CS/INFO 3300; INFO 5100
Homework 8
Due 11:59pm Friday, May 1

Your work should be in the form of an HTML file called index.html with one <p> element, one <svg> element and one <script> element. For this homework we will be using d3.js. **In the <head> section of your file, please import d3 using this tag:**

```
<script src="https://d3js.org/d3.v5.min.js"></script>
```

Create a zip archive containing your HTML file plus **the associated data file(s)** and upload it to CMS before the deadline. You will be penalized for missing data files or non-functional code. Check carefully for any issues with your file paths.

For this homework you have a choice between two problems. You do not need to complete both problems, and submissions that contain both will not receive extra credit. Should you submit a file with <p> or <svg> tags for both problems, the grader will flip a coin to decide which problem to grade.

As we are reaching the end of the term, many individual design and implementation decisions will be left to you. We will provide a general list of requirements you must meet with your submission. This format will be very similar to that of the final take-home exam.

Option A: Create a treemap

For this problem, you will construct a space-filling diagram (treemap) of United States exports. Included in the assignment file is a dataset, **2019-domestic-exports.hierarchy.json**. This file contains hierarchical data which we have already put into tree-like form (i.e. you do not need to use `d3.rollup()`). The root node contains all US domestic export volume for 2019 (gathered from dataweb.usitc.gov). Inside of the node is a property, `Children`, which contains a list of child nodes corresponding to different categories of exports. Each of those nodes has a property, `Children`, containing leaf nodes with individual kinds of exports and their export volume in US dollars ($) as properties. Every node contains two properties, `SITC Code`, corresponding to a standard export code system, and `Description`, a verbal description of the export category.

Please implement the following components for your final visualization:

- Import the JSON file asynchronously
- Use d3.hierarchy to build a hierarchy structure for your data
- Use d3.treemap to add coordinates to your data hierarchy
- Draw a treemap in a SVG canvas that is sized between 400px and 800px wide and 400px and 800px tall.

- Implement a mouseover feature for providing more details about individual rectangles through a `<text>` element and visual highlighting of some kind
- Implement a color scale to distinguish different categories of exports (i.e. `height==1` nodes) in the treemap

Your treemap should include the following design considerations:

- Load quickly with minimal delay.
- Be of sufficient size that most rectangles are visible (there will be small rectangles for minor exports that may be hard to spot).
- Make use of padding and spacing so that all rectangles are distinguishable.
- Be sorted in a way that makes it clear which export in each category is the largest.
- Use color to clearly distinguish rectangles between the 9 SITC categories of exports.
- When a user moves their mouse over a rectangle, a `<text>` label should appear describing the rectangle in question, and some change in of stroke, shading, or opacity should indicate which one has been selected.
- The `<text>` label should be visible and obvious to the user. Positioning is up to you, but you should minimize clipping.
- When the mouse leaves a rectangle, the annotations and changes should be cleaned up gracefully – no traces should remain.
- Including permanent labels on all rectangles (as was done in the class notes) is optional here. The only text label you must include is one during mouse hover.

You are welcome to include in your `<p>` a description of any design trade-offs or dilemmas you encountered. There may be cases where you feel that you are trading between bad options – describing these will help graders judge the effectiveness of your overall design.

Option B: Create a chord diagram

For this problem, you will construct a chord diagram of character co-occurrence in the Game of Thrones book series. Included in the assignment file is a dataset, **thrones-cooccur.json**. This file contains nodes and edges representing relationships between book characters (gathered from kaggle.com/mmmarchetti/game-of-thrones-network-analysis). Each node has several properties including the `Name` of the character, that character's `Affiliation`, and a `Weight` of all co-occurrences (not necessary for this chart). Each edge has several properties, including the `source` character name, `target` character name, and `weight` corresponding to the number of co-occurrences the characters share. Edges are bidirectional, so your edge matrix should be symmetrical across the diagonal, as it was in class notes.

Please implement the following components for your final visualization:

- Import the JSON file asynchronously

- Use JavaScript to put your data into matrix form (*hint: you might need a dictionary that converts from* `Name` *to index in the nodes array in order to properly build a matrix*)
- Create appropriate d3 generators for building a chord diagram
- Draw a chord diagram, including outer ring and ribbons in a SVG canvas that is sized between 400px and 800px wide and 400px and 800px tall
- Color each ring element by the character's `Affiliation`
- Appropriately color ribbons by either their source or target character's `Affiliation` (ideally one would use a gradient, but that is not required here)
- Implement a mouseover feature for providing more details about each individual character. When a user hovers over a ring segment for a character, that segment as well as its ribbons should be highlighted. A `<text>` label, appropriately rotated on the circle, should indicate the `Name` of the character. (*hint: you may want to give segments, ribbons, and text labels a class corresponding to the character's* `Name`*. this would allow you to later find them in a mouseover event and highlight them all together*)

Your chord diagram should include the following design considerations:

- Load quickly with minimal delay.
- Be of sufficient size that most segments and ribbons are visible (there will be small elements for minor characters that may be hard to spot).
- Make use of padding and spacing so that all segments are distinguishable.
- Follow the sorting order of the nodes as included in the dataset, which should order segments by `Affiliation`.
- Use color to clearly distinguish segments between the different `Affiliations`.
- When a user moves their mouse over a ring segment, a `<text>` label should appear describing the segment, and some change in of stroke, shading, or opacity should indicate which one has been selected. Ribbons should similarly be highlighted
- The `<text>` label should be visible and obvious to the user. You should position and rotate the label so that it "sticks out of" the circle as labels in the class notes do.
- When the mouse leaves a segment, the annotations and changes should be cleaned up gracefully – no traces should remain.
- Including permanent labels on all segments (as was done in the class notes) is optional here. The only text label you must include is one during mouse hover.

You are welcome to include in your `<p>` a description of any design trade-offs or dilemmas you encountered. There may be cases where you feel that you are trading between bad options – describing these will help graders judge the effectiveness of your overall design.