姓名 学号

- 张进华 201900150221

实验题目 使用cola.js和dagre-d3.js可视化数据

- 话说可视化的实验还真挺有意思的，这次让我们体验下面两个网站，然后使用cola.js和dagre.js，我们实验6使用d.js实现过force-directed layout，那么这次就是在实验6的基础上修改然后实现了dagre-d3.js

实验内容

1. 使用 https://ialab.it.monash.edu/webcola/
2. 使用 https://github.com/dagrejs/dagre

- 本次实验内容已同步到**本人博客**，可点击查看

实验步骤

1. 体验cola.js

这个小实验通过力导向图可视化树形数据结构，数据集链接可在文末找到

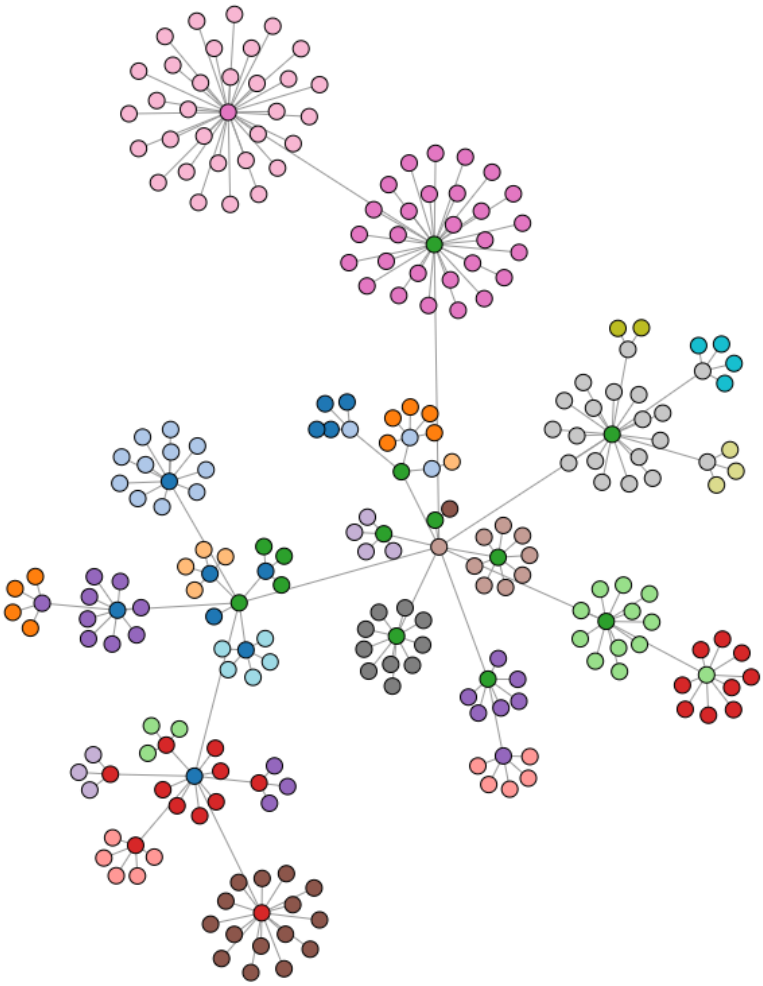
实验原理了解

- cola.js是浏览器中基于约束的布局，与 D3 力布局相比：
  - cola.js 实现更高质量的布局，CoLa 具有更好的收敛特性
  - 在交互式应用程序中更加稳定（无"抖动"),平滑地滑到布局目标函数中的局部最小值
  - 它允许用户指定的限制，如对齐和分组;
  - 它可以自动生成限制：
  - 避免重叠节点
  - 为定向图形提供流布局;
- Force-directed graph drawing

> 力导向布局算法是一类绘图算法，它仅仅基于图的解构本身来绘图，而不依赖于上下文信息。可以用于描述关系图的结点之间的关系，把结点分布到画布上合理的位置，比如描述企业之间的关系，社交网络中的人际关系等。

实验效果

- 为了让大家有欲望能看下面的内容，附上实验效果，如图所示，可拖拽交互



关键操作代码注释

- 本次实验操作与实验6大体上差不多，区别在于要用cola.js替换 D3 force ，首先将布局尺寸设置为SVG图形尺寸，至于引入 d3.js 以及 cola.js 两个文件的代码可在文末找到，就不多说了

```
    var force = cola.d3adaptor()
        .linkDistance(30)
        .size([w, h]); // 设置布局尺寸为svg图形尺寸
```

