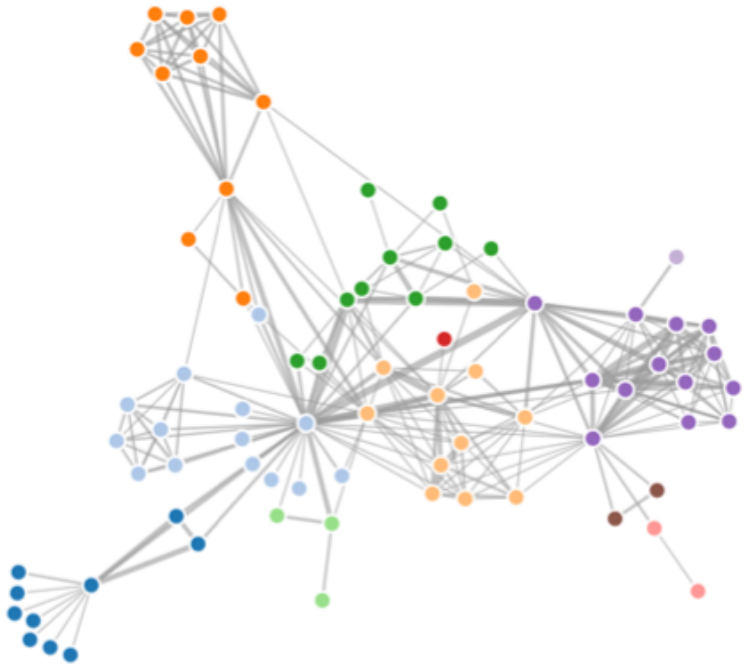
D3 Tutorial

Force-directed Layout

Edit by Jiayi Xu and Han-Wei Shen, The Ohio State University

Force-directed Layout

- Uses a **physics based simulator** for positioning visual elements
  - Can automatically position nodes in an aesthetically pleasing way

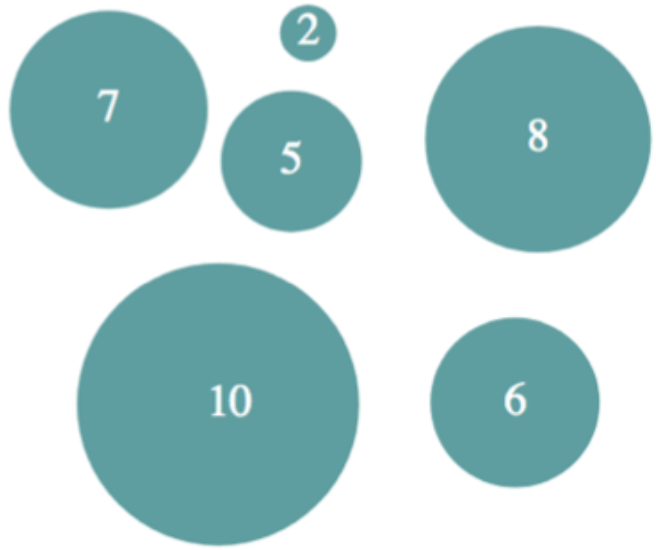


A simple example

- We create a very simple example to set up a force simulation
- Set screen size

```
var width = 300, height = 300;
```

- Data
  - Six nodes with values



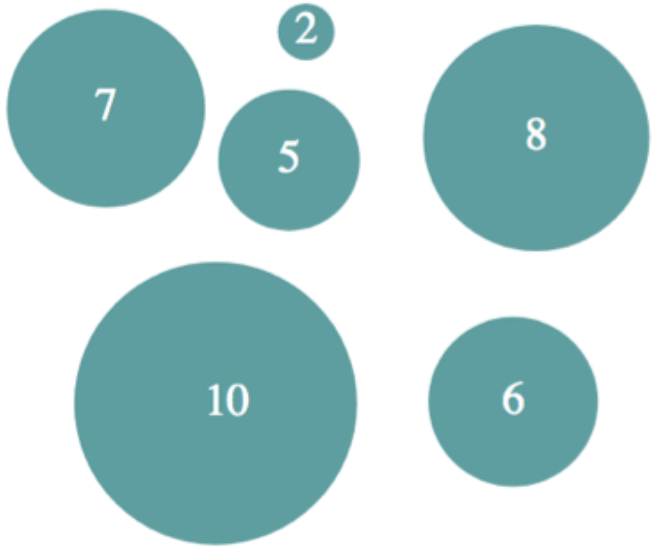

```
var nodes = [{value: 5}, {value: 10}, {value: 2}, {value: 6}, {value: 7}, {value: 8}];
```

