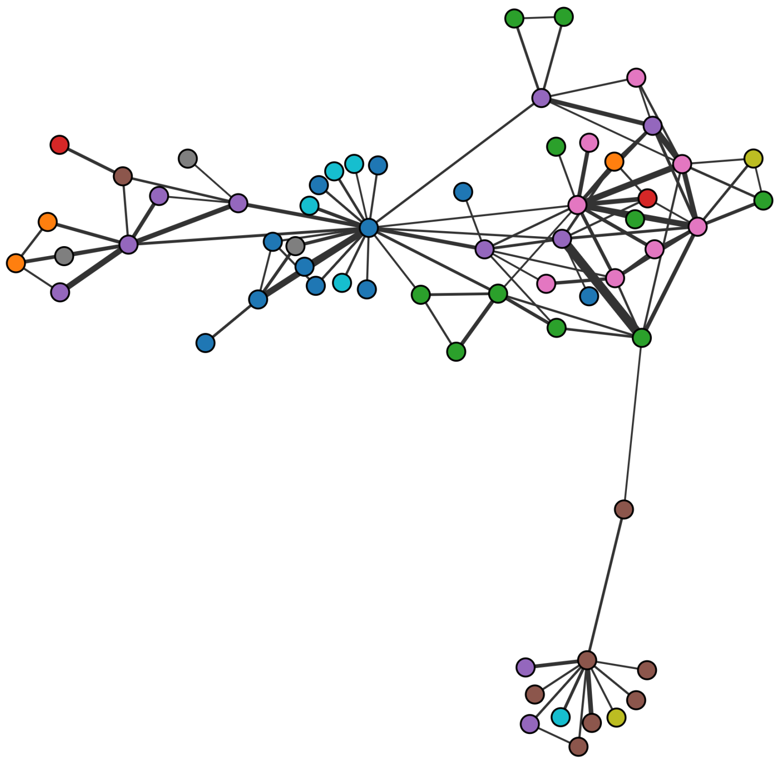
CS/INFO 3300; INFO 5100
Homework 8
Due 11:59pm Wednesday, November 1

Goals: Practice using d3 to create a network diagrams

1. You will visualize a network dataset, `thrones-cooccur.csv`. This dataset contains character "co-occurrence" in the Game of Thrones book series. An algorithm counted the number of times that each character appeared in a chapter along with another character to generate the network. For example, if Danaerys appears with Irri in three separate chapters, then the network connects these two characters with a `weight` of 3. Characters have also been labeled based on faction `Affiliation` so that patterns might emerge.

Example (your diagram may not look identical to this):



A. Following your <p> element, create an **SVG element 600px in width and 600px in height**. Within a `<script>` tag, use d3 to create a **<g> element** within your SVG to contain your network diagram. Using either promises or await, **load the dataset** into memory. Use `console.log()` to examine the dataset. You will find that it contains an object with two keys, `nodes` and `edges`. We recommend that you assign these two values to variables for ease.

Create two scales for your diagram. First, create a `scaleLinear` to help adjust stroke width of lines based on the `weight` value of each edge. We want thicker lines to appear for stronger co-occurrence connections. To do so, set the domain of your scale by computing the `d3.extent` of the `weight` values in the edges array. Set the range to be `[1, 5]`. Second, create a `scaleOrdinal` using the `d3.schemeCategory10` color scheme which you will use to color nodes by each character's `Affiliation` value.

B. Construct a `d3.forceSimulation` model for your network diagram. You can use the data from the `nodes` key in the dataset as **nodes** in the model. Your model should include the following forces:

- A **linking force for edges** in the network. Use data from the dataset's `edges` key to build your links. Source and target in the edges array correspond to the `Name` property of nodes in this dataset, so **be sure to set `.id()` properly for this force so that it knows what to look for in each of the nodes when connecting them.**

- A **many body repulsive force** between all nodes. Tune the strength of this force so that both clusters and outliers are evident and remain completely within the canvas. A value around `-20` should work fine.

- A **centering force** that keeps all nodes in the center of the chart. You do not need to set any strength value.

C. Make a function, `render()`, that **uses a data join to draw edges** and a **data join to draw nodes**.

Draw the `edges` first so that they do not appear to be on top of the nodes. Use your linear scale to set the `stroke-width` of the black connecting lines based on the `weight` value of each edge. **Make sure that `opacity` remains at the default of 1** for performance reasons.

Draw circles 5px in radius for each node and **set their `fill` color using the color scale you made earlier**. Recall that your color scale is supposed to be based on the `Affiliation` value for each node. Give them a **1px outer stroke** in a dark grey or black color. Be sure to use `join()` properly so that you only create nodes/edges once and update all of them each time `render()` is called.

Finally, add an `.on("tick")` call to your force simulation to call your `render()` function. If your simulation quickly gets slow or has ghostly trails, check your join for issues with what it is selecting each time `render()` is called.

D. Adjust your code so that you can **drag nodes around the screen using your mouse.**

Use the `d3.drag()` tool demonstrated in class instead of writing your own drag tool. Use `.fx` and `.fy` parameters on your nodes in order to deliver smooth animations, and reheat the simulation as necessary to permit node movement using `alpha()` and `alphaTarget()`.

BONUS. (No extra credit offered, this is just for completionism)
When the user starts dragging a node, **the name of the character should appear in a text label**. The label can either be placed in a corner of the SVG canvas or follow the mouse. The label should disappear when the drag ends.