- 然后cola.js直接最小化理想链接距离和图中实际距离之间的差异,只需将 linkDistance 设置为适合节点大小的值
- 由边连接的节点对具有垂直分离约束，要求源与目标之间的距离最小，还要求节点边界框不能重叠，完全按照 D3 力布局来指定节点和链接，但是约束是一个新参数，包含指定约束的数组,构造如下

```
var constraints = []; // 约束集合
    for (var i = 0; i < 5; i++) {
        constraints.push({ "axis": "x", "left": i, "right": 10, "gap":
nodeRadius });
        constraints.push({ "axis": "y", "left": i, "right": 10, "gap":
nodeRadius });
    }; // 其定义为graph.nodes[i].y + gap <= graph.nodes[10].y
```

- 然后读取数据，将层级数据平铺，创建连接关系，与实验6相同，d3.layout.tree().links()函数返回一个连接对象数组，用来表示每个给定节点对象从父结点到子节点间的连接，建立树结构

```
        force // 节点数据和连接属性添加到力布局
            .nodes(nodes)
            .links(links)
            .constraints(constraints) // 添加约束
            .symmetricDiffLinkLengths(5) // 计算每个链接上的理想长度，以在高阶节点周围留出额外空间
            .avoidOverlaps(true) // 在布局进行时动态生成约束，以防止节点的边界框相互滑动
            .start(10,15,20); // start 最初将应用10次没有约束的布局迭代，15次只有结构约束的迭代和20次具有所有约束（包括反重叠约束）的布局迭代
```

- flatten()函数用来将层级化的数据集铺平，实现原理与实验6相同
- 之后绑定连接数据和节点数据，并设置相应属性

  注册tick事件处理函数，基于力布局的计算结果更新所有circle元素的位置和所有link元素的首尾位置

2.体验dagre-d3.js

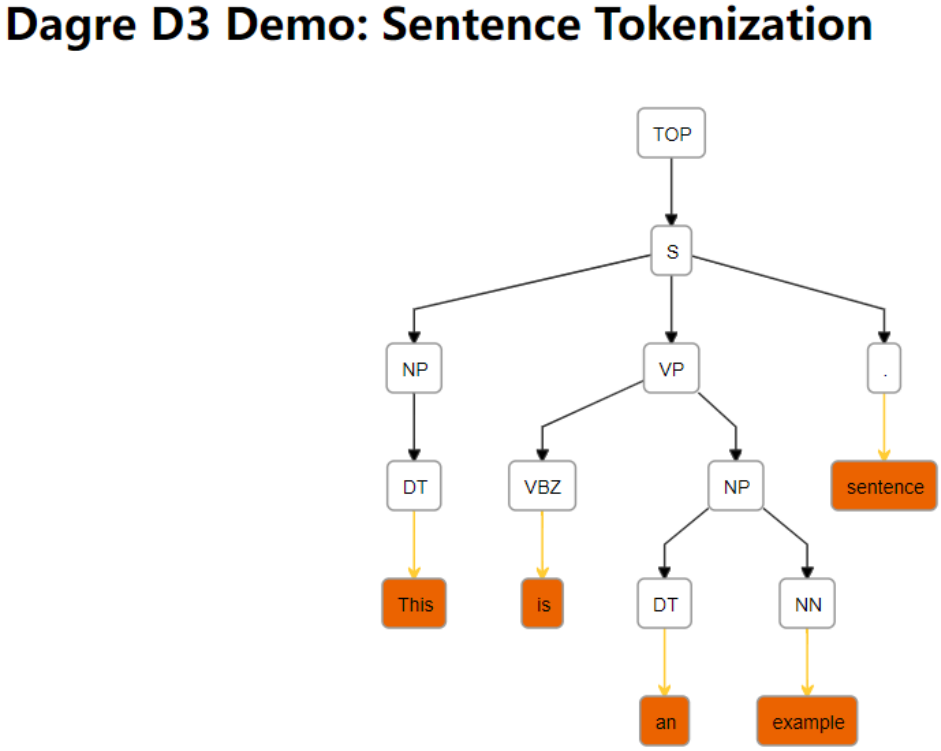
这个小实验使用dagre-d3.js来可视化句子标准化的示例， 展示了如何将 CSS 类应用于渲染图。