  - We just intend to position the six nodes on the screen **without overlapping**

A simple example

- Radiuses of circles encode the values of nodes
  - Create a scale mapping values to radiuses
- Create six circles

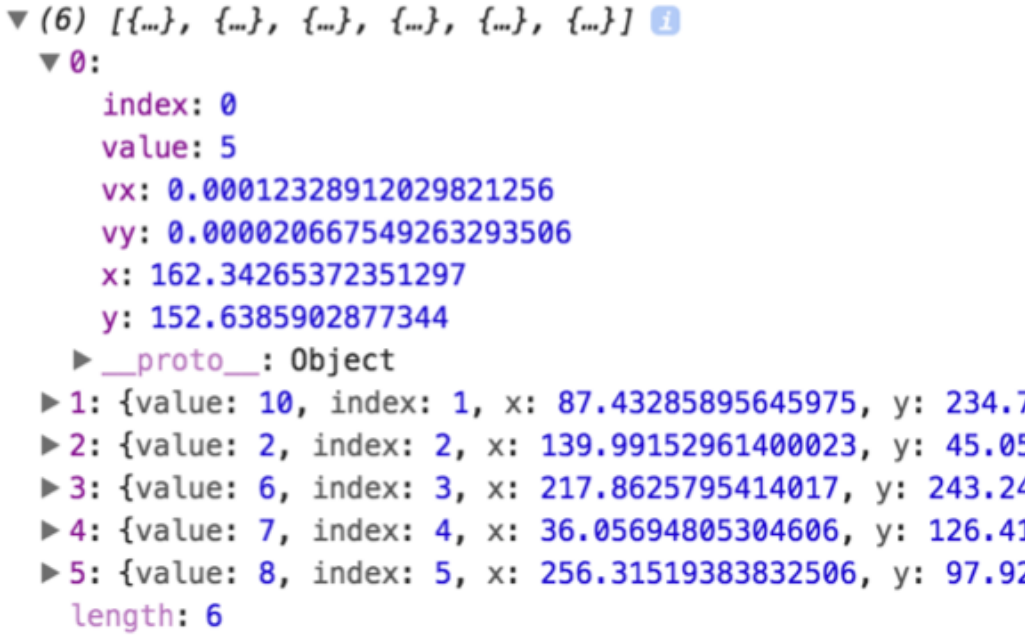
```
var value2radius = d3.scaleLinear()
    .domain([0, d3.max(nodes, function(d) {
        return d.value;
    })])
    .range([0, 50]);

var circles = d3.select('svg')
    .selectAll('circle')
    .data(nodes)
    .enter()
    .append('circle')
    .attr('r', function(d) {
        return value2radius(d.value);
    });
```



A simple example

```
var simulation = d3.forceSimulation(nodes)
    .force('center', d3.forceCenter(width / 2, height / 2))
    .force('collision', d3.forceCollide().radius(function(d) {
        return value2radius(d.value);
    }))
    .on('tick', ticked);
```

- Create a simulation by d3.forceSimulation()
  - The simulator starts automatically
  - Iterates 300 times by default
- The simulator assigns some attributes to nodes
  - index
    - Zero-based
  - *x* and *y*
    - the *nodes'* current positions
    - Update in each iteration
  - *vx* and *vy*
    - the *nodes'* current velocities
    - Update in each iteration



A simple example

```
var simulation = d3.forceSimulation(nodes)
    .force('center', d3.forceCenter(width / 2, height / 2))
    .force('collision', d3.forceCollide().radius(function(d) {
        return value2radius(d.value);
    }))
    .on('tick', ticked);
```

