that meet the condition in blue, and deselect them after the loop. We created a funtion allLeave(target, sources) to deselect all edges in blue by this method. The `target` is i and sources are the selected edges.

User Input Validation Check for Prim

Since the input for Prim should be a symmetric matrix and, and all the grid values on the diagonal should be 0, we have improved the legitimacy test for the user's input. When the user enters different data in a symmetric location cell, we detect and give a hint of an input error instead of directly being able to produce a graph based on the user's input. Finally, we add the coordinates of the points in the head and left side of the matrix to facilitate the user's understanding.

2D Array Tag a cell

We added `assignVariable` function in Array2DTracer.js, which can help to add a specified tag beneath a given cell in a 2D array. This function can be reused for adding labels on a specific cell in the 2D array. It takes three parameters; the first is the text string of the tag, the second and third are the vertical and horizontal index of the cell.

2D Array Highlighting

Developers can easily set up the highlights in a 2D-Array structure data or table elements. All the methods for implementing the function are sealed in the file: components/DataStructures/Array/Array2DTracer.js | The file includes many useful methods including select(), deselect() which are used to highlight and unhighlight a specific cell in the 2D array respectively. The latest version of the file contains 5 different colours to be chosen as the highlighting colour. If you want to add some new colours for further use, you can simply add the colour with an id in the `Array2DTracer.js`, and also the callback functions and properties in `Array2DRenderer/` needs to be modified. Besides, `styles/global.scss` has the preset colours for global usage, the newly added colours also need to be added into that style file.

SVG documentation

- Bézier curve part for interconnected curved edges.

https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Paths

Bézier Curves

The cubic curve, `C`, is the slightly more complex curve. Cubic Béziers take in two control points for each point. Therefore, to create a cubic Bézier, three sets of coordinates need to be specified.

```
C x1 y1, x2 y2, x y
(or)
c dx1 dy1, dx2 dy2, dx dy
```

The last set of coordinates here (`x`,`y`) specify where the line should end. The other two are control points. (`x1`,`y1`) is the control point for the start of the curve, and (`x2`,`y2`) is the control point for the end. The control points essentially describe the slope of the line starting at each point. The Bézier function then creates a smooth curve that transfers from the slope established at the beginning of the line, to the slope at the other end.

NOTE: for the function in GraphTracer/index.js to calculate the control point for the Bézier curve. The curvature can be easily changed by modifying the magic number (30 in the code). larger for higher curvature.

- Edge's arrow for the SVG is drawn by 4 coordinates. (0,0) (0,6), (6,3),(0,0). triangle.
- the <path> tag could be learned in the previous link in this part
- the <marker> tag is for declaring the SVG element for further usage by calling the id "makerArrow"

```
<marker id="markerArrow" markerWidth="6" markerHeight="6" refX="3" refY="3" orient="auto">
  <path d="M0,0 L0,6 L6,3 L0,0" className={styles.arrow} />
</marker>
```

Further Work

General:

- Keep the same shape (even dragged) after stepping back the visualization.

Prim:

- More animation need to be added into the visualization. For example, make the highlighting and unhighlighting switching becomes a smoother sliding animation.
- The colours for graph edges and nodes need to be synchronized.
- Symmetric inputs autocorrection: when users type in any illegal values, such as making the matrix asymmetric, which is not allowed for Prim's algorithm, the valid check should be able to detect it and autocorrect the input values.

Warshall:

- Refactor the colour of implementation in GraphTracer. Maybe maintaining a colour queue to avoid dealing with complicated overlapping issues. (detailed in RedBackCode Review (in files))
- Smoother effect in-between the two-states switching.
- Showing how the algorithm is being optimized step by step.

Team Boxjelly

Very few architectural changes have been done by team Boxjelly.

-colour scheme

Designing a colour blindness friendly colour scheme, please refer to this document: Justification for color blindness friendly color scheme design

To change the color schemes, the code is located under : \src\styles\global.scss

-User interface-related changes

Most of the related file is under src/styles/App.scss and src/BorderResize.js

-Pointers

Bruteforce string searching and Horspool string searching has it's own pointer method, is a overloading method of it's parent function, the relative file is under /src/components/DataStructures/Graph/GraphRendererRect

-Algorithm animation for Horspools

The animation part is implemented by src/algorithms/controllers/horspoolStringSearch.js

The pseudo code(.real file) is in src/algorithms/pseudocode/horspoolStringSearch.js

For horspool's string searching, we have also created a new structure from Array2D class called TwoArray2D to visualize the shift table. It could be find under src/components/DataStructures. It includes a few functions which are very similar to Array2D.

If you are going to add new functions in this class, one thing to notice is that you should use get_position() to get the position information of that value. Input take it as a normal 2D array. But it actually is splitted into to array with fixed length. The first value returned refers to which2D array it's in; The second value returned refers to which 1D array it's in – so it's a little bit counterintuitive: x here actually is the vertical one; The last value is the position of that element in the 1D array.