姓名

- 张进华

学号班级

- 智能19 201900150221

实验题目

- force-directed layout的算法

实验内容

- 理解force-directed layout的算法，构建用例跑一下d3的fore-directed layout

实验原理

The Barnes-Hut Approximation

了解The Barnes-Hut Approximation算法的来源、目的以及执行步骤，巴恩斯-胡特算法涉及三个步骤：

1.构建空间索引（例如四树）

2.计算质量中心

四树细胞的质量中心只是其四个子细胞中心的加权平均值。

3.估计力

最后算法的时间复杂度可以达到O(n)

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。

实验步骤

- 关键代码注释
- 首先将布局尺寸设置为SVG图形尺寸

```
var w = 1280,
    h = 800,
    z = d3.scale.category20(); // 有序颜色尺度

    var force = d3.layout.force()
        .size([w, h]); // 设置布局尺寸为svg图形尺寸

    var svg = d3.select("body").append("svg")
        .attr("width", w)
        .attr("height", h);
```

- 然后读取数据，将层级数据平铺，创建连接关系，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

```
d3.json("../../data/flare.json", function(root) {
      var nodes = flatten(root),
          links = d3.layout.tree().links(nodes);

      force // 节点数据和连接属性添加到力布局
          .nodes(nodes)
          .links(links)
          .start();
```

- flatten()函数用来将层级化的数据集铺平

```
function flatten(root) {// 层级化的数据集平铺
      var nodes = [];
      function traverse(node, depth) {
        if (node.children) { // 递归搜索孩子节点
          node.children.forEach(function(child) {
            child.parent = node;
            traverse(child, depth + 1);
          });
        }
        node.depth = depth;
        nodes.push(node);
      }
      traverse(root, 1); // 根节点开始
      return nodes;
    }
```

- 之后绑定连接数据和节点数据，并设置相应属性

```
// 绑定连接数据
var link = svg.selectAll("line")
          .data(links)
        .enter().insert("line")
          .style("stroke", "#999")
          .style("stroke-width", "1px");
// 绑定节点数据
var node = svg.selectAll("circle.node")
          .data(nodes)
        .enter().append("circle")
          .attr("r", 4.5)
```

```
            .style("fill", function(d) { return z(d.parent && d.parent.name);
}) // 设置节点颜色
            .style("stroke", "#000")
            .call(force.drag); // 拖拽函数
```

- 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

```
force.on("tick", function(e) { // source和target指定连接对象关联的两个节点
        link.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; });

        // 将cx、cy属性设置为d的x和y,便于控制节点
        node.attr("cx", function(d) { return d.x; })
            .attr("cy", function(d) { return d.y; });
      });
```

实现效果

- 如图所示