- Add two forces by .force(*name*, *force*)
  - *name* is defined by yourself
  - d3.forceCenter(*x, y*)
    - Nodes will be attracted to a center [*x*, *y*]
  - d3.forceCollide()
    - Prevents nodes from overlapping
    - Collision force repels two nodes when they are closer than the sum of their radiuses

A simple example

```
var simulation = d3.forceSimulation(nodes)
    .force('center', d3.forceCenter(width / 2, height / 2))
    .force('collision', d3.forceCollide().radius(function(d) {
        return value2radius(d.value);
    }))
    .on('tick', ticked);
```

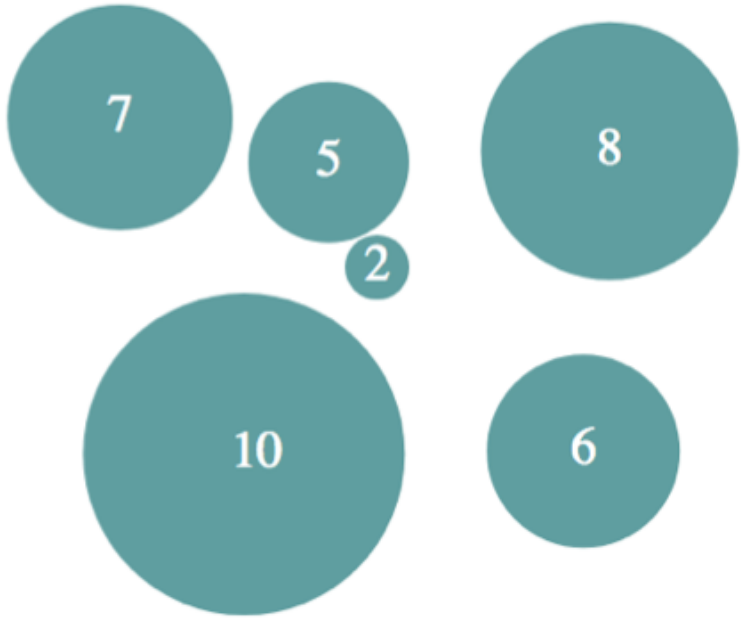
- *simulation*.on('tick', *functionName*)
  - In each iteration, the simulator will call the specified function once (here, the *ticked* function on the right)
  - *ticked* function
    - In each iteration, reset the *x* and *y* of circles on the screen based on current nodes' positions
- *simulation*.on('end',*functionName*)
  - Call the function once after the simulation ends

```
function ticked() {
    d3.selectAll('circle')
        .attr('cx', function(d) {
          return d.x
        })
        .attr('cy', function(d) {
          return d.y
        });
}
```

Fix nodes

- To fix a node in a given position, you may specify two additional properties:
  - *fx* - the node's fixed *x*-position
  - *fy* - the node's fixed *y*-position
  - Can just use *fx* or *fy*
- For example
  - Fix a node on the center

```
var nodes = [
    {value: 5},
    {value: 10},
    {value: 2, fx: 150, fy: 150},
    {value: 6},
    {value: 7},
    {value: 8}
];
```



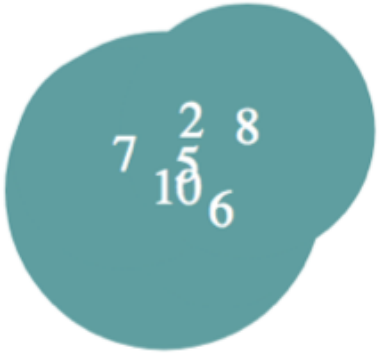
Force functions

- The power and flexibility of the force simulation is centered around **force functions** which adjust the position and velocity of elements to achieve a number of effects such as attraction and repulsion
- Force functions are added to the simulation using .force(*name*, *force*)
  - The first argument is a user defined id
  - The second argument is a force function
- D3 provides a number of useful built-in functions
  - Force from a static object
    - forceCenter, forceX, forceY, forceRadial
  - Force between nodes
    - forceCollide, forceManyBody, forceLink