实验原理了解

- dagre 是专注于有向图布局的 javascript 库，由于 dagre 仅仅专注于图形布局，需要使用其他方案根据 dagre 的布局信息来实际渲染图形，而 dagre-d3 就是 dagre 基于 D3 的渲染方案。图布局包含了以下5个重要概念：
  - graph，即图整体，用来配置图的全局参数。
  - node，即顶点，dagre 在计算时并不关心 node 实际的形状、样式，只要求提供维度信息。
  - edge，即边，edge 需要声明其两端的 node 以及本身方向。例如A -> B表示一条由 A 指向 B 的 edge。
  - rank，即层级，rank 是流程图布局中的核心逻辑单位，edge 两端的 node 一定属于不同的 rank，而同一 rank 中的 node 则会拥有同样的深度坐标（例如在纵向布局的 graph 中 y 坐标相同）。
  - label，即标签，label 不是必要元素，但 dagre 为了适用更多的场景增加了对 edge label 的布局计算。

实验效果

- 同样先附上实验效果，如图所示，可拖拽交互进行移动放缩



关键操作代码注释

- 首先引入 d3.js 以及 dagre-d3.js 两个文件，然后定义css样式，主要为结点样式、连接边样式以及叶结点样式
- 叶结点样式，定义动画效果，实现不断地变化叶结点颜色，实现渲染效果

```
g.type-TK> rect { /* 设置“TK”节点的颜色*/
    fill: #ccff00;
    -webkit-animation:myAnimation 30s; /* Safari and Chrome */
}
/* 动画 */
@-webkit-keyframes myAnimation /* Safari and Chrome */
{
    0%   {fill:#ccff00;}
    15%  {fill:#ff0000}
    30%  {fill:#ff3300;}
    45%  {fill:#ff6600;}
    60% {fill:#ff9900;}
    75% {fill:#ffcc00;}
    100% {fill:#ffff00;}
}
```

- 其他样式比较常规,后期在绘制时对叶结点的连接边做了特殊处理，让其颜色区别于其他边

```
text { /* 文本内容样式*/
     font-weight: 300;
     font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
     font-size: 14px;
 }

 .node rect { /*节点样式*/
```

```
        stroke: #999;
        fill: #fff;
        stroke-width: 1.5px;
    }

    .edgePath path { /*路径样式*/
        stroke: #333;
        stroke-width: 1.5px;
    }
```

- 然后创建输入图，调用dagreD3.graphlib.Graph()函数

```
    // 创建输入图
    var g = new dagreD3.graphlib.Graph()
        .setGraph({})
        .setDefaultEdgeLabel(function() {
            return {};
        });
```

- 接下来设置节点类型，便于后面绘制时被drawNodes 函数使用

```
// 设置nodeclass，被drawNodes 函数使用
    g.setNode(0, {
        label: "TOP",
        class: "type-TOP"
    });
```

- 同时在这里设置矩形框的样式，设置圆角，让其更美观

```
 g.nodes().forEach(function(v) {
        var node = g.node(v);
        //圆角半径
        node.rx = node.ry = 5;
    });
```

- 然后设置连接边属性，特殊边特殊处理，及对叶结点连接边改变颜色

```
    // 设置边样式
    g.setEdge(3, 4,{style: "stroke: #ffcc33; fill: none;",//线样式
        arrowheadStyle: "fill: #ffcc33;stroke: #ffcc33;",//箭头样式
        arrowhead: 'vee' });// 箭头形状样式
```

- 接下来创建渲染器，绘制图形，同时设置一个SVG组，便于整体处理，定义位置

```
   // 创建渲染器
    var render = new dagreD3.render();

    // 设置一个 SVG 组
    var svg = d3.select("svg"),
        svgGroup = svg.append("g");

    // 运行渲染器,绘制最终图形
    render(d3.select("svg g"), g);
```

- 最后实现图的居中对齐以及鼠标交互

```
// 整个流程图的位置在界面中，左右上下居中显示
    var xCenterOffset = (svg.attr("width") - g.graph().width) / 2;
    svgGroup.attr("transform", "translate(" + xCenterOffset + ", 20)");
    svg.attr("height", g.graph().height + 40);

    // 建立拖拽缩放
    let zoom = d3.zoom()
        .on("zoom", function () {
            svgGroup.attr("transform", d3.event.transform);
        });
    svg.call(zoom);
```