Force functions – d3.forceCenter

- d3.forceCenter(*x, y*)
  - Nodes will be attracted to a center [*x*, *y*]
- Usually, we should use *forceCenter* together with d3.forceCollide()
  - Otherwise, the nodes overlap
  - E.g., we remove the collision force in the previous example

```
var simulation = d3.forceSimulation(nodes)
    .force('center', d3.forceCenter(width / 2, height / 2))
    // .force('collision', d3.forceCollide().radius(function(d) {
        // return value2radius(d.value);
    // }))
    .on('tick', ticked);
```



Force functions – d3.forceX and d3.forceY

- The x- and y-positioning forces push nodes towards a desired position
- Create a generator
  - d3.forceX().x(a *value* or a *function*)
  - d3.forceY().y(a *value* or a *function*)
- For example, we can push nodes along the line $x = 50$
  - Encode the values of nodes into *y* by *forceY*

```
var xForce = d3.forceX().x(50);
var yForce = d3.forceY()
    .y(function(d) {
        return d.value * 10;
    });
```

```
var simulation = d3.forceSimulation(nodes)
    .force('x', xForce)
    .force('y', yForce)
    .force('collision', d3.forceCollide().radius(nodeRadius))
    .on('tick', ticked);
```

$x = 50$

Force functions – d3.forceX and d3.forceY

- .strength()
  - We can set the strength of the *forceX* and *forceY* by
    - *forceGenerator*.strength(*a value or a function*)
  - The default of strength is 0.1

strength: 0.1

$x = 50$

```
var xForce = d3.forceX().x(50).strength(1);
```

$x = 50$

- Large strength of *forceX* affects the effect of *forceY*

Force functions – d3.forceRadial

- Push nodes towards the closest point on a given circle
  - The circle has a specified radius centered at (*x*, *y*)
  - If *x* and *y* are not specified, they default to (*0*, *0*)
- d3.forceRadial()
  - .radius(a *value* or a *function*)
  - .x(a *value* or a *function*)
  - .y(a *value* or a *function*)

Force functions – d3.forceRadial

- Example
  - Create 50 empty nodes

```
var nodes = d3.range(50).map(function(d) {
    return {};
});
```

  - Push nodes to a circle of 100px radius
    - centered at (0, 0)

```
var radialForce = d3.forceRadial().radius(100);

var simulation = d3.forceSimulation(nodes)
    .force('r', radialForce)
    .force('collision', d3.forceCollide().radius(nodeRadius))
    .on('tick', ticked);
```

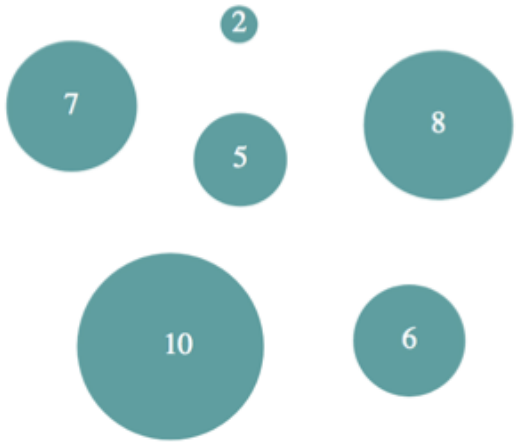
Force functions – d3.forceCollide

- *forceCollide* treats nodes as circles with a given radius, rather than points, and prevents nodes from overlapping
  - More formally, two nodes *a* and *b* are separated so that the distance between *a* and *b* is at least *radius*(*a*) + *radius*(*b*)
- Must specify the radius of the nodes by
  - d3.forceCollide().radius(a *value* or a *function*)
- Also, we can set the strength by
  - .strength(a *value* or a *function*)
  - Defaults to 0.7

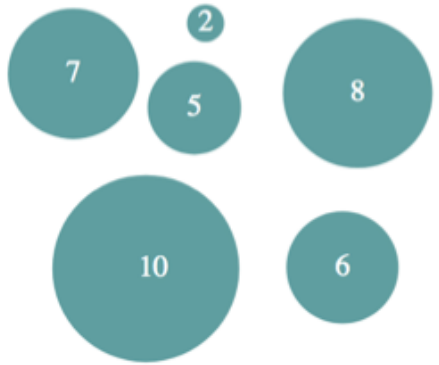
Force functions – d3.forceManyBody

- The many-body (or n-body) force applies mutually amongst all nodes
  - If the strength is negative, simulate electrostatic charge (repulsion)
  - If the strength is positive, simulate gravity (attraction)
- .strength(a *value* or a *function*)
  - Defaults to -30, repulsion

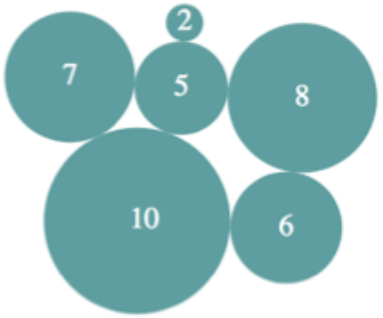
strength: -10



strength: 0

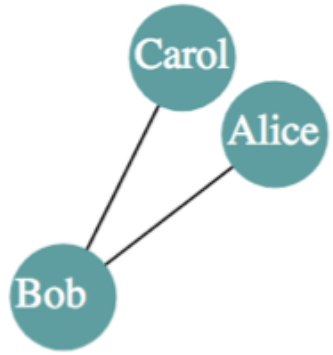


strength: 10



Force functions – d3.forceLink

- The link force pushes linked nodes together or apart according to the desired link distance
  - Useful for network data
- For example:
  - Three persons: Alice, Bob, Carol
  - Relationship: Alice knows Bob, Bob knows Carol

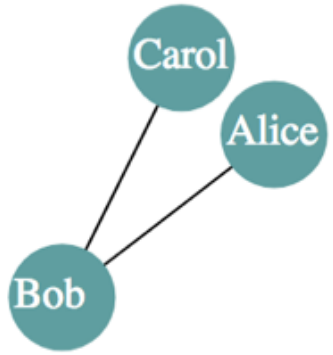



```
var nodes = [
  {"id": "Alice"},
  {"id": "Bob"},
  {"id": "Carol"}
];
```

```
var links = [
  {"source": "Alice", "target": "Bob"},
  {"source": "Bob", "target": "Carol"}
];
```

Force functions – d3.forceLink

```
var linkForce = d3.forceLink()
    .links(links)
    .id(function(d) {
        return d.id;
    })
    .distance(100);
```



- .links(links)
  - Set the data of links
  - Each link is an object like

    {

    “source”: source node id,

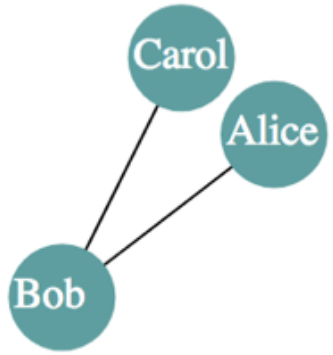
    “target”: target node id

    }
    - Must contain *source* and *target*
    - Also can contain other attributes of links

```
var links = [
  {"source": "Alice", "target": "Bob"},
  {"source": "Bob", "target": "Carol"}
];
```

Force functions – d3.forceLink

```
var linkForce = d3.forceLink()
    .links(links)
    .id(function(d) {
        return d.id;
    })
    .distance(100);
```



- .id(function(d) {return d.**id**; })
  - Identify each node based on the *id* of nodes
  - The values of *source* and *target* are corresponding to the *id* of nodes

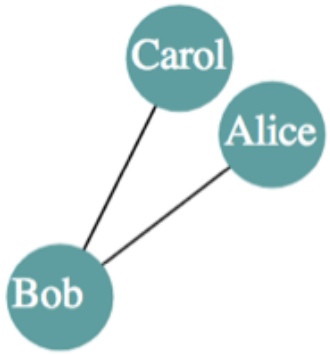
```
var nodes = [
  {"id": "Alice"},
  {"id": "Bob"},
  {"id": "Carol"}
];

var links = [
  {"source": "Alice", "target": "Bob"},
  {"source": "Bob", "target": "Carol"}
];
```

Force functions – d3.forceLink

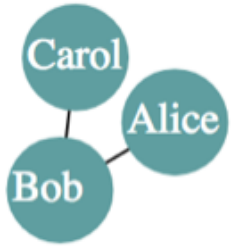
```
var linkForce = d3.forceLink()
    .links(links)
    .id(function(d) {
        return d.id;
    })
    .distance(100);
```

- .distance(a *value* or a *function*)
  - Set desired distance for each link
    - Defaults to 30
  - Each link pushes nodes together or apart to reach its desired distance

distance: 100

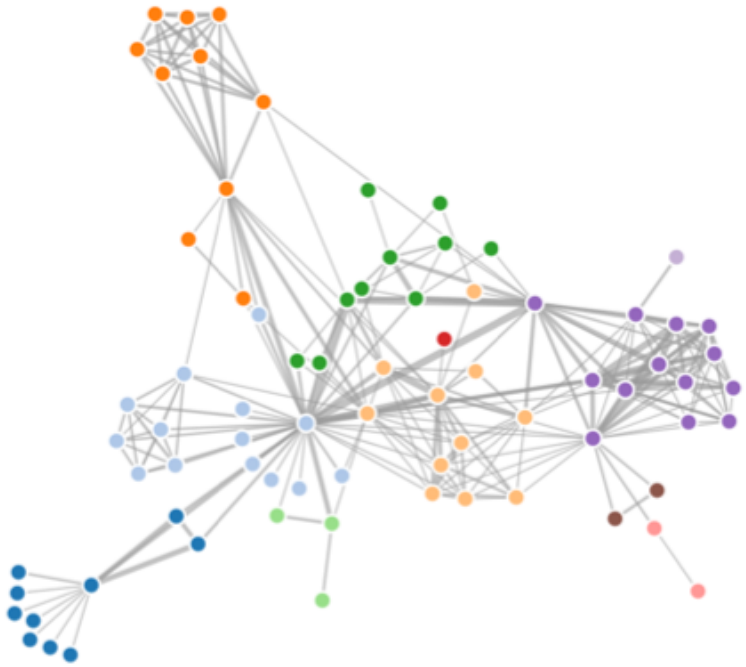


distance: 50

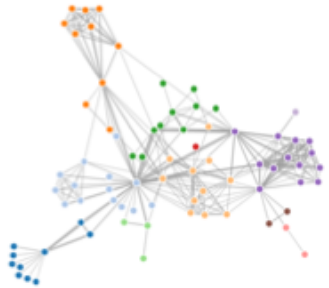


Try a real data - Les Misérables

- We will visualizes character co-occurrences in Victor Hugo's *Les Misérables*



Les Misérables - Data

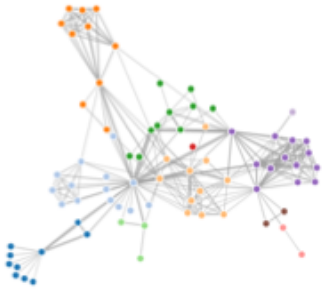


- Data
  - *miserables.json*
- Nodes
  - Id: character names
  - Groups based on their
- Links
  - Source and target
  - Value: number of occurrences
    - Count one if two characters appeared in the same chapter


```
{
  "nodes": [
    {"id": "Myriel", "group": 1},
    {"id": "Napoleon", "group": 1},
    {"id": "Mlle.Baptistine", "group": 1},
    {"id": "Mme.Magloire", "group": 1},
    {"id": "CountessdeLo", "group": 1},
    {"id": "Geborand", "group": 1},
    {"id": "Champtercier", "group": 1},
    {"id": "Cravatte", "group": 1},
    {"id": "Count", "group": 1},
    {"id": "OldMan", "group": 1},
    {"id": "Labarre", "group": 2},
    {"id": "Valjean", "group": 2},
```

```
"links": [
    {"source": "Napoleon", "target": "Myriel", "value": 1},
    {"source": "Mlle.Baptistine", "target": "Myriel", "value": 8},
    {"source": "Mme.Magloire", "target": "Myriel", "value": 10},
    {"source": "Mme.Magloire", "target": "Mlle.Baptistine", "value": 6},
    {"source": "CountessdeLo", "target": "Myriel", "value": 1},
    {"source": "Geborand", "target": "Myriel", "value": 1},
    {"source": "Champtercier", "target": "Myriel", "value": 1},
    {"source": "Cravatte", "target": "Myriel", "value": 1},
    {"source": "Count", "target": "Myriel", "value": 2},
    {"source": "OldMan", "target": "Myriel", "value": 1},
    {"source": "Valjean", "target": "Labarre", "value": 1},
    {"source": "Valjean", "target": "Mme.Magloire", "value": 3},
    {"source": "Valjean", "target": "Mlle.Baptistine", "value": 3},
    {"source": "Valjean", "target": "Myriel", "value": 5},
```

Les Misérables - Init



```
var width = 960;
var height = 600;
var svg = d3.select("svg");
svg.attr("width", width).attr("height", height);

var color = d3.scaleOrdinal(d3.schemeCategory20);
```

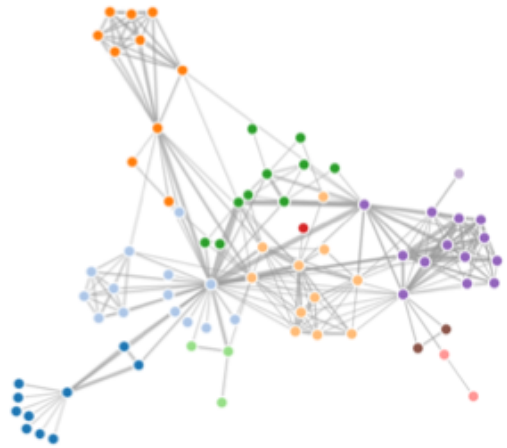
- Set the width and height of the screen
- Create a colormap for different groups
  - d3.schemeCategory20 is a built-in colormap

Les Misérables – Lines and Circles

```
d3.json("miserables.json", function(error, graph) {
  if (error) throw error;

  var lines = svg.append("g")
    .attr("class", "links")
    .selectAll("line")
    .data(graph.links)
    .enter().append("line")
    .attr("stroke-width", function(d) {
      return Math.sqrt(d.value);
    });

  var circles = svg.append("g")
    .attr("class", "nodes")
    .selectAll("circle")
    .data(graph.nodes)
    .enter().append("circle")
    .attr("r", 5)
    .attr("fill", function(d) {
      return color(d.group);
    });
```

- Load data into *graph* variable
- Create lines to show links
  - Line width encodes the co-occurrence of characters
- Create circles to represent characters
  - Colors encode different groups

Les Misérables – Force Simulation



```
var linkForce = d3.forceLink()
  .links(graph.links)
  .id(function(d) {
    return d.id;
  });

var simulation = d3.forceSimulation(graph.nodes)
    .force("link", linkForce)
    .force("charge", d3.forceManyBody())
    .force("center", d3.forceCenter(width / 2, height / 2))
    .on("tick", ticked);
```

- Import data of *links* and *nodes* into the force simulation
- *forceLink* will cause linked nodes to be close
- *forceManyBody* will push nodes apart
  - Because the strength is -30 by default (repulsion)
- *forceCenter* will push the whole graph towards the center of screen

Les Misérables - Update

```
function ticked() {
  circles
      .attr("cx", function(d) { return d.x; })
      .attr("cy", function(d) { return d.y; });

  lines
      .attr("x1", function(d) { return d.source.x; })
      .attr("y1", function(d) { return d.source.y; })
      .attr("x2", function(d) { return d.target.x; })
      .attr("y2", function(d) { return d.target.y; });
}
```

- In each iteration
  - Update the positions of nodes
  - Update the endpoints